User Interface Specification for Interactive Software Systems

Process-, Method- and Tool-Support for Interdisciplinary and Collaborative Requirements Modelling and Prototyping-Driven User Interface Specification

Dissertation zur Erlangung des akademischen Grades des Doktor der Naturwissenschaften (Dr. rer. nat.)
Universität Konstanz
Mathematisch-Naturwissenschaftliche Sektion
Fachbereich Informatik und Informationswissenschaft

Vorgelegt von Thomas Memmel

Tag der mündlichen Prüfung: 29. April 2009
1. Referent: Prof. Dr. Harald Reiterer
2. Referent: Prof. Dr. Rainer Kuhlen

For Cathrin

Acknowledgements

Prof. Dr. Harald Reiterer

I thank my advisor, Prof. Dr. Harald Reiterer, for more than 6 years of great teamwork. Since I joined his work group as a student researcher, his guidance and friendship have helped me to reach high goals and achieve scientific recognition. I thank Harald for his creative contributions and his unfailing support, which made him the best supervisor I could imagine. Every time I read the Dr. in front of my name, I will think about the person who made it possible. It was Harald! Moreover, I thank him for teaching me many skills, of which especially purposefulness and persuasive power opened up a world of possibilities.

My colleague and friend Fredrik

Among the other researchers in the human-computer interaction work group, special thanks are due to my colleague Fredrik Gundelsweiler. Fredrik and I started working for Harald at the same time, and since then we have shared many experiences. I worked with Fredrik at Siemens AG in Munich, and we both gained international work experience during our stay at DaimlerChrysler AG in Singapore. Although our work topics diverged after we worked for the electronics division of Mercedes-Benz, being a PhD student without being able to sit face-to-face with Fredrik in the office would not have been nearly so much fun. I wish with all my heart that Fredrik and I will continue to be close friends and comrade-in-arms in the field of human-computer interaction. As one should never change a winning team, I am sure we will be able to profit from mutual advice and candidness. We travelled through various ups and downs in the last decade and I would not have missed a single experience we shared.

Members of the workgroup HCI

I would also like to thank Jens Gerken, who helped me in evaluating the experimental tool that is presented in my thesis. As a usability expert, Jens' expertise in usability evaluation made it possible to enhance both my conceptual and practical work. Moreover, I thank Florian Geyer and Johannes Rinn, who were Master's students and assistant researchers during my time as a PhD student. Their extraordinary engagement and their reliability helped to make the software tool presented in this thesis a successful contribution to research communities all over the world. I am sure that you and all the other students I was able to supervise will have successful careers.

My girlfriend Cathrin

Most importantly, I thank my girlfriend Cathrin for her stamina in staying at my side and supporting me through six months of hardship and hard work. Without her, my PhD project would have failed. She helped me to stay focused and resolute. Thanks to her love and friendship, I am beginning a new period of life. Having finished my studies and moved to Switzerland, where I have signed a contract with Zuehlke Engineering AG, I look forward to the best possible life with her by my side.

For being a remarkable person and companion, I thank Christian Daum, who departed this life in spring 2007. Thank you, Christian, for a wonderful time in South East Asia and for being my friend.

My parents

I would also like to thank my parents, who gave me the opportunity to study and to continue my education after both the Bachelor's and Master's degrees. Their trust in my willingness and determination always strengthened my drive to contribute value to the field of computer science. I especially thank my father for his belief in my business instincts. As a team, we were a particularly good balance of wise experience with youthful 'get up and go'. I am proud that you let me contribute to the success of our family business during my studies.

General acknowledgement

I thank all the people mentioned here for having an interest in my profession and work, for respecting my job and the associated stress and strains, and for supporting me in reaching my goals over and over again.

Thank you all.

Thomas

Abstract

Specifying user interfaces (UIs) is a fundamental activity in the UI development life cycle as the specification influences the subsequent steps. When the UI is to be specified, a picture is worth a thousand words, and the worst thing to do is write a natural-language specification for it. In corporate specification processes, Office-like applications dominate the work style of actors. Moving from text-based requirements and problem-space concepts to a final UI design and then back again is therefore a challenging task. Particularly for UI specification, actors must frequently switch between high-level descriptions and low-level detailed screens. Good-quality specifications are an essential first step in the development of corporate software systems that satisfy the users' needs. But the corporate UI development life cycle typically involves multiple actors, all of whom make their own individual inputs of UI artefacts expressed in their own formats, thus posing new constraints for integrating these inputs into comprehensive and consistent specifications for a future UI.

This thesis introduces a UI specification technique in which these actors can introduce their artefacts by sketching them in their respective input formats so as to integrate them into one or more output formats. Each artefact can be introduced in a particular level of fidelity (ranging from low to high) and switched to an adjacent level of fidelity after appropriate refining. The resulting advanced format is called an interactive UI specification and is made up of interconnected artefacts that have distinct levels of abstraction. The interactive UI specification is forwarded to the supplier, who can utilize its expressiveness to code the final UI in precise accordance with the requirements.

The concept of interactive UI specification integrates interdisciplinary and informal modelling languages with different levels of fidelity in UI prototyping. In order to determine the required ingredients of interactive UI specifications in detail, the different disciplines that contribute to corporate UI specification processes are analyzed in similar detail. For each stage in the UI specification process, a set of artefacts is specified. All stakeholders understand this set, and it can be used as a common vehicle. Consequently, a network of shared artefacts is assembled into a common interdisciplinary denominator for software developers, interaction designers and business-process modellers in order to support UI modelling and specification-level UI design. All means of expression are selected by taking into consideration the need for linking and associating different requirements and UI designs. Together, they make up the interactive specification.

The UI specification method presented in this thesis is complemented by an innovative experimental tool known as INSPECTOR. The concept behind INSPECTOR is based on the work style of stakeholders participating in corporate UI specification processes. The tool utilizes a zoom-based design room and whiteboard metaphor to support the development and safekeeping of artefacts in a shared model repository. With regards to the transparency and traceability of the rationale of the UI specification process, transitions and dependencies can be externalized and traversed much better by using INSPECTOR. Compared to Office applications such as Microsoft Word or PowerPoint, INSPECTOR provides different perspectives on UI specification artefacts, allows the actors to keep track of requirements and supports a smooth progression from the problem-space to the solution-space. In this way, co-evolutionary design of the UI is introduced, defined, and supported by a collaborative UI specification tool allowing multiple inputs and multiple outputs.

Finally, the advantages of the approach are illustrated through a case study and by a report on three different empirical studies that reveal how the experts who were interviewed appreciate the approach. The thesis ends with a summary and an outlook on future work directed towards better tool support for multi-stakeholder UI specification.

Zusammenfassung

User Interface (UI) Spezifikationsprozesse involvieren unterschiedliche Akteure mit jeweils eigenen Ausdrucksmitteln. Insbesondere Organisationen, die zwar Softwareanwendungen entwickeln möchten, jedoch bei Planung und Umsetzung auf externe Dienstleister angewiesen sind, müssen einen geeigneten Rahmen für die interdisziplinäre Zusammenarbeit innerhalb und außerhalb des Unternehmens bestimmen. Der Erfolg der UI Entwicklung ist dabei meist von erheblicher Bedeutung für die Auftraggeberorganisation. Intern eingesetzte Anwendungen bestimmen die Effizienz und Effektivität softwaregestützter Prozesse maßgeblich. Interaktive Systeme, die in Produkte eingebettet werden oder einen Kommunikationskanal zum Kunden darstellen, müssen Design, Identität und meist auch Qualitätsmerkmale eines Unternehmens und einer Marke transportieren.

Dadurch ergeben sich Herausforderungen bei der Umsetzung von Anforderungen in gutes UI Design. Durch einen Mangel an interdisziplinären und kollaborativen Methoden und Werkzeugen dominieren dabei heute vor allem textbasierte Spezifikationsdokumente. In der Auftraggeberorganisation fungieren in der Regel Office Anwendungen als Spezifikationswerkzeuge (z.B. Word oder PowerPoint). Diese reichen jedoch mangels Interaktivität nicht aus, um innovative und kreative Prozesse zu unterstützen, sowie Aussehen und interaktives Verhalten moderner UIs zu beschreiben.

In dieser Dissertation wird eine Spezifikationstechnik für interaktive Systeme vorgestellt, mit der Benutzer-, Aufgaben- und Interaktionsmodelle sowie unterschiedlich detaillierte UI Prototypen miteinander verbunden werden können. Dadurch entsteht eine erlebbare UI Simulation, die im Vergleich zu interaktiven UI Prototypen zusätzlich den visuellen Drill-Down zu Artefakten der Anforderungsermittlung erlaubt. Das Resultat wird in dieser Arbeit als interaktive UI Spezifikation bezeichnet, mit der eine höhere Transparenz und Nachvollziehbarkeit von Designentscheidungen und Ergebnissen möglich ist. Dies führt zu einem durch Prototypen getriebenen Spezifikationsprozess, in dem Ideen, Visionen, Alternativen und Designergebnisse permanent visuell dargestellt werden können. Auf diese Weise wird die Kreativität und Zusammenarbeit gefördert, sowie die Position der Spezifikationsverantwortlichen gestärkt. Die frühe Entwicklung von Prototypen verhindert späte und kostspielige Änderungen und die Auftraggeberorganisation kann UI Qualität und Gebrauchstauglichkeit bereits vor Beauftragung eines Dienstleisters sicherstellen.

Um die genauen Bestandteile einer interaktiven UI Spezifikation bestimmen zu können, werden die hauptsächlich an einem UI Spezifikationsprozess beteiligten Akteure hinsichtlich ihrer Disziplin identifiziert. Auf dieser Grundlage wird für alle wichtigen Bereiche des UI Spezifikationsprozess jeweils mindestens ein Ausdrucksmittel (z.B. Diagramm, Prototyp) bestimmt, welches die beteiligten Akteure verstehen und gemeinsam anwenden können. Auf diese Weise wird ein Baukastensystem geschaffen, welches die in weiten Teilen recht unterschiedlich arbeitenden Disziplinen Software Engineering, Mensch-Computer Interaktion und Geschäftsprozessmodellierung auf einem gemeinsamen Nenner zusammenfügt. Durch die geschickte Auswahl geeigneter Notationen und Ausdruckmittel kann ein Netzwerk von Anforderungen und Designartefakten entstehen, welches das zu entwickelnde UI in allen wichtigen Facetten spezifiziert. Schließlich wird ein experimentelles Werkzeug namens INSPECTOR vorgestellt, das die Entwicklung interaktiver UI Spezifikation unterstützt.

Die Ergebnisse und die Anerkennung der vorgestellten Arbeiten werden durch unterschiedliche Evaluationsstudien dargelegt. Darüber hinaus werden am Ende der Arbeit Chancen für die Weiterentwicklung der vorgestellten Spezifikationsmethode und des Werkzeugs INSPECTOR diskutiert, um weitere Verbesserungsmöglichkeiten für UI Spezifikationsprozesse aufzuzeigen.

Parts of this thesis were published in:

Memmel, Thomas; Reiterer, Harald (2008): Model-Based and Prototyping-Driven User Interface Specification to Support Collaboration and Creativity. In: International Journal of Universal Computer Science (J.UCS). Special issue on New Trends in Human-Computer Interaction. ISSN: 0948-695X. December, 2008

Memmel, Thomas; Brau, Henning; Zimmermann, Dirk (2008): Agile nutzerzentrierte Softwareentwicklung mit leichtgewichtigen Usability Methoden - Mythos oder strategischer Erfolgsfaktor? In: Brau, H., Diefenbach, S., Hassenzahl, M., Koller, F., Peissner, M. & Röse, K. (Hrsg.), Fraunhofer IRB Verlag: Stuttgart., Usability Professionals 2008, 223-227

Memmel, Thomas; Geis, Thomas; Reiterer, Harald (2008): Methoden, Notationen und Werkzeuge zur Übersetzung von Anforderungen in User Interface Spezifikationen. In: Brau, H., Diefenbach, S., Hassenzahl, M., Koller, F., Peissner, M. & Röse, K. (Hrsg.) , Fraunhofer IRB Verlag: Stuttgart, Usability Professionals 2008, 45-48

Memmel, Thomas; Reiterer, Harald (2008): User Interface Entwicklung mit interaktiven Spezifikationen. Proceedings of the Mensch & Computer 2008: Viel mehr Interaktion, 8. Konferenz für interaktive und kooperative Medien, Oldenbourg Verlag, 357-366

Memmel, Thomas; Vanderdonckt, Jean; Reiterer, Harald (2008): Multi-fidelity User Interface Specifications. Proceedings of the 15th International Workshop on the Design, Verification and Specification of Interactive Systems (DSV-IS 2008), Kingston, Canada, 43-57

Memmel, Thomas; Geyer, Florian; Rinn, Johannes; Reiterer, Harald (2008): A Zoom-Based Specification Tool for Corporate User Interface Development. Proceedings of the IADIS International Conference on Interfaces and Human Computer Interaction (IHCI 2008, Amsterdam, The Netherlands), 368-370

Memmel, Thomas; Geyer, Florian; Rinn, Johannes; Reiterer, Harald (2008): Tool-Support for Interdisciplinary and Collaborative User Interface Specification. Proceedings of the IADIS International Conference on Interfaces and Human Computer Interaction (IHCI 2008, Amsterdam, The Netherlands), 51-60

Memmel, Thomas; Reiterer, Harald (2008): Inspector: Interactive UI Specification Tool. Proceedings of the 7th International Conference On Computer Aided Design of User Interfaces (CADUI) 2008, Albacete, Spain, 161-174

Memmel, Thomas; Reiterer, Harald (2008): Inspector: Method and tool for visual UI specification. Proceedings of the 3rd IASTED International Conference on Human Computer Interaction (IASTED-HCI, Innsbruck, Austria), Acta Press, Canada, 170-179

Memmel, Thomas; Reiterer, Harald; Ziegler, Heiko; Oed, Richard (2008): User Interface Specification In Complex Web-Based Information Spaces. Proceedings of the 3rd IASTED International Conference on Human Computer Interaction (IASTED-HCI, Innsbruck, Austria), Acta Press, Canada, 180-185

Memmel, Thomas; Reiterer, Harald; Ziegler, Heiko; Oed, Richard (2007): Visual Specification As Enhancement Of Client Authority In Designing Interactive Systems. Proceedings of 5th Workshop of the German Chapter of the Usability Professionals Association e.V. (Weimer, Germany), In: Kerstin Roese, Henning Brau: Usability Professionals 2007, Frauenhofer IRB Verlag, Stuttgart, 99-104

Memmel, Thomas; Gundelsweiler, Fredrik; Reiterer, Harald (2007): Agile Human-Centered Software Engineering. Proceedings of the 21st BCS HCI Group conference (HCI 2007, University of Lancaster, UK), In: Linden J. Ball, M. Angela Sasse, Corina Sas, Thomas C. Ormerod, Alan Dix, Peter Bagnall and Tom Mc Ewan: "HCI...but not as we know it", British Computer Society, 167-175

Memmel, Thomas; Heilig, Mathias; Schwarz, Tobias; Reiterer, Harald (2007): Visuelle Spezifikation interaktiver Softwaresysteme. Proceedings of the 7th Mensch & Computer conference (MCI 2007, Weimar, Germany), In: Tom Gross, Mensch & Computer 2007, Oldenbourg Verlag, Weimer, Germany, 307-310

Memmel, Thomas; Heilig, Mathias, Reiterer, Harald (2007): Model-based visual software specification. Proceedings of the IADIS International Conference on Interfaces and Human Computer Interaction (IHCI 2007, Lisbon, Portugal)

Memmel, Thomas; Gundelsweiler, Fredrik; Reiterer, Harald (2007): CRUISER: a Cross-Discipline User Interface & Software Engineering Lifecycle. Proceedings of the 12th International Conference on Human-Computer Interaction (HCII 2007, Beijing, China), In: Julie Jacko: Human-Computer Interaction - Interaction Design and Usability (Part I), Springer-Verlag, Berlin, Heidelberg 2007, 174–183

Memmel, Thomas; Reiterer, Harald; Holzinger, Andreas (2007): Agile Methods and Visual Specification in Software development: a chance to ensure Universal Access. Proceedings of the 12th International Conference on Human-Computer Interaction (HCII 2007, Beijing, China), In: C. Stephanidis: Universal Access in Human-Computer Interaction - Coping with Diversity (Part I), Springer-Verlag, Berlin, Heidelberg 2007, 453–462

Memmel, Thomas; Reiterer, Harald (2007): Visuelle Spezifikation interaktiver Systeme mit Modell- und XML-basierten Prototyping-Werkzeugen und Werkzeugketten. Proceedings of the 1st conference on Software Engineering Essentials (SEE 2007, Munich, Germany). In: Jan Friedrich, Andreas Rausch, and Marc Sihling , IfI Technical Report Series IfI-07-07, 78-92

Gundelsweiler, Fredrik; Memmel, Thomas; Reiterer, Harald (2007): ZUI Konzepte für Navigation und Suche in komplexen Informationsräumen. In: Prof. Dr.-Ing. Juergen Ziegler, Oldenbourg Wissenschaftsverlag , i-com, Zeitschrift für interaktive und kooperative Medien, 6 (1), 38-48

Memmel, Thomas; Bock, Carsten; Reiterer, Harald (2007): Model-driven prototyping for corporate software specification. Proceedings of the 1st Conference on Engineering Interactive Systems (EHCI-HCSE-DSVIS'07), IFIP International Federation for Information Processing 2008. In: J. Gulliksen et al., EIS 2007, LNCS 4940, 158–174 (Available online: http://www.se-hci.org/ehci-hcse-dsvis07/accepted-papers.html)

Memmel, Thomas; Gundelsweiler, Fredrik; Reiterer, Harald (2007): Prototyping Corporate User Interfaces - Towards A Visual Specification Of Interactive Systems. Proceedings of the 2nd IASTED International Conference on Human Computer Interaction (IASTED-HCI '07, Chamonix, France), Acta Press, Canada, 177-182

Gundelsweiler, Fredrik; Memmel, Thomas; Reiterer Harald (2004): Agile Usability Engineering. Proceedings of the 4th Mensch & Computer conference (MCI 2007, Paderborn, Germany), In: Keil-Slawik, R.; Selke, H.; Szwillus, G.: Mensch & Computer 2004: Allgegenwärtige Interaktion, Oldenbourg Verlag, München, 33-42

Contents

List of Figures

List of Tables

Chapter 1 Introduction

"Organizations wanting to be innovative need to move to a prototype-driven culture." (Schrage 1999), p. 195

1.1 Problems of Software Development

Cost overruns, late deliveries, and cancellations are common

Thousands of companies worldwide waste billions of dollars a year on poorly conceived and poorly implemented information systems. Several global studies estimate that at least one quarter of internal software-development initiatives are cancelled outright and written off as total losses (Schrage 1999). Cost overruns, late deliveries, and cancellations are common. Although the situation has improved slightly (see Figure 1), a recent Chaos Report shows that the proportion of software-development projects that overrun time and budget is still more than one half (The Standish Group 2006). One of the major causes of both cost and time overruns is restarts. For every 100 projects that start, there are 94 restarts. The average across all companies is 189% of the original cost estimate. The average overrun is 222% of the original time estimate. The top four reasons for this are (1) frequent requests for changes from users, (2) overlooked tasks, (3) users' lack of understanding of their own requirements, (4) insufficient user-analyst communication and understanding (Lederer & Prasad 1992; Bias & Mayhew 1994).

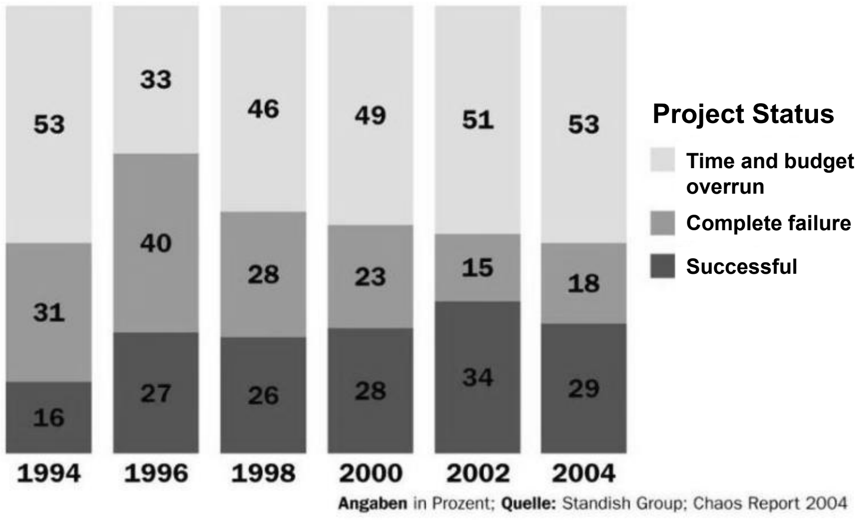


Figure 1: Although the number of successful software projects was almost 50% higher in 2004 than a decade before, the relative amount of projects that overran time and budget remained the same (Hackmann 2007)

Missing or incomplete requirements are one of the most commonly found reasons for project failure

Accordingly, the Standish Group identifies missing or incomplete requirements as one of the most commonly found reasons for project failure (The Standish Group 2003). The National Institute of Standards and Technology (NIST 2002) found that although an average of 70% of all errors are created during the requirements-analysis phase, they are not identified until real end-users get to see the results of software development. Most problems therefore emerge shortly before software rollout. In the last 10 years, the situation has changed only slightly, as (Pressman 1992; Karat 1993aa; Karat 1993bb) have already indicated that 80% of all software-development costs occur after the product has been released, and therefore 80% of software maintenance is due to unfulfilled or unforeseen user requirements; only 20% is due to bugs or reliability problems.

Failed acceptance tests cause frustration

Because of failed acceptance tests and 'bad' software quality, software developers often express enormous frustration because their clients seem to be unable to describe the software product they want (Schrage 1999).

Late-cycle changes blow software-development budgets

But especially from a financial perspective, this dilemma is a catastrophe for software projects. Unfortunately, it is about 40–100 times more expensive to fix problems in the maintenance phase of a program than in the design phase (Boehm 1981). A change may cost 1.5 units of project resource during conceptual design, 6 units during early development, 60 during systems testing and 100 during post-release maintenance (Pressman 1992). If an error is found during the early claims-analysis stage, its removal costs about 100 euros. If the same error needs to be repaired after a failed user-acceptance test, the costs are 50 to 100 times higher. This gives a figure of 7500 euros on average. One of the most critical reasons for software-development failure is therefore the deferral of software testing with end-users. The reasons for this can be found in the nature of common software-development processes.

The structure of typical development processes

Typically, software projects start with a requirements analysis. Experts therefore conduct a context analysis, define product goals and gather domain knowledge. The usual procedure is that stakeholders, among them potential end-users, are interviewed, and the results are then written down in a requirements sheet. For this purpose, (Robertson & Robertson 1999) distinguish between functional and non-functional requirements. Functional requirements specify what the product must do and are very often expressed through action statements (e.g. check, calculate, record, retrieve). Functional requirements are therefore primarily deduced from the fundamental purpose of the software. Non-functional requirements are properties that the functionality must have. Among them are look-and-feel requirements, usability requirements and performance requirements.

The Volere process recommends prototyping functional and non-functional requirements

The Volere process model (see Figure 2), for example, recommends that certain requirements be prototyped during claims analysis. The purpose is the clarification of uncertain or complex stakeholder needs. This includes prototyping the 'look and feel', i.e. the user interface (UI), of a software product. The prototype is intended to show the implication of a requirement and not to express the requirement itself. However, the problems suffered by many application development projects suggest that the importance of prototyping functional and non-functional requirements has not yet been adequately reflected in the practical application of (theoretical) software-development methodologies.

Prototyping does not take place as intended

Although prototyping is useful in any situation where a requirement is not clear, there are several reasons why prototypes are often disregarded as a valuable methodology in many software-development projects, for example:

- **Cost-saving measures**. Requirements prototypes are usually throw-away products and are not expected to evolve into the finished product (Robertson & Robertson 1999). It is therefore likely that project managers save on their development, especially when the role of look and feel and usability is underestimated or misunderstood, and stakeholders are used to textual descriptions of requirements.

- **Form of contract**. If an external IT supplier is assigned to design and code the software system, the production of prototypes is usually deferred until the specification sheet has been produced (Schrage 1999). The consolidated specification is usually necessary for forming a contract between client and supplier. It is unlikely that the supplier will start working before the specification is finalized.

- **Delaying influence**. Even if prototypes are created during early stages, the chance of them having significant impact on the consolidation of written requirements is very low. Depending on the complexity of the software that has to be built and factors such as time and budget, building a first prototype can take (the IT supplier) several weeks, or even up to a month (Schrage 1999). This is mainly due to poor responsibility assignment and excessive up-front processes. Results are not likely to be received until there is no longer any opportunity for extensive changes.

Prototypes come too late in the process

Consequently, software developers tend to produce first prototypes very late in the overall process, if at all, although well-established lifecycle models do recommend it for the earlier stages. With regards to the Volere process model, prototyping is deferred until the 'analyze, design and build' phase, which follows the consolidation of the requirements specification (see Figure 2).

Prototyping is often deferred until the analyze, design and build phase

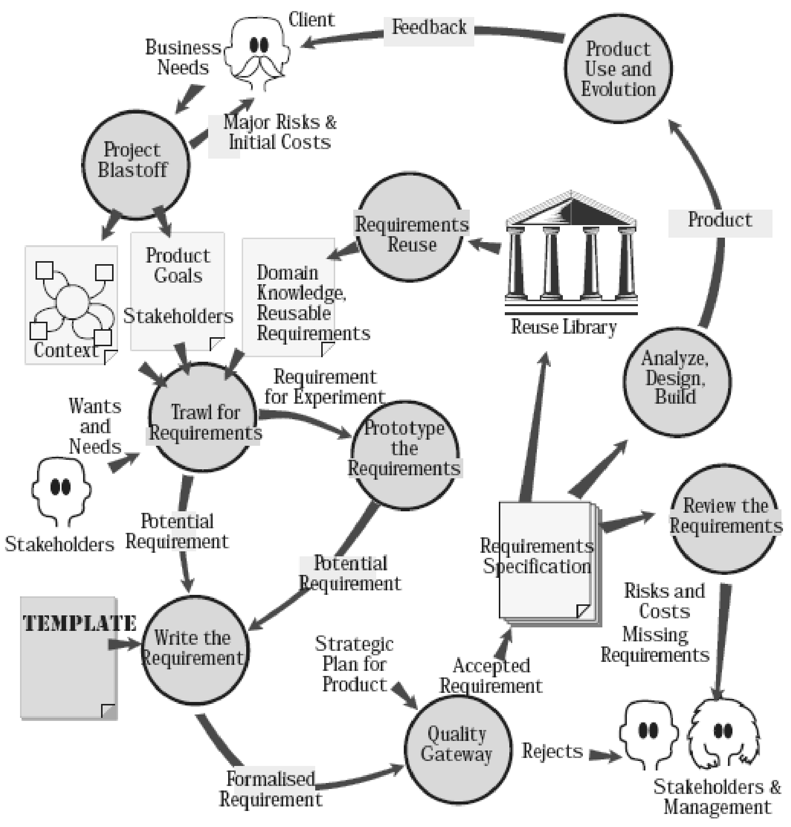


Figure 2: The Volere Process Model (Robertson & Robertson 1999)

Then, unfortunately, the real development process does not formally start until a plan is in place and there have been several meetings about what is supposed to be done (Löwgren & Stolterman 2004). Written requirements will be translated into systems requirements and are then forwarded to decision-makers, management and other stakeholders for approval. Usually, a certain number of modifications will be incorporated and the quality of the requirements will be assessed at several quality gates (see Figure 2). Not until the client gives the go-ahead, will the development team finally translate the requirements into a (functional) requirement specification. Later on, the software-requirements specification describes what has to be done in the subsequent coding and (functional) testing phase (Schrage 1999).

Prototyping, design and implementation do not start until requirements have passed several quality gates and have been translated into a specification document

The break with reasonable recommendations for early-stage prototyping seriously affects the way software projects are executed. Software developers typically perform an extensive requirements analysis and try to determine precisely what stakeholders and end-users need. Although this course of action is per se necessary to set up a software project, it is difficult to understand user needs without prototyping (see Chapter 1.2). The overall process requires thoughtful balancing between the abstract requirements data and enabling clients, i.e. stakeholders and end-users, to visualize their requirements. The dimension of time is a critical success factor for software development. The crux of the matter is the point in time when developers start to externalize the collected requirements data into something tangible that users can look at and 'feel'.

Thoughtful balancing between abstract requirements data and requirements visualization

Independent of the overall duration of the up-front requirements engineering (RE), it is critical to externalize design vision with prototypes as soon as possible in order to incorporate feedback. If the real work, which is the actual implementation of a piece of software that stakeholders can look at, begins too late, it is likely that clients will not like the software that is delivered to them (Schrage 1999):

It is critical to externalize design vision with prototypes as soon as possible

"(Then) the development team is enraged (and) the internal client is similarly furious with the developers for being inflexible geeks who don't understand their business" (Schrage 1999), p. 19

Requirements engineering practice has to take prototyping seriously

The described pathology can be avoided by changing the way the up-front RE is embedded in a software project. This means taking recommendations about early-stage prototyping seriously. The developers could quickly move to prototyping the software as soon as the top twenty or thirty of the requirements have been analyzed (Schrage 1999). In order to make appropriate change to established software-development methodologies, radical changes in organizational-development cultures are necessary.

The right prototypology

The reasons why applied practice differs from common lifecycle models must therefore be identified and well understood. In order to better incorporate prototyping efforts into software-development practice, several fundamental issues have to be addressed in order to identify the right prototypology, for example:

- **Appropriate fidelity**. Discuss different fidelities of prototyping and their contribution to the software-development process with regards to the externalization of functional and non-functional requirements.
- **Pressure of time and budget**. Determine prototypes that can be developed within time and budget constraints in order to overcome objections to early-stage prototyping.
- **UI development fragmentation**. Identify the kind of prototypology (Schrage 1999) that can take client and supplier relationships into account and become part of contract-forming.
- **UI specification support**. Consider forms of prototyping that can become recognized as well-engineered models that evolve into specifications and final products.
- **Adequate traceability**. Externalize the interdependency of models and design, turning the UI development into a transparent engineering process.

1.2 Prototyping-Driven Specification and the User Interface

Specification-driven prototyping

If stakeholders are unable to look at, and ‘feel’, their requirements at an early-stage, both the specification document and the first prototype will not match the real end-users’ needs. Without prototyping, it is almost impossible for even the most sophisticated clients to get the software that meets their requirements (Schrage 1999), because the course of action lacks up-front requirements visualization and evaluation. In such project environments, the first externalizations of requirements can be called ‘specification-driven prototypes’ as it is primarily the specification that drives the subsequent prototyping process. In specification-driven prototyping cultures, the specification sheet must first be consolidated, and only then can end-users access prototypes of the software system. Without prototyping, software developers miss the opportunity for innovation and design variety is heavily restricted. The use of informal demonstrations has the power to facilitate communication between many groups of people. Charismatic prototypes that invite collaboration and enhancement tend to engage people in charismatic conversations (Schrage 1999):

Innovative teams generate innovative prototypes

“Innovative teams generate innovative prototypes. The corollary is that an organization has to get the right people working together on the right project to create the right prototypes that can be turned into right products. [...] (And), innovative prototypes generate innovative teams. [...] An interesting prototype emits the social and intellectual equivalent of a magnetic field, attracting smart people with interesting ideas about how to make it better.” (Schrage 1999), p. 28

Prototypes shape ideas

Prototypes give ideas a more defined shape, which is the most important part of the design process. It has the function of bridging the abstract vision and the detailed presentation of the design thoughts (Löwgren & Stolterman 2004). There will be recurrent leaps between the concrete and the abstract, as well as between the details and the whole. Switching between artefacts that have a different grade of abstraction is among the most important tasks of designers (Campos & Nunes 2006) and is necessary in order to be able to narrow the design space towards the best solution.

Box 1.2.1 Artefacts
Artefacts can be defined as anything that is produced or consumed by process or activity (Holt 2005). An artefact is an aspect of the material world that has been modified over the history of its incorporation into goal-directed human action. By virtue of the changes wrought in the process of their creation and use, artefacts are simultaneously ideal (conceptual) and material. They are ideal in that their material form has been shaped by their participation in the interaction of which they were previously a part and which they mediate in the present (Cole 1996).

People can much more easily articulate what they need by playing with prototypes

The forceful employment of prototypes would lead to situations, in which the development team could continuously present their interpretation of stakeholder needs and the corresponding design solutions to the client. People can much more easily articulate what they need by playing with prototypes. Prototypes can turn stakeholders into partners in a collaborative software-development process. Prototypes externalize conflicts and require stakeholders to handle trade-offs. The sooner individuals can access simulations of their requirements, the earlier they can recognize the need for modification and decision-making. This, in turn, decreases the risk of costly late-cycle changes (see Chapter 2).

Prototyping-driven specification

Instead of simply producing a specification-driven prototype, software projects must rely on a prototype-driven specification process that employs simulations to drive the development. This means that the specification will be mainly defined through tangible externalizations of product concepts and design vision.

Prototype-driven design is superior to specification-driven methods

From the early stages of the software-development process, team conversations then take place around tangible UI designs, because visualizing concepts is even more important than articulating them (Schrage 1999). This hypothesis can be traced back to prototyping experiments such as the one by (Boehm et al. 1984). They revealed that the productivity of a prototype-driven design, measured in user satisfaction per invested man-hour, is superior to specification-driven methodologies. Organizations that are able to manage their models, prototypes, and simulations can gain a significant competitive advantage (Schrage 1999).

Specification and prototyping has to be well balanced

As specifications and prototypes can be either reinforcing or conflicting artefacts, managing the dialogue between them is essential for the design of innovative products. If this dialogue is poorly managed, even the most extensive specification document can be invalidated by the first visual prototype. Conversely, a completely design-driven prototyping stage can lead to designs that are not cost-effective and are technically unfeasible.

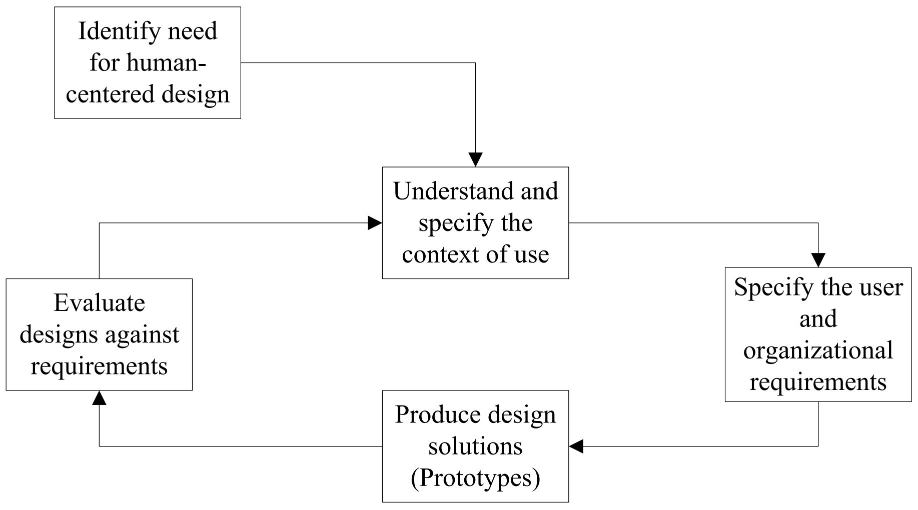

Identify need for human-centered design
Understand and specify the context of use
Specify the user and organizational requirements
Produce design solutions (Prototypes)
Evaluate designs against requirements

Figure 3: The human-centred design activities in ISO 13407 (DIN EN ISO 13407 1999)

The medium of prototyping is widely recognized and is a competitive success-factor of design

Fortunately, the medium of prototyping is a means of communication that has been recognized by stakeholders of various disciplines related to the design of modern software systems. In software engineering (SE), prototypes are used to verify functional specifications and models, as well as for understanding problems by carrying out user inspections and testing. In usability engineering (UE), prototypes are recognized as an artefact for iterative UI design. They are employed for requirements analysis, producing alternative designs and for gathering user feedback (see Figure 3). Here, the role of prototypes is therefore a particularly deep-seated method for propelling user-friendly development and is primarily related to the UI of a software product (Preece et al. 2002). In today's software market, usable products are desirable products. Providing task-adequate functionalities is mandatory, but there is great competition to deliver them with a high grade of usability. Ease of use therefore differentiates software products in a highly competitive market place. Ease of use brings an added value that culminates in a higher degree of customer satisfaction, continuing business and higher revenues (see Chapter 2). In addition, customer satisfaction provides market differentiation (Jones & Sasser 1995).

Prototyping-driven UI specification

Consequently, new approaches to RE and software development need to make special provision for prototyping-driven specifications of the UI. This involves the following challenges related to prototyping and the specification of usable software UIs:

- **UI prototyping**. Identify different kinds of prototypes as applied in software and UE to determine a common course of action.
- **UI specification**. Identify prototyping methods that support UI specification and that integrate smoothly with modelling and requirements management.
- **Prototyping-driven UI specification**. Develop a specification method that is mainly propelled by the externalization of design vision and rationale.

1.3 User Interface Specification and Software Development

Structured approaches are required to design, specify, and verify interactive systems

Experts from SE and UE agree that structured approaches are required to design, specify, and verify interactive systems (Constantine & Lockwood 1999b; Elnaffar 1999; Barbosa & Paula 2003; Blomkvist 2005; Petrie 2006) so as to obtain a high usability of their UI (Metzker 2002; Newman 2003). The design, the specification, and the verification of user-friendly and task-adequate UIs have become a success-critical factor in many domains of activity.

Most SE methodologies do not take the UI into account

Nevertheless, most SE methodologies, among them, for example, the Rational Unified Process (RUP), the V-Model, and the new agile development approaches, still do not propose any mechanisms for explicitly or empirically identifying and specifying user needs and usability requirements (Kazman et al. 2003; Seffah & Metzker 2004). In large part, this explains why large numbers of change-requests to modify developed systems are made after their deployment, a phenomenon that is frequently observed (see Chapter 1.1).

Integrating UE into SE processes

To counter this trend, all UI-related issues must be addressed and assessed from the very beginning, when the software systems are defined. Because UE and SE are disciplines made up of very distinct populations, it is a demanding challenge to integrate specific methods that properly take the UI and its non-functional requirements into account. As outlined by (Seffah et al. 2005b), UE must be a core part of every software project, but the path is littered with major obstacles, fallacies, and myths that hamper the combining of the best of both disciplines:

- **The meaning of usability**. For software engineers, the term is typically associated with ease-of-use and user-friendly design. This perspective is propagated by the Volere process model (see Figure 2), for example, which simply puts the term usability among other non-functional requirements. Human-computer interaction specialists, in turn, regard UE as an interdisciplinary field, which deals with user and task analysis, context of use, and aesthetics, for example.
- **The people gap.** UI development is seldom allocated sufficient time in the early phase of a software project (see Chapter 1.1). An appropriate integration of UE methods usually only happens if the usability expert is also a strong programmer and analyst. But the typical usability engineer often does not understand the technical issues of a software product.
- **The responsibilities gap**. The role of the UI is often perceived as that of a decorative add-on to the back-end of the software system. Once the software engineers have finished coding the functionality, the usability experts design the interface layer. Conversely, the usability people regard their work as a primary part of the development process and would like to supervise the programmers while they code a system back-end that best supports the UI. Both disciplines have to accept that neither one in isolation is capable of creating usable software systems.
- **The modularity fallacy**. Traditional software architecture decomposes the systems into different parts, thereby allowing the work to be apportioned between the UI and underlying functionalities. In reality, system features can have significant impact on the UI and vice versa. Software developers and UI designers must be aware of these relationships (see Chapter 3).
- **The dispensability of UE**. Software engineers often feel that their project cannot afford to spend much time on usability and UI design. They worry that iterations will never end, because they are anxious that the usability people will continue the refinement cycle until everything is perfect. Essentially, well-defined usability goals, together with a thoughtful balance between user- and task-centring, will successfully prevent an over-extended trial and error process. Moreover, light-weight, agile methods can help to speed up the design of

usable systems and make the process more cost-effective (Memmel et al. 2007e).

- **The organizational shift**. The integration of UE methods into established software engineering lifecycles is also an issue in the organization's natural inertia. UE must become understood as a paradigm shift. This corresponds to the paradigm shift that comes with the creation of prototyping-driven specifications instead of specification-driven prototyping.

In order to properly integrate UE methods into SE processes, the communication between users, usability experts and developers must be mediated and improved. The applied RE methods must be extended for collecting information about users and usability. SE and UE artefacts, especially those for requirements modelling, must be aligned to the greatest extent possible in order to allow the development of expressive prototypes and the assembly of a specification of the functionality, and of the UI in particular (see Chapter 2).

1.4 Research Objectives

Challenges for UI development

The development of software systems faces very demanding challenges. High UI quality is required for software products to be successful. However, UI-related methodologies are not adequately considered in practice. Several critical ingredients therefore contribute to development failure: the importance of the UI, the separation of professions, particularly SE and UE, and consequently a lack of methods, tools, and process models that integrate all stakeholders and their discipline-specific demands.

RE, prototyping and specification processes have to take the UI into account

UE and UI development methods are still misunderstood ingredients of today's software-development processes. In order to be able to implement user- and task-adequate interactive systems with high UI usability, RE, prototyping and specification processes have to take the UI into account in more detail. In fact, special UI prototyping and UI specification methods must be provided to adequately address issues related to the UI.

The art of UI specification must be well defined

The art of UI specification (see Chapter 3), in particular, must become a well-defined part of software-development projects. Unfortunately, the ways to specify a UI are not well documented. In contrast to the wide proliferation of general software-specification methods, literature lacks sufficient guidance on how best to specify a UI. This is especially critical for corporate[1] UI development projects, where an (internal) client assigns an (external) IT supplier to code the UI of a software system. This requires a structured approach with clear touch and transfer points. The dialogue between UI prototyping and UI specification in prototyping-driven processes therefore has to be determined in detail.

Research objectives

Accordingly, this thesis has the following research objectives related to the analysis, design and specification of software products in which the UI has great weight:

- **Identify current UI development practice**. Outline the nature of current UI development processes and analyze the common practice in UI specification. Identify the right balance of RE and prototyping suitable for the development and specification of usable software products.
- **Bridge the gaps between software and UE**. Provide means for bridging the gaps between the disciplines that mainly participate in software development and UI design. Moreover, expand the perspective by taking into account other disciplines that also influence UI development.
- **Understand UI specification.** UI specification is closely related to UI development. More precisely, UI specification processes differ from actual development processes, especially in respect of the missing implementation stage. Furthermore, the end product of the specification process is more a description than lines of code. As most other activities are similar, it is important to understand the best practice in UI development in order to be able to successfully define approaches to corporate UI specification.
- **Define a common denominator in UI-related modelling**. Derive a shared course of action in UI-related requirements modelling that allows for a usability-driven user, task and behaviour specification. Disciplines can be bounded through changing the RE up-front in respect of the applied modelling languages and stakeholder attitude.

[1] In this thesis, the term corporate is used to (a) describe a development process triggered by an (non-IT) organization or department, which is dependent on internal or external IT suppliers, (b) refer to value chains that directly affect the client organization (e.g. in terms of new software)

- **Determine suitable UI prototyping methods**. Analyze the pros and cons of different forms of UI prototyping and isolate those methods that best contribute to UI specification requirements. By employing prototypes as vehicles for both design and UI specification, it is possible to bridge existing gaps while making the overall design process more efficient and effective, resulting in lower development costs, but improved software quality.
- **Develop a best practice for UI specification**. Based on the UI-related modelling and prototyping techniques discussed, identify a well-defined UI specification methodology that can support human-centred software-development processes. Take existing approaches documented in literature and industry into account.
- **Introduce interactive UI specifications.** Present the idea of a new form of 'living specification artefact' that will be likely to replace text-based documents. Assemble associated artefacts to visualize interdependencies between models and the UI layer. Externalize the network of requirements and provide means for requirements traceability and a high transparency of the design rationale.
- **Present experimental tool support**. Derive the requirements for a software tool that support multidisciplinary UI specification processes. Build an experimental UI specification tool that is based on the identified common denominator in UI requirements modelling and prototyping, and that supports the production of interactive UI specifications.
- **Evaluate the results**. Provide evidence of the added value of interactive UI specifications and the tool support provided.
- **Provide future prospects**. Outline opportunities for the further enhancement of both method- and tool-support for UI specification processes in general, and interactive UI specifications in particular. A special focus is to set up an expansion of the developed UI specification approach in terms of multi-modal in- and output, and to explore the opportunity to further lessen the weight of the common denominator by omitting more artefacts.

1.5 Thesis Outline

In order to address the research objectives presented here, this thesis is structured in modular parts that result in the introduction of a UI specification method and the corresponding experimental tool support.

Chapter 2

Chapter 2 discusses in detail the increasing importance of high UI quality to further promote the necessity of a well-defined and structured UI specification method. In order to provide a sound basis for it, the disciplines of UE and SE are outlined. As both disciplines play an important role in most UI development processes, a part of Chapter 2 discusses their characteristics in detail. In the process, the particularities of corporate software and UI development and specification processes are explained. This includes a closer look at business-process modelling (BPM) as a third pillar of corporate UI specification processes. Accordingly, this chapter explains the challenges of multi-stakeholder UI specification processes, which can be traced back to the various different disciplines that project members come from. Chapter 2 then outlines the shortcomings of, as well as the changes desired in, current UI specification practice. This gives rise to a requirement for a method- and tools-support that is adequate for the demands of UI specification.

Chapter 3

Chapter 3 leads on to discuss the characteristics of UI development and specification that are fundamental to design for usability and user experience. This chapter serves as a basis for defining the essential ingredients of a sound UI specification, which is one of the main focal points of this thesis. Accordingly, the art of UI specification is discussed in detail. In this connection, the role of style guides, claims and design rationales for UI specification practice is analyzed. This comes with a comparison of formal and informal approaches, as well as a discussion of model-based and model-driven approaches to UI specification. With regards to the disciplines that mainly contribute to corporate UI specification processes, ways of bridging the gaps between SE, UE and BPM with an agile model-based approach are explained in detail. Accordingly, we present agile methods as a common ground for all three disciplines. Based on the definition of the required ingredients and the right formality of UI specifications, we infer a new UI specification method that employs 'interactive UI specifications' as a means to define an interactive software system. The term 'interactive' refers to the concept of making the process visually externalized to the greatest extent possible. This concerns both the artefacts and the medium of the UI specification itself, which should no longer be a text-based document, but a running simulation of how the UI should look *and* feel.

Chapter 4

Chapter 4 continues to introduce the concept of the interactive UI specification in detail. In this chapter, the main ingredients of an interactive UI specification are identified. The chapter discusses the stage of UI-related modelling of users, tasks and UI behaviour requirements, and presents in detail the models that are suitable to be part of an interactive UI specification. The chapter continues the identification of UI specification methods with regards to prototyping. Although the medium of prototyping is common to all disciplines, the right prototypology (Schrage 1999) must be thoughtfully selected. In this context, prototyping techniques are explained in detail and analyzed according to their potential contribution for prototyping-driven UI specification processes. Consequently, the chapter introduces the common denominator in UI-related modelling. The common denominator is required for collaborative UI specification and is the basis for the experimental tool support - adequate for the problem - that is presented later on. Moreover, the compatibility of the proposed common denominator with an interdisciplinary lifecycle model for an agile human-centred software-development lifecycle is presented. The lifecycle gives guidance on how to specify high-quality UIs in an agile time and manner, considering modern UI design approaches.

Chapter 5

Chapter 5 describes related work and presents the most recent developments in the field of UI modelling and specification. With regards to corporate UI specification, chapter 5 points out to the gap in modelling and tool support.

Chapter 6

Chapter 6 introduces the tool known as INSPECTOR, which is designed to support interdisciplinary teams in gathering user needs, translating them into UI-related requirements, designing prototypes of different fidelity, and linking the resulting artefacts to an interactive UI specification. This chapter explains in detail the conceptual model of INSPECTOR, as well as the Zoomable User Interface (ZUI) approach that characterises its interaction style. The main part of the chapter is an extensive presentation of a use-case study that demonstrates the capabilities of INSPECTOR and the underlying concept of interactive UI specifications.

Chapter 7

Chapter 7 discusses the feedback gathered on the suitability of both the method and the tool for corporate UI specification processes. This chapter therefore contains a presentation of results from a questionnaire-based evaluation, an extended diary study and interviews with various experts. Accordingly, the chapter also explains the enhancements to INSPECTOR that were implemented as a result of the evaluation studies.

Chapter 8

Chapter 8 starts by summarizing the work that has been done. The chapter then provides an outlook for future projects related to a common denominator in UI-related modelling, prototyping and UI specification. In particular, this chapter discusses the chances of further lowering the number of models and representations that make up an interactive UI specification. On the other hand, the comprehensiveness of the method is examined and analyzed for future enhancements. For example, the potential of working with INSPECTOR in a shared, zoomable information landscape, using multi-modal input devices for collaborative modelling and design, is considered in detail.

Chapter 2 Corporate User Interface Development in a Nutshell

"The average UI has some 40 flaws. Correcting the easiest 20 of these yields an average improvement in usability of 50%. The big win, however, occurs when usability is factored in from the beginning. This can yield efficiency improvements of over 700%." (Landauer 1995)

Content of this chapter

This chapter describes the aspects of usability and the UI, and how they are to be incorporated into UI development processes. Understanding the main properties of UI development is essential in order to be able to define new UI specification processes. The practice of UI specification is closely related to UI development, because both activities include analysis, modelling and design tasks. In Chapter 2.1, the general importance of usable UIs is deduced from different approaches to product quality. Accordingly, Chapter 2.2 explains the impact of UI quality in terms of measurable return on investment (ROI). Chapter 2.3 then discusses the question of which engineering approaches exist in order to enable organizations to design for high UI quality and usability. This comes with an analysis of the different engineering disciplines that usually take part in corporate UI development processes. In Chapter 2.4, the main characteristics of SE, UE and BPM are outlined. This includes a discussion of important terms and definitions. In addition, the implications of typical client and supplier relationships are presented in Chapter 2.5. The shortcomings of, and required changes to, UI specification processes are argued in Chapter 2.6. The key points in Chapter 2.7 summarize the chapter.

2.1 Quality of Use

The face of software applications

Through the 1970s, it became clear that an important component of software development is the UI. With more and more software being developed for interactive use, attention to the needs and preferences of end-users intensified (Rosson & Carroll 2002). The UI is the face of almost all software applications. It determines the capabilities, limitations and organization of our work with computers. The more end-users become increasingly diverse, the more interactive systems must be compared and evaluated with respect to usability.

Quality of the UI

It is therefore very important to make the user's interaction as simple and intuitive as possible to allow efficient use. There are several reasons why the quality of a UI is critical (Shneiderman & Plaisant 2004):

- Life-critical systems require error-free interaction and rapid performance.
- Office and entertainment software requires ease of learning and subjective user satisfaction to compete in the market.
- Experimental systems or expert systems that focus on the enhancement or replacement of existing methodologies, like creative or cooperative interfaces, need to fulfil very high expectations to be adopted in practice.

Competing factors of influence

In general, software development is restricted to four concerns: features, time-to-market, cost and quality. And these aspects are all related: improvements in one may affect at least one of the others negatively. If more features are added to the product, quality must drop or time-to-market must slip. If time-to-market is cut, features must drop or quality must drop (Berkun 2002).

"There is no way around the natural tradeoffs of resources, time and quality. Beyond the standard set of tradeoffs, ease of use adds an additional dimension to the decision-making logic. The definition for quality changes to mean something other than pure engineering quality." (Berkun 2002)

The challenge is to develop usable software in time and budget

Cost, time-to-market, and quality – of back-end and UI – are the three main concerns that make software projects true challenges. Costs should be minimized to increase profit and market share, quality should be maximized to attract and satisfy customers and time-to-market should be minimal to reach the market before competitors do (Bengtsson 2002).

Practical approaches to quality

(Garvin 1984; Bevan 1995) distinguish four practical approaches to quality that provide an indication of how quality can be achieved in practice. All dimensions can be well mapped onto UI quality:

- **Product quality**: an inherent characteristic of the product determined by the presence or absence of measurable product attributes.
- **Manufacturing quality**: a product that conforms to specified requirements.
- **Economic quality**: a product that provides performance at an acceptable price, or conformance to requirements at an acceptable cost.
- **User-perceived quality**: the combination of product attributes that provide the greatest satisfaction to a specified user.

Product quality

Quality is generally treated as a property of a product. The product view of quality seeks to identify those attributes that can be designed into a product or evaluated to ensure quality (Bevan 1995). What is required for quality systems is specified in (DIN EN ISO 9001 1987). With regards to usability, the UI is the property that contributes much to product quality. Accordingly, (DIN EN ISO 9126 1991) categorizes software product quality in terms of functionality, efficiency, portability, reliability, maintainability and usability.

Manufacturing quality

The specification of product quality provides the requirement for the quality system. Manufacturing quality is achieved if the product matches the specification, although the quality of the end product can be no better than the quality of the specification (Bevan 1995). This can give direction to the quality of software and the UI development and specification processes. The methods and tools employed have significant impact on engineered product quality. The applied specification methods directly affect the result of software development.

Economic quality

Economic quality considers the relationship of cost and product quality in the manufacturing process, and price and product quality on the customer market. (DIN EN ISO 8402 1994) therefore defines quality as a product's ability to satisfy certain usage requirements. In this context, usability is described as the quality of a system with respect to ease of learning, ease of use, and user satisfaction (Rosson & Carroll 2002). With regards to the variety of user intentions and needs, a product often requires many-sided characteristics to satisfy customer expectations. These characteristics can be expressed in terms of usability goals and user-experience goals.

User-perceived quality

Consequently, user-perceived quality can be described by relating quality to the needs of the user of an interactive product. This extends the definition of quality in respect of user needs, user tasks and the context in which a product is used. If a product is to help users to achieve their goals, the quality perceived by the user can be measured as effectiveness, efficiency and satisfaction. (Bevan 1995) describes the quality of use as the result of the interaction between the user and product while carrying out a task in a technical, physical, social and organizational environment.

Develop usable software within time and budget

After all, quality of use provides a means of measuring the usability of a product, and usability is defined in this way in (DIN EN ISO 9241-11 1998). (Mayhew 1999) defines UE as the discipline that provides structured methods for achieving usability in UI design during product development (see Chapter 2.4). With regards to costs and time-to-market, the challenge is to develop usable software with high UI quality within time and budget. But high costs often prevent developers from meeting all the usability requirements and preclude the implementation of outstanding UI designs. It is therefore important to recognize the impact of usability aspects on both the quality of a product and the cost structures of software-development processes. In order to further motivate and underline the aspects and benefits of UI development, the return on investment of usability activities is discussed in the following chapter.

2.2 The Return-on-Investment of Providing Usable Product UIs

The significance of the UI

Many existing development processes focus exclusively on adherence to technical and process specifications. As a consequence, the developed systems generally meet all functional requirements, but are difficult to use with effectiveness, efficiency and satisfaction. This is in contrast to the significance of the UI in software development. The UI may account for up to 66% of the lines of code in an application and 40% or more of the development effort (MacIntyre et al. 1990). As outlined in Chapter 2.1, one of the software product qualities that receives increased attention is therefore the usability of the UI. But ultimately, software development is also driven by economics and project managers must make wise choices about the relative costs and benefits of UI design features.

UE can reduce the time and cost

During software development, UE can reduce the time and cost of development efforts through early definition of user goals and usability objectives, and by identification and resolution of usability issues. (Marcus 2002) reports that when managers were polled regarding the reasons for the inaccurate cost estimates, the top four reasons were requests for changes by users, overlooked tasks, users' lack of understanding of their own requirements, and insufficient communication and understanding between users and analysts. Another study of SE cost estimates showed that 63% of large projects significantly overran their estimates. The four reasons rated as having the highest responsibility for the overrun were related to UE (Nielsen 1993).

UI usability addresses customer expectations

Companies committed to UI usability do more to meet customer expectations and exceed anticipated earnings (Karat 1997). Poor usability has an impact not only on system reliability, but also on customer acquisition, retention, and migration. (Pressman 1992) has shown that for each phase of development that proceeds without usability testing, the cost of fixing usability problems increases by a factor of 10. Moreover, systems designed with an UE approach (see Chapter 2.4.2) have typically reduced the time needed for training by around 25% (Landauer 1995).

Bad usability results in economic risk

Due to the importance of UI quality (see Chapter 2.1) and the potential for cost reduction, many companies strive to increase their influence on the UI design of their corporate software systems. Bad UI design is a large economic risk. With business-to-customer software, the UI conveys important (emotional) values such as corporate design (CD) and corporate identity (CI). A company website must create brand awareness, convey product values, enable product search and comparison (Gundelsweiler et al. 2007b; Klinkhammer et al. 2007), allow contact with retailers, and has to increase customer loyalty. Both information visualization and navigation design are also important for corporate web sites and digital sales channels.

Web applications and eCommerce

Web applications, such as the car configurator, play an important role in the sales planning and disposal of extra equipment. Forrester audited 20 major sites, finding just 51% compliance with simple web-usability principles (Manning et al. 1998; Nielsen 1998). eCommerce sites could increase sales by 100% or more as a result of usability, but configurator-driven sites can probably increase sales by at least 500% by emphasizing usability. More importantly, they can probably avoid 9 out of 10 returns by eliminating most incorrectly designed items (Nielsen 2002).

Having a competitive edge in usability is crucial for eCommerce sites

Usability is important for all websites, but for eCommerce sites, having a competitive edge in usability is crucial. Usability-engineered sites enable users to be more efficient and productive. Ideally, online shopping should be enjoyable rather than frustrating. But reality often looks different: in Jared Spool's study of 15 large commercial sites, users could find information in only 42% of the times even though they were taken to the correct home page before they were given the test tasks (Spool 1997). Jakob Nielsen reported on another study from Zona Research, which found that 62% of web shoppers had given up looking for the item they wanted to buy online and 20% had given up more than three times during a two-month period. A website is likely to lose approximately 50% of the potential sales from the site because people 'can't find stuff'. Moreover, it will lose repeat visits from 40% of the users who do not return to a site when their first visit resulted in a negative experience (Nielsen 1998).

Customer experience is crucial

Users should not have to waste time searching for products or figuring out how to buy them; nor should they have any doubt that their personal information is secure (Donahue 2001). Poor customer experiences can have a devastating effect. Nearly half of online shoppers will abandon their shopping cart if charges are not transparent during the checkout process (Charlton 2007; Gold 2007). The average abandonment rate for shopping carts is 60%, of which 12% give up before arriving at the checkout. This means 48% of potential customers bail out at the checkout stage (Charlton 2007).

Office and intranet applications must support efficiency and effectiveness

Office applications and intranet (enterprise) portals are supposed to help employees to work more efficiently and effectively. If they do not do this, employees are unable to perform their tasks and the usage may be wholly incorrect and time-consuming, and lead finally to a reduced acceptance. Consequently benefits decrease, costs for user support explode, and the target for return on investment is missed. Inadequate use of UE methods in software-development projects have been estimated to cost billions a year in lost productivity (Landauer 1995).

Poor usability means poor employee productivity

On a corporate intranet, poor usability means poor employee productivity. The Nielsen Norman Group has found that the productivity of a corporate intranet gains from a major improvement in intranet usability by 72% on average (Schindler 2008). Accordingly, office software requires ease of learning and subjective user satisfaction to compete in the market (Shneiderman & Plaisant 2004).

The usability of embedded systems

Considering embedded systems, a wide range of different interactive systems exists in the German automotive industry, for instance, (Memmel et al. 2007a; Memmel et al. 2008e). For example, in-car software applications are supposed to support the driver while travelling, e.g. with GPS navigation or dynamic traffic information. Operating such systems must never compromise road safety, and the respective UIs must provide intuitive and easy-to-use navigation concepts to reduce driver distraction to the lowest value possible (Rudin-Brown 2005). A famous car manufacturer, for example, received negative feedback for the UI of its in-car information system and endangered the company's reputation (Nielsen 2004a).

Reasons for usability problems

Several studies have shown that 80% of total software-maintenance costs are related to users' problems with the software system and do not derive from technical software misconceptions (Boehm 1991). Among them, 64% are usability problems (Landauer 1995) that relate to the UI of a software product. They occur because some usability requirements will not be discovered until the software has been implemented or deployed. As noted by (Folmer & Bosch 2004), this is caused by usability requirements that are weakly specified, the application of usability and RE techniques that have only limited ability to capture all requirements, changing requirements, and usability testing that is performed too late due to a lack of early assessment tools.

Usability and business growth

From a financial perspective, this sorry story becomes even more obvious. (Landauer 1995) provides evidence that computers have not improved the efficiency and quality of most information work, and that difficult-to-use computer systems are a major factor in restricting economic growth. In contrast, products that are able to increase employee or customer satisfaction provide the base for business growth and competitive success. (Wixon & Jones 1995) document a case study of a usability-engineered product that achieved revenues that were 80% higher than for the first release developed without UE, and 60% above project expectations.

Usability as success-critical factor

The design, the specification, and the verification of user-friendly and task-adequate UIs have become a success-critical factor in many domains of activity. As discussed, UE is most effective at the beginning of the product-development cycle, and applying human factors in the initial design can greatly reduce costly redesign, maintenance, and customer support, which can otherwise substantially eat away at profits. Nevertheless, in many organizations UI design is still an incidental or opportunistic by-product and UE methods are not sufficiently embedded in the overall software-development lifecycle. Accordingly, structured approaches (see Chapter 2.3) are required to model, specify, and build interactive systems with high usability (Metzker 2002).

2.3 Approaches to UI Development

A UI is the software component of an application that translates a user action into one or more requests for application functionality, and that provides the user with feedback about consequences of his or her action (Myers & Rosson 1992). UI development is the overall process of designing how a user will be able to interact with a software application. UI design is involved in many stages of product development, including: requirements analysis, information architecture, interaction design, screen design, user testing, documentation, and help-system design. UI designers may require skills in many areas, including: graphic design, information design, SE, cognitive modelling, technical writing, and a wide variety of data collection and testing techniques (Preece et al. 2002). In accordance with this diversity, various approaches for developing usable interactive systems are imaginable. The circumstances for UI specification are similar, which is why a closer look at UI development approaches is intended to help the determination of new specification methods.

Approaches to UI design

(Wallace & Anderson 1993) presented four main approaches to UI design that developed due to discipline-specific perceptions: the craft approach, cognitive engineering, enhanced SE and the technologist approach (see Table 1).

Table 1: General approaches to UI design; based on (Wallace & Anderson 1993; Reiterer 2000)

Criteria	**Craft approach**	**Cognitive-engineering approach**	**Usability-engineering approach**	**Technologist approach**
Philosophy	Craft: design through skill and experience	Applied science: apply theory of human information processing	Engineering: incorporate HCI into SE	Engineering: automate or support the engineering process
Source of Quality	Talent	Theory	Methods	Tools
Character	Monolithic, evolutionary	Structured transformation	Structured transformation	Black-box generation or design-aid tools
Focus	Analysis - Implementation - Evaluation	Evaluation, Analysis	Analysis	Implementation
Role of Practitioner	Craftsman, Artist	Psychologist, Ergonomist, Usability specialist	Software developer, analyst	Software tools developer
Methods and Tools	Brainstorming, prototyping	Evaluation methods and tools, Task analysis methods	Enhancement of traditional software engineering methods	Formal grammars, Generators, UIMS, Style Guides, Class Libraries,

The craft approach

With the craft approach, the development of the UI is based on the design skills and experience of the craftsman. As the approach relies primarily on skilled leadership, it has no specific process model. This may also lead to a lack of requirements documentation, which in turn limits traceability and transparency of design decisions. Moreover, with increasing complexity of software systems and their UIs, a figurehead-driven development methodology that relies on a single artist quickly becomes overstrained. The potential to mass-produce high quality (software) products is therefore very limited (Wallace & Anderson 1993; Reiterer 2000).

The cognitive-engineering approach

The cognitive-engineering approach is grounded on theories of cognitive psychology. For the development of software UIs, the approach takes theories of human information processing and problem solving into account (Reiterer 2000). Accordingly, the approach is interested in context of use and user feedback. On the one hand, the advantage of this course of action is the opportunity to decrease costly late-cycle changes due to user and task analysis. On the other hand, studying user needs can be expensive in itself. As the approach is mainly evaluation-driven, developers lack adequate guidance on how to avoid usability issues from arising during the early stages. (Reiterer 2000) states that solving these shortcomings was one of the motives for the UE approach.

The usability-engineering approach

UE is a discipline with roots in several other basic disciplines, including cognitive psychology and SE (Mayhew 1999). The UE (see Chapter 2.4.2) approach incorporates UI-related development methods into SE (see Chapter 2.4.1). The idea behind the approach is to share UI design knowledge within the development team. In the mid 80's, new technologies such as multimedia and information visualization required that engineering teams included experts other than psychologists and computer programmers. Educational technologists, developmental psychologists and training experts also joined corporate development. Naturally, this raises the need for techniques that bridge these usually distinct disciplines. The combination of SE and UE increases the overall complexity of the development process, which consequently depends on appropriate methods and tools. It is therefore necessary to use shared notations that both programmers and usability experts can understand and apply (see Chapter 2.3).

The technologist approach

The technologist approach envisions that high UI quality comes with the allocation of appropriate software-development tools. These are intended to help developers to build usable software systems, although they have to compete with time and budget. If developers can employ the right tools, more time can be spent considering promising design solutions and making the right decisions. With the technologist approach, most of the design work remains in the hands of the software engineer, who is capable of coding the system. The approach is therefore related to manufacturing quality (see Chapter 2.1) as the outcome of the technology-driven engineering process depends heavily on the methods and tools employed, and how they support the developer in specifying, modelling and building the final system. Accordingly, the technologist approach relies heavily on the appropriate incorporation of UI design support into development tools. If this does not happen, the approach is doomed to failure.

Combination of approaches

Both the craft- and the cognitive-engineering approaches are inappropriate for facing the challenges of multifaceted software projects, as they rely on special expertise in the hands of a few. The most promising approaches to software and UI development are therefore the UE and the technologist approach. The UE approach aims to enhance common SE methodology by providing the means necessary to build usable UIs. The demand for suitable tool support as well is obvious and important. Successful designs, however, are rarely created through automated tools alone.

Usability and technologist approach

Because hardware and software become more and more aligned, the objective of UI development is to provide both visually attractive devices and usable UIs that successfully manage functionality. Therefore, an IT organization has to put particular emphasis on setting up interdisciplinary approaches to software and UI development in order to face the challenges of the market. The combination of the UE approach together with the provision of appropriate tools in the technologist approach is a very promising link for development processes.

Methods and tools

This also moves UE methods and the tools of technologists to the centre of research for groundbreaking UI specifications. In order to define the best practice for a shared course of action, the main contributing disciplines have to be analyzed in detail. Their compatibility is fundamental for a successful UI-specification process.

2.4 Disciplines Participating in UI Development Processes

Multidisciplinary team setup

Due to the importance of understanding how users act and react while working with computers, various stakeholders such as psychologists and sociologists (see Table 1) have become involved in design processes. Because of the growing significance of implementing aesthetically pleasing designs in different media, graphics designers, animators, product designers or marketing experts become members of development teams (Preece et al. 2002).

Challenges and cause of breakdown

But setting up development teams with people from different backgrounds has problems as well as advantages. Actors with different expertise typically interact with different means of communication and might focus on different aspects when talking about design problems (see Chapter 3). In practice, multidisciplinary teams often prevent themselves from being successful. Misunderstandings and communication breakdowns can significantly slow down progress. Moreover, if the experts involved do not know the information requirements of other team members, sharing knowledge and design thoughts with the right people is extremely difficult.

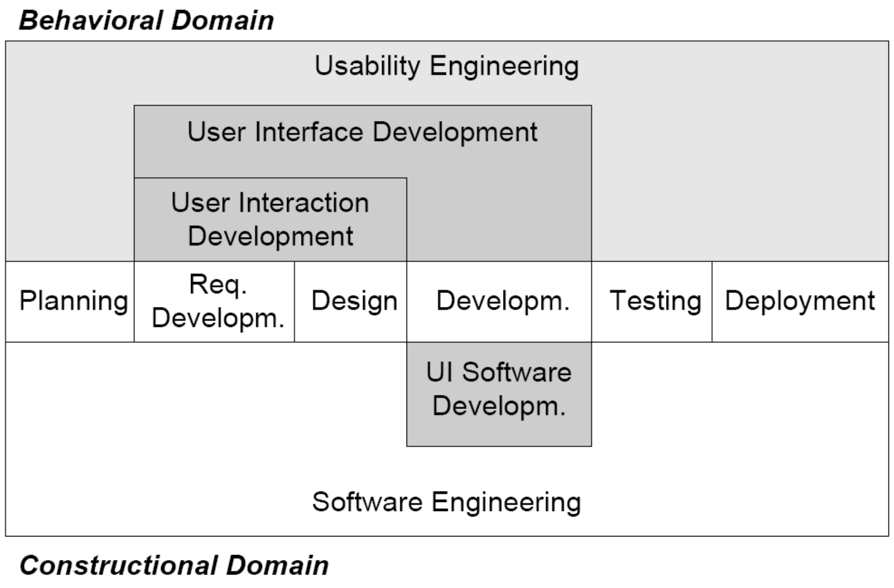


Figure 4: Scope of usability and software engineering process concepts (Metzker 2005)

Software and usability engineering

In multidisciplinary team setups, SE and UE contribute the most to UI development (Preece et al. 1994). In (Borchers 2001; Benyon et al. 2005; Sharp et al. 2007), SE and human-interface design, including human factors and the application domain, are determined as key disciplines of UI development. Software development, i.e. coding, is the core activity of SE. Yet, SE additionally comprises special activities and techniques such as analysis, modelling and managing (see Chapter 2.4.1). An analogous relationship exists between UI development and UE. (Hix & Hartson 1993) decompose the process of software and UI development into a behavioural and a constructional part to distinguish the responsibilities of both disciplines (see Figure 4). The behavioural part describes activities related to the interaction between the user and the system. The constructional part is labelled as UI software development. Here, interaction is described in terms of algorithms, data-flow models, UI widgets, and UI description languages.

Opportunity for collaboration

Following this separation of concerns, the actual development is the minimum interface between the disciplines, but the complexity of modern software systems requires much earlier and more systematic collaboration. Moreover, the early stages are especially critical for UI specification, which typically ends before the implementation starts. To determine the opportunity for collaboration, the main contributing disciplines are presented in the following pages.

2.4.1 Software Engineering

"Software engineering is an approach to software development that involves defining application requirements, setting goals and designing and testing in iterative cycles until goals are met." (Mayhew 1999), p. 3

Key properties of software engineering

SE is the science that deals with all problems related to the design, implementation and maintenance of software systems. SE is founded on the ideas of structured programming: programmers first define the major structures of a software system and then recursively decompose each structure into substructures. The driving vision is to gain control over design activities by making the software-development process explicit and systematic.

(Sommerville 2004) distinguishes two key properties of SE:

- **Technical engineering**: Technicians build systems and employ theories, methods and tools. They try to solve the problems at hand, even if no appropriate method or tool is available. They know the financial and organizational constraints and try to find solutions inside the scope of the environmental context.
- **All aspects of software development**: Besides technical processes, software engineers also consider project management and the development of new processes, methods and tools that will help to face future challenges in software development.

The waterfall model

An early model for SE was the waterfall model. It organized software development into a series of modular phases, beginning with the analysis of functional requirements and continuing through software design, implementation, testing and maintenance. In this process, each phase produces one or more documents that are passed on as a specification for the work of the next phase. The software-design stage, for example, produces a design specification that is used during the implementation.

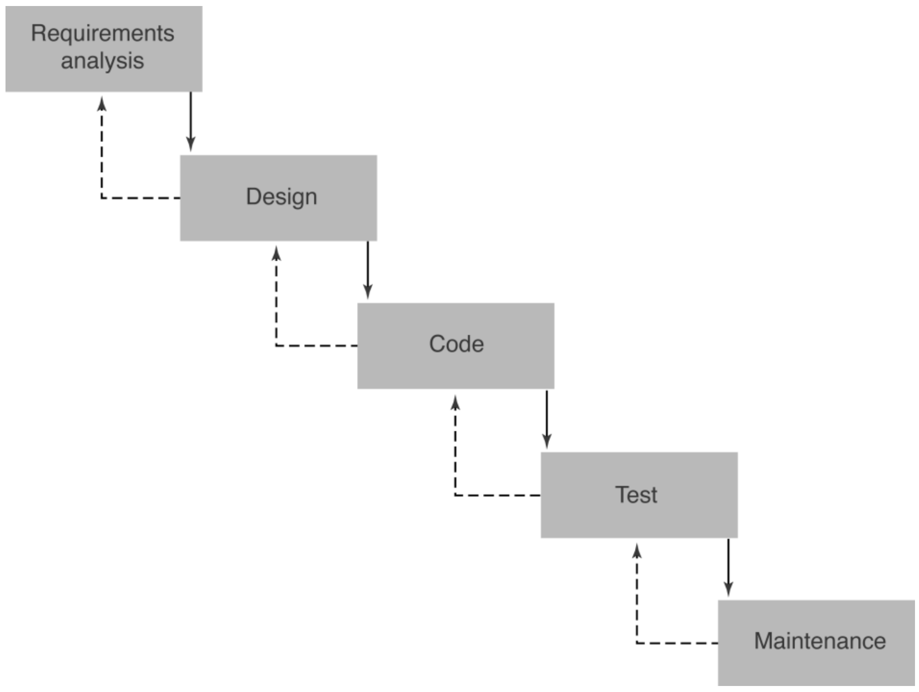


Figure 5: The Waterfall Model of software development; as presented in (Sharp et al. 2007)

Shortcomings of linear development

Separating software development into different modules is a good idea, but software development requires multiples views and working with trade-offs and competing design ideas. Many requirements first emerge during later stages of design and cannot be fully specified in advance. Therefore, a linear flow of design specification is unlikely to work for complex application domains. With the waterfall model, every step must be completed before the next phase begins (Sommerville 2004). But requirements change over time, as businesses and the environment in which they operate change rapidly (Preece et al. 2002). Moreover, in reality the different stages typically overlap. This demands some degree of iteration in software development (Rosson & Carroll 2002), but the idea of iteration was not successfully circulated among authorities, although it was in the mind of the waterfall's creator (Royce 1970). With some feedback to earlier stages (see Figure 5), some results of review sessions were incorporated into development. Even with the iterative model, however, the opportunity to review and evaluate with users was not built into the waterfall (Preece et al. 2002).

Prototyping and iterative development

A complement to structured development is prototyping. A prototype implements ideas that are abstract, making them concrete, viewable and testable (Rosson & Carroll 2002). As outlined in Chapter 1, prototypes can be built at many phases. They can be used to evaluate requirements, high-level design, or detailed software design (Boehm 1988). In the Spiral Model (see Figure 6), the feedback provided by prototyping is used to guide further development, but importantly, it may also be used to transform or reject designs based on an assessment of development risk. In contrast to the waterfall model, not all requirements need to be defined up-front. The spiral lifecycle already incorporates the identification of key stakeholders, who can also be end-users of the software system. Key stakeholders are required to define win conditions that are later used to determine whether the software product satisfies user needs (Preece et al. 2002). In this connection, SE recognises different kinds of prototypes, such as throwaway prototypes and evolutionary prototypes (see Chapter 4).

The Spiral Model

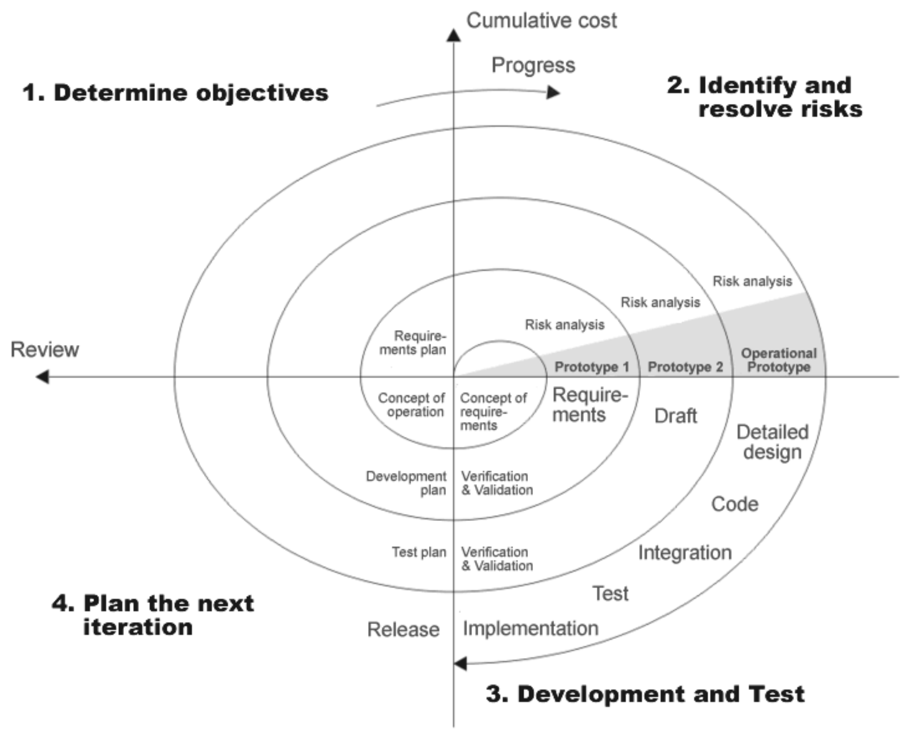


Figure 6: The Spiral Model of software development; based on (Boehm 1988)

One way to integrate prototyping with structured design is to view prototyping as a requirements-analysis method. Another approach is to assume that prototyping will be used to revise design documents. In iterative development, design documents are produced as an output of each phase, and then continually modified through prototyping and testing. The final version of the design specification describes the system that was built (Rosson & Carroll 2002).

Software-engineering methods

In general, software-development and design processes can be characterized along a spectrum from largely engineering-oriented, i.e. formal and model-based, to informal exploratory design. SE has tended to take a model-based approach (see Chapter 3) to design, using either graphical models, exemplified by the Unified Modelling Language (UML) or formal models (Fowler 2003). Moreover, SE also recognises some lightweight model-based approaches such as rapid application development (RAD) or agile methods such as extreme programming (XP), introduced by (Beck 1999), and agile modelling (Sutcliffe 2005). SE has a long tradition of producing methods, and effective uptake of such methods in the industry. Structured methods such as Structured Systems Analysis and Design Method (SSADM), a waterfall method, and their successors have evolved into the Rational Unified Process (RUP). The RE lifecycles in SE are still relatively immature. Here, the Volere process model (Robertson & Robertson 1999) is among the most successful methodologies (see Table 2).

Table 2: Contributions of software engineering to theory, methods and tools (Sutcliffe 2005)

Contribution	**Examples**	**Validation approaches**
Theory	Formal models, CSP, CCS	Formal reasoning, peer citation
Design process models	SSADM, RUP-UML, OO methods, RAD and Agile Methods, requirement methods: Volere; VORD, SceniC, scenarios, use-cases	Utility demonstration, case study, use in practice, effectiveness in practice
Product (support tools)	CASE tools, Rationale Rose, UML model editors, goal modellers, model checkers	Utility demonstration, case study, use in practice, effectiveness in practice
Analysis methods	Ethnographic studies, some surveys	Experiments, insight gained, triangulation of results

As exemplified by the lifecycles of the waterfall model and the spiral model, the fundamental activities of software processes are requirements analysis, requirements specification, software implementation (coding), and validation.

Functional and non-functional requirements

In SE, the requirements analysis differentiates between different kinds of requirements as introduced in Chapter 1. Software requirements, on a higher level, describe what the final system has to do and define constraints with regards to functionality and implementation. Functional requirements relate to services the software has to provide, while non-functional requirements characterize them in terms of efficiency, effectiveness, scale and usability. User requirements are analyzed for discussions with end-users and are usually externalized with the help of natural language and visual notations. In contrast, system requirements describe functionality in detail and are developed as a means of communication among programmers. The requirements specification summarizes all developed artefacts, and all participating parties must be able to understand and review its content.

Modelling and UML

The process of gathering and analyzing an application's requirements, and incorporating them into a program design, is a complex one. Many methodologies exist that define formal procedures specifying how to go about it. UML, which is widely supported by industry, is methodology-independent. Regardless of the methodology used to perform analysis and design, UML can express the results. UML 2.0 defines thirteen types of diagrams (see Table 3), divided into three categories: six diagram types represent static application structure; three represent general types of behaviour; and four represent different aspects of interactions (OMG 2007):

- **Structure diagrams** include the Class Diagram, Object Diagram, Component Diagram, Composite-Structure Diagram, Package Diagram, and Deployment Diagram.
- **Behaviour diagrams** include the Use-case Diagram (used by some methodologies during requirements gathering), Activity Diagram, and State Machine Diagram.
- **Interaction diagrams**, all derived from the more general Behaviour Diagram, include the Sequence Diagram, Communication Diagram, Timing Diagram, and Interaction Overview Diagram.

Table 3: The diagrams of UML 2 (Ambler 2004b)

Diagram	**Description**
Activity Diagram	Depicts high-level business processes, including data flow, or models the logic within a system.
Class Diagram	Shows a collection of static model elements such as classes and types, their contents, and their relationships.
Communication Diagram	Shows instances of classes, their interrelationships, and the message flow between them. Communication diagrams typically focus on the structural organization of objects that send and receive messages. Formerly called a Collaboration Diagram.
Component Diagram	Depicts the components that compose an application, system, or enterprise. The components, their interrelationships, interactions, and their public interfaces are depicted.
Composite Structure Diagram	Depicts the internal structure of a classifier (such as a class, component, or use-case), including the interaction points of the classifier to other parts of the system.
Deployment Diagram	Shows the execution architecture of systems. This includes nodes, either hardware or software execution environments, as well as the middleware connecting them.
Interaction Overview Diagram	A variant of an activity diagram that overviews the control flow within a system or business process. Each node/activity within the diagram can represent another interaction diagram.
Object Diagram	Depicts objects and their relationships at a point in time, typically a special case of either a class diagram or a communication diagram.
Package Diagram	Shows how model elements are organized into packages as well as the dependencies between packages.
Sequence Diagram	Models the sequential logic, in effect the time-ordering of messages between classifiers.
State Machine Diagram	Describes the states an object or interaction may be in, as well as the transitions between states. Formerly referred to as a state diagram, state-chart diagram, or a state-transition diagram.
Timing Diagram	Depicts the change in state or condition of a classifier instance or role over time. Typically used to show the change in state of an object over time in response to external events.
Use-case Diagram	Shows use-cases, actors, and their interrelationships.

UML and UI development

For programmers, UML diagrams are usually very helpful in producing a shared understanding and specifying backend system capabilities. Many practitioners envision UML as being the appropriate vehicle for UI development as well. In general, developers probably need only 20 percent of the models to do 80 percent of the modelling work (Ambler 2004b). UML is undoubtedly very strong for specifying the structure and the functionality of the application, but it is seldom used to specify usability-related information (Sutcliffe 2005). Various UML-based UI description languages (UIDL), for example (Abrams & Phanouriou 1999; Silva & Paton 2003; Carter et al. 2005), have been developed in order to overcome this limitation, but the extensions also lead to more formalism and decreased understanding for business people (Löwgren & Stolterman 2004; Zetie 2005).

UML and UI specification

UML is undoubtedly one of the most powerful modelling languages for SE. For UI design issues, however, UML can only describe interactions formally and is unable to visualize real UI behaviour. (Constantine & Lockwood 2001) note that the traditional use of scenarios within interface design, that of capturing the context, goals, intentions and values of a user, and the use of artefacts, is quite different from how use-case diagrams are used in UML. In UML, use-cases do indeed concern users, but only in a fragmented way, in which each user goal is dealt with in isolation. Hence, instead of capturing the user's goals and how the user intends to reach them by means of the UI, UML models the system interface and the required system functions. Hence, UML use-cases are not designed, nor are they useful, for understanding and identifying with the user. This is consistent with a general problem of UML: the focus is on technically-oriented design, including architectural and implementation issues (Nunes & Cunha 2000), not human-centred specification of functionality and behaviour. As a result, UML loses the global view of the application as users perceive it. This makes it difficult to envisage how the interaction will take place, which is essential in order to plan for, and evaluate, the quality of use of the final product. Consequently, the business user and IT analyst may think that they both agree on a design, only to discover down the line that they had very different detailed implementations and behaviours in mind (Zetie 2005). On the whole, UML on its own is not adequate for specifying both the look *and* feel of interactive UIs.

SE and usability

It can be stated that most SE methods focus on maintainability and reliability of software products. Most software-development methods and tools should help to develop software that successfully addresses these properties. The efficiency and effectiveness with which a software product can be applied depends on various factors. Among them, UE is a stand-alone discipline that must be considered in detail (Sommerville 2004). Concerning UI development, many organizations do not employ specialists that work on UI-related issues. Accordingly, software engineers often have to take care of UI design, too (Sommerville 2004), and are required to use this to contribute to the success of the software system. This raises the additional need for a good understanding of UE.

2.4.2 Usability Engineering, Human-Computer Interaction & Interaction Design

Usability describes the extent to which a product, which is used by specific users in a specific context, fulfils specific tasks in an effective, efficient, and satisfactory way.

"Usability is a measurable characteristic of a product user interface that is present to a greater or lesser degree." (Mayhew 1999)

Definitions of usability

The term usability is not defined homogeneously. (Shneiderman 1992) defines usability from the user perspective as: ease of learning, speed of user task performance, low user error rate, and subjective user satisfaction.

DIN EN ISO 9126

In (DIN EN ISO 9126 1991), usability is defined as the capability of the software product to be understood, learned, used by, and attractive to, the user, when used under specified conditions.

DIN EN ISO 9241

In (DIN EN ISO 9241-11 1998), usability is described as the extent to which a product can be used by a specified set of users to achieve specified goals (tasks) with effectiveness, efficiency and satisfaction in a specified context of use.

Perspectives on usability

From the perspective of the software-development process, a helpful definition of usability is one that defines it as being constituted by a set of attributes of the software product that can be optimized by a structured engineering process (Metzker 2005). The modern view of usability is made up of three main perspectives: human performance, learning and cognition, and collaborative activity.

Human performance

Usability entered software development during both the early and late stages of design. In order to understand requirements, end-users and context of use are analyzed. To achieve a high UI quality (see Chapter 2.1), designers also have to respect human diversity with regard to physical, cognitive and perceptual abilities to allow accessibility for all type of users (see cognitive approach to UI development in Chapter 2.3). The question of whether or not proposed software solutions meet design specifications regarding human performance is tested during human-factors evaluation.

Human-computer interaction

The increasing prominence of personal computers in society made usability issues more visible. When psychologists, sociologists and philosophers started to analyze how people learn to use computers, a new area of shared interest between computer science and cognitive science emerged. It was called human-computer interaction (HCI).

"Human-Computer Interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them." (Hewett et al. 1992), p. 5

The scope of HCI

HCI is a diverse scientific and applied field where the focus is on how people use computers and how computers can be designed to help people use them more effectively. HCI is concerned with how interactive information systems can be designed in a user-friendly way, which involves, for example, user-oriented analysis and modelling of requirements, and principles, methods, and tools for the design and certification of interactive systems. Moreover, HCI is employed with the visualization and exploration of information spaces. With regards to the evaluation of information systems and contexts of use, HCI is propelling the integration of user-friendly design of information systems in software development.

HCI methods and tools

Popular topics of HCI are the development of design tools and design methods, the design of information architecture, interaction techniques, and interaction devices. With HCI, usability was no longer seen as just assuring the quality of finished systems, but became a comprehensive process that has its own design and development techniques (see Table 4). HCI has a model-based tradition of design driven by task analysis. Besides the strong focus on user and task modelling, scenarios are gaining increasing success in UI development. The discipline has produced a large number of international standards, such as (DIN EN ISO 13407 1999). The discipline has therefore had significant influence on UI development practice. HCI can also claim success in evaluation and quality assurance. A very large set of evaluation methods is widely practised (Nielsen 1993; Nielsen 1994). Several theories, notably ACT-R (Anderson et al. 1997), or structural methods such as GOMS (Gray et al. 1993), have proven to be successful.

Table 4: Contributions of HCI to theory, methods and tools (Sutcliffe 2005)

Contribution	**Examples**	**Validation approaches**
Theory	ACT-R, EPIC, GOMS, activity theory, distributed cognition	Experiments, computer models, design influence, principles
Design-process models	Task-analysis methods, user-centred design, evaluation methods, scenario-based design, principles and guidelines, ISO standards.	Usability, utility, use in practice, effectiveness in practice
Product (support tools)	User interface design environments, UI toolkits	Usability, utility, use in practice, effectiveness in practice
Analysis methods	Ethnographic studies and experimental analysis of UI designs	Experiments, insight gained, triangulation of results

From HCI to usability engineering

Within HCI, the discipline of UE emerged as a subsidiary branch. The term UE, which has already been used during the earlier chapters of this thesis, was coined by usability professionals from Digital Equipment Corporation (Good et al. 1986) and needs to be explained in detail. UE aims at the development of methods, tools, and procedures that are important for the design of usable software products. Its primary concern is usability, which is an important software product quality (see Chapter 2.1) and a non-functional requirement in software development (see Chapter 2.4.1). UE is concerned with the analysis of users and tasks, and of the context of use. Accordingly, UE is related to RE (as part of SE).

UE defined

(Mayhew 1999) defines UE as the discipline that provides structured methods for achieving usability in UI design during product development. According to (Faulkner 2000) UE is an approach to the development of software and systems that involves user participation from the outset and guarantees the usefulness of the product through the use of a usability specification and metrics.

"Usability Engineering is a discipline that provides structured methods for achieving usability in user interface design during product development." (Mayhew 1999), p. 2

UE methods

From the very beginning, UE focused on the design of the UI and on engineering effective interactive presentations of information and functions. Usability has therefore developed its own terminologies and methods (see Table 4). The importance of usability and the UI led to the inclusion of user-interaction scenarios into design specifications. Later, user scenarios began to appear at the front of specification documents (Rosson & Carroll 2002). Scenario-based design (Carroll 2000a; Carroll 2000b; Rosson & Carroll 2002) has emerged as an important design approach. Today, usability engineers help to determine what functionality is required and adequate for the problem, as well as how the functionality should be presented to the user. For this purpose, one of the most important aspects of UE is the formulation of usability criteria. Consequently, the methodological focus is mainly on evaluation procedures. Most procedures are both iterative and based on continuous UI prototyping. The development model of user-centred design (Norman & Draper 1986), for example, is well-grounded on prototyping styles of development (Sutcliffe 2005). Consequently, usability-engineered products are developed in an interplay of design and evaluation (Mayhew 1999), which is also fundamental to the lifecycle of (DIN EN ISO 13407 1999).

UE lifecycles

One of the most popular UE lifecycles was developed by (Mayhew 1999). Each phase (see Figure 7) is accompanied by important activities. During the requirements-analysis phase, project stakeholders - assembled in a multidisciplinary team to ensure complete expertise - will conduct field studies, look at competitive products, create user profiles, develop a task analysis, and define usability goals, for example.

Usability goals

Specific usability goals help to focus UI design efforts during the process by giving developers something concrete to aim for. Design efforts can then focus on solutions that can be successfully assessed against the defined usability goals. Consequently, usability goals drive decision-making and evaluation. (Mayhew 1999) divides usability goals into qualitative and quantitative ones. Qualitative usability goals describe properties such as ease of learning and ease of recall. It is usually more difficult to decide whether the goals were met in the final design, as they are not quantified. Quantitative usability goals (or performance / satisfaction goals), in contrast, are objective and measurable. They describe task completion times, error rates and ease of use (efficiency and effectiveness), to give some examples.

Some very important usability goals were defined by (Preece et al. 2002). Effectiveness is a measure to determine how good a system is at doing what it is supposed to do. This also involves the user's capability to work efficiently with the interactive software system provided. Accordingly, efficiency refers to the way a system supports the users in carrying out their tasks. Safety involves protecting the user from dangerous conditions and undesirable situations. This involves preventing the user from making serious errors by reducing the risk of incorrect input, and providing users with various means of recovery should they make errors. Having good utility is important in terms of the functionality of software. All features required to execute certain tasks should be available to the user. Above all, the software system should be easy to learn. This means that users should be able to work with software without too much effort. This also relates to the issue of memorability. Once learned, it should be easy to remember how to use a system. Especially when a software application is only used infrequently, users should not have to relearn it. Users should therefore be supported in remembering how a system is to be used.

Design, testing and development

In the further course of the UE lifecycle, during the early conceptual design phase, developers begin to brainstorm design concepts and metaphors, develop screen flow and a navigation model, begin design with paper and pencil, create low-fidelity prototypes or high-fidelity detailed design. First externalizations of design vision will be evaluated by doing walkthroughs of design concepts and conducting usability testing on low-fidelity prototypes. Due to the iterative character of the process, design is continuously refined due to ongoing usability testing. During the implementation of the final system, the usability team works closely with the delivery team. They will, for example, make sure that basic usability and design principles such as affordance, constraints, mapping, visibility and feedback are taken into account (Norman 2002). In Mayhew's UE lifecycle, standards and guidelines – as results of design work - are documented in style guides. Finally, the team creates a design specification (see Chapter 2.8). After the deployment of the software product, usability issues will be reported back to the team. With surveys or field studies, developers obtain information about actual use.

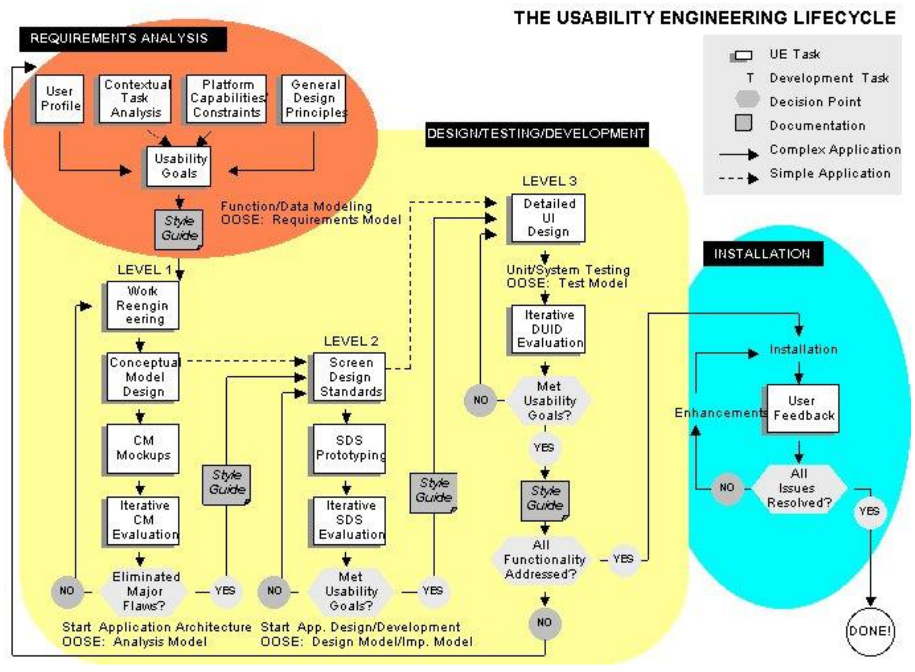


Figure 7: The UE lifecycle as defined by (Mayhew 1999)

Nomenclature of UI-related practice

As outlined in the previous chapters, UI development is a highly interdisciplinary process. Accordingly, so are HCI and UE. Due to this diversity, many different names for applied practice can be found in the literature. They can, for example, be distinguished by looking at typical job descriptions.

From interface designers to information architects

HCI professionals are known as interaction designers, usability experts, graphic designers, user-experience experts, etc. (Belenguer et al. 2003). With a greatly diversified range of interactive products and the growing need to 'get the UI right', a variety of job descriptions has emerged (Preece et al. 2002):

- Interaction designers - people involved in the design of all the interactive aspects of a product.
- Usability engineers - people who focus on evaluating products, using usability methods and principles.
- Information architects - people who come up with ideas of how to plan and structure interactive products.

- User-experience designers - people who do all the above but who may also carry out field studies to influence the design of products.

Although UE is a well-established and broadly used term, (Preece et al. 2002) state that the discipline of interaction design is the umbrella term covering all of these aspects that are fundamental to all disciplines, fields, and approaches concerned with researching and designing computer-based systems for people.

Interdependent fields

Due to this perspective of interaction design (IxD), the term must be discussed in more detail. According to (Preece et al. 2002), IxD can be viewed as essential to disciplines, fields, and approaches that are concerned with researching and designing interactive systems.

"By interaction design, we mean designing interactive products to support people in their everyday and working lives" (Preece et al. 2002), p.6

Due to its relationship to HCI and SE, IxD is also studied from a wide variety of perspectives from many diverse areas, including (among others): computer science, psychology, ergonomics, information sciences, graphic design, and sociology.

Interaction design as design discipline

One interpretation is to view IxD as a design discipline, distinguished by its focus on the digital design materials: software, electronics and telecommunications. As a design discipline, it is more closely affiliated with industrial design and architecture than with engineering and behavioural science. This definition involves design disciplines such as industrial design, graphic design and architectural design. If software were something that the computer user just looked at, rather than operated, traditional visual design would still be at the centre of software design. If the spaces were actually physical, rather than virtual, then traditional product and architectural design would suffice. But computers have created a new medium, one that is both active and virtual. Designers in the new medium need to develop principles and practices that are unique to the computer's scope and fluidity of interactivity (Winograd 1997). Hence, interaction designers focus on the flow of interaction, the dialogue between person and computer, how input relates to output, stimulus-response compatibility, and feedback mechanisms. This is in contrast to a visual or graphics designer, who may be trained in designing visualizations for static media but not necessarily in the dialogue that is present in all interactive media (Usability First 2005). The goal of this perspective, therefore, is bringing design to software as well, as explained in (Winograd 1996; Löwgren 2008).

Interaction design as extension of HCI

The other interpretation of IxD is to see it as an extension of HCI. HCI was originally oriented mainly towards field studies (of existing user populations, their cognitive traits and current practices, for example) and evaluations (of an existing product or a proposed product concept, for example). However, it was found that the impact on the resulting products, and ultimately on the benefits for the users, would be greater if HCI practitioners and researchers engaged in the design rather than merely pointing out usability problems after the fact. Hence, the HCI palette of methods, tools and responsibilities was extended to encompass more creative and generative activities (Löwgren 2008). In this context, interaction design also contrasts with information architecture. An information architect looks at the organization of information to make the structure of a complex system easy to conceptualize and navigate, but is not usually focused on low-level interactions, for instance. For example, an information architect may design the structure of an entire website, but not have as much interest in the design of individual pages and how users interact with forms and other controls (Usability First 2005).

Converging perspectives on IxD

The diversity of technology and application areas has given rise to a much wider set of concerns related to the design of interactive systems. In addition to providing systems with good usability, it is also important to address users' feelings about software. The focus of software systems changed towards a dominance of discretionary use for fun, pleasure and recreation over instrumentally motivated use for solving work-related tasks. The increasing amount of design activity and the increasing focus on what HCI calls user experience are therefore the two main factors motivating the growing tendency for HCI to adopt IxD as a more appropriate label for the field.

The tendency of the convergence of IxD as a design discipline and IxD as an extension of HCI can also be witnessed in hiring policies and work practices in professional IxD contexts as well as in the increasing amount of cross-disciplinary research where designers collaborate with scholars from an HCI background (Löwgren 2008).

Interaction design as label for HCI

Due to this phenomenon, the discipline of IxD will be used throughout this thesis to relate to the modern mixture of HCI and UE with aspects of design. Since the gap between SE and HCI (see Chapter 3) becomes less significant when the HCI specialist is also a strong programmer and analyst (Seffah et al. 2005b), we use the term interaction designer as a synonym for an HCI expert who combines knowledge of usability, graphics design, and coding. This is consistent with the interpretation of (Preece et al. 2002; Löwgren & Stolterman 2004; Sharp et al. 2007; Löwgren 2008). The interaction designer is responsible for developing the content, behaviour, and appearance of the interaction design. People in this role are directly responsible for ensuring usability, including user performance and satisfaction. They are concerned with critical design issues such as functionality, sequencing, content, and information access, as well as such details as what menus should look like, how forms should be formatted, which input device is appropriate, and how to ensure consistency across an interface. A major part of the interaction designer's job is also concerned with setting measurable usability specifications, evaluating interaction designs with users, and redesigning based on analysis of users' evaluations of an interface (Hix & Hartson 1993; Marion 1999).

User-experience goals

In IxD, instrumental quality concepts such as usability and usefulness lost out in relative importance to experiential concepts addressing the non-instrumental qualities of use (including aesthetic and social qualities). Accordingly, in their book "Interaction Design - Beyond Human-Computer Interaction", (Preece et al. 2002) introduce user-experience goals. At a top level, usability goals and user-experience goals can be distinguished as presented in Table 5. The right strategy in designing interactive systems should take both usability and user-experience aspects into account. Software and UI development therefore have to come up with design approaches that can also guarantee quality in terms of functionality, usability and user experience (Jordan 2000). Usability goals are concerned with meeting specific usability criteria (e.g. efficiency) and user-experience goals are concerned with quantifying the quality of the user experience (e.g., being aesthetically pleasing).

Table 5: Usability and user-experience goals (Preece et al. 2002)

Usability Goals	**User-experience Goals**
Effective to use, efficient to use, safe to use, have good utility, easy to learn, easy to remember how to use	Satisfying, aesthetically pleasing, enjoyable, supportive of creativity, engaging, pleasurable, rewarding, exciting, fun, entertaining, provocative, helpful, surprising, motivating, enhancing sociability, emotionally fulfilling, challenging

Apparent usability and emotional design

Besides developing for effectiveness and efficiency, user experience is for example concerned with designing UIs that are aesthetically pleasing and enjoyable to use. With regards to HCI, the joy of use is a emotional component that extends the scope of UE in terms of aspects of emotional usability (Reeps 2006). A variety of terms for describing user experience and user-experience goals has emerged, especially apparent usability (Kurosu & Kashimura 1995) and emotional design (Norman 2004). (Blythe et al. 2004) distinguish between fun and enjoyment i.e. pleasure. According to their definition, enjoyment depends on the grade of intensity with which a user is absorbed by a task. Fun, conversely, is explained as a kind of distraction. Good design can make routine work fun, while creative and non-routine work has to attract and challenge the user in order to be enjoyable. Another modern term for user experience is hedonic quality and describes the stimulation of the user (e.g.

with interesting features) and his identification with a product. Hedonic quality is also related to evocation, as users especially enjoy using products that have a personal meaning (Hassenzahl et al. 2000 ; Reeps 2006). HCI experts must identify ways to introduce novelty and surprise with their interfaces without sacrificing too much ergonomic quality (e.g. familiarity). Software designers must find a subtle balance of both quality aspects rather than maximizing them independently of each other.

Interaction design process

The process of IxD (Preece et al. 2002) can be separated into a few main phases of development that can be inferred from (DIN EN ISO 13407 1999): (1) identifying needs and establishing requirements for the user experience, (2) developing alternative designs to meet these, (3) building interactive prototypes and, at a later stage, the final system and UI, which can be communicated and assessed, (4) evaluating what is being built throughout the process and the user experience it offers. All in all, the phases of IxD are very similar to the UE lifecycle of (Mayhew 1999). The four phases can also be seen as the common denominator of most usability-related lifecycles.

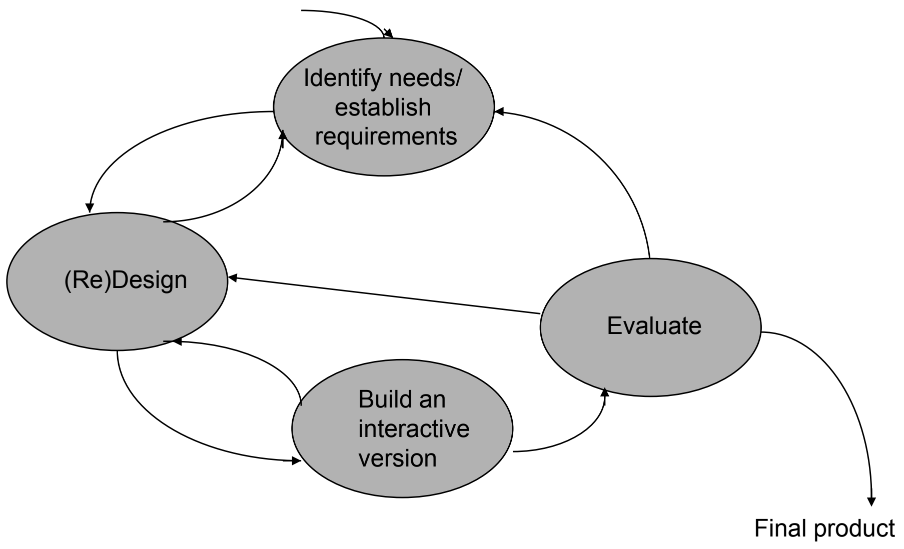

Identify needs/ establish requirements
(Re)Design
Evaluate
Build an interactive version
Final product

Figure 8: Interaction design process model (Preece et al. 2002)

2.4.3 Business-Process Modelling

Integration of business people

Developing UIs in an industrial context also requires the integration of business people. They usually want to take part in the analysis, conception, and development of information systems as well as the relevant business-process reengineering (Malhotra 1998). Typically, such activities are driven by managers most of whom have a background in economics and marketing. Involving people from the general business world therefore integrates the majority of other stakeholders i.e. those outside the realms of UE and SE.

Processes in everyday life

Processes are an integral part of everyday life. Every time someone performs any kind of action, that person carries out a process. But using processes effectively is often not quite straightforward. Without a good knowledge of what is actually going on, it is difficult to perform a task adequately.

Processes in organizations

It is not just people that follow processes, but that every organization relies on a number of processes all to function properly. Process modelling is therefore arguably one of the most important aspects of an organization in terms of the management and control of activities. These activities range from high-level business activities right down to detailed technical processes. Although the term ‘process’ is widely used, it can have many different meanings. A process can be (1) a series of actions, changes of functions bringing about a result, (2) a series of operations performed in the making or treatment of a product, (3) a set of interrelated activities that transforms inputs into outputs. (Holt 2005) describes a process as an approach to doing something that consists of a number of activities, each of which will produce and/or consume some sort of artefact.

Business-process modelling

The term business-process modelling (BPM) describes any process-modelling exercise that is performed in order to enhance the overall operation of a business. Thus, the activities of BPM serve the purpose of representing both the current and future processes of an enterprise, so that the current process may be analyzed and improved. BPM is typically performed by business analysts and managers who are seeking to improve process efficiency and quality. Business analysis is very much concerned with RE practice, and is therefore related to the early stages of UE and SE. The process improvements identified by BPM often require IT involvement, which underlines the need for a common denominator (see Chapter 4).

Kinds of processes

Processes can occur in many different forms. When a process is written down, it usually has the form of a standard, a procedure or a set of guidelines. (Holt 2005) distinguishes the following levels of detail:

- Very high-level processes, for example international standards.
- High-level processes such as industry standards. An industry standard is driven by the actual industry and usually does not have the recognition of international and national standards. However, an industrial standard may achieve international recognition, as in the case of UML.
- Medium-sized processes, such as company standards and processes. Large companies in particular usually have well-defined process models and standards
- Low-level processes, such as in-house procedures. A procedure describes how a process must be implemented. A single process can be implemented in different ways using different procedures.
- Very low-level processes, such as guidelines. They typically show a preferred or best-practice approach to carry out a procedure. This includes specific methods and methodologies.

Most organizations today attempt to visualize the way they work in order to teach their workers and to preserve corporate knowledge. These kinds of representations are called business processes. Information systems make a large contribution to process improvement. The infrastructure available for processing and communication helps to reduce costs and time. IT support for business-process management is increasing, and many solution providers are offering frameworks to develop customized applications instead of generic ones that require organizations to adapt in order to be able to adopt them (Sousa et al. 2008c).

Difficulties in process modelling

An organization has to overcome several difficulties to be able to present its corporate structures and knowledge in terms of well-defined business processes.

Simplified representation

First of all, process descriptions must not be too long. An extensive and complex process description will overwhelm users. It is therefore necessary to provide simplified representations of process descriptions that can, at a higher level, be understood at a glance. On the other hand, a process description must not be too short either. Moreover, the models of an organization's processes will relate to, or rely on, other process models and international standards.

Adequate nomenclature

While managing the complexity of business-process models, the process descriptions have to stay closely connected to existing practice. If new processes differ too much from existing processes, they are unlikely to be accepted. Accordingly, the user terminology and technical nomenclature must be embedded in the core process model of the organization to ensure that the maximum number of people understand the process in an unambiguous way.

Awareness and perception

For people to use a process, they must be aware of the process in the first place. Most often, a process description is printed out and then buried under dozens of other documents. With tools such as ARIS (see Figure 9), it is relatively simple to make process descriptions available to all members of an organization on their electronic desktops. If people can look at the process description easily and if they are provided with different views of the process, they will understand the process and become aware of its advantages and organizational benefits.

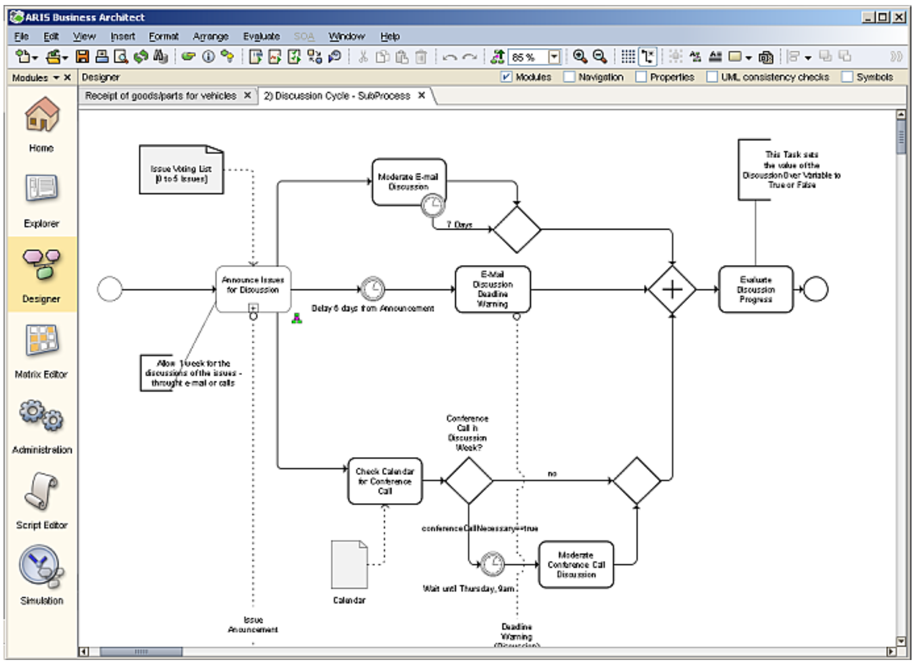

ARIS Business Architect
File Edit View Insert Format Arrange Evaluate SOA Window Help
85 %
Modules
Designer
Modules
Navigation
Properties
UML consistency checks
Symbols
Receipt of goods/parts for vehicles
2) Discussion Cycle - SubProcess
Home
Explorer
Designer
Matrix Editor
Administration
Script Editor
Simulation
Issue Voting List [0 to 5 Issues]
Moderate E-mail Discussion
7 Days
This Task sets the value of the DiscussionOver Variable to True or False
Announce Issues for Discussion
Delay 6 days from Announcement
E-Mail Discussion Deadline Warning
Evaluate Discussion Progress
Allow 1 week for the discussions of the issues - throught e-mail or calls
Conference Call in Discussion Week?
no
Check Calendar for Conference Call
conferenceCallNecessary==true
Moderate Conference Call Discussion
Wait until Thursday, 9am
Calendar
Issue Announcement
Deadline Warning

Figure 9: A business-process model in ARIS; from (IDS Scheer AG 2008a)

Business-process models and the UI

A business-process model is an important artefact, but not the only one to be considered in the definition of the application's user interaction. The roles of the people who are responsible for the activities (and for what it is necessary to do in each one) are essential requirements for designing all the UIs that support the roles they play in the organization. Therefore, when the process changes, the software should probably change accordingly. But determining what exactly needs to be changed is subject to a precise requirements control and traceability, which becomes more complex in proportion to the software's size and change frequency.

Traceability between business process and UI

In the realm of UI development, actors need to be concerned about traceability from the business process to the UI. Traceability enables business analysts to predict the impact of process changes on the user interaction and to propose changes in the processes when the user interaction is improved. This strategy accords with the definition of traceability, which is the ability to describe and follow the life of a requirement in both a forward and backward direction, providing critical support for software maintenance (Gotel & Finkelstein 1994).

If the software and UI development process lacks traceability, a great deal of time is spent on meetings to explain the business processes and to make suggestions on UI design. Moreover, quality assurance checks will be necessary to make sure that the UIs are in accordance with the business processes before they are sent to development (Sousa et al. 2008c). By analyzing business processes, potential enhancements can be recognized and UI requirements can be inferred (Gruhn & Wellen 1998). Furthermore, complex processes can be visualized by means of UI prototypes to support stakeholders' understanding of the concept (Sukaviriya et al. 2007).

2.5 Special Problems of Corporate-Development Processes

The interaction layer, as the interface between system and user, is the area where IxD, SE and BPM are required to work together in order to ensure that the resulting software product behaves as specified during RE. The interaction designer will be in charge of bringing it all together and he finally has to specify the UI. In corporate organizations, this assignment can be even more difficult. A 3-year analysis of UI development practice in industrial organizations (among them Dr. Ing. h.c. F. Porsche AG and Daimler AG) brought to light the fact that UI design remains too much a marginal activity that deserves more attention. UI-related methods are not sufficiently implied in the overall software-development lifecycle (Memmel et al. 2007a; Memmel et al. 2007g; Memmel et al. 2008e). This is consistent with the general phenomenon that UI-related methods are still widely underestimated in the software-development field.

Survey of applied tools for UI specification

In a survey of developers and business experts in the electrical engineering department of Dr. Ing. h.c. F. Porsche AG, we - together with (Bock & Zühlke 2006; Memmel et al. 2007a) - found that managers, those with responsibility for specific functions, domain experts and interaction designers apply only standard office applications for software specification (Memmel et al. 2007a). Out of 12 stakeholders we interviewed, all use Microsoft (MS) PowerPoint, 10 use MS Excel, 7 use MS Word, 6 paint their requirements with simple drawing tools and a few with more tool experience use MindManager (2) and MS Visio (1). None apply more sophisticated CASE-tools.

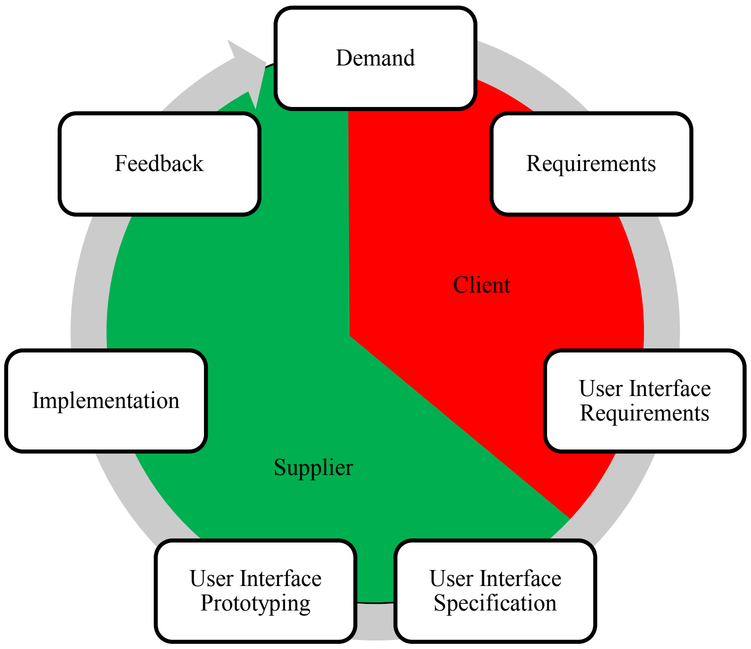

Demand
Requirements
Client
User Interface Requirements
User Interface Specification
User Interface Prototyping
Supplier
Implementation
Feedback

Figure 10: Usual assignment of client and supplier responsibility in the UI development process of organizations with a specification-driven prototyping culture (simplified lifecycle model)

The dominance of office applications

As well as users' familiarity with MS Office applications, the most important reasons for the dominance of Office applications is the difficulty of customizing or even building CASE-tools (Isazadeh & Lamb 2006) and, more importantly, by their poor usability (Jarzabek & Huang 1998) as well as the great effort needed in learning to use them and the difficulty of understanding the related terminologies. The process is therefore visually awkward, which is very disadvantageous when designing UIs (Horrocks 1999). Consequently, there is a great need for a standardized specification process. As with the use of natural language, however, employing applications such as Microsoft PowerPoint, Word, Excel or Visio also highlights their critical shortcomings for engineering interactive systems. First of all, stakeholders choose their favourite office application independently and in accordance with their individual preferences. This inevitably leads to a wide variety of formats that often cannot be interchanged without loss of precision or editability. Secondly, those who are responsible for actually coding the software system (i.e. '*real*' software developers) will use completely different tools during the implementation process. Consequently, the effort invested in drawing PowerPoint slides or Excel sheets does not help the programming of the final system. Furthermore, virtual prototypes cannot automatically be created from such specifications within the limits of justifiable effort (Bock & Zühlke 2006).

Client and supplier responsibility

Consequently, the responsibility of a client during software projects is narrowed to informal RE practice (see Figure 10). Different departments hesitate to agree upon requirements and ask potential end-users for their needs. Consequently, at this stage functional and non-functional (UI) needs are extracted from narrative business requirements. With regards to the UI, requirements for the look and feel of the software system are derived from the collected requirements. Afterwards, the UI requirements are translated into a UI specification sheet. This may be done with the help of, or solely by, an external supplier. The supplier will also be responsible for building first prototypes and finally implementing the software system. As discussed in Chapter 1, this is described as a specification-driven UI development process, which comes with several disadvantages (see Chapter 2.6).

Ultimately, we distinguish between two different families of tool users:

- **Client**: actors such as business personnel, marketing people, domain experts, or HCI experts use office-automation applications such as word processors and presentation software (Memmel et al. 2007a; Memmel et al. 2007g; Memmel et al. 2008e) to document a user's needs and their contexts of use (Calvary et al. 2003) in order to define the problem space. They will translate the needs as perceived from the real world, and their contextual conditions, into general usage requirements and evaluate their work at several quality stages. At this stage, responsibility is typically shared with, or completely passed on to, a supplier.

- **Supplier**: actors with a sophisticated IT background (e.g., programmers or designers) translate usage requirements into UI and system requirements, deliver prototypes, and conclude the process in a UI specification. They prefer working with UI builders, and - using more formal, precise and standardized notations - they narrow the solution space towards the final UI.

2.6 Shortcomings of, and Changes Required to, Corporate UI Development

Groups of stakeholders

As explained in the previous chapters, different groups of engineering stakeholders need to be linked by capable tools. Product managers define strategic goals and benchmarks and gather requirements together with interaction designers. The latter usually develop a navigation concept, create appropriate dialogue sequences, and are responsible for early- and late-stage usability evaluations. Interaction designers need to guarantee alignment with corporate design and corporate identity and, together with other stakeholders, write a style guide. Technical experts define test settings for assessing the software product in simulation frameworks on desktop computers as well as – taking the example of the car industry - on the embedded system in a prototype vehicle. IT suppliers are usually assigned to build the specified software system (see Chapter 2.5). The difference between different categories of actors tends to result in a mixture of formats. This makes it difficult to promote concepts and creative thinking down the supply chain without media discontinuities and loss of precision (Memmel et al. 2007a).

Text-based UI development as common ground of UE, SE & BPM

The text-based documents that are produced easily in the initial stages quickly reach a complexity of several hundred pages and are therefore hard to maintain. Due to the various formats used, there is a critical danger of loss of precision and misinterpretation. But most importantly, those responsible for actually prototyping and implementing the system will use completely different tools. They need to translate text-based documents into SE models and code. Hence, work is duplicated i.e. that which has already been textually described is then reworked with code. Consequently, the effort invested in drawing PowerPoint slides or Excel sheets helps neither prototyping nor the implementation of the final system.

"Textual requirement specifications are not enough, as users do not have the time to get deeply involved in the system and therefore need to be assisted during the specification of their requirements. Visual requirements specification seems to be well-suited for this challenge" (Rashid et al. 2006)

Negative factors of current UI specification practice

Due to the absence of sophisticated modelling languages, the transition from abstract text-based system descriptions to a running system is a black-box process. This makes it difficult to crosscheck the implementation against the underlying requirements at the quality gateway. The following negative factors therefore contribute to UI development failure:

- The difficulty in switching between abstract and detailed models due to a lack of interconnectivity (Campos & Nunes 2007). This leads to difficulties in travelling from problem space to solution space, which in turn makes the overall UI development a black-box process.
- The burial of mission-critical information in documents that are difficult to research and have very awkward traceability. Experts are overruled when the UI design rationale is not universally available in the corresponding prototypes.
- The perpetuation of unrecognized cross-purposes in client and supplier communication, which can lead to a premature change or reversal of UI design decisions, the implications of which will not be realized until later stages.
- The resulting misconceptions that lead to costly change requests and iterations, which torpedo budgets and timeframes, and endanger project goals.

In all, the lack of a common course of action and the use of inappropriate, incompatible terminologies and modelling languages (Zave & Jackson 1997) that prevent even the minimum levels of transparency, traceability and requirements-visualization that would be adequate for the problem, is a big challenge.

A picture is worth a thousand words

Summing up, the worst thing that any company can do is attempt to write a natural language specification for an interactive software system, especially when the UI is particularly significant (Horrocks 1999). When interactive behaviour has to be specified, a picture is worth a thousand words (Horrocks 1999). If the client is unable to assess both the look and feel of a system before forwarding the specification to a supplier, late-cycle changes will be an expensive necessity. Early prototypes are needed for evaluation with end-users in order to provide rapid feedback and to guide the overall specification process towards an ultimate design solution (D. Fitton et al. 2005).

UI development practice has to change

Facing demanding timeframes and critical budgets on the one hand and the importance of corporate information systems on the other hand, typical UI engineering processes have to change. Considering the value of corporate design and corporate identity, a company cannot restrict its responsibilities to the definition of user needs and functional requirements. Using abstract paper-based UI specification documents is a promising starting-point for every software project. But textual specifications usually fail to map real UI behaviour. They can only sketch out the look of a UI, and are unable to externalize the feel. Consequently, the assessment of user performance and user experience during actual interaction is postponed until later stages of design. This is too late if the UI behaves inappropriately. Clients therefore want to extend their duties in respect of UI prototyping and UI specification. The specification is forwarded to the supplier and a fundamental feature is iterative prototyping, done by the client himself. With regards to Chapter 1, this is defined as a prototyping-driven specification process and is desired by organizations that want to propel creativity and innovation.

Client and supplier responsibility in a prototyping-driven specification culture

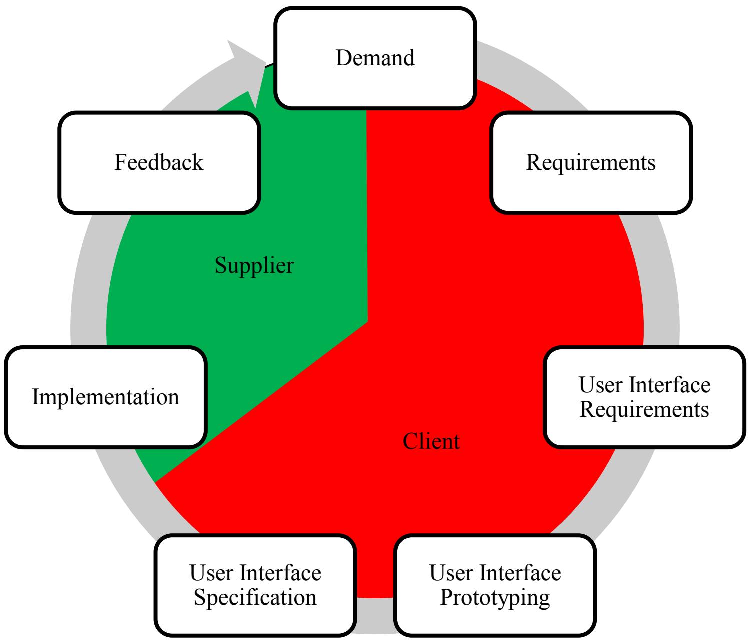

Demand
Requirements
User Interface Requirements
User Interface Prototyping
User Interface Specification
Implementation
Feedback
Supplier
Client

Figure 11: Assignment of client and supplier responsibility in the UI development process of organizations with a prototyping-driven specification culture (simplified lifecycle model)

Desired shift in UI specification practice

Because of the immaturity of their UI development processes, industrial clients decided to reassign responsibilities and began to change their UI specification practice, as outlined in the following:

- Due to the strategic impact of most software, clients want to increase their UI-related competency in order to reflect corporate values through high UI quality (Memmel et al. 2007g; Memmel et al. 2008e).
- Whereas conceptual modelling, prototyping or evaluation have always been undertaken by suppliers, the client himself now wants to work in the solution space and therefore needs to develop the UI specification in-house (Memmel et al. 2007a). This involves a fundamental change in the IT supply chain in favour

of a prototyping-driven UI specification and development process (see Chapter 1).

- As it is nearly impossible to specify a UI by means of Office-like applications, actors, who are accustomed to text-based artefacts, now require new approaches. These must take advantage of the commonalities of the contributing disciplines, and the task of learning the required modelling languages and understanding how to apply these new tools must not be an unreasonably difficult one.
- The role of the supplier becomes limited to programming the final system. The client can identify a timetable advantage from this change. For a company, acquiring its own know-how in important areas such as interaction design, prototyping and evaluation, therefore also means more flexibility in choosing or changing the IT supplier. Having an in-house competency in UI-related topics, the client becomes more independent and can avoid costly and time-consuming iterations with external suppliers.

Ultimately, the desired changes to current UI development practice depend upon the identification of shared methods, appropriate tools and the corresponding definition of an interdisciplinary UI development and specification process. This comes with the need for a change in the mindset of all stakeholders and a changing corporate-development culture.

2.7 Key Points

- Usability of the UI contributes to software product quality (see Chapter 2.1). The quality of the product, however, can only be as good as the specification that describes its properties. The methods and tools that support the specification and development for usability and user experience therefore have to be thoughtfully selected and applied. This challenge is a major part of this thesis and is discussed extensively in the following chapters.
- Usability is a success-critical economic factor. UI-related activities that tend to increase the overall quality of an interactive system must play a major role in development, for example in terms of effort and lines of code. Taking the UI into account as an important part of software development is justified by a high ROI. Companies that are able to design for enhanced usability of their software products gain positive customer feedback and can reduce costs (see Chapter 2.2). This thesis therefore proposes a process, method and tool for the specific purpose of specifying a UI in typical client and supplier relationships.
- UI specification is closely related to UI development, which is why an understanding of UI design approaches is essential in order to be able to develop new approaches to corporate UI specification.
- UE and SE contribute the most to UI development processes (see Chapter 2.4). But due to the participation of many other disciplines such as design, marketing and business administration, the engineering process must be based on a common course of action that also integrates BPM. Collaboration must begin from the early stages of design and continue along the whole IT supply chain.
- As UI quality is often a by-product of software development, structured approaches are necessary to guide UI development. To this end, a combination of the UE and technologist approach to UI development is a promising starting point for designing interactive systems. If a development team can depend upon adequate tool support that supports the combination of professions such as UE, SE and BPM, the outcome is likely to be superior to that of rather more isolated and non-interdisciplinary processes.
- In corporate UI development processes, the complexity introduced by the multidisciplinary context is tightened through the diversity of the client and supplier relationship. The separation of concerns comes with the assignment of different tasks that are either shared or completely encapsulated. This leads to predefined and critical transfer points that, in the main, shape a specification-driven UI development process. The mixture of formats and incompatible terminologies and means of communication must be replaced by a common course of action. Moreover, the client's competency in UI-related requirements analysis, UI modelling and UI specification must be extended. This raises a need for methods and tools that take into account stakeholders' familiarity with informal UI specification, but nevertheless bridge the disciplines, as well as the existing communication and responsibility gaps, with a common denominator for UI development and specification.

Chapter 3 Fundamentals of Corporate User Interface Specification

"Usability engineering adapts (the) general components of software engineering to provide an engineering-like process for the design and development of usable product user interfaces." (Mayhew 1999), p. 3

Searching for a common denominator

As outlined in the previous chapters, UI-related engineering processes normally involve many stakeholders. This demands approaches that help to bridge interdisciplinary gaps. The participation of non-technical personnel inevitably leads to the demand for a common course of action throughout the lifecycle. There is a need to have one common denominator of communication (see Figure 12). Otherwise stakeholders *"may think that they both agree on a design, only to discover down the line that they had very different detailed implementations and behaviors in mind" (Zetie 2005).*

The art of UI specification

Bridging the gaps between SE, IxD and BPM is necessary to be able to identify shared modelling languages and means of prototyping that can be understood by all stakeholders. The process of determining a shared approach provides grounds for examining the artefacts needed to drive the UI specification process.

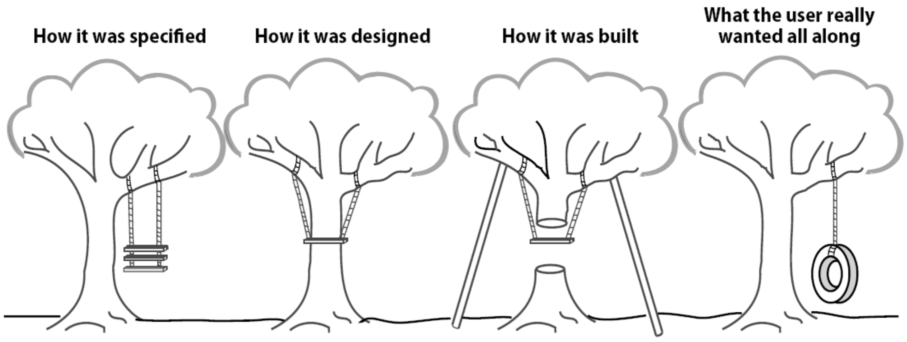


Figure 12 The understanding between stakeholders using paper-based UI specification (Zetie 2005)

Interaction designers often take a special role in interdisciplinary design. According to (Borchers 2001), interaction designers deliver the highest ROI of any members of a UI development team. When it comes to identifying problems in UI designs, IxD methods are up to three to four times more effective than SE techniques. Therefore, interaction designers often take a leading role in interdisciplinary design and the specification of UIs. However, their means of modelling have to be extended in respect of interoperable or supplementary artefacts of SE and BPM.

Chapter overview

In the following chapters, the art of UI specification will initially be discussed bearing in mind the interaction designer's point of view. This comes with a discussion of the different kinds of artefacts that contribute to the UI specification process, such as style guides or documents that preserve the design rationale (see Chapter 3.1). Taking into account typical specification methods of IxD and SE, informal and formal approaches to UI specification are compared and the best practice for corporate specification processes is inferred (see Chapter 3.2). With regards to the different kinds of artefacts that co-exist in the contributing disciplines, the role of patency and the role of models are discussed. Considering different levels for formability, this debate includes a comparison of model-based and model-driven approaches to UI specification. After all, models are the vehicles that make it possible to find ways to bridge IxD, SE and BPM, as presented in Chapter 3.3 using a semi-formal model-based approach. In Chapter 3.4 the idea of interactive UI specifications is inferred from the commonalities of the disciplines analyzed. Interactive UI specifications means the combination of models and designs in an interactive prototyping-driven specification format that is able to externalize look and feel, requirements, and design decisions at the level of detail required for the IT supplier that builds the system.

3.1 The Art of UI Specification

Confronted with a new design situation, a designer will often start working by developing initial design visions. These will be sketchy and diffuse in the beginning, and will be elaborated later when the understanding of the design space becomes more detailed. From that stage on, UI design is also substantially merged with the craftwork of user and task analysis, as most modern UI development and UE processes already propose (see Chapter 2). Concurrently with or subsequent to requirements analysis, early design thoughts will be translated into first externalizations of the vision, which can also be called the operative image (Löwgren & Stolterman 2004). This activity is the most important part of the design process, as it has the function of bridging the abstract vision and the detailed presentation of the design thoughts (see Figure 13).

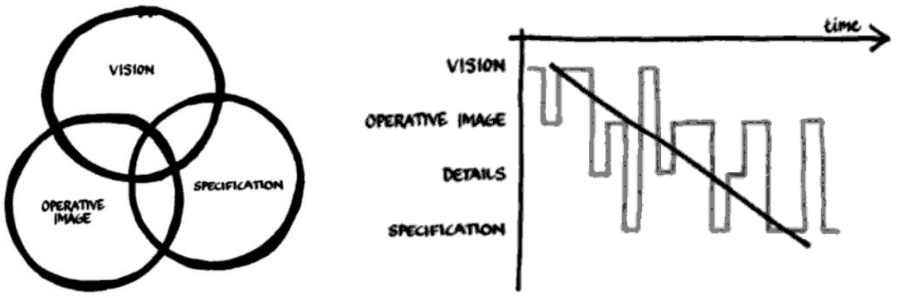


Figure 13: Interdependencies and movements in UI specification (Löwgren & Stolterman 2004)

UI specification activities are not recognized as independent practice

Switching between artefacts of a different grade of abstraction is among the most important tasks of designers, and one that is necessary in order to be able to narrow the design space towards the most promising solution. There will be recurrent jumping between the concrete and the abstract, as well as between the details and the whole. In UE development lifecycles, the operative image will undergo first evaluations in order to become good enough in terms of design, aesthetics, usability and various other dimensions of UI quality. The relevant people will then turn the operative image into a detailed representation and finally into the UI specification. After this point, the construction process begins and the UI must be implemented as specified. (Löwgren & Stolterman 2004) explain that designers usually find it very difficult to separate certain steps or phases of the design process. Major disruptions such as those caused by changes of responsibilities or actors, for example, could potentially harm the overall progress. All told, UI specification activities are not recognized as independent practice during UI development due to the fluid boundaries of the abstract, detailed and final design stages.

The art of UI specification in today's literature

In addition, what exactly makes up the UI specification is unfortunately not clearly defined in the UI design theory. While best practice for designing and evaluating interactive systems is well defined in various lifecycle models and international standards, the art of UI specification receives less attention and is poorly described in the literature. The reason is that most designers regard the transformation of requirements into visual externalizations with different fidelities as an ongoing and fully dynamic dialectical process (Löwgren & Stolterman 2004). The vision, operative image and UI specification are expected to influence each other continuously throughout the development lifecycle (see Figure 13).

Style guides

In Mayhew's UE lifecycle (see Figure 7) the detailed UI design evolves from a conceptual model (Level 1) and UI prototypes (Level 2). The transformation from the abstract to the concrete is taking place during iterative refinements coupled with UI evaluations. In order to make this translation process as transparent as possible, Mayhew suggests style guides as the kind of artefact that records design knowledge and rationale. After each design and development phase, the style guide is supplemented with information about the design decisions made and whether and how all requirements-analysis data is applied to the design. The style guide is also a very important instrument if design teams are physically dispersed. Team members can then refer to a shared knowledge base. The product style guide also documents screen-design standards to ensure the consistency of UIs across all parts of the interactive software system. And naturally, the information stored in the style guide is also a valuable source of design intelligence for later projects.

Style guides are important artefacts in designing interactive systems

Ultimately, one of the main purposes of the style guide is to communicate design concepts to the developers who are in charge of coding the UI. At this stage, the style guide incorporates information about the design rationale. This information is linked to the underlying requirements data (e.g. user profiles, task analysis, usability goals, work-style models, conceptual models, standards and guidelines, stakeholder and end-user feedback), and helps programmers to understand and follow the motives of designers. Style guides are among the most important artefacts in designing interactive systems. They preserve requirements data and design intelligence and function as a common knowledge base for UI designer and developers. They help developers to understand the motives of the designers.

Style guides are not an equivalent to UI Specifications

The detailed UI is usually generated based on the conceptual model design and the screen-design standards that have been documented in the product style guide. Developers will implement the UI directly from the style guide if the product to be designed is simple or everybody on the design team is very experienced with UE. But in most cases, the UI designer will additionally have to create a detailed UI specification and provide it to the programmers for coding purposes. Therefore a product style guide is not a detailed UI specification (Mayhew 1999). A detailed UI specification describes all actual and detailed display designs and the interactions with them. Only where aspects of the design cannot be effectively illustrated is it supposed to contain text. A style guide, in contrast, is a text-only document describing general standards and design standards to determine to some extent how these displays and interactions have to be finalised. Mayhew, however, does not provide detailed advice on the ingredients required for a complete and sound UI specification.

Learning from style guides

Although style guides differ from UI specifications in many ways, they do provide some important links. Because UI specification teams need to have a shared understanding about the system that is to be built, having a common data repository is fundamental. One of the main ideas of style guides, namely the documentation and safekeeping of design principles, guidelines, decisions and design rationale, is therefore equally important to UI specifications. Hence, a UI specification should also include information about the style. Very significant problems with style guides are that they can easily reach a size of hundreds of pages and can require hundreds of hours of effort for their creation. Bearing in mind new methods of prototyping-driven UI specification, styles guides should therefore also be transformed to the greatest extent possible into some more tangible or interactive form. Tool-supported approaches could use structured or hierarchical methods to organize design knowledge (Moran & Carroll 1996).

Design decisions and histories

Preserving information about the design process contributes to UI specification in different ways. Maintaining a comprehensive design rationale offers an opportunity to make the work of IxD more analytical and useful. Keeping track of many possible decisions and histories is an effective tool for evaluating trade-offs in future designs.

"A design rationale is a detailed description of the history and meaning of an artefact. For example, it can enumerate the design issues that were considered during a design process, along with the argument and evidence brought to bear on them." (Carroll 1991), p. 80

The claims approach for saving design rationale

Design rationale can be a language for stakeholders during UI design, but different stakeholders often speak different languages, are motivated by different values, and see different technical issues when looking at a design problem (Carroll 1997). The claims approach describes how UE can capture such design knowledge (Sutcliffe 2000). Claims are psychologically motivated design rationales. They express the pros and cons of a UI design as a usability issue and thereby encourage designers to consider design trade-offs rather than accepting a single design solution (Seffah et al. 2005a).

3.2 Approaches to Interactive Systems User Interface Specification

Communication between interaction designers, programmers and other stakeholders often happens through the medium of text. This is also the case with style guides, which are well-known artefacts of UI design and which are intended to document communication and decision-making. Specifications are different from style guides (see Chapter 3.1), but are also an established form of communication in engineering, manufacturing and business when it comes to agreement on specific requirements. Specifications can be described as the production of a precise description of an existing or intended software system (Dix 2003). The specification of a UI is not just a matter of defining the presentation. A relevant feature of a UI is also the description of its functional behaviour (Hussey 1996). Supplementing a UI specification with additional information can also add significant value.

Format and content of UI specifications

With regards to the multidisciplinarity of corporate UI-related processes and the variety of information that contributes to UI design, the challenge is to define a framework that provides well-defined borders for creating sound UI specifications. Research in formal UI specification methods has been a major concern of HCI (Dix 1987), but recent approaches tend to employ less formal methods, such as prototypes for specification (Dix et al. 2003; Jose 2003). The main reason for this is that formally written specifications have significant disadvantages in readability and traceability due to a lack of abstraction (Hussey 1996).

In interaction design, we have many notations to choose from, arising from the various disciplines that underpin our field. How quickly should we formalize our ideas in a structured notation, and for how long should we leave the ideas fluid and flexible? (Preece et al. 2002), p. 222

The scope of UI specification

The specification of an interactive system should not determine the algorithms and data structures used to build the system, as this is the proper domain of the programmer. But it should describe precisely the behaviour of the system, as the programmer may not be qualified to make such decisions, and the level of commitment at the time that the issue is uncovered may mean that the design choice has been determined by foregoing implementation choices (Dix 1995). After all, this closely reflects the current situation in client-supplier relationships as discussed in Chapter 2. Accordingly, the requirement of employing a certain degree of formality is consistent with the need to enable the client to make important design decisions and to specify the UI on his own. In the following, both the format and content of UI specification are therefore discussed in detail. This includes a debate on formal and informal specification and a discussion on model-based and model-driven UI development, as models are the artefacts of the early stages of UI design and are therefore the basis of the translation and UI specification processes.

3.2.1 Formal vs. Informal User Interface Specification

"Taken strongly, formalism in mathematics and computing is about being able to represent things in such a way that the representation can be analysed and manipulated without regard to the meaning." (Dix 2003), p. 1(431)

Notations in computer science

From the discussion on the disciplines that take part in corporate UI specification, we know that various different notations exist. In the context of requirements analysis and software development, notations help to analyze or specify the behaviour of a software product. In order to adequately support development, notations have to have some key characteristics (see Table 6).

Important properties of notations

A formal notation should be well focused upon the problem and should therefore support the description of UI design and dialogue structures. The notation should also take the background of stakeholders into account and make sure that comprehensibility and formalism are balanced. Especially with regards to iterative UI design, the notation should be easy to change. One of the most important aspects is the usefulness of the notation for the application domain and the appropriate level of detail or formality.

Table 6: Criteria that make formal methods function well; adapted from (Dix 2003) with a focus on UI specification

Criteria	**Description**
Useful	The notation is to address a real problem and should be focused on the UI dialogue structure
Appropriate	The notation provides just enough detail. The formalism must not deal with too much detail, as this makes it hard to see the things you really need
Communication	Pictorial representations and clear visualizations of flow are adequate means of communication with the client. Formal methods are often claimed to be a means to improve communication within a design team, because of their precision. However, when precision is achieved at the cost of comprehensibility there is no real communication
Complementary	The notation should follow a paradigm different to that of implementation. The notation must allow one to see the system from a different perspective, in this case one more suitable for producing and assessing the UI design
Fast payback	Reduce the lead-time to see the first running system. Spending a lot of time up-front is laudable, but results must be visualized early in the process. Dialogue flow charts do not produce long-term savings, but support early-stage production of running systems
Responsive	The notation must support a rapid turnaround of changes. Changes to the UI design of a system can occur frequently, as requirements become clear with every iteration cycle. In this process, a feeling of control and comprehension makes it easier to safely make changes
Reliability	Clear boilerplate code is less error-prone. Although the transformation process from diagram to code is not automated, it is a fairly automatic hand process applying and modifying boilerplate code templates. The heavy reuse of standard fragments greatly increases the reliability. This view of reliability through reuse can be extended in terms of taking software and HCI patterns (Gamma et al. 1995; Borchers 2001) into account
Quality	The clear labelling of diagrams makes it easy to track whether all paths through the UI dialogue have been tested. The visualization of requirements and dialogue flow therefore plays an important role with regards to the quality of a software and UI development process
Maintenance	It should be easy to relate bug and enhancement reports to the UI specification. If the screens presented to the user include information labels, it is easy to track bug reports or change requests

What is formal?

Formal notations used in software and UI development usually all try to remain abstract and be detached from the way the system is coded, but still be precise about some aspect of its behaviour. In contrast to an informal description, a formal description can say precisely whether or not a real system satisfies a description (Dix 1995). In all, formal notations can be categorized into two main groups (Dix 2003):

- **Finite process notations:** capture the sequences and processes of computation in terms of a finite number of stages, phases, states or steps. This includes many diagrammatic representations such as state-transition diagrams, flow charts and textual notations such as formal grammars and production rules.
- **Infinite state notations:** are somewhat mathematical notations using either a set and function notation, or a more algebraic representation.

Semi-formal graphical notations

With regards to the differentiation of formal notations, the question is whether a graphical notation can be formal. Diagrammatic formalisms (i.e. finite process notations) are often recognized as being less formal in terms of being less rigorous or more accessible. For many stakeholders, diagrams are more tangible and immediately appealing. Nevertheless, they can be just as rigorous as more obviously mathematical looking notations (Dix 2003). Hence, a diagram can be formal, informal or somewhere in between. Diagrammatic dialogue notations are not less formal because they are graphical. After all, if a dialogue notation has textual annotations that require informal interpretation (see Figure 14), it is defined as being semi-formal. However, the structure of the diagrams (i.e. shapes and connections) is still formal and capable of formal analysis.

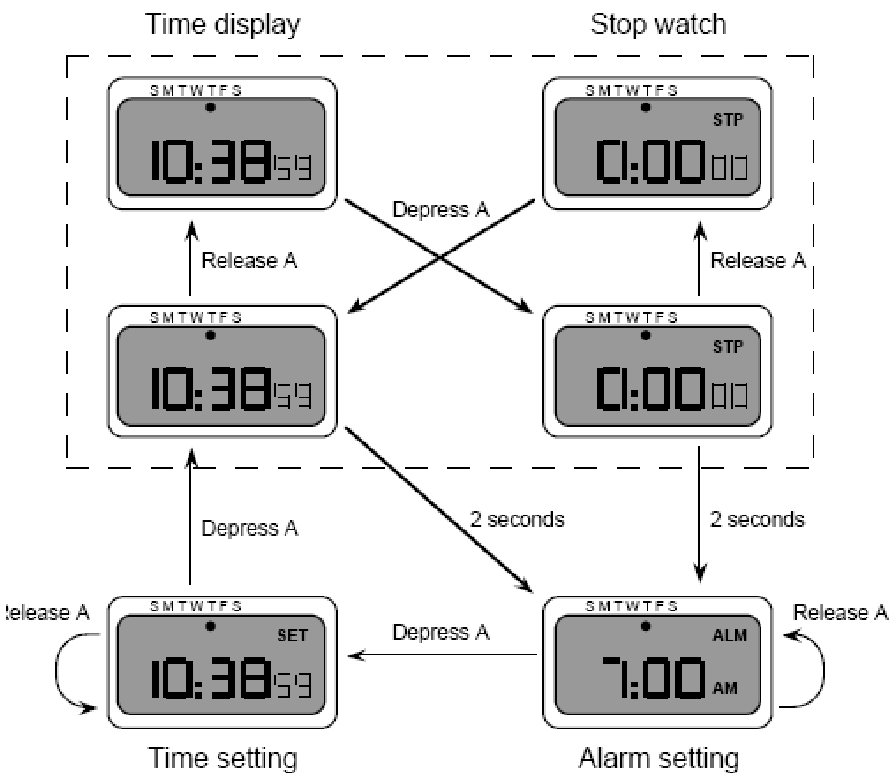


Figure 14: A state-transition network incorporating elements of UI design; from (Dix 1995)

Advantages and disadvantages of formality

The advantage that comes with the precision of formal notations is that they externalize design decisions that otherwise might not be noticed until the final system is being implemented and delivered. The disadvantage of formal methods is their decreased acceptance, which can be widely explained as being due to lack of understanding among non-IT personnel. In large and mature IT organizations (e.g. IT suppliers, service providers), some stakeholders may be able to work with formal methods (Dix 2003). This is, however, not true for non-IT organizations as we analyzed them in the context of this thesis, for example (see Chapter 2).

"Structured engineering approaches that are focused on systems building and favour use of formal representations, since they support the analysis and transition to implementation that is needed for making effective, efficient and reliable software. Approaches that rely on participation of people unskilled in formal methods, naturally favour informal representations, as they enhance the dialogue with end-user and support validation. Unless political arguments are used, it should be possible to combine formal and informal representation, to get the benefits of both. In the end, the system description will have to be executable, so a formal representation must eventually be derived from the informal ones. Another important argument for formal representations is that of scale; to bridge the gap between stakeholders at higher organizational levels, some formalization is desirable to ensure proper understanding and agreement." (Traetteberg 2002), p. 15

Advantages and disadvantages of informality

The advantages of informal representations include their visual and concrete nature. The roughness of many informal representations seems to be considered an advantage. (Gross & Yi-Luen Do 1996) identify ambiguity/vagueness, abstraction, precision and uncertainty/commitment as important characteristics of design representations. Formal models, in turn, focus more on being concise, complete and final representations. Moreover, there are few possibilities for expressing informal characteristics in formal diagrams. (Jones & Sapsford 1998) accordingly note that informal representations have the advantage that they do not impose any conceptual constraints on the drawings. However, to ensure that agreement is really based on common understanding, it is necessary that more formal representations be used during later stages, when it is important to be able to share the design with other stakeholders. For example, the AUTOMOTIVE RE process shown in Figure 15 presents the process of refinement, starting with informal and vague requirements but ending with a consistent specification that depends on a successful translation of (usually) text-based description into more expressive kinds of notations.

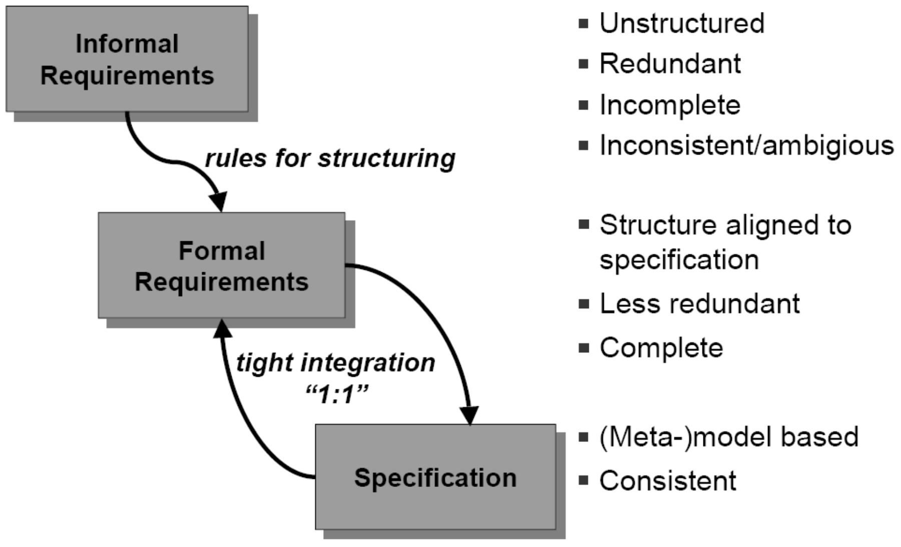


Figure 15: The AUTOMOTIVE requirements-engineering process (von der Beeck et al. 2002)

Pragmatic semi-formal dialogue notations

This suggests that formal notations should be packaged so that non-IT experts are able to understand and apply them without having a steep learning curve (i.e. a high threshold). One way this can be achieved is through notations that have formal underpinnings, but that are applied in a more pragmatic and approximate way. Semi-formal dialogue notations (see Figure 14), which have a graphical form, therefore tend to be more interdisciplinary artefacts. Dialogue specification (see Table 7) is the least mathematical kind of formal notation and therefore the one most easily used by the non-formalist stakeholder.

Table 7: Uses of formal methods; based on (Dix 1995)

Use of formal method	**Description**
Specification of individual interactive systems	Concentrates on a specific system and the complete specification of all aspects of its behaviour. Its purpose is to clarify design decisions, to expose inconsistency and to act as a contract with the implementer. UIs can be extremely complex and being able to deal with them at a more abstract level is thus even more important than it is for general software.
Generic models of interactive systems	Their purpose is to give new insight into general problems as the properties of the problem-domain are analyzed. They can be used as part of a formal development process to constrain the design of specific systems. In other words, results from the analysis of generic models can be applied to the formal specifications of specific systems.
Dialogue specification and analysis	Dialogue notations are used to describe specific systems, but at a different level of detail than a full formal specification. They are concerned with the steps of the user interaction but typically do not fully specify the meaning attached to the user actions.

Formal methods and HCI

(Dix 2003) identifies a huge growth in the use of diagrammatic UI specifications, especially in the use of formal UML. Even in HCI, the employment of formal methods has gradually grown and is now a topic of various established conferences (e.g. DSV-IS, CADUI, EHCI, TAMODIA). Here, UI specification has become almost synonymous with formal methods. One of the main uses of formal methods in IxD is therefore (1) to specify the behaviour of specific systems, (2) to assess potential usability measures or problems, and (3) to describe the system in sufficient detail so that the implemented system matches the requirements. As a side effect, the UI specification process forces the designer to think about the system clearly and to consider issues that would otherwise be missed (Dix 2003). After all, writing down ideas and requirements in a structured fashion helps to visualize the big picture and to externalize missing information. The notation that is used here should not influence the number or nature of the ideas that the interaction designer generates.

Correspondence between different kinds of models

To achieve more consistency between different kinds of models and notations, there must be a close correspondence of structure between the formal and the informal concepts. Because stakeholders cannot all understand the requirements properly the first time, (Dix 2003) proposes a prototyping-based UI development process. But prototyping alone is fundamentally flawed unless it is guided by an analytical and theoretical framework. Here, different kinds of notations can help to understand requirements and support the translation of requirements into UI design. A road map for UI specification of interactive systems should therefore take into account both of the different kinds of notations, i.e. UI-related models, and UI prototypes.

Prototyping-driven UI specification

Any form of specification involves some formal parts, about which it makes precise statements, and some informal parts, mostly in the form of textual comments. In order to overcome the limitations of textual UI specifications, as discussed in Chapters 1 and 2, text must be supplemented by interactive UI prototypes. The prototypes then function as living descriptions of the look and feel of the UI and help to understand and validate requirements. The prototyping process therefore is propelled and guided by informal, semi-formal and/or formal methods and drives the overall UI specification process. The assembly of different kinds of artefacts, i.e. models and prototypes, into a more expressive kind of prototyping-driven UI specification (see Chapter 1) reflects the common practice in today's UI development processes. Material from all phases of development that impact on the development of the final system must be forwarded to the developers (see Chapter 3.4).

A road map for UI specification

Many argue that formal methods are too difficult, cannot be scaled and require too much training (Dix 2003). Particularly during the early stages of interaction design, 'back-of-an-envelope' notations are more widely used to capture initial, sketchy ideas (Preece et al. 2002). The gap between informal and formal methods therefore especially exists in the minds of designers and clients. What is still missing, however, is a framework that integrates both formal and (semi-)formal approaches and determines the stage from which more formal notations should be used. Such a framework could provide adequate notations for different target groups in order to ensure understandability. For communicating with developers, more formal and technical notations are the much better choice. Otherwise, more informal notations are preferable.

Choosing the medium for the message can affect how the message is received and hence the meaning that is communicated, so it's important to get the medium right. (Preece et al. 2002), p. 222

The right kinds of models

If different kinds of notations are to be integrated into a UI specification, it is necessary to determine their power. This refers to the graphical representation as well as to the ability to deduce from models information that is usually hidden due to its level of abstraction. More precisely, the question is whether models should allow the automatic generation of related material, such as the UI code, or whether models should rather function as expressive RE information items that support a handcrafted generation of UI design. This leads to a discussion of model-based and model-driven approaches to UI specification, a topic closely related to the discussion of formality.

3.2.2 Model-Driven and Model-Based User Interface Specification

Models – a definition

In order to handle the complexity of many software projects, models have a long history in SE and IxD, and model-based approaches to development are today common in both disciplines (see Chapter 2). (Bock 2007b) defined models with regards to the multidisciplinarity of IxD processes based on the general theory of models (Stachowiak 1973). Accordingly, a model is described as

- A representation of a natural or artificial original.
- An image that does not have all the characteristics of the original it represents, but only those properties that are relevant to its creator or user.
- An abstraction that is not automatically associated with an original, but is mapped to it by its creator or user at a certain time.

A model is therefore a less detailed, rather incomplete image of reality. The purpose of carrying less detail is abstraction, which in turn tends to reduce the complexity of the original presentation (Oberquelle 1984). Modelling accordingly describes the development of models and their association with real objects.

Advantages of modelling

The most obvious advantage of modelling is that the person working with the model is likely to be able to identify key characteristics of the represented object more easily, because other information is simply hidden. In designing interactive systems, models therefore help to structure problems (Constantine & Lockwood 1999b) and in understanding a problem space more quickly. Particularly when many different stakeholders work on a problem, which is typical in most UI specification processes, modelling helps to build a common knowledge base.

Disadvantages and challenges of modelling

However, the development of models must be grounded in the basic concepts and processes of an IT organization. If models are built simply for the purpose of documentation, it is likely that developers will not like them. Just as there is a need to have a healthy balance between requirements analysis and the externalization of design vision, it is also important to balance modelling and design. If stakeholders are unable to extract any added value from modelling, then the purpose of modelling has been completely misunderstood. It is therefore important to keep track of the advantages to both project and individuals from modelling. Most probably, such advantages are an increased overview of the problem at hand, a more structured view of the requirements and a more transparent connectivity of requirements and design. If the models can be electronically stored - which is usually the case with most computer-generated models - they can then be preserved for future projects. Saving design artefacts can, in turn, increase the efficiency of a process (Bock 2007b).

Modelling with UML

Although the UML is an established industry standard, its broad acceptance in the IxD community has been hindered. Despite UML profiles and many attempts at improvements (Blankenhorn & Jeckle 2004) the UML is still particularly unsuitable for modelling UIs (Silva & Paton 2000). UML is visually too awkward, as it cannot (visually) express the look and feel of a UI. Apart from software engineers, other stakeholders usually cannot understand UML. Moreover, even system developers find the diagrams too uninspiring, and notations and editors too restrictive, all of which impedes rather than helps their work (Jarzabek & Huang 1998).

Models in UE and SE

Consequently, in computer science there is a growing special research agenda in modelling and constructing UIs, for example at the CADUI conference (Memmel & Reiterer 2008; Sousa et al. 2008b). A significant area within this field has been model-based and model-driven UI development and specification. The idea behind both approaches is closely related to the application of (semi-)formal methods. Models usually allow important UI properties to be deduced from them. This can be done in either a manual, i.e. intellectual and handcrafted, process or a (semi-)automatic, computer-supported way. Throughout the SE and IxD community, the nomenclature of both approaches is not clearly defined and it is therefore important to use a well-determined differentiation in this thesis.

Model-driven and model-based approaches to UI specification

With regards to UI specification, the term model-driven UI specification is used in this thesis to describe specification processes that use models to generate the specification of the final UI in a more or less automatic process (see Table 8). The reason is the relatedness to the nomenclature of model-driven architecture (MDA). The idea of MDA is the use of semantically expressive and machine-readable models that can be used to deduce software architectures and code on different levels of abstraction. The synchronization of models and code is, above all, a regular source of serious problems in model-based development processes. MDA aims at overcoming the weaknesses of model-based software development by taking code as a by-product. Otherwise, and differing from (Dix 2003), we reserve the term model-based UI specification to describe processes in which models play an important role, but in which UI design is still an intellectual process driven by a thoughtful interaction designer (see Table 8). The main difference between model-based and model-driven UI specification is the role of the models as artefacts and the corresponding formality of the notation (see Figure 16).

Table 8: Comparison of model-based and model-driven development and specification

Model-based development	**Model-driven development**
Relies upon informal and semi-formal notations	Mostly employing strict and formal notations
UI specification is based on the development of different models, but models do not carry all the information required to develop the design	UI specification is driven by the development of models, and the models contain detailed information that allows the automatic deduction of the design
Models support the understanding of requirements and decision-making of developers	Generate the UI design automatically from models
Interface design includes the relationship of widgets and requirement models. Clear connections enhance traceability and transparency, but UI specification development stays a handcraft.	Interface design and concrete widgets, for example an executable UI specification, are automatically deduced from formal notations

Model-driven UI specification

In a model-driven approach, models have a much more central meaning. They are the fundamental elements of the actual product development, or more precisely, the coding of the UI. Because models are abstract and formal (usually to a much higher extent than in model-based approaches), they focus on important UI properties that can be transformed into code by the push of a button. The interaction designer creates the models using a notation or design environment and then a software tool automatically generates the actual UI. Such approaches proved to be particularly powerful, for example when targeting multiple devices (Vanderdonckt 2005). Model-driven UI generation is based on the idea of deducing system properties, code and other artefacts directly from (abstract) formal models (see Figure 16). Accordingly, models are essential items in the development process. As described previously, abstractness in this context means a focus on the main characteristics of the software that are essential, providing the required features and a high UI quality, for example. After all, the models *are* the code, giving them an equal importance. Naturally, working with such models comes with the limitation that much information has to be integrated into the models in advance. (Bock 2007b) has summarized the main contributions of model-driven approaches as the following:

- **Reduction of development time:** as executable code can be generated directly from models.
- **Higher development quality:** automatic transformation and formalism support standardization and tend to be less error-prone, because manual changes are required less frequently.
- **Better maintainability:** changing models will change the code.
- **Increased Reusability**: once defined, models and transformation rules (which translate models into code) can be reused as often as necessary. And reusability also provides the opportunity to set up new projects on existing knowledge.

- **Reduction of complexity:** abstraction decreases the amount of information that has to be processed in order to understand a problem at hand. Modelling languages therefore lower the threshold for getting the UI specification right.
- **More interoperability and portability:** Through a separation of specification and functionality as well as the associated transformation rules, a UI design can easily be duplicated for a variety of platforms or screen sizes.

The most significant advantages come from different levels of abstraction. Many problems have to be solved only once at a high level of abstraction and not – as before – once in the implementation level and a second time for documentation. Model-driven specifications can therefore be established as the backbone of model-driven development processes, with significant potential for clients and their collaboration with suppliers. If formal, electronic – and thus machine readable – specifications can be exchanged between all stakeholders, the specification problem as well as the communication problem in traditional development processes (Rauterberg et al. 1995) can be overcome.

Model-based UI specification

In a model-based approach, the artefacts especially serve as a structured way of documentation. In this context (Traetteberg 2002) uses models for capturing knowledge, ideas and suggested solutions for a UI design. Moreover, sound models help to identify problems and shortcomings. Because of the clear connection between representation and dialogue design, and the user, task and behaviour models (see Chapter 2), model-based UI specifications enable the interaction designer to *"make more informed decisions and gain better insight into the effect each interface element has on the overall task"* (Puerta 1997), p. 46. After all, model-based approaches focus on providing support for analysis and reasoning. In this context, the handcrafted mappings from abstract requirements to abstract UI and to more concrete dialogue design are the main steps in a model-based and iterative design process (van der Veer & van Welie 2000). In addition to task and dialogue models, these three steps rely on additional supporting models that capture knowledge of both the problem and solution.

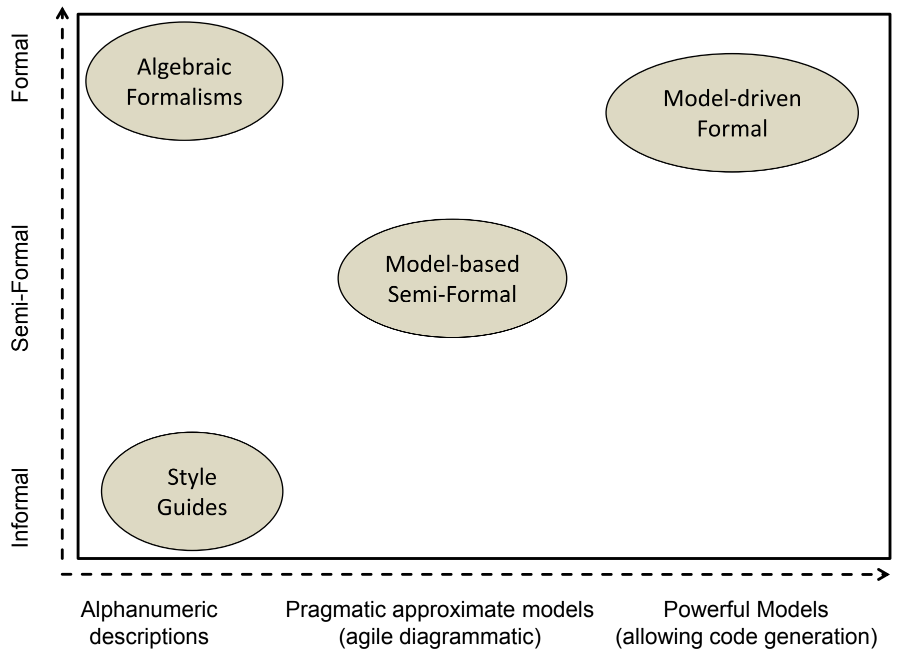
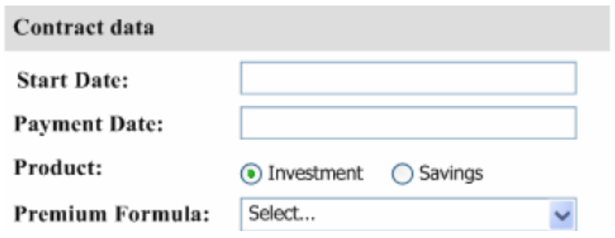

Formal
Semi-Formal
Informal
Algebraic Formalisms
Model-driven Formal
Model-based Semi-Formal
Style Guides
Alphanumeric descriptions
Pragmatic approximate models (agile diagrammatic)
Powerful Models (allowing code generation)

Figure 16: Classification of model-based and model-driven UI specification

Comparing model-based and model-driven UI specification

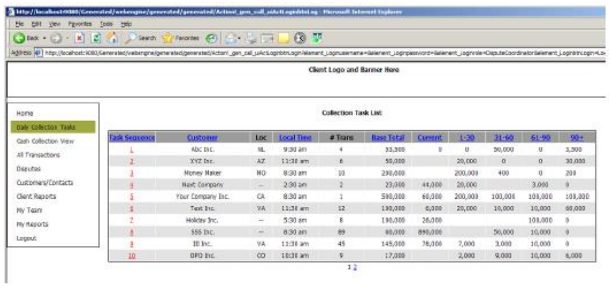
Design involves thinking about the problem and solutions domain, tasks and information, dialogue and interaction. Designers and practitioners tend to prefer model-based approaches since humans do not actually think in terms of abstractions and formalisms. In addition, modelling languages and tools that have less focus on formalism can potentially be better integrated with corporate industrial methods for systems engineering and development, because non-IT stakeholders work less formally in their everyday business. Formal representations are considered hard to read and write, and support only those educated and trained in their use (Traetteberg 2002). Hence, *"concrete and informal representations like sketches and prototypes should be used and not abstract and formal diagrams or models."* (Traetteberg 2002), p. 14. As described previously, semi-formal diagrammatic notations can be the working compromise between formal and informal notations. A semi-formal and model-based approach has the advantage of taking into account the work style of stakeholders on the one hand, and the advantages of approximate and incomplete design artefacts that drive the UI specification process on the other (see Figure 16). This makes them a best practice for corporate UI specification processes.



Contract data

Start Date:	
Payment Date:	
Product:	[✓ Investment] Savings
Premium Formula:	Select...

(Sousa et al. 2008c)



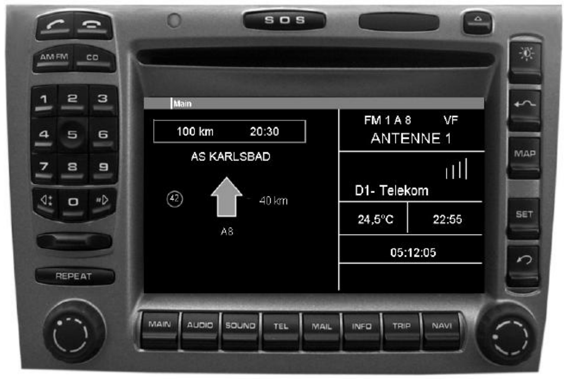
Client Logo and Banner Here

Collection Task List

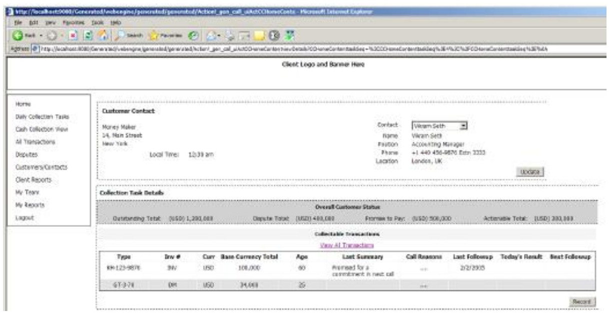
Task Sequence	Customer	Loc	Local Time	# Trans	Base Total	Current	1-30	31-60	61-90	90+
1	Abc Inc.	NL	9:30 am	4	33,300	0	0	30,000	0	3,300
2	XYZ Inc.	AZ	11:38 am	6	50,000		20,000	0	0	30,000
3	Money Maker	MO	8:30 am	10	200,000		200,000	400	0	200
4	Next Company	—	2:30 am	2	23,000	44,000	20,000		3,000	0
5	Your Company Inc.	CA	8:30 am	1	500,000	60,000	200,000	100,000	100,000	100,000
6	Test Inc.	VA	11:38 am	12	100,000	6,000	20,000	10,000	10,000	60,000
7	Holiday Inc.	—	5:30 am	8	100,000	26,000			100,000	0
8	SSS Inc.	—	8:30 am	89	90,000	890,000		50,000	10,000	0
9	III Inc.	VA	11:38 am	45	145,000	78,000	7,000	3,000	10,000	0
10	DPD Inc.	CO	10:38 am	9	17,000		2,000	9,000	10,000	6,000

1 2

(Sukaviriya et al. 2007)



(Bock 2007b)



Client Logo and Banner Here

Customer Contact

Money Maker
14, Main Street
New York
Local Time: 12:39 am

Contact	Vikram Seth
Name	Vikram Seth
Position	Accounting Manager
Phone	+1 440 456-9876 Extn 3333
Location	London, UK

Collection Task Details

Overall Customer Status

Outstanding Total: (USD) 1,200,000 Dispute Total: (USD) 400,000 Promise to Pay: (USD) 500,000 Actionable Total: (USD) 300,000

Collectable Transactions

View All Transactions

Type	Inv #	Curr	Base Currency Total	Age	Last Summary	Call Reasons	Last Followup	Today's Result	Next Followup
KH-123-9876	INV	USD	100,000	60	Promised for a commitment in next call	...	2/2/2005		
GT-3-78	DM	USD	34,000	25		...			

(Sukaviriya et al. 2007)

Figure 17: Overview of examples of UIs generated with model-driven approaches

Application domains

Moreover, it is unrealistic to think that sophisticated UIs can be completely generated by automatic means, without any need for handcrafted reworking and refinement (Bock 2007b). In other words, the more unusual and less straightforward the appearance of a UI has to be, the less likely it is that a model-driven UI specification will be capable of automatically generating the design from abstract models. This inevitably leads to a necessary differentiation between model-based and model-driven UI specification approaches in respect of the kind of UIs that can be developed (see Table 9).

Table 9: The differentiation between model-driven and model-based UI specification with regards to the UI that is to be built

Interfaces that should be specified with a model-driven approach	**Interfaces that should be specified with a model-based approach**
Focus on safety and technical-constraints	Focus on innovation in terms of design and aesthetics
Standard UI layouts, elements and widgets	Huge variety of UI design elements, non-standard widgets and presentations
	Innovative UIs with a focus on both usability and user experience

With regards to the application of formal notations it is important to recognize that the strong formal-methods community places a greater emphasis on particular domains where safety, cost or complexity make the effort of formal methods worthwhile. Very formal methods and notations of UI specification can therefore be assigned to software projects where, for example, safety plays a more important role than aesthetics or joy of use (see Table 9). For UIs that are constrained in respect to specific hardware that they are associated with, formal methods will also work very well. For example, (Memmel et al. 2007a) were able to successfully use a model-driven approach to specify in-car information systems. However, the UIs generated with model-driven approaches tend to be more straightforward in terms of their UI layout and the elements and widgets used (see Figure 17).

Towards semi-formal model-based UI specification

Because of its greater pragmatism, semi-formal modelling offers promising beginnings for a unification of engineering disciplines. A model-based and less formal approach is consistent with agile principles and practice, and, agility is closer to the work style that stakeholders in corporate UI specification processes are familiar with. Hence, a semi-formal model-based UI specification approach offers a better opportunity for bringing RE, SE and UE closer together. This is a convergence that is badly needed for coping with the technical and organizational complexity of interdisciplinary and networked development processes for corporate software systems. With a semi-formal model-based approach to UI specification, UI-related models can become more accessible to developers untrained in formal modelling.

3.3. Bridging the Gaps for Model-Based Semi-Formal Specification

The models applied in a semi-formal model-based UI specification approach should be based on few and simple basic concepts and constructs, to make them easier to read and write. The languages should not force the modeller to be more specific and formal than the development process requires. The languages should also not force the modeller to include details before they are actually needed. They should support a gradual transition from general to more specific and formal statements. And the visual notation should be flexible (Traetteberg 2002), but nevertheless allow for integration with existing languages used in the disciplines of SE, IxD and BPM. With regards to all disciplines, it is necessary to identify a common ground for modelling that will help to bridge the disciplines and define an appropriate set of shared modelling and design artefacts (see Chapter 4).

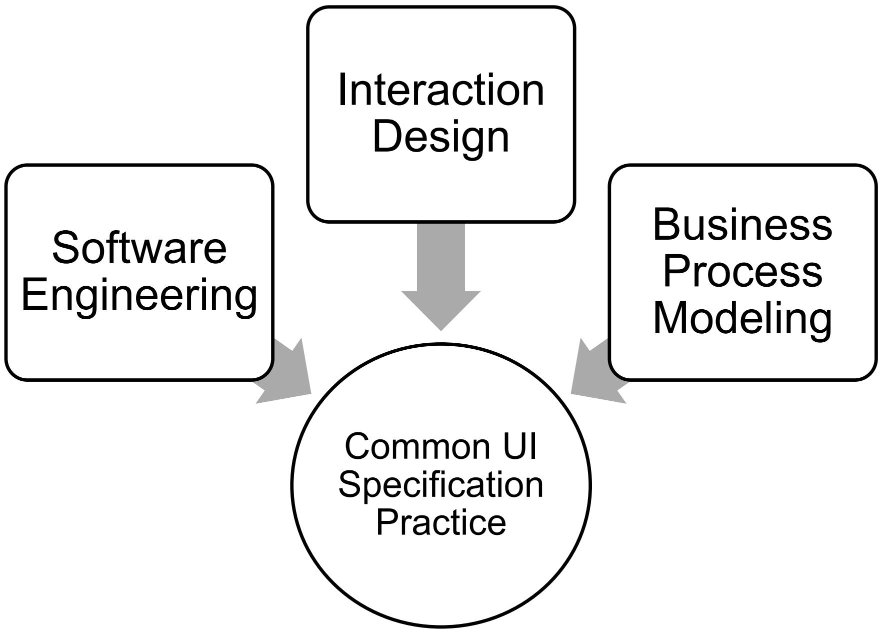


Figure 18: Searching for a UI specification practice common to SE, IxD and BPM

3.3.1 Integrating Interaction Design and Software Engineering

Differences of disciplines

As discussed in Chapter 2, software engineers are generally trained in topics such as system architecture or database design, while interaction designers are concerned with ease of use, ease of learning, user performance, user satisfaction and aesthetics, for example. An interaction designer normally has a black-box view of the back-end system, while the software engineer has a deeper understanding of the architecture and code behind the UI (Campos 2004). SE considers how functional requirements are translated into a running system. Although both disciplines have reached a certain degree of maturity, they are still practiced very independently (Pyla et al. 2003). Consequently, software developers and interaction designers, end-users and business personnel express themselves in quite different fashions, ranging from informal documents (e.g. scenarios) to formal models (e.g. UML).

Integrating interaction design and software engineering

IxD and SE are recognized as professions made up of very distinct populations. Each skill set is essential for the production of quality products, but no one set is sufficient on its own (Buxton 2003). But as well as the implementation of pure functionality, corporate software products demand the integration of usability and design consideration into the development process. The SE community often considers usability to be primarily a property of the presentation of information, i.e. of the UI (Hix & Hartson 1993). A result of this assumption is that UI design is often postponed until the later stages of development. (Folmer & Bosch 2004) identify two risks with this approach:

- Assumptions may be built into the design of the architecture that may unwittingly affect the UI design.
- Assumptions may be built into the UI that are not supported by the architecture, for example interaction issues (such as support for a wizard or an undo), information architecture issues (such as a separation of data from presentation), and interface issues (such as visual consistency).

Software architecture and the UI

Software quality attributes such as performance or reliability are to a considerable extent defined by the software architecture. The software architecture also has a major impact on the interaction and information architecture. Designing a usable system is therefore more than simply ensuring a usable UI. On the one hand, a slow and buggy system architecture with a good UI is not considered usable. On the other hand, the most reliable and well-performing system architecture is not usable if the user cannot understand how to use the system. Designing a well-performing, reliable and flexible architecture that can support unforeseen usability requirements is quite a challenge. Software architecture is an important instrument for reaching high UI quality. In raising the awareness of this relationship, software engineers and interaction designers must recognize the need for a closer integration of practices and techniques (Folmer & Bosch 2005).

Interdependent parts of a whole

However, the behaviour of the system and the feel of the UI are very much dependent on each other. The more important the UI component becomes for a software application, the more significant is its impact on the back-end system. Hence, SE and IxD need to overlap at the interaction layer in order to develop usable systems. If the collaboration at the interaction layer is well defined and working successfully, the time-to-market can be dramatically shortened by (having) well defined interaction points at which the teams can re-sync and communicate (Gunaratne et al. 2004). A better balancing of the different views of the system, and hence better usability, is achieved when software is designed in multi disciplinary teams (Folmer & Bosch 2005). The specification methods and process employed have to take this into account accordingly.

Integration issues

As there is a need for both professions to talk the same language, both disciplines need to agree on how to describe components of the UI. Several conference workshops highlighted major integration issues, which are summarized in Table 10. For each integration issue indicated by (Seffah et al. 2005b), we suggest methods of application in a cross-discipline engineering process.

Table 10: Integration issues for SE and UE, partly based on (Seffah et al. 2005b)

Integration issue (Seffah et al. 2005b)	**Method of application**
Mediate and improve the communication lines between users, usability experts and developers	Use medium-weight artefacts, work with toolkits appropriate for collaborative design with all actors
Extend artefacts for UI specification and conceptualization	Use models known by both professions and adjust their expressiveness
Enhance object-oriented SE notations and models	Use interoperable notations understood by all actors
Extend RE methods for collecting information about users and usability	Include principles, practice and adequate methods from HCI into RE up-front.
Develop new processes for interactive systems design	Reconsider methods, keeping in mind shorter time-to-market and business demands
Represent design artefacts including prototypes using different formalisms	Enable all actors to model and test the UI without coding and too much need for formalism
Convey user-centred design attitudes, not just tools and methods, to support user-centred design activities	Integrate tools that support all professions by transporting patterns, design knowledge, design principles

Interdisciplinary modelling languages

Very formal or complex models are an inappropriate base for communication, especially so for collaborative design processes with high user- and business-stakeholder participation. As an interdisciplinary modelling language for requirements gathering, research suggests scenarios (Jarke 1999; Rosson & Carroll 2002; Seffah et al. 2005a; Sutcliffe 2005) - known as user stories (light-weight scenarios) in agile development (Beck 1999) - and prototypes (Sutcliffe 2005) as bridging techniques for IxD, and SE. Choosing the right models in an interdisciplinary context is crucial to avoid misunderstandings (Constantine et al. 2003).

Scenarios as bridging technique

In SE, scenarios - as a sequence of events triggered by the user - are generally used for requirements gathering and for model checking. Such a use-case scenario is used to identify a thread of usage for the system to be constructed and provide a description of how the system will be used (Pressman 1992). In contrast, IxD applies scenarios to describe software context, users, user roles, tasks and interaction (Rosson & Carroll 2002). In all, scenarios can be used as a vision of the system by explaining functionality and interactive behaviour, as well as being a description of users and their tasks, as outlined in Chapter 4. Scenarios can support the design process and are well established in both disciplines. They can therefore form the common ground between the disciplines (Sutcliffe 2005).

Prototypes as bridging technique

Prototypes are also applied by both disciplines, although their role in SE was less important until agile approaches refocused attention on them as vehicles for inspections and testing. Prototypes in SE can also be used to verify functional specifications and models as well as for understanding problems by doing user inspections. Agile Modelling (AM) already recognizes prototypes as a type of small release, which can be continuously changed and fine-tuned (Blomkvist 2005). In SE, however, prototypes are often used to assess whether requirements are adequately addressed and satisfied. With regards to UI design, this can be especially problematic. Just because a design idea satisfies a requirement does not necessarily mean that it does it the best way possible (Lim et al. 2008). There might be design alternatives that do so, but they might be missed if the team decides on a preliminarily design just based on very strict rules of fulfilment. IxD mainly recognizes them as an artefact for iterative UI design and distinguishes different dimensions of prototypes (Rudd et al. 1996). Here, prototypes are used to explore design spaces, including all possible design alternatives and the associated trade-offs. In spite of these differences, prototypes are clearly one of the most compatible links between SE and UE.

Bridging the gap with agile development

Just as with prototyping and scenarios as common vehicles, IxD and (agile) SE can converge on other shared principles and practices by using more methods familiar to both professions and by speaking the same language. Faced with the demands of corporate project environments, a reliance on heavyweight methods such as formal models (SE), or massive documentation such as style guides (IxD) is far too expensive for the design of interactive products. After all, greater cost-effectiveness and flexibility were among the main arguments for lighter, agile methods. The bottom line is that some informal methods of agile development are close to IxD practice and are therefore the pathfinder for a common course of action.

The role of agile modelling

We therefore believe that cross-discipline agile methods are the optimum, and workable, compromise. As agile approaches already exist in both SE (Beck 1999; Cockburn 2001) and IxD (Constantine 1996; Constantine & Lockwood 1999a; Constantine & Lockwood 1999b; Constantine et al. 2003; Constantine & Lockwood 2003; Gundelsweiler et al. 2004; Memmel et al. 2007e; Memmel et al. 2007d; Memmel et al. 2007f; Memmel et al. 2008a), they can be the interface for a common and balanced software lifecycle (Blomkvist 2005). Prototypes can include and convey the attitudes of both IxD and SE. Agile models can usually be generated and changed quickly, but still be sufficiently detailed to act as a medium of communication (Blomkvist 2005) with stakeholders. They can preserve design experience in a visual language understood by everybody. On the other hand, scenarios can support the design process as they are used to test assumptions and stimulate creation. They can form the common ground between the disciplines and need to be integrated throughout the life cycle (Rosson & Carroll 2002; Sutcliffe 2005).

Cross-discipline development

In contrast to classic, heavyweight SE processes such as the V-Model, agile methods begin coding at a very early stage while having a shorter up-front RE phase and less documentation. Following the paradigm of XP, implementation of code takes place in small increments and iterations, and the customer is supplied with small releases after each development cycle. During a short requirement analysis known as the exploration phase, the development team writes user stories in an attempt to describe user needs and roles; the people interviewed need not necessarily be the real users of the eventual software product. (Gundelsweiler et al. 2004; Memmel et al. 2007e) explained that XP therefore often fails to collect real user data and starts coding based only on assumptions about user needs. Agile methods are less rigid than XP and take more care over initial requirements-gathering as they provide room for low-fidelity (essential) prototyping, activity diagrams or use-case diagrams (Cockburn 2001; Ambler & Jeffries 2002). Nevertheless, the analysis phase is finished as soon as requirements have been declared on a horizontal level, because the iterative process assumes that missing information will be filled in at later stages. Development in small increments may work properly as long as the software is not focused on the UI. When designing UIs, continual changes to the UI due to fast iterative design may give rise to conflicts with user expectations and learnability, cause inconsistency and finally lead to user dissatisfaction (Constantine 2004). Evaluation of small releases with stakeholder participation does not ensure that the whole system provides a consistent conceptual, navigational or content model (Gundelsweiler et al. 2004).

Considering the weight of IxD methods

The discussions about agile approaches to UI design led to a movement in the IxD community, which began to reconsider its user-centred heavyweight lifecycles (Constantine & Lockwood 1999b; Gellner & Forbrig 2003; Holzinger & Slany 2006; Memmel et al. 2007e). Both SE and IxD have to cope with a shorter time-to-market, in which the quality of the delivered software must not suffer. The continuous shortening of development lifecycles is therefore a great challenge both for management and the methods and tools applied. Many experts, who are likely to have a home in both SE and IxD, consequently faced up to the issue by developing approaches to light-weight or so called agile human-computer interaction, e.g. eXtreme usability (Holzinger & Slany 2006).

Balancing principles and practices of IxD and AM

What is required is a balanced hybrid process, which must be consistent with the principles and practices of both IxD and SE (Blomkvist 2005). In order to identify interfaces between agile SE and IxD, different approaches to UI design have to be discussed in respect of their agile potential and their different contributions to a cross-discipline process (see Table 11). The chosen UI specification approach must, however, be able to produce the quality of the UI and the software product as a whole as required (see Chapter 2).

UI design approaches

Like XP, original user-centred design is a highly iterative process. It differs from agile methods, however, since real users are taken into account and the development team tries to understand user needs and tasks before any line of code is written. The lifecycles of UE processes such as Scenario-Based UE (Rosson & Carroll 2002) or the UE Lifecycle (Mayhew 1999) provide numerous methods and tools that should support the designer in gathering all of the required information. Most of these methods are rated as heavyweight, due to their claim to analyze and document as much as possible about users, workflows, context, etc. right from the beginning.

User-centred design

(Constantine 2004) argues that user-centred design produces design ideas and visual prototypes out of the assembled requirements data by a rather magical process in which the transformation from claims to design is neither comprehensible nor traceable. Such a black-box designer produces creative solutions without being able to explain or illustrate what goes on in the process. Furthermore, user-centred design tries to make the resulting, often diverse, design alternatives converge into a single solution, which is then continuously evaluated and refined. User-centred design may therefore take a long time, or even fail, if too many users are involved and narrowing the design space is difficult. Iteration may create the illusion of progress, although the design actually goes round in circles and solutions remain elusive. On the whole, a one-to-one integration of user-centred design processes and methods is, in general, inappropriate for an agile course of action.

Table 11: Comparison of design approaches, adapted from (Constantine 2002); presented in (Memmel et al. 2007e)

User-Centred Design	**Usage-Centred Design**	**Activity-Centred Design**	**Thoughtful IxD**
Focus is on users: user experience and user satisfaction	Focus is on usage: supporting task goals	Focus is on activities	Focus is set by designer, dependent on project situation
Driven by user input	Driven by models	Driven by activities	Driven by UI designer
Substantial user involvement	Selective user involvement	Authoritative user involvement	Thoughtful user involvement
User studies Participatory design User feedback User testing	Explorative modelling Model validation Usability inspections	Sequential activity / task analysis Emotional design	Visual thinking with prototypes Participatory design Visual specification
Design by iterative prototyping	Design by modelling & abstract prototyping	Design by understanding activities	Design by switching btw. abstract and detail
Very varied, informal, or black-box process	Systematic, fully specified white-box process	Unspecified, rule-breaking black-box process	Depends on guidance and authority
Design by trial-and-error	Design by engineering	Design by authority	Design by visual thinking

Usage-centred design

Constantine suggests a concentration on tasks instead of on users and presents methods for an adequate requirements analysis. His usage-centred design approach takes up the basic philosophy of AM and concentrates on essential and easy to understand semi-formal models such as the role model, task model or content model. With the application of these medium-weight (agile) methods IxD becomes more formal, but the simplicity of their syntax still enables collaborative design with stakeholders. The models of usage-centred design can be applied by both IxD and SE personnel. Their thoughtful and common application can be an effective commitment to a UI design by engineering rather than by trial and error (Constantine 2004). Constantine also suggests prototypes as a tool for requirements elicitation, but argues that filling in detail too early often leads to premature decisions. He recommends abstract prototypes (Constantine 1998), which can be associated with low-fidelity prototyping.

User performance vs. user experience

Although the list of success stories for usage-centred design is creditable, the products praised tend to support user performance rather than user experience. Following usage-centred design, the project can be considered as successful when users are able to achieve their task goals. This cannot be the only aspiration of a modern design approach, however. For a modern cross-discipline software lifecycle, IxD must take into account more topics than user performance alone.

Activity-centred design

This is where Donald Norman's recently proposed activity-centred design approach (Norman 2005) comes in. Like Constantine, he argues that by concentrating on the user instead of on his tasks, user-centred design may be misleading and a time-consuming detour. But he also argues that task-completion time is probably not always the silver-bullet factor that causes software to be successful. Products causing a high joy of use can attain great user acceptance even when they lack usability. Norman therefore votes for the integration of emotional design issues and the stronger consideration of user satisfaction. Based on findings in the agile-usability group, Constantine is currently reworking his usage-centred design approach and is discussing the issue with Donald Norman:

"I am working to make activity theory more systematic and accessible to designers: to enable them to model and understand activity as quickly and agilely as practical so they can collaborate better with agile developers (...)." (Constantine 2007)

Thoughtful interaction design

In (Löwgren & Stolterman 2004), the designer, in order to design such highly usable and aesthetic systems, switches between 3 levels of abstraction: vision, operative image and specification. If the designer is confronted with a design situation, an often sketchy and diffuse vision initially emerges. Frequently, several visions are promising and therefore compete to be implemented, eventually resulting in a chaos of conflicting visions. The initial version of the operative image is the first externalization of the vision, e.g. captured in mock-ups or elaborated interactive (high-fidelity) prototypes. It enables manipulation, stimulation, visualization and decision-making for the most promising design. The operative image is eventually transformed into a specification for the final design if it is sufficiently detailed. In contrast to user-centred design or usage-centred design, (Löwgren & Stolterman 2004) describe design as neither a linear nor an iterative process. They identify no clear division between design and construction, but rather a continuous interaction between vision, operative image and specification. The designer wants to learn as much about the design space as possible, keeping the number of possible solutions high and narrowing the design towards the best solution as late as possible.

The elements of user experience

Taking into account the concepts of the approaches to UI design that have been discussed, a clear tendency towards an increase in the importance of user experience can be identified. Classic user-centred processes are no longer capable of covering all aspects of UI development in modern interactive systems design. The following cornerstones for specifying interactive UIs can be deduced from the discussion of UI design approaches:

- **Focus.** The focus is on both user performance and user experience. The process of UI specification therefore has to incorporate means that are able to take both aspects into account in equal measure.
- **Modelling**. Some degree of (semi-)formality, such as in the model-based usage-centred design approach, helps to turn UI specification into a transparent process. By an engineering-like approach to UI specification, the translation process from specific artefacts, i.e. models, into visual externalizations, i.e. UI prototypes, becomes more controllable.
- **Design**. The informality of user-centred design continues to be important in terms of visual thinking and participatory design. UI prototyping in particular is therefore the right means for communicating with stakeholders that are not on the design team. Prototypes drive the UI specification process by forcing stakeholders to become more objective and look at the consequences of their modelled requirements.
- **Control**: User experience is developed through a rule-breaking emotional design process and requires thoughtful and experienced interaction designers in control of the overall process. They are in charge of switching between user-input (user-centeredness), modelling (usage-centeredness) or authoritative decision-making.

The key aspects identified for UI specification now need to be structured in a way that makes sure that all important and relevant issues that determine usability and user experience are covered.

"The user-experience development process is all about ensuring that no aspect of the user's experience with your site happens without your conscious, explicit intent. This means taking into account every possibility of every action the user is likely to take and understanding the user's expectations at every step of the way through that process. It sounds like a big job, and in some ways it is. But by breaking the job of crafting user experience down into its component elements, we can better understand the problem as a whole." (Garrett 2002), p.21

The 5-S approach

(Garrett 2002) defined five elements that determine user experience, namely strategy, scope, structure, skeleton and surface (see Table 12). The scope of his '5-S'-approach was the internet, and he therefore distinguished between different kinds of websites, namely classical, primarily textual websites with static collections of information on the one hand, and dynamic, database-driven sites that are constantly evolving and have a much more complex presentation (rich internet applications) on the other. But because content (text) and presentation (e.g. graphics, structure) both affect the UI of software products in general, the different layers also give guidance on the design of software products in general.

Table 12: The elements of user experience and their alignment with bridging the gaps due to semi-formal modelling and different levels of abstraction

Element of 5S	Description
Surface	The concern here is the visual design, or the look and feel of the finished software product.
Skeleton	The skeleton defines UI design by the placement of UI elements and the arrangement of navigational items. The skeleton therefore determines the placement of concrete widgets and content. The skeleton optimizes the **information design** through the arrangement of these elements for maximum effect and efficiency, and in a way that facilitates understanding. Screen elements indirectly determine **navigation design** and how the user will move through the information architecture.
Structure	Structure defines the way in which the various features and functions of a software product fit together. With regards to the UI, structure therefore determines how users travel through UI dialogues and what each dialogue is supposed to provide to the user. **Interaction structure** is developed through interaction design, in which the system's behaviour in response to the user is defined. For information spaces, the structure is the **information architecture**, i.e. the arrangement of content elements within the information space.
Scope	The scope of software is defined through is features and functions. Strategy is translated into scope through the creation of **functional specifications**: a detailed description of the 'feature set' of the product. Scope also takes the form of **content requirements**, i.e. a description of the various content elements that will be required.
Strategy	Strategy incorporates the **needs** of different stakeholders, for example the organization that develops the software and the requirements of end-users. Strategy therefore also relates to return-on-invest, corporate **business goals** and what the outcome of delivering and using a software product must be. Moreover, the relationship of needs is especially important.

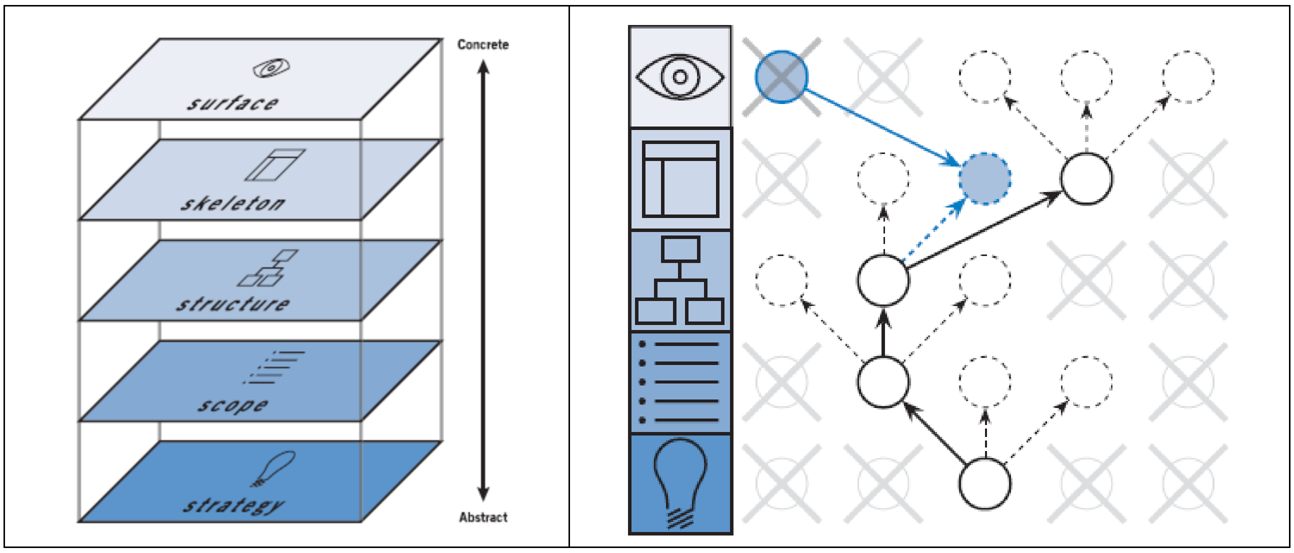


Figure 19: The different levels of abstraction when travelling from strategy to surface (left); decision-making and path towards a winning UI design; from (Garrett 2002)

The '5-S' provides a framework to guide design thinking, which can also be used to structure UI specification activities. On the lowest plane of the UI specification process, the interaction designer and the other experts and stakeholders will be concerned with the strategy behind the development of the software product (see Figure 19). The strategy therefore includes considering user needs, stakeholder needs and business visions or goals, as well as the relationship between all these aspects. The highest plane is concerned with the details of the appearance of the software product and therefore determines the look and feel at specification level in respect of completeness and maturity. Each plane, i.e. level of abstraction, is dependent on the other planes above and below it. (Garrett 2002) argues that when activities on each level do not align with those above and below, projects are derailed, deadlines are missed, and costs begin to skyrocket as the development team tries to piece together components that don't fit naturally. Hence, the aim of increasing project success and the ROI of interdisciplinary UI development with SE, UE and BPM can be pursued by following a structured approach similar to '5-S'.

3.3.2 Interaction design and business-process modelling

Modelling techniques

There are many different BPM methods that have been used with varying degrees of success. Most of them are based on visual techniques, i.e. they are usually a kind of diagram. The right modelling language has to support two different aspects of business processes. Structural aspects emphasise entities and relationships within the model. The model presents what each entity looks like and what each entity does. Behavioural aspects externalize the order in which things happen and under which conditions. This includes the matters of time, sequence and contextual scenarios. Flow charts are among the most frequently used diagrams for process modelling. The notation also is very popular in SE and UE (see Chapter 4). The business-process modelling notation (BPMN) is more specifically targeted towards process modelling. The idea behind BPMN is to provide a notation that all business people can understand and apply (Business Process Management Initiative 2008). UML is also recognized as a BPM language. Reasons for its increasing importance for process modelling are its popularity, its international acceptance and its standardization. The popularity of UML for BPM can support the search for shared means of communication among stakeholders.

Business-process modelling and the three evils of life

Most business-process notations, and BPMN is an example, have a similar structure to represent the sequence of work and the decomposition of the organizational complexity (see Chapter 2). Activities, linked with each other, and decision points, providing other paths to follow, have a deterministic graph representation. They can be decomposed into other activities and these other activities can be decomposed in turn, forming a tree representation where a new level is created when motivated by the necessity to better describe what is to be done. With the introduction of an appropriate notation for business-process modelling, an organization has to overcome three 'evils of life' for modelling (Holt 2005):

- **Complexity**: in the case that information is unstructured or the overall information architecture and relationships are very complex.
- **Lack of understanding**: if the source information is poorly understood, even well-working processes are not very robust in the face of change or tailoring.
- **Poor communication**: if process knowledge must be shared between several individuals, everyone needs to externalize their personal experience. Especially where a process is a tool used in daily business, documenting all pertinent knowledge is sometimes difficult (tacit knowledge). Moreover, very badly written text can lead to misunderstandings and ambiguity.

Seven views of business processes

Although BPMN is also a widespread modelling language, it is too narrow to meet all the requirements of process modelling. Stakeholders all have a different viewpoint as to a process's requirements. In fact, within any process there will be a number of viewpoints. Accordingly, (Holt 2005) has presented seven different views that need to be considered during process modelling (see Table 13). Four of these views are realized by structural diagrams, for which BPMN has no support. The requirement view specifies the overall aims and is essential for validation. Requirements need to be checked periodically, as they change over time. Stakeholders need to know if the process requirements have changed. The process-structure view specifies the structure of concepts and terminology used and helps to pin down the language used. The process-content view identifies all the processes of interest for a system. For each of these processes, the view encourages the analysis of actual requirements and process-description breakdown. The stakeholder view identifies shareholder roles within an organization, project or system. This view presents shareholders in a classification hierarchy, which can be done in UML class diagrams. The process-behaviour view describes the behaviour of a single process. Each process of the process-content view has a process-behaviour view. The behaviour view can be modelled in UML with an activity diagram. The information view is concerned with identifying the key artefacts from the system and then identifying their inter-relationships. This is to guarantee inter-process consistency in order to make sure that processes are compatible. The process-instance view ties everything together and links back, showing instances of processes and stakeholders and how they interact (Holt 2005).

Table 13: Support for important views in UML and BPMN; based on (Holt 2005)

View	UML Representation	BPMN Representation
Requirements	Use-case Diagram: requirements shown as use-cases, stakeholders as actors	No representation: stakeholder information only found as swim lanes, no concept for requirements
Process Structure	Class Diagram	No representation
Process Content	Class Diagram: process as classes, artefacts as attributes; activities as operations	No representation: understanding content through looking at diagram in detail
Stakeholder	Class Diagram: each stakeholder shown as a class	No representation: stakeholder information only found as swim lanes
Process Behaviour	Activity Diagram: stakeholders shown as swim lanes, activities as invocations, artefacts as objects	Business-process diagram: stakeholders shown as swim lanes, activities as activities, artefacts as data object
Information	Class Diagram: artefacts shown as classes	No representation: artefacts only found as data objects on business-process diagram
Process Instance	Sequence Diagram: each process shown as a lifeline	Business-process diagram: connected sub-processes show executions in sequence

Structural and behavioural view

For each view, from one to several different diagrams are necessary. (Holt 2005) suggests class diagrams to model structural aspects of a process and activity diagrams to model behavioural aspects. Activity diagrams are derived from flow charts and can be said to be the UML-like notation of them. Sequence and use-case diagrams can be used to model behavioural aspects on a higher level. Use-case diagrams also contribute a requirements view of a process. It can be used to model the context of a system from the perspective of different stakeholders. Most process models have to contain a very large number of processes. It is therefore important to partition processes into smaller parts. When the key features of processes have to be defined, it is important to understand the activities that happen in the process. Activities are usually iterative although they may have been modelled in a linear fashion. Relationships and interactions between elements of a process can be shown through graphical paths in the process model.

“Though there are several process notations used in the industry, we find the basic information in business processes having to do with ‘who’ will do which task ‘when’ in the business process, the sequence of business tasks, and what are the inputs and outputs of a task, to be the most useful set of information for starting a user interface design.” (Sukaviriya et al. 2007)

BPM and UI development

With regards to UI development, business processes can play an important role. It is important to maintain a close connection between business processes and the UI. Business analysts analyze and model business processes that serve as requirements for the UI. Having high UI quality normally aligns with business needs (see Chapter 2). Product knowledge must not be owned just by business analysts, but must be shared with other stakeholders. Concerning the UI, interaction designers must have a clear understanding of the processes of an enterprise. In many application domains, e.g. banking, the process view has significant influence on the functionality and usability of the software and the UI. This is especially the case with applications that directly map the nature of a process to the UI, such as form-based account or credit application processes (banking) or configuration tools for consumer products. The information buried in various BPM artefacts must therefore be visually externalized to be able to design for user experience. (Sousa et al. 2008c) accordingly argue that business processes alone are a limited representation for visualizing the information needed for UI design, because business processes (1) ignore how an activity is accomplished, (2) do not encompass all tasks that are intrinsic to user interaction, and (3) are not detailed enough to describe individual user behaviour. Considering the importance of user interaction, business processes must therefore be aligned with the notations of SE and IxD.

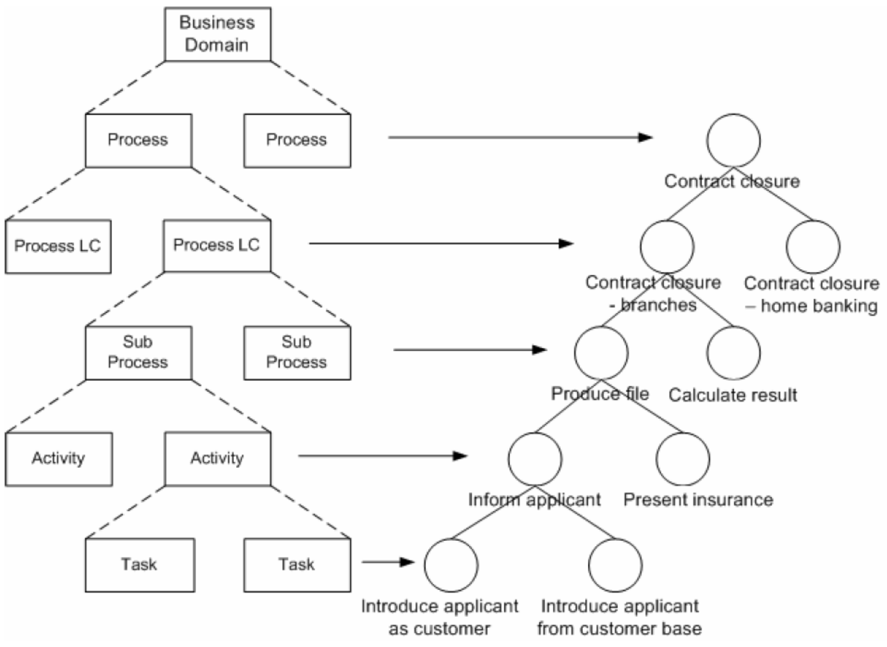


Figure 20: Mapping of BPM and task model; from (Sousa et al. 2008c)

BPM and Task models for UI development

(Sousa et al. 2008a; Sousa et al. 2008b; Sousa et al. 2008c) propose task models as an intermediate modelling language that could bridge process modelling and UI development. The task model represents the tasks performed by users when interacting with a system. Task models are a strong representation for UI design because they contain decomposition in a hierarchical structure, which provides an overview of the user interaction. In Figure 20 the six decomposed layers of the business-process model are associated with the hierarchical levels of a task model. The association starts with the end-to-end process layer because the business domain represents the overview of the process architecture. Changing from a business perspective to a user-centred perspective, it is necessary to focus on certain aspects of user interaction. Task models can be used as a common denominator between business processes and UI design. Business processes are often an abstract representation of the business tasks. They differ from task models, which are a concrete set of tasks that help UI designers visualize what could happen during user interaction. As a result, business-process models can be used as requirements, not in direct connection with the UI design, but for the creation of task models before the UI design (Sousa et al. 2008c).

3.4 The Concept of Interactive User Interface Specifications

Separation of concerns

When developing interactive UIs, close cooperation between team members with different backgrounds, knowledge and experience is one of the key success factors. One approach for coping with the inherent technical complexity of interactive systems and the organizational complexity stemming from the indispensable interdisciplinary teams is a strict separation of concerns (Dijkstra 1976), i.e. the modularization of development tasks as proposed by the Seeheim model (Green 1986). In the realm of UI specification, (Bock 2007a) for example proposed the separate specification of layout, content and behaviour, and the corresponding models. Layout relates to screen design and the user-friendly and task-adequate arrangement of dialogue objects. Content refers to the definition of information to be displayed. Finally, behaviour describes the dynamic parts of a UI with respect to controls available on a specific target platform and a system's business logic. Changing a specific model must consistently affect dependent models and consequently the visual presentation in terms of a UI prototype as well. For the generation of the UI prototype, the different parts finally have to be integrated. As well as modularization, abstraction also offers further means for coping with technical and organizational complexity. By presenting information at different levels of abstraction, it is possible to provide developers with only the relevant information for their specific development tasks.

The 5-S approach for UI specification

Structuring different aspects of requirements modelling and design is essential for a sound UI specification process. The '5-S' approach proposed by (Garrett 2002) also distinguishes between different concerns that relate very well to the separation into layout, content and behaviour (see Figure 21). Its different levels of abstraction make it a perfect framework for UI specification that is segregated into different stages.

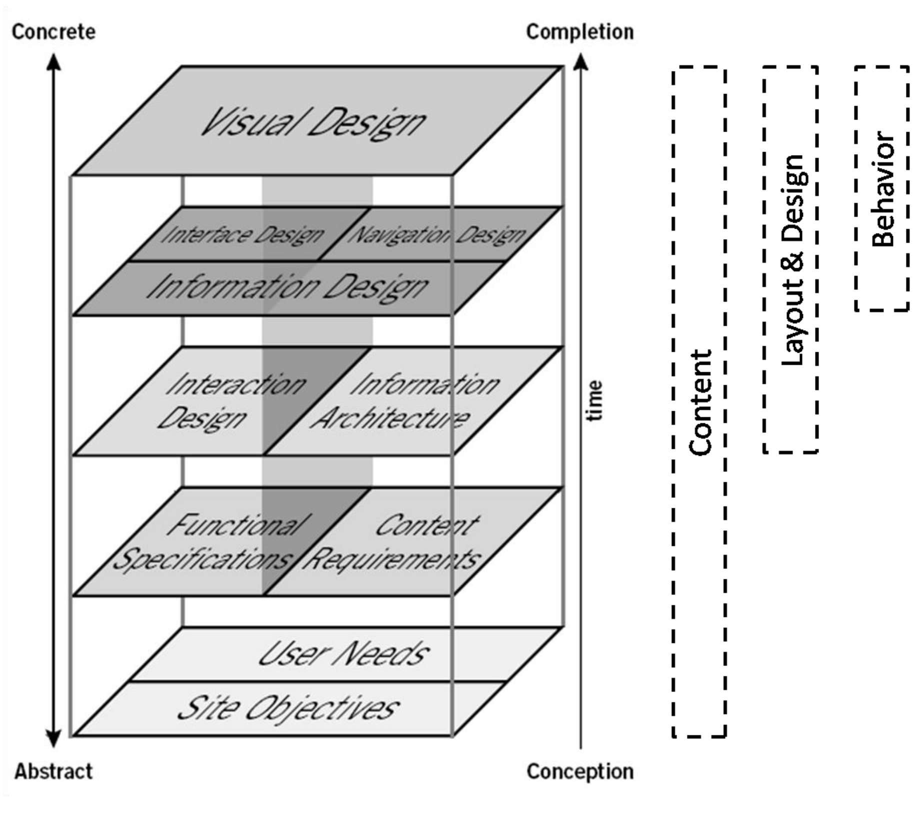


Figure 21: The elements of user experience (5-S-approach) related to content, layout and design, and behaviour of the UI; based on (Garrett 2000)

The stages determine the UI specification process by initiating a journey from largely text-based artefacts (descriptions of user needs or objectives/business goals) through graphical notations, i.e. semi-formal models (see Chapter 3.2), to a detailed UI prototype. Here, content is especially defined through the rather abstract artefacts that are developed at the early stages. Layout, as well as functionality and structure, is defined through the graphical notations. And behaviour, i.e. the interactive mesh up of look and feel, is most notably specified at the detailed UI design layer (see Figure 21).

The spectrum between the abstract and the detail

With regards to the different layers of the '5-S'-approach, the spectrum between the abstract and the detailed (see Figure 22) must be filled with appropriate artefacts that can successfully drive the UI specification process. As discussed in Chapter 3.3, the bandwidth of UI specification artefacts is bounded by scenarios on the one end, and by detailed UI design prototypes that determine the UI specification on the other. This aligns with the '5-S'-approach, as scenarios are a widely used notation for documenting user needs or objectives and guidelines. Both means of expression are especially common to IxD and SE. From all disciplines, including BPM, a variety of useful models can be considered as an important part of a UI specification. However, the other notations required to transparently and traceably travel from one end of the spectrum to the other are not yet determined in the literature. Considering the elements of user experience in the '5-S'-approach, there is at least a framework that can guide the search for the missing elements.

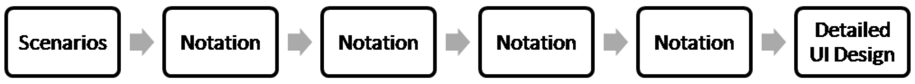


Figure 22: The open spectrum between scenarios and prototypes of detailed UI design

Broadening perspectives

(Richter & Flückiger 2007) propose a set of models and means of UI modelling and specification (see Table 14). Their approach fits very well into the bandwidth of artefacts determined through the in-depth analysis of IxD, SE and BPM (see Chapter 3.3). But the models presented by (Richter & Flückiger 2007) do not, however, explicitly consider the issue of bridging the gap. The authors focus their UI specification approach on UE and therefore do not discuss the contribution of different kinds of dialogue notations (see Chapter 3.2) such as flow charts. Hence, the perspective must be extended in order to determine a complete set of artefacts that make up a UI specification.

Table 14: Overview of SE/BPM and UE/IxD methods that contribute to UI specification; excerpt from (Richter & Flückiger 2007)

Activity	**SE/BPM Practice / Models**	**UE/IxD Practice / Models**
Modelling	Business modelling, use-case diagram, use-cases	Personas, scenarios, storyboards, UI prototyping
Specification	Use-case model, use-case specification, non-functional requirements, sequence diagrams	Scenarios, storyboards, UI prototypes, style guides

New approaches to UI specification

Design-driven, well-elaborated UI specifications can significantly enhance the quality of a software product. Considering the value of CD and CI, important design decisions that affect usability, and look and feel are critical. Current specification practice is at odds with the demanding factors of corporate interactive system development (see Table 15). Critical parts of the system, including in particular the real UI behaviour, have to be evaluated before the supplier writes a single line of code. This ensures that the later UI matches corporate values and therefore avoids design failure as well as costly late-cycle changes. The UI specification needs to be as complete, unambiguous and expressive as possible. Whenever the supplier is unsure about the client's demands, he should be able to pop-up an authoritative interactive UI specification to get guidance on the required UI properties. Furthermore, a standardized UI specification is non-sensitive to a change of supplier and provides more flexibility in choosing appropriate partners. Today, incomplete UI specifications are only successful because of the backup of well-developed client-supplier relationships and this frequently causes certain dependencies. But the supplier also benefits if the UI specification method previously used is compatible with his own course of action. He is then able to reuse existing UI descriptions or even code.

Table 15: Current and required UI specification practice

Current shortcomings	**Necessary action**
Heavyweight processes with huge number of artefacts	Travel light, due to pressure of time
Inappropriate Office-applications preferred due to ease of use; CASE-tools avoided due to poor usability	Provide means of expression focused on modelling the UI specification. Help to increase modelling skills of actors
Paper-based documents come with ambiguity and easily become inconsistent	Use interactive UI specifications to specify the interaction layer
Supplier assigned to build detailed prototypes	Look and feel must be evaluated as soon as possible
Throw-away prototypes for design and business vision	Use authoritative UI design to guide IT supplier
Different means of modelling lead to costly late-cycle changes	Bridge the gaps in communications with problem-adequate tool support
Process relies on long-term relationship with supplier	Become more flexible by increasing own expertise
Existing HCI models are decoupled from the overall development process	Identify a common denominator and use interoperable models

Towards new forms of corporate UI specification

We therefore need new approaches to UI specification practice that go beyond the usual prototyping efforts. With regards to semi-formal model-based approaches, the following goals for corporate UI specification can be defined:

- To provide a framework for understanding and using different semi-formal models and different UI design representations, i.e. UI prototypes.
- To develop an integrated set of diagram-based semi-formal models for specifying and describing UIs. The models should be interoperable with, or based on, languages used in information-systems engineering in order to bridge the gaps between the disciplines.
- To outline how models can be used for capturing design knowledge for reuse across problems and projects.

Interactive UI Specifications

Although prototypes (goal 1) can be a common denominator for UI design at the interaction layer, each discipline still needs to employ certain (semi-formal) models (goal 2) during UI specification. As an integrating vehicle of UI specification, we propose interactive UI specifications. They are based on the concept of detailed UI prototypes, which can function as UI specifications (Rudd et al. 1996; Memmel et al. 2007g; Memmel et al. 2008e), but they include additional information that is 'below' the design layer (see Table 16). Through a visual drill-down, for example, subsidiary artefacts such as dialogue notations or story-like descriptions can be accessed (see Figure 23).

Table 16: Main differences between prototypes and interactive UI specifications.

Interactive UI Prototypes	**Prototyping-Driven Interactive UI Specifications**
Vehicle for requirements analysis	Vehicle for requirements specification
Exclusively models the UI layer; may be inconsistent with specification and graphical notations	Allows drill down from UI to models; relates UI to requirements and vice versa
Either low fidelity or high fidelity	Abstract first, specification design later
Supplements text-based specification	Widely replaces text-based specification
(Mostly) driven by specification	Driven by UI prototypes (prototyping-driven UI spec.)
Design rationale saved in other documents	Incorporates design knowledge and rationale

Interactive UI specifications additionally extend the value of UI prototypes in terms of increased expressivity and transparency of design rationale (goal 3). Because switching between the concrete and the detail is what drives design, the prototypes function as an extension of the mind and visualize the requirements in a way that is more tangible than text. The prototypes are therefore driving the specification process (prototyping-driven UI specification; see Chapter 1). For this purpose, they are permanently dependent on the created models, which incorporate the needs and requirements that have to be translated into the UI design.

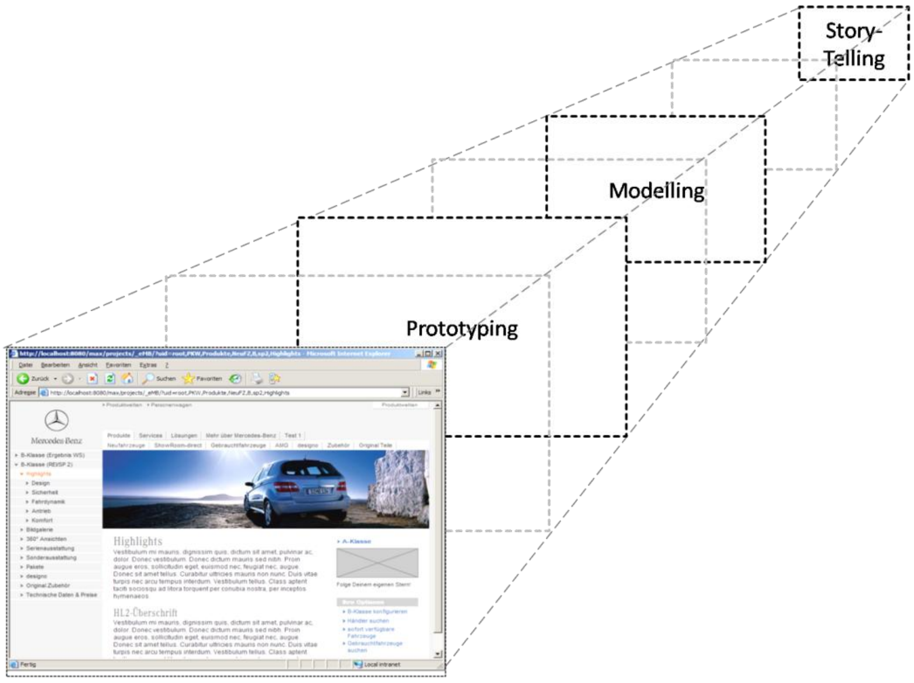


Figure 23: Layers below the UI design specification, using the example of the Mercedes-Benz website, as published in (Memmel et al. 2007g; Memmel et al. 2008e)

UI prototypes vs. interactive UI specifications

As outlined in Table 16, there are several differences between UI prototyping and the interactive UI specifications as we propose them. Whenever IT suppliers need guidance on how the system must look and feel, they can pop-up the simulation and easily build the corporate software system accordingly. Integrated information, accessible at both the UI and in the underlying models, makes the UI design rationale transparent for stakeholders. Paper-based documents become less important and the overall process becomes more reliant on expressive interactive representations.

Advantages of interactive UI specifications

By sharing and collaboratively discussing interactive UI specifications, different groups of stakeholders and users can crosscheck the UI design with their requirements (correctness, clearness). It is unlikely that certain user groups will be ignored when stakeholders have access to a UI prototype (completeness). When certain user tasks demand exceptional UI design, a visual simulation will be more capable of expressing such complex parts of the system and able to illustrate their meaning and compliance in the system as a whole (consistency, traceability). Ambiguity, redundancy, missing information and conflicts will also be more obvious. The interaction designer will be able to identify ambiguity through the evaluation of UI expressiveness, affordance and mapping. He will be able to identify redundancy and conflicts when assessing screen spacing, layout or navigation structure. The absence of specific information will attract attention through ‘white spots’ on the screen or missing visual components. With interactive and expressive designs, interaction and functional issues can be addressed sooner, and the identification of usability requirements can be done as soon as the early stages of design (Folmer & Bosch 2004) and before coding starts. Unnecessary system functionality can be identified through UI evaluation methods rather than by reading through text. All in all, an interactive UI specification can be assessed more easily and to some extent the creation of a prototype proves the convertibility of the UI design into a final system.

Perspectives on the interactive UI specification

Interactive UI specifications can significantly reduce the effort on the programming side as well: building the UI of a system with the help of a running simulation (prototype) is much easier than doing it from scratch based on textual descriptions. A developer (IT supplier) can quickly look at the simulation in order to get a visual impression of the requirements. If the derivation of a certain UI element or a whole dialogue flow is unclear, the developer can drill down to the lower levels of the interactive UI specification to understand the design rationale. He could look at design alternatives to understand the decisions made or he could even take a closer look at the tasks modelled or the user requirements (see Figure 24). Moreover, when the creation of prototypes takes place in a model-based UI specification process, models can carry forward standards and design rules. Once properly defined, they can be externalized by different representations and in a different level of abstraction and formality. On the one hand, this eases access for different stakeholders. On the other hand, while a running simulation is the visual outcome, the underlying models capture design knowledge and decisions.

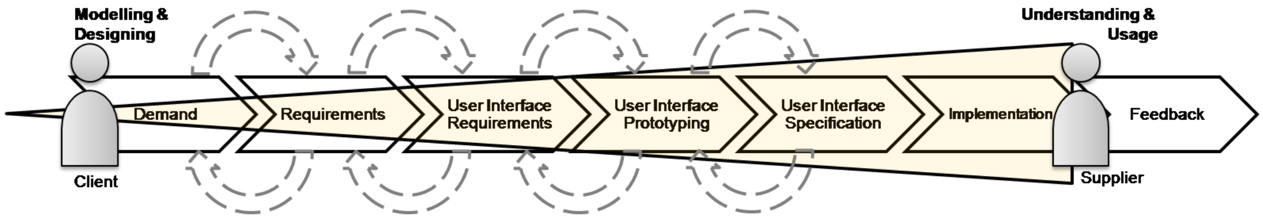


Figure 24: The UI specification process from the perspective of client (left) and supplier (right)

Interactive UI specification as complete package

In summary, interactive UI specifications can drive the design process by including all of the aspects important for describing the UI such as scenario descriptions, user profiles, and semi-formal (agile) models (see Figure 25). An additional contribution is their ability to record design knowledge in the integrated artefacts. They enable a reuse of experience from previous UI design projects by simply running the UI specification rather than scanning through documents. Instead of being a supplement to textual specifications, they decrease the number of text documents, which tend to be substituted by a running simulation that is capable of being experienced. They support a strongly needed traceability of requirements (Hoffmann et al. 2004).

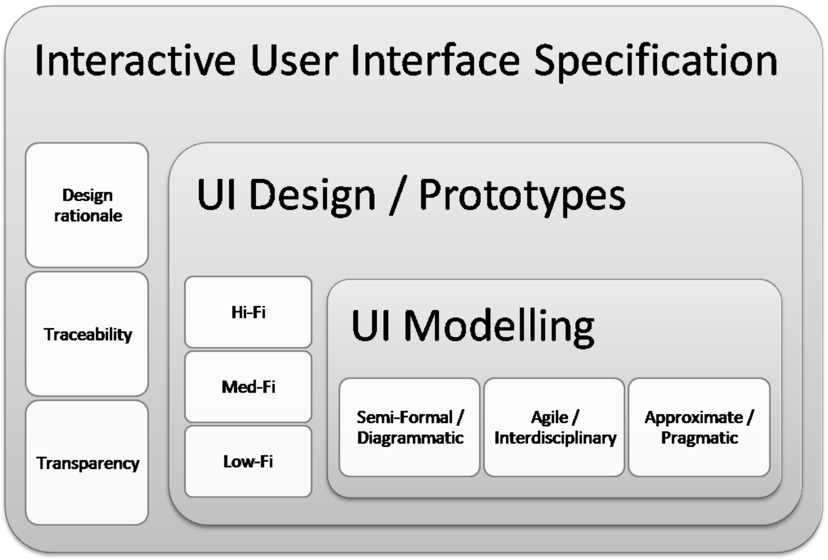


Figure 25: The composition of interactive UI specifications

3.5 Key Points

- In UI design there is recurrent jumping between the concrete and the abstract, as well as between the details and the whole. Switching between artefacts of a different grade of abstraction is among the most import tasks of designers. The process of UI specification therefore has loose boundaries between abstract, detailed and final design stages.
- Style guides are important instruments for saving design knowledge and design rationale. But a style guide is not a UI specification. (Mayhew 1999) proposes that the UI specification describes the UI design in detail, mainly using visual cues and graphics, and using text just where aspects of the design cannot be illustrated. However, what exactly makes up a UI specification is still a 'white spot' in IxD literature.
- With regards to UI specification, style guides contribute the idea of having a shared and elaborated knowledge repository. The safekeeping of design principles, guidelines and rationale is important for UI specification and requires adequate formats.
- The right format of UI specification must be determined with regards to the formality of its notation. Informal, text-based notations are a good starting point for many projects, but are an insufficient means of communication and UI design during later stages due to ambiguity and a lack of precision, for example. Formal notations are very precise, but most stakeholders are unable to employ them correctly due to their strict and often complex terminology. Ultimately, semi-formal diagrammatic notations are a good compromise and provide an easy-to-use access to UI modelling, while still maintaining enough formalism to guide UI specification.
- Semi-formal model-based UI specification relies on the identification of models that are able to propel the interdisciplinary process of UI specification. Therefore, a careful choice of models that are intended to contribute to requirements analysis and UI design is necessary. For a shared repository of models, a common ground for SE, IxD and BPM has to be identified. By considering the principles and practice of agile modelling, ways of bridging these disciplines can indeed be identified. Agile models match very well with the concept of semi-formality, by being less strict and more approximate and pragmatic.
- Through the identification of interdisciplinary interfaces and semi-formal modelling and UI prototyping, a new means for UI specification can be inferred. The concept of interactive UI specifications incorporates the richness and expressivity of UI style guides and incorporates aspects of design rationale with agile, semi-formal diagrammatic models plus their visual externalizations presented through UI prototypes. On the whole, the unification of important artefacts within a new form of UI specification, being interactive in terms of presenting its contents, contributes very positively to the demands of corporate UI specification.
- In order to define the ingredients of a sound interactive UI specification in detail, different kinds of semi-formal models must be determined. Moreover, adequate kinds of UI prototypes must be chosen from the variety of prototypologies that exist in SE, IxD and BPM. The identification of both the right set of models and the right set of UI prototyping methods leads directly to the development of a common interdisciplinary denominator for UI-related modelling and design. To a great extent this constitutes interactive UI specifications and fills a 'white spot' in IxD literature.

Chapter 4 The Common Denominator For Interactive UI Specifications

Following the discussion on a road map for interdisciplinary UI specification, this chapter presents the most appropriate ingredients for interactive UI specifications in corporate-development processes as discussed in the previous chapters. In accordance with the earlier consideration of different grades of formality, the set of artefacts presented in this chapter depends on a semi-formal model-based UI specification process that is shaped by agile models and expressive externalizations of UI prototypes. At all stages, the proposed artefacts are intended to bridge the gaps between the disciplines and therefore build upon the similarities identified and presented in Chapter 3.

The structure of this chapter

Chapter 4 is structured as follows: in Chapter 4.1 we present the different kinds of modelling notations suitable for the purpose of UI specification. The notations are categorized into models for describing problem-domain, users, tasks and behaviour (i.e. dialogue specification, see Chapter 3). In Chapter 4.2, different kinds of UI prototypologies are discussed and forms of prototyping that offer good support for the process of UI specification are determined. The whole set of models and means of prototyping is finally summarized in Chapter 4.3, together with a graphical preparation of the proposed common denominator. The chapter ends with some key points.

4.1 UI Modelling

Why UI modelling?

The extensive text of UI style guides or documents that describe screen-design standards is often ambiguous and can leave room for the danger of different interpretations. Formal models (e.g. UML) are generally too complex for UI design and interaction designers, in comparison with SE experts, usually do not have a sophisticated modelling background. In interdisciplinary UI specification, the interaction designer must create communication between disciplines that are normally divided. The approach known as interactive UI specifications (see Chapter 3) employs specific modelling languages to enable different actors to express and model their requirements in a way compatible with the expertise of others. Finally, the different requirements are combined in an interactive specification that simulates the look and behaviour of the UI.

"Using models as part of user interface development can help capture user requirements, avoid premature commitment to specific layouts and widgets, and make the relationship between an interface's different parts and their roles explicit." (Silva & Paton 2003)

Semi-formal modelling of different views of the UI

The process of UI specification can only be successful, if it is based on a solid organisation and documentation. Adequate methods for requirements analysis and specification are necessary to make sure the specification can be used by developers and for communicating issues with end-users. As outlined in Chapter 3, a semi-formal model-based approach, in combination with various views of the requirements, is a promising starting point. The use of different notations to externalize the same problem or idea offers good support for communication with the user (Forbrig 2001). The modelling of an interactive system can be divided into user modelling, task modelling, interaction modelling and problem-domain modelling. A user model characterizes users in terms of their experience with specific tasks, their organizational work style and their preferences concerning the interaction with the UI. A task model describes static and dynamic aspects of a work style. A task is determined by a target, a certain number of sub-tasks, the objects worked on, and the existing work equipment. The interaction model describes the structure and the behaviour of the interactive UI. The problem-domain model can also be described as a business-object model or business process and refers to the attributes, methods, relationships and behaviour of objects. This includes work material and tools. The problem-domain model therefore describes the context and environment in which the modelling process takes place. Although technical and UI aspects are usually not included in the problem-domain model, it is an important source of knowledge for the UI specification supply chain (see Chapter 4.1.1).

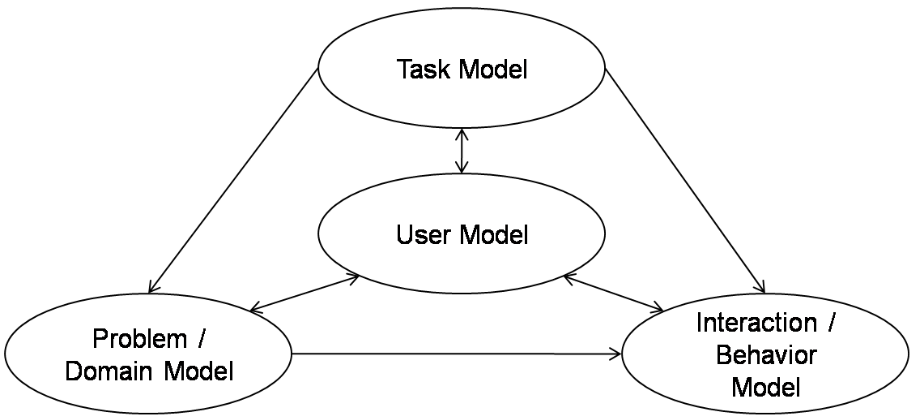


Figure 26: Relationships between models of a model-based UI specification; from (Forbrig 2001)

Modelling in sequence and iteration

In usage-centred design (Constantine & Lockwood 1999b), models guide the designer throughout the UI development process (see Figure 27). The final visual and interaction design is derived more or less directly from a content model or abstract prototype (Constantine 1998; Constantine 2003) that describes the content and organization of the UI independently of its detailed appearance and behaviour. The content model is itself based on a comprehensive task model expressed in the form of 'essential use-cases' or 'task cases' (Constantine & Lockwood 1999a). Task cases, in turn, support user roles as represented in the user role model. In usage-centred design, the problem-domain consists of modelling in the operational model, which documents the organizational environment and context (Constantine & Lockwood 2002).

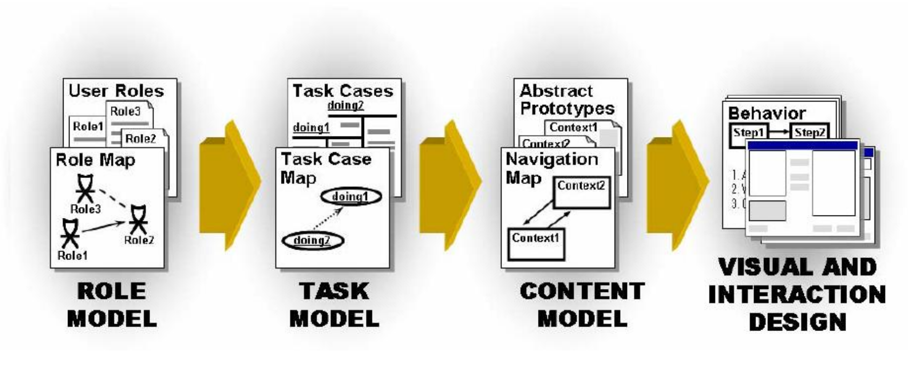


Figure 27: Logical dependency of models in usage-centred design; from (Constantine 2005)

UI specification vs. data gathering

In the following sub-sections this sequence is also employed for the definition of a corporate UI specification process. However, the development of a UI specification and the contributing artefacts is not a linear process, but a highly iterative and networked one, as most UI development approaches make clear (see Chapter 2). Hence, the different kinds of models and prototypes presented in the following chapters usually depend on other artefacts. For example, writing scenarios depends on a previous analysis of stakeholders and directly provides input to task modelling and interaction design. A close relationship between the different artefacts is therefore essential and, as explained in Chapter 3, is one of the primary goals introduced by the concept of interactive UI specifications. Moreover, the artefacts presented also depend on different methods of data gathering that make UI modelling and UI design possible in the first place. However, the process of data gathering is outside the scope of this thesis on developing interactive UI specifications. The ingredients of the specification are simply the manifestations of the collected information. Nevertheless, opportunities to support the data-gathering process with the proposed approach to interactive UI specification and the experimental tool (see Chapter 6) will be discussed in the Outlook section of this thesis (see Chapter 8).

4.1.1 Problem-Domain Modelling: Methods and Artefacts

Why problem-domain modelling?

In the realm of UI specification, the problem-domain model (Forbrig 2001) or operational model (Constantine & Lockwood 1999b; Constantine & Lockwood 2002) describes the context and environment in which the UI-related modelling process takes place. With problem-modelling focused on the UI, the organizational preconditions that influence the UI specification process can be taken into account. In the following, the means for identifying problems (IxD) and domain models (SE), and for developing problem-solving and strategic goals and visions (BPM) are presented. Each of the methods discussed is soundly anchored in its mother-discipline and can contribute an appropriate format or content, or both, to the development of interactive UI specifications.

Business vision, strategy and the 5-S approach

Understanding the problem-domain and identifying potential improvement is also related to the first level of the 5-S approach for designing user experience. In the 5-S framework (see Chapter 3), strategy incorporates the needs of different stakeholders, including the organization that develops the software product. Besides the necessity to model needs, it is also part of the strategy stage to determine objectives that are supposed to drive the process. Here, the definition of mission statements and goals helps to build a common ground for all stakeholders. A business vision describes corporate business goals and what the outcome of delivering or using a software product must be. Many software projects lack a transparent set of goals and thus the actors may lose track of the project's aims. It therefore makes sense to include vision statements in the interactive specification package to provide information about the motivation and motives for designing a specific interactive system.

Table 17: Problem-domain models contributed from different disciplines; (see Chapters 2 + 3)

Contributing Discipline	**User Model**
Software Engineering	Domain models (Ambler & Jeffries 2002; Ambler 2004b)
Usability Engineering	Problem scenarios (Rosson & Carroll 2002); operational model (Constantine & Lockwood 1999b)
Business-Process Modelling	Business Vision (Collins & Porras 1996; QuickMBA.com 2008; Rational Software Corporation 2008)
Common denominator	Problem scenarios (Rosson & Carroll 2002)

4.1.1.1 Problem Scenarios (UE)

Understanding current practice

Discussions with stakeholders are essential to gain an impression of the problem-domain. A feature may have different consequences for different stakeholders, so it is important to consider effects that might occur in other scenarios. Problem scenarios tell the stories of current practice. They do not emphasize problematic aspects of current practices, but describe activities in the problem-domain. It is as important to know what is working well as it is to find the difficulties that must be addressed. Problem scenarios give insights into the current situation, describing the needs and opportunities. Problem scenarios are therefore the initial models of the analysis phase in scenario-based UE (Rosson & Carroll 2002) and feed the subsequent design and construction phase (see Figure 28). For the process of developing an interactive

UI specification, problem scenarios are also a good starting point, as they help to develop a common understanding of current difficulties and opportunities for enhancement. Moreover, the format of problem scenarios facilitates the easy collection of this information (see Table 18). Unlike in later stages of UI specification, the textual format of scenarios at this stage can help to describe the status quo directly from storytellers' current experiences as well as from their memories. Deducing the important information from such a scenario is then an important part of user and task modelling, having a concrete semi-formal format. Actors in problem scenarios are hypothetical stakeholders based on the characters of people interviewed. The stories are carefully developed to reveal aspects of the stakeholders and their activities that have implications for design. Other members of the project team are able to read the problem scenarios and understand the work-related issues that the field study has uncovered.

Problem scenarios and claims

In scenario-based UE (Rosson & Carroll 2002), scenario writing is always connected to claims analysis (see Figure 28). Claims describe features of a situation that have an influence on the stakeholders. Problem scenarios describe how particular features are believed to impact activities and experiences. Claims provide a more explicit analysis of these impacts.

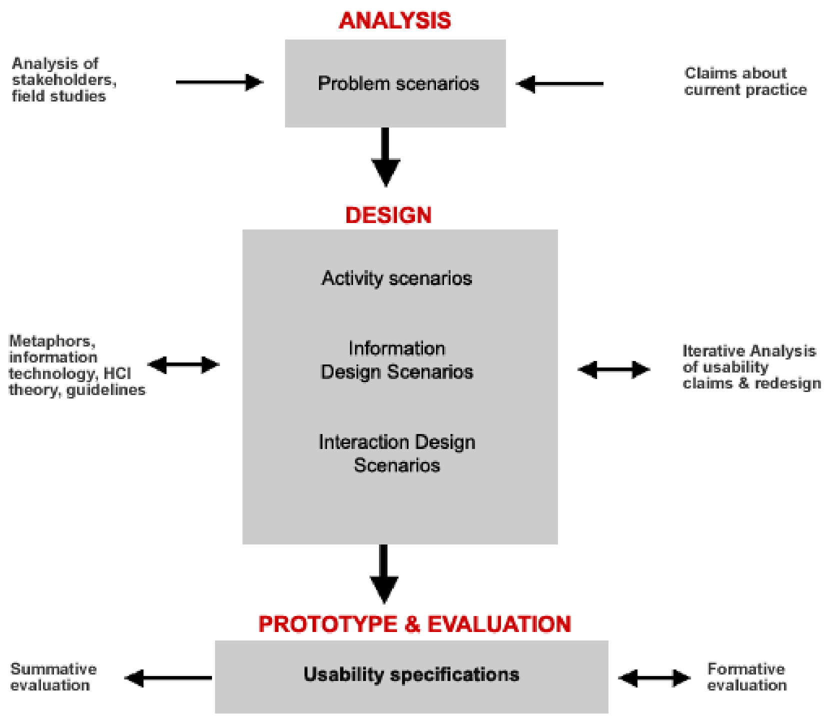

ANALYSIS
Analysis of stakeholders, field studies
Problem scenarios
Claims about current practice
DESIGN
Activity scenarios
Metaphors, information technology, HCI theory, guidelines
Information Design Scenarios
Iterative Analysis of usability claims & redesign
Interaction Design Scenarios
PROTOTYPE & EVALUATION
Summative evaluation
Usability specifications
Formative evaluation

Figure 28: The scenario-based usability engineering lifecycle (Rosson & Carroll 2002)

A problem claim is about the positive and negative effects of features on the stakeholder's experience. Claims are examples of trade-offs, where trade-offs are reflected in the claim's upsides and downsides. The difference is that claims are tied to specific artefacts and activities, while trade-offs are general statements of competing concerns. Claims can be seen as instances of general trade-offs and play several important roles:

- Claims describe the impact of a particular feature on the stakeholder.
- Claims analysis records why scenarios have been written by isolating the most important features of the narratives.
- Claims extend scenarios, pointing to possible effects that a feature might have in other scenarios.

- Claims analysis promotes a balanced view of a situation (positive and negative impacts).
- Claims motivate design reasoning. Designers will try to increase positive impacts while decreasing negative ones.

Goals and obstacles

Methods of determining the scenarios usually involve finding goals (Rolland et al. 1999). A goal is a statement of a particular problem that, to meet a user's requirement, needs to be solved (Alexander et al. 2004). A scenario can be expressed as a series of actions and system responses that attempt to reach the goal. During this attempt, obstacles may prevent the goal from being reached. These obstacles should be documented in the problem scenario to guide stakeholders during their efforts in solving the relevant issues and thereby enhancing the quality of the software.

Table 18: Problem scenarios in brief; based on (Rosson & Carroll 2002)

Primary use	Understanding current practice
Modelling technique / form	Written text; one scenario per stakeholder, respectively several scenarios if stakeholder is involved in many activities
Advantages	Easy to develop; easy to understand (by all stakeholders); can be enhanced during development stages; emphasizes verbal communication; low detail first, more detail later
Disadvantages	Interviewed stakeholders need not necessarily be the end-users; time-consuming to create
Related to	Stakeholder analysis, task analysis

4.1.1.2 Domain Modelling (SE)

Introduction to domain modelling

A domain model captures the most important types of objects in the context of the business. The domain model represents the 'things' that exist or events that transpire in the business environment (Jacobson 1992). The domain model provides a conceptual framework of the things in the problem space and helps one to think and focus on the semantics of the context. Typically, a domain model also determines specific terms that fit the application domain particularly well.

Class diagrams

Following (Constantine & Lockwood 1999b; Ambler & Jeffries 2002; Ambler 2004b), class diagrams are typically used to explore domain concepts in the form of a domain model. Each domain class denotes a type of object. It is a descriptor for a set of things that share common features. Classes can be (1) business objects (represents things that are manipulated in the business), (2) real-world objects (things that the business keeps track of), (3) events that transpire. A domain class has attributes and associations with other classes. It is therefore important that a domain class is given a good description.

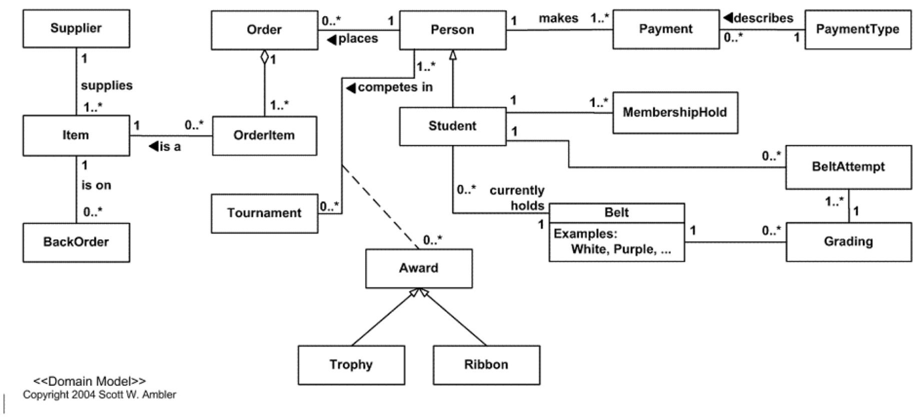


Figure 29: A simple domain model for a university IT application (Ambler 2004b)

In the domain model, each domain class denotes a type of object. It has attributes, associations, and additional rules. An attribute is the description of a named slot of a specified type in a domain class. An association is a relationship between two (or more) domain classes that describes links between their object instances. Associations can have roles, describing the multiplicity and participation of a class in the relationship. Additional rules are mostly complex and can be shown not with symbols, but with attached notes. Modelling with class diagrams is compatible with BPM as suggested by (Holt 2005)

CRC cards

A domain model identifies fundamental business entity types and the relationships between them (see Table 19). As an alternative notation to class diagrams, domain models can also be depicted as a collection of Class Responsibility Collaborator (CRC) cards, a slim UML class diagram, or even a slim data model (Ambler 2004b). A domain model based on a CRC model is a collection of standard index cards (see Figure 30). CRC cards were originally introduced as a technique for teaching object-oriented concepts, but they have also been used successfully in XP, for example as story cards (Beck 1999). In domain modelling, a class represents a collection of similar objects, a responsibility is something that a class knows or does, and a collaborator is another class that a class interacts with to fulfil its responsibilities (Ambler 2004b).

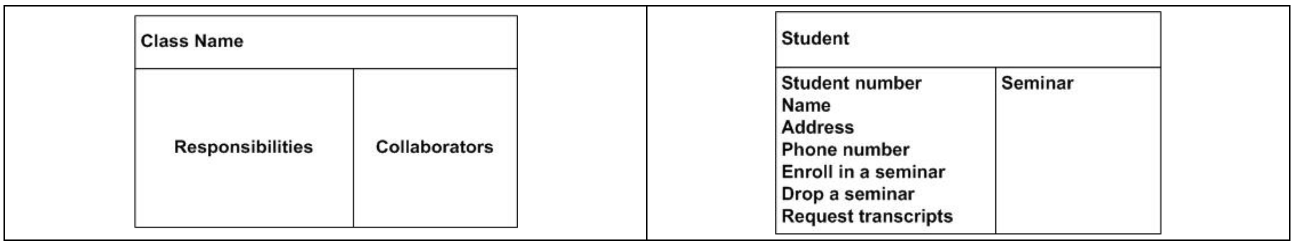


Figure 30: Example of CRC cards (Ambler 2004b)

Popular applications

Domain modelling is popular due to business modelling in RUP. Here, business modelling means the business context, i.e. the scope, of the system. Common modelling activities include the development of (1) a context model (often a data-flow diagram) showing how the system fits into its overall environment, (2) a high-level business requirements model (often an essential use-case model), (3) a glossary defining critical business terms, (4) a domain model (often a class diagram or data diagram) depicting major business classes or entities and (5) a business-process model (often a data-flow diagram or activity diagram) depicting a high-level overview of the business process to be supported by the system.

Table 19: Domain models in brief

Primary use	Capture the most important types of objects in the context of the business
Modelling technique / form	Class diagram; CRC card; one class for each object in the problem-domain
Advantages	Good overview of business objects in the problem-domain; structured format; can guide programming
Disadvantages	Difficult to read for non-IT people who do not understand UML (especially critical during earlier stages)
Related to	Task modelling (e.g. essential use-cases), data-flow diagrams, activity diagrams

4.1.1.3 Business Visioning (BPM)

In corporate UI specification projects, a general motive for producing high-quality software products comes from business goals. Typical business problems such as low user productivity or high user-training costs can lead to an increased consideration of usability (see Chapter 2). The business vision defines the set of goals and objectives at which the business modelling effort is aimed.

"The good news is, the simple act of reviewing our goals and activities on a daily basis, serves, in and of itself, to ensure we don't forget them - thereby keeping them fresh, clear, specific and at the front of our mind." (Shearstone 2008)

Business visions in UI specification

A business vision for the organization in which a software system is to be deployed is meant to provide a good understanding of what the objectives and change-potentials are. The process of defining a business vision usually takes place throughout the earlier stages of the development lifecycle. Accordingly, a business vision also plays an important role during the up-front stages of the UI specification supply chain. The potential contribution of a business vision for an interactive UI specification is the clarity regarding the future of the business, and how this future can be best supported by a usable and enjoyable interactive UI. Accordingly, a business vision is often a very emotional artefact about passion and dreams, without any formal restriction.

Table 20: Steps in developing a business vision (Rational Software Corporation 2008)

Question	**Result**
How can we do things differently?	Ideas about which business use-cases to change, and the kinds of changes that are wanted.
How will it work?	Ideas and suggestions about changing use-cases, workflows, or technology
How well will it work?	New performance measures and metrics for the business use-case
What things must go well?	Critical success factors, such as people, technology, and products
What things might not go well?	Risk factors and potential barriers to the implementation of the business vision, such as resource-allocation, the organizational, cultural, technical and product factors, markets and environments, or costs

A business-vision document encapsulates very high-level objectives of the business modelling effort. It communicates the fundamental issues related to the software and UI development project and is a measure and benchmark against which all results can be validated. The business-vision document will be read by managers, funding authorities, business workers in business modelling, and developers in general. It is created early in the process, and is used as a basis for modelling business cases (see Chapter 4.1.3). The development team has to make sure that the business vision incorporates input from all concerned stakeholders. In all, a business vision can cover the following aspects (Rational Software Corporation 2008):

- Names and descriptions of new or changed business use-cases.
- An overview and brief descriptions of the future business use-cases, emphasizing how they differ from the current ones. These descriptions are to stimulate discussion and they should present objectives in clear terms.
- Measurable properties and goals for each business use-case, such as cost, quality, lifecycle, lead-time, and customer satisfaction. Each goal should be traceable to the business strategy and say how it supports that strategy.
- A specification of the technologies that will support the business use-cases, with special emphasis on information technology.
- A list of possible future scenarios. As much as possible, the specification should predict how the business use-cases will have to change.
- A list of critical success factors; that is, factors critical for the successful implementation of the business vision.

For developing a business vision, it is necessary to define a common vocabulary using the most common terms and expressions in the corporate problem-domain. In this way, textual descriptions remain consistent and misunderstandings among project members about the use and meaning of terms are avoided. Thus, the business vision gains measurable and realistic goals, which in turn will highlight any changes that are necessary, and where and why (see Table 20). With regards to the role of strategy in designing interactive systems, the business vision helps to align the actions of actors with the overall mission of the UI development throughout the specification lifecycle (see Table 21).

Components of a business vision

(Collins & Porras 1996) provide a framework for understanding business vision and articulating it in a mission statement. The mission statement communicates the firm's core ideology and visionary goals, generally consisting of the following three components (see Figure 31): (1) core values to which the company is committed, (2) core purpose of the company, (3) visionary goals that the company will pursue to fulfil its mission. In the context of UI specification, the three ingredients of a business vision can be adapted and focused on the objective of creating interactive systems with a high UI quality.

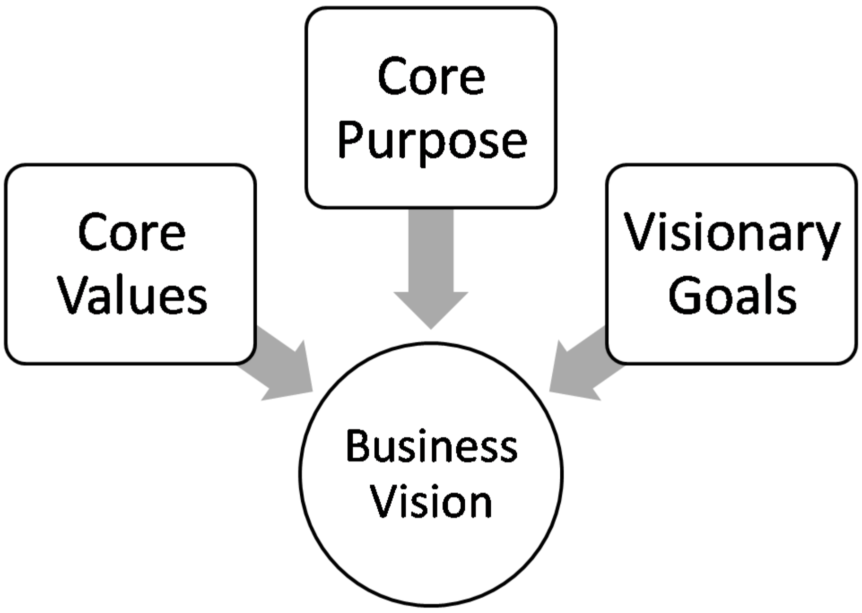


Figure 31: The influencing factors of a business vision; based on (Collins & Porras 1996)

Core Values

The core values are central to the company and reflect the deeply held values of the organization. One way to determine whether a value is a core value is to ask whether it would continue to be supported if circumstances changed and caused it to be seen as a liability. Hence, core values are an axiomatic philosophical backbone. (Collins & Porras 1996) provide a few examples of core values, such as excellent customer service, pioneering technology, creativity, integrity, and social responsibility. Consequently, in the context of UI specification, core values can relate to the quality of software systems. An example would be a commitment to being technologically up-to-date in order to be able to provide outstanding features through a highly usable and emotionally pleasing UI. This could, for example, also include a pledge to prefer innovative means of interacting with UIs to more conservative ones.

Core Purpose

The core purpose is the reason why the organization exists. This core purpose is an idealistic ‘reason for being’ and is expressed in a carefully formulated mission statement, which helps and directs people to meet business goals. During UI specification activities, the core purpose can significantly influence design thinking and it helps to make sure the software product is aligned with corporate goals and style guides, including corporate design and corporate identity. To isolate the core purpose, it is useful to ask ‘why’ in response to mission statements. For example, if a market research firm initially states that its purpose is to provide market research data to its customers, asking ‘why’ leads to the fact that the data is to help customers better understand their markets. The stated ideology should not be a goal or aspiration but rather, it should portray the firm as it really is. Any attempt to state a value that is not already held by the firm's employees is likely not to be taken seriously.

Visionary Goals

Visionary goals are the lofty objectives that the corporate management decides to pursue and that frequently also affect corporate software development. This vision describes some milestone that the firm will reach at an often vaguely defined point in time in the future. In contrast to the core ideology and strategy, visionary goals are rather longer term and therefore more challenging, and it is often uncertain whether they can be achieved. But once such a lofty goal is reached, it needs to be replaced by a new one in order to provide continuous spirit to the organization. A very recent example of a systematic implementation of a visionary goal comes from BMW. With the in-car internet access system ‘BMW ConnectedDrive’ built into the new BMW 7 series, the company continues to follow its visionary goals of making the sedan more interactive. Since the innovative BMW iDrive, the car manufacturer is a developer of cutting-edge car infotainment systems. In 2008, the BMW 7 series is the first car with a fully functional embedded web-browser, which is able to display a web page at standard full size (1280 x 640 pixels).

Figure 32: BMW ConnectedDrive – a successful example of ambitious visionary goals (BMW 2008)

For corporate UI specification projects, visionary goals are not directly related to a concrete design task, but rather make up the surrounding framework. For example, if an organization is aware of certain issues or challenges in the future, it can design a software system such that its architecture, or substantial parts of it, can be used as a basis for the long-term goal. A visionary goal can also influence the choice of UI design with respect to the technology used. If interactivity is going to play a major role, for example for a better product presentation, the specification team might select a rich internet application (e.g. Flash or Silverlight) that can be continually evolved instead of remaining with classic HTML. This, in turn, prepares the ground for UIs that are likely to be more user-friendly and more aesthetically pleasing.

Table 21: Business-vision development in brief

Primary use	Know a company's mission to help direct people to meet a business's goals
Modelling technique / form	Text-based business-vision document; structured into different sections; includes other information, for example about stakeholders, users, markets etc.; one business-vision document incorporates the drive for the project
Advantages	Frank, emotional description of desires, dreams and visions; emotional artefacts, communicates desires and dreams
Disadvantages	Too many different desires may make it difficult to determine clear goals (‘too many cooks spoil the broth’)
Related to	Business-case modelling

Form and structure

As with a strategy in the 5-S framework, which is modelled through needs and objectives, a business vision can likewise be documented in a text-based fashion (see Table 21). The business-vision document, however, differs from a scenario-like notation. As suggested by the (Rational Software Corporation 2008), a business-vision document should be well structured in the following sections: (1) introduction (purpose, scope, definitions of terms, references, overview), (2) positioning (business opportunity, problem statement, product position statement), (3) stakeholder and customer descriptions (e.g. market demographics, stakeholder profiles, customer profiles, key stakeholder and customer needs), (4) business modelling objectives, (5) constraints, (6) quality ranges, (7) precedence and priority, (8) other requirements (e.g. system requirements, performance requirements, usability requirements).

4.1.1.4 The common denominator for problem-domain modelling

Determining the right format

As discussed, understanding the problem-domain can have many facets. With regards to UI specification, the disciplines of SE, IxD and BPM provide some meaningful methods for making the context and environment of a UI specification project more tangible. With domain models (SE), experts can build class diagrams that visualize the objects relevant to the development of the UI. This can include a variety of objects, ranging from business objects to stakeholders or technical resources. But although the class diagrams provide a good overview of the participating entities, modelling in UML is difficult for many stakeholders, especially in corporate UI specification work (see Chapter 3). For problem-domain modelling, the means provided by IxD and BPM are likely to be much more appropriate in most project settings. Both the problem scenario (IxD) and the business-vision document (BPM) depend on a textual description of issues, goals and purposes that drive the UI development process.

Early-stage text-based artefacts

As outlined in the previous chapter, text-based artefacts need to be substituted by more pictorial, diagrammatic and unambiguous forms of UI specification. With regards to the proposed structure of interactive UI specifications and the 5-S approach, however, it is intended to base the specification process on frank and unconstrained story-telling about preconditions and goals. At this stage, the medium of text is still appropriate and is, in fact, especially useful for recording issues and feelings. As outlined in this chapter on problem-domain modelling, capturing emotions is essential for documenting business vision.

Problem-scenarios as common denominator

As a result of these findings, the format of problem scenarios is proposed as a common denominator. As with scenarios in general, they can be easily adapted to reflect the business vision that is supposed to drive the UI specification process. For this purpose, the scenario could also be written following a particular structure in order to ease its readability. Naturally, problem-domain modelling is tightly meshed with many other modelling activities along the UI specification process. Accordingly, a problem scenario sometimes will also describe end-users and their tasks in order to draw a more detailed picture of the whys and wherefores of the project. As mentioned at the beginning of Chapter 4, the problem scenario and its important mission statements can be iteratively amended with additional information. Although there will be more specific means for documenting this information - user or task models, for example - it is the absolute objective of interactive UI specifications to keep related and interdependent artefacts consistent. Because the interactive UI specification will be forwarded to third parties, and it is likely that people who were not part of the specification team will drill down to the lower elements of it, consistency is crucial in order to maintain traceability and transparency. It is therefore in everyone's interest that early-stage problem scenarios are continuously refined and kept up to date.

4.1.2 User Modelling: Methods and Artefacts

Know your users

The precondition for designing UIs for interactive systems is the development of a user model. ‘Knowing the user’ is therefore an important and critical dimension of UI specification. There is no perfect user (see Figure 33) and users’ needs, goals and capabilities have to be considered. The user model allows several relevant aspects to be taken into account (Balzert 2000), for example individual user preferences and capabilities, the user’s reaction to different usage problems and system responses that are adapted to the individual user.

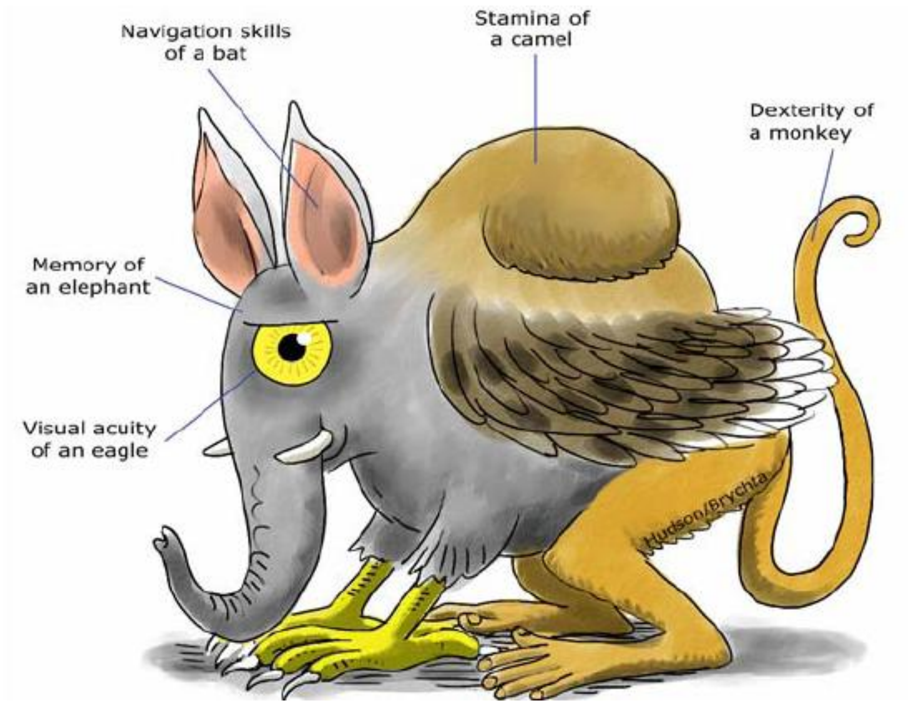


Figure 33: The view of many developers of the ‘perfect user’, from (Hudson 2003)

A variety of advantages for the end-user emerge if software development in general, and UI development in particular, creates a user model. Such systems are supposed to support the user with contextual help, adaptive dialogue flow or user-specific access to features, or by preventing the monotony that could harm effectiveness and efficiency. All three disciplines - SE, IxD and BPM - provide means to understand and model the user of interactive software systems (see Table 22) in order to increase software quality. It is, however, no surprise that most techniques for user modelling come from the field of IxD.

Table 22: User models contributed from different disciplines, derived from Chapters 2 + 3

Contributing Discipline	**User Model**
Software Engineering	User Stories (Beck 1999), User Profiles
Usability Engineering	User Profiles (Mayhew 1999), User/Problem Scenario (Rosson & Carroll 2002), User Roles & Role Maps (Constantine & Lockwood 1999b), Personas (Cooper 1995; Constantine & Lockwood 1999b; Cooper et al. 2007)
Business-Process Modelling	Personas (Holtzblatt 2002), Class Diagram/Stakeholder View (Holt 2005)
Common denominator	User Scenarios, Personas, Role Maps (Constantine & Lockwood 1999b)

In order to identify a common denominator for user modelling, the pros and cons of a variety of techniques are discussed in the following pages. The different techniques, i.e. models and notations, come from the contributing disciplines discussed in the previous chapters of this thesis. At the end of section 4.1.2, personas and role maps are shown to be the appropriate means of a common course of action in interdisciplinary UI specification processes.

4.1.2.1 User Stories (SE)

User stories

In SE, user stories are used to create time estimates for the release-planning meeting in agile projects that travel along a XP lifecycle. XP introduced the practice of expressing requirements in the form of user stories, short descriptions of functionality - told from the perspective of a user - that are valuable to either a user of the software or the purchaser of the software (see Table 23).

Table 23: User stories in brief; based on (Wells 1999b)

Primary use	Release planning, effort/risk estimation; describe functionality
Modelling technique / form	Written text; 80-100 stories per project
Advantages	Low level; just enough for estimation; emphasize verbal communication; low detail first, more detailed later; terminology of the user
Disadvantages	Stakeholders interviewed are not necessarily the end-users;
Related to	Use-cases, user roles

User stories in XP and agile UE

During a short requirements analysis called the exploration phase in XP, the development team and the (on-site)customer write user stories in an attempt to describe user needs and roles. They are used instead of a large requirements document and are developed using simple index cards (see Figure 34). User stories are written by customers as things that the system needs to do for them. The stories should avoid details of specific technology and be focused on user needs and benefits (Wells 1999b). A user story is in the format of about three sentences of text written in the customer's terminology. User stories can also drive the creation of acceptance tests in XP or usability tests in agile UE lifecycles (Memmel et al. 2007e; Memmel et al. 2007f).

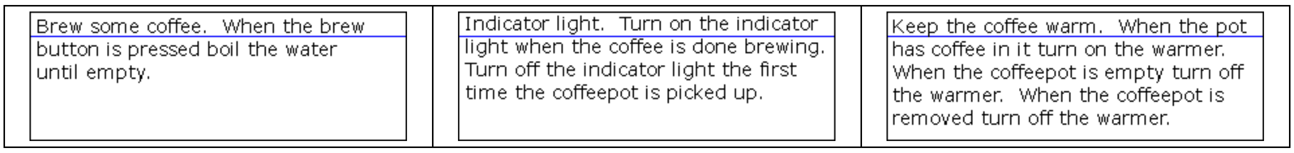
Card, conversation, and confirmation

User stories have three critical aspects. Because user story descriptions are traditionally handwritten on note cards, (Jeffries 2001) calls these aspects 'card', 'conversation', and 'confirmation':

- Card. User stories are written on cards (see Figure 34). The card does not contain all the information that makes up the requirement. Instead, the card has just enough text to identify the requirement, and to remind everyone what the story is (Jeffries 2001).
- Conversation. The requirement itself is communicated from customer to programmers through conversation: an exchange of thoughts, opinions, and feelings. The conversation is largely verbal, but is often supplemented with documents (Jeffries 2001).
- Confirmation. No matter how much discussion or documentation is generated, it is just key aspects of the user story that add confirmation to what kind of software product is sorely needed (Jeffries 2001).

Advantages of user stories

Written language is often very imprecise and ambiguous. Consequently, there is no guarantee that stakeholders and developers will interpret a statement in the same way. Because they do not contain all of the relevant information, user stories emphasize verbal communication in order to overcome this limitation (AM: Model to communicate). And user stories can be used in project planning because they help to estimate how difficult or time-consuming it will be to develop them (see Table 23). User stories should only provide enough detail to make a reasonably low risk estimate of how long the story will take to implement. Another advantage is that user stories encourage the team to defer collecting details. An initially incomplete story can be written and then replaced by more detailed stories once it becomes important to have the details. This technique makes user stories ideal for time–constrained projects. A team can very quickly write a few dozen stories to give them an overall feel for the system.


Brew some coffee. When the brew button is pressed boil the water until empty.

Indicator light. Turn on the indicator light when the coffee is done brewing. Turn off the indicator light the first time the coffeepot is picked up.

Keep the coffee warm. When the pot has coffee in it turn on the warmer. When the coffeepot is empty turn off the warmer. When the coffeepot is removed turn off the warmer.

Figure 34: Example of user stories for a coffee maker; from (Wells 1999a)

Disadvantages of user stories

On the other hand, iterative refinement is very critical for UI design. Development in small increments may work properly as long as the software is not focused on the UI. With the UI, however, things are quite different (Memmel et al. 2007e). Today, changes to software architecture can have a severe impact on what the user sees and interacts with (see Chapter 3.3.1). Another problem with user stories from an IxD point of view is that the people interviewed need not necessarily be the real users of the eventual software product (see Table 23). XP thus often fails to collect real user data and starts coding based only on assumptions about user needs (Gundelsweiler et al. 2004).

User stories are not use-cases

In the end, it is important to recognize that user stories are not use-cases. One of the most obvious differences between stories and use-cases is their scope. Both are sized to deliver business value, but stories are kept smaller in scope because they are used in scheduling work. Another important difference between use-cases and stories is their longevity. Use-cases are often permanent artefacts that continue to exist as long as the product is under active development or maintenance. Stories, on the other hand, are not intended to outlive the iteration in which they are added to the software. Another difference is that use-cases and stories are written for different purposes. Use-cases are written in a format acceptable to both customers and developers so that each may read, and agree with, the use-case. The purpose of the use-case is to document an agreement between the customer and the development team. Stories, on the other hand, are written to facilitate release and iteration planning, and to serve as placeholders for conversations about the users' detailed needs.

4.1.2.2 User Profiles (UE)

User profiles

Understanding who will use a system and knowing their characteristics is the first step in designing a good UI that meets users' needs. User profiles help to identify and describe the intended users of the system so that the developers can build meaningful usability features into their design. With well-defined user profiles, UI design can be more effective and less time-consuming. As the potential end-users are modelled with the profiles, better design decisions can be made. User profiles therefore contribute to the design of UIs that are appropriate for both user and task (see Table 24).

Table 24: User profiles in brief

Primary use	Understanding users by their key characteristics
Modelling technique / form	Text-based; overview through matrix
Advantages	Very detailed form of user modelling
Disadvantages	Can be vague or imprecise
Related to	User scenarios

A user profile is a *"characterization of a system's target population providing information about the users that is useful in making design decisions. The information* *in a user profile is usually obtained through a questionnaire given to the target users. A typical user profile might explore, for instance, how much experience users have with computing or a particular piece of software, what type of physical limitations they have, how frequently they perform common tasks, or what education or reading level they might have." (Usability First 2008)*

Table 25: A template for documenting a user profile (Rational Software Corporation 2008)

Representative	Who is the stakeholder representative for the project?
Description	Brief description of the stakeholder type
Type	Qualify the stakeholder's expertise, technical background, and degree of sophistication
Responsibilities	List the stakeholder's key responsibilities with regards to the system being developed
Success Criteria	How does the stakeholder define success? How is the stakeholder rewarded?
Involvement	How is the stakeholder involved in the project?
Deliverables	Are there any additional deliverables required by the stakeholder?
Comments / Issues	Problems that interfere with success and any other relevant information go here

Advantages

The idea behind user profiles is to enable the interaction designer to consider specific characteristics of a population of users. As knowing the user is critical, the purpose of a user profile is to establish the needs and requirements of a certain user group and to take them into account during UI design and specification. Once the interaction designer has identified the user groups, it is important to characterize them with regards to the UI. Following (Mayhew 1999), this can be done with the help of some key properties:

- Psychological characteristics (e.g. attitude, motivation).
- Knowledge and experience (e.g. typing skill, task experience).
- Job and task characteristics (e.g. frequency of use, task structure).
- Physical characteristics (e.g. colour blindness).

In Mayhew's UE lifecycle, user profiles should be among the first artefacts created. For the determination of user characteristics, the interaction designer can use data-gathering techniques. With interviews or questionnaires, for example, the most important aspects can be analyzed. From the summarized data, the designer can extract initial high-level conclusions for the UI design. A user-profile matrix for one to several users based, for example, on categories such as those presented in (see Table 25), allows the team to understand more clearly the unique tasks and characteristics of each user type for the system.

Disadvantages of user profiles

On the other hand, writing good user profiles is difficult. *"Many projects conduct a user analysis resulting in a large set of users and their characteristics. This information is grouped in user profiles and is used to evaluate the validity and need for diverse aspects of the project. Unfortunately many of the user profiles (...) are often vague, contradictory and very often of little use to the project team."* (Ghosh 2004), p.5

4.1.2.3 Stakeholder-View Class Diagrams (BPM)

Relationships of stakeholders

In BPM, class diagrams can be used to model the relationships of all stakeholders that contribute to the UI development (see Figure 35). By its nature, a stakeholder view, which is modelled through class diagrams, contains not just the end-users of the software, but also other participants. Stakeholder types can be as divergent as users, departments, and technical developers. To provide usable software products that meet stakeholders' and users' real needs, it is necessary to involve all of the stakeholders. In the realm of IxD, the real end-users of the system must be identified to ensure that the stakeholder community, modelled in the stakeholder-view class diagram, adequately represents them.

Stakeholder templates

In order to be able to model stakeholders in detail, the (Rational Software Corporation 2008) proposes a template to capture important aspects. Such a stakeholder profile (i.e. a user profile, see Table 25) provides the background and justification for why the requirements are needed. The format can be freely chosen and popular means such as user profiles might be feasible. While the stakeholder class diagram provides an overview, the user profiles in table-based format are well suited to the task of adding the details.

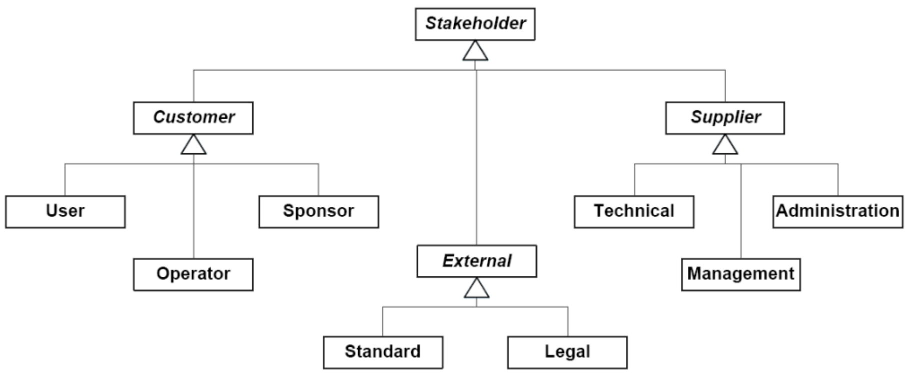


Figure 35: Stakeholder-view class diagram (Holt 2005)

Suitability for UI specification

As with class diagrams for problem-domain-modelling, the stakeholder-view diagram comes with the advantage of an increased overview of the participating stakeholders (see Table 26). But the stakeholder view incorporates other participants besides the real end-users. Although it might therefore not be the perfect fit for focused user modelling in a UI specification process, the diagrammatic format is in many ways superior to text-based descriptions.

Table 26: Stakeholder-view class diagrams in brief; based on (Holt 2005)

Primary use	Modelling the relationships of stakeholders
Modelling technique / form	Class diagrams (UML); text-based template for description of details of each stakeholder; one class diagram for all identified stakeholders and their interrelationships;
Advantages	Provides overview of all stakeholders and their associations;
Disadvantages	Notation might be too abstract for non-technicians due to UML style
Related to	Problem-domain models (same diagrammatic notation); user profiles

4.1.2.3 Personas (UE)

Introducing personas

The personas technique was introduced by Alan Cooper in (Cooper 1999). Personas are fictitious characters that can be used to model different user types who are the intended users of a software product. They are developed from data collected from interviews with users. The fictitious user is expected to support the development of software products that meet the users’ needs and goals.

As a store for critical information, the fictitious user functions as a vehicle for creating empathy and identification. Previously, the terms that described the process of working with fictitious users varied. For example, (Mikkelson & Lee 2000) spoke of user archetypes and (Hasdogan 1996) of user models. But since (Cooper 1999), the term persona is the most widespread.

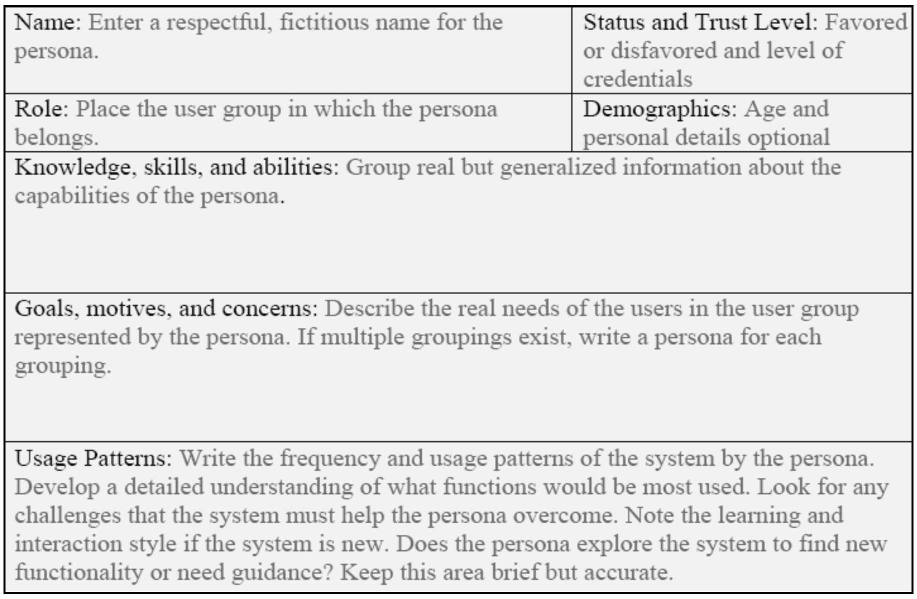
User goals and behaviour

A user persona is usually a representation of the goals and behaviour of a whole user group. Personas are most often used as part of a user-centred design process and can therefore be classified as a contribution from IxD. They are also popular in

SE, especially in the realm of agile usage-centred development (Constantine & Lockwood 1999b).

The format of personas

Personas are intended to guide decisions about a product, such as features, interactions, and visual design. The description of a fictitious user can include information about computer skills, typical activities, and a description of a typical day or week in the user's life. They are usually captured in 1-2 page descriptions that include a variety of information, for example behaviour patterns, goals, skills, attitudes, and environment, with a few fictional personal details to bring the persona to life (see Figure 36 and Figure 37).



Name: Enter a respectful, fictitious name for the persona.	**Status and Trust Level:** Favored or disfavored and level of credentials
Role: Place the user group in which the persona belongs.	**Demographics:** Age and personal details optional
Knowledge, skills, and abilities: Group real but generalized information about the capabilities of the persona.	
Goals, motives, and concerns: Describe the real needs of the users in the user group represented by the persona. If multiple groupings exist, write a persona for each grouping.	
Usage Patterns: Write the frequency and usage patterns of the system by the persona. Develop a detailed understanding of what functions would be most used. Look for any challenges that the system must help the persona overcome. Note the learning and interaction style if the system is new. Does the persona explore the system to find new functionality or need guidance? Keep this area brief but accurate.	

Figure 36: The Microsoft agile-personas template; from (Miller & Williams 2006)

Bob is 52 years old and works as a mechanic with an organisation offering road service to customers when their car breaks down. He has worked in the job for the past 12 years and knows it well. Many of the younger mechanics ask Bob for advice when they meet up in the depot as he always knows the answer to tricky mechanical problems. Bob likes sharing his knowledge with the younger guys, as it makes him feel a valued part of the team.

Bob works rolling day and night shifts and spends his shifts attending breakdowns and lockouts (when customers lock their keys in the car). About 20% of the jobs he attends are complex and he occasionally needs to refer to his standard issue manuals. Bob tries to avoid using the manuals in front of customers as he thinks it gives the impression he doesn't know what he's doing.

Bob has seen many changes over the years with the company and has tried his best to move with the times. However he found it a bit daunting when a new computer was installed in his van several years ago, and now he has heard rumours that the computer is going to be upgraded to one with a bigger screen that's meant to be faster and better.

Bob's been told that he will be able to access the intranet on the new computer. He has heard about the intranet and saw once in an early version on his manager's computer. He wonders if he will be able to find out want's going on in the company more easily, especially as customers' seem to know more about the latest company news than he does when he turns up at a job. This can be embarrassing and has been a source of frustration for Bob throughout his time with the company.

Bob wonders if he will be able to cope with the new computer system. He doesn't mind asking his grandchildren for help when he wants to send an email to his brother overseas, but asking the guys at work for help is another story.

Figure 37: 'Bob': a personas example from (Calabria 2004)

In most cases, a persona also covers the user's fears and aspirations. In contrast to scenarios, personas enhance engagement and reality (Mikkelson & Lee 2000; Grudin & Pruitt 2002). For example, posters with photographs of the personas and their information can be displayed in places frequented by the team (Grudin & Pruitt 2002).

Advantages of personas

The use of personas comes with several advantages (see Table 27). They give abstract data a personal human face. Interaction designers are then able to get in touch with the real person behind the data more easily. Personas therefore support the design team in brainstorming sessions, use-case specification, and feature definition. According to (Cooper 1999; Cooper 2004), personas therefore make several contributions to the design of interactive systems:

- Personas help team members in sharing a specific and consistent understanding of various audience groups. Data about the groups can be put in a proper context and can be understood and remembered in coherent stories.
- With personas, team members' solutions can be guided by how well they meet the needs of individual user personas. Features can be prioritized on the basis of how well they address the needs of one or more personas.
- Personas give user needs a human appearance and can increase empathy with the people represented by the demographics.
- Due to their usable form, personas are easy to communicate and discuss. All stakeholders can absorb user-related data in a understandable format (Grudin & Pruitt 2002; Pruitt & Adlin 2006). In this way, personas also help prevent design pitfalls that may otherwise be overlooked.
- Personas help to avoid the 'the elastic user' phenomenon (Cooper et al. 2007), by which is meant that, while making product decisions, different stakeholders may define the end-user according to their own individual perceptions.
- Personas also help to prevent a self-referential design process, which happens when the design team unconsciously integrates subjective mental models into the user model.
- Personas provide help for designers in keeping the focus on the target users and in having a shared understanding of the users in terms of their goals, capabilities and contexts.

On the whole, personas are a medium for communication that supports the engineering team in creating a focus on users and work contexts. They help to determine complete user models even if only partial knowledge about end-users is available. Personas are defined by personal, practical, and corporate goals and by a relationship to the system to be designed. They take into account how the end-user will most probably feel when using the system.

"The tool of persona use forces one to decide precisely whom one is designing to support. Each persona has a gender, age, race, ethnic, family or cohabitation arrangement, and socio-economic background. This forces existing assumptions about users to the surface and provides an effective avenue for changing or countering them." (Grudin & Pruitt 2002)

Shortcomings of personas

As well as their advantages, personas also have some disadvantages (see Table 27) as outlined by (Nielsen 2004b):

- Some stakeholders may think that the real world can be mapped onto fictitious uses.
- When communicating with individual end-users, some might feel uncomfortable about being compared to fictitious archetypes.
- Some requirements cannot be documented with personas and the stakeholders then have to switch to different means of modelling.

- Unlike user stories or user profiles, for example, personas should not be reused, because every project typically needs its own fictitious users to guide design thinking.
- Personas are difficult and time-consuming to create, and the production of personas requires a well-developed skill-set.

Table 27: Personas in brief

Primary use	Developing fictitious users to focus on target users, their goals, capabilities and contexts
Modelling technique / form	1-2 page descriptions with picture of fictitious end-user; one persona can describe a single user or a group of users, or a user role that summarizes many individuals
Advantages	Give abstract data a human appearance; raise empathy; easily understandable and structured form
Disadvantages	Difficult to create, requires lots of training; reuse not recommended
Related to	Use-cases

4.1.2.4 User Scenarios (UE)

User scenarios

A user-interaction scenario is a story about people and their activities. User scenarios describe one or more users engaged in some meaningful activity with an existing or envisioned system. User scenarios focus on what happens from the perspective of someone using the system. Following (Miller & Williams 2006), a scenario describes the system's behaviour through a sequence of concrete interactions with its users, who are trying to achieve some goal. Scenarios reflect a concrete path or set of steps toward a goal.

"The relationship between the user and the system is characterized in a scenario as follows: The scenario identifies the person as having certain motivations toward the system, describes the actions taken and some reasons why these actions were taken, and characterizes the results in terms of the user's motivations and expectations (Carroll 1995)." (Miller & Williams 2006), p. 3

Advantages and common use

As personas attempt to reach their goals, the scenario records the specific steps that are taken. The combination of the motivations, knowledge, and goals of the personas results in their behaviour. This behaviour is exactly what a scenario is intended to capture (see Table 28). Personas are used to feed the scenarios and tasks deemed important to achieving business goals and evolving the interaction between the user and the UI. The names of the personas are used in the description of the scenario. Because writing narratives often includes much more than just descriptions of the user(s), pure user scenarios are rather rare. They are usually merged with activity or interaction scenarios (see Figure 28)

Table 28: User scenarios in brief

Primary use	Describing the UI behaviour from the perspective of someone who uses the system
Modelling technique / form	Text/narrative; from one page to many pages
Advantages	Narrative descriptions of user-UI interaction, easy to understand and model
Disadvantages	May lack clarity and structure
Related to	Personas; other types of scenarios

Disadvantages

(Mikkelson & Lee 2000) criticized scenarios, as represented by (Karat & Bennett 1991; Carroll 1995; Carroll 2000a; Carroll 2000b; Rosson & Carroll 2002) for example, for lacking clarity and consistency in the descriptions of users. It could therefore make sense to combine scenarios with personas. Once a set of personas is constructed and provided with sets of goals, and once team members have accepted and assimilated them, then meaningful scenarios can be constructed around them (Cooper 1999). In turn, scenarios are less effective when not built on personas (Grudin & Pruitt 2002).

4.1.2.5 User Roles and Role Maps (SE)

Personas and user roles

As described earlier, personas describe fictitious users. Personas are figurative models rather than abstract models. They are constructed to resemble real users, even down to photos and personal information. Personas sound like real people, whom one of the project stakeholders might know personally. In this way, personas can easily encourage stakeholder empathy that, in turn, makes it easier for developers to think like the potential end-user. In contrast, user roles do not resemble real people. User roles are very abstract artefacts that focus on aspects relevant to UI presentation. User roles (Constantine 1996; Constantine & Lockwood 1999b) emerged as an elaboration of the SE construct of actors (Jacobson 1992), which required adaptation to better suit the needs of IxD. User roles describe relationships between users and the systems being used. Compared to personas, user roles are a more technical and formally structured model. The goal in user-role modelling is to capture what is most salient in the most concise form (Constantine 2005).

Table 29: User roles in brief

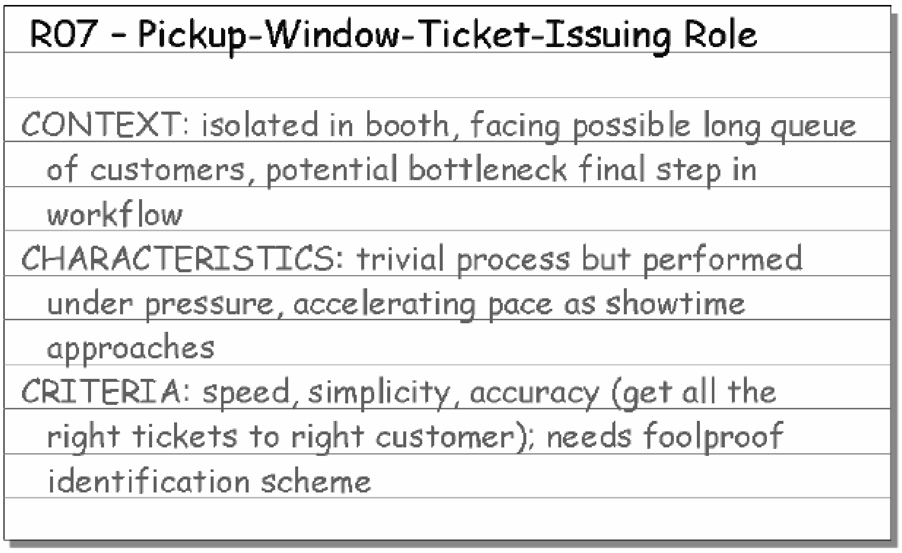
Primary use	Presenting a collection of needs, interests, expectations and behaviours
Modelling technique / form	Abstract artefact; card-based; one role can summarize several personas; three to five relevant people per role
Advantages	More technical, less fictitious than personas; describes context, characteristics, and criteria; encourages strip-down of user roles into small and convenient entities
Disadvantages	Less detailed than personas
Related to	Personas; role maps

Context, characteristics and criteria

A user role is usually a collection of user needs, interests, expectations, and behaviours in relation to a particular UI (see Table 29). Most commonly, a user role is determined by the context in which it is played, the characteristics of its performance, and the criteria that must be met by the design to support successful performance of the role. Context includes the overall responsibilities of the role, and the larger work and environmental context within which it is played. Characteristics refer to typical patterns of interaction, behaviours, and attitudes within the role. Criteria include any special functional facilities needed for effective support of the role, along with design objectives of particular importance to that role. Such criteria for effective support of a role are sometimes referred to as usability or user-experience attributes.

Card-based modelling

As discussed earlier, agile methods play an important part in corporate UI specification due to their ability to support cost-effective development of usable software systems. In this context, usage-centred design (Constantine & Lockwood 1999b) has brought some principles and practices of the agile world into IxD (see Chapter 3). In conjunction with the use of card-based techniques in XP - for example, in the context of user analysis (story cards) or evaluation defect cards (Gundelsweiler et al. 2004) - Constantine proposes to document user roles on cards. With simple index cards (see Figure 38), user roles can be created and managed easily. Hence, user roles are usually described on a single index card and seldom if ever take up more than one side (Constantine 2005). One of the operating principles of card-based modelling is that anything that does not fit on a single index card is too complicated and should be further simplified and condensed. This principle is derived from agile software development and gives good support to the development of a structured understanding of the problem-domain and the end-users. Unlike the situation with personas, brevity means that no attempt is made to create an interesting, engaging, or recognizable user portrait. Following (Constantine 2005), user roles are usually ranked by anticipated frequency and by business importance. The agile modelling technique for making these rankings is to sort the index cards representing the roles into order (Constantine 2002; Constantine & Lockwood 2002). On the basis of the combination of these two potentially different views of priority, a small subset of roles are identified as 'focal' roles. Focal roles serve as a central focus for the rest of the design process, but not to the exclusion of other user roles.



R07 – Pickup-Window-Ticket-Issuing Role

CONTEXT: isolated in booth, facing possible long queue of customers, potential bottleneck final step in workflow
CHARACTERISTICS: trivial process but performed under pressure, accelerating pace as showtime approaches
CRITERIA: speed, simplicity, accuracy (get all the right tickets to right customer); needs foolproof identification scheme

Figure 38: Example of condensed, card-based model of a user role; from (Constantine 2005)

User roles and personas

Another difference in the two approaches is that with personas the ideal is to fully describe a small number of archetypal users. User-role modelling seeks to cover the playing field with a collection of interrelated but distinct descriptions of all the various roles that users can play in relation to the system. In most cases, user roles will outnumber personas, but the personas will be more elaborate.

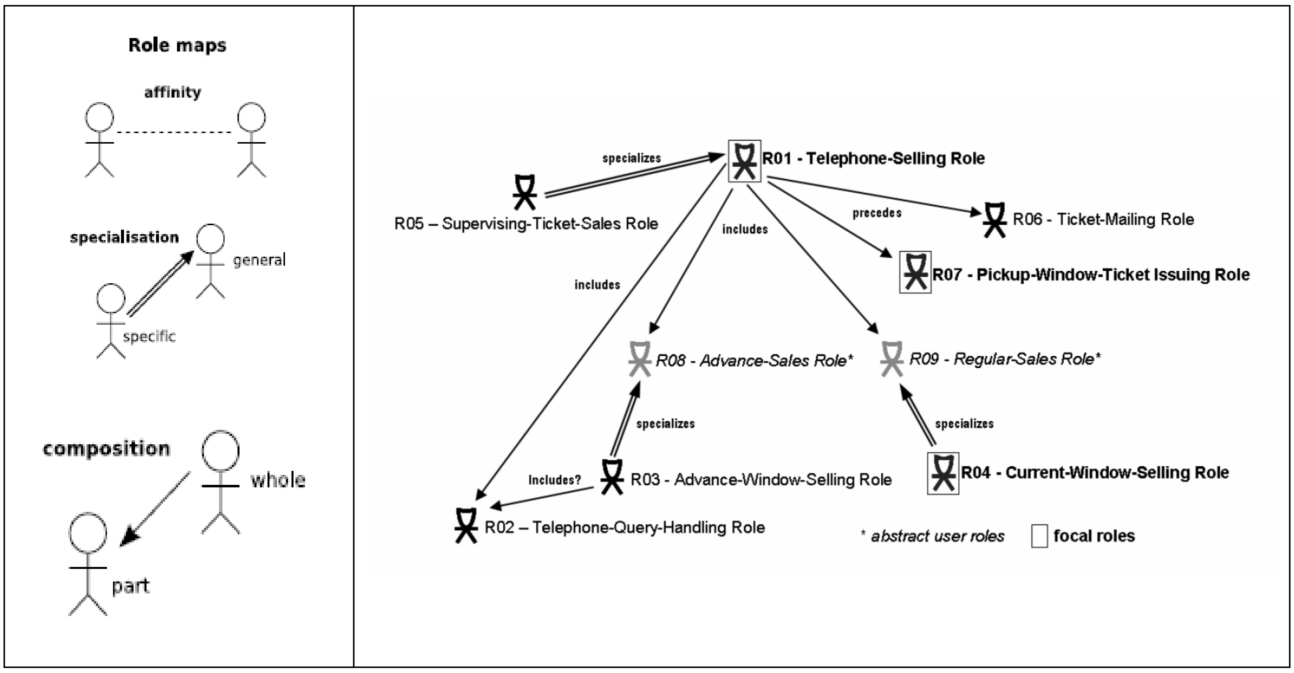

Role maps
affinity
specialisation
general
specific
composition
whole
part
specializes
R01 - Telephone-Selling Role
R05 – Supervising-Ticket-Sales Role
includes
precedes
R06 - Ticket-Mailing Role
R07 - Pickup-Window-Ticket Issuing Role
includes
R08 - Advance-Sales Role*
R09 - Regular-Sales Role*
specializes
specializes
Includes?
R03 - Advance-Window-Selling Role
R04 - Current-Window-Selling Role
R02 – Telephone-Query-Handling Role
* abstract user roles
focal roles

Figure 39: The notation for user-role maps (left); based on (Constantine & Lockwood 1999b); a user-role map (right) exemplified by a ticketing application; from (Constantine 2005)

Role Maps

For a given application, the roles in a complete catalogue of user roles are typically not independent of each other. Some roles, for example, are best thought of as specialized variants of others, and some may be combinations that include two or more other roles. This interdependence makes it possible to map out all the roles without a lot of repetition. For complex problems, the relationships between user roles can be represented diagrammatically in a separate user-role map (see Figure 39). In addition to specialization, affinity or composition, user roles may have a workflow relationship in which one role depends on the prior performance of another.

4.1.2.6 The common denominator for user modelling

Personas and scenarios

(Grudin & Pruitt 2002; Pruitt & Grudin 2003; Pruitt & Adlin 2006) suggest using personas in conjunction with scenarios as a compromise between the actor (SE) and the on-site customer (XP). User involvement is one of the most important elements for UI specification. However, in many situations the real end-users are not available and (Miller & Williams 2006) found that personas lead to a more comprehensive understanding of the end-user.

"The powerful psychological identification with and engagement between an analyst and a persona can bring about the inclusion of complex and realistic social and political aspects of the persona within a scenario". (Miller & Williams 2006), p. 1

Personas and user roles

In UI specification, the stakeholders will usually identify several roles that interact with a system. A role typically stands for many different types of users. For example, some people in the role may be more experienced users, whole others may only have a superficial knowledge of the system. Thus, just looking at roles does not provide enough information to make key decisions that could make the system more appealing to its users. Consequently, once these roles have been determined, experts can further refine them by providing personas (Miller & Williams 2006). (Johnson 2003) explained that there are at least three to five types of relevant people per user role. The number of personas that has to be produced can therefore be determined by the number of user roles. To keep it simpler, several characters can be merged into one persona, which then acts as a proxy for a group of users. The persona, therefore, provides a means to talk and reason about the group through the characteristics of one fictional individual, the persona (Miller & Williams 2006). (Constantine 2005) gives advice on the context in which personas and user roles should be developed simultaneously. The verisimilitude of personas is especially caused by their conversion of abstract information into the more tangible. In corporate UI specification processes, most stakeholders do not have the capability to think abstractly. They prefer to make design concepts visible as soon as possible. Hence, stakeholders who are more comfortable in a world of concrete and recognizable objects than in one of abstractions will prefer personas to user roles. Naturally, creating sound and consistent personas is a laborious task and requires more resources than developing abstract user models on index cards.

Bridging the gaps

All things considered, where empathetic identification with users is possible, user-role models should be accompanied by personas to draw a more detailed and living picture of the end-users. But if time is short, resources are limited, or stakeholders are able to work with abstract descriptions, user roles are usually enough. In addition, many software developers prefer unemotional work artefacts and will therefore prefer more abstract descriptions. Hence, role maps can function as a common ground for user modelling, and personas provide additional details on demand. With regards to BPM, (Holtzblatt 2002) has added a business view to personas to include business goals into design efforts. Moreover, the diagrammatic notation of role maps aligns with the advantages of the stakeholder-view class diagram, popular in BPM. However, with regards to UI specification, role maps benefit from a focus on the end-users of a software system but give less attention to other stakeholders. Consequently, personas are appropriate as a common denominator for SE, IxD and BPM.

"We all have a problem with seeing things from another person's perspective. Personas are one way of redressing the balance. (...) Personas are used to represent important groups of users, but in a much more tangible and accessible way than user profiles. Most projects have only a handful of personas." (Hudson 2003)

In complex projects with many different users, personas should be accompanied by role maps to outline relationships and dependencies. Because of the tight relationship of personas to scenarios, the problem scenarios produced for understanding the problem-domain can also be used to additionally describe certain users in narratives. This, however, can be seen as an optional activity and therefore scenarios are not essential for user modelling. Moreover, with regards to the stakeholders, the process of UI specification is supposed to become more visually based and less text-based, so that scenarios would not be a suitable choice during later stages.

Actors, roles and the subsequent UI specification process

For UI modelling, personas and role maps are the appropriate choice for interactive UI specifications. Together, they are a compatible contribution from SE and IxD. In usage-centred design (Constantine & Lockwood 1999b), user roles depend on actors and, subsequent to user modelling, propel the task-modelling phase. For interdisciplinary UI specification, the concept of a user role is almost the same as the notion of the actor as introduced by (Jacobson 1992), with one important difference. (Constantine & Lockwood 1999b) explain that in the original definition, actor included non-human entities such as hardware or software. For designing interactive systems, the focus is on humans that interact with the UI. Accordingly, only human actors are at the centre of interactive UI specifications.

4.1.3 Task Modelling: Methods and Artefacts

"Task analysis is used mainly to investigate an existing situation, not envision new systems or devices. It is used to analyze the underlying rationale and purpose of what people are doing: what are they trying to achieve, why are they trying to achieve it, and how are they going about it?" (Preece et al. 2002), p. 231

One of the main targets of task-oriented approaches to UI development is to support the user in executing his daily assignment. For this purpose, a variety of different notations exists (see Table 30). However, all notations must allow user tasks to be specified in a hierarchical and sequential way. The sequential character of a user task points to the steps that must be traversed in order to reach a certain goal. Usually, a sequence can be subdivided into smaller parts. The hierarchical aspect of a task also takes the limitations of human cognition into account and therefore aims to structure a task into specific parts.

With regards to task modelling, the notations of usage-centred design (Constantine & Lockwood 1999b) are difficult to assign to just one discipline (see Table 30). Because essential use-cases are frequently referenced to in IxD literature such as (Sharp et al. 2007), we assign them to the models of UE. Task maps, in contrast, are solely used in (Constantine & Lockwood 1999b), which is more an SE method open to UE.

Table 30: Task models contributed from different disciplines, derived from Chapters 2 + 3

Contributing Discipline	**Task Model**
Software Engineering	Usage scenarios (Ambler 2006d); use-cases (Jacobson 1992); use-case diagrams (Jacobson 1992); task maps (Constantine & Lockwood 1999b)
Usability Engineering	Activity scenarios (Rosson & Carroll 2002); essential use-cases (Constantine & Lockwood 1999b)
Business-Process Modelling	Business cases; business use-case diagrams (Scheer 1998)

4.1.3.1 Usage Scenarios (SE)

Usage scenarios describe a real-world example of how people interact with a system (see Table 31). Accordingly, they are often used in combination with personas. They describe the steps, events, and/or actions that occur during the interaction. Usage scenarios can be very detailed. They describe how a user works with the software. Usage scenarios are applied in several development processes, often in different ways.

Table 31: A usage scenario example from (Ambler 2006d); slightly shortened

Scenario: A successful withdrawal attempt at an automated teller machine (ATM).	
1	John Smith presses the 'Withdraw Funds' button
2	The ATM displays the preset withdrawal amounts ($20, $40, …)
3	John chooses the option to specify the amount of the withdrawal
4	The ATM displays an input field for the withdrawal amount
5	John indicates that he wishes to withdraw $50 dollars
6	The ATM displays a list of John's accounts, a checking and two savings accounts
7	John chooses his checking account
8	The ATM verifies that the amount may be withdrawn from his account
9	The ATM verifies that there is at least $50 available to be disbursed from the machine
10	The ATM debits John's account by $50
11	The ATM disburses $50 in cash

Scenarios: between use-cases and sequence diagrams

In unified process lifecycles such as RUP or the Agile Unified Process (AUP) presented by (Ambler & Jeffries 2002; Ambler 2004b), usage scenarios are used to help move from use-cases (see Chapter 4.1.3.3) to sequence diagrams (see Chapter 4.1.4.4). The scenario narrates a single path though a use-case. Differing from scenarios, the use-case would include, and discuss, alternative paths through a task. To describe all possible paths through an activity, the development team therefore has to write several usage scenarios. Later on, a sequence diagram would determine the implementation logic for each scenario (Ambler 2006d). In many projects, scenarios are unlikely to be kept as artefacts if the developers follow the agile practice of discarding temporary models. In fact, although scenarios can be one of the initial models, more specialized forms of notation should be preferred for specifying users and tasks.

Rules for scenario writing

In order to make this process of transformation as efficient as possible, scenario writing should follow some simple rules: (1) write at least one scenario for every stakeholder, (2) analyze at least one, preferably more claims (see Chapter 4.1.1.1) from each scenario, (3) write multiple scenarios for stakeholders who have many tasks.

Scenarios and use-cases

Due to the nature of scenarios, there are several differences between use-cases and scenarios. The most important difference with regards to user modelling is that the use-case typically refers to generic actors, whereas the scenario refers to examples of the actors. This makes scenario writing more personalized, which in turn increases the understandability of user needs and requirements. Scenario descriptions can be very useful in managing design trade-offs, as they shed light on the detailed interactive behaviour of the UI (see Table 32).

At the same time, scenarios are more flexible because they keep the design space open for further change. The advantage of personalization and empathy is also fundamental to the development of personas. It is therefore obvious that usage scenarios could be beneficially linked to persona models. As described above, the formal difference between scenarios and use-cases is the variety covered. Usage scenarios describe a single path of logic whereas use-cases typically describe several paths (the basic course plus any appropriate alternative paths).

Table 32: Usage scenarios in brief

Primary use	Describing a path through a use-case; describing interactive behaviour
Modelling technique / form	Narrative text; table-based notation; one scenario for each path through a use-case
Advantages	Easy to produce; can be developed by every stakeholder; increases understanding of a use-case through narration
Disadvantages	Not very precise; transformation into structured form required for development
Related to	Personas, use-cases, sequence diagrams

4.1.3.2 Activity Scenarios (UE)

Activity design

Activity scenarios are the main artefacts used during activity design in a scenario-based UE (Rosson & Carroll 2002). Accordingly, they follow the development of problem scenarios earlier in the process (see Chapter 4.1.1.1). If designers focus too much on users' current practices (discussed with problem scenarios, for example), they may miss important opportunities for innovation. Writing activity scenarios is therefore the first phase in the scenario-based UE of design reasoning, in which the problems and opportunities of current practice are transformed into new ways of behaving (see Table 33). Apart from that, activity scenarios are similar to other kinds of scenarios, and they can therefore be described very briefly.

Table 33: Activity scenarios in brief

Primary use	Introducing concrete ideas about new functionality
Modelling technique / form	Narration; text-based
Advantages	Easy to develop; easy to understand; motivate innovative thinking
Disadvantages	Difficult to analyze for task modelling; may be very unstructured
Related to	Other types of scenarios

Moving from problem scenarios to activity design is the point when the project team first introduces concrete ideas about new functionality and new ways of thinking about users' needs and how to meet them. During activity design, scenarios and claims serve as a flexible and inexpensive medium for imagination and discussion. The focus on realistic situations helps designers to generate specific ideas. Problem scenarios are transformed into activity-design scenarios through a combined process of brainstorming about design ideas, reasoning from previous claims and working through the general concerns of activity design. Scenarios are evaluated and refined by identifying key features and their consequences for use via claims analysis.

Point of view analysis

By writing activity scenarios, designers identify objects that play an important role in a scenario and then construct a point of view (POV) for each object. Each POV is created by doing a scenario walk-through from the perspective of a hypothetical software object. POV analysis moves the project in the direction of software design, and it can therefore be a basis of discussion between usability and software personnel.

4.1.3.3 Use-cases (SE)

Use-cases

While interaction designers developed scenario-based methods (see Chapter 4.1.3.1), software engineers developed use-cases. Use-cases and scenarios are related, because a scenario story is usually the detailed narrative of a specific path through a use-case. The use-case, in turn, is more general and often also includes a variety of alternatives in user-system interaction. Both notations are known in both communities, however, and even business modelling employs a use-case notation to model business cases (see Chapter 4.1.3.8).

Use-cases model the course of events that can take place provided there is some user input. The case thus specifies all possible interactions between the user and the system. Use-cases were established by (Jacobson 1992) and proved to be successful during software specification for describing functional aspects of the system. (Jacobson 1992) employed use-cases as a means to identify the necessary classes of a program.

"To develop a use case, first identify the actors, i.e., the people or the other systems that will be interacting with the system under development. Then examine these actors and identify their user goal or goals in using the system. Each of these will be a use case." (Preece et al. 2002), p. 230

Since Jacobson originated use-case modelling, many others have contributed to improving this technique, including Alistair Cockburn (Cockburn 2000; Cockburn 2008) and Scott Ambler (Ambler & Jeffries 2002; Ambler 2004b; Ambler 2004a; Ambler 2006a). Their approaches will be discussed in the following pages.

Informal and formal use-cases

A use-case is a sequence of actions that provides a measurable value to an actor. Another way to look at it is that a use-case describes a way in which a real-world actor interacts with the system. In a system use-case, high-level implementation decisions are included. System use-cases (Ambler 2006a) can be written in both an informal manner (see Table 34) and a formal manner (see Table 35). A formal use-case is captured in a document based on a long template with fields for various sections. This is the most common understanding of the meaning of a use-case in SE.

Table 34: Enrol-in-seminar example of an informal system use-case; from (Ambler 2006a)

Name: Enrol in Seminar (Identifier: UC 17) **Basic Course of Action:**	
-	Student inputs name and student number
-	System verifies the student is eligible to enrol in seminars. If not eligible then the student is informed and use-case ends.
-	System displays list of available seminars.
-	Student chooses a seminar or decides not to enrol at all.
-	System validates that the student is eligible to enrol in the chosen seminar. If not eligible, the student is asked to choose another.
-	System validates that the seminar fits into the student's schedule.
-	System calculates and displays fees
-	Student verifies the cost and indicates a wish either to enrol or not.
-	System enrols the student in the seminar and bills the student for it.
-	The system prints enrolment receipt.

Informal use-cases

An informal use-case is also sometimes called an essential use-case. But as this nomenclature is also used by Constantine's use-case notation (see Chapter 4.1.3.5), it is not used to refer to the SE-style use-case. The informal notation differs from the formal one in terms of its very brief, bullet-point style that contains just enough information to communicate the idea (Ambler 2006a).

Basic elements of a use-case

The use-case name provides a unique identifier for the use-case. It should describe a goal and it should be sufficient for every stakeholder and end-user to understand what the use-case is about. A use-case might also have a summary section. This is used to capture the essence of the use-case in order to provide a quick overview. The purpose of each use-case is to discuss a primary use-case scenario, which is the most typical course of events. In the informal version of a use-case, the main steps are not necessarily numbered, whereas in the formal version they are (Ambler 2006a).

Table 35: Enrol-in-Seminar as a formal system use-case; from (Ambler 2006a)

Name: Enrol in Seminar (Identifier: UC 17) Description: enrol an existing student in a seminar for which they are eligible.	
Preconditions: the student is registered at the University.	
Postconditions: the student will be enrolled in the course they want if they are eligible and a place is available.	
Basic Course of Action:	
1	The use-case begins when a student wants to enrol in a seminar.
2	The student inputs their name and student number into the system via UI23 Security Login Screen.
3	The system verifies that the student is eligible to enrol in seminars at the university in accordance with business rule BR129 Determine Eligibility to Enrol.
4	The system displays UI32 Seminar Selection Screen, which indicates the list of available seminars.
5	The student indicates the seminar in which they want to enrol.
6	The system validates that the student is eligible to enrol in the seminar in accordance with business rule BR130 Determine Student Eligibility to Enrol in a Seminar.
7	The system validates that the seminar fits into the student's existing schedule in accordance with business rule BR143 Validate Student Seminar Schedule.
8	The system calculates the fees for the seminar based on the fee published in the course catalogue, applicable student fees, and applicable taxes. Apply business rules BR180 Calculate Student Fees and BR45 Calculate Taxes for Seminar.
9	The system displays the fees via UI33 Display Seminar Fees Screen.
10	The system asks the student if they still want to enrol in the seminar.
11	The student indicates that they want to enrol in the seminar.
12	The system enrols the student in the seminar.
13	The system informs the student the enrolment was successful via UI88 Seminar Enrolment Summary Screen.
14	The system bills the student for the seminar in accordance with business rule BR100 Bill Student for Seminar.
15	The system asks the student if they want a printed statement of the enrolment.
16	The student indicates they want a printed statement.
17	The system prints the enrolment statement UI89 Enrolment Summary Report.
18	The use-case ends when the student takes the printed statement.
Alternative Course A: The Student is Not Eligible to Enrol in Seminars.	
A.3	The registrar determines that the student is not eligible to enrol in seminars.
A.4	The registrar informs the student they are not eligible to enrol.
A.5	The use-case ends.
Alternative Course B: The Student Decides Not to Enrol in an Available Seminar	
B.5	The student views the list of seminars and does not see one in which they want to enrol.
B.6	The use-case ends.
Alternative Course C: The Student Does Not Have the Prerequisites	
C.6	The registrar determines that the student is not eligible to enrol in the seminar they chose
C.7	The registrar informs the student they do not have the prerequisites
C.8	The registrar informs the student of the prerequisites they need
C.9	The use-case continues at Step 4 in the basic course of action

Elements of a formal use-case

In a formal use-case, the preconditions section is used to convey any conditions that must be true when a user initiates a use-case. The postconditions, in turn, summarize parts of the system affected after the scenario is complete. The alternative courses are the variations on the main path (see Table 35). The alternative paths also make use of the numbering of the basic course of events to show at which point they differ from the basic scenario. A formal use-case model can also refer to business rules. Such rules determine how an organization conducts its business with regard to a use-case. Business rules may be specific to a use-case or apply across all the use-cases, and indeed all the business.

Table 36: A use-case following the use-case template of (Cockburn 2000; Cockburn 2008)

USE-CASE	**Buy Goods**	
Goal in context	Buyer issues request directly to our company, expects goods to be shipped and billed.	
Scope & level	Company, summary	
Preconditions	We know buyer, their address, etc.	
Success end condition	Buyer has goods; we have money for the goods.	
Failed end condition	We have not sent the goods; buyer has not spent the money.	
Primary, secondary actors	Buyer, any agent (or computer) acting for the customer. Credit card company, bank, shipping service	
Trigger	Purchase request comes in.	
DESCRIPTION	Step	Action
	1	Buyer comes in with a purchase request
	2	Company captures buyer's name, address, requested goods, etc.
	3	Company gives buyer information on goods, prices, delivery dates, etc.
	4	Buyer signs for order.
	5	Company creates order, ships order to buyer.
	6	Company mails invoice to buyer.
	7	Buyers pays invoice.
EXTENSIONS	Step	Branching Action
	3a	Company is out of one of the ordered items: 3a1. Renegotiate order.
	4a	Buyer pays directly with credit card: 4a1. Take payment by credit card (use-case 44)
	7a	Buyer returns goods: 7a. Handle returned goods (use-case 105)
SUB-VARIATIONS		Branching Action
	1	Buyer may use phone in, fax in, use web order form, electronic interchange
	7	Buyer may pay by cash or money order check credit card

Use-case briefs

Not all use-cases are written by filling in a form. Some use-cases are written as unstructured text, called use-case briefs (Cockburn 2001). Use-case briefs differ from user stories in two ways. First, since a use-case brief must still cover the same scope as a use-case, the scope of a use-case brief is usually larger than the scope of a user story. A use-case brief will typically tell more than one story. Second, use-case briefs are intended to live on for the life of a product. User stories, on the other hand, are discarded after use. Finally, use-cases are generally written as the result of an analysis activity, while user stories are written as notes that can be used to initiate analysis conversations.

Use-case briefs answer questions such as 'What are the goals the system under discussion must support'. They are unlikely to answer the questions of 'why' and 'how' a function should be implemented. Another difference is that use-cases and stories are written for different purposes. Use-cases are written in a format acceptable to both customers and developers, so that each may read, and agree with, the use-case. The purpose of the use-case is to document an agreement between the customer and the development team. Stories, on the other hand, are written to facilitate release and iteration planning, and to serve as placeholders for conversations about the users' detailed needs.

Brief, casual and fully dressed

Due to the variety of use-case models, (Cockburn 2000) identifies three degrees of detail in writing use-cases: briefs, casual use-cases and fully dressed use-cases. The use-case brief consists of two to four sentences summarizing the use-case. It fits well into a spreadsheet cell, and allows the other columns in the spreadsheet to record business priority, technical complexity, release number, and other planning information. The casual use-case consists of a few paragraphs of text covering the items mentioned above. The fully dressed use-case features the long template with fields for stakeholders, minimum guarantees, postconditions, business rules, performance constraints, and so on. With formal text sitting in a semi-formal structure, (Cockburn 2002) proposes a middle road between formal and informal use-cases (see Table 36). In Cockburn's approach, use-cases include actors' goals in order to shift attention away from pure functions. If the software supports those goals, the software will yield the greatest business value. Therefore, a use-case is structured into two sections: the sequence of actions when everything goes well, followed by various small sequences describing what happens when the various goals and subgoals are not achieved.

Use-cases of mass destruction

Instead of adopting several modelling techniques and using each one when appropriate, some people assume that use-cases must include all requirements. This approach produces large use-cases that are difficult to understand and manage (Ambler 2004a). A formalized system use-case frequently refers to specific UI components, whereby design issues creep into the use-case description. Because the UI will work differently depending on the implementation technology, the logic of the system use-cases, and thus the flow of the UI dialogue, will also be affected. By referring to other artefacts instead of embedding the information into other use-cases, the risk of creating use-cases of mass destruction (Ambler 2004a) is reduced.

Task cases

For many experts, however, Cockburn's model was still not adequate for UI development. They developed three variants of use-cases, namely feature lists, story cards, and task cases. Larry Constantine pursued the writing of scenario descriptions to help in user interface (UI) design. He found he did not need to record the system's internal processing steps or its interaction with supporting actors. He could manage with just a very simple two-column structure. The left column listed what the user was trying to accomplish at each step. The right column listed the system's responsibility. This two-column table provides enough information to allow people to create a good UI design. He renamed it a task case or essential use-case, to avoid confusion with system specification use-cases. The essential use-case notation is discussed in detail in Chapter 4.1.3.5.

Advantages of use-cases

One of the biggest advantages of writing use-cases is that they are traceable. Because of their notation style and the linking with other use-cases, requirements are related to each other and traceability increases (see Table 37). Informal use-cases in particular are easy to understand and so can be an excellent bridge between software developers and end-users. But there is a learning curve involved in interpreting use-cases correctly. Because use-cases are related to scenarios, however, they also help to tell stories. Such a story or scenario is even more understandable. Because use-cases are concerned with the interactions between the user and the system, they make it possible for interaction designers to become involved.

Problems with use-cases

On the other hand, use-cases are not well suited to easily capturing the non-functional requirements of a system. Although use-cases are good for usage requirements, they are not so good for exploring UI requirements. This is critical for UI specification. It is difficult to determine the level of UI dependency to incorporate in the use-case. While use-case theory suggests that the UI should not be reflected in use-cases, many find it awkward to abstract from this aspect of design as it makes the use-cases difficult to visualize. But use-cases should not be used to describe UI designs for two main reasons, even though the use-case form permits it. Firstly, a use-case is normally intended as a requirements document, and the UI design is a design created by trained people after they have been told what the system should do. Secondly, UI design is an iterative process, at least until the initial prototyping stage narrows the design space and the basic UI architecture is agreed upon. Writing the UI design into use-cases would lead to frequent updates - much more frequent than any project team could cope with. Other forms of description, such as Constantine's task cases or UI prototypes do a better job when working with the UI (Cockburn 2002).

Table 37: Use-cases in brief

Primary use	Modelling the course of events that can take place provided there is some user input
Modelling technique / form	Table-based (in-)formal; using use-case template
Advantages	Add traceability; can be turned into scenario(s) or being derived from one/many
Disadvantages	Might include too much; inappropriate for describing UI-related issues
Related to	Usage scenarios, use-case diagrams, task maps

4.1.3.4 Use-case Diagrams (SE)

The concept of use-case diagrams

Use-case diagrams provide an overview of the usage requirements for a system or UI and are closely related to table-based use-case notations (see Table 38). They are a UML-specific notation for modelling use-cases (see Table 3) and a well-established means of modelling in SE (Jacobson 1992; Ambler & Jeffries 2002; Ambler 2004b; Sommerville 2004) and BPM (Holt 2005).

The elements of the use-case diagram

The most important parts of a use-case diagram are the use-cases, the actors and the associations. A use-case describes a sequence of actions that provides something of measurable value to an actor, and it is drawn as a horizontal ellipse. The actor is a person, organization, or external system that plays a role in one or more interactions with the system in question. In many use-cases, more than one actor is involved. Actors are drawn as stick figures. With regards to user-role modelling (see Chapter 4.1.2.5), actors are to be replaced by roles in UI specification. The notation, however, does not necessarily need to change. Figure 40 shows an example from a lecture organization system from (Ambler 2006f). Students are enrolling in courses with the potential help of registrars. Professors input the marks that students earn on assignments, and registrars authorize the distribution of transcripts (report cards) to students.

Diagrammatic associations

Associations between actors and use-cases are indicated in use-case diagrams by solid lines (see Figure 40). An association exists whenever a user role is involved with an interaction described by a use-case. Associations are modelled as lines connecting use-cases and roles to one another, with an optional arrowhead on one end of the line. The arrowhead is often used to indicate the direction of the initial invocation of the relationship or to indicate the primary actor within the use-case. However, associations do not represent directed flows of information. Information is flowing back and forth between the actor and the use-case. The relationship between use-cases can also be determined by specialization, by the includes-dependency and by the extends-dependency. The specialization relationship allows inheritance, while the includes-relationship allows a use-case to be decomposed into different parts. The extends-relationship states that a use-case is part of another one. (Holt 2005) reports on another frequently used relationship, which is the constrains-relationship. It is used to link functional and non-functional requirements. While the functional requirement represents some function of the software, the non-functional requirement can constrain the way the function can be realized. In BPM, such non-functional requirements can be quality requirements (e.g. meeting certain standards), implementation requirements (e.g. the technology used) or environmental requirements.

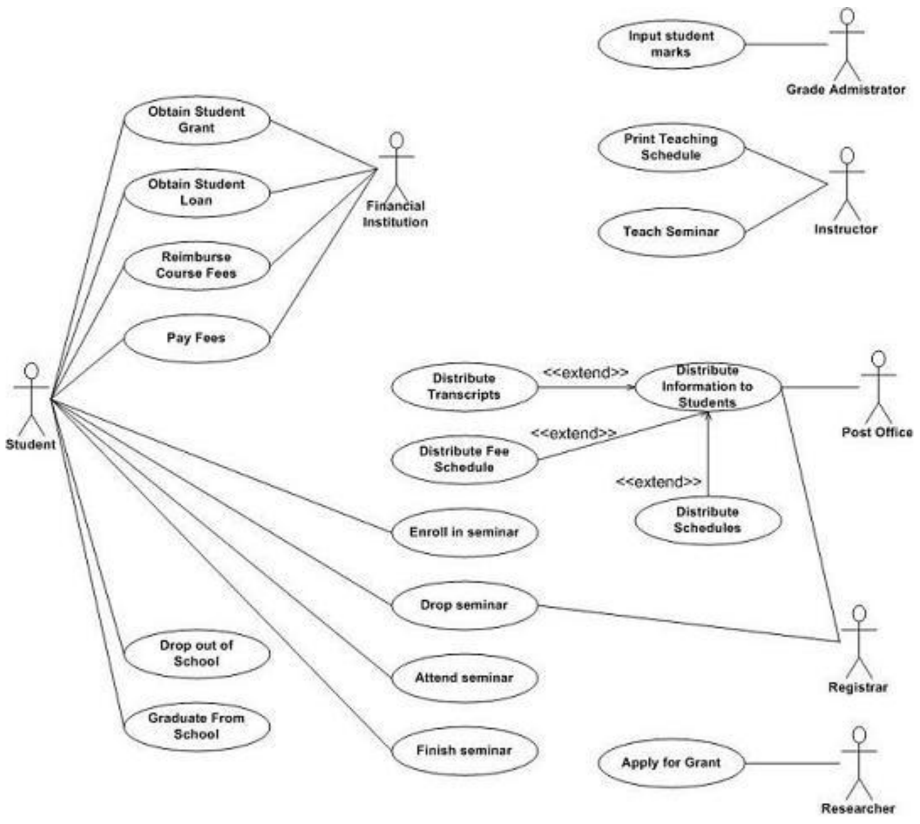


Figure 40: A use-case diagram (Ambler 2006f)

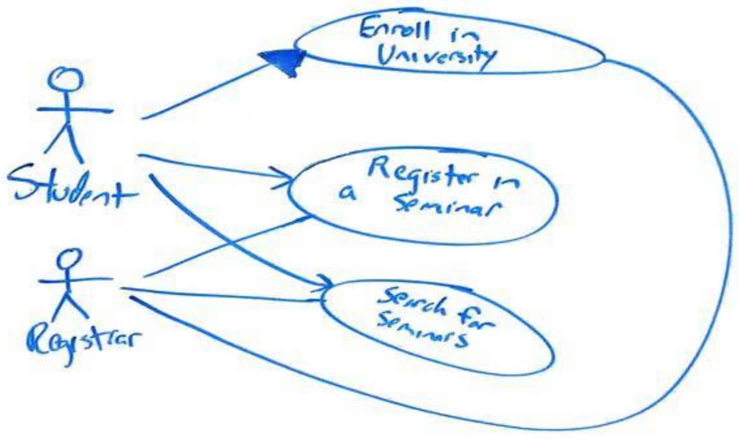


Figure 41: An agile and sketchy use-case diagram (Ambler 2006f)

Keeping use diagrams agile

In order to make use-case diagrams less formal and easier to develop even for non-IT specialists, they can be kept simple. Agile developers typically create use-case diagrams on a whiteboard (see Figure 41). In AM, the principle that content is more important than representation allows us to abstract from the strict UML notation. The result is a more pragmatic and approximate use-case diagram, but one that is still good enough to describe the use-case in a semi-formal and diagrammatic way (see discussion on the right formality of notations in Chapter 3.2). Moreover, agile use-case diagrams do not need to be complete, because modifying the diagram is allowed at any point in iterative development (Ambler 2006f).

Developing use-case diagrams

(Ambler & Jeffries 2002; Ambler 2004b) recommend that modelling use-case diagrams is started by identifying actors or user roles (Constantine & Lockwood 1999b). Their interaction with the system should then be analyzed to identify an initial set of use-cases. The diagram is then used to connect actors or user roles with the use-cases in which they are involved. In every case, if a user role *"supplies information, initiates the use case, or receives any information as a result of the use case, then there should be an association between them*" (Ambler 2006f).

Table 38: Use-case diagrams in brief

Primary use	Providing an overview of the usage requirements
Modelling technique / form	Semi-formal diagrammatic; pragmatic, approximate agile notation
Advantages	Expresses interrelationships between actors/roles and cases; need not be complete
Disadvantages	Associations between elements might not be clear without annotations; user roles not part of the typical use-case diagram
Related to	Usage scenarios, use-case diagrams, task maps

4.1.3.5 Essential Use-cases / Task Cases (SE)

Essential use-cases

Use-cases are one of the cornerstones of SE practice, but in their original form (Jacobson 1992) were not well suited to UI design issues, having too many built-in assumptions and premature decisions about the UI (see Chapter 4.1.3.3). The abstract, simplified form of use-cases created for usage-centred design came to be known as task cases or essential use-cases (Constantine 1996; Constantine & Lockwood 1999a; Constantine & Lockwood 1999b; Constantine & Lockwood 2001; Constantine et al. 2003; Constantine & Lockwood 2003) and have become widely used in IxD. The need to anchor the task model in an appropriate understanding of users quickly became apparent to those working with essential use-cases (Constantine 2005).

"Essential use cases were developed by Constantine and Lockwood (1999) to combat what they see as the limitations of both scenarios and use cases [...]. Essential use cases represent abstractions from scenarios, i.e. they represent a more general case than a scenario embodies, and try to avoid the assumptions of a traditional use case [...] (by a) division between user and system responsibilities." (Preece et al. 2002), p. 230

User intention and system responsibility

An essential use-case is a simplified use-case that captures the intentions of a user in a technology- and implementation-independent manner. The essential use-case is especially suitable for describing how the system reacts to user input in terms of its interactive behaviour. A fully documented essential use-case is a structured narrative, expressed in the language of the application domain and of users in particular roles. For each role, the associated task cases must cover all the responsibilities of that role. In practice, task-case models are more fine-grained than typical use-case models. The increased granularity helps in partitioning the UI design into small and understandable tasks. This, in turn, facilitates the development of a more modular and flexible UI architecture.

Data-gathering for essential use-cases

(Constantine & Lockwood 1999b) suggest asking stakeholders the following questions from the point of view of the users in order to develop good essential use-cases:

- What are users in this role trying to accomplish?

- To fulfil this role, what do users need to be able to do?
- What are the main tasks of users in this role?
- What information do users in this role need to examine, create, or change?
- What do users in this role need to be informed of by the system?
- What do users in this role need to inform the system about?

The format of essential use-cases

Essential use-cases are typically written in two-column format (see Figure 42). Users in their role attempt to do something with the system and receive one or more responses to their action. In an essential use-case, the flow is apparent from the spacing of the actions and responses. Essential use-cases should also have a unique name. They can also carry preconditions and postconditions.

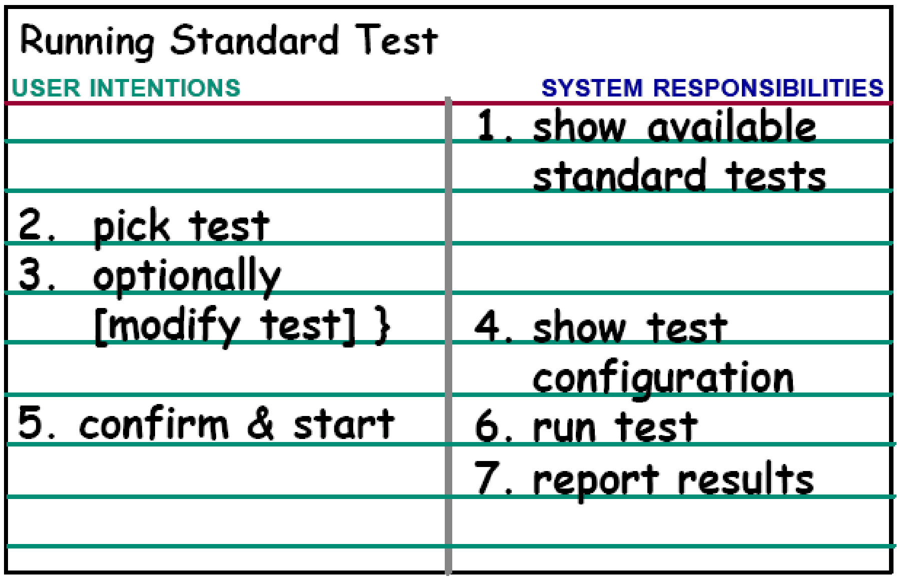
"For guiding human interface design, the focus on the system and on concrete action and interaction proved to be problematic, leading to the development of task cases, a more abstract form modeling user intentions rather than actions and system responsibilities rather than responses." from (Constantine et al. 2003), p. 4

Focal use-cases

Usually, some essential use-cases can be considered as especially important for UI design. Known as 'focal use-cases', they refer to patterns of interaction that are most relevant to the system. The level of importance can be due to user requirements, stakeholder interests or business goals. In agile development, focal use-cases can be linked to focal user roles. Hence, focal elements of the system and the UI can be prioritized in iteration planning.

Advantages of essential use-cases

A great advantage of essential use-cases is their focus on what is really happening at the UI while not concerning themselves with implementation issues (see Table 39). While traditional system use-cases (see Chapter 4.1.3.3) contain many assumptions about how the system and the UI could be built from a technological point of view, essential use-cases concentrate rather on what the user intends to do (see Figure 42). Concrete steps about how the interaction will be executed are not included, which helps to understand requirements without making premature commitments about specific solutions.


Running Standard Test

USER INTENTIONS	SYSTEM RESPONSIBILITIES
	1. show available standard tests
2. pick test	
3. optionally [modify test] }	4. show test configuration
5. confirm & start	6. run test
	7. report results

Figure 42: An example of a task case / essential use-case; from (Constantine et al. 2003)

Disadvantages of essential use-cases

When used with essential use-cases, task modelling can be technology-independent. Essential use-cases are never completely system-independent, however. The modeller will always develop essential use-cases having in mind the scope of and vision of the project. In this way, even abstract use-cases can influence the development process in some way.

Table 39: Essential use-cases in brief

Primary use	Modelling user-UI interaction patterns; task modelling
Modelling technique / form	Table-based notation
Advantages	Focus on the essential patterns of interaction; technology- and implementation-independent
Disadvantages	Never fully system-independent
Related to	Use-cases; use-case / task maps

4.1.3.6 Use-case / Task Maps (SE)

Hierarchies of use-cases

Normally, use-cases do not exist in isolation and many hundreds of them may be developed in the requirements-engineering process. For UI development, (Constantine & Lockwood 1999b) propose a hierarchy of uses cases in order to visualize dependencies and sequences of user-system interaction. Such hierarchical use-cases can be called use-case- or task-case maps or, more simply, task maps. They are intended to describe the interrelationships of user tasks in order to guide UI design from a wider perspective. With the task map, small sub-tasks can be arranged as an expressive overview of the activities the user wants to perform. With regards to the essential use-case notation, each case in the use-case map is described in a table-based textual style. Due to the nature of tasks, and as with classical use-case diagrams, cases in the task map can be classified in terms of specialization (i.e. classification), extension, composition or affinity (see Figure 43). Such relationships help to understand links between tasks, which in turn supports the design of UIs suitable for specific workflows.

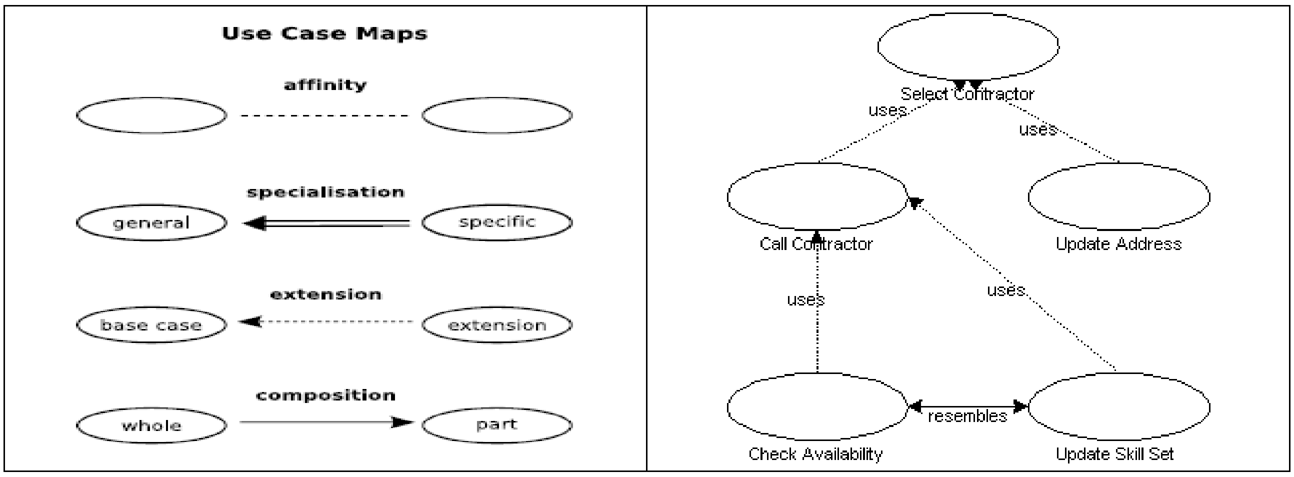


Figure 43: The notation for task maps; an example for a task map (UIDesign 2003)

Classification

The relationship of classification refers to use-cases that are specialized forms of other use-cases. This principle is related to the concept of inheritance in object-oriented programming. A more specialized sub-case "-is-a (kind of)" another more general case. The advantage of this concept is that shared patterns of user-system interaction do not have to be written over and over again. Redundancy is prevented through a cross-reference of cases.

Extension

The relationship of extension is relevant to use-cases that extend each other. An extension exists if a use-case provides additionally inserted, or alternative, patterns of interaction, but the rest of the course stays the same. A simple example is the relationship between ‘changing an image’ and ‘browsing an image’. The use-case for browsing images extends the use-case for changing images. This means (1) the course for browsing images is modelled separately for ease of reuse, (2) the pattern for browsing images does not occur in the ‘changing an image’ use-case, (3) but the necessity for browsing could become relevant at any time during changing. In order to visualize this interrelationship, the table-based notation of the essential use-case ‘browsing images’ should carry the names of the essential use-case(s) it extends.

Composition

Use-cases can be easily decomposed into smaller parts. Composition then means that the supercase makes use of the patterns of interaction stated in the subcases. The ‘uses’ relationship employs the principles of classification and extension. With composition, the supercase explicitly refers to the subcases that are needed to complete a user task. This is what distinguishes composition from classification and extension, where the supercase does not know about its related subcases. In general, composition should be reserved for required parts or sub-sequences. Where additional or alternative interaction is possible, but not necessary to complete a task, extension should be used.

Affinity

The relationship of affinity is usually relevant during the early stages of modelling. In situations where the exact kind of relationship is still unknown, use-cases can be connected with a dotted line. But affinity can also be helpful in the later stages of UI specification. Due to the connection between use-cases, they are likely to be positioned close to each other on the use-case map. With regards to UI design, simply being in the vicinity of use-cases can highlight information items that should be placed close to each other. In other situations, some use-cases that have been identified as being somehow related can turn out to be, in fact, similar. In order to keep the requirements analysis traceable and authentic, both use-cases might be kept on the use-case map. The affinity relationship can then also indicate the equivalency of use-cases.

Table 40: Task maps in brief

Primary use	Visualizing hierarchies of use-cases
Modelling technique / form	Diagrammatic, semi-formal
Advantages	Visually externalizes interrelationships of use-cases
Disadvantages	None
Related to	Use-cases; essential use-cases; use-case diagrams

4.1.3.7 Concurrent Task Trees (SE)

A notation intended to easy model-based UI design

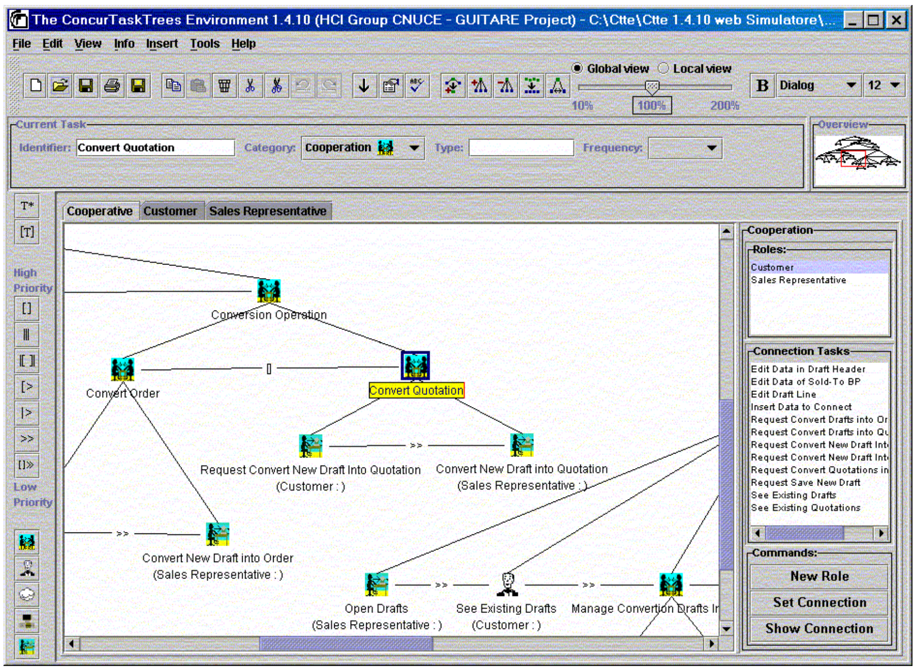
ConcurTaskTrees (CCT) is a notation for task-model specifications that has been developed by (Paterno et al. 1997; Paternò 2000b; Mori et al. 2002) to overcome limitations of notations previously used to design interactive applications. Its main purpose is to be an easy-to-use notation that can support the design of real industrial applications, which usually means applications with medium-large dimensions. The CTT notation is supported by the ConcurTaskTrees Environment (CTTE), which is a sophisticated application widely appreciated in the research community (see Figure 44).

GOMS

The motives for CTT come from the limitations of text-based notations (see Table 41), such as GOMS (Goals, Operators, Methods, Selection Rules) (Dix et al. 2003) and the User Action Notation (UAN) developed by (Hix & Hartson 1993). With GOMS it is possible to describe the action sequences required to reach a goal and group them into methods that may have an associated selection rule to indicate when their performance is recommended (Mori et al. 2002). GOMS does not consider user errors or the possibility of interruptions, and is focused on modelling sequential tasks. (Paternò 2000a) therefore notes that it can be inadequate for distributed applications (such as web-based applications).

UAN

UAN is also just a textual notation and specifies user action and system feedback at a low level only. With UAN, it is possible to describe the temporal relationships between asynchronous tasks. UAN is a table-based notation indicating user actions, system feedback, and changes in the state of the application. However, the lack of UAN tool support has prevented a wider spread of the approach (Mori et al. 2002).



The ConcurTaskTrees Environment 1.4.10 (HCI Group CNUCE - GUITARE Project) - C:\Ctte\Ctte 1.4.10 web Simulatore\...
File Edit View Info Insert Tools Help
Global view
Local view
10%
100%
200%
Dialog
12
Current Task
Identifier: Convert Quotation
Category: Cooperation
Type:
Frequency:
Overview
High Priority
Low Priority
Cooperative
Customer
Sales Representative
Conversion Operation
Convert Order
Convert Quotation
Request Convert New Draft Into Quotation (Customer :)
Convert New Draft into Quotation (Sales Representative :)
Convert New Draft into Order (Sales Representative :)
Open Drafts (Sales Representative :)
See Existing Drafts (Customer :)
Manage Conversion Drafts In
Cooperation
Roles:
Customer
Sales Representative
Connection Tasks
Edit Data in Draft Header
Edit Data of Sold-To BP
Edit Draft Line
Insert Data to Connect
Request Convert Drafts into Or
Request Convert Drafts into Qu
Request Convert New Draft Int
Request Convert New Draft Int
Request Convert Quotations in
Request Save New Draft
See Existing Drafts
See Existing Quotations
Commands:
New Role
Set Connection
Show Connection

Figure 44: The ConcurTaskTrees Environment (Paternò 2008a)

CTT extends GOMS and UAN

In contrast, CTT comes with a graphical syntax, a rich set of (temporal) operators for modelling asynchronous and synchronous tasks and the hierarchical structure of tasks. In all, the main features of ConcurTaskTrees are (Mori et al. 2002; Paternò 2008b):

- Hierarchical structure: the hierarchical structure helps to decompose a task into smaller problems, while still maintaining the relationships between the smaller parts of the solution. The hierarchical structure provides a large range of granularity allowing large and small task structures to be reused, and it enables reusable task structures to be defined at both a low and a high semantic level.
- Graphical syntax: a graphical syntax is often (not always) easier to interpret; in this case it should reflect the logical structure, so it should have a tree-like form.
- Concurrent notation: operators for temporal ordering are used to link subtasks at the same abstraction level. This sort of aspect is usually implicit, expressed informally in the outputs of task analysis. Making the analyst use these operators is a substantial change to normal practice. The reason for this innovation is that after an informal task analysis designers must express clearly the logical temporal relationships. This is because such ordering should be taken into account in the UI implementation to allow the user to perform at any time the tasks that should be active from a semantic point of view.
- Focus on activities: this allows designers to concentrate on the most relevant aspects when designing interactive applications that encompass both user- and system-related aspects, thus avoiding low-level implementation details that at the design stage would only obscure the important factors.

Due to its characteristics, the CTT's notation has been shown to be an expressive and flexible notation able to represent concurrent and interactive activities. It is at the same time a compact representation that is able to provide much information in an intuitive way without requiring excessive efforts from the users of the notation (see Table 41).

Table 41: CTT in brief

Primary use	Hierarchical task modelling
Modelling technique / form	Diagrammatic, semi-formal; textual informal (early descriptions, scenarios)
Advantages	Visual notation that extends text-based forms; can be combined with informal descriptions; special tool support (CTTE); can be saved to XML format
Disadvantages	Rather formal; generated UIs are straightforward and less innovative from today's point of view; not a standard in SE, IxD or BPM
Related to	Use-cases; task maps; scenarios

Working with CTT

It has been demonstrated in practice in industry that workers without a background in computer science can apply CTT. If it is difficult to immediately create the model from scratch, CTT provides the possibility of loading an informal textual description of a scenario or a use-case and interactively selecting the information of interest for the modelling work. To develop a task model from an informal textual description, the CTTE (see Figure 45) user will analyze the description of the scenario and identify the main tasks that occur in this description. In this way, the designer can first identify tasks, then create a logical hierarchical structure, and finally, complete the task model (Mori et al. 2002).

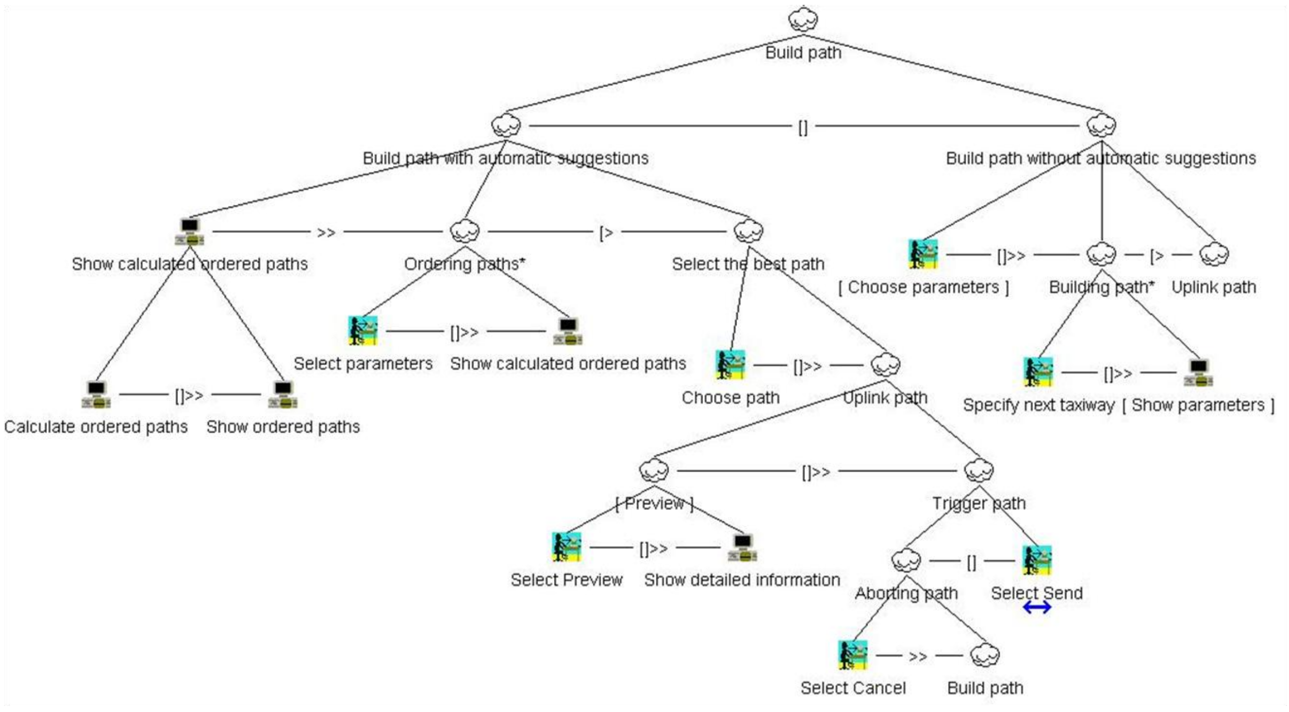


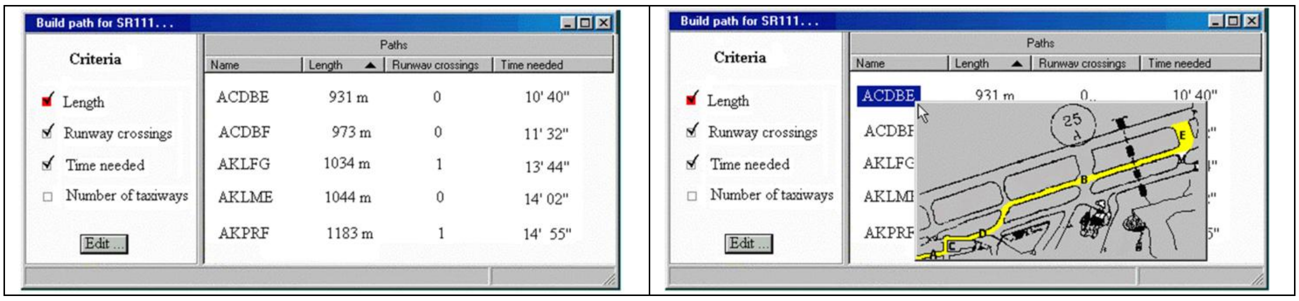
Figure 45: An excerpt of a task model developed with CTTE (Paternò 2000a)

Rendering UIs from the CTT format

As with other tools, CTTE allows the task model to be saved in XML format. With a specific DTD format for task models specified by CTT, information can be exchanged with other modelling environments such as rendering systems that are able to generate the UI for specific platforms using the task model as an abstract specification.

Discussion

The CTT notation and the CTTE were developed over a long period of time and attracted wide attention in the community. However, the UIs that are generated on the basis of CTT are rather mundane from the design point of view. Instead of being very attractive and aesthetic UIs, the strengths of CTT-made UIs lie rather in the field of safety critical applications (Paternò et al. 1998), for example in the aerospace industry (see Figure 46). But on the whole, CTT contributes some very important elements for identifying a common denominator in task modelling. The tree notation avoids similarities with UML as far as possible. Nevertheless, the notation provides a sophisticated set of shapes and connectors that - especially during the earlier stages - can be associated with additional textual descriptions. The rich set of temporal operators associated with the hierarchical structure allows designers to describe flexible behaviours in both individual and multi-user activities, taking into account possible interruptions and the various ways in which such behaviours may evolve (Mori et al. 2002). On the other hand, the means provided for modelling focus on just a small sub-set of required artefacts and the CTT notation is not a real standard in SE or any other discipline. The developers of CTT therefore considered integration with IBM Rational Rose and a use-case add-on in order to obtain more thorough support in modelling.


Build path for SR111...

Criteria

- ✓ Length
- ✓ Runway crossings
- ✓ Time needed
- ☐ Number of taxiways

Paths			
Name	Length ▲	Runway crossings	Time needed
ACDBE	931 m	0	10' 40"
ACDBF	973 m	0	11' 32"
AKLFG	1034 m	1	13' 44"
AKLME	1044 m	0	14' 02"
AKPRF	1183 m	1	14' 55"

Figure 46: UI developed with CTT (Paternò 2000a)

4.1.3.8 Business Use-Cases (BPM)

Business use-cases define business processes and how the business serves its customers. The business use-case aids in understanding the context and scope of the business. While the system use-case model represents the scope of an application, a single business use-case can have many system use-case models associated with it, where each use-case model represents a single application.

Table 42: An essential business use-case exemplified by the enrol-in-seminar use-case from (Ambler 2006e)

User Intention	System Responsibility
Student identifies him/herself	Verify eligibility to enrol via BR129 Determine Eligibility to Enrol. Indicate available seminars
Choose seminar	Validate choice via BR130 Determine Student Eligibility to Enrol in a Seminar. Validate schedule fit via BR143 Validate Student Seminar Schedule Calculate fees via BR 180 Calculate Student Fees and BR45 Calculate Taxes for Seminar. Summarize fees Request confirmation
Confirm enrolment	Enrol student in seminar Add fees to student bill Provide confirmation of enrolment

Business use-cases describe business processes. These processes are documented as a sequence of actions that provide observable value to a business actor. A business use-case is basically a story that relates a business problem to a solution and a

Text-based business use-cases

solution to a technology, and as a consequence to a research issue. Business use-cases facilitate the understanding between an organization and a research group since they set a common context and simple natural language for interaction. They evolve in complexity and change in character as the development process continues. In some methodologies, they may begin as brief business use-cases, evolve into more detailed system use-cases, and then eventually develop into highly detailed and exhaustive test cases.

Essential business use-cases

Text-based business use-cases are between essential use-cases and system use-cases, although leaning towards the essential end of the spectrum (Ambler 2006e). Essential use-cases and business use-cases are very similar, but a business use-case is often more focused on the business process and existing technology. The essential business use-case therefore includes references to business rules (see Table 42). Due to the principle of avoiding use-cases of mass destruction (Ambler 2004a), the usual essential use-case just references another artefact that documents the business vision. Similar to essential use-cases, text-based business use-cases are relatively easy for business people to write and very easy for untrained people to read, and so they have a growing use in business-process reengineering work (Cockburn 2002).

Business use-case diagram

The business use-case can also be a diagram illustrating the scope of the business being modelled (see Table 43). The diagram contains business actors (roles played by organizations, people, or systems external to the business), their goals for interacting with the business and the services or functions they request from the business (Scheer 1998). A very popular modelling tool for business use-case diagrams is ARIS (see Figure 47). A primary purpose of the model of business use-cases and actors is to describe how the business is used by its customers and partners. Activities that directly concern the customer, or partner, can be presented, as can also the supporting or managerial tasks that indirectly concern the external party.

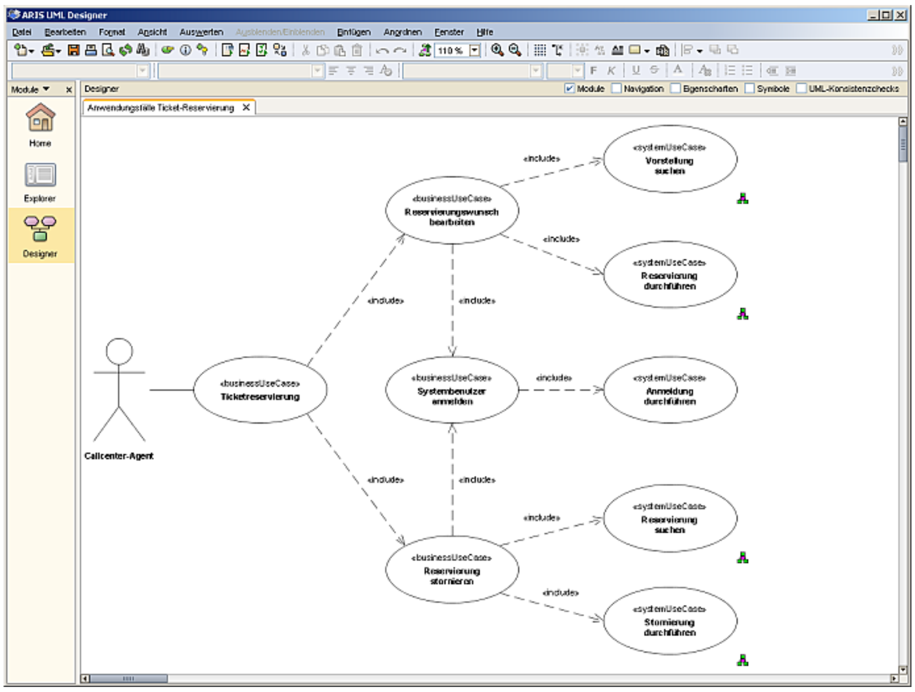

ARIS UML Designer
Datei Bearbeiten Format Ansicht Auswerten Ausblenden/Einblenden Einfügen Anordnen Fenster Hilfe
110 %
Module
Home
Explorer
Designer
Designer
Module Navigation Eigenschaften Symbole UML-Konsistenzchecks
Anwendungsfälle Ticket-Reservierung
Callcenter-Agent
«businessUseCase» Ticketreservierung
«businessUseCase» Reservierungswunsch bearbeiten
«businessUseCase» Systembenutzer anmelden
«businessUseCase» Reservierung stornieren
«systemUseCase» Vorstellung suchen
«systemUseCase» Reservierung durchführen
«systemUseCase» Anmeldung durchführen
«systemUseCase» Reservierung suchen
«systemUseCase» Stornierung durchführen
«include»

Figure 47: UML business use-case diagram in ARIS UML Designer (IDS Scheer AG 2008b)

The business use-case diagram is very similar to the classical use-case diagram (see Chapter 4.1.3.4). The visual model of a business can provide important insights into its logical validity and how it might be improved. It serves as the link between IT strategy and implementation (Österle 1994). As with SE, and from the point of view of software development, business use-case diagrams can be described in more detail with sequence diagrams (Scheer 1998; Holt 2005), as described in Chapter 4.1.4.3.

Table 43: Business use-cases in brief

Primary use	Modelling a business use-case; understanding context and scope of application
Modelling technique / form	Table-based (essential); diagrammatic (use-case diagram notation)
Advantages	Notation similar to use-case models; easy to understand
Disadvantages	None
Related to	Essential use-cases; use-cases

4.1.3.8 The common denominator for task modelling

Commonalities in UI modelling

From the discussion of task modelling, several important findings can be deduced. Firstly, task modelling recognises a wide variety of notations, ranging from text-based ones to diagrammatic ones. Secondly, SE, IxD and BPM factually share some of the notations, although the nomenclature used is different. For example, the essential use-case notation can also be the basis for business use-cases. Moreover, the use-case notations applied in SE can be easily stripped down in order to be compatible with the notations of IxD and BPM. Thirdly, the diagrammatic notations, such as use-case diagrams and task maps, are particularly well suited to providing an overview of users' tasks. Text-based notations, in turn, provide a much higher level of detail and provide insight into the exact task patterns. Existing hybrid notations such as CTT could potentially contribute to a common denominator, but the notation is not known as a standard outside a specific community of researchers. This makes its application difficult in corporate and industrial processes. Nevertheless the CTT notation prompts us to avoid a strict UML style and to combine diagrammatic and textual descriptions.

Essential use-cases and task maps

Problems scenarios (see Chapter 4.1.1.1) were identified as the text-based starting point for domain modelling. With personas and role maps, the set of models for UI specification was supplemented with a compact and structured model on the one hand, and a visually enhanced one on the other. In task modelling, text-based and visual forms of modelling also go hand in hand. Due to their applicability in all three disciplines and their focus on the UI, essential use-cases are the perfect means for describing user tasks in detail. As with role maps in user modelling, task maps accordingly provide the opportunity to 'maintain the big picture'. Software developers, interaction designers and business modellers are able to use task maps because their notation is derived from use-case diagrams.

Use-case diagrams

However, it is difficult to be the necessary link between the different models if tasks and users are modelled only in separate artefacts. In UI specification, having permanent associations between users and their tasks is critical. Hence, UI specification also needs a notation such as use-case diagrams. As explained previously, the actors in classical use-case diagrams are to be replaced by user roles. Consequently, the UI-focused use-case diagram will provide visual support for modelling and managing users and tasks. If a use-case diagram includes both cases and user roles, it becomes a very important mediator between personas and role maps on the one hand, and essential use-cases and task maps on the other. With regards to an agile approach in UI specification, the use-case diagram does not need to be perfect and formal. Following (Ambler 2004b; Ambler 2006f), simple associations between the shapes of the diagram are good enough, which means that creating and understanding use-case diagrams becomes less important. In addition, the motive for following the link between the interrelated artefacts is increased, because looking at one single diagram is not enough.

4.1.4 Behaviour Modelling: Methods and Artefacts

Behaviour models can be used to identify shortcomings in use-cases and erroneous steps in interaction patterns. Therefore drawing flow charts and other types of flow diagrams is among the most important activities in UI modelling. With respect to UI design, interaction sequences have to be smoothened and consolidated. In the following, the most relevant kinds of notations for behaviour and dialogue flow modelling are discussed with regards to UI specification. As outlined in Chapter 3, the focus is on pragmatic and semi-formal notations. From the field of SE, agile models contributed by (Ambler & Jeffries 2002; Ambler 2004b) therefore play a dominant role. Consequently, UML models are not in the front line when identifying models with the right behaviour, whereas it is important to recognize that Ambler's models are derived from the more formal notations. In addition, both IxD and BPM recognize many notations for modelling the flow of UIs (see Table 44). Due to the great variety of existing notations, those models that are closely related are described together in one section. The common denominator for behaviour modelling is subsequently presented at the end of this chapter.

Table 44: Behaviour models contributed from different disciplines, derived from Chapters 2 + 3

Contributing Discipline	**Behaviour Model**
Software Engineering	Sequence diagrams (Ambler 2004b), activity diagrams (Ambler 2004b), state charts / state machine diagrams (Ambler 2004b), flow charts (Preece et al. 2002; Ambler 2004b; Sharp et al. 2007), data-flow diagrams Ambler 2004a), UI flow diagrams (Ambler 2004b); storyboards (Robertson & Robertson 1999)
Usability Engineering	Flow charts, UI storyboards (Preece et al. 2002; Richter & Flückiger 2007; Sharp et al. 2007), navigation maps (Constantine & Lockwood 1999b)
Business-Process Modelling	Sequence diagrams (Holt 2005), activity diagrams (Holt 2005)

4.1.4.1 State Charts (SE), Flow Charts (UE, SE) and Data-flow Diagrams (SE)

State charts

The state-chart language (Harel 1987) is the formalism that is the basis of the state-chart diagram. State-chart diagrams describe the behaviour of reactive systems such as modern event-based UIs (Traetteberg 2002; Trætteberg 2004; Trætteberg 2008) and has been used for UI design (Horrocks 1999). The strength of state charts is their support for hierarchical states.

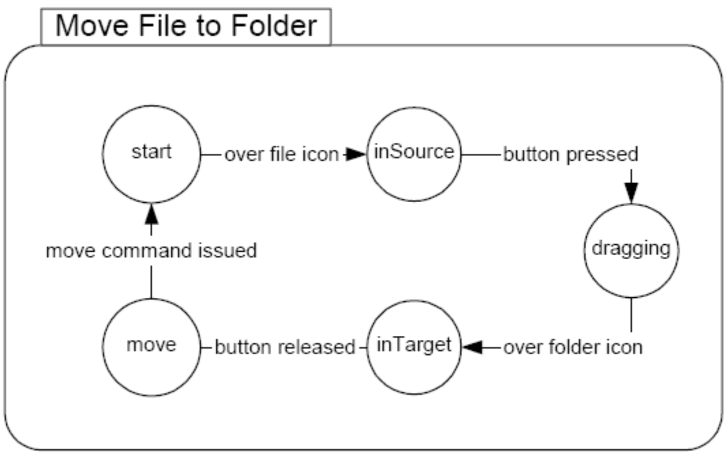


Figure 48: State charts for modelling the modes of direct manipulation; from (Trætteberg 1998)

State charts in UI design

(Markopoulos & Marijnissen 2000) suggest using state charts for modelling the high-level navigation of dialogues and (Trætteberg 1998; Trætteberg 2004; Trætteberg 2008) propose a UI-modelling method based on state charts, called DIAMODL (see Chapter 5.1). For example, state charts could be used for modelling the modes that are typical of direct-manipulation interfaces. In Figure 48, each step of a mouse gesture is modelled as a state, and transitions are triggered by appropriate mouse events. In this case, the various states model how a source file is dragged to, and dropped onto, a folder (Trætteberg 1998).

Advantages and shortcomings of state charts

From the perspective of SE, state charts are a good choice for UI specification, because the notation is widely known among UML experts. On the other hand, this advantage is also the most significant shortcoming of state charts. Because of the abstract notation, not all parts of the UI can be visually modelled in a very expressive way.

The state-chart notation is good for specifying interaction patterns such as the 'move file to folder' example. The notation can no longer be used, however, as soon as abstract states have to be replaced by highly detailed screens that tend to describe the look of a whole interface. At this stage, other notations such as storyboards or UI prototypes become more appropriate. As proposed by (Mrdalj & Jovanovic 2002), state charts can then formally represent user interaction at the UI (see Figure 49). Each independent event is then appropriately modelled as a transition in the state chart, while a detailed UI design increases stakeholders' understanding.

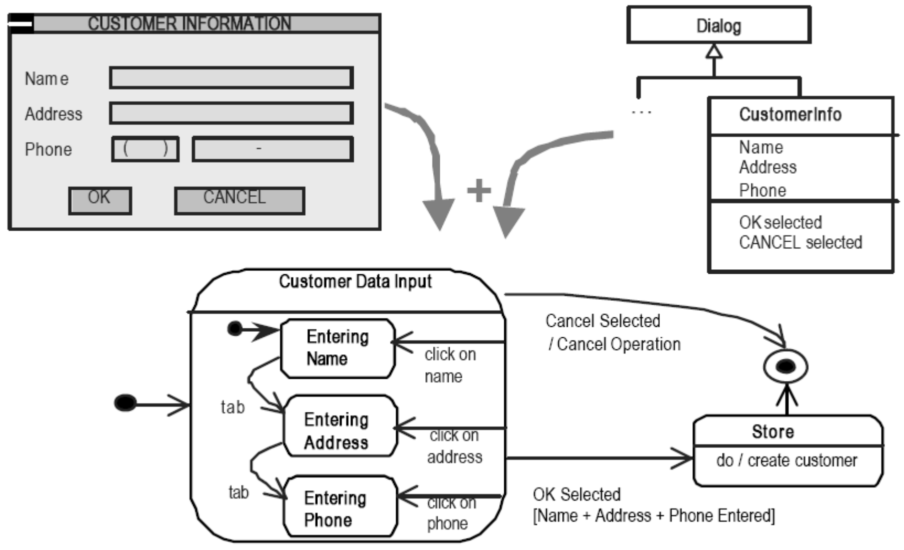


Figure 49: State charts and UI prototype; from (Mrdalj & Jovanovic 2002)

Flow charts

Flowcharts are a modelling technique popularized for structured development in the 1970s (Ambler 2006g). It is common to use flow charts to model the logic of large software modules. But flow charts are also typically used to describe the detailed logic of a business process. In a flowchart there are three basic elements (see Figure 50). Squares represent activities or tasks, diamonds represent decision points, and arrows represent the flow of control. Flowcharts now have some additional types of symbols, such as off-page connectors. They can be used when a diagram becomes too big and there is a need to connect two separately modelled diagrams to each other. In addition, input/output symbols represent printed reports and data storage options.

Agile flow charts

For the purpose of UI modelling with many stakeholders who are not technically skilled, agile flow charts are particularly appropriate. (Ambler & Jeffries 2002; Ambler 2004b; Ambler 2006g) suggest modelling flow charts on a whiteboard for discussions with visual aids. In agile environments, it is the discussions with stakeholders that are more important than the purpose of modelling itself.

Data-flow diagrams

Data-flow diagrams (Stevens et al. 1974; Gane & Sarson 1979) are process models used to describe data flow within a system. They are especially popular as a result of their application in structured analysis (SE). They show the flow of data from external entities into the system and the data that is moved from one process to another, as well as its logical storage (Ambler 2006c). It is common to employ data-flow diagrams to model the context of a system, which shows the interaction between the system and outside entities. The data-flow diagram also helps to divide the logic and interrelationships of a system into smaller parts.

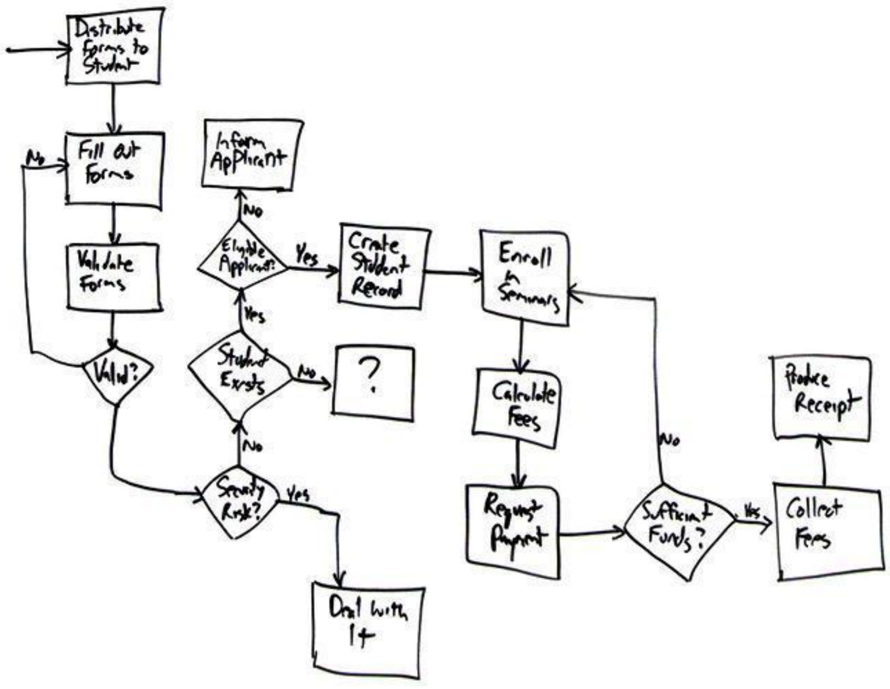


Figure 50: An agile flow-chart diagram (Ambler 2006g)

The elements of data-flow diagrams

The diagrammatic notation of data-flow diagrams is based on four elements (see Figure 51). Squares represent external entities that are either sources of, or destinations for, data. Rounded rectangles represent processes, which handle some input. Open-ended rectangles indicate electronic or physical data stores. Arrows show the flow of data, which can be either electronic data or physical items.

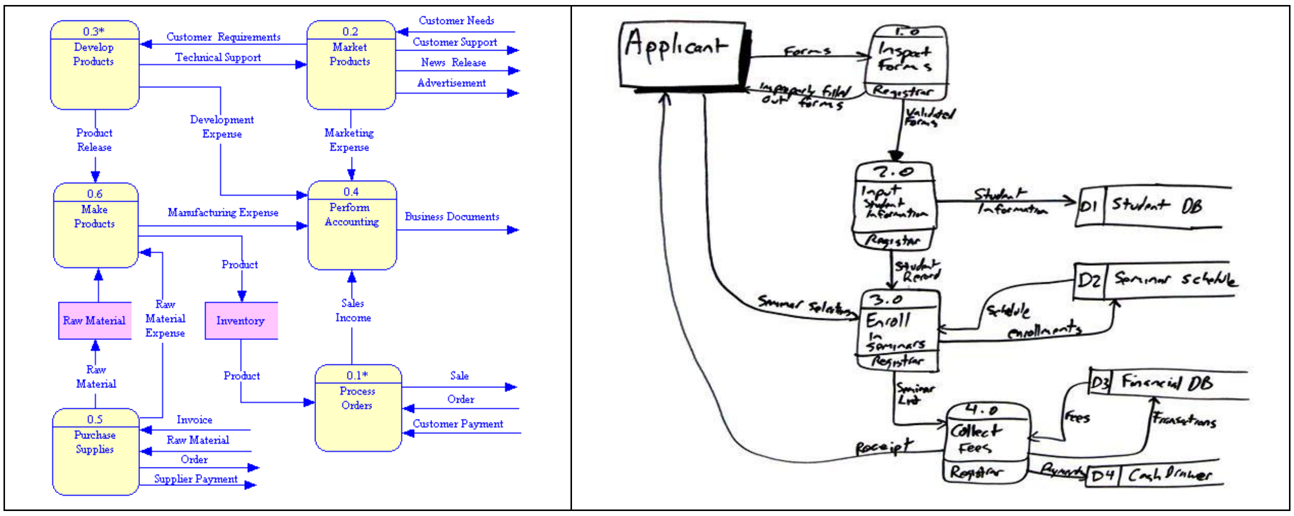


Figure 51: Data-flow diagrams in a formal (Gane & Sarson 1979) style (left) from (Excelsoftware 2008), and the corresponding agile (right) notation of (Ambler 2006c)

(Ambler 2006c) proposes some modelling rules for creating data-flow diagrams. First, all processes must have at least one data flow in and one data flow out. Second, all processes should modify the incoming data, producing new forms of outgoing data. Third, each data store must be involved with at least one data flow. Fourth, each external entity must be involved with at least one data flow. And fifth, a data flow must be attached to at least one process. In the end, it is possible to keep data-flow diagrams agile by creating small models and employing simple and common tools such as whiteboards.

Modelling rules

Table 45: Flow models (state charts, flow charts, data-flow diagrams in brief

Primary use	Modelling the high-level navigation of dialogues (state charts); describing the behaviour of UIs; describing the detailed logic of a business process (flow charts); modelling data flow within a system (data-flow diagrams)
Modelling technique / form	Diagrammatic with defined shapes
Advantages	Notation is widely known (especially in SE); can be linked to UI prototypes
Disadvantages	Inadequate if abstract states must be replaced by highly detailed screens (i.e. coming close to storyboarding and prototyping)
Related to	Storyboards; UI flow diagrams; UI prototypes; use-cases

In summary, state and data-flow diagrams are a suitable means for documenting and understanding the flow of logic in a system and consequently at parts of the UI (see Table 45). Because UI design today is very much interlinked with software development, data-flow diagrams provide good support to the consideration of databases or external entities that have an influence on the course of events. But state events need to be linked to concrete representations of the UI design to allow traceability while travelling from the abstract to the detail.

Flow diagrams in UI specification

4.1.4.2 UI Flow Diagrams (SE), Storyboards (UE) and Navigation Maps (UE, SE)

Because of their close relationship, UI flow diagrams, UI storyboards and navigation maps are analyzed in this section for their potential contribution to a common denominator for UI specification. Although all three methods have different names, they all model the same aspects of UI design. It is therefore important to recognize the best characteristics of all three notations of this kind in order to determine the right dialogue-flow notation for interdisciplinary UI specification.

UI flow diagrams are intended for high-level relationships and interactions. Up to now, UML has not supported UI flow diagrams or storyboard-like notations (Ambler 2006h), which has led to a variety of different notations in UI development. Consequently, UI flow diagrams are also called storyboards, interface flow diagrams, and navigation maps (Constantine & Lockwood 1999b). This leads to exactly the situation that a common denominator for UI specification is meant to avoid. Models of UI flow help to specify the high-level relationships between major UI elements and thereby ask fundamental usability questions (Ambler 2006h). This is why the different notations must be combined into a modelling language that is able to unite different groups of stakeholders under the umbrella of interdisciplinary UI specification.

UI flow diagrams

In the UI flow diagram (see Figure 52), boxes represent major UI elements and arrows represent the possible flows between them. Accordingly, arrows in the UI flow diagram are related to transitions in activity diagrams (Ambler 2006h).

The elements of the UI flow diagram

Following (Ambler 2006h), UI flow diagrams are typically used to model the interactions of users with the UI as defined in a single use-case. A single use-case can refer to several screens and provides insight into how the UI is used. Based on the information in the use-case, the UI flow diagram reflects the behavioural view of the single use-case. UI flow diagrams enable the designer to gain a high-level overview of the UI, which can be called the architectural view of the UI (Constantine &

UI flow diagrams

Lockwood 1999b), i.e. a navigation map (see below). The overview supports the interaction designer in understanding how the system, and the UI in particular, is expected to work. In this way, UI flow diagrams help to determine the usability of the UI. If there are many boxes and many connections, it may indicate poor usability caused by too much complexity (Ambler 2006h).

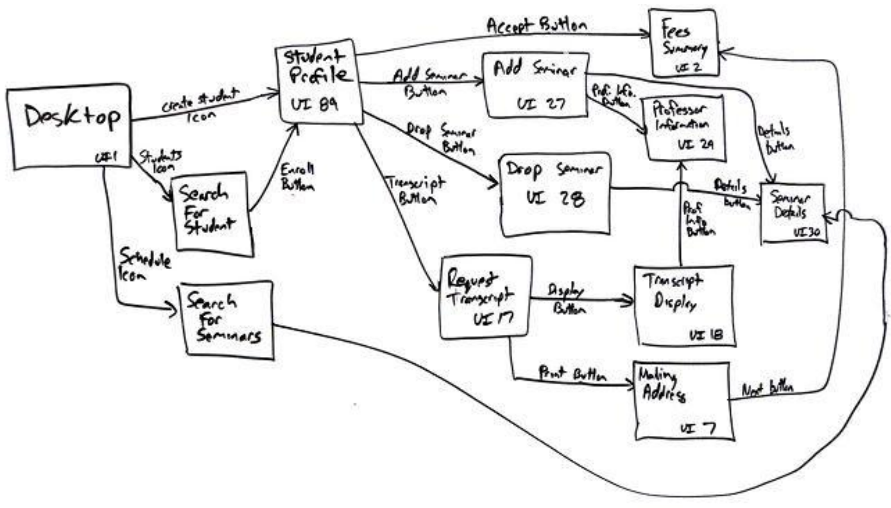


Figure 52: UI flow diagram; from (Ambler 2006h)

Storyboards

In the film and cartoon industry, storyboards are very popular. It helps the producer to communicate the scenes of a movie to the actors. While discussing the storyboard, some actors might have suggestions for how to change a scene, for example to make the movie more expressive or exciting. The storyboard also helps to picture the flow of events in the movie. It therefore gives the director the opportunity to ensure people will be able to understand the sequence of actions later on in the movie. The linked pictures in the storyboard identify the story line (Robertson & Robertson 1999). In UI specification, storyboards illustrate interactive behaviours by showing what the UI looks like before and after an end-user event occurs (Lin 1999). In general, a storyboard is the visualization of a scenario. As in the film industry, the scenario is supposed to help the interaction designer in getting things right before any line of code is written. Because many stakeholders speak different technical languages, as explained in the previous chapters, storyboards are the visual spearhead of analysis and abstract modelling.

The fidelity of UI storyboards

In order to picture the interactions that take place at the UI, scenarios can have different kinds of presentation. In many cases, even very sketchy and comic-like storyboards will provide very good results from discussions with stakeholders. The process of sketching storyboards is also well supported by some successful tools, for example DENIM (Lin et al. 2000; Lin et al. 2001; Newman 2003) and DAMASK (Lin 2003b; Lin 2003a). Both tools (see Figure 53) will be explained in more detail in the relevant chapter of this thesis (see Chapter 5). Alternatively, UI storyboards can also be designed with very high-fidelity representations of the look of the software. This is especially helpful when concrete UI elements or layouts have to be discussed. UI storyboards can also include drawings of the stakeholders involved. This kind of notation often gives storyboards a comic-like appearance.

(Richter & Flückiger 2007) explain that UI storyboards are especially important in situations where, for example, just the text alone of a usage scenario is not expressive enough. The pictorial elements of UI storyboards are thus particularly helpful in explaining new and very innovative concepts. The visual representation also helps to create more understanding of the design requirements; if user experience plays an important role, for example. Above all, storyboards should be applied for modelling dialogue flow, visualizing difficult or new concepts, and presenting important aspects of the context of application. In order to be able to externalize this information, the UI storyboard tells a story and thus exemplifies the usage of the interactive system. The story is intended to explain interdependencies of UI elements or UI states and has to highlight critical or unresolved design issues. The example chosen should therefore have some complexity and involve the user roles or personas that are most likely to use the system in the way explained.

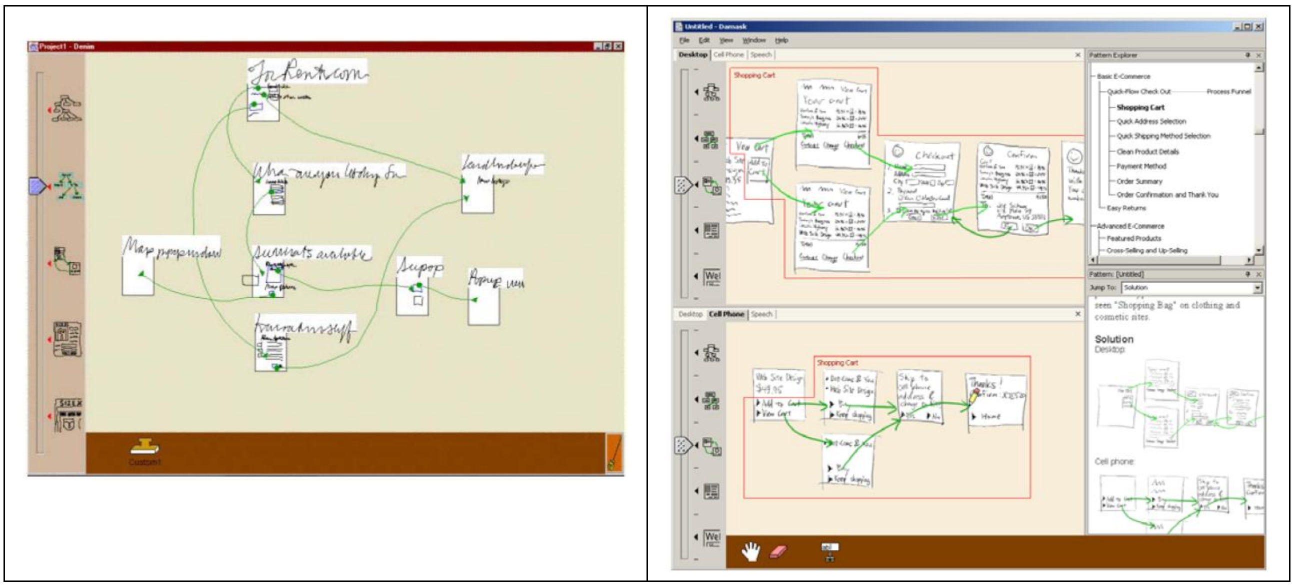


Figure 53: DENIM and DAMASK in storyboard view; from (Lin et al. 2000; Lin 2003a)

The role of storyboards

The ingredients of a UI storyboard

The UI storyboard usually evolves over time and changes iteratively during the UI specification project. Until it reaches its final, specification-mature state, from one to many different storyboards may have been designed. Different UI storyboards preserve a lot of design knowledge, because they provide pictorial examples of how alternative dialogue flows could have looked. Every UI storyboard usually carries along information about user needs, business processes, new or changed functionality, layout and design detail. Naturally, one might also develop a different storyboard for the same design solution, but focus on different aspects (i.e. design elements, functions).

Advantages of UI storyboards

All told, UI storyboards are very helpful in propelling the development towards the best UI design solution. To this end, storyboards provoke ongoing discussions through the visual externalization of design trade-offs and the confrontation with possible flows of dialogue. Especially when the storyboard is used to discuss different technical solutions, it is the perfect means for business people and end-users to communicate with programmers and vice versa. They make UI storyboarding fun, which in turn helps to raise the motivation to contribute to the design process (Robertson & Robertson 1999).

Navigation Maps

Navigation maps, as proposed by (Constantine & Lockwood 1999b; Constantine & Lockwood 2002), help to model the complexity of an interactive system. Small and simple interaction spaces are usually easier to understand than very complex information architectures. The trade-off in designing interaction spaces is to find the right balance between integrating information in the same context of use and distributing interaction patterns over many separate contexts. Accordingly, (Constantine & Lockwood 1999b) also talk of 'context navigation maps', because the navigation at the UI might include transitions between different interaction spaces. An interaction space can be defined as an encapsulated usage scenario that supports reaching a specific goal. As soon as scenarios overlap or depend on each other, there exists a transition between them. The transitions indicate an exchange of information or the path of the user, which leads through several scenarios.

The navigation map (see Figure 54) visualizes the overall architecture of the UI in terms of a network of interaction contexts. In the navigation map, each interaction space is represented by a rectangle. Arrows between the objects represent the possible transitions between those interaction spaces, which the use might traverse. Navigation maps can have different purposes. The behavioural view models the transitions associated with travelling through a single use-case.

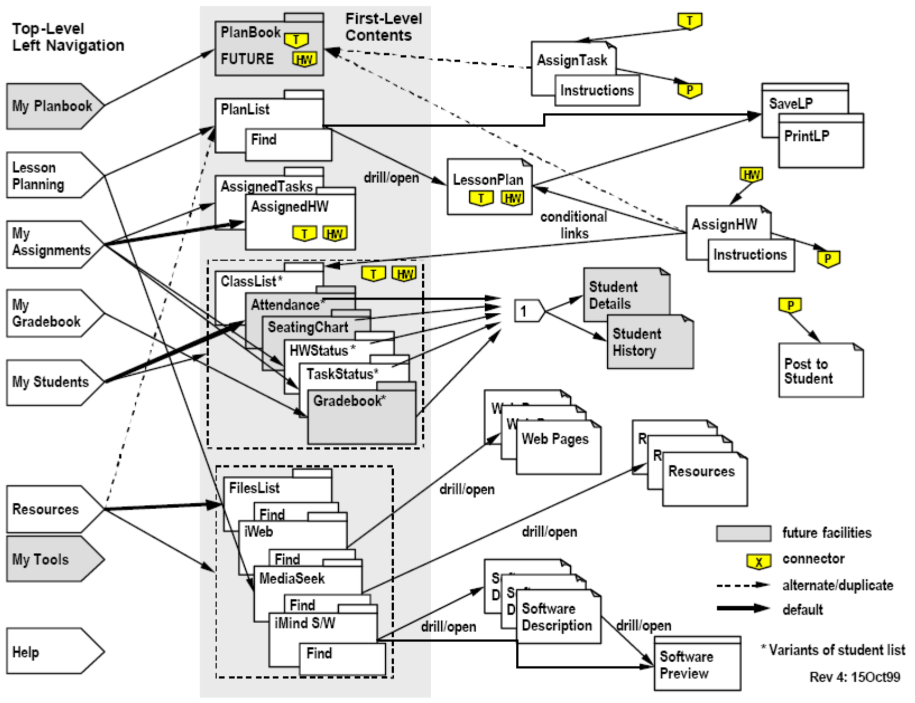


Figure 54: A navigation map (Constantine & Lockwood 2002)

The integration of all use-case transitions into one single navigation map is called the architectural view, because such a diagram provides the ultimate overview of all interrelationships. The sequential view of modelling navigation maps is characterized by interaction spaces that are connected to each other in the order of appearance according to a use-case. In contrast to the behavioural view, the sequential view can contain the same interaction space several times, for example if it appears many times in the dialogue flow. This is why the sequential navigation map is closely related to UI storyboarding. It is especially suited to communicating with the user. Through modelling navigation maps, very long chains of transitions can be identified and analyzed with regards to usability. Navigations maps are therefore powerful tools in understanding the overall organization of complex UIs (see Table 46). The navigation map makes it easy to experiment with different arrangements of UI flow in a quite simple notation. After all, if there is any difference between navigation maps and storyboards, it is that navigation maps tend towards modelling the big picture of interaction patterns. However, both notations are able to model the same aspects of UI design.

Table 46: Storyboards (i.e. navigation maps or UI flow diagrams) in brief

Primary use	Modelling the interactions of users with the UI as defined in a single use-case; gaining a high-level overview of the UI
Modelling technique / form	Semi-formal diagrammatic; sketchy; sequences of detailed UIs
Advantages	Supports understanding of system and UI; drives communication about trade-offs; externalizes the story line of the UI; gives overview of transitions between navigation spaces (navigation map)
Disadvantages	Informality may raise disbelief in the usability of the method
Related to	Use-cases; UI prototypes; scenarios

4.1.4.3 Sequence diagrams (SE, BPM)

Modelling the sequence of tasks

Sequence diagrams support modelling the sequence of tasks (Traetteberg 2002; Sommerville 2004) and the flow of logic within a system in a visual manner. They visualize how a group of actors or objects operate and communicate over time (Jacobson 1992; Balzert 2000). Communication is interpreted as interaction, information flow and/or message passing. In the design phase, the sequence diagram describes how objects invoke each other's services or methods. As with activity diagrams (see Chapter 4.1.4.4.), sequence diagrams model the behavioural aspects of a system, but at a high level (Holt 2005). From the perspective of experienced software architects, sequence diagrams (see Figure 55) are among the most important design-level models for (agile) software development (Ambler 2004b; Ambler 2006i).

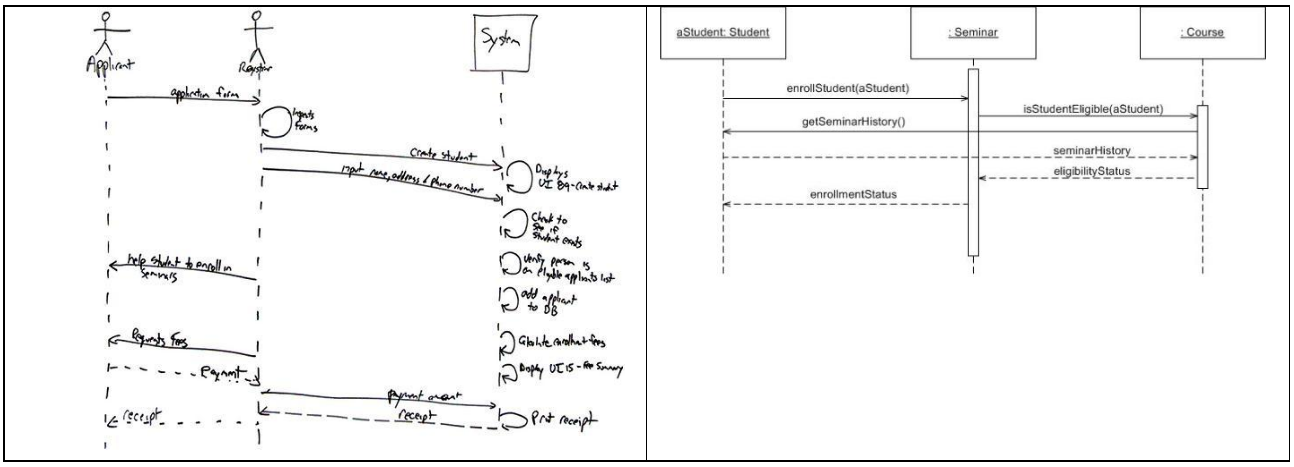


Figure 55: An agile sequence diagram (left) and a normal UML sequence diagram (right); from (Ambler 2004b; Ambler 2006i)

Sequence diagrams in UI specification

In the context of UI specification, sequence diagrams can therefore be used to model usage scenarios. The logic of a usage scenario is usually related to one or several use-cases (Ambler 2004b; Ambler 2006i). If the UI architecture must be described with regards to the implementation of the interactive system, sequence diagrams can also be used on a more abstract level of UI development, namely to explore the logic of a complex interaction operation or function that affects the UI.

Sequence diagrams and UI prototyping

Developing sequence diagrams is heavily influenced by the UI prototype, especially in understanding the different steps in a scenario or the sequence of scenarios (Mrdalj & Jovanovic 2002). Sequence diagrams can therefore be used to formally describe UI navigation. Each form of interaction with the use-case needs then has an appropriate representation in the sequence diagram (see Figure 56). The dialogue flow leads to the definition of the interactions in the sequence diagram. For each screen, all generated events are captured by the appropriate messages in the sequence diagram.

The elements of sequence diagrams

In UI specification, the boxes across the top of the sequence diagram will usually represent instances of use-cases. The dashed vertical lines below the boxes are lifelines, which indicate how long the element of a use-case is relevant during the scenario being modelled. The vertical lines are combined with boxes on the lifelines if the object is active, i.e. if interaction patterns of the use-case are affected by the sequence of logic. However, drawing such activation boxes is not important and the sequence diagram can still be understood if they are omitted (Ambler 2004b). Messages are indicated as (labelled) arrows. Return messages are optionally indicated using a dashed arrow. Textual annotations can be placed on the sequence diagram to explain certain steps or messages in the flow of logic.

Maintaining agility

With regards to stakeholders, it is important to keep the diagrams simple in interdisciplinary UI specification. Again, whiteboards can be adequate for drawing sequence diagrams together with stakeholders. As (Ambler 2004b) suggests, the greatest value from drawing sequence diagrams comes from the discussion they usually provoke while looking at the flow of the dialogue (see Table 47).

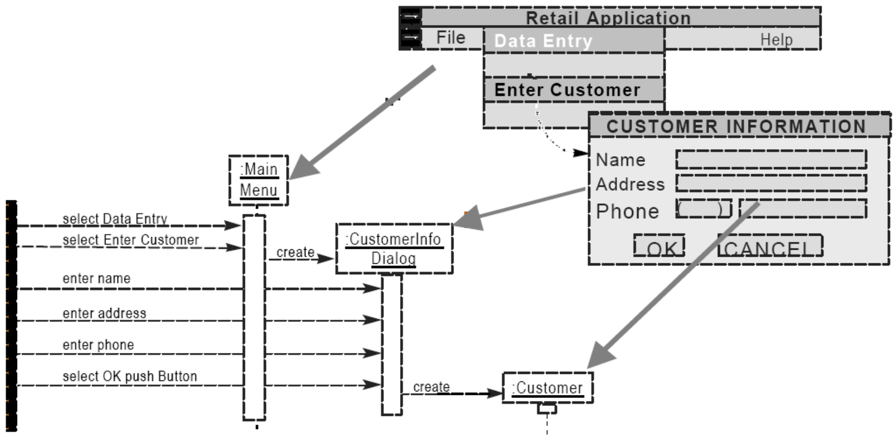


Figure 56: The relationship of sequence diagram (left) and UI prototype; from (Mrdalj & Jovanovic 2002)

Moreover, modelling sequence diagrams should be done in small increments (AM: model in small increments). It is more appropriate to create smaller models that focus on specific parts, rather than trying to get everything in one single diagram. Because sequence diagrams are very close to visual coding, software developers in the UI specification team might want to create very accurate diagrams. As this is a time-consuming activity, the team should focus on the most important set of diagrams in order not to spend too much time on sequence diagrams compared to other modelling activities. With regards to the UI, this means that particularly the very complex sequences of UI flow can be modelled, while straightforward interaction patterns are not.

Table 47: Sequence diagrams in brief

Primary use	Modelling the flow of logic within the system in a visual manner; visualizing logic of a use-case or scenario
Modelling technique / form	Diagrammatic style
Advantages	Relationship to scenarios and use-cases; provoke discussion; describe UI formally
Disadvantages	Can become very complex; very close to coding
Related to	Use-cases; scenarios; UI prototyping

4.1.4.4 Activity diagrams (SE, BPM)

Activity diagrams

The activity diagram is used for both analysis and design, e.g. for modelling business processes or to control flow (Traetteberg 2002). Activity diagrams are arguably sophisticated flow charts (Ambler 2006g) and can be said to be derived from them (Holt 2005). In fact, they are the object-oriented equivalent of flow charts and data-flow diagrams (Ambler 2004b).

Activity diagrams and class diagrams

Activity diagrams make tangible the behavioural aspect of the overall UI model. In contrast to sequence models (see Chapter 4.1.4.3), activity diagrams are used to model detailed behaviour (Holt 2005). In SE, activity diagrams are related to class diagrams. Whereas class diagrams identify attributes and operations, activity diagrams specify the order in which operations are executed and the information flow. In this context, the activity diagram also models responsibilities to identify which entity of the system is asked to provide a special service. The agile activity diagram in Figure 57 depicts one way to model the logic of the 'enrol in seminar' use-case (see Table 34 & Table 35), including not only the basic course of action but also the alternative courses.

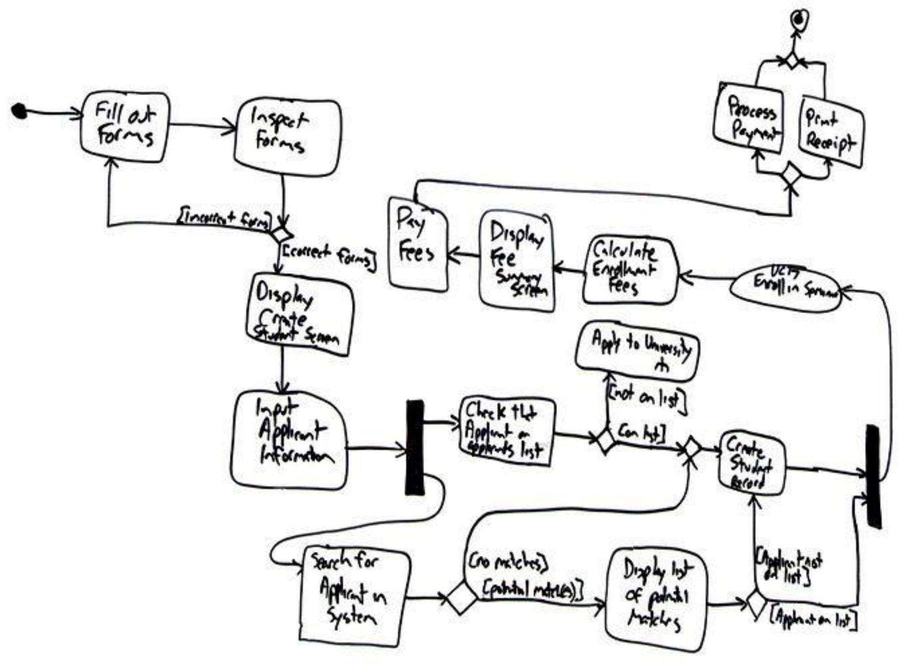


Figure 57: An agile activity diagram; from (Ambler 2004b; Ambler 2006j)

Activity diagrams in UI design

Activity diagrams are well suited to capturing a series of actions taken on behalf of a user at the UI. They can be a strong addition to the UI modeller's tool kit (Lieberman 2004). They provide the ability to construct a roadmap of user functionality that accurately represents the paths a user can follow. However, the activity diagram does not discuss in detail why or when a particular path is followed. Instead, this information is in the use-case notation (see Chapter 4.1.3.3).

The elements of the activity diagram

The elements of an activity are quite simple to understand and also reveal the close relationship to flow charts and data-flow diagrams. The following shapes can be found in a typical activity diagram (Ambler 2004b):

- Initial node: the filled-in circle is the starting point of the diagram.
- Activity final node: the filled circle with a border is the ending point.
- Activity: the rounded rectangles represent activities that occur.
- Flow: the arrows on the diagram.
- Fork: a black bar that denotes the beginning of parallel activity.
- Join: a black bar that denotes the end of parallel processing. All flows going into the join must reach it before processing may continue.
- Condition: text on a flow, which must evaluate to true to traverse the node.
- Decision: a diamond with one flow entering and several leaving.
- Merge: a diamond with the implication that one or more incoming flows must reach this point before processing continues.

Figure 58 illustrates the elements of the activity diagram of a wizard UI, which assists users in creating new printer ports and installing printer driver files. In the example, found in (Lieberman 2004), the wizard is a series of UI screens that lead the user through the printer configuration process. The resulting activity diagram has several branches in its flow, as well as some associated data elements. With regards to the design of the UI, this means that if the user does not provide a mandatory data element, then an exception will occur and an error message will be displayed at the interface. Accordingly, activity diagrams can be used to model error handling at the UI level, which belongs to the most relevant aspects of UE.

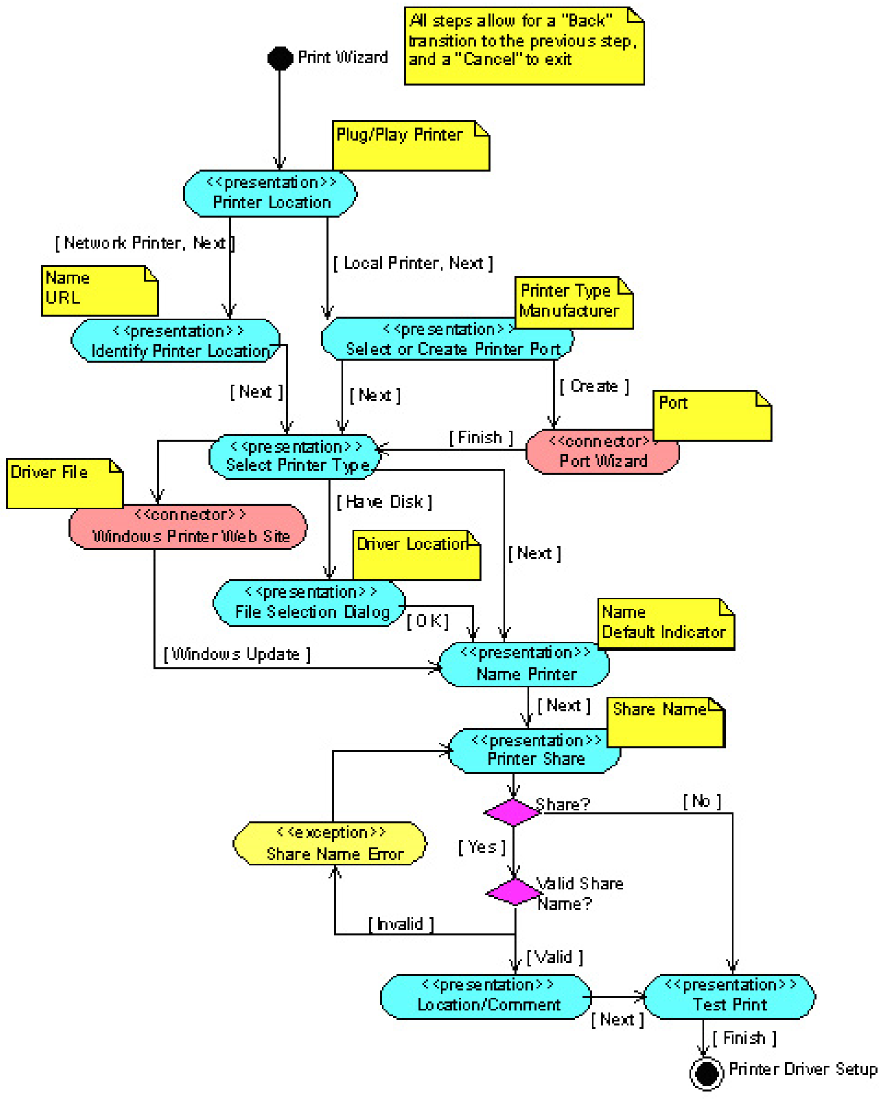

All steps allow for a "Back" transition to the previous step, and a "Cancel" to exit
Print Wizard
Plug/Play Printer
<<presentation>> Printer Location
[Network Printer, Next]
[Local Printer, Next]
Name URL
<<presentation>> Identify Printer Location
Printer Type Manufacturer
<<presentation>> Select or Create Printer Port
[Next]
[Next]
[Create]
Port
[Finish]
<<connector>> Port Wizard
<<presentation>> Select Printer Type
Driver File
<<connector>> Windows Printer Web Site
[Have Disk]
Driver Location
[Next]
<<presentation>> File Selection Dialog
[OK]
Name Default Indicator
[Windows Update]
<<presentation>> Name Printer
[Next]
Share Name
<<presentation>> Printer Share
Share?
[No]
<<exception>> Share Name Error
[Yes]
Valid Share Name?
[Invalid]
[Valid]
<<presentation>> Location/Comment
<<presentation>> Test Print
[Next]
[Finish]
Printer Driver Setup

Figure 58: The UI flow of a print wizard modelled in an activity diagram; from (Lieberman 2004)

Activity diagrams in BPM

Because of their ability to visually model behaviour, activity diagrams are also popular in BPM. In his pragmatic guide to BPM, (Holt 2005) proposed activity diagrams for modelling the behaviour of business processes. The notation is identical to the one proposed by (Ambler 2004b).

Table 48: Activity diagrams in brief

Primary use	Modelling business processes and controlling flow; representing path users can follow
Modelling technique / form	Semi-formal diagrammatic; agile
Advantages	Relatedness to flow charts and data-flow diagrams helps bridge the gaps
Disadvantages	No support for task hierarchies
Related to	Flow charts; data-flow diagrams

4.1.4.5 The common denominator for behaviour modelling

IxD is not enough – widening the horizon of interaction designers

For behaviour modelling with UI storyboards and flow charts, IxD only recognizes two popular notations. With regards to interdisciplinary UI specification, it is therefore necessary to widen the horizon of UI-related modelling by taking into account the notations that are popular in SE and BPM. In fact, the determination of a common denominator in behaviour modelling is easier than one might expect.

UI storyboards

Although differing in name, UI storyboards (IxD) are very closely related to the UI flow diagrams that have been proposed by AM (SE). Navigation maps, which are also proposed by software development, are close to IxD because of their incorporation in usage-centred design (Constantine & Lockwood 1999b). However, navigation maps do tend to model the big picture and focus on the transitions between interaction spaces. UI storyboards and UI flow diagrams, in contrast, enable the step-by-step modelling of a dialogue flow. Accordingly, both navigation maps and a storyboard-like notation are sensible ingredients of a set of UI-related models for UI specification. Because UI flow diagrams and storyboards are exactly the same, which of the two notations is actually chosen is irrelevant.

Activity diagrams for interdisciplinary modelling with developers

Considering state diagrams, flow charts are well suited to representing a common denominator in interdisciplinary modelling because their notation is related to activity diagrams and data-flow diagrams (Ambler 2004b). With regards to the traceability of design decisions and requirements, the selection of closely related models is arguably preferable. However, the close relationship to activity diagrams is one aspect in favour of choosing just one of the two diagrams. Because activity diagrams can be closely linked to UI prototypes and their notation is almost identical to that of flow charts, an interdisciplinary UI specification team should have no difficulty in applying them as a common modelling language. Furthermore, if activity diagrams are used, software developers will feel more comfortable in describing UI behaviour and thinking about the UI.

Data-flow diagrams for UI modelling with regards to data providers

Because data-flow diagrams are nevertheless important for modelling the dependency of the UI on data providers, they still have a right to be integrated into a shared modelling repository. However, the team potentially has the opportunity to combine both kinds of diagrams by integrating associations to data providers into the activity diagram. This is feasible, because activity diagrams are the object-oriented equivalent of data-flow diagrams (Ambler 2004b).

Besides storyboards, navigation maps and activity diagrams, sequence diagrams are also interesting options for UI modelling. With a sequence diagram, the UI specification team can model complex flows of events at the UI layer, taking into account especially those parts of the system architecture on which the UI depends. Ultimately, the diagrams discussed contribute directly to the UI prototyping stage, and the design decisions based on behaviour modelling have a profound influence on the design layer

4.2 UI Prototyping

"The act of bringing thoughts into material form is not incidental to the act of creation but is itself constitutive of and essential to creation." (Clark 2001), p.9

UI Prototyping and AM

In the scope of UI specification, there is a need for both incomplete and abstract prototypes as well as for prototypes that determine the UI design in unambiguous detail. Moreover, the means provided for prototyping must align with agile semi-formal models that support bridging the gaps between the participating disciplines. Indeed, the development of UI prototypes aligns with basic principles (see Table 49) and agile practice (see Table 50).

Table 49: Outline of core principles of agile development and their compatibility with early-stage UI prototyping; based on (Memmel et al. 2007c).

Agile principle	**Compatibility with UI prototyping**
Model With A Purpose	The purpose of low-fidelity prototyping is to drive creativity and design collaboration. That of high-fidelity prototyping is to simulate the feel of the UI for specific use-cases
Rapid Feedback	The time between an action and the feedback from that action is critical for efficient development. Tools must allow rapid (high-fidelity) prototyping
Embrace Change	Prototypes should allow easy enhancement or change.
Incremental Change	Stakeholders should be able to create low-fi prototypes first, then more sophisticated hi-fi ones later

Table 50: Outline of core practices of agile development and their compatibility with early-stage UI prototyping; based on (Memmel et al. 2007c)

Agile practice	**Compatibility with UI prototyping**
Active Stakeholder Participation	Active stakeholder participation can be easily promoted with prototyping (PD).
Apply The Right Artefacts	Some modelling languages are inappropriate for describing parts of the system. Everybody understands prototypes.
Create Several Models In Parallel	Different disciplines apply different models. Different design alternatives are explored in parallel and expressed through different models.
Iterate To Another Artefact	When a prototype cannot describe certain parts of the system, other kinds of externalization must be used. For example: switch from low-fi to hi-fi.
Model To Communicate	Prototypes need to look attractive in order to show them to decision makers. High fidelity can be used for discussion and release planning.
Model To Understand	Prototyping helps in understanding and exploring the problem space. Prototypes support the stakeholders even during early stages of design.
Use The Simplest Tools	Most stakeholders are used to certain expressive software, e.g. PowerPoint. Prototyping tools should work equally easily.

The variety of prototypes

In order to cope with a complex UI design process, and beginning at the early stages, designers often need to externalize the actual design thinking through sketchy visual representations such as drafts or wireframes. As described previously, they encourage the discussion and articulation of problems, and prototyping therefore makes sure that the product is actively designed rather than simply produced by chance. With more interactive and complex external representations, the designer carries out a dialogue about design solutions and ideas. Prototypes that are visually more detailed help us to overcome the limitations of our cognitive abilities to process, develop, and maintain complex ideas and to produce a detailed operative image (Löwgren & Stolterman 2004). The overall prototyping process can therefore minimize the risk of making the wrong design decisions and leads the way towards a winning design solution (see Figure 59). The reduction in risk that results from decision-making in the design process is balanced by the constant generation of new ideas and creativity that open up new opportunities to improve the design (Buxton 2007).

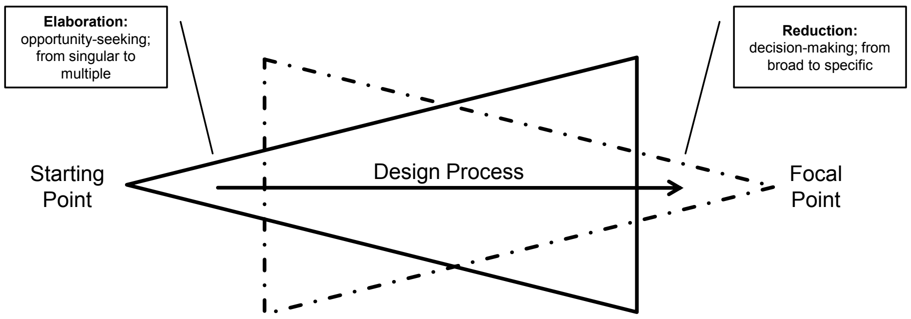

Elaboration: opportunity-seeking; from singular to multiple
Reduction: decision-making; from broad to specific
Starting Point
Design Process
Focal Point

Figure 59: Overlapping funnels in the design process; from (Buxton 2007)

Ultimately, prototypes can be classified in terms of different dimensions. Among them, the dimensions of low- and high-fidelity are the most popular. With regards to the multifaceted challenges of UI specification, this thesis will take a closer look at the nature of prototypes and the process of creating them in order to determine the best practice for UI specification-level prototyping.

4.2.1 Prototypologies

Prototypes as boundary objects

As discussed in Chapter 3, prototypes are instruments applied by both SE and UE. While SE uses them as vehicles for inspections, testing and incremental development (XP's small release), IxD mainly uses prototypes for participatory design. For both disciplines, prototypes are an *"excellent means for generating ideas about how a UI can be designed, and it helps to evaluate the quality of a solution at an early stage"* (Bäumer et al. 1996). In all, prototypes can be boundary objects for bridging SE, UE and other stakeholders in the UI design. They can serve as the common language to which all can relate (Bäumer et al. 1996; Rudd et al. 1996; Gunaratne et al. 2004).

Table 51: Purposes of prototyping, based on (Bäumer et al. 1996)

Goal	**Description**
Evolutionary prototyping	- Continually adapt a system to a rapidly changing environment - Ongoing effort to improve an application
Experimental prototyping	- Used to test hypotheses - Try out technical solutions to meet requirements - Communicate about technical and usability issues - Gather experience about the suitability and feasibility of a particular design
Exploratory prototyping	- Used when problem at hand is unclear - Express to developer how something should work and look - Design exploration of a variety of solutions - Understand user needs and tasks and how they can be supported by IT - Elicit ideas and promote cooperation and communication between stakeholders

Purposes of prototyping

As there are several approaches to prototyping, it is important to first determine the purpose of any prototype you build (see Table 51). (Bäumer et al. 1996) distinguish three main types of prototypes. When enhancing or changing an existing system, evolutionary prototypes allow testing of how the next version of a software product will look and feel. For designing a UI from scratch, experimental and exploratory prototypes are the appropriate vehicles for propelling the design process. Experimental prototypes can be compared to spike solutions in XP, while exploratory prototyping, for example with card-based or paper prototypes (see Figure 60 and Figure 61), is very closely related to participatory design, which is an applied method of RE in both UE and SE. Exploratory prototyping is a suitable means for driving communication and understanding requirements through producing tangible design artefacts. The characteristics of exploratory prototypes are therefore very appropriate for them to also play an important role in UI specification.

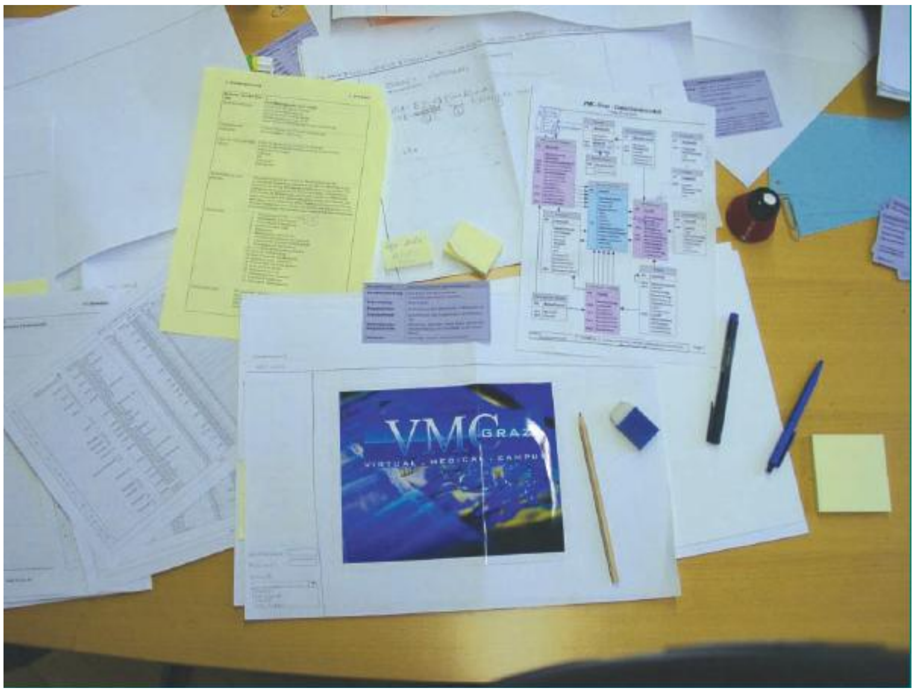


Figure 60: The production of paper prototypes; from (Holzinger 2004)

Prototypes used in SE

(Lichter et al. 1993) identified four different kinds of prototypes that are especially popular in SE (see Table 52). They describe a presentation prototype as the visualization of design aspects that should drive communication between the stakeholders, or more precisely in the context of UI specification, between the client and the supplier. If problems or misunderstandings arise while exploring a presentation prototype, a more focused prototype is required to clarify the situation. With a more functional 'prototype proper', certain trade-offs can be discussed.

Table 52: Classification of prototypes in SE; based on (Bäumer et al. 1996; Schneider 1996)

Type	**Description**
Presentation prototype	Supports the initiation of a project; present important aspects of the UI; illustrate how an application solves given requirements
Functional prototype, prototype proper	Temporary, executable system; implements specific, strategically important aspects of the UI and functionality; share experiences, opinions and arguments; discuss design rationale and trade-offs
Breadboard	Investigate technical aspects such as system architecture or functionality; study alternative designs to foster creativity
Pilot system	Very mature prototypes that can be practically applied

With a breadboard prototype, issues related to technical solutions or architecture should be evaluated. By comparing different design solutions, breadboards can be used to drive the creativity of the team. The fourth type of prototype in this categorization is the pilot system. A pilot system, although still classified as a prototype, is already very close to the final UI. With regards to the level of detail of UI specifications, the maturity of pilot systems is in some situations also necessary.

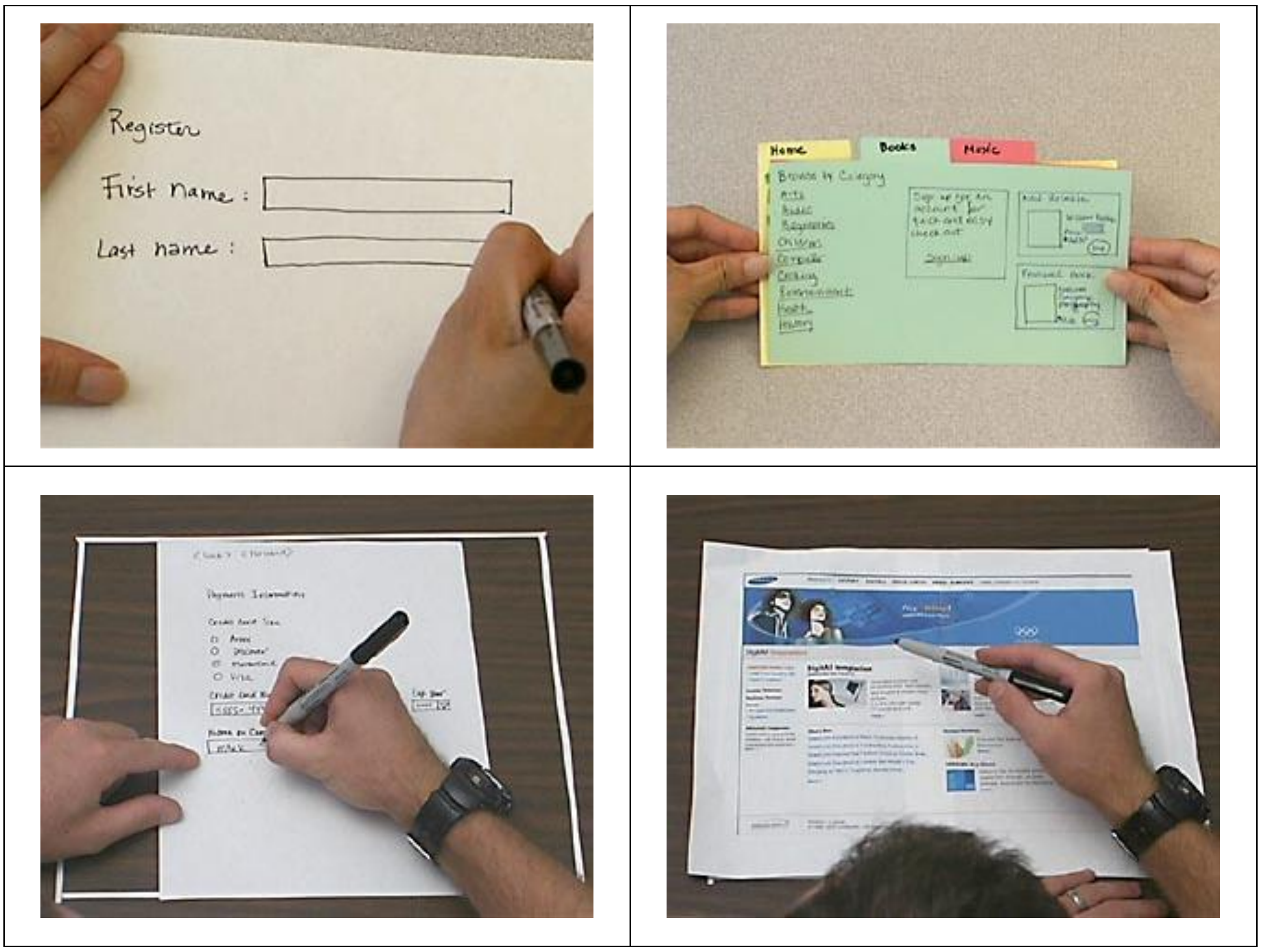


Figure 61: Card-based prototypes and paper prototypes (Nielsen Norman Group 2008)

Sketching-based prototyping

It is, however, much more important to employ other means of prototyping earlier in the process to be able to foster creativity and concurrent design. Prototyping for externalization and representing design ideas is fundamental in designing interactive systems. As an alternative to abstract prototypes such a wireframes, sketches can be used at the early stages of the UI design and specification process.

Table 53: The sketch-to-prototype continuum (Buxton 2007)

Sketch	**Prototype**
Evocative	Didactic
Suggest	Describe
Explore	Refine
Question	Answer
Propose	Test
Provoke	Resolve
Tentative	Specific
Noncommittal	Depiction

There are some important differences between prototyping and sketching, as outlined in Table 53. With tentative sketches, experts and stakeholders ask questions about the design or layout. Sketches provoke discussion and encourage consideration of alternatives or trade-offs. The more precise prototype will then be used to answer the questions. Prototypes of varying detail and focus will be used to refine the suggestions prompted by the sketch in order to make the process more specific.

Characteristics of sketches

Following (Buxton 2007), sketches have the following characteristics:

- Quick: A sketch is quick to make (and quick to change).
- Timely: a sketch can be provided when needed.
- Inexpensive: a sketch is cheap.
- Disposable: if you can't afford to throw it away, it is not a sketch; the value of sketches comes from their disposability.
- Plentiful: sketches tend not to exist in isolation; their meaning comes from the context, i.e. a collection or series of sketches (UI storyboarding).
- Clear vocabulary: the style or form signals that it is a sketch (i.e. approximate, not accurate).
- Distinct gesture: there is a fluidity to sketches that gives them a sense of openness and freedom (i.e. less concrete and formal).
- Minimal detail: include only what is required to render the intended purpose or concept; detail may be distracting; going beyond 'good enough' is a negative.
- Appropriate degree of refinement: the drawing suggests just the level of precision that corresponds to the level of certainty in the mind of the designer at the time the sketch is created.
- Suggest and explore rather than confirm: sketches don't tell, they suggest. The value of sketches is their ability to act as a catalyst for the desired and appropriate behaviours, conversations, and interactions.
- Ambiguity: sketches are intentionally ambiguous, and much of their value derives from their being able to be interpreted in different ways.

With regards to interdisciplinary UI specification, an important purpose of sketching is to simulate interpretations. In this context, sketching is a cognitive process and it is manifested through a kind of conversation between the designer(s) and their sketches (see Figure 62). The sketch represents externally an idea that is already in the head(s) of the designer(s). In contrast, the sketch can also be used to try out new ideas and to create first, incomplete visions of new designs. By looking at the sketch, the designer sees problems and relationships between the objects that have been drawn. Unintended and accidental discoveries propel the (iterative) design process.

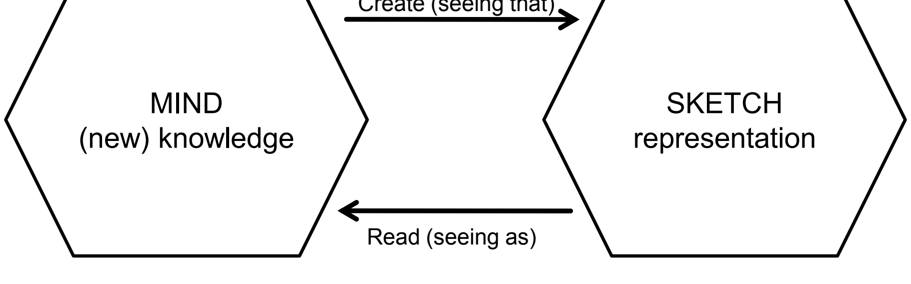


Figure 62: A sketch of a dialogue with a sketch; based on (Buxton 2007)

Sketching tools

Accordingly, many designers use sketching to externalize early design visions, and several design tools have been developed to support this function. For example, two very popular sketch-based prototyping tools are SILK (Landay 1996; Landay & Myers 2001) and DENIM (Lin et al. 2000; Lin et al. 2001; Newman 2003). SILK uses a tablet-based input device to enable easy sketching on its electronic drawing canvas. The sketchy UIs thus developed can be associated with basic functionality that enables a click-through of the UI story that has been drawn. With DENIM, the interaction designer can prototype whole dialogue flows, for example those of an entire website. Both tools will be presented in detail in the related research in Chapter 5.

A road map for prototyping?

The various ways of prototyping can easily lead to a certain degree of confusion in terms of questioning what kind of prototype should be produced at which time in the process. The approaches of (Lichter et al. 1993; Bäumer et al. 1996) presented earlier are primarily based on different ways of using prototypes in development and design situations. But these approaches do not enable the interaction designer to really understand what kind of prototype he should build.

Low- and high-fidelity prototypes

In this context, the low-fidelity versus high-fidelity debate (see Table 54) has a long history. For early-stage prototyping during RE, the degree to which the prototype accurately represents the UI design and – even more importantly – the interaction behaviour, is the determining factor guiding the development process.

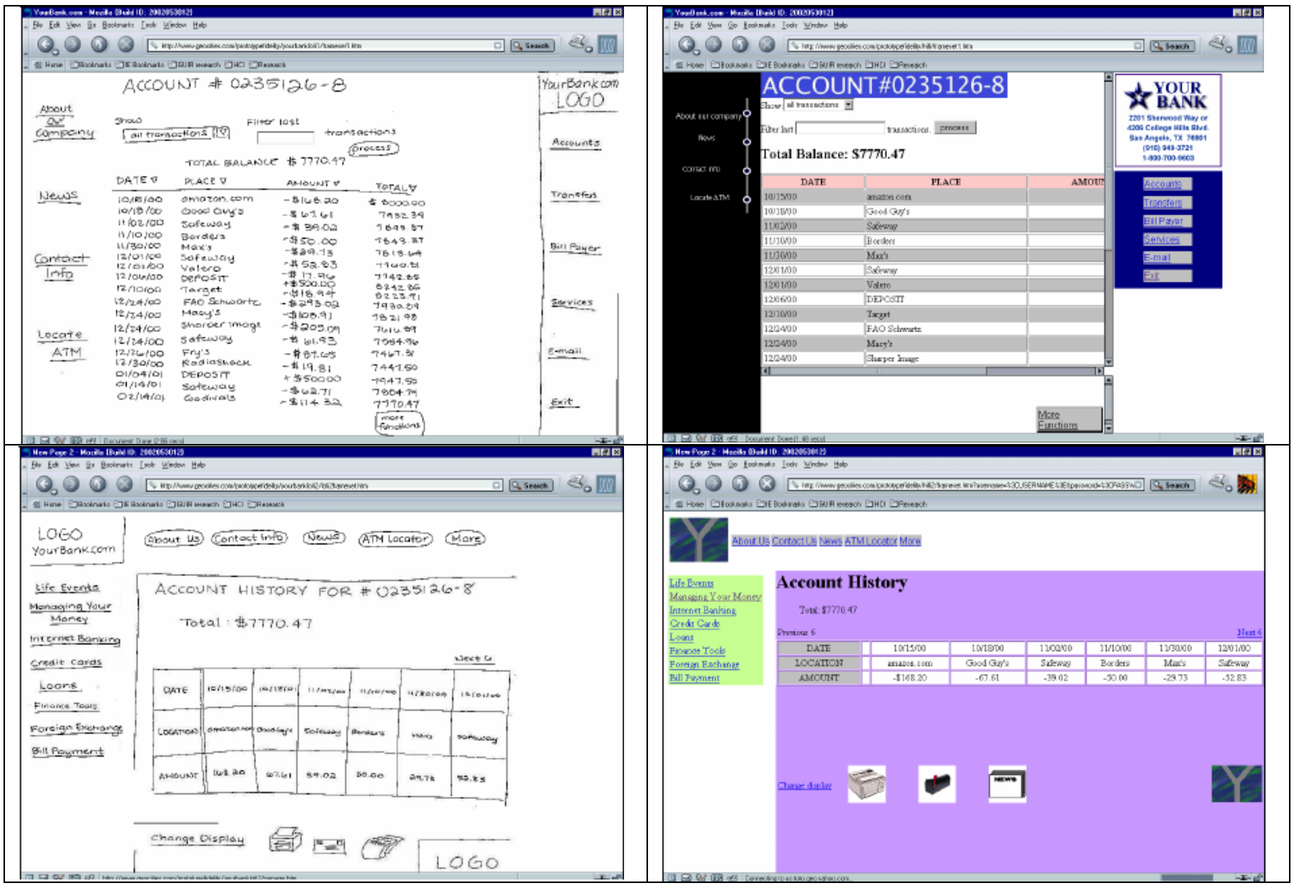
Table 54: Advantages and disadvantages of low- and high-fidelity prototyping, based on (Rudd et al. 1996)

Type	Advantages	Disadvantages
Low-Fidelity	- less time & lower cost - evaluate multiple design concepts - communication device - address screen-layout issues - identify market requirements - proof-of-concept	- limited usefulness for usability tests - poor detailed specification to code to - navigational and flow limitations - limited error checking - limited utility after requirements established
High-Fidelity	- partial/complete functionality - fully interactive - use for exploration and test - look & feel of final product - living UI specification - marketing & sales tool	- time-consuming to create - more expensive to develop - inefficient for proof-of-concept designs - blinds users to major representational flaws - management may think it is real

Abstract or low-fidelity prototypes are generally limited in function but only need limited prototyping effort. They usually do not require programming skills and coding. They are constructed to facilitate discussion of UI concepts and design alternatives, rather than to model the user interaction with a system. Low fidelity prototypes (see Figure 63, left) therefore mainly demonstrate the look, but rarely demonstrate the feel, of a UI. They will show design direction but will not provide details about how navigation is going to work or what interaction behaviour is like (Rudd et al. 1996).

Table 55: Popular low-fidelity prototyping methods (Memmel et al. 2007f)

Method	Description
Content inventories	Simple lists inventorying the information of controls to be collected within a given interaction context
Sticky notes	Visual content inventories; incorporate position and spatial relationship between UI contents
Wireframes	Schematics outline the areas occupied by interface contents
Paper prototypes, Paper mock-ups	Rough sketches of the UID; for usability studies or quick reviews; rated as fastest method of rapid prototyping
Storyboarding	Sequence of paper prototypes, e.g. arranged with users



Total Balance: $7770.47

DATE	PLACE	AMOUN
10/15/00	amazon.com	
10/18/00	Good Guy's	
11/02/00	Safeway	
11/10/00	Borders	
11/30/00	Max's	
12/01/00	Safeway	
12/01/00	Valero	
12/06/00	DEPOSIT	
12/10/00	Target	
12/24/00	FAO Schwartz	
12/24/00	Macy's	
12/24/00	Sharper Image	

Account History

Total: $7770.47

Previous 6 Next 6

DATE		10/15/00	10/18/00	11/02/00	11/10/00	11/30/00	12/01/00
LOCATION		amazon.com	Good Guy's	Safeway	Borders	Max's	Safeway
AMOUNT		-$168.20	-67.61	-39.02	-50.00	-29.73	-52.83

Figure 63: Account history pages for the two websites in both low fidelity and high fidelity. Low-fidelity websites are on the left and high-fidelity on the right. The top row is website 1 and the bottom row is website 2. From (Walker et al. 2002)

Low-fidelity prototyping methods

Among widely known low-fidelity prototyping methods (see Table 55), paper prototyping (Walker et al. 2002) is one of the cheapest and fastest visual techniques one can employ in a design process (see Figure 61 and Figure 63). It is also popular as a method for rapid prototyping (Gutierrez 1989). Low-fidelity prototypes should be used for exploring the design space. Methods such as canonical abstract prototypes (see Figure 64) and wireframes (see Figure 65) require little effort and allow a quick comparison of alternatives. They therefore support prioritizing design alternatives and narrowing the design space.

Paper, or 'Wizard of Oz', prototypes are often comprised of hand-rendered drawings on sheets of paper and generally have low levels of visual refinement. It is possible that they may be broad or deep, but somewhat uncommon that they be both. They tend to have a very low richness of interactivity, and low richness of data model. Because of these attributes, it is very difficult to use them for timed tasks, to test interactive features within an interface, or to represent the scale of the actual data space of the domain.

Shortcomings of low-fidelity prototypes

Although simpler designs can externalize design problems more economically and provide the starting point for discussion, they are too sketchy and too vague to give further guidance for developers. Moreover, they lack aesthetics and cannot represent CI and CD to business stakeholders. From a certain phase of UI design onwards, the developers therefore have to switch from abstract to detail.

High-fidelity prototyping methods

Realistic prototypes help resolve detailed design decisions in layout, visual presentation, and component selection, as well as finding issues in interaction design and interface behaviour (Constantine 2003). If a developer has to present his design visions to less experienced users, executives, or a more technical audience, *"a more robust and aesthetically invested prototype might be appropriate"* (Berkun 2000). High-fidelity prototypes range from detailed drawings to fully interactive simulations (see Table 56), which show real system behaviour rather than just presenting static screens. They address issues such as navigation and work flow, as well as the matching of design with user models (Rudd et al. 1996). High-fidelity prototypes should not be used for exploring design alternatives. Simpler designs can externalize initial design problems better and more economically. They provide the starting point for discussion and requirements engineering. But once they have helped to narrow the design space to the most promising solution(s), they are then too sketchy and vague to give guidance for developers.

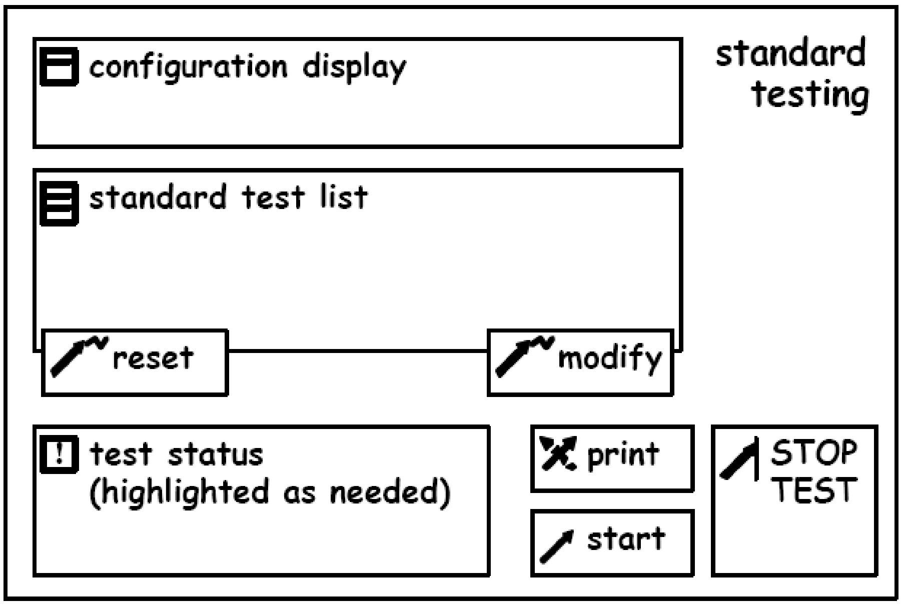


Figure 64: A (canonical) abstract prototype or wireframe; from (Constantine et al. 2003)

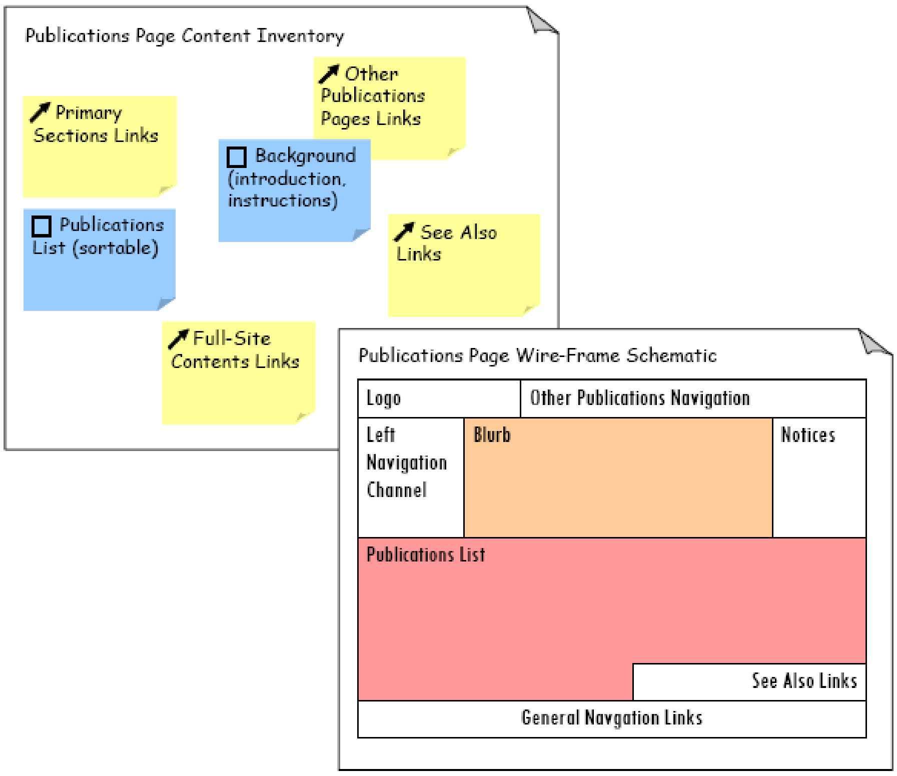
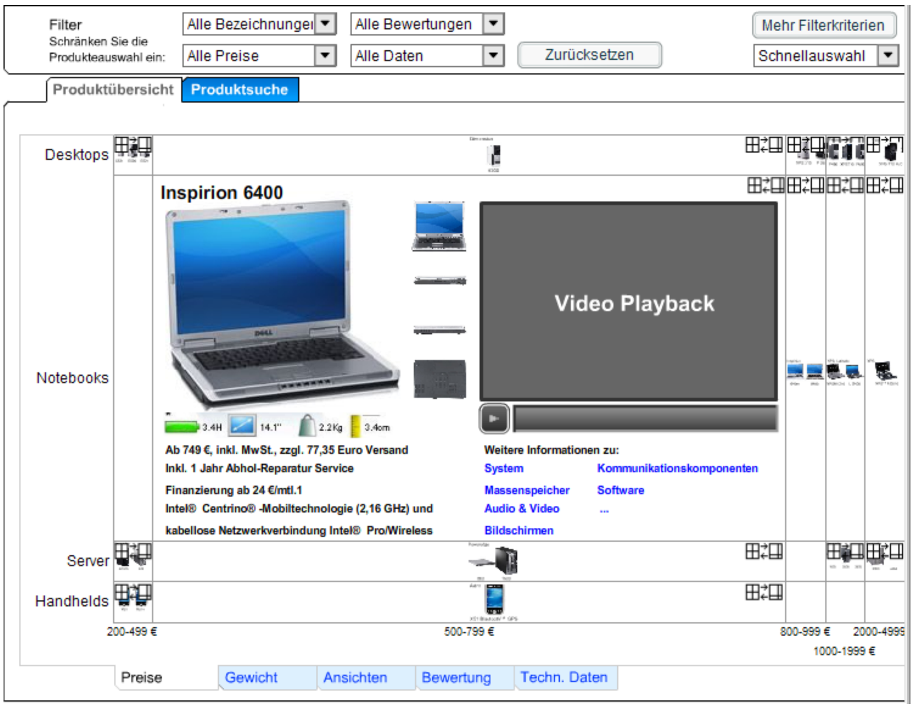

Publications Page Content Inventory
Other Publications Pages Links
Primary Sections Links
Background (introduction, instructions)
Publications List (sortable)
See Also Links
Full-Site Contents Links
Publications Page Wire-Frame Schematic
Logo
Other Publications Navigation
Left Navigation Channel
Blurb
Notices
Publications List
See Also Links
General Navgation Links

Figure 65: Content inventory (upper left) and wireframe schematic (lower right); from (Constantine 2003)



Ab 749 €, inkl. MwSt., zzgl. 77,35 Euro Versand
Inkl. 1 Jahr Abhol-Reparatur Service
Finanzierung ab 24 €/mtl.1
Intel® Centrino® -Mobiltechnologie (2,16 GHz) und kabellose Netzwerkverbindung Intel® Pro/Wireless

Weitere Informationen zu:
System
Kommunikationskomponenten
Massenspeicher
Software
Audio & Video
...
Bildschirmen

Figure 66: An example (the DELL website) of a high-fidelity prototype of a product search; from (Gundelsweiler et al. 2007b)

PowerPoint and HTML prototypes

PowerPoint or HTML prototypes are often slideshows of screens, sometimes linked together using even-triggering UI elements to simulate interactivity (see Figure 63, right). These artefacts can range from low to high in visual refinement, are likely to be either broad or deep, have low to medium richness of interactivity, and low to high richness of data (see Table 57). These prototypes are nearly as simple as paper prototypes in terms of rapid generation and modification, but offer at least some interactivity and sense of flow through the artefact (McCurdy et al. 2006).

Table 56: Overview of high-fidelity prototyping methods, partly based on (Bäumer et al. 1996), presented in (Memmel et al. 2007f)

Method	**Description**
Graphical Mock-ups	Images of the UI, e.g. created with Adobe Photoshop, Microsoft Powerpoint, HyperCards
HTML prototypes	(Partly-)Functional simulations implemented in HTML. Popular tool: Adobe Dreamweaver
Interface builders	Complete development environment for graphical design

Prototypes rendered in Adobe Flash (see Figure 66), for example, range from low to high in respect of visual resolution, and from low to high in respect of both breadth and depth. They can range from low to high in both richness of interactivity and richness of data. Prototypes built using these technologies often require higher levels of effort and expertise, but these artefacts can be designed to leverage any of the five dimensions depending on the goals of the implementers (see Table 57).

Comparing low and high fidelity

Constantine argues that filling in detail too early often leads to premature decisions. He suggests abstract prototypes (Constantine 1998; Constantine 2003), which can be associated with low-fidelity prototyping (see Figure 64). In general, detailed high-fidelity prototypes are rated as time consuming and costly, whilst more abstract and simple low-fidelity prototypes are known to be easy and cheap to produce (Greenberg 1998). And although the application of low fidelity and high fidelity prototyping is comparatively effective at detecting usability issues (Virzi et al. 1996; Walker et al. 2002; Sefelin et al. 2003), users are likely to prefer working with more detailed prototypes. They get a better feeling for how the product will behave and can therefore make more valuable recommendations about functionality and usability.

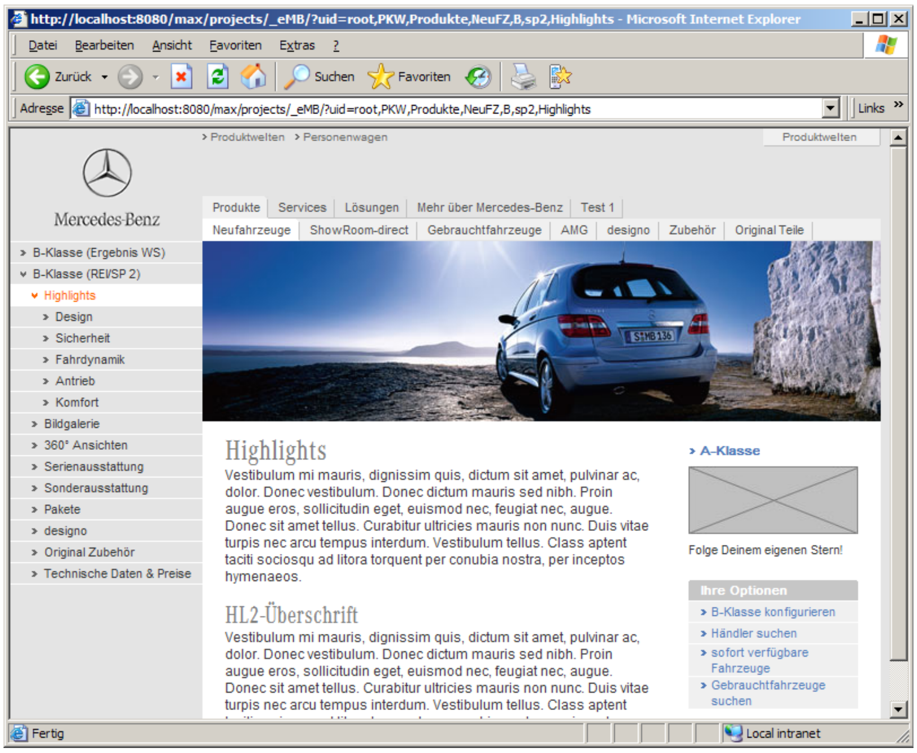
Prototypes that are visually more detailed help us to overcome the limitations of our cognitive abilities to process, develop, and maintain complex ideas and to produce a detailed operative image. This helps to discover missing, unknown and unclear requirements and reduces the risk of costly late-cycle rework. Unfeasible, undesirable, or costly UI behaviour can be identified early. Only detailed simulations allow an assessment of real UI behaviour and they are therefore particularly necessary for crosschecking the feel of the UI with the requirements.

Overcoming the fidelity barrier

Although a prototype's fidelity is an important property in guiding interaction designers in the way they build prototypes, the simple distinction of low- versus high-fidelity prototypes is often too limiting (Lim et al. 2008).

"Although the terms 'low fidelity' and 'high fidelity' are often used to characterize different prototyping approaches, the concept of 'fidelity' has a tendency to conflate several orthogonal aspects of the artefact. For example, it is unclear whether 'fidelity' refers primarily to level of functionality, level of visual polish, or level of interactivity (among others)." (McCurdy et al. 2006)*, p.2*

In order to provide a more flexible taxonomy for classifying prototypes, (McCurdy et al. 2006) identified five dimensions along which a prototype can be characterized. Each dimension presented in Table 57 can be implemented independently with low fidelity or high fidelity.


http://localhost:8080/max/projects/_eMB/?uid=root,PKW,Produkte,NeuFZ,B,sp2,Highlights - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten
Adresse http://localhost:8080/max/projects/_eMB/?uid=root,PKW,Produkte,NeuFZ,B,sp2,Highlights
Links
› Produktwelten › Personenwagen
Produktwelten
Mercedes-Benz
Produkte Services Lösungen Mehr über Mercedes-Benz Test 1
Neufahrzeuge ShowRoom-direct Gebrauchtfahrzeuge AMG designo Zubehör Original Teile
B-Klasse (Ergebnis WS)
B-Klasse (REI/SP 2)
Highlights
Design
Sicherheit
Fahrdynamik
Antrieb
Komfort
Bildgalerie
360° Ansichten
Serienausstattung
Sonderausstattung
Pakete
designo
Original Zubehör
Technische Daten & Preise
A-Klasse
Ihre Optionen
B-Klasse konfigurieren
Händler suchen
sofort verfügbare Fahrzeuge
Gebrauchtfahrzeuge suchen
Fertig
Local intranet

Highlights

Vestibulum mi mauris, dignissim quis, dictum sit amet, pulvinar ac, dolor. Donec vestibulum. Donec dictum mauris sed nibh. Proin augue eros, sollicitudin eget, euismod nec, feugiat nec, augue. Donec sit amet tellus. Curabitur ultricies mauris non nunc. Duis vitae turpis nec arcu tempus interdum. Vestibulum tellus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos.

Folge Deinem eigenen Stern!

HL2-Überschrift

Vestibulum mi mauris, dignissim quis, dictum sit amet, pulvinar ac, dolor. Donec vestibulum. Donec dictum mauris sed nibh. Proin augue eros, sollicitudin eget, euismod nec, feugiat nec, augue. Donec sit amet tellus. Curabitur ultricies mauris non nunc. Duis vitae turpis nec arcu tempus interdum. Vestibulum tellus. Class aptent

Figure 67: Mixed-fidelity prototype produced in MAXpro, a prototyping tool developed at Daimler AG (Memmel et al. 2007g; Memmel et al. 2008e)

Mixed-fidelity prototyping

Having more variety in prototyping is significantly eased by the sheer number of new tools. Recent advances in prototyping tools have made it increasingly easy to create mixed-fidelity prototypes that are high fidelity on some of these dimensions and low fidelity on others. These include not only specialized toolkits such as SILK and DENIM (see Chapter 5), but also commercial off-the-shelf software such as Adobe Flash, or domain-specific prototyping environments such as MAXpro (Memmel et al. 2007g; Memmel et al. 2008e), developed at Daimler AG for the rapid prototyping of websites (see Figure 67). These tools make it easier to create more targeted prototypes and to avoid spending resources on areas that are not in the focus of the prototyping effort.

Table 57: Five dimensions for prototyping to overcome the limitations of the fidelity approach (McCurdy et al. 2006)

Level of Visual Refinement:	How refined should the prototype be from a visual standpoint? Artefacts at the low end of this scale include hand-drawn sketches and box-and-line wireframes. Artefacts at the high end include fully resolved, pixel-accurate mock-ups
Breadth of Functionality	How broadly is the functionality represented within the prototype? A broadly functional prototype gives users a better understanding of the range of capabilities that the interface will ultimately provide, and offers the opportunity to challenge system-wide issues
Depth of Functionality	To what level of detail is any one feature or sequence represented? One could imagine having a single path through the interface modelled in the prototype all the way though to its conclusion. Having a task modelled to its conclusion allows designers to interrogate the interface's capabilities with user evaluations such as think-aloud studies and cognitive walkthroughs.
Richness of Interactivity	How are the interactive elements (transitions, system responses to user inputs, etc.) captured and represented to the user by the prototype? Paper prototypes and sketches have traditionally represented the lowest fidelity with respect to interactivity, although efforts such as SILK and DENIM have been explicitly designed to increase the interactive richness of hand-drawn interfaces. Higher levels of interactivity have historically come at the cost of development expense, time, and inflexibility.
Richness of Data Model	How representative of the actual domain data is the data employed by the prototype?

The anatomy of prototypes

The different fidelities of prototyping are supposed to support the interaction designer during UI design and UI specification. By choosing a certain level of detail for a prototype, the interaction designer intends to be able to focus on what interests him the most or what might invite stakeholders to participate to the greatest extent possible in the process of design and decision-making. Accordingly, the prototype is in charge of focusing and filtering (see Figure 68), without distorting the understanding of the whole.

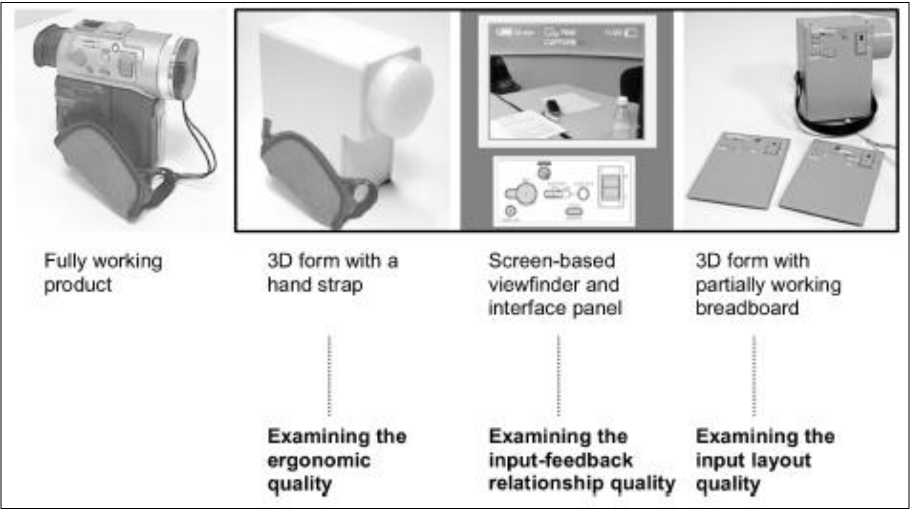

Fully working product
3D form with a hand strap
Screen-based viewfinder and interface panel
3D form with partially working breadboard
Examining the ergonomic quality
Examining the input-feedback relationship quality
Examining the input layout quality

Figure 68: Prototypes that represent different qualities of interest to a designer in order to filter out different aspects of a design; from (Lim et al. 2008)

(Lim et al. 2008) describe this as a fundamental prototyping principle and propose a framework to define the anatomy of prototypes more precisely, and not just through the dimension of fidelity. Two main aspects define their framework of prototyping classification. Firstly, prototypes are for traversing a design space, leading to the creation of meaningful knowledge about the final design as envisioned in the process of design and, secondly, prototypes are purposefully formed manifestations of design ideas.

Prototypes as filters

For traversing a design space, stakeholders need inspiring artefacts. Prototypes can be inspiring and support creativity simply through their incompleteness. A lack of completeness forces everyone who is thinking about the design to consider various ideas. Playing around with ideas and artefacts is, in turn, just what can lead to astonishingly innovative results (Schrage 1999). And, *"when incomplete, a prototype reveals certain aspects of a design idea – this is, it filters certain qualities."* (Lim et al. 2008), p. 7.

Consequently, in order to be a valuable support, prototypes that drive the innovative and experimental process of design should be as incomplete as possible. At the same time, the prototype should still help to filter those qualities of a product that need to be explored and examined. A major advantage of filtering is a reduction of complexity. The interaction designer can focus on single aspects of a product, which usually decreases the anxiety of 'not seeing the wood for the trees'. Naturally, for UI specification, incomplete prototypes are not enough. In particular, the layer of UI design, which has to determine how the UI of the software product must look, needs to be relatively complete. But as a means for driving creativity and taking different designs into account seriously, the UI specification process must also include incomplete prototypes. In the end, they are the artefacts that carry an enormous amount of design knowledge and design rationale along the supply chain. With regards to the fidelity of prototypes, incompleteness can certainly be reached by both low- and high-fidelity prototypes.

Filtering dimensions

(Lim et al. 2008) propose five different filtering dimensions (see Table 58) that cover all aspects of the prototyping of interactive systems. The appearance dimension addresses physical properties of a design. The data dimension is the information architecture and the data model of a design. The functionality dimension covers the functions that can be performed by the design. Designers could, for example, develop different scenarios in order to address different functions with different prototypes. The interactivity dimension addresses many aspects of HCI, for example feedback and behaviour. The spatial-structure dimension describes how components of the prototype are related to each other. This includes the layout of information items or UI widgets.

Table 58: Variables that influence a prototype's filtering capabilities; from (Lim et al. 2008)

Filtering Dimension	**Example Variables**
Appearance	Size, colour, shape, margin, form, weight, texture, proportion, hardness, transparency, gradation, haptic, sound
Data	Data size, data type, data use, privacy type, hierarchy, organization
Functionality	System function, users' functionality need
Interactivity	Input behaviour, output behaviour, feedback behaviour, information behaviour
Spatial structure	Arrangement of interface or information elements; relationship between interface or information elements (2D, 3D), (in-)tangible, or mixed

The correlation of filtering dimensions

All dimensions mentioned usually affect each other. A decision on the data dimension, for example, can have an impact on the interactivity of a UI, which in turn can affect the overall appearance of a software product (see Figure 69). But although several dimensions might be closely linked, each dimension should be taken care of separately. This means, for example, that different prototypes should focus on one individual filter in the beginning, in order to make sure that the individual concepts are consistent, complete and sound on their own. With usability issues, it is often the case that 'the devil is in the details' and other features of a product might deflect from important aspects of the UI.

Prototypes as manifestations of design

Through the process of making prototypes, interaction designers constantly evaluate their ideas. As with the iPod (see Figure 69), groundbreaking good designs are seldom conceived at the first attempt. Design is a continuous coupling of internal activities and external realization activities. With a framework for UI specification and a common denominator for UI-related modelling, we intend to have as much externalization as possible, (Lim et al. 2008) explain that the process of design is indeed very much affected by iterative interaction with external manifestations of thinking. Such external entities can be models, as presented in the beginning of this chapter, or UI prototypes.

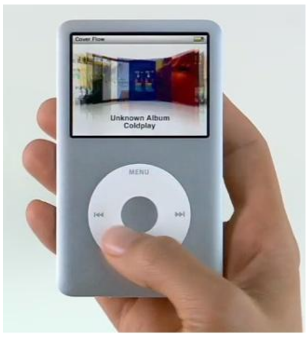
		
Data: several thousand songs can be accessed	Interactivity: with the wheel drive, songs can be accessed easily and quickly; with the wheel, the user can select songs, adjust volume, etc	Appearance: the physical wheel (size, haptic), display and visualization of information are all affected by the wheel interaction.

Figure 69: The correlation of filtering dimensions using the example of the Apple iPod (images from apple.com)

For a better engineering experience, the external artefacts function as an 'extended mind'. Sketching, being a widespread and tool-supported activity in prototyping, extends the designer's internal memory, for example. The paper preserves design ideas and allows the designer and others to re-perceive the created artefacts. In order that the world can speak back to us (Schön 1987), *"the act of bringing thoughts into material form is not incidental to the act of creation but is itself constitutive of and essential to creation."* (Clark 2001), p.9. For UI specification, the externalization of design thought can take different forms. Considering the need for electronic tool support, however, all forms of manifestation have to happen on digital material. There is, therefore, much less restriction compared to physical materials, which in turn leads to a larger design space.

Table 59: Variables of dimension of manifestation; from (Lim et al. 2008)

Manifestation dimension	**Definition**	**Example Variables**
Material	Medium (either visible or invisible)	Physical media, e.g., paper, wood and plastic; tools for manipulating physical matters; computational prototyping tools, e.g. Adobe Flash
Resolution (or fidelity)	Level of detail or sophistication of what is manifested	Accuracy of performance, e.g., feedback time responding to an input by a user – giving user feedback in a paper prototype is slower than in a computer-based one; appearance details; interactivity details; realistic versus faked data
Scope	Range of what is to be manifested	Level of contextualization

Dimensions of manifestation and filtering

With the manifestation dimensions, the anatomy framework of (Lim et al. 2008) gives advice on how to form prototypes. This includes the material, the resolution and the scope of the prototype (see Table 59). The resolution of the prototype is similar to the differentiation of fidelity, as discussed earlier in this chapter. The meaning of scope addresses the focus of a prototype, i.e. the main purpose of building the prototype. For example, if the designer of the iPod tried out different colour schemes to decide which one best fits the device's screen, he might well have built prototypes with the clear scope of colouring.

4.2.2 Specification-Level User Interface Prototyping

Economics of prototyping

Bearing in mind the wide variety of dimensions that tend to determine the quality, purpose and appearance of a prototype, for example, it is necessary to consider the right form of prototyping for the purpose of UI specification. Of course, prototypes usually differ significantly from the end-product, as they are not the ultimate target of the design process. Normally, therefore, the interaction designer's mindset is different from that of the person who is responsible for coding the final UI. The economic principle of prototyping accordingly states that the best prototype is the one that, in the simplest and most effective way, makes the possibilities and limitations of a design idea visible and measurable. In the context of UI specification, this demands a justification for the incorporation of rather abstract prototypes as a means to communicate design knowledge and rational. For the purpose of UI specification, however, very highly detailed and accurate prototypes also need to be integrated. The principle of multi-fidelity is explained well by the CAMELEON reference framework (Calvary et al. 2003).

The CAMELEON reference framework

For identifying the right prototypologies for UI specification, the CAMELEON reference framework provides a sound structure. The framework proposes four levels of abstraction for UI tools: tasks and concepts, abstract UI design, detailed UI design, and the final UI. The first layer, tasks and concepts, is related to the idea presented here of incorporating different models (see Chapter 4.1). The other levels address the requirement to have different levels of UI presentation, each having its own important role in the UI specification process (see Table 60).

From the abstract to the concrete

An abstract UI is a canonical expression of the rendering of the domain concepts and functions in a way that is independent of the interactions available on the targets. An abstract UI can also be a sketch, as discussed in Chapter 4.2.1. A concrete UI turns an abstract UI into an interactor-dependent expression. Although a concrete UI makes explicit the look and feel of the final UI, it is still a mock-up. The final UI can be generated from a concrete UI, and can be expressed, for example, in source code such as Java, HTML or XML. It can then be interpreted or compiled as a pre-computed UI and plugged into a run-time environment (Calvary et al. 2003). It is, however, important to distinguish the final UI, as defined in the CAMELEON reference framework, from the final UI as software developers would program it. The last stage of the CAMELEON process is still not the completely coded UI, but rather a collection of UI elements and designs that is able to precisely guide the software developer in coding the UI. The code templates provided in this way can mean a significant gain in efficiency.

Table 60. A comparison of the levels of UI detail of the CAMELON reference framework and interactive UI specifications

	CAMELEON Reference Framework	**Interactive UI Specification**
Abstract UI	Canonical expression	Sketch; drawing
Concrete UI	Explicitly show the final look and feel of the final UI, but still in a static and mock-up fashion	Low to mixed fidelity
Final UI	(Automatically) generated from a concrete UI; expressed in source code; but still a pre-version of the programmed UI	Specification-level design that guides programmers; expressed in a reusable markup language (e.g. XAML)

Levels of prototyping for UI specification

Accordingly, a UI specification has to incorporate both abstract and concrete UIs. Moreover, it would be a great advantage in terms of reusability and conservation of precision if the final UI could also be generated from the concrete UI. With regards to the scope of UI specification and the participating disciplines in the industry, this should happen in the background and without any need for coding. As there needs to be a very detailed design that precisely formalizes the desired look and feel of the software product, we integrate all three layers of the CAMELEON reference framework into the concept of interactive UI specification (see Table 60). In this context, however, the final UI is not generated automatically. The final UI design level is rather the most mature externalization and it serves as the authoritative part of the UI specification that will later guide programmers. But similarly to the idea of (Calvary et al. 2003), it should also be possible to present the final UI design in a code-like fashion, for example by translation into a XML description language.

Defining forms of prototyping for UI specification

In Table 61 we summarize the kinds of prototypes we propose for interdisciplinary UI specification on top of the semi-formal models for user, task and behaviour modelling. With regards to UI specification, the first four of the five dimensions of prototyping defined by (McCurdy et al. 2006) are taken into account. The level of fidelity is said to be low if the requirements representation only partially evokes the final UI without representing it in full detail. Between high fidelity and low fidelity, there is medium fidelity. Normally, UI requirements only involve one representation type, i.e. one fidelity level at a time. But due to the variety of actors' inputs, several levels of fidelity could usefully be combined together, thus leading to the concept of mixed-fidelity. Ultimately, three different stages of UI prototyping for UI specification can be determined. Each stage can be named according to the level of fidelity it is supposed to have. With regards to the anatomy framework of prototypes, however, fidelity is not just related to the resolution of the prototype. The prototypes are therefore mapped to the dimensions of the anatomy framework of prototyping in order to explain the characteristics of each prototyping stage in detail.

Table 61: Characteristics of prototypes used for UI specification

Dimension	**Low-Fidelity/Abstract Prototype**	**Mixed-Fidelity Prototype**	**Detailed/Specification-Level Prototype**
Appearance	Sketch; wireframe; canonical	Mixture of sketch and UI widgets	Concrete UI, detailed UI elements, UI widgets
Data	Various, depending on application domain	Various, depending on application domain	Various, depending on application domain
Functionality	Point and click	Point and click, lists, texts, scrolling	Mature interactive behaviour
Interactivity	Linking	Linking, navigation structure; animations (Flash)	Linking, navigation structure, animations (Flash), widget behaviour
Spatial structure	2D	2D	2D
Material	Electronic sketching; ePaper-based prototypes	Electronic	Electronic; code or mark-up description language
Resolution (or fidelity)	Sketches or abstract elements	Mixture of abstract elements and concrete widgets	Detailed UI elements
Scope	Drive design thinking; guide design through incompleteness and abstraction; have wide design space	Approximation to final UI design; communication tool; support shrinking design space	Precise formalization of UI design; guide design to ultimate solution

Abstract or low-fidelity prototypes

The first stage is called a low-fidelity or abstract prototyping layer. This stage is required to allow the interaction designers as well as the other stakeholders to sketch early design ideas and to conduct sessions of participatory design. Incomplete and abstract prototypes, for example in the shape of wireframes, help to identify design issues and guide design thinking accordingly. In many design processes, abstract prototypes such as paper prototypes are created on physical material. With regards to corporate UI specification, even abstract prototypes need to be developed with electronic support. Teams are usually dispersed and require computer-supported artefacts that can be easily shared and forwarded. It is therefore an important requirement for UI specification tool support that means be provided for electronic sketching, for example with a pen device and a drawing board. As successfully demonstrated by (Lin 1999; Lin et al. 2000; Lin et al. 2001; Lin 2003a), electronic abstract prototypes can already be linked to expressive UI storyboards and in this way, even sketchy UI elements are able to hint at the desired UI behaviour.

Mixed-fidelity prototypes

The second stage is shaped by mixed-fidelity prototypes, which are defined by a mixture of sketchy- and concrete-content elements. Through the incorporation of concrete elements such as UI widgets, user interaction with realistic buttons or lists can take place. With regards to the UI specification process, mixed-fidelity prototypes help to shrink the design space towards the most promising solution. While sketchy elements still propel discussion among stakeholders, the concrete UI elements guide the specification team in choosing the most appropriate widgets.

Detailed specification-level prototypes

The most mature stage of prototyping is determined by detailed prototypes. Their appearance defines how the UI has to look and feel. The IT supplier will use the specification-level prototype as a living UI specification, which is popped up to guide the implementation. As outlined in Chapter 3, the specification-level prototype will be more important than text-based specification documents. It therefore has to be precise and incorporate detailed representations of desired layouts, widgets and other means of interactivity. The specification-level prototype must be able to include concrete objects, which help to express the intended look and feel. In the web domain, for example, Adobe Flash clips could be integrated in the detailed prototype. For desktop applications, detailed components based on Microsoft Silverlight technology could be built into the prototype. The advantage is that the Silverlight component is based on the same technology that is used in the final implementation, which makes the reuse of high-detail components of the prototype possible. Accordingly, the specification-level prototype should provide an export mechanism that makes it possible to base the coding on its mature presentation.

4.3 The Common Denominator for Corporate UI Specification

The network of artefacts

With the identification of modelling languages and prototyping techniques that are suitable for a UI specification, the common denominator for the definition of interactive UI specifications (see Chapter 3) can be summarized. From the different stages of UI modelling, a sequence of models and design artefacts is established. Figure 70 shows the visual representation of the derivation of the common denominator. Each discipline contributes several modelling notations, of which a maximum of two notations per level is chosen as a shared means for UI specification. The matrix of specification artefacts that has been defined in this manner has two dimensions. On the vertical axis in Figure 70 we distinguish the models according to their level of abstraction. Models at the bottom are more abstract (i.e. text-based, pictorial), whereas those at upper levels become more detailed and 'design-like' with regards to the specification of the UI. On the horizontal axis, we identify appropriate models for UI specification. Accordingly, we differentiate between the grade of formality of the models and their purpose and expressivity. Models with a comparable right to exist are arranged at the same level. At each stage we identify a common denominator for all three disciplines as a part of the evolving interactive UI specification. The choice of models and prototyping techniques is based on the requirement of having permanent transparency of requirements analysis and design thinking, and continuous traceability while travelling from requirements to solutions and back. Consequently, there is an opportunity to create a network of artefacts that can be linked and assembled to a UI specification space that can be well traversed and well understood by stakeholders of all kinds.

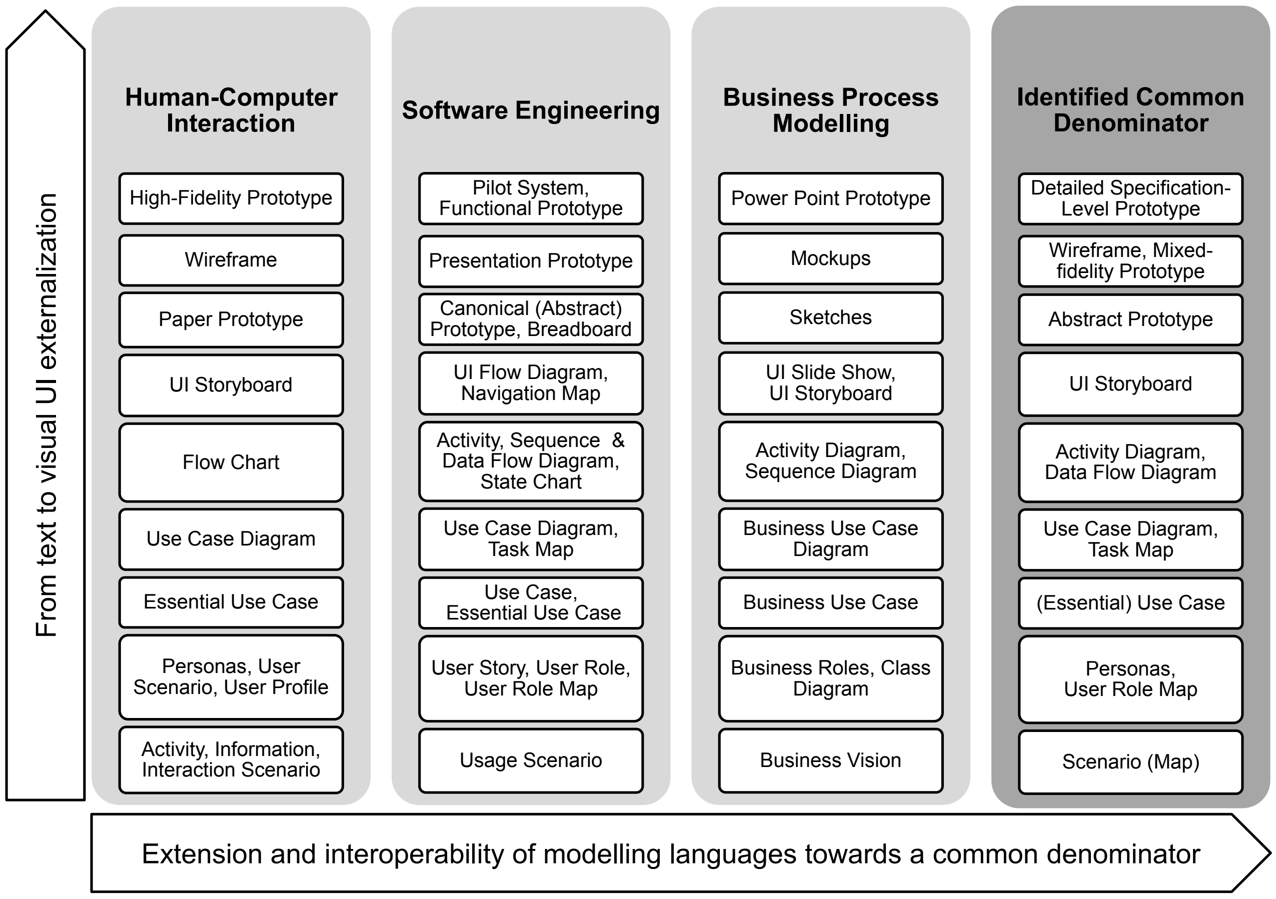

From text to visual UI externalization
Human-Computer Interaction
High-Fidelity Prototype
Wireframe
Paper Prototype
UI Storyboard
Flow Chart
Use Case Diagram
Essential Use Case
Personas, User Scenario, User Profile
Activity, Information, Interaction Scenario
Software Engineering
Pilot System, Functional Prototype
Presentation Prototype
Canonical (Abstract) Prototype, Breadboard
UI Flow Diagram, Navigation Map
Activity, Sequence & Data Flow Diagram, State Chart
Use Case Diagram, Task Map
Use Case, Essential Use Case
User Story, User Role, User Role Map
Usage Scenario
Business Process Modelling
Power Point Prototype
Mockups
Sketches
UI Slide Show, UI Storyboard
Activity Diagram, Sequence Diagram
Business Use Case Diagram
Business Use Case
Business Roles, Class Diagram
Business Vision
Identified Common Denominator
Detailed Specification-Level Prototype
Wireframe, Mixed-fidelity Prototype
Abstract Prototype
UI Storyboard
Activity Diagram, Data Flow Diagram
Use Case Diagram, Task Map
(Essential) Use Case
Personas, User Role Map
Scenario (Map)
Extension and interoperability of modelling languages towards a common denominator

Figure 70: The common denominator in interdisciplinary UI-related modelling

Scenario maps

Navigation maps turned out to be especially suited to modelling the transitions between different interaction spaces. As discussed in Chapter 4.1, problem scenarios are very helpful in initializing the UI specification process by describing the application and problem-domain and by determining specific business goals and design visions. For interactive UI specification, a 'scenario map' is proposed as the potential starting point for modelling. It is related to the navigation map in terms of the externalization of relationships between different interaction spaces. With the scenario map, the UI specification space is therefore cut into small and manageable pieces that need to be modelled and designed. In order to describe the intended design and the envisioned functionality, text-based problem scenarios are directly associated with the scenario map to explain the purpose and goals of each interaction space.

User roles and personas

For UI modelling, personas guide the UI specification process by raising empathy with user needs. They are important work artefacts that will be applied throughout the UI specification supply chain for crosschecking design solutions against user needs and user goals. Their text-based format is related to writing scenarios, but the templates proposed in IxD literature provide a structured basis for later stages of UI specification. In order to establish relationships between users that have been modelled and personas that have been developed, user-role maps are employed for the visualization of associations and interdependencies. At this stage, IxD can adopt a very important model from the usage-centred design approaches of SE, which are actually hybrid processes that are intended to bridge software development and IxD.

Essential use-cases

Essential use-cases are also text-based, but distinguish user intention and system responsibility in a well-defined style. They help to keep the focus on the UI in task modelling. With regards to the desired network of artefacts, essential use-cases can on the one hand be related to each and be divided into interaction patterns small enough to be well modelled. On the other hand, essential use-cases are the detailed description of task execution that is missed in use-case diagrams or task map.

Use-case diagrams and task maps

The use-case diagram, in turn, integrates user roles and cases in an overview, from which the modeller can drill-down into the detail. The interrelationship between many different tasks can be outsourced from the use-case diagram and visualized in a task map. The task map can show the split of a large case into smaller parts, including the presentation of associations. It therefore removes some load from the use-case diagram, which no longer has to be the all-inclusive model that is suitable for everything. Both notations are more anchored in the SE and BPM community, although development paradigms such as usage-centred design have already attracted the attention of IxD for engineering approaches to UI design. Ultimately, the common denominator requires the interaction designer to learn such notations, while the software developer, for example, has to become comfortable with the detailed user descriptions recorded in personas.

Activity and data-flow diagram

During later stages of UI specification, the UI flow can be well modelled with activity diagrams. Data-flow diagrams assist in modelling the dependencies of the UI on data providers and vice versa. Pragmatic and approximate agile models ease the process of creating such artefacts for non-technicians. Because the flow charts known in IxD are closely related to activity diagrams, the threshold in switching over to this kind of notation should be low for interaction designers. As both of these flow diagrams model the interaction events on a rather abstract level, an interactive UI specification needs to incorporate UI storyboards as well.

UI storyboards

UI storyboards help in thinking through the course of events. They are assisted by sketchy to high-detail design solutions. Because UI storyboarding depends on UI prototyping, it has the role of a mediator in UI specification. As explained in detail in Chapter 6, the UI storyboard can help to switch between models and designs.

Low- to high-fidelity prototypes

As a result, the levels of prototyping fidelity discussed here support the UI specification process with a mixture of imprecise sketches that propel discussion and design thinking, and concrete and precise designs that determine the final UI design. The specification-level design will be the entry point for stakeholders, who like to experience and evaluate the UI specification. It serves as the surface of the specification process, which on demand reveals more information about decision-making and alternative solutions.

4.4 Key Points

- For UI modelling, the disciplines of SE, IxD and BPM recognize a variety of notations that have to be filtered with regards to their applicability in a network of modelling languages. The derived common denominator for UI specification is grounded upon agile and semi-formal notations that allow all stakeholders to take part in the modelling and UI design process.
- For modelling the problem-domain, text-based scenarios that also incorporate the description of corporate values, core purposes and visionary goals are the appropriate means for interdisciplinary UI specification at the early stages.
- Interdisciplinary user modelling should make use of personas, because they are able to raise empathy and understanding for the real end-users. The fictitious descriptions of users are superior to abstract user stories, as designing for user experience demands the identification of stakeholders with potential users and customers. In this context, role maps can interrelate personas well, and they provide an overview of the users that have to be taken into account. Accordingly, individual profiles are merged to personas, which function as a proxy for them and which is then referenced in the role-map model.
- The common denominator for task modelling can be defined by essential use-cases and task maps. The essential use-case describes task-related interaction patterns in detail, while the task map supports the modelling of an overview of all relevant cases. The use-case diagram assists in linking tasks to user roles or personas, and vice versa. With regards to a shared means of modelling, all three notations have a very low application threshold for experts across the disciplines.
- Activity and data-flow diagrams are well suited to modelling the behavioural view of UI requirements. While SE and BPM already apply them on a regular basis, IxD must switch from flow charts to a somewhat object-oriented kind of notation. However, the threshold for changing to activity and data-flow diagrams is low, because the notations are very similar and can be kept approximate and pragmatic. This prevents the ‘fear of contact’ that generally exists with the notations of UML.
- UI storyboards are telling a story about UI flow and help to set up the optimal interaction sequence. The UI storyboard visually shapes UI requirements and is closely related to UI prototyping. It is therefore very suitable for bridging modelling and UI design with a mediating notation.
- Prototyping for UI specification usually has different goals, each of them expressed with a different prototypology. With regards to the CAMELEON reference framework, abstract, concrete and final representations of the UI are to be incorporated into interactive UI specifications. Sketchy designs support creative thinking and can be saved as artefacts that determine the starting point of UI design. Mixed-fidelity prototypes make the design more concrete, but still come with approximate elements. The detailed designs then shape the UI on the most mature level and function as the uppermost layer of the UI specification.
- A common denominator for corporate UI specification can be identified without the development of new modelling languages. A concentration on existing notations and the careful selection of those models that are well related helps to determine a common course of action. The creation of a network of interrelated modelling languages and design artefacts bridges the gaps between the disciplines and makes the UI specification process more tangible, traceable and transparent.

Chapter 5 Related Work

"By 2020 all business software will be visualized prior to development, the same way that visualization is a common practice in the design of every car, airplane and semiconductor today" (iRise 2008)

Introduction

Through the examination of UI development processes in general and the required ingredients of corporate (interactive) UI specification in particular, the previous chapters underlined that there is no conflict between model-based UI design and classical user-centred design. This is because modern model-based design also has a strong focus on task, domain and user models (see Chapter 4). There are many important characteristics of models in general, that makes it desirable to combine model-based design and UI design in corporate UI specification. The appropriate tool support should assist in moving between abstract and concrete representations, and change the level of completeness and formality according to the needs of the UI specification process. Because interdisciplinary UI specification with multiple stakeholders is best supported by flexible and semi-formal representations, the analysis of related work focuses on similar modelling approaches and the appropriate UI tools.

Extensive tool analysis

(König 2008) categorized several software applications into tools for RE (e.g. Axure Pro, Borland Caliber, IBM Rational, iRise Studio, Serena Composer, Telelogic Doors), for formal modelling (e.g. EclipseUML, iUML, MetaEdit+), for UI construction (e.g. Microsoft Expression Blend, Adobe Designer), for UI prototyping (e.g. MS Visio, JavaSketchIt), for widget and general UI design (e.g. Adobe Photoshop, Corel Draw), and for UI animation (e.g. Adobe Flash, Adobe Flex). To accord with the capabilities of most interaction designers and other actors in UI specification processes as described earlier, an appropriate tool must avoid the need for coding if it is to become popular and widely accepted, and to be commonly used. Besides abstract prototyping, detailed simulations must be possible to evaluate UI behaviour (Memmel et al. 2007c). Modelling should provide room for informality and must be decoupled from UML to be applicable to all stakeholders. Moreover, the deduction of design solutions from abstract models must be traceable and transparent. (König 2008) concludes that only a very few UI tools currently support corporate UI specification tasks. While some UI tools just allow high-fidelity UI prototyping and facilitate simulations of the later system, other tools such as IBM Rational are packed with features far beyond the needs and capabilities of most actors who take part in UI specification. In contrast, in some otherwise promising UI tools the necessary artefacts of early-stage modelling, such as personas, essential use-cases, or task maps are missing. This prevents current tools from adequately supporting the thought processes and design tasks that have to be accomplished in order to create usable, effective, and innovative UIs (Memmel et al. 2008b).

The most promising solutions

All in all, there is no sign of any compromise between overwhelming IxD experts with features and formality on the one hand, and restricting their handcraft to UI prototyping on the other (Memmel et al. 2007c). Accordingly, there is as yet no silver bullet UI specification tool that perfectly addresses the requirements of corporate UI development. But in order to analyze the best features and approaches currently available, two commercial and nine scientific tools (see Table 62) that could potentially be applied in prototyping-driven UI specification will be discussed in this chapter. Because an integrated tool that supports a modelling and design process based on a common denominator is still missing, the tools analyzed and the corresponding methodological approaches mainly represent partial solutions.

Commercial tools

Commercial tools, particularly iRise and Axure, are currently extremely successful and they became widely applied in the non-IT industry for helping stakeholders of all kinds in UI prototyping. The main philosophy of these tools is to provide means of design that do not require the stakeholders to learn new means of expression. As with prototyping with PowerPoint, which is the most common corporate tool for UI design (see Chapter 3), with iRise and Axure the user does not need to write a single line of code. However, both tools are very much focused on UI design and lack adequate means to model the UI requirements in an expressive way. Ultimately, the analysis of commercial tools helps to underline the requirements for adequate tool support and the corresponding underlying modelling approach.

Table 62: Overview of tool support and related modelling approaches

Name	Derivation	Type of tool support
Axure, iRise	Commercial Product	Prototyping-driven UI specification tools that support collaboration of many stakeholders and disciplines
CanonSketch, TaskSketch, WinSketch, MetaSketch	Research Project	Modelling and UI prototyping tools that are intended to increase traceability of requirements; strongly related to usage-centred design and model-driven development
SILK, DENIM, DAMASK	Research Project	Sketching and storyboarding tools that support the early stages of design in model-based UI development
Diamodl	Research Project	Model-based approach that utilizes mock-ups of the UI and connects them with flowcharts that reveal the underlying functionality
SketchiXML, GrafiXML	Research Project	UI specification tools based on usiXML and the CAMELEON reference framework for model-driven UI development

Research projects

In the context of UI specification, several research projects also contributed some valuable results. Tools such as SILK or DENIM give particular support to the early stages of UI specification. They enable the computer-aided sketching of single screens as well as UI storyboards. The analysis of these tools will highlight the added value of sketches that are integrated in (interactive) UI specifications. Accordingly, the discussion of both tools links to the theoretic examination of sketch-based prototyping in Chapter 4. The research group of Vanderdonckt (Vanderdonckt 2008) works intensively on tools that are based on the CAMELEON reference framework (see Chapter 4). With SketchiXML and GrafiXML, the work group links to pragmatic tools that help stakeholders in designing the UI. The advantage of usiXML is the exchangeability of artefacts. Both tools are therefore noteworthy and contribute fundamental ideas for sharing work artefacts. DIAMODL, in turn, aims to support UI experts in modelling UI-related requirements and helps to link UI models to UI design. The discussion of this kind of tool support depends on the examination of semi-formal approaches to UI modelling and analyzes the need for developing models.

CanonSketch, TaskSketch and WinSketch

Tools such as CanonSketch, TaskSketch and WinSketch are the most interesting solutions evaluated in this chapter. They represent approaches that try to combine different means of modelling with different levels of UI design. They are therefore the existing tools that are the most closely related to the common denominator for UI specification and the corresponding interrelationships of UI-related artefacts. The analysis of these tools will focus especially on the kind of models that can be developed and the connectivity of models and design. The goal of these tools is to ease the transitions in design practice by exploiting relationships between artefacts and consequently fostering traceability. In 2008, the history of these tools is being carried forward by MetaSketch, which combines usage-centred design with the concept of meta-modelling.

Wisdom vs. Common denominator

Because of the close relationship of CanonSketch, TaskSketch and WinSketch, as well as the Wisdom approach to the common denominator (see Chapter 4) and INSPECTOR (see Chapter 6), the overview of related work ends with a detailed comparison of both UI specification suites.

5.1 Axure Pro and iRise Studio

As outlined in Chapter 3, non-IT organizations lack adequate means to manage their UI specification processes. A new generation of prototyping and UI specification tools is trying to address this shortcoming. The most popular representatives of this group of tools are Axure Pro and iRise Studio, and iRise is the more sophisticated and will be discussed in some detail.

Substitute text-based specifications

Paralleling the discussion in this thesis, both software companies realized the overwhelming nature of text-based specification documents that tend to incorporate all the information gathered along the supply chain of the software development (see Figure 71). Accordingly, both Axure Pro and iRise Studio try to substitute textual descriptions of look and feel with living UI prototypes. With regards to the ingredients of complete UI specification documents, the following analysis will indicate that both tools focus almost exclusively on the design layer, and consequently lose sight of requirements modelling (see Chapter 4). However, their approach is promising and, to judge by their sales figures, they are achieving great success.

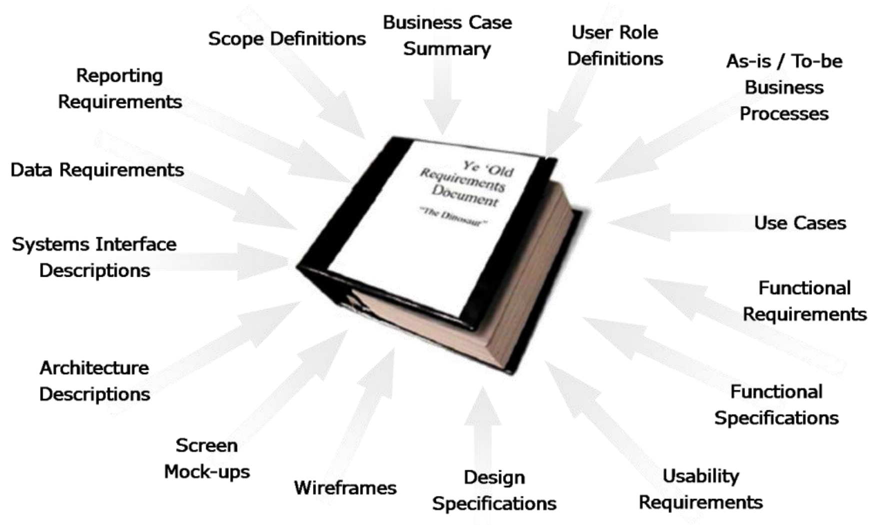


Figure 71: Didn't you read the specification document? (iRise 2008)

Easy to model, easy to share

The main aim of Axure and iRise is to provide tool support that (1) can be operated in a way comparable to standard office software in respect of visual appearance and usability and (2) offers better (ex-)changeability of artefacts.

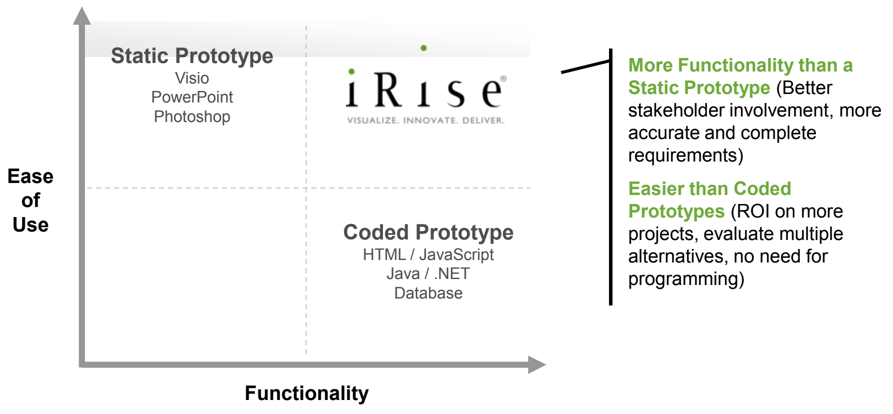

Static Prototype
Visio
PowerPoint
Photoshop
iRise
VISUALIZE. INNOVATE. DELIVER.
Coded Prototype
HTML / JavaScript
Java / .NET
Database
Ease of Use
Functionality
More Functionality than a Static Prototype (Better stakeholder involvement, more accurate and complete requirements)
Easier than Coded Prototypes (ROI on more projects, evaluate multiple alternatives, no need for programming)

Figure 72: The iRise approach (iRise 2008)

For example, high-fidelity prototyping with the iRise tool is intended to be as easy as with PowerPoint and follows two fundamental guidelines (see Figure 72), as presented in (Memmel et al. 2007c):

- **More functionality than a static prototype**: allow better stakeholder involvement, deliver more accurate and complete requirements, validate requirements through expressive simulation.
- **Easier than coded prototypes**: evaluate multiple alternatives, rapidly simulate business requirements without any programming.

Axure Pro in a nutshell

Axure's tool is intended to support design in a way that makes the process more tangible. Due to the opportunity to experience the design by running a prototype, stakeholders are invited to actively take part in the UI specification process. As an important by-product of prototyping, the design and the design decisions are documented within the specification-level prototypes. With Axure Pro, stakeholders can rapidly design UIs in a drag-and-drop environment with features such as snap-to-grid and zoom. Pages can be created easily and organized in a sitemap. Ready-made standard UI widgets can be edited and formatted, but individual widgets and design elements can also be created. With dynamic interactions that can be added to the design, the screens developed can then be assembled into an executable UI simulation. Flow shapes and connectors can be used to sketch scenarios and dialogue flow, from which the UI storyboard can be deduced (see Figure 73). Although Axure Pro comes without an additional server component, several actors can work simultaneously on a project and maintain a history of changes through a check-in/check-out system.

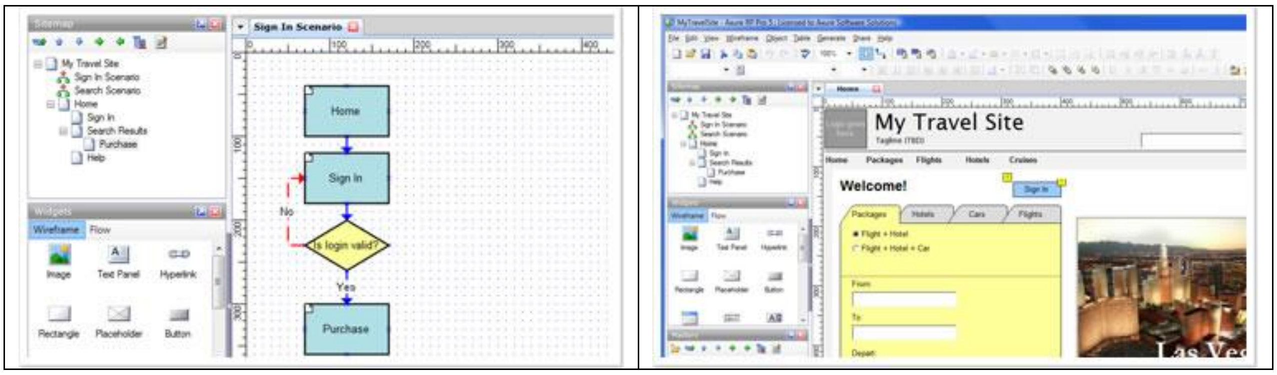


Figure 73: Modelling page flow (left) and the UI design (right) with Axure Pro

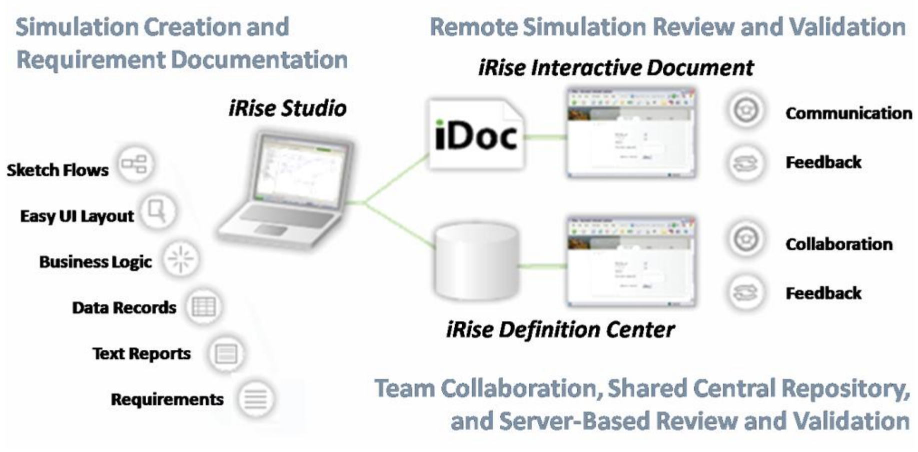

Simulation Creation and Requirement Documentation
iRise Studio
Sketch Flows
Easy UI Layout
Business Logic
Data Records
Text Reports
Requirements
Remote Simulation Review and Validation
iRise Interactive Document
iDoc
Communication
Feedback
Collaboration
Feedback
iRise Definition Center
Team Collaboration, Shared Central Repository, and Server-Based Review and Validation

Figure 74: Overview of the iRise product suite (iRise 2008)

Because text-based documents are usually still required even if the UI specification is expressed by running interactive simulations, Axure Pro provides the functionality to generate specifications in Microsoft Word format with screenshots, annotations and interactions that have been modelled with the tool, and the built-in templates can be adjusted to corporate standards.

iRise in a nutshell

Compared to Axure Pro, iRise Studio is much more complex and today comes with several optional components (see Figure 74). Because the software compiles the running simulation in a proprietary format, iRise-made specifications must be viewed in either iRise Studio or the iRise player. The latter does not allow changes to the original specification, but enables the viewer to annotate the screens developed and the associated requirements.

The UI requirements are usually modelled in a very simple way and rely on textual descriptions similar to scenarios. The iRise definition centre is the server component that supports real-time collaboration. The iRise server, which optionally extends the iRise Studio installation, adds the following capabilities (iRise 2008):

- Team members can collaborate within a secure, shared workspace – whether they are physically next to each other or a continent apart.
- Team members can work together in parallel on the same simulation instead of losing valuable time in a waterfall environment.
- Check-in/out capabilities enable working in a disconnected mode while still allowing others to access the latest simulation for review.
- Feedback from distributed reviewers is centralized in the repository and easily viewed via an in-context to-do list.
- The single source of truth is available at a central location, instead of in many documents spread throughout hard drives and network locations.
- Granular permission-based security prevents unauthorized access.

Due to the architecture of the 'iRise product suite', many important issues in UI specification can be successfully addressed. But because UI requirements can only be documented on a very high-level with iRise, the tool's main focus is still UI specification at the UI design layer (see Figure 75).

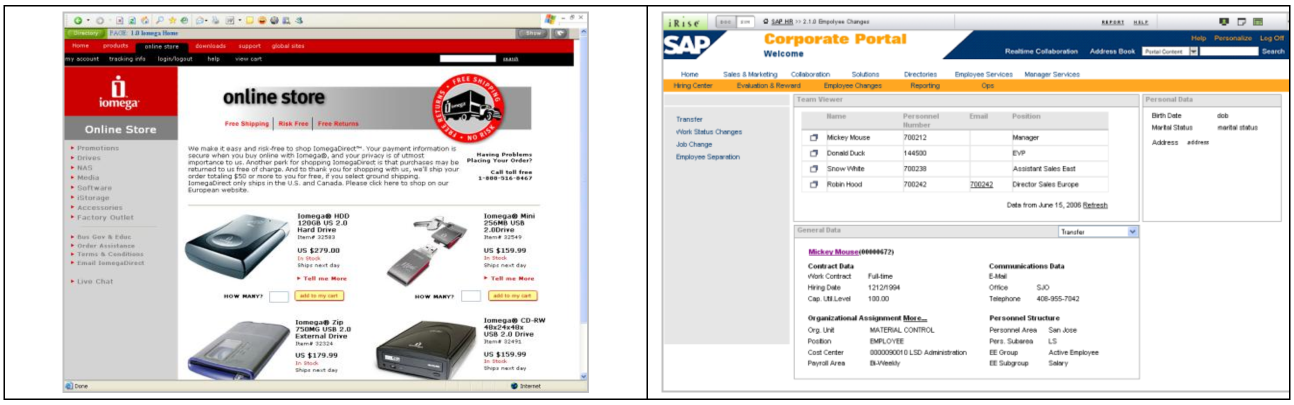


Figure 75: High-fidelity UI prototypes built with iRise (iRise 2008)

Accordingly, the main functionalities of iRise can be summarized as the following:

Main characteristics of iRise

- **Persona-based or campaign-based scenarios**: creating dynamic scenarios that reuse the same functional elements to demonstrate different interactive experiences for different personas or campaigns.
- **Dynamic data records**: incorporating real-life data records that can be created, saved, updated and deleted to realistically represent different user experiences.
- **User-experience-driven behaviour**: defining rule-based business-, data-, navigational-, and display-logic to dynamically represent user interaction and behaviour.

- **Reusable components**: creating a library of common functional design assets and only changing a design asset once to propagate the changes across all instances in the simulation.
- **Options and alternatives**: dynamically representing different options and alternatives to get feedback on what more properly meets business objectives and customer needs.

iRise in UI specification

Up to now, many large companies have bought and applied iRise in their UI specification processes. Most of the customers in iRise's reports on success stories are not IT organizations, which associates them with the automotive organizations that were analyzed with regards to UI specification practice in Chapters 2 and 3. The payback of requirements visualization in a prototyping-driven UI specification process is therefore evident (see Table 63). The use of simulation along the supply chain adds significant value to all stages of development (see Table 64). This, in turn, stimulates the development of an even more appropriate tool support that also integrates means for UI modelling (see Chapter 5.7).

Table 63: iRise success stories and the main payback from the tool (iRise 2008)

Payback	**iRise Customer**
20-30% reduction of project cost	AAA Insurance
15%–25% reduction in definition time	Cardinal Health
88% time reduction in task analysis & documentation	CitiGroup
80% reduction in requirements-related defects	Comerica
50% reduction in requirements cycle time; 25% reduction in overall SDLC	CompuCredit
30% increased speed to market	KeyBank
50% increased speed to market	M.D. Anderson
49% reduction in overall SDLC	Merrill Lynch
50% reduction in requirements cycle time	Sentara
20% reduction in SDLC	Sprint
58% reduction in customer support contact rate	Wachovia
25% acceleration of the development cycle	WaMu

As reported in (Memmel et al. 2007c), the iRise studio does indeed provide an easy-to-use UI. The whiteboard (see Figure 76) is related to storyboards (UE) and can be used to outline the navigational model of an application. Screens are represented by small icons and can be connected by dragging one icon onto another one.

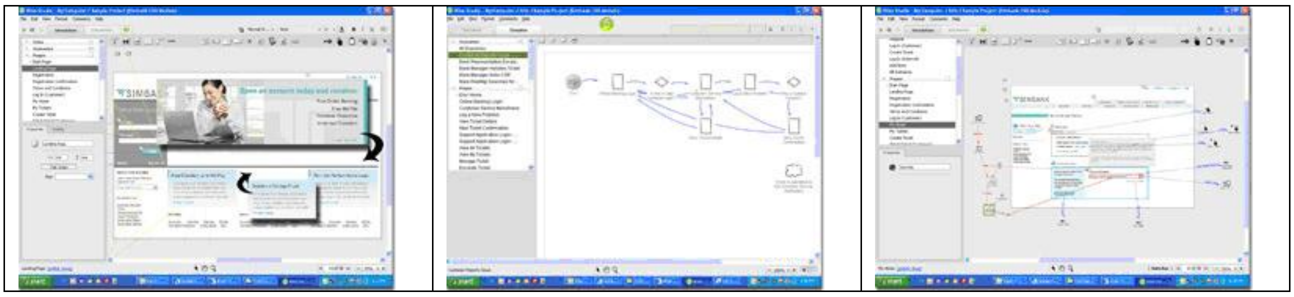


Figure 76: Creating and connecting UI dialogues with iRise (iRise 2008)

A closer look at iRise

For designing the UI of a screen, iRise allows zooming from whiteboard into detail. Texts, images, widgets and external objects (e.g. Adobe Flash objects) can be dropped onto the screen, and a highly interactive UI can be composed. Contents may be designed in outline at first and refined later. Following the idea of UI design patterns, masters and templates can be designed for reuse across different screens. Updating masters will automatically update the copies of the template screens used elsewhere in the application storyboard. Altogether, iRise provides three different levels of modelling, namely the high-level page flow, screen design and text view. All visual and textual information is exported to an iDoc, which is an executable simulation of the modelled UI. The iDoc player allows access to all three modelling layers without the main iRise application. The player is perfect for evaluation with stakeholders and provides functionality to annotate the simulation with sticky notes. As a supplement to the iDoc, iRise offers the possibility of saving descriptions and static images of the modelled UI in Microsoft Word format. But the absence of an integrated drawing tool requires the additional application of a picture-editing tool for sketching UI widgets, for example. When more sophisticated UI behaviour is necessary, embeddable objects such as Adobe Flash need to be generated separately. iRise will help to model ordinary UIs, but otherwise needs to become part of an interrelated tool-chain. In addition, the iDoc format is a compiled format and – in contrast to using XML – freedom of exchange with other RE or CASE tools is restricted. iRise therefore does not provide access to source code that could be reused for the implementation of the final system.

Table 64: iRise along the development lifecycle: the benefits of simulation (iRise 2008)

Project Evaluation & Estimating	**Requirements**	**Build**	**Testing**	**Training & Deployment**
High-level ideas are visualized and refined via the simulation	Creating documents and screenshots is replaced by an interactive simulation	Simulation provides clarity for design and development resources	QA starts to define test scripts from the simulation earlier in the lifecycle	End-users can be trained before the application is delivered
Application process flows are communicated and discussed	Complex scenarios are expanded visually and iterated upon	Simulation and functional specs are input to the solution-evaluation activities	Updated or new requirements are communicated; easier to update than text or static screens	Sales and marketing use the simulation during road shows
Better estimates are made based on the evaluation of the simulation	User acceptance begins before a single line of code is written	Time-consuming conversations over ambiguous requirements are avoided; more clarity	Execution of test scripts can begin immediately	
	Review and validation sessions	HTML templates jumpstart developers	User-acceptance testing earlier in the lifecycle	
	Unambiguous and correct specifications are generated			

5.2 SILK, DENIM and DAMASK

The origin of electronic storyboarding

The concept of UI simulation by electronic storyboarding and interactively linked dialogues that makes up the basis of Axure and iRise is not entirely new. (Landay & Myers 1995b; Landay & Myers 1995a; Landay 1996) introduced a design environment that focused on providing formal means of expression for early interface ideas. Because most designers in practice prefer to sketch their ideas on paper or on a whiteboard, they developed a tool called SILK (Sketching Interfaces Like Krazy) that allows designers to sketch interfaces using an electronic tablet and stylus. In contrast to real sketches, this electronic sketch is interactive, which enables simple interface behaviours to be specified. While early experimental prototypes were limited to a single screen for sketching, the technique was improved by utilizing the concept of UI storyboards. A number of subsequent UI designs can now be linked with connecting lines that are used to simulate UI flow. Designers are enabled to specify how the screens should change in response to user actions.

Sketching UIs with SILK

SILK combines the advantages of both sketching and traditional UI builders, yet it avoids many of the limitations of these approaches. SILK's UI is made to be as unobtrusive as pencil and paper. In addition to providing the ability to rapidly capture UI ideas, SILK enables the designer to move quickly through several iterations of a design by using gestures to edit and change the sketch. Changes and written annotations made to a design over the course of a project can also be captured and reviewed. Thus, unlike paper sketches, SILK-made sketches can evolve without forcing the designer to repeatedly start from scratch (Landay & Myers 1995b). For individual screens, SILK tries to recognize UI widgets and other UI elements as they are drawn. Although the recognition takes place as the sketch is made, it is non-intrusive and users will only be made aware of the recognition results if they choose to exercise the widgets. As soon as a widget has been recognized, it can be exercised. Employing SILK therefore preserves the benefits of pen and paper, namely that drawings can be produced very quickly and the medium of representation is still flexible. The informal and sketchy look of the screens stimulates creativity and innovation in design (see Chapter 4).

Specifying Behaviour

Easing the specification of the UI layout and structure solves much of the design problem, but a design is not complete until the behaviour has also been specified. Unfortunately, the behaviour of individual widgets is insufficient to test a working interface. For example, SILK not only needs to model how a button operates, but it must also know what interface action should occur when a user presses the button. Storyboarding with SILK allows the specification of the dynamic behaviour between widgets and the basic behaviour of new widgets or application-specific objects, such as a dialogue box appearing when a button is pressed (Landay & Myers 1995b). Sequencing is expressed by drawing arrows from buttons to screens that appear when the button is pressed. Figure 77 (right) shows three subsequent screens that simulate a rotation of a rectangular object. As the connecting lines imply, the press of a button changes the state of the UI and therefore simulates the behaviour of rotating an object by pressing a button. Each time the button is clicked, the rectangle in the drawing window rotates 60 degrees (Landay & Myers 1995b; Landay 1996).

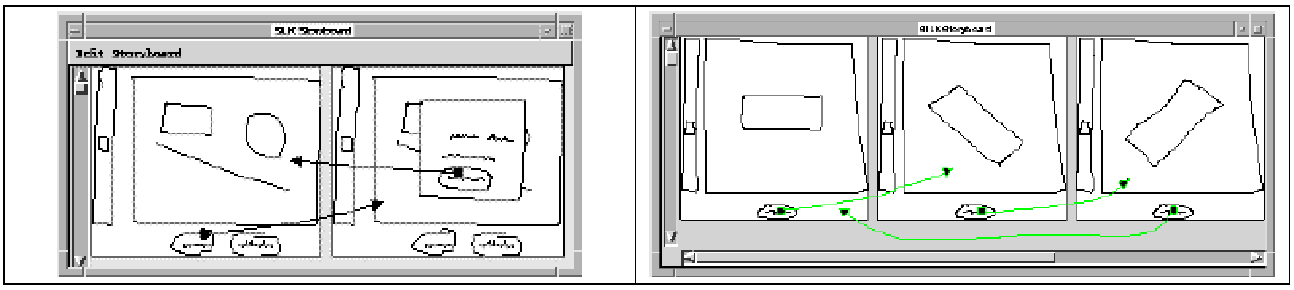


Figure 77: Sketching and storyboarding with SILK; from (Landay & Myers 1995b; Landay 1996)

Improvements over paper-based sketches

As discussed in Chapter 4, sketches allow designers to quickly record design ideas in a tangible form, and they do not require the designer to specify details that may not be important or yet known. Electronic sketches additionally remedy some of the weaknesses of paper sketches (Landay 1996):

- **Editing and Reuse**. One of the drawbacks of paper sketches is that they are hard to modify as the design evolves. The designer must often redraw features that have not changed. One way to avoid this repetition is to use an erasable whiteboard. SILK allows a designer to easily edit sketched UI designs by using simple gestures. SILK's history mechanisms will allow designers to reuse portions of old designs and quickly bring up different versions of the same UI for testing or comparison.

- **Design Memory**. Paper sketches lack support for 'design memory', i.e. the design rationale. The sketches may be annotated, but a designer cannot easily search these annotations at a later date to find out why a particular design decision was made. Practicing designers have found that the annotations of design sketches serve as a diary of the design process, which is often more valuable to the client than the sketches themselves (Boyarski & Buchanan 1994). Using SILK, changes made to a design over the course of a project can be reviewed, including viewing the attached written annotations.

- **Interactivity**. One of the biggest drawbacks to using paper sketches is the lack of interaction possible between the paper-based design and a user. In order to actually see what the interaction might be like, a designer needs to 'play computer' and manipulate several sketches in response to a user's verbalized actions. SILK allows the designer to quickly sketch rough design ideas and to test these designs by interacting with them. SILK blends the advantages of both sketching and traditional user-interface builders, yet avoids many of the limitations of these approaches. The system tries to recognize UI widgets and other elements as they are drawn. The designer can later specify the higher-level behaviour of the sketched UI elements, an example being the action to be performed when a user clicks on a button (see Figure 77). As soon as the designer is satisfied with the UI, SILK will replace the sketches with real widgets and graphical objects.

DENIM: zoom-based sketching

The concept of SILK was progressed in the succeeding tools called DENIM (Lin et al. 2000; Lin et al. 2001; Newman 2003) and DAMASK (Lin 2003b; Lin 2003a).

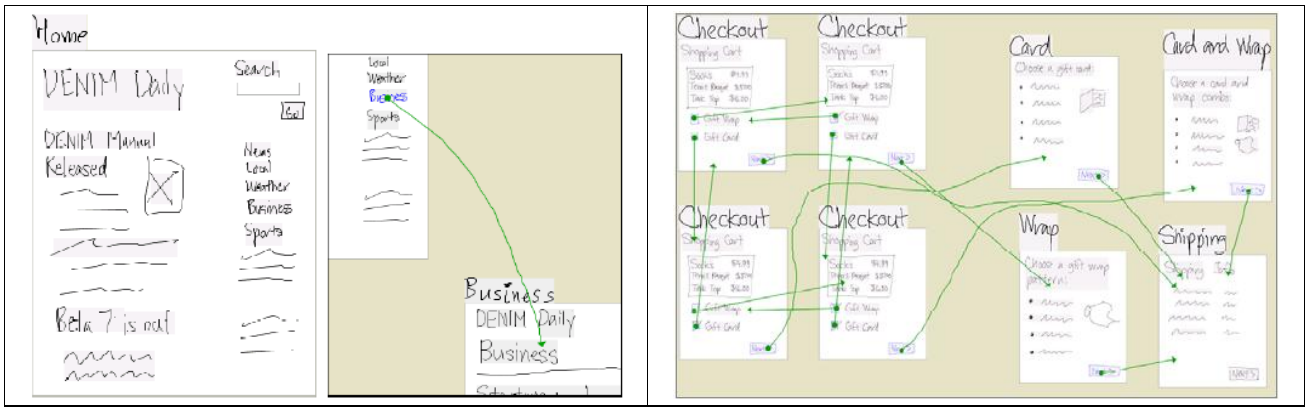
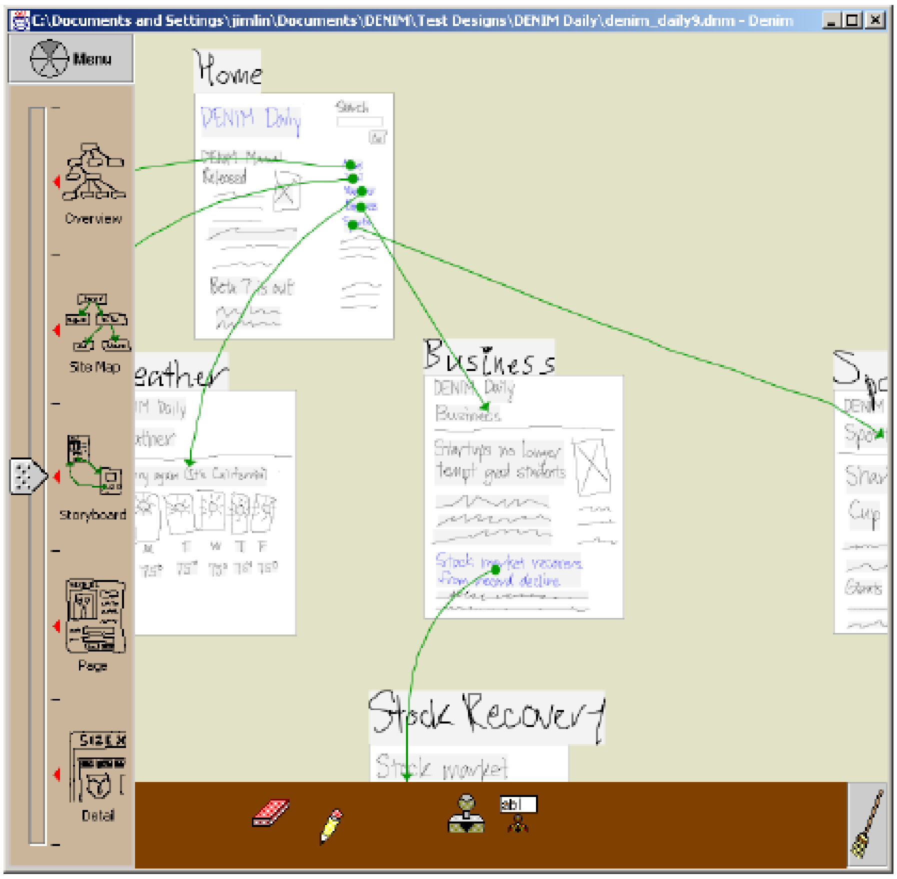


Figure 78: A UI dialogue created in DENIM with the label 'Home' and a link to another UI state called 'Business' (left); combinatorial explosion: transitions depending on two states lead to four pages and eight arrows (right); both (Lin et al. 2002)

The improvements in DENIM in relation to SILK are based on a zoom and patterns study, which concluded that designers usually sketch at different levels of detail: site maps, storyboards and individual pages. DENIM utilizes a zooming interface to visualize these forms of abstraction, but lacks recognition of sketches. DENIM has one window (see Figure 79) with three main areas. The centre area is a canvas where the user creates screens (e.g. web pages), sketches the contents of those pages, and draws arrows between the pages to constitute their relationship (see Figure 78).

On the left is a slider that is used to set the current zoom level. The bottom area is a toolbox that holds tools for drawing, panning, erasing, and creating and inserting reusable components. Instead of pull-down menus, DENIM uses techniques geared towards pen interaction. Pie menus and pen gestures can be used for quickly executing the most common commands, such as copying, pasting, and panning. Designers test out the interaction of their designs in DENIM'S run mode using a separate DENIM browser. The designer can navigate through the site design exactly as in a web browser, clicking on links and using the back and forward buttons.



C:\Documents and Settings\jimlin\Documents\DENIM\Test Designs\DENIM Daily\denim_daily9.dnm - Denim
Menu
Overview
Site Map
Storyboard
Page
Detail
Home
DENIM Daily
Business
eather
Stock Recovery

Figure 79: DENIM displaying a sketch of five web pages and six transition arrows in the storyboard view (Lin et al. 2002)

DAMASK: pattern-based UI design

DENIM and DAMASK allow designers to specify their own reusable components. These components can be as simple as a new kind of widget or as complex as a template for a web page. DAMASK extends this flexibility by adding support for sketching interface representations of several devices such as PCs, cell phones or voice. The storyboards for the different devices are presented in a combination of split screen and tabs (see Figure 80). Additionally, DAMASK introduced an interaction pattern browser that provides predefined behaviours for specific purposes (see Figure 81).

Developing for multiple platforms

DAMASK can create interactive UI sketches for multiple devices from the sketch of just one device and the design patterns used in that sketch. The UI designs thus generated will be of such a sufficient quality and usefulness that the designer will spend less effort in modifying the generated sketches than in creating them from scratch. At a high level, DAMASK includes a catalogue of design patterns that designers can use in their designs. Each design pattern contains specific examples of how the pattern has been used in other projects, and several generalized solutions capturing the essence of the examples. Designers are able to create their UI designs by sketching and by adding instances of patterns to their design for one device (Lin 2003b; Lin 2003a).

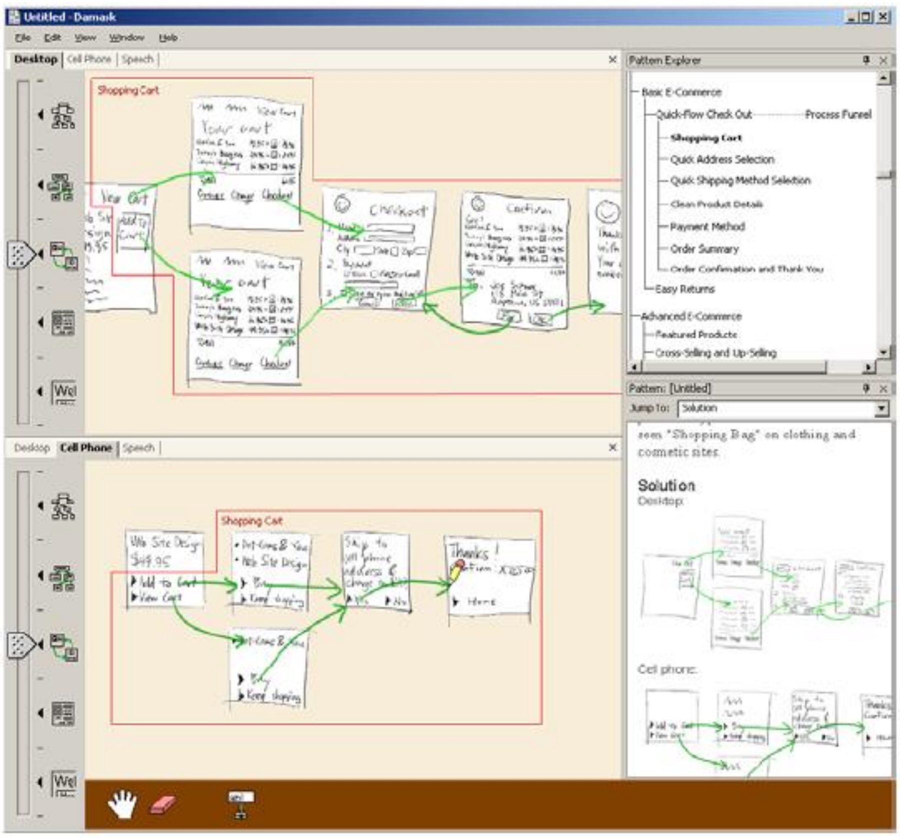


Figure 80: The UI of DAMASK with different areas for taking into account different platforms, such as desktop (top) and cell phone (bottom)

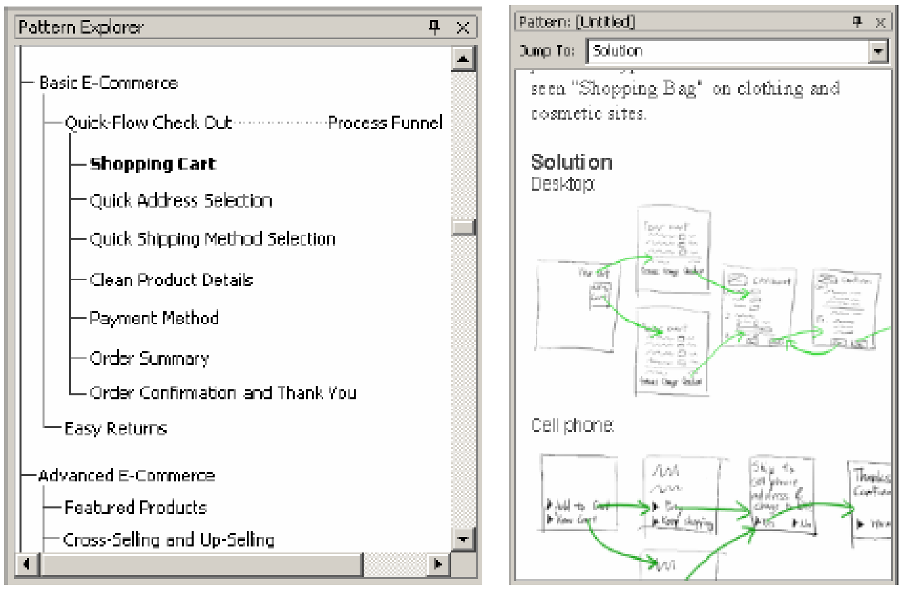


Figure 81: Right: DAMASK’s pattern explorer (Lin 2003b; Lin 2003a)

5.3 SketchiXML and GrafiXML

usiXML

The following tools are based on a novel XML-based UI description language that supports cross-platform and multiple-device development. The XML format is called 'usiXML' (Vanderdonckt et al. 2004; Vanderdonckt 2008) and it can be thought of as the scientific counterpart to Microsoft's XAML.

SketchiXML

SketchiXML (see Figure 82) supports the narrowing of the work transition from early sketches to high-fidelity designs (Coyette et al. 2006; Coyette et al. 2007). SketchiXML enables designers and end-users to sketch UIs with different levels of detail. The tool is intended to convey informal specifications of the UI presentation and dialogue. The scope of SketchiXML is to combine, in a flexible way, the advantages of tools such as DENIM with the advantages of tools such as JavaSketchIt (Caetano et al. 2002). While JavaSketchIt contributes algorithms for hand-sketch recognition, DENIM adds abilities for UI storyboarding and designing interactive behaviour. SketchiXML additionally integrates new features such as interface critiquing, computer-aided generation of UI specifications, and code generation for multiple computing platforms (the latter is no longer SketchiXML's unique selling point – DAMASK also has this ability).

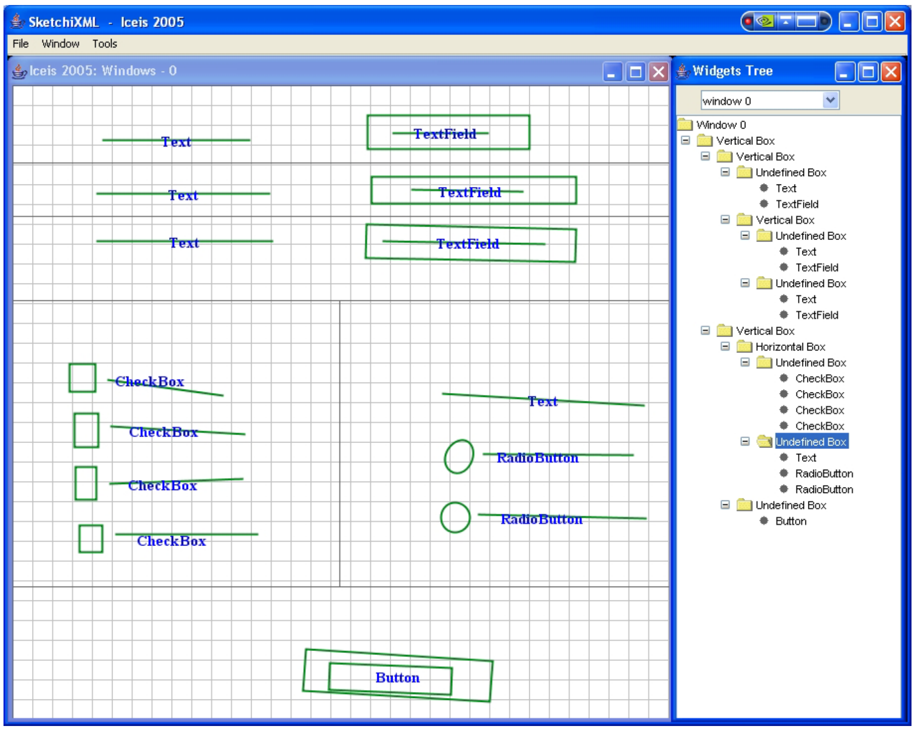


Figure 82: Sketching the UI with SketchiXML (Vanderdonckt 2008)

Sketch recognition

Based on a trainable sketch recognizer, SketchiXML automatically converts the developed low-fidelity sketches of UI states into XML documents that can be reused in graphical XML editors during subsequent development steps. The next step in the UI design process consists of importing the UI specifications generated with SketchiXML in a high-fidelity editor such as GrafiXML.

GrafiXML

GrafiXML (Lepreux et al. 2007; Michotte & Vanderdonckt 2008) is a graphical tool for drawing UIs for multiple computing platforms (see Figure 83). The UI thus created can also be exported to usiXML. GrafiXML is quite similar to other UI builders, except that it manipulates more widget properties than physical ones, and it saves any UI in UsiXML instead of in a particular code format. In this way, it is possible to maintain multiple localized versions of the same user interface and attach them to particular contexts of use (Vanderdonckt 2008). Moreover, with usiXML interpreters, it is feasible to generate JAVA or Adobe Flash UIs (Michotte & Vanderdonckt 2008).

Context modelling

Once the UI is designed, it can be associated with the particular context of use that is relevant to it. In GrafiXML, the context of use is composed of at most three aspects: a user (characterized by attributes such as task experience, disabilities, motivation, experience, and preferences), a platform, and an environment (e.g. level of noise, location, neighbourhood, stress level). The context editor used to specify these three models is invoked from the composer and automatically generates usiXML specifications corresponding to the target specified. All inputs are achieved by direct manipulation of the concepts involved in the models. A property sheet is then available for those aspects that cannot be specified graphically (Vanderdonckt 2008).

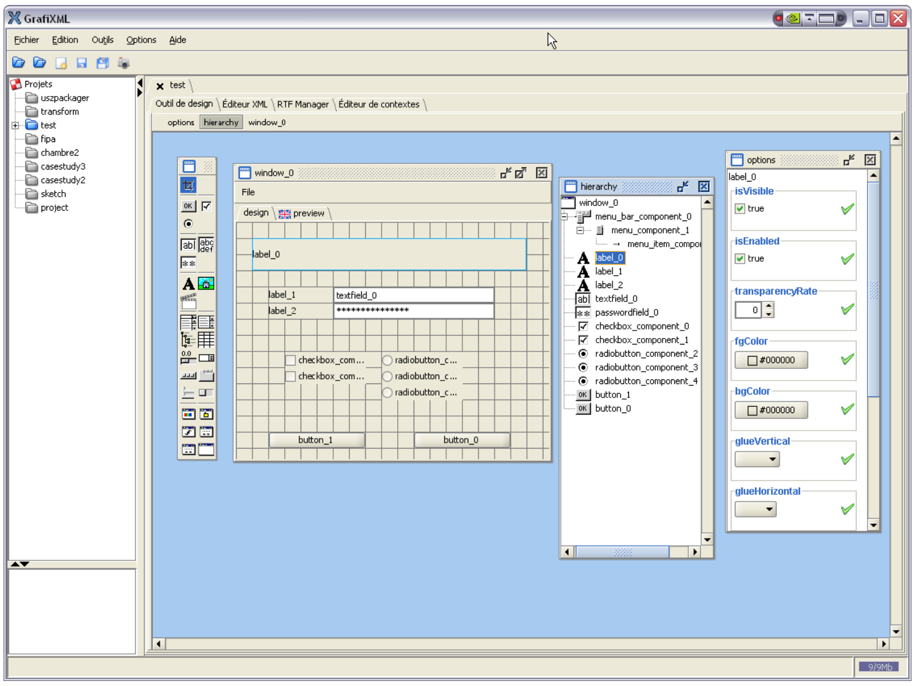
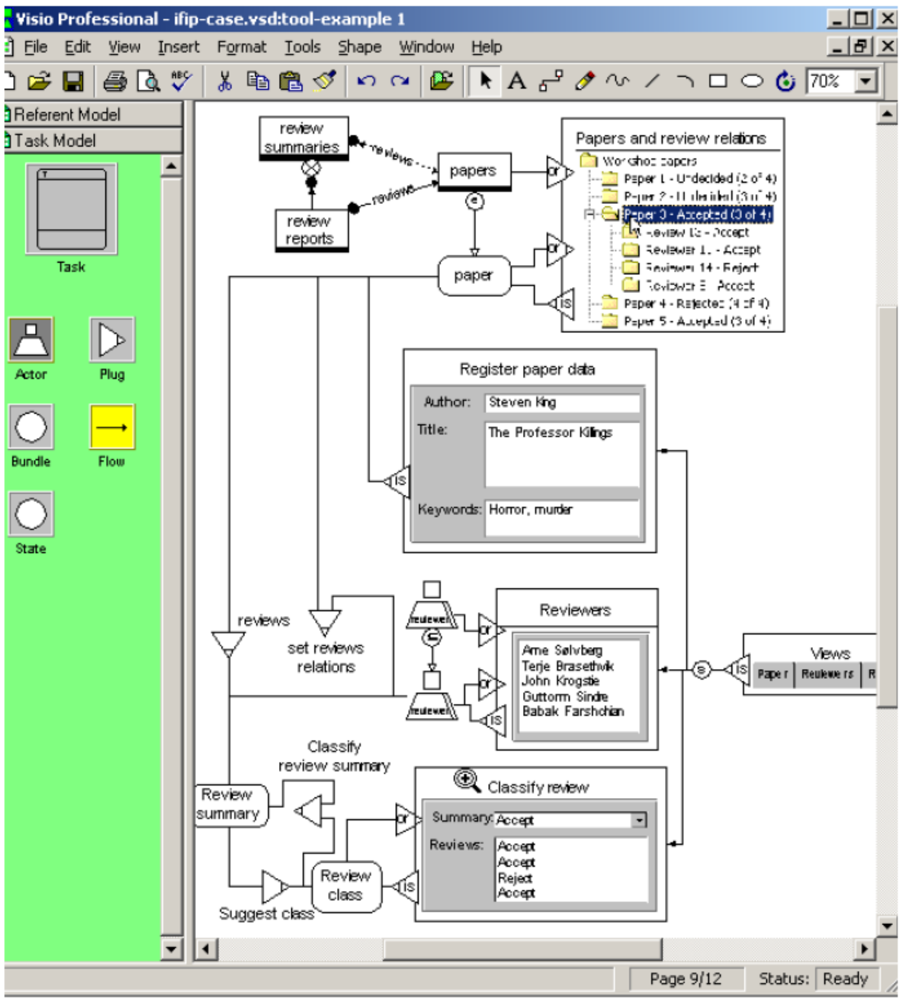


Figure 83: Modelling the UI with GrafiXML (Vanderdonckt 2008)

5.4 Diamodl



Author: Steven King
Title: The Professor Killings
Keywords: Horror, murder

Arne Sølvberg
Terje Brasethvik
John Krogstie
Guttorm Sindre
Babak Farshchian

Summary: Accept
Reviews: Accept Accept Reject Accept

Figure 84: A tool for DIAMODL; from (Trætteberg 2003)

Combine modelling and UI development

(Trætteberg 2004; Trætteberg 2008) introduced a model-based development environment, based on MS Visio, called 'DIAMODL' (see Figure 84). With an approach that combines modelling and GUI building, (Trætteberg 2004) intends to bridge the separation of concerns and disciplines. His approach aims to address the following issues of model-based UI specification:

- **Compatibility**. Make models work better with prototyping techniques. Support processes where models are derived/built bottom-up from concrete designs, not just top-down. Models must complement concrete representations and their relationship must be clear.
- **Complexity**. Simplify modelling languages and notations.
- **Trialability**. Design methods that may be used for limited parts of a design/project. Build tools that make methods easier to try and that may be integrated into existing ones.
- **Relative advantage**. Focus on the areas where most is gained, e.g. flexibility/tailorability. Make models complement existing models and design representations, and provide means for moving between them.

Different views of the UI

In accordance with these objectives, DIAMODL utilizes mock-ups of UI components and then enables them to be connected with flowcharts that reveal the underlying functionality. DIAMODL starts with a GUI builder and then adds additional value with specific models and constructs in order to shape the underlying logic of the UI. The user can choose between different views, i.e. UI only, model only or a hybrid view (see Figure 84). (Trætteberg 2004) focuses on the logical structure of interactions that are displayed next to UI mock-ups. Relationships and flows between these distinct states of the UI are also visualized with connecting lines. By using such a hybrid view, the notation can be used as an abstract UI specification. In order to exchange the constructed UIs with developers, (Trætteberg 2004) uses XML as the external format.

DIAMODL going agile

Due to a survey in industry, the developers of DIAMODL are working on integrating DIAMODL in a semi-formal approach, and have chosen Constantine's abstract prototypes (Constantine 2003) as a starting point. The finding that corporate stakeholders are trying to work less formally is similar to the results of interviews made in the context of this thesis (see Chapter 3). Accordingly, (Trætteberg 2004) envisions a step by step method that moves from abstract and informal means of expression to more detail and formality during later stages, or when required. Gradually adding modelling constructs when the precision provided by models is needed is therefore a sound extension of UI prototypes.

5.5 CanonSketch, TaskSketch, WinSketch and MetaSketch

Canonical prototypes and modelling

CanonSketch and TaskSketch (Campos & Nunes 2004a; Campos & Nunes 2004b; Campos 2005a; Campos 2005b; Campos & Nunes 2005b; Constantine & Campos 2005; Campos & Nunes 2007) are based on the idea of a hybrid view of UI design using canonical abstract UI prototypes and a variety of UI models. In order to match the work styles of practitioners, both tools are based on a novel UML-based notation called 'Wisdom' (Whitewater Interactive System Development with Object Models). Due to the widespread incompatibility of UML with corporate UI specification, the Wisdom method is especially focused on UI-related issues. It concentrates on those models relevant to describing the UI (Nunes & Cunha 2000; Nunes & Cunha 2001)

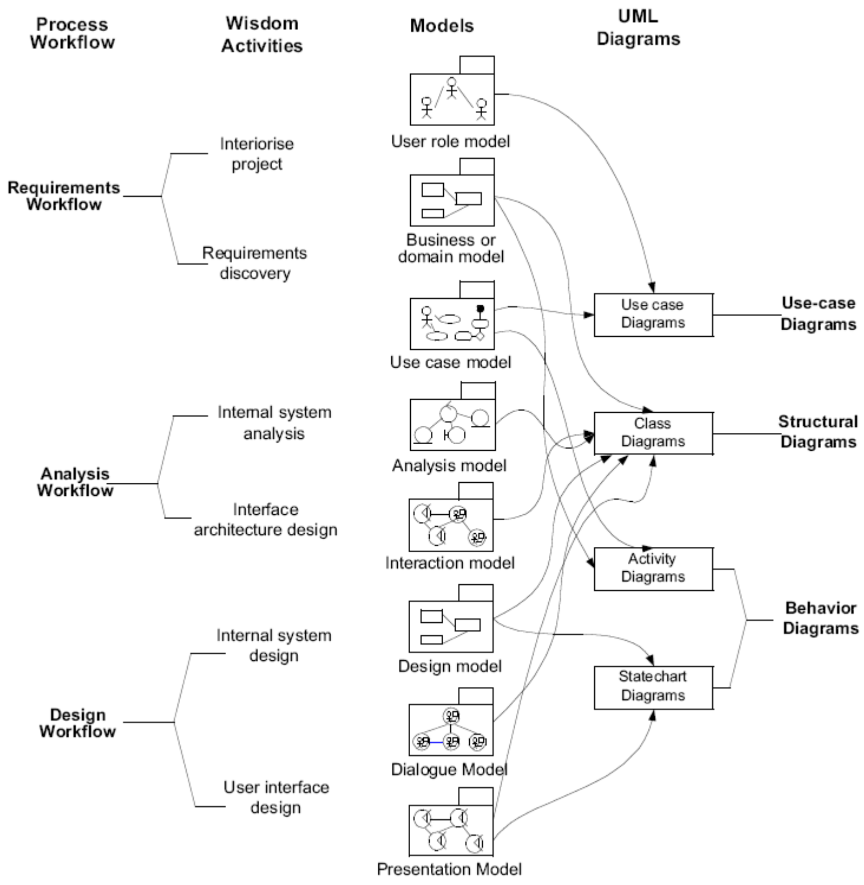


Figure 85: Workflows, activities, models and diagrams in Wisdom (Nunes 2001; Nunes & Cunha 2001)

Modelling with Wisdom

Wisdom is object-oriented and uses the UML to specify, visualize and document the artefacts. It is specifically adapted for developing interactive systems because it uses, and for this purpose it extends, the UML to support IxD techniques. However, Wisdom is a lightweight SE method, because it concentrates on a small set of models. Wisdom is related to iterative UI prototyping efforts, which ultimately lead to the final UI (Nunes & Cunha 2001). Wisdom neither includes nor recommends any specific UI design methods, however.

Figure 85 summarizes the four main workflows and shows the corresponding Wisdom activities, models, and diagrams used to specify the models. In all, Wisdom is based on seven different models and uses four types of diagrams. In (Nunes 2001), an eighth notation was added, namely role models, because the absence of user models may have led to considerable inconvenience when using this approach. Even with this extension, however, the Wisdom approach only models roles indirectly through use-case diagrams and does not provide an explicit model for user modelling. In summary, the diagrams employed are all known from the discussion of modelling techniques in search of the common denominator (see Chapter 4). As it is, Wisdom only uses a subset of UML (approx. 29% of the total concepts in UML 1.1).

The models of Wisdom

In the Wisdom approach, use-cases play an important part in capturing functional requirements and driving the whole development process. The use-cases also serve as the major input for finding and specifying task flows. Later, activity diagrams represent in more detail the desirable and realistic task flows performed by the users. Activity diagrams are then used throughout the entire development process to prototype and design the UI. Wisdom can also support modelling with the essential use-case notation, which allows the structure of use and the structure of the UI to be connected. Despite its focus on the UI, Wisdom also takes technical issues into account and allows notes on non-functional requirements to be attached to use-cases (Nunes & Cunha 2001). All told, Wisdom models allow the linking of abstract requirements to the UI design through a systematic process that travels from models to the surface. CanonSketch and TaskSketch also have a strong focus on supporting transitions between design artefacts.

CanonSketch

CanonSketch was the first tool presented by (Campos & Nunes 2004a; Campos & Nunes 2004b). It introduces three distinct levels for modelling and UI design, namely (1) UML class diagrams, (2) abstract canonical prototypes and (3) concrete HTML for presenting the UI design. The class diagrams can be arranged in an easy-to-use modelling environment. The view of the UI can be easily switched from the class diagram to the other two presentations by simply clicking on a tab-like navigation aid at the top of CanonSketch's canvas (see Figure 86).

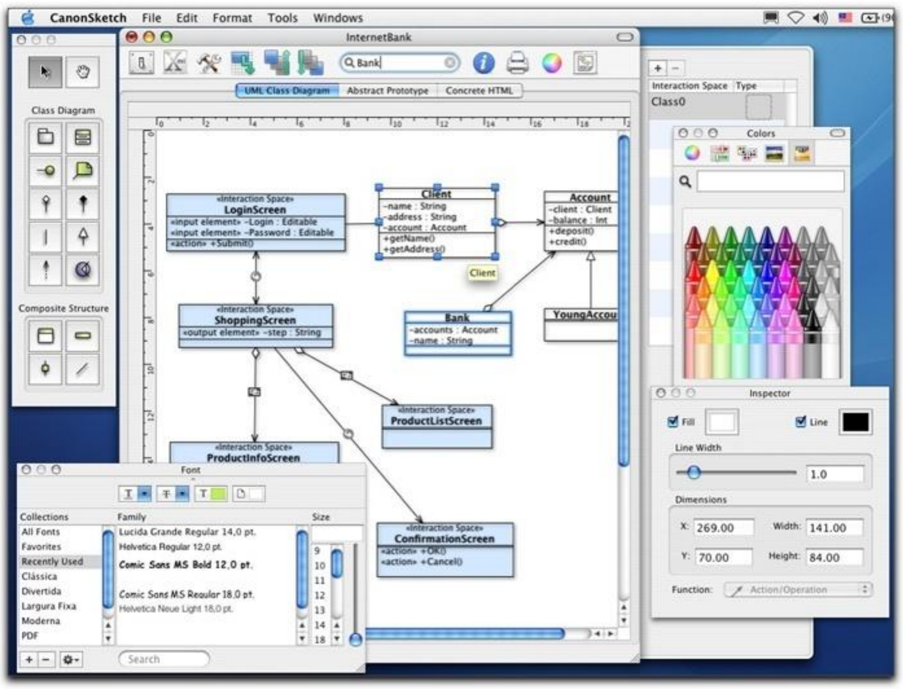

CanonSketch File Edit Format Tools Windows
InternetBank
UML Class Diagram
Abstract Prototype
Concrete HTML
Class Diagram
Composite Structure
«Interaction Space» LoginScreen
«Interaction Space» ShoppingScreen
«Interaction Space» ProductListScreen
«Interaction Space» ProductInfoScreen
«Interaction Space» ConfirmationScreen
Client
Account
Bank
YoungAccou
Colors
Inspector
Font

Figure 86: Class modelling with CanonSketch (Campos & Nunes 2004b)

Later on, abstract models can be transformed into canonical abstract representations of the UI, which in turn can be exported to HTML (see Figure 87). The model-driven approach (see Chapter 3.2.2) of CanonSketch therefore allows the same model to be displayed at different levels of abstraction and supports the generation of code from the developed models. But as a conceptual model for the UI has to exist before tasks can actually be designed, CanonSketch is inadequate because it lacks support for early process phases. This sets the stage for the tool's brother in arms, called 'TaskSketch'.

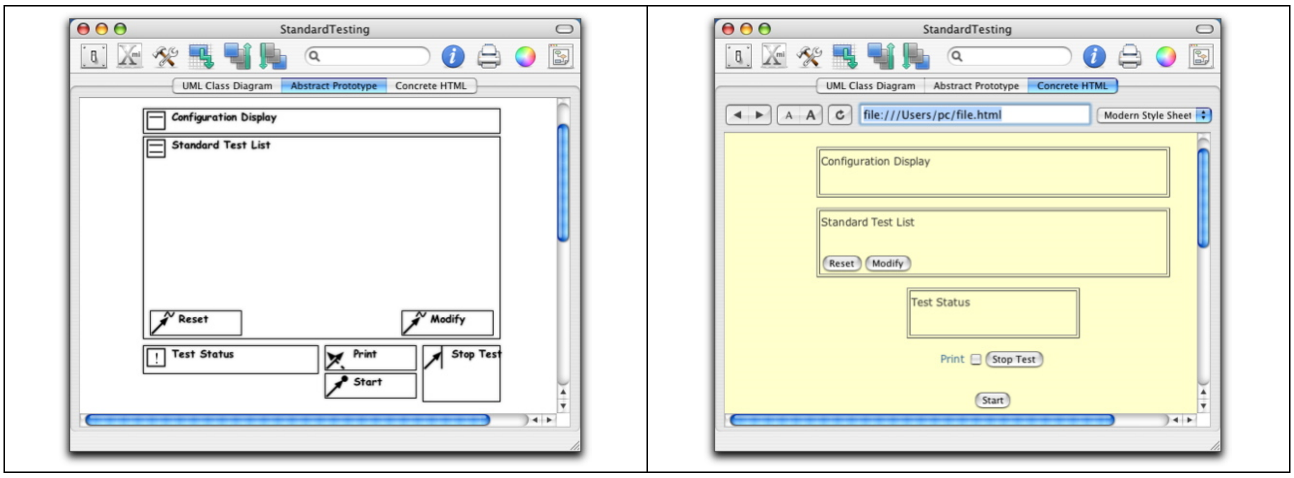


Figure 87: Canonical abstract prototype and HTML version developed with CanonSketch

Task Sketch

In contrast to CanonSketch, TaskSketch (Campos & Nunes 2004b) is focused on linking and tracing use-cases for requirement analysis, before actually designing the UI with CanonSketch. Again, the previously discussed Wisdom approach is utilized to adapt use-cases and activity diagrams to UI design. With TaskSketch, the user can develop activity diagrams, use-case narratives and a participatory view of the left side of the split pane (see Figure 88).

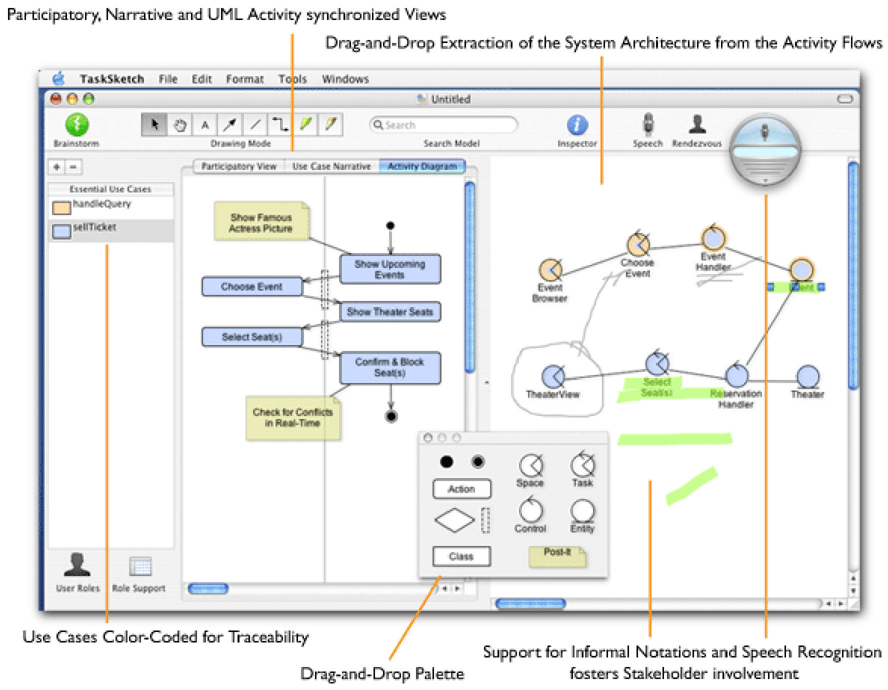


Figure 88: The main user interface of TaskSketch (Campos & Nunes 2005b)

Use-cases are then highlighted to visualize relationships with the corresponding models. TaskSketch supports speech input and group dynamics. During brainstorming, stakeholders can share ideas for concepts and tasks in a text field on the bottom of the tool's UI. Concepts and ideas are then clustered by dragging them close to other items. TaskSketch is therefore helpful in meetings or during discussions. Its playful character is intended to foster collaboration among stakeholders.

With WinSketch, the concepts of CanonSketch and TaskSketch were further enhanced. WinSketch combines the functionality of its two predecessors. Users are guided through the usage-centred design process by following a horizontal hierarchical navigation that is structured into the basic entities ‘requirements’, ‘analysis’ and ‘design’. Each step is accompanied by one or more diagrams that represent model-based views of different degrees of abstraction within the process. Bookmarks of models are then used to trace mutual relationships between requirements or models and corresponding interface components. Due to its close relationship to CanonSketch and TaskSketch, WinSketch provides effective support in tracing requirements. However, (Geyer 2008) found that relationships between artefacts are sometimes barely visible. As diagrams are nearly always displayed in small frames, it is hard to gain an overview and to switch between different representations. The design rationale of the UI specification can therefore be hard to understand.

WinSketch

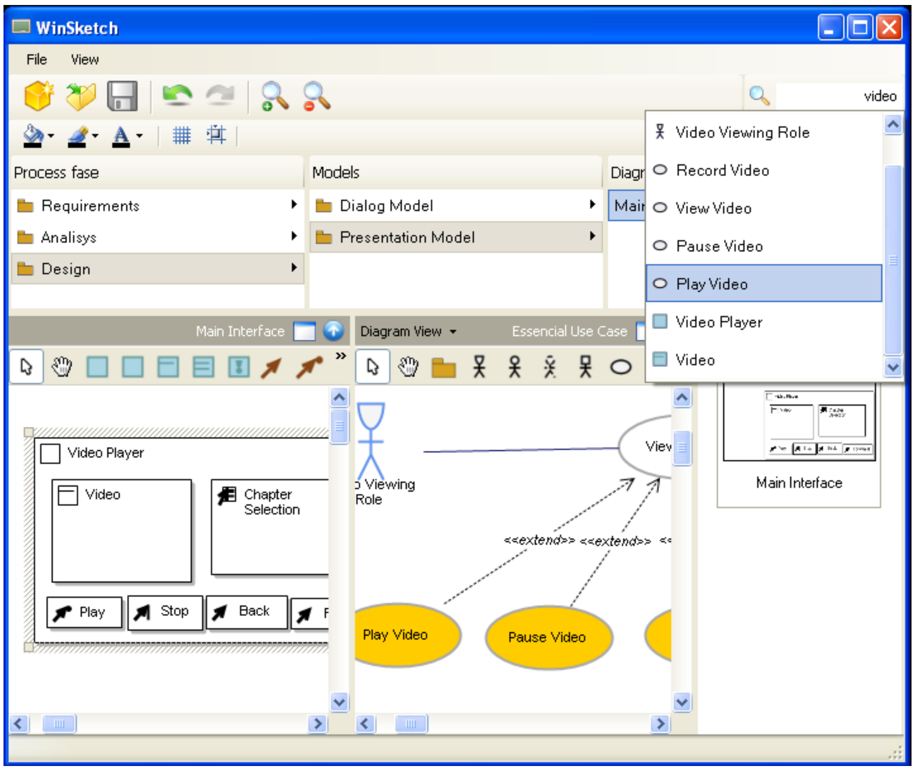


Figure 89: The UI of WinSketch (Alfonseca et al. 2006)

WinSketch (see Figure 89) was not presented to the scientific community, but was probably the basis for MetaSketch (Nóbrega et al. 2007; Nóbrega 2008). The motive behind MetaSketch is also to bridge the gaps between the disciplines, especially SE and IxD. The tool is designed to overcome the differences between the disciplines by moving interdisciplinary modelling to the meta-modelling level. Accordingly, MetaSketch is a workbench that allows the definition of new modelling languages based on OMG (The Object Management Group) standards such as MOF 2.0, OCL 2.0 and XMI 2.1, and it is specially tailored for creating new members of the UML family of language. It also means that MetaSketch can be used to extend and adapt version 2.0 of the UML, creating new possibilities and opportunities in the context of model-driven UI development approaches. In (Nóbrega et al. 2007), MetaSketch is used to extend UML 2.0 with canonical abstract prototypes (see Chapter 4) and CTT (see Chapter 4). Due to its meta-modelling approach, MetaSketch is more flexible and potentially allows the integration of many more models in order to provide shared means of UI modelling. But ultimately, the approach is still very close to SE practice and requires a good understanding of UML, because the models created must be formal and precise enough to be able to generate code.

MetaSketch

5.6 Wisdom vs. The Common Denominator

A closer look at approaches assumed to be related

Both the Wisdom approach and the common denominator presented in this thesis are meant to ease the process of UI development or specification by concentrating on a thoughtfully selected set of modelling languages and design artefacts. The close relationship between them requires a closer examination of any important differences in order to be able to clearly separate both the methods (see Table 65) and the associated experimental tools.

User modelling

Reconsidering the ingredients of Wisdom, it becomes very obvious that the approach does not take user models and UI design into account properly. In contrast to the common denominator, the Wisdom approach does not recommend artefacts such as personas to model the user of the interactive system in detail. But, Wisdom recognizes role models, and the integration of role models into use-case diagrams follows the basic ideas of usage-centred design (Constantine & Lockwood 1999b). The Wisdom approach is therefore very close to the common denominator, while the latter is more systematic in taking important user models into account.

"The Wisdom user role model is presented using UML use case diagrams with stereotyped relationships between actors." (Nunes 2001), p. 135

(Problem-)Domain modelling

Taking one step back, the two approaches also differ in terms of modelling the problem-domain, i.e. with respect to the (business) context and environment of use. With regards to the 5-S approach presented in Chapter 3, the common denominator is designed to support the early stages of business visioning and the development of goals in a rather informal way. Therefore, diagrammatic notations were not taken into account at this modelling layer. The Wisdom approach, in contrast, uses class diagrams in the way usual for most SE processes (see Chapter 4). At this stage, the motives for bridging the gaps between the disciplines are not easily identifiable when looking at the models proposed by Wisdom. However, activity diagrams and (business) use-case diagrams are also part of the common denominator, although at later stages in the UI specification process. To sum up, the Wisdom approach clearly identifies its close relationship to UML and model-driven development. The systematic formality is therefore an obvious difference to the common denominator proposed in this thesis.

Task and behaviour modelling

In task modelling, both approaches are very similar. The common denominator explicitly supports modelling task maps, but because their notation is very similar to that of use-case diagrams, Wisdom may be able to support them as well. A similar situation exists with regards to the notations proposed by both methods for behaviour modelling. Although state charts and data-flow diagrams differ, they both belong to a comparable class of notations (see Chapter 4) and it can be seen that both approaches consider more than one notation for modelling UI flow. In addition, Wisdom is also compatible with CTT, which almost makes CTT an additional model included in this approach (Nunes 2001). As a bridging notation, CTT is well recognized among researchers and possibly also among open-minded software developers with an academic background. In corporate UI specification, however, CTT has been little acknowledged. Consequently, the common denominator differs from the Wisdom approach in the sense that well-known models, in a pragmatic and approximate agile form, are preferred to very specific and non-standard notations. It is therefore the practical applicability that distinguishes the two methods.

UI prototyping

Considering the later stages of UI specification, the common denominator explicitly entails various UI design methods for fostering creativity, narrowing design spaces thoughtfully, and specifying the UI in a high level of detail. Wisdom, in contrast, does not incorporate different stages of UI design into its set of UI-related artefacts. Again, this underlines the clear dominance of UML-related modelling in the Wisdom approach. The common denominator clearly distances itself from this focus and is intended to support prototyping-driven, creative UI specification processes.

Table 65: Comparison of the Wisdom approach to UI development and the common denominator

	Wisdom (see Chapter 5.5)	**Common denominator (see Chapter 4)**
Philosophy	Model-driven UI development	Model-based UI development
Main goal	Leverage automatic and flexible generation of UIs from high-level models; promote artefact exchange between UML and task-modelling tools	Support prototyping-driven UI specification by linking models and UI design; enable the forwarding of detailed interactive UI specifications
Basic concept	Extend models of UML to incorporate IxD into software development	Combine existing means of modelling into a common denominator
Domain Models	Class diagrams (domain model) and activity diagrams (business process); optional business use-case diagrams (business structure)	Problem scenarios
User Models	Users represented as roles in use-case diagrams (i.e. role models)	Personas, role maps
Task Models	Use-case diagrams, essential use-cases, CTT	Use-case diagrams, essential use-cases, task maps
Behaviour Models	Activity diagrams, state-chart diagrams, CTT	Activity diagrams, data-flow diagrams
UI Design	Models must support UI design; no integration	Sketching, canonical/abstract forms, detailed UI

Target audience

On the whole, the Wisdom approach can be categorized as an approach allocated in the world of SE, but with significant add-ons to UML for incorporating IxD. Nevertheless, the notations employed consistently focus on the SE-side of potential artefacts. The formality of the approach, the motivation for automatic code generation from its models, and the absence of UI prototyping from its repository make it rather inappropriate for the creativity-driven development of innovative designs. Furthermore, these aspects also hamper the population of stakeholders typically working on UI specification in the non-IT industry. Accordingly, Wisdom is more suitable for software-developing organizations, while the common denominator is designed to enable 'normal' people to take part in UI specification. In summary, the two approaches can be said to be related, but must be carefully distinguished with respect to their fundamental intention.

Comparing the associated tool support

Finally, in anticipation of the presentation of the INSPECTOR tool (see Chapter 6), this comparison can now be widened to include the experimental tools that were developed to prove the applicability of the modelling methods already discussed. With regards to Table 65, the tools differ in terms of the methodologies they are based on. From an analysis of literature and reports on the Wisdom tools, two very obvious differences can be derived from the presentation in Chapter 5.5.

Patency of UI specification support

Firstly, INSPECTOR is designed to cover all aspects of UI specification, including UI prototyping in differing fidelities in corporate organizations, and has a corresponding, clearly defined target audience that, in all probability, is quite different from the kind of users that are likely to apply the Wisdom tools (e.g. due to the use of strict UML models). Although WinSketch integrates all of the steps supported in its predecessors, it lacks the different levels of UI prototyping that are necessary to propel the specification process.

Appearance

Secondly, and as a consequence of the previous point, INSPECTOR uses a completely different approach to computer-aided UI specification. It tries to map the work styles of corporate stakeholders to its largely unconstrained zoom canvas. The Wisdom tools were also developed with reference to a work-style study, and they successfully satisfy demands such as more traceability and transparency. However, the Wisdom workbenches employ very convenient ways of linking and tracing artefacts, which can easily cause a 'lost in specification space' syndrome (see Chapter 7). Modelling and designing with zooming and panning allows a closer relationship of frequent activities (e.g. switching detail) to their interactive mapping and presentation on the canvas, as successfully demonstrated by DENIM, for example.

5.7 The Gap In Modelling and Tool Support

König (2008) gave insight into various UI-related modelling and design tools. Most of them primarily focus on providing graphical programming functionality to enable designers to build functional prototypes. Nevertheless, they do not offer adequate conceptual design support together with process support. Consequently, these applications tend to be used when the real design work is already done. In contrast, real UI development tools often concentrate on the other extreme, namely the construction of abstract models. Although modelling is extremely important for corporate UI specification, tool support must incorporate both the model and the design world. The approaches and tools analyzed in this chapter are the most promising vehicles in thoughtful and design-oriented UI specification processes. As outlined previously, none of the discussed tools is the all-in-one device suitable for every purpose along the supply chain. Each one has its own focus, which, however, serves as a pathfinder in defining more appropriate tool support for UI specification. In the following, the lessons learned from taking into account related work are summarized in order to derive some key points (see Chapter 5.8) that strongly influence the proposed experimental tool presented in Chapter 6.

Learning from sketch-based UI specification

One reason for the lack of acceptance of formal UI and modelling tools is the fact that many solutions do not match or augment the work practices of designers and other stakeholders who do not have a sophisticated IT background or who simply refuse to apply complex tools. Many approaches force designers to think at a high level of abstraction too early in the design process. They do this by forcing them to design in terms of abstract widgets or by specifying a task model that is then transformed into a concrete UI. In reality, designers are accustomed to thinking about concrete UIs from the very beginning of the design process. This habitual course of action is well supported by SILK, DENIM and DAMASK. The approaches used by these tools allow UI designers to specify designs at a more abstract level, but with a vocabulary that designers understand, via sketches (SILK, DENIM) and design patterns (DAMASK). The design solutions thus developed certainly do not represent the final UI (e.g. with regards to CAMELEON), and neither can the generated UIs be used without modification (Lin 2003b; Lin 2003a). However, all three tools are very much focused on early-stage UI prototyping, which in turn makes them less attractive for detailed UI specification with interdisciplinary modelling. Because requirements and conceptual modelling, as well as high-fidelity design, are not taken into account, additional transitions in tool usage are necessary. The overall process is still complex, and the level of complexity may even increase as yet another tool is added to the production chain (Vanderdonckt et al. 2004).

Lessons learned from CanonSketch, TaskSketch, WinSketch and MetaSketch

CanonSketch was the first tool that used canonical abstract prototypes and the UML-like Wisdom notation, supplemented by a functioning HTML UI design layer. TaskSketch is a modelling tool that focuses on linking and tracing use-cases, by means of which it significantly facilitates development tasks with an essential use-case notation. Altogether, TaskSketch provides three synchronized views: the participatory view uses a post-it notation to support communication with end-user and clients, the task-case view is targeted towards designers and is a digital version of index cards (well-known artefacts of user-centred or agile developers) and the UML activity diagram view is suitable for software engineers. WinSketch combines the best of both approaches. MetaSketch moves the focus of its predecessors to meta-modelling and thus makes the approach much more flexible. MetaSketch is decoupled from the Wisdom approach and makes it possible to define independent sets of models in order, for example, to extend UML by means of IxD. A related idea concerning model-driven UI development with a meta-modelling approach was presented in (Memmel et al. 2007a). With a separation of development concerns, different levels of abstraction and a simulation framework, an advanced UI modelling method was established. Although it was necessary to pre-define a domain-specific language (high-threshold), the results added significant value to a previously long-winded UI specification process (high-ceiling). But because the proposed tool chain was targeted towards the later stages of the process, Office applications (see Chapter 2) remained dominant during earlier phases. MetaSketch differs from this approach by integrating all the necessary tools into one workbench, rather than separating the concerns and relying on a tool chain. MetaSketch is also designed to take early-stage models, for example problem-domain models or user models, into account. However, both solutions share one important shortcoming with regards to the specification of innovative UIs. The use of a formal approach, targeted towards the generation of code from models, is severely limiting with respect to freedom in creativity and promotion of innovative ideas (Memmel et al. 2008f). This also applies to MetaSketch.

For defining tool support for corporate UI specification processes, harnessing concepts from all four tools is a promising starting point. Nevertheless, any successor approach must learn from the shortcomings and needs to be different in some important areas (see Chapter 5.6). Firstly, and in contrast to CanonSketch, more detailed UI prototyping must be supported, because the high-fidelity specification-level externalization of design vision is especially important in corporate UI design processes. Conversely, more freedom in the early stage of design must also be incorporated. CanonSketch, together with it successors, lacks means for sketching, although sketch-based modelling and design is essential for designing innovative interactive systems (see Chapter 4). Secondly, many more ways of modelling (earlier text-based artefacts, task models and interaction diagrams) that take into account the proposed common denominator (see Chapter 4) must be integrated (Memmel et al. 2008f). Although the Wisdom approach is related to the method proposed in this thesis, it does not take detailed user descriptions such as personas into account (see Chapter 5.6). Instead, the Wisdom approach, and the tools based on it, tend to start with task modelling, which may be traceable back to its relationship to usage-centred design (Constantine & Campos 2005). Thirdly, moving the problem of UI specification to the meta-modelling layer significantly shrinks the design space and decreases the opportunity for innovation (Memmel et al. 2008f). In addition, although MetaSketch is a promising integrative toolkit for UI development, meta-modelling is quite inappropriate for the kind of stakeholders who contribute to corporate UI specification. On balance, by integrating and linking different means of expression, CanonSketch, TaskSketch, WinSketch and MetaSketch provide excellent means to increase the traceability of requirements. But unfortunately the tools either lack additional means of expression that would allow the designer to build up an appropriate network of UI-related artefacts that would be suitable along the whole UI specification, or they shift the focus to a modelling layer unusable by stakeholders untrained in formal modelling.

Contributions of usiXML-based SketchiXML and GrafiXML

The tools based on the CAMELON reference framework follow a similar approach in providing complete UI specification support, but none of the tools is an all-in-one device for the whole process. Accordingly, several tools must be combined into a tool chain, which then benefits from a continuous and consistent interchange of created designs and models because of usiXML. In accordance with the CAMELON reference framework, SketchiXML supports transitions from abstract UI to final UI, but is itself mainly concerned with the abstract stage. GrafiXML should be used during later stages to elaborate the concrete and detailed design using typical UI widgets or self-made elements. But once again, tasks and concepts are neglected and have to be bridged with other tools (Memmel et al. 2008f). Nevertheless, employing an XML-based specification language for intermediate and final designs is an appealing option as it facilitates interoperability and accessibility throughout the process.

Lessons learned for adequate tool support

As things stand, an integrative tool approach appropriate for creative, prototyping-driven corporate UI specification processes is still missing, and the lack of adequate support is the point of origin for providing experimental ideas to fill in the gap in research and commercial distribution. Tools such as DIAMODL already veer in the direction of bridging the gaps between disciplines and integrating the design and modelling worlds. Many approaches, however, are accompanied by new modelling languages and terminologies that tend to make the world of UI specification even more complicated. Nevertheless, the idea of linking models with design is exactly what is necessary for designing interactive systems with high usability. Taken together, the solutions currently available and published are very useful in the context of this work (see Table 66). With regards to the common denominator presented in Chapter 4, the tool support described in Chapter 6 is based on existing notations and artefacts. It therefore avoids the fabrication of more and more new design languages, but in contrast is intended to simplify corporate UI specification by a concentration on essential and common resources.

Table 66: Lessons learned from related work

Tool(s)	**Lesson learned**
Axure, iRise	Support UI prototyping and detailed UI specification, and determine a prototyping-driven UI specification process with regards to stakeholder collaboration and iterative development; easy to learn due to imitation of MS Office functionality; check-in/check-out mechanism to foster collaboration and real-time feedback; storyboard as hybrid view to switch between text-based descriptions and design layer
CanonSketch, TaskSketch, WinSketch, WinSketch	Switching between the abstract and the detail can be simplified; early stages of design must also be incorporated in model-based UI specification tools; traceability is important; collaboration fosters creativity and innovative thinking;
SILK, DENIM, DAMASK	Computer-aided sketching of UIs can be superior to paper-based methods if based, for example, on a whiteboard metaphor and adequate input mechanisms; annotations and speech are saved to remember design decisions; versioning sketches helps to consider trade-offs and alternatives; whiteboard plus zooming helps to keep an overview of the UI specification and supports switching views; patterns support stakeholders without design background to develop good UIs;
Diamodl	Combination of the worlds of modelling and design in one tool, which is based on MS Visio; promising starting point for developing integrating solutions that fit the requirements of corporate UI specification; use of lightweight models of usage-centred design for corporate UI specification in the industry
SketchiXML , GrafiXML	Well-defined exchange format (usiXML) allows lossless transitions between work steps

5.8 Key Points

- The proposed common denominator (see Chapter 4) integrates a variety of modelling languages and design layers that can also be found in existing commercial tools and research prototypes. Accordingly, choosing lightweight and agile models for corporate UI specification (e.g. CanonSketch, TaskSketch, DIAMODL) seems to be an appropriate choice that enables non-IT experts to participate in the process. The models chosen must then be assembled into a traceable network of linked artefacts.
- Many tools focus on a small subset of models and are therefore unable to cover all aspects of UI specification. iRise and Axure are based on the vision that RE will eliminate abstract artefacts and instead concentrate on visual prototyping. This extreme perspective is foiled by tools such as CanonSketch or TaskSketch, which correctly take both models and design into account. With regards to interactive UI specification, the required tool support must cover all stages of the process, starting with sketches (e.g. DENIM) and then moving to mixed- and high-fidelity designs (GrafiXML, CanonSketch, iRise) on the one hand, and modelling with different means of expression on the other (e.g. TaskSketch, DIAMODL).
- Traceability and transparency are the most important aspects of sound UI specification. In interactive UI specification, travelling from the abstract to the detail is among the most frequent activities. As with iRise, the UI storyboard can be used as a mediator between design and modelling (see Chapter 4).
- Support for collaboration is an important issue in interdisciplinary, multi-stakeholder UI specification projects. The approach of using interactive UI specifications must therefore be supported by tools that can handle parallel iterative development.
- DENIM and DAMASK provide the opportunity to save versions of sketches, including the associated annotations. The advantages of versioning must be extended to all kinds of artefacts developed for the purpose of UI specification. Versions of models and designs capture the design knowledge and decisions. They enable UI simulation with different combinations of UI screens and allow an easy return to older design solutions or models.
- The whiteboard metaphor used in SILK, DENIM and DAMASK is a promising approach for interdisciplinary UI specification. Drawing on a canvas and developing handcrafted sketches is more inviting that applying formal tools that require the learning of specific languages or terminologies. Commercial tools such as iRise also employ zooming for switching between UI storyboard and detailed design. Together with a zoomable user interface (ZUI) approach, even very complex specification spaces can be captured on a whiteboard. Zooming then adds an additional dimension to the planar 2D-layer and enables movement from the overview to the detail. Because this characteristic of zooming is closely related to the multi-level structure of the common denominator, a ZUI is a promising paradigm for UI specification tool support.
- Being able to export the UI specification to an exchangeable format is necessary in order to feed programmers with detailed UI specification designs that can be reused. The usiXML description language is a promising approach for exchanging UI-related information between various tools. With regards to corporate UI specification, however, supporting commercial solutions such as Microsoft's XAML format is more reasonable. The tool support presented in Chapter 6 will nevertheless be equipped with both formats to be able to contribute to the usiXML supply chain of (Vanderdonckt 2008).

Chapter 6 INSPECTOR: Interactive User Interface Specification Tool

Interdisciplinary UI Specification Tool Remedy

In this chapter the experimental tool known as 'INSPECTOR' is presented in detail. The name INSPECTOR is the abbreviation for 'Interdisciplinary UI Specification Tool (Remedy)' and refers to the requirements of corporate UI specification processes.

From theory to practice

In Chapter 6.1 the main requirements for adequate tool support is summarized on the basis of the discussion in Chapters 2-3 and the identification of a common denominator for UI specification in Chapter 4. In addition, the findings from the analysis of related work in Chapter 5 are also taken into account in shaping a modern, innovative and supportive framework that bridges the gaps between stakeholders and their disciplines. Naturally, the experimental tool was created to translate perfectly the idea of interactive and prototyping-driven UI specifications from theory into practice.

Understanding the conceptual model of INSPECTOR

In Chapter 6.2, the most important aspects of the conceptual model of INSPECTOR are outlined. At this point, the metaphor of a whiteboard, which has already cropped up several times in the previous chapter, will be mapped to the requirements for INSPECTOR. As in the cases of DENIM and DAMASK (see Chapter 5), the potential of a zoom-based whiteboard UI will be explained in some detail.

Tool demonstration and technical background

After a summary of the technical background (see Chapter 6.3), the capabilities of the INSPECTOR tool, which was developed from 2006 to 2008 and presented to the UI specification community at several conferences and workshops, are demonstrated in detail in Chapter 6.4. For this purpose, an example from the automotive industry is used to work through a process of design consideration and UI specification. This process is part of the day-to-day operations of the business. Using the example of the specification process for parts of a car configurator for the Mercedes-Benz website, the advantages of the INSPECTOR approach are illustrated and then finally summarized in the key points of Chapter 6.5.

6.1 Adequate Method- and Tool-Support

The reality in UI development

Most UI development tools are inappropriate for supporting actors from different disciplines in designing interactive systems. They all possess their own particular inputs of UI artefacts expressed in their own formats, and these formats are generally heterogeneous and incompatible. It is therefore generally recognized by SE, IxD and BPM specialists that structured approaches are required to model, specify, and build interactive systems with high usability (Metzker 2002). Nevertheless, in many non-IT organizations (see Chapter 2), UI design is still an incidental or opportunistic by-product and appropriate methods are not sufficiently embedded in the overall corporate software-development process. Even if they are integrated, their contribution remains marginal, thus reducing the expected positive impact on software quality.

Low threshold, high ceiling and wide walls

As explained in (Memmel & Reiterer 2008; Memmel et al. 2008f), this reality can be explained by the fact that most integrated development environments (IDEs) are inappropriate for supporting actors from different disciplines in designing interactive systems. Formal UI tools prevent many actors from taking part in collaborative design if they do not have adequate knowledge of specific terminologies. On the other hand, being too informal leads to misunderstandings and conflicts in communication with programmers. Moreover, on further examination, many tools turn out to be more focused on requirements management than on providing support in extracting requirements from user needs and translating them into good UI design. After all, despite - or perhaps precisely because of - the vast functionality of many tools, the outcome is often unsatisfactory in terms of UI design, usability and aesthetics. This is described as the high threshold - low ceiling phenomenon of UI tools

(Campos & Nunes 2004a). Due to the lack of appropriate tools, many actors tend instead to use tools they are familiar with and that can be categorized as being low threshold (for application) - low ceiling (of results), a phenomenon observed in (Campos & Nunes 2007). In order to easily produce some results with reasonable efforts, an IDE should have a low threshold: the threshold with which one can obtain a reasonably good UI should be as low as possible. On the other hand, an IDE should have a high ceiling: the maximum overall performance of the IDE should be as high as possible. To these two dimensions, a third one - wide walls – is usually added (see Figure 90). An IDE should have walls that are as wide as possible, meaning that the range of possible UIs that can be obtained via the IDE should cover as many different UIs as possible. This cultural change must be supported by an integrating UI specification tool that enables the translation of needs into requirements and subsequently into good UI design.

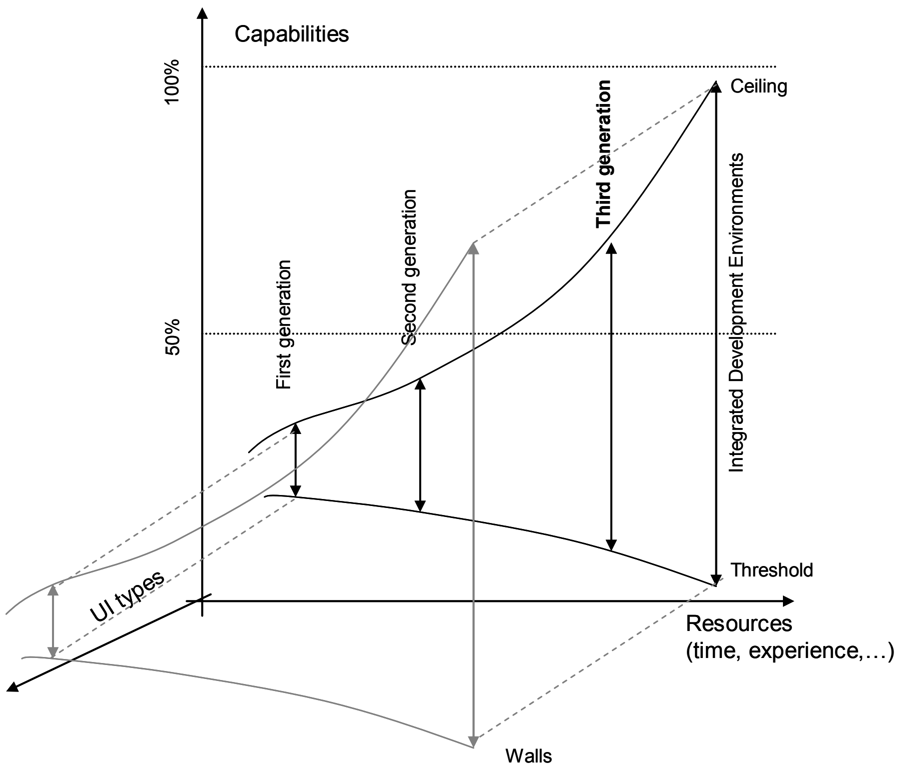


Figure 90: Threshold vs. ceiling vs. walls for expressing the capabilities of IDEs (Memmel & Reiterer 2008)

Design principles for UI tools

In accordance with (Campos & Nunes 2006), (Geyer 2008) summarized the most important guidelines that UI specification tools should follow:

- **Explorability**. UI tools should effectively support the explorative nature of design processes. As UI design processes lead to a variety of artefacts, it is necessary to provide means to creatively explore large design spaces.
- **Expressiveness**. As modelling and design relies heavily on visual presentation, the means of expression should not be limited. In expressing their ideas, designers need to be supported with a variety of informal and formal tools.
- **Guidance**. A design tool should guide the overall design process just enough to make it successful, but not so rigidly that it would constrain the actions of the designer. The employment of selected notations provides good guidance along the UI specification supply chain.
- **Desirability**. As design tools are used by designers and other stakeholders who like to contribute to the design process, they have to be engaging and look at-

tractive to become adopted in practice. The models and design artefacts employed need to invite stakeholders to apply them.

Creativity-driven UI specification

Explorability and expressiveness imply that UI specification tools should support abstract thinking and detailed modelling. Accordingly, stakeholders should be guided in travelling along a UI specification process that incorporates tasks involving thinking and building. In contrast to most UI tools, the real work style of stakeholders who take part in corporate UI specification is to be taken into account. With tool support that is appropriate for both the user and the problem, creative processes can be simplified. Creativity has been rightly recognized as a key to economic growth and social transformation. Following (Florida 2002; Florida 2005), the future is shaped by technology, talent and tolerance. Accordingly, support for creativity has to attract the most innovative minds and empower them to accelerate the pace of discovery and innovation (Shneiderman et al. 2006b).

A framework for creativity

According to the mega-creativity framework by (Shneiderman 2000; Shneiderman 2003), the inspirational, the structural and the situational models support the creative thinking necessary in UI development processes. The inspirational model promotes techniques such as brainstorming, free association and imaginative thinking. Creativity can emerge if stakeholders and designers can break away from their existing mindset and are enabled to perceive the problem from a new perspective. This can be supported by visual techniques that present loose relationships between artefacts, such as mind maps (Geyer 2008). The structural model supports creative processes by analyzing previous work. Visualizing how things work currently and modelling the status quo, for example in flow charts, is the key to understanding a problem. (Shneiderman 2000) recommends iterative structural modelling with incremental methods that support going back and making changes. The situational model considers the social context as a key part of creativity. Accordingly, the influence of social challenges, mentors and peers creates a strong desire to innovate in pursuit of recognition. According to (Shneiderman 2000), the context is incorporated by consultation with people and discussion with actors in the field.

Design principles for UI specification tools

Based on this creativity framework, (Shneiderman et al. 2006b) summarize concrete design principles for the development of tools that support creative processes. What distinguishes these principles from other UI principles is that they emphasize easy exploration, rapid experimentation, and fortuitous combinations that lead to design innovations. While some of these guidelines explicitly address UI tools, they also aim at improving general creative processes that require a combination of novel artefacts in the domains of computer programs, scientific writing, engineering diagrams or artwork. The most important of the guidelines that are relevant to UI specification are summarized in the following, using just very slightly adapted versions of the original wording by (Shneiderman et al. 2006b):

- **Support Exploration**. An important requirement for creativity is to be able to try out many different alternatives. Users must be encouraged to explore the UI specification space and need support for sketching and trying out dozens of ideas. A UI specification tool must also be simple and easy to use, so that project stakeholders can model different artefacts quickly. Finally, tools must be pleasurable and fun to use. When people are stressed or have to concentrate too much simply to use the tools, they will have fewer cognitive resources available for finding creative solutions to their tasks (Shneiderman et al. 2006b).
- **Low threshold, high ceiling, and wide walls (see Figure 90)**. Effective UI tools should make it easy for novices to get started (low threshold), but also allow experts to work on increasingly sophisticated projects and develop powerful results (high ceiling). The concept of wide walls means that tools should support and suggest a wide range of explorations. One strategy for achieving all three dimensions is to include elements and features that can be used in many different ways. The design challenge is to be specific enough so that users can quickly understand how to use the features (low threshold), but general enough so that users can continue to find new ways to use them (wide walls).

The tool should help users to learn how to use the features so that they can understand the variety of possible uses (Shneiderman et al. 2006b).

- **Support Many Paths and Many Styles**. Research distinguishes between 'left brain' thinkers (logical, analytical; people who can do maths) and 'right brain' thinkers (holistic, intuitive; people who can draw). In many organizations, the 'left brain' approach is extolled and viewed as superior to the 'soft' approach. With regards to bridging the gaps between the disciplines and interdisciplinary UI specification, a UI tool should pay special attention to making sure that technologies and activities are accessible and appealing to the 'right brainers' as well (Shneiderman et al. 2006b).
- **Support Collaboration**. An important implication of the diversity of stakeholders is the need to provide support for collaboration. In corporate UI specification, most creative work is done in interdisciplinary teams. It is important that the creativity-support tools allow each person to contribute by using their own talent. A UI specification tool should also allow team members to work on their own parts of the UI in parallel (Shneiderman et al. 2006b).
- **Support Open Interchange**. The creative process is often supported by a single tool but will more often require that the user orchestrate a variety of tools, each of which supports part of the task. This was well demonstrated by the analysis of related work in Chapter 5. The common solution is therefore that creativity-support tools interoperate with other tools. This includes the ability to easily import and export data in widespread formats such as XML in general, or usiXML (Vanderdonckt et al. 2004; Vanderdonckt 2008) in particular. Another approach is to integrate all necessary artefacts and models in one specification space. This can be achieved with a professional 'plug-in' architecture (Shneiderman et al. 2006b).
- **Make It As Simple As Possible - and Maybe Even Simpler**. Reducing the number of features can actually improve the user experience in working with UI tools. What initially seems like a constraint or limitation for many experts or programmers can, in fact, foster new forms of creativity – for example, through the various opportunities created by interdisciplinary collaboration. The goal is to develop tools that offer the simplest ways to do the most complex things (Shneiderman et al. 2006b).
- **Iterate, Iterate - Then Iterate Again**. Iterative design using UI prototypes is also important for creativity-support and UI specification tools. With creativity-support tools, users should be encouraged to play around with the materials and try out several alternatives, to change direction in the middle of the process, and to take things apart and create new versions (Shneiderman et al. 2006b).
- **Design for Designers**. In designing new creativity-support tools, it is important to design for designers (i.e. stakeholders) – that is, to design tools that enable others to design, create, and invent things. With regards to corporate UI specification, this also means that non-IT experts should be able to design UIs without specific knowledge of SE or IxD. As outlined in Chapters 4 and 5, paper and pencil is a tool commonly used by creative practitioners such as architectural designers. Hand-drawn sketches and diagrams have been found essential for architects' creative contemplation. The process of drawing helps designers who are engaged in 'reflection-in-action' (Shneiderman et al. 2006b).

Bridging the gaps with creativity

UI specification and creativity-support tool research can contribute to bridging multiple disciplines including SE and IxD. Researchers from one discipline may not appreciate the relevance of, and their dependence on, activities outside their discipline, thereby failing to take advantage of progress already made by others. But interdisciplinary work can accelerate progress for all and improve quality. With adequate tool support, both firms and individuals are able to innovate in product development (von Hippel 2005). Developing an understanding of how work in one discipline is useful to another helps to foster creativity and UI quality.

Agile semi-formal UI specification

With the common denominator, semi-formal and rather approximate and pragmatic means of modelling and UI design were established as a shared vehicle in corporate UI specification. Because agile notations are integrated in the common denominator, corporate UI specification processes - which are based on prototyping-driven interactive UI specifications as proposed in this thesis - could well be categorized as being agile. Taking into account the design room and whiteboard metaphor, this affinity is strengthened by agile practices such as 'Display Models Publicly'. This practice also underlines the importance of shared design rooms in UI development (Gundelsweiler et al. 2004).

Wall of wonders

With INSPECTOR, this design room is moved into the virtual world, making it an electronic 'wall of wonders' (Gottesdiener 2002). As explained by (Ambler 2006b), computer-aided support for sharing artefacts does not contradict the nature of agility.

"You should display your models publicly, often on something called a 'modeling wall' or a 'wall of wonders.' This supports open and honest communication in your team because all of the current models are quickly accessible to them, as well as with your project stakeholders because you aren't hiding anything from them. Your modeling wall is where you post your models for everyone to see;[...]. Your modeling wall may be physical [...]. Modeling walls can (also) be virtual, such as an internal web page that is updated with scanned images." (Ambler 2006b)

Addressing agile principles and practice

Naturally, INSPECTOR can do more than just safekeep scanned images of UI-related artefacts. Moreover, considering the UI specification method and the conceptual model underlying the proposed solution for tool support, INSPECTOR also needs to match the most important agile principles and practices (Ambler & Jeffries 2002; Ambler 2004b) outlined in Table 67 and Table 68. As discussed in (Memmel et al. 2007a), agile principles and practice help to bridge the gaps between the disciplines. Accordingly, designing INSPECTOR with regards to agile values strengthens its suitability for interdisciplinary, model-based UI specification processes.

Table 67: The compatibility of model-based semi-formal UI specification with some important agile principles

Agile principle	**Compatibility with model-based semi-formal UI specification**
Model with a Purpose	Switching between different models allows the design rationale of a requirement to be understood and its origin to be traced; modelling therefore adds significant value to the UI specification process
Maximize stakeholder ROI	By allowing all stakeholders to participate in the UI specification process, they can contribute to the final solution without media discontinuities, loss of precision or overwhelming UI tools; interactive UI specifications are intended to prevent costly late-cycle changes and rework
Multiple models	No single model is sufficient for all UI specification needs. By providing different modelling languages for different stages of the process, several ways of expressing problems are available; the relationship or associations between the models is to be clear
Rapid feedback	Because the model-based UI specification process is driven by prototypes, interactive UI specifications can usually provide 'living simulations' of the UI at any time
Embrace change	Models should allow easy enhancement or change. Agile models only need to be approximate and tend to be rather pragmatic. They therefore allow changes without requiring the user to be excessively precise
Software is your primary goal	The goal of interactive UI specifications is to produce a specification-level UI design and reusable code for suppliers, rather than extraneous documentation. With regards to this goal, a model-based approach just focuses on the models to be developed and on understanding the design solution

Dimensions of work style

In order to describe the work style of stakeholders in UI specification processes, the dimensions defined by (Campos & Nunes 2005a; Campos & Nunes 2006) help to structure the most important activities of UI design and specification processes. The dimensions can be grouped under three main categories: (1) notation style-related dimensions (perspective, formality and detail), (2) tool usage style-related dimensions (traceability, functionality and stability) and (3) collaboration style-related dimensions (asynchrony and distribution). Each dimension is highly relevant to UI specification tool support and accordingly is taken into account in the summary of tool requirements outlined in Table 69.

Table 68: The compatibility of model-based semi-formal UI specification with some important agile practices

Agile Practice	Compatibility with model-driven development
Active stakeholder participation	The common denominator invites all stakeholders to take part in modelling and designing interactive systems. Sketches and UI prototypes are a language everybody understands. Easy-to-use tool support that matches one's usual work style fosters involvement
Apply the right artefacts	Some modelling languages are inappropriate for describing specific parts of the system. The common denominator provides a sound set of artefacts for all stages of UI specification
Iterate to another artefact	When a specific artefact is unable to express certain parts of the system, a linked and associated network of artefacts supports switching to a different means of modelling
Model in small increments	Interactive UI specifications can be developed iteratively and incrementally; models and UI design can be charged with different levels of detail and become mature during later stages
Model with others	Stakeholders can model according to their expertise. Different models are combined to simulations and the interactive UI specification

Overview of requirements for a new UI specification tool

With regards to the requirements for developing tools able to support creativity and interdisciplinary UI specification, several detailed needs can be deduced. Table 69 summarizes all the important key requirements for a new and innovative UI specification tool that is adequate for the kind of corporate UI specification processes outlined in Chapter 3. Accordingly, the following overview of tool requirements takes the different capabilities of stakeholders into account. The requirements in Table 69 also address the models and prototyping artefacts that make up the common denominator. They need to be properly integrated and visualized, and team members and stakeholders of all kinds need to be able to use them easily (low threshold). Nevertheless, the way the specification artefacts are integrated must allow high quality, expressive and detailed UI specifications (high ceiling) for a variety of corporate interactive software systems (wide walls).

Table 69: Requirements for interactive UI specification tools on the basis of (Campos & Nunes 2006; Shneiderman et al. 2006a; Campos & Nunes 2007; Memmel et al. 2007a; Geyer 2008); notation for tracing requirement to possible technical solution adapted from (Memmel et al. 2008b)

Identified Need	Principle(s)	Tool requirement
All **kinds of actors** must be able to proactively take part in the UI specification	Low threshold Make it as simple as possible - and maybe even simpler Support many paths and many styles Active stakeholder participation Model with others Maximize stakeholder ROI	Incorporate the common denominator. Support for design assistance and creative thinking for everybody. Guide through the process by offering navigation aids. Respect discipline-specific tool usage and knowledge. Employ interaction concepts familiar to all actors and compatible with current work practice. Employ metaphors to map tool support to corporate work style.

Identified Need	**Principle**	**Tool requirement**
Stakeholders must **prevent costly late-cycle changes** and specify the UI in detail to ensure UI quality	Explorability Support exploration Maximize stake-holder ROI	Provide adequate means of UI prototyping to allow up-front usability evaluation of look and feel; help to detect and document usability issues. Implement means to easily provide **feedback** on the UI and the underlying models.
Stakeholders must be able to provide **traceability** of the design rationale and maintain **transparency** during the translation of models into UI design and vice versa. Stakeholders need to work at **different levels of detail** and must be able to implement smooth **transitions** from problem-space concepts to solution space, and back	Traceability Guidance Expressiveness Support many paths and many styles Model with a purpose Multiple models Iterate, iterate - then iterate again Apply the right artefacts Iterate to another artefact	Enable **keeping track** of all artefacts during the design process and allow structured organization. Provide functions to **interconnect artefacts** to allow switching back and forth between different (levels of) models and UI designs. Keep **design decisions** for later reference to extend traceability. Effectively communicate final designs and their **rationale.** Provide high-level, abstract models to facilitate problem solving and realistic (or figurative) prototypes to address high-detail design issues. Allow **smooth progression** between the abstract and detail. Support moving from high-level descriptions of the UI (e.g. navigation maps) to detailed screens Provide zoom-based **navigation aids** to trace models to the concrete UI element and to trace a UI element to the model that it implements; implement versioning; mark the version relevant to the UI specification.
Stakeholders must be supported in finding and shaping the most promising **UI design solution**, as well as specifying different kinds of interactive UIs	Explorability Support exploration Embrace change High ceiling Wide walls Functionality Design for designers	Encourage **exploration** by providing means for building both abstract and detailed **UI prototypes**. Make sure designing different versions of a UI is easy and quick, as is making changes to it. Extend the **designers mind** by externalizing design **vision**. Provide a huge variety of UI elements, models and templates to support the design of different types of UIs. Support designers in sketching, especially in the early stage of the process. Provide templates and UI patterns to support and guide UI design.
Stakeholders need to establish a **shared understanding** of the problem-domain **Bridge the gaps** between the disciplines and increase **communication** and **collaboration**	Perspective Expressiveness Formality and detail Support collaboration	Provide different **perspectives** on artefacts. Allow stakeholders to model with different levels of **formality** (informal to semi-formal) to foster the comparison of design alternatives and creativity. Concentration on a specific subset of modelling artefacts, which best leverages collaboration. Provide the modelling languages and design artefacts proposed by the common denominator (see Chapter 4. Allow **sharing** and **distribution** of design artefacts among team members; imposes a structure, containers and persistence; artefacts do not have to disappear after meetings.
Integrate UI specification into UI development **supply chain**	Support Open Interchange Software is your primary goal	Provide **output** in XML or other common format for description of UI specification (e.g. usiXML or XAML).

6.2 Conceptual Model

The common denominator of modelling languages presented in Chapter 4 consists of modelling languages and UI prototypes of different fidelity. The different levels support travelling from the abstract to the detail, and back. Through an arrangement in layers, the common denominator maps the application of different UI specification artefacts on a hierarchy of notations. In this chapter, the conceptual model for the innovative and experimental UI specification tool, which is presented in Chapter 6.4, is explained in detail. The tool support is based on the hierarchy of notations for UI specification and on two fundamental UI concepts:

- **A design room and whiteboard metaphor**, which is meant to take into account the familiar semi-formal work style in corporate UI specification processes as well as the requirement to extend the mind of the designers by sketching ideas and vision at early stages. The idea of an electronic whiteboard has already been applied in earlier concepts for UI specification tools (see Chapter 5) and successfully takes into consideration the usual stakeholder environment, namely 'design rooms' or 'war rooms'.
- **A zoomable user interface (ZUI)**, which maps the hierarchy of modelling languages and design artefacts extremely well on the screen of UI specification tool support. With regards to the whiteboard metaphor, the ZUI will visualize the overview of the UI specification space on the one hand, and the detail of specific artefacts on demand on the other hand. This concept copies the interaction of stakeholders with whiteboards: moving closer to drawings that have been developed reveals more detail, while stepping back and looking at several drawing, for example on several whiteboards, provides an overview of all artefacts modelled.

Although the INSPECTOR tool relies heavily on a zoom approach, the topic of ZUIs will not be expanded on in this thesis. A more detailed discussion of ZUIs can be found in (Memmel 2005; Buering 2007), for example.

6.2.1 The Design Room And Whiteboard Metaphor

Design and war rooms

In corporate project settings, stakeholders usually share one or more rooms. It is quite common to use this kind of room for organizational discussions as well as for brainstorming and problem-solving. Because of the variety of purposes such a room can have, it is often called a 'design room'. Some shared workspaces are also called 'war rooms', because stakeholders use these rooms for facing trade-off dilemmas and making difficult design decisions. Such rooms can significantly facilitate communication and collaboration between stakeholders, which is of the utmost importance for corporate UI specification teams (see Table 69). Design rooms (see Figure 91), such as described by (Karat & Bennett 1990), are the *"public memory and conscience" (Beyer & Holtzblatt 1997), p. 204* of the team and provide an environment that allows discussion and the exploring of ideas. The walls of the room serve as a repository for all design artefacts. The visitor to the room can easily discover the details of the existing artefacts or gain an overview of all documents. If artefacts are hand-drawn, they can be easily exchanged with new documents quickly created during design sessions. For example, rapid UI storyboarding can take place in this way for exploring multiple alternative paths through a user task.

The design room as metaphor

The concept of the design room (see Figure 91) can work as a valuable metaphor in UI tool development. The room addresses the contextual work style of stakeholders very well. The idea of the design room is therefore also interesting for the world of electronic artefacts and virtual shared workspaces. With regards to interactive UI specifications (see Chapter 3), an electronic specification space integrates all of the important documents. As with the design room, different stakeholders should be able to access this space in order to look at, discuss and change the artefacts it stores. Because the artefacts are maintained during the whole UI specification lifecycle, the design room increases the traceability between models and designs. By drawing associations between artefacts pinned on the wall, different documents become related to each other. Compared to small desktop work places, the zoomable design room enables the visualization and comparison of all relevant data.

"Writing ideas on the wall is a way of interacting with the data. It provides a way to capture design ideas so that the design team can act on them, and everyone can feel they contributed something to the design. Posting ideas clears people's heads to go on to something new or to build an idea up into something larger." (Beyer & Holtzblatt 1997), p. 202

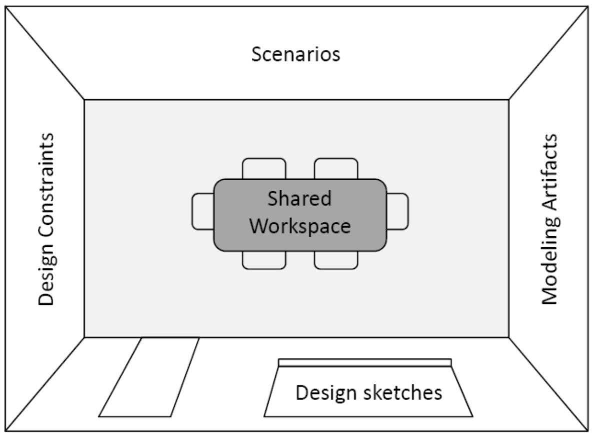


Figure 91: A design room; from (Geyer 2008) based on (Preece et al. 1994)

Naturally, in applying the design room as a metaphor for a UI specification tool, the challenges of this approach must also be considered. The number of artefacts that can potentially be pinned up on the wall must not overwhelm stakeholders. In a real-world setting, the team will usually try to cluster and manage the artefacts. An electronic counterpart of a design room must also provide means for structuring and filtering the specification space, for example in order to enable a focus on aspects relevant at a particular point in time.

Issues with metaphors

In IxD, the fundamental idea of metaphors is to transport concepts from the real world to the world of the computer. This includes the mapping of the way humans think (Benyon et al. 2005) about interaction concepts in the electronic world. In interacting with the UI of an application on a computer, the human user can refer to concepts known from the real world if the metaphor is adequately chosen. Then, the UI metaphor supports the user in developing a mental model of the system. If the response of the system to the user's action accords with the expectations inferred from the metaphor, the user is likely to learn the software application more easily. In practice, operating systems and complex software applications often employ several metaphors to decrease the threshold of using and understanding their functionality and purpose (Preim 1999). Combinations of several metaphors are feasible, but must not lead to artificial or arbitrary expressions of UI behaviour (Cooper et al. 2007). Ultimately, a successful metaphor combines characteristics from the real world and characteristics from the interactive system. For example, the design room can have much more functionality as a computer-supported application, because there are no physical limitations and the UI specification space is infinite. Even if the metaphor is indeed used for a UI specification tool, it is still acceptable to talk about a design room (albeit an electronic one). It is, however, important not to try to copy the real room without any loss or adaptation (Hudson 2000). This means that the UI of a specification tool does not need to present the virtual design room in terms of walls, pins and artefacts. The concept of this tool support should concentrate more on the basic idea of allocating artefacts in a large specification space, allowing different stakeholders to access and manipulate the embedded information.

Combining design room and white-board metaphor

For the conceptual model of a UI specification tool that addresses the requirements outlined in Table 69, the combination of the design room metaphor with a whiteboard metaphor is obligatory. Whiteboards are a frequently used medium for collaboration and rapid prototyping of concepts and designs. Whiteboards are widely applied in agile processes and also as the basis for sketching and sketch-based tools as discussed in Chapters 4 and 5. The whiteboard promotes creative thinking, as it provides an informal space for collecting and expressing design ideas.

"While the focus of the design process lies in the creation, manipulation and relation of artefacts on the whiteboard, collaborative features of the room metaphor may also turn out helpful in physical design. The whiteboard metaphor primarily offers features that support a creative design process. Its whiteboard features efficiently facilitate informal means of expression […]." (Geyer 2008)*, p. 70*

Computer-supported whiteboards in UI specification

Due to the nature of real whiteboards, the metaphor must break physical borders to add value to the activities of UI specification. Normal whiteboards just provide some limited features for arranging and relating different artefacts. Accordingly, frequent changes to elements of the whiteboard might cause many changes and much rework. A computer-supported whiteboard can ease the process of relating and linking artefacts, as well as maintaining different versions of artefacts. This eases the comparison of design alternatives and provides much better support for keeping track of changes and interdependencies between models and UI design. Having unlimited space enables the user to draw gigantic diagrams and designs.

"If you are using a real whiteboard, you may go up to a diagram on the board and say 'I think that should go there'. As you say the words 'that' and 'there', you point at the relevant parts of the diagram. This is called deictic reference or simply deixis."(Dix et al. 2003), p. 682)

Design room and whiteboard hand-in-hand

In summary, the design-room metaphor, in combination with the whiteboard metaphor, also addresses demands for interdisciplinary stakeholder collaboration. Both of these electronic metaphoric concepts function as shared spaces, and stakeholders can discuss different artefacts. Interdisciplinary communication is therefore supported by a shared deixis between actors (Dix et al. 2003). The deixis depends on shared artefacts for this communication (see Figure 92). The artefacts can support communication in terms of underlining verbal arguments or by being the vehicle of communication themselves.

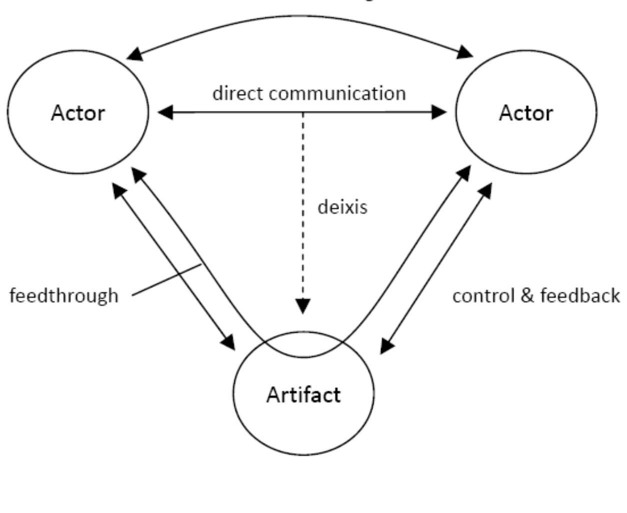


Figure 92: Control and feedback from shared artefacts; from (Dix et al. 2003; Geyer 2008)

"As actors control and manipulate artefacts, others may observe these actions and respond with feedback."(Geyer 2008), p. 91

With linking and association features, users of a tool designed in this way are able to easily navigate between artefacts spread over a huge UI specification space, i.e. the design room. Interrelated artefacts thus provide a means for tracing requirements and understanding the design rationale due to more transparency. Different aspects of interest can be clustered and, on demand, the content is moved into the focus of those stakeholders discussing or working on the issue. Creating or changing artefacts then brings in the whiteboard metaphor, because the manipulation of artefacts in the virtual design room takes place using the work style known from the whiteboard, namely wide freedom of creativity and unconstrained means for diagrammatic modelling and UI prototyping.

"So different representations allow us to see different things about a design. Furthermore different kinds of formal representation can allow yet more views on the artefact during design." (Dix 2003), p. 6 (437)

6.2.2 Zoom-Based User Interface Specification Tool-Approach

Two-dimensional UI specification space

With regards to the selection of the design room and whiteboard metaphor, the right UI paradigm has to be found in order to successfully map the concepts of both metaphors to the interface of a UI specification tool. As explained earlier, the employment of both metaphors does not mean that a computer-supported design room and whiteboard need to be displayed in a way that resembles the real word. In contrast to a real room, the concepts have to be transferred into the two-dimensional space because the artefacts that will be developed for UI specifications will also be two-dimensional, and a different solution is likely to make the tool support unnecessarily awkward (Heim 2007). For example, (Cockburn & McKenzie 2002) found that users' ability to quickly locate information items decreases in 2.5 or 3D (see Figure 93). In their evaluation study, users experienced the 3D interface as being more cluttered and less efficient (Buering 2007).

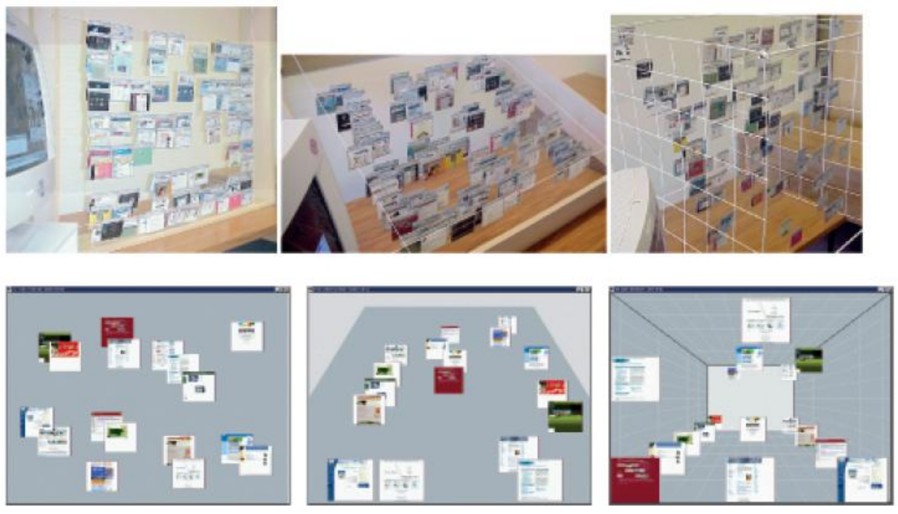


Figure 93: Set of physical and virtual data mountain systems with different dimensionality factors (Cockburn & McKenzie 2002)

Scrollable specification space

Following (Mynatt 1999; Pook et al. 2000; Pook 2001; Baudisch et al. 2002; Cockburn et al. 2006), the interface of a specification tool could be based on the concepts of 'scrollable space', 'segmented space' and 'zoomable space'. The scrollable canvas may be familiar to most users (Mynatt 1999) when it comes to visualizing information spaces that are considerably larger than the screen, but comes with several limitations that need to be overcome when designing and specifying large corporate interactive systems:

"(Using scrollable space), users are concerned about losing track of artefacts in the virtual scrollable space, when it is only three to four times larger than their real whiteboard. Scrollable spaces are increasingly awkward to handle, as navigation is unidirectional and artefacts are spread along a wide range of screen space." (Geyer 2008)*, p. 76*

Segmented UI specification space

A solution that segments the specification space can overcome some of the limitations of scrollable space, but also introduces some critical issues.

"The most common workaround (in displaying information spaces larger than the screen) is to display a fraction of the data and then let the users linearly navigate the off-screen space via scrolling. However, for identifying important information in a large data set this approach is tedious, error-prone, and particularly slow." (Buering et al. 2008)*, p. 1*

Although the view can be switched by visual references or thumbnails that are offered (see Figure 94), the separation of content is unable to externalize important interrelationships between artefacts. Because traceability and transparency are very limited in this case, segmented spaces are also not very suitable for UI specification tool support.

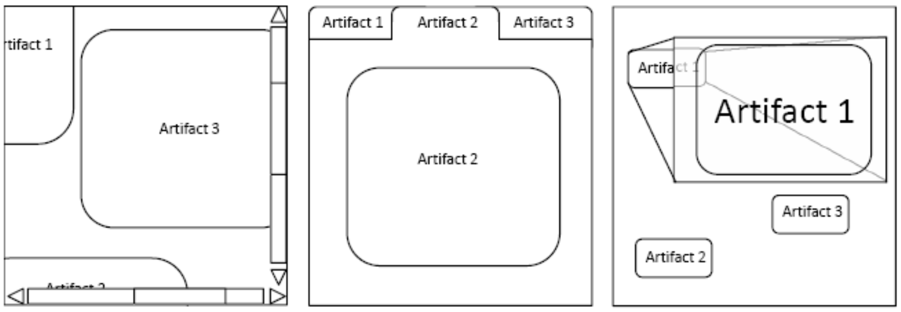


Figure 94: Options for spatial navigation between artefacts on the whiteboard: scrolling (left), switching tabs (centre) and zooming (right); from (Geyer 2008)

Zoomable UI specification space

Linking up with the wide experience in research with ZUIs, for example the work of (Grün et al. 2005; Gundelsweiler et al. 2007a; Buering et al. 2008), a zoom-based specification space comes with several advantages. Previous research indicates that ZUIs outperform scrolling interfaces on desktop computers in terms of user performance and preference (Kaptelinin 1995; Björk et al. 2000; Gutwin & Fedak 2004). A ZUI supports interaction with different views of requirements and therefore helps to build a cross-discipline visual software specification.

"While zoomable user interfaces have been discussed since at least 1993 (Perlin & Fox 1993), no definition of a zoomable user interface has been generally agreed upon. [...] We consider the two main characteristics of zoomable user interfaces to be (a) that information objects are organized in space and scale, and (b) that users interact directly with the information space, mainly through panning and zooming". (Hornbæk et al. 2002), p. 4

"The concept of zoomable user interfaces (ZUIs) has been found to improve the user performance for a variety of exploration scenarios. ZUIs facilitate data presentation based on the assumption that navigation in information spaces is best supported by tapping into our natural spatial and geographical ways of thinking" (Buering et al. 2008)*, p. 1; based on (Perlin & Fox 1993)*

Space-scale approach

To implement this approach, objects on the canvas must be organized in space and scale. For example, one could zoom-out from a high-fidelity UI prototype to a low-fidelity UI prototype and continue to drill down to modelled, semi-formal diagrams that describe the UI and its contents (Memmel et al. 2007c). In addition to zooming, panning (navigating sideways at constant scale) helps to explore artefacts on the same level. This approach is closely related to the slider for switching between different levels of sketching and UI storyboarding in DAMASK (Lin 2003b) or DENIM (Lin et al. 2000; Lin et al. 2001; Newman 2003). Another popular example of space-scale visualizations is the various TreeMaps-implementations, for example by (Shneiderman 1998; Bederson et al. 2003; Blanch & Lecolinet 2007). These applications underline the capability of space-scale visualization in visualizing large amounts of data in a usable way, which is highly relevant to complex UI specification spaces.

For UI specification, a ZUI needs to be able to intuitively map the dependencies and transitions of different models. In order to gain an overall and detailed view of the system, all stakeholders must be able to navigate between the different abstraction levels and to switch between the models. As it is necessary to support various stakeholders in producing high-detail interactive UI specifications, the zoom-based tool approach of (Lin 2003b), for example, has to be adapted with respect to the kind of modelling languages embedded. Accordingly, tool support for interactive UI specification can build upon previous work (see Chapter 5). The tool presented in Chapter 6.4, which is known as INSPECTOR, employs the ZUI approach for switching between different levels of fidelity through a visual drill-down process.

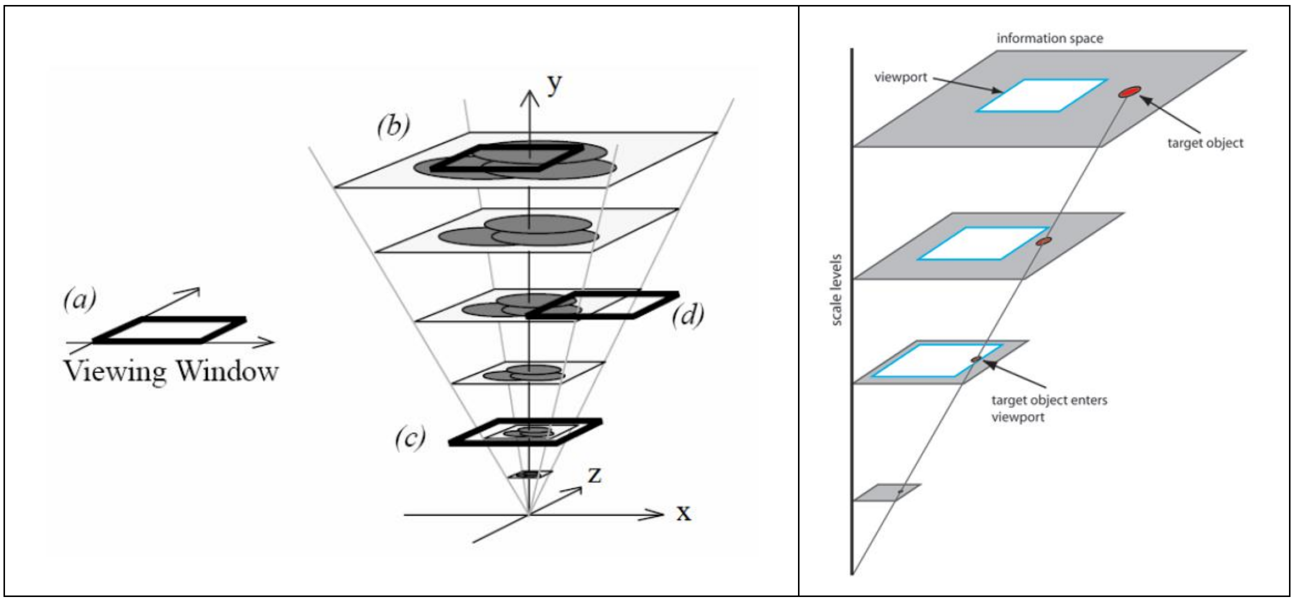


Figure 95: The construction of a space-scale diagram (left): from (Cockburn et al. 2006) based on (Furnas & Bederson 1995); space-scale diagram visualizing a zoom-out operation to locate a target object (right): from (Buering et al. 2008)

Zooming and panning

The concept of a ZUI is illustrated in Figure 95. The space-scale diagram, based on (Furnas & Bederson 1995), shows a viewing window (a), which is shifted around the 3D diagram to obtain all possible views of the original 2D surface by panning and zooming operations. In Figure 95, left, (b) shows a zoomed-in view of the circle overlap, area (c) represents a zoomed-out view including the entire original picture, and (d) is a shifted view of a part of the picture. Figure 95 (right) illustrates this concept in a model of a 2D information space that is viewed at different magnification levels. The viewport, i.e. the portion of the information space that is visible, is shown by the blue rectangle. The larger grey area is the off-screen space. If the user needs to navigate to the red target object, he zooms out until the target object enters the viewport. Once the object is successfully located, the user can access the object's details by zooming back in (Buering et al. 2008). The space-scale concept can also be described in terms of a portal to information:

“A portal is an object in the form of a rectangle that includes the coordinates, x, y, and scale, of the region of the virtual world that is to be shown in the portal when it is first displayed. The display of the contents of the portal, and panning and zooming by the user in the portal, are handled by the client.”(Pook 2001), S. 168

Principles and concepts of ZUIs in a nutshell

ZUIs offer panning and zooming as the main techniques for direct interaction and navigation (Perlin & Fox 1993). The appearance of information in ZUIs is mainly based on a linear scaling of objects (geometric zooming), and also on displaying information in a way that is dependent on the scale of the objects (semantic zooming). Various other zoom techniques support interaction with ZUIs. For example, automatic zooming organizes selected objects automatically at the centre of the UI (Furnas & Zhang 1998). Animated zooming supports the user in exploring the topology of an information landscape and in understanding data relationships (Bederson & Boltman 1998; Bederson & Boltman 1999).

“Two different kinds of zooming can be distinguished. Most common is geometric zooming, in which objects are simply magnified. Zooming in, the object’s size increases, and vice versa. This approach is found in many standard software applications such as PDF readers or image editors. Semantic zooming, in contrast, is a more sophisticated concept, in which objects change their appearance as the amount of screen real estate available to them changes. [...]Overall, the goal of semantic zooming can be summarized as providing the users with the most meaningful object representation at each magnification level.” (Buering et al. 2008), p.20

Another valuable zoom technique is the goal-directed zoom (Woodruff et al. 1998). Goal-directed zooming means that users choose a representation of an object, and the change in scale and translation is performed automatically by the system. In this way, navigation tasks can be decreased and an application can make sure that objects are always displayed at an optimal scale factor (Buering et al. 2008). At the same time, hints about the current zoom factor and the current position in the information space can be given in order to avoid disorientation (Pook et al. 2000). Moreover, a usable ZUI will prevent ‘desert fog’ (Jul & Furnas 1998). Desert fog describes regions of the zoom-canvas UI that do not contain information about navigation possibilities. If users enter such an area, they cannot decide to what extent they need to pan and/or zoom in order to leave the area and navigate towards their real target in the information space. Accordingly, a ZUI has to provide links to areas of interest in every situation in order to avoid disorientation. In this context, (Jul & Furnas 1998) also introduced ‘critical zones’. The ZUI should provide hints to relevant information even if the corresponding viewport has not yet been reached. Another common way of supporting the user’s cognitive map of the information space is an overview window. Users can develop mental spatial maps faster in environments where they can quickly and simply view the entire information space (Ware 2004). Consequently, zooming and overview+detail features can be combined (Cockburn et al. 2006). From the analysis of various studies, for example (Card et al. 1999; Baudisch et al. 2001; Hornbæk et al. 2002; Plumlee & Ware 2006), it can be deduced that, although there is a performance degradation due to visual switching between different views, an overview window is helpful when the complexity of the information space is too great to be held in visual working memory (Buering et al. 2008).

Mapping the hierarchy of UI modelling to a zoom-based interface

In summary, ZUIs simplify the illustration of hierarchies by a representation using different levels of scaling and enlargement. This advantage is very appropriate for visualizing a hierarchy of interdependent modelling languages as developed with the common denominator for UI specification (see Chapter 4).

“ZUIs are often used on hierarchically structured datasets with a known structure. Creating an information space for a ZUI requires the developer to provide graphical objects visible in the top level view of the space that summarize those objects found when users zoom. These objects will then summarize those objects to be found as users continue to zoom. A hierarchical structure is thus created.”(Pook 2001), p. 144

Ultimately, the different UI specification artefacts that are supposed to be contained in interactive UI specifications need to be arranged on the canvas of the zoom-based UI specification tool. Figure 96 shows the mapping of UI-related specification artefacts to different zoom levels of a hierarchically assembled specification space.

Arranging artefacts

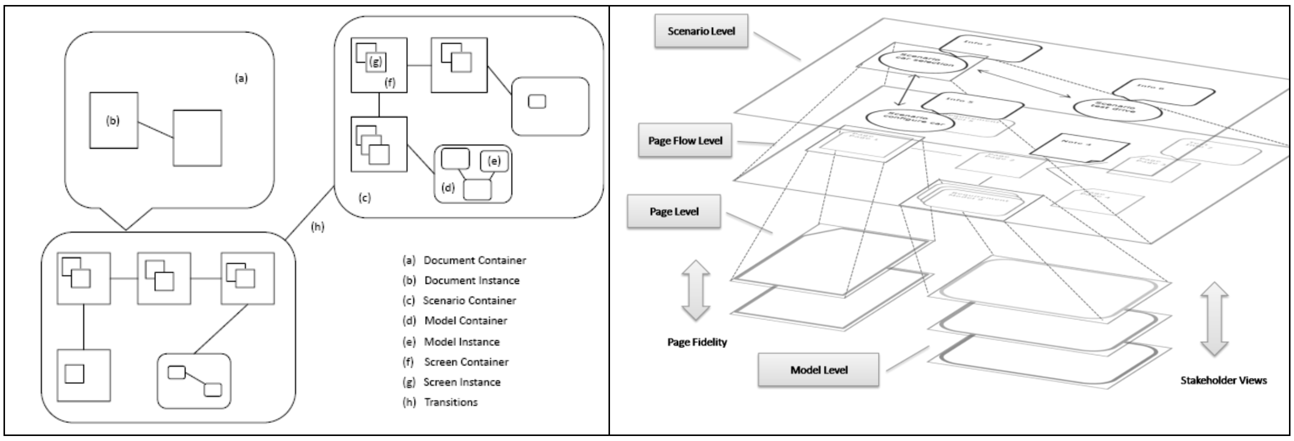


Figure 96: Nesting of UI specification artefacts (left) and layers of the specification space; from (Memmel et al. 2007b; Geyer 2008)

The idea of nesting objects (see Figure 96 and Figure 97) comes from the clustering in tree-like visualizations such as proposed by (Good 2003). Clustered objects provide overview and enable efficient exploration by zooming. The inherent spatial structure of ZUIs helps to increase the degree to which huge of amounts of information are withheld until the user explicitly asks for a visualization of specific parts of it. In interactive UI specifications, the information can be clustered with regards to the different kinds of models (for example, user models, task models or behaviour models) and different levels of UI prototypes (ranging from sketch-based design to specification-level UI presentations).

Nested structures and clustering

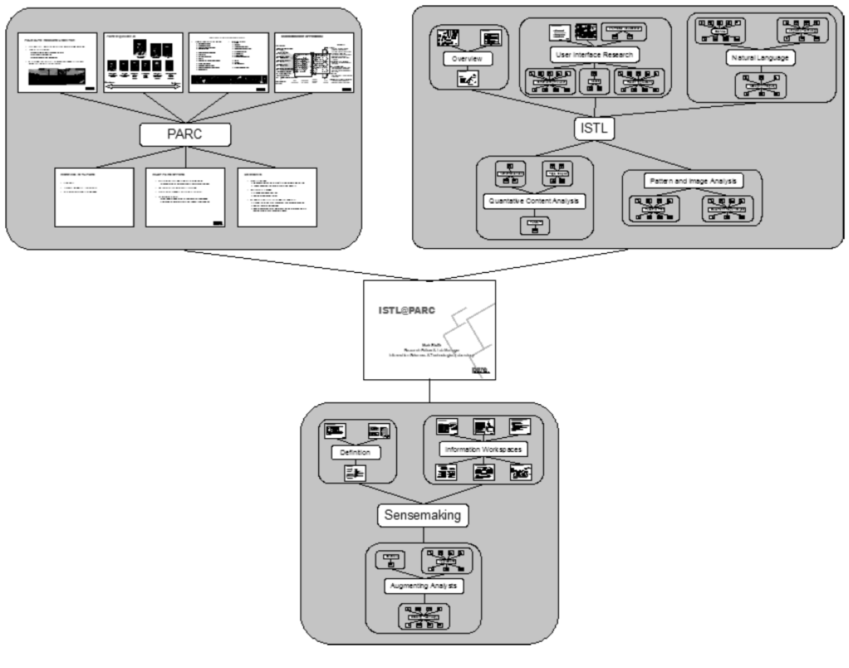


Figure 97: An example of an automatic layout of nested elements with a network format (Good 2003)

Zooming operations allow exploration of the specification space, and navigational aids of the kind discussed above support the user in keeping overview and transparency. In this context, arranging related artefacts close to each other will help to decrease panning and zooming activities.

Zoom-support for travelling along the UI specification supply-chain

Summing up, because ZUIs organize information in space and scale, they perfectly match the requirements of UI-related artefact management even in complex UI specification projects. The zoom approach helps to implement the design room and whiteboard metaphor in INSPECTOR, as the interaction with the ZUI is similar to navigating in a design room or in front of a whiteboard. As ZUIs are suitable for hierarchical representations, they support visualization of the multiple degrees of abstraction that accompany the UI specification process. ZUIs enable users to perceive spatial relationships between artefacts and to harness human perception capabilities in landmark navigation, all of which is beneficial for creative thinking and free association. The ZUI approach therefore supports the required increase in traceability and transparency. With a zoom-based UI specification space, stakeholders can develop interactive UI specifications that consist of a huge variety of different artefacts for user modelling, task modelling and behaviour modelling, as well as UI designs. The artefacts can be interrelated through links and associations, thereby allowing smooth travelling along the UI specification process (see Figure 98).

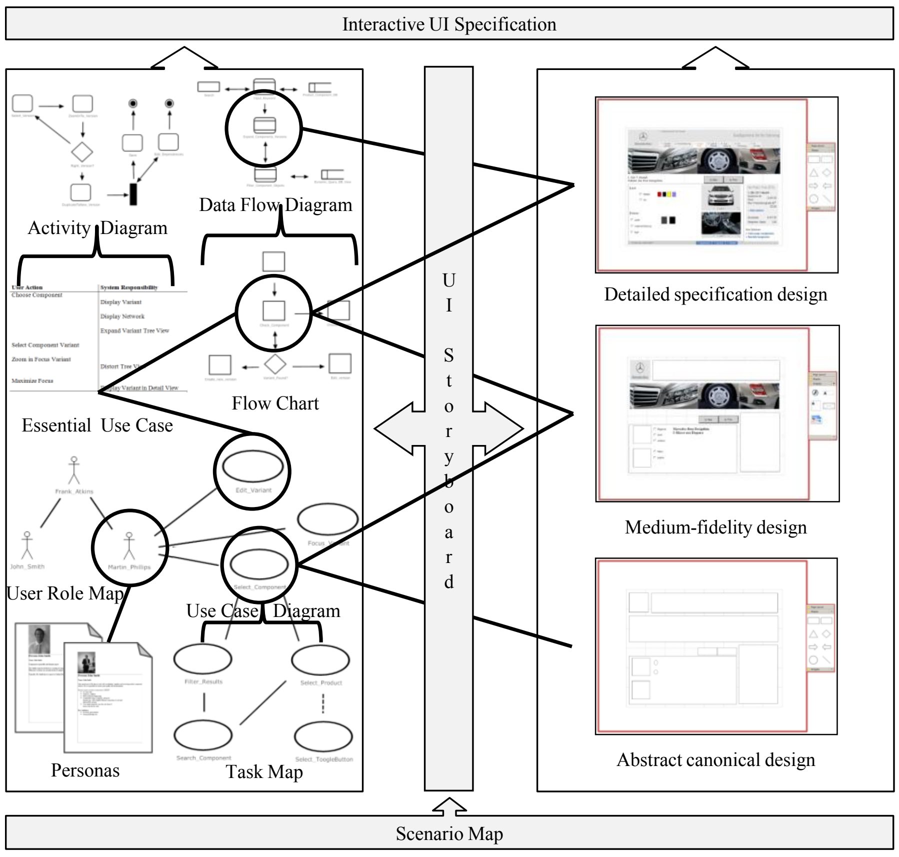


Figure 98: Correlation of models and UI designs; exemplified modelling and design throughput; as published in (Memmel et al. 2008f)

The role of storyboards and scenarios

INSPECTOR implements the common denominator for UI specification almost one-to-one, but takes UI storyboards out of the hierarchy of artefacts. As explained in Chapter 4, the UI storyboard can function as a mediator between the world of UI design and the world of notations ranging from text-based to diagrammatic. As proposed, scenarios are intended to be the starting point for the UI specification project, because they express visions, initial ideas and (business) goals. The scenario map is particularly well suited to work on the early stages of UI specification processes. Usability and user-experience goals, business and design vision and reusable requirements can be captured within the information bubbles at the scenario layer (Memmel & Reiterer 2008).

Panning and zooming trough the specification space

In this context, scaling objects linearly (geometric zooming) and displaying information in a way that is dependent on the scale of the objects (semantic zooming) helps to improve the presentation of artefacts and to bridge work transitions (Ware 2004; Memmel & Reiterer 2008). Automatic zooming automatically organizes selected objects on the UI. Animated zooming can be utilized to narrow the work transitions frequently found between artefacts produced in many process phases of UI design. Animated zooming also supports the user in exploring the topology of the information space and in understanding data relationships. Goal-directed zooming techniques are helpful in supporting the iterative nature of the design process, as they enable efficient navigation between widespread artefacts in the specification space (e.g. detailed UI design and a use-case or personas in Figure 98).

Exemplified specification throughput

For switching between models and UI designs, the user can manually zoom in and out and pan the canvas. During user modelling, for example, a user shape can be linked to, and be part of, user roles, personas, and use-cases. Zooming-in on a user shape reveals more details about the underlying personas. The use-case shapes, in turn, can be part of a superordinate task map and can be linked accordingly (see Figure 98). Moreover, zooming in a particular case could link to an essential use-case description and reveal more detail on user and system responsibilities. At this stage, activity and data-flow diagrams help during interaction modelling. The user can link every model to UI designs of different fidelity and vice versa (see Figure 98).

During modelling, or while traversing relationships by panning and zooming, hints about the current zoom factor and the current position in the information space can be given in order to avoid disorientation. In order to support evaluations, INSPECTOR comes with a feedback component that allows annotations to be attached to any object in the UI specification space. They are used to review requirements models, to integrate results of UI evaluation studies or to incorporate notes about trade-offs or design decisions. Annotations are then accessible through a management component that allows a direct zoom navigation to the artefacts concerned.

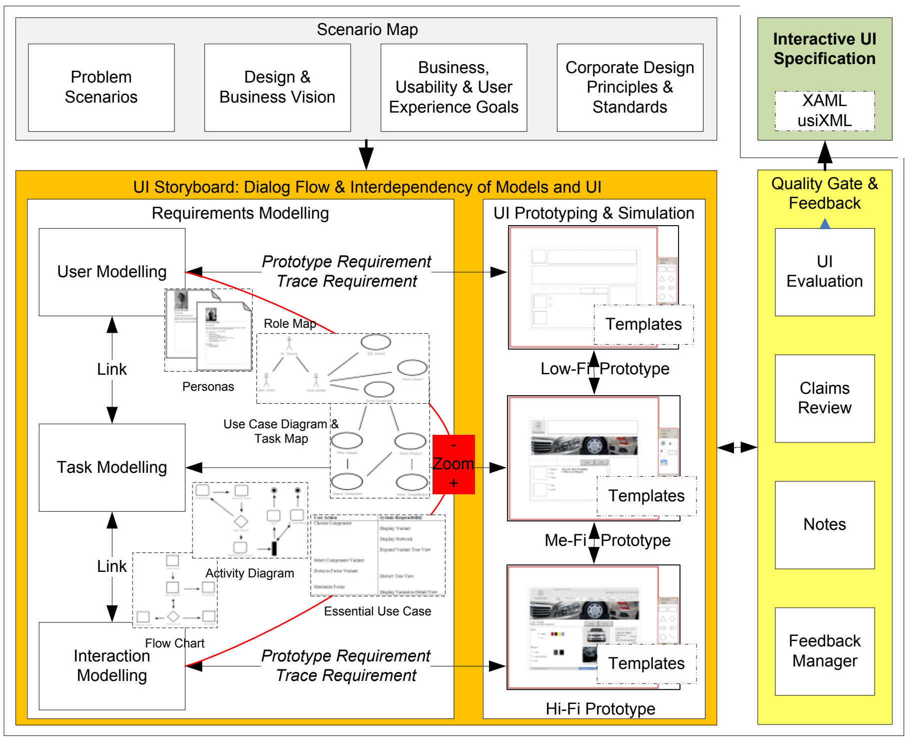


Figure 99: An example of modelling and design throughput with INSPECTOR (Memmel & Reiterer 2008)

UI specification process

In the end, the modelling and design throughput outlines the scheme of a UI specification process (see Figure 99). Naturally, this is just an example, and it is not the only kind of process supported by INSPECTOR and the common denominator.

6.3 Technical Framework

In order to provide some overview of the technical information, this section gives a brief explanation of the main concepts of INSPECTOR's implementation. A more detailed insight into the coding of INSPECTOR can be found in (Geyer 2008). By the end of 2008, the INSPECTOR project had about 15,000 lines of code.

Programming environment and platform

INSPECTOR is based on the .NET Framework and the C# programming language from Microsoft. Accordingly, the Visual Studio Framework 2005 was chosen as the environment for the development of the tool. In addition to advanced programming support based on rich code libraries, Visual Studio 2005 provides a visual designer for UI development. With additional external libraries (DLLs) and components, the basic functionality of Windows Forms was extended throughout the project. INSPECTOR utilizes onboard .NET widgets for the representation of the main window and corresponding basic menu widgets from the 2.0 Windows Forms library. Other components, such as the scalable canvas, toolboxes and document viewers are integrated using external libraries (for example for the integration of OpenOffice or Adobe Flash).

Piccolo: a ZUI framework

For the development of its zoom functionality, INSPECTOR utilizes the .NET Piccolo framework by (Bederson et al. 2004). Piccolo is a high-level API for creating, and interacting with, two-dimensional structures based on the scene-graph model. Previous versions of the main core of the Piccolo framework, which today is still under development with the sponsorship of Microsoft Research (Microsoft Research 2008), have already been successfully applied in the development of Pad++ (Perlin & Fox 1993), developed in C++, and JAZZ (Bederson et al. 2000), developed in Java. Piccolo offers a scalable zoom canvas and helps to organize objects in a scene-graph hierarchy, which fits very well with the requirement to nest artefacts in the UI specification space (see Chapter 6.2). The .NET version of Piccolo is based on the underlying GDI+ layer of the .NET Framework for rendering two-dimensional graphics. Piccolo offers a hierarchical structure of objects and cameras, which allows the developer to create, group, and manipulate objects in a simple manner. Piccolo's infrastructure also provides well-designed rendering and interaction management. To represent requirements objects, canvas classes from Piccolo were extended and customized appropriately. For the design and rendering of graphical objects on the canvas surface, the Enhanced Metafile Format (EMF) is used; this is natively supported by GDI+ and therefore has a good rendering performance. The EMF format supports both vector graphics and bitmaps. It is therefore suitable for all kinds of diagrams and pictures needed to represent abstract models. Other graphical elements such as embedded PDF files, documents or other control elements are already supported by Piccolo (Memmel et al. 2007b).

Model-view-controller software pattern

To deal with the complexity of a UI specification tool and the various views required to model and design a UI, INSPECTOR uses the popular model-view-controller (MVC) pattern for software architectures (see Figure 100). Accordingly, the visual representation of artefacts (the view) is separated from the management of data and information items (the model). The data model maps the hierarchical representation of the zoom-based specification to a data structure of elements. The data model manages all possible kinds of notations and UI objects and provides functions such as insert, delete or search (of elements). Each element of the nested data structure exists as its own class, and the classes determine how the objects are drawn on the zoom canvas at different levels of scale. The controller is used to control and synchronize the associated views (i.e. overview and detail) with their data representation. As illustrated in Figure 101 (5), INSPECTOR has an overview window, which is a common way of supporting the user's cognitive (i.e. spatial) map of an information space. When changes occur, the views are updated immediately. Views are synchronized by employing the Observer Pattern. All views register themselves in a global observing object that provides instant update messages for all views. By using this design it is guaranteed that the two representations remain consistent at any one time (Memmel et al. 2007b).

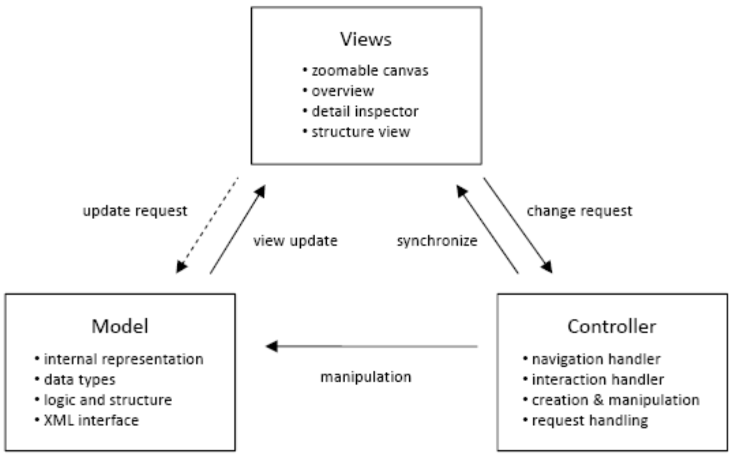


Figure 100: Model-view-controller pattern for INSPECTOR (Geyer 2008)

6.4 Inspector: Interdisciplinary UI Specification Tool

Demonstrating compliance with requirements

In this section, the INSPECTOR tool will be demonstrated and explained in detail by travelling through a case study. With regards to the requirements for adequate UI specification tool support (see Chapter 6.1; Table 69), the focus of the case study is to illustrate the following capabilities and features of INSPECTOR:

- How does INSPECTOR increase traceability and transparency in the UI specification process?
- How does INSPECTOR support switching between abstract and detailed representations of artefacts and designs, thereby supporting transitions between problem and solution space?
- How does INSPECTOR encourage the exploration of design alternatives, a requirement strongly related to versioning and organizing design solutions, as well as encourage collaboration?
- How does INSPECTOR map the common denominator to its concept and metaphors, thereby contributing to bridging the gap between the disciplines and allowing stakeholders to collaborate based on a common understanding?
- How does INSPECTOR make sure that the interactive UI specification developed can be reused by the supplier in implementing the final system?

Car configurator case study

In order to address these issues by using the example of an easy-to-understand case study, the domain of automotive interactive systems was chosen for the example. INSPECTOR will therefore be demonstrated by developing a small UI specification for a car-configurator program, a very important application for most car-making companies and one that usually contributes extensively to the success of automotive digital sales channels. The example of a car configurator has already been used in a variety of publications related to this thesis to discuss the need for new ways of UI specification (Memmel et al. 2007g; Memmel et al. 2008e). It therefore makes sense to present the mature version of INSPECTOR by utilizing the same kind of usage scenario. Accordingly, the case study will be presented from the perspective of ‘Dr. Z’, who is an interaction designer in a very large automotive organization.

Foreword on the presentation of the case study

In addition, it must be noted that INSPECTOR’s appearance and feature-set represent the status quo of the tool’s development at the end of 2008. The subsequent sections therefore already incorporate the revisions of the tool discussed in the analysis of the evaluation studies in Chapter 7. And with regards to the most important requirements for tool support in UI specification, features that are linked to important principles and practices will be particularly highlighted in order to allow the contribution and benefit of INSPECTOR to be traced back to the identified requirements (see Table 69). The fundamental interaction concepts that make up INSPECTOR are explained as soon as modelling or design activities depend on them.

The case study is presented in a linear process from problem-domain modelling straight through to specification-level UI design. Naturally, INSPECTOR has been developed to be used in highly iterative and incremental processes. However, the demonstration of the tool is simplified by a levelled and straightforward presentation of the development of an interactive UI specification following a simple waterfall-throughput of the lifecycle presented in Figure 99. Taking into account the different levels of the common denominator, and to improve readability and understanding, the case study is segregated into four main parts, namely user modelling, task modelling, behaviour modelling and UI prototyping. In accordance with the correlation of models and UI designs shown in Figure 98, storyboards are presented together with the initial stage of problem-domain models because they function as mediators between design and both text-based and diagrammatic notations of requirements.

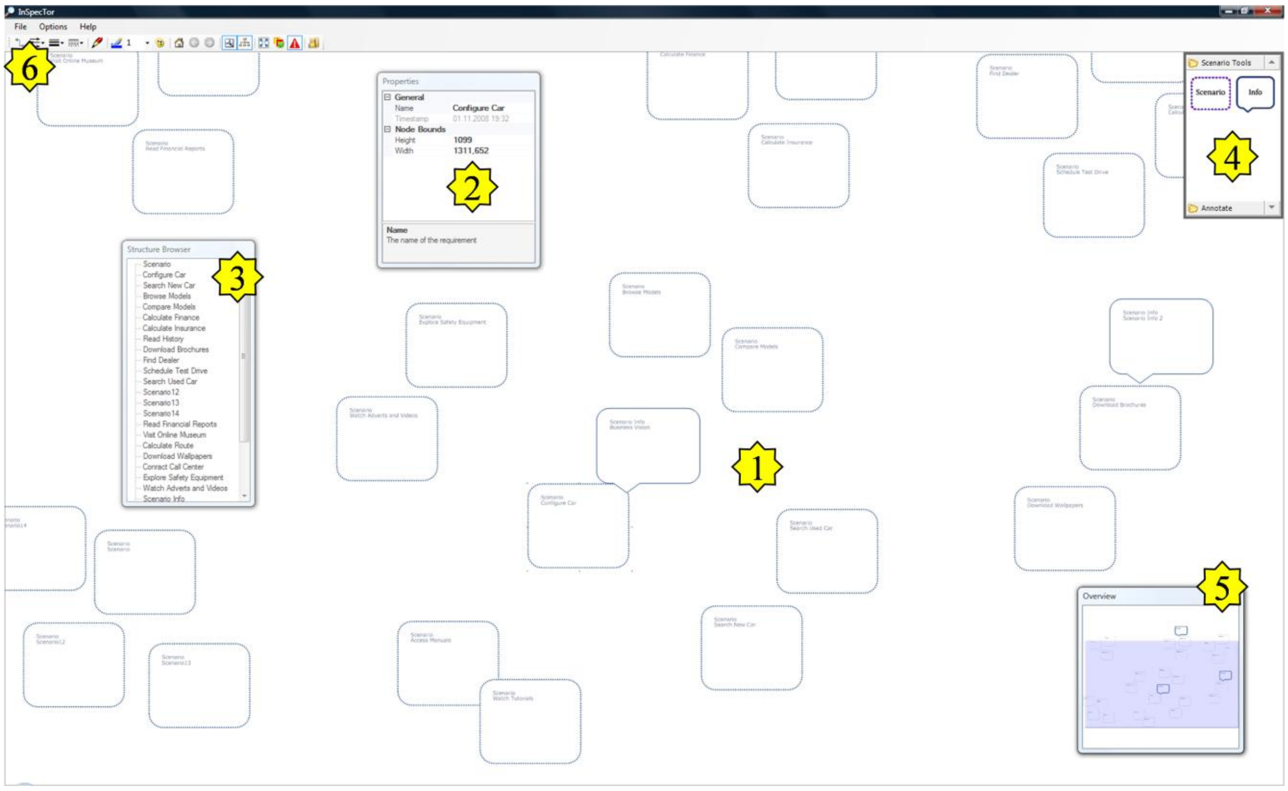
6.4.1 Problem-Domain Modelling And UI Storyboarding

Starting up INSPECTOR

When Dr. Z starts INSPECTOR, the tool displays a wide zoom canvas (see Figure 101, 1) that provides an overview of the artefacts he has created up to this point and arranged in the UI specification space. Let us assume that Dr. Z has opened a previously saved interactive UI specification that already contains a variety of artefacts.

Scenario layer and scenario map

As Dr. Z enters the zoom-world of INSPECTOR on the scenario layer, he initially sees a number of round rectangles, each of them representing a scenario. All scenarios are part of the resulting scenario map (see Chapter 4). With regards to the common denominator, Dr. Z. uses the scenarios to describe the application and problem-domain and for determining specific business goals and design visions. In this case study, Dr. Z has arranged a variety of scenario shapes to refer to the very large number of aspects of interest covered by an automotive website. The shapes were dropped onto the zoom-canvas from the toolbox on the upper-right of the UI (see Figure 101, 4). The internal properties dialogue window (see Figure 101, 2) helps Dr. Z to name and resize the elements. The structure browser (see Figure 101, 3), which can be shown on demand and then arranged freely on the canvas, helps Dr. Z to maintain an overview of all artefacts in the specification space (the principle: e.g. explorability). For this purpose, the structure browser behaves like a typical explorer and is based on a tree visualization that shows the whole hierarchical structure of the specification landscape (the principle: e.g. make it as simple as possible). The orientation on the scenario level and the layers below it is additionally supported with the small overview window (see Figure 101, 5).



Properties
General
Name Configure Car
Timestamp 01.11.2008 19:32
Node Bounds
Height 1099
Width 1311,652
Name
The name of the requirement

Structure Browser
- Scenario
- Configure Car
- Search New Car
- Browse Models
- Compare Models
- Calculate Finance
- Calculate Insurance
- Read History
- Download Brochures
- Find Dealer
- Schedule Test Drive
- Search Used Car
- Scenario12
- Scenario13
- Scenario14
- Read Financial Reports
- Visit Online Museum
- Calculate Route
- Download Wallpapers
- Conract Call Center
- Explore Safety Equipment
- Watch Adverts and Videos
- Scenario Info

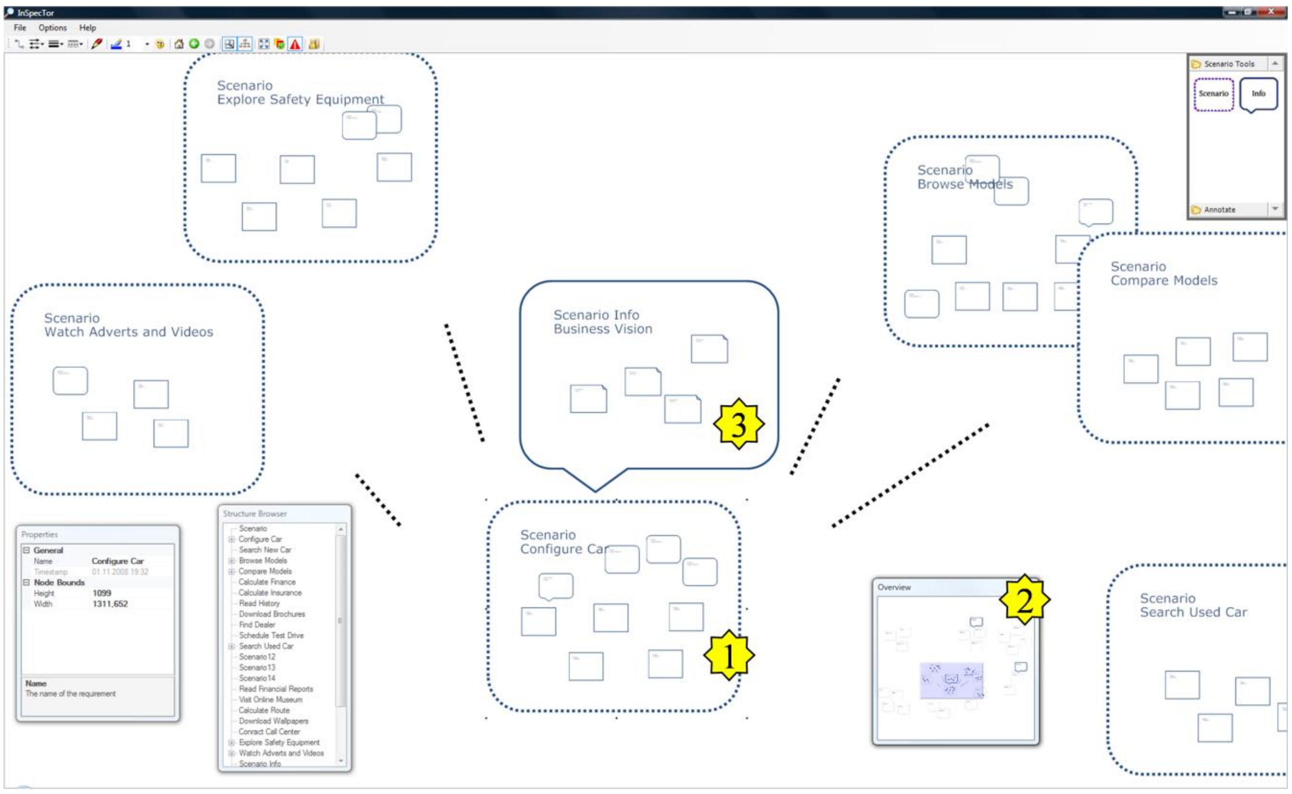
Figure 101: Overview of the scenario map and the initial UI specification level on opening INSPECTOR

Modelling interaction spaces

As outlined in Chapter 4, the scenario map is related to the navigation map in respect of the externalization of relationships between different interaction spaces. With the scenario map, Dr. Z cuts the UI specification space into small and operable pieces that he can interrelate with a variety of common connectors from the toolbar (see Figure 101, 6; see Figure 102). In order to describe the intended design and the functionality envisioned, Dr. Z associates the scenario shapes with information bubbles that contain information describing the purpose and goal(s) of each interaction space, for example the business vision (see Figure 102).

Zooming and panning

Dr. Z can zoom the canvas freely by holding down the 3rd mouse button. The zoom is triggered by either scrolling with the mouse wheel or by double-clicking an artefact on the canvas. If Dr. Z zooms smoothly with the mouse wheel, elements on the canvas decrease or increase in size (geometric zoom) according to the speed at which he turns the wheel. Hence, Dr. Z can use INSPECTOR as if he were working in front of a huge whiteboard, stepping back and forth to switch his degree of interest. At the same time as zooming objects in size, Dr. Z can see additional information on the whiteboard as he gets closer. For example, the semantic zoom reveals the artefacts that are nested into the parent object or placeholder (see e.g. Figure 102, 1), as soon as Dr. Z has zoomed to the predefined threshold that triggers a change in the level of detail that is displayed. The overview window visualizes the current position in the overall UI specification space. Dr. Z can now also pan the canvas by moving the purple rectangle around in the overview (see Figure 102, 2).



Properties

⊟ General	
Name	Configure Car
Timestamp	01.11.2008 19:32
⊟ Node Bounds	
Height	1099
Width	1311,652

Name
The name of the requirement

Structure Browser

- Scenario
- ⊞ Configure Car
- Search New Car
- ⊞ Browse Models
- ⊞ Compare Models
- Calculate Finance
- Calculate Insurance
- Read History
- Download Brochures
- Find Dealer
- Schedule Test Drive
- ⊞ Search Used Car
- Scenario12
- Scenario13
- Scenario14
- Read Financial Reports
- Visit Online Museum
- Calculate Route
- Download Wallpapers
- Conract Call Center
- ⊞ Explore Safety Equipment
- ⊞ Watch Adverts and Videos
- Scenario Info

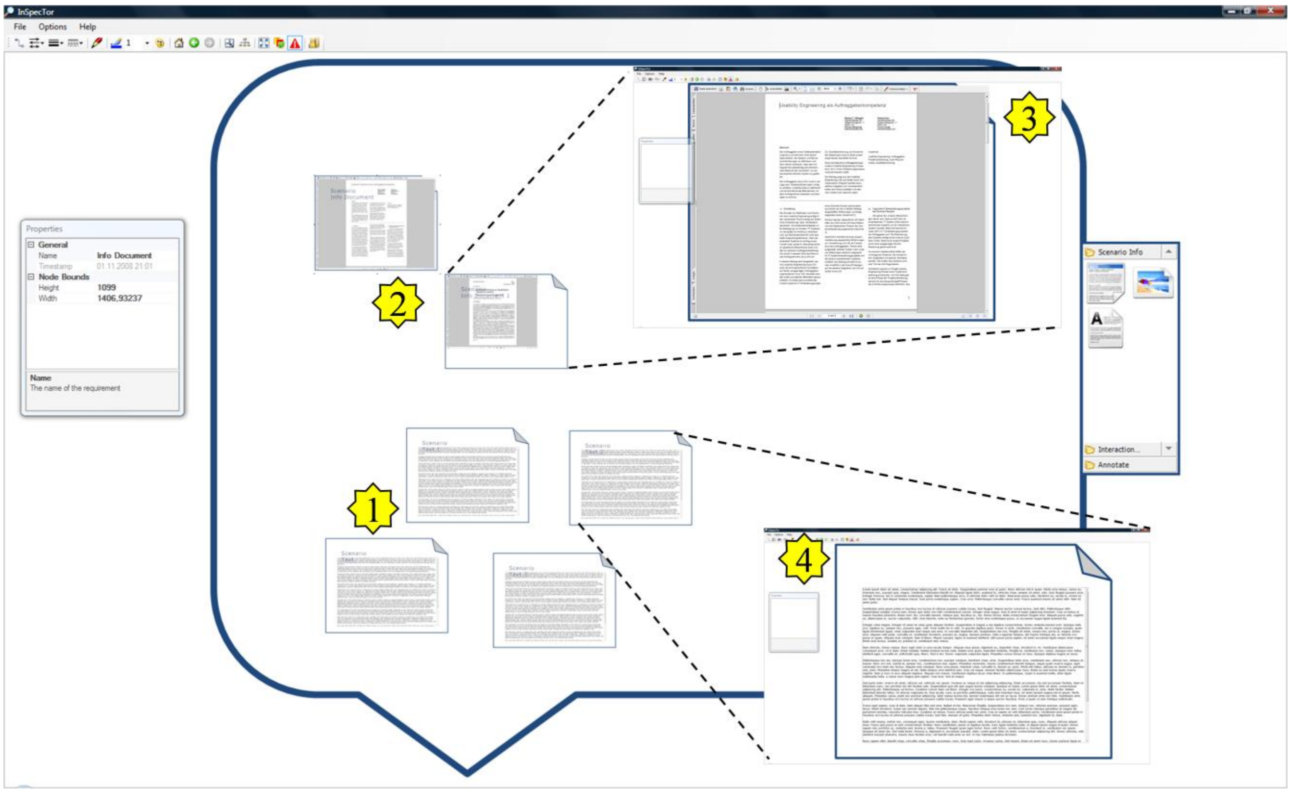
Figure 102: Magnified view of a sub-part of the scenario map; semantic zoom revealing nested artefacts and associations between scenarios

Writing narratives and embedding documents

As an alternative to the mouse wheel, Dr. Z can use the 1st mouse button to zoom into selected artefacts on the canvas. A double click triggers a goal-directed, automatic zoom that centres the focused object on the canvas and magnifies it to the size that enables work on its content. Today, Dr. Z wants to model the car configurator. Dr. Z therefore wants to substantiate his intention by adding existing corporate documents and statements regarding business goals to the information bubbles related to the car-configurator scenarios (see Figure 102, 3).

Problem-scenarios and mission statements

The information bubble is the carrier for typical textual narratives on the problem-domain. Accordingly, Dr. Z and some of his colleagues write several problem scenarios narrating the challenges and previous experience of the organization (i.e. who later is the client) in developing a car configurator. For this purpose, INSPECTOR provides a built-in text-editor (see Figure 103, 1). Dr. Z will also incorporate the scenario texts written by stakeholders and end-users to make the package complete. Because they are familiar with the means of expression (i.e. text), they are invited to participate in the specification process (the principle: e.g. active stakeholder participation, model with others, low threshold). As an interaction designer, Dr. Z may also describe end-users, their tasks and some corporate mission statements in the scenarios he composes, in order to draw a more detailed picture of the domain of the car configurator and the specification project he faces. To add existing resources, INSPECTOR allows the embedding of all kinds of common document formats, such as Adobe PDF, Microsoft Word or Excel, and a variety of image types (the principle: e.g. iterate to another artefact). Embedded documents are directly presented just where Dr. Z dropped them (see Figure 103, 2). A zoom-in to one of the artefacts contained in the information bubble will magnify the respective object, allowing Dr. Z to read or edit it (see Figure 103, 3+4).



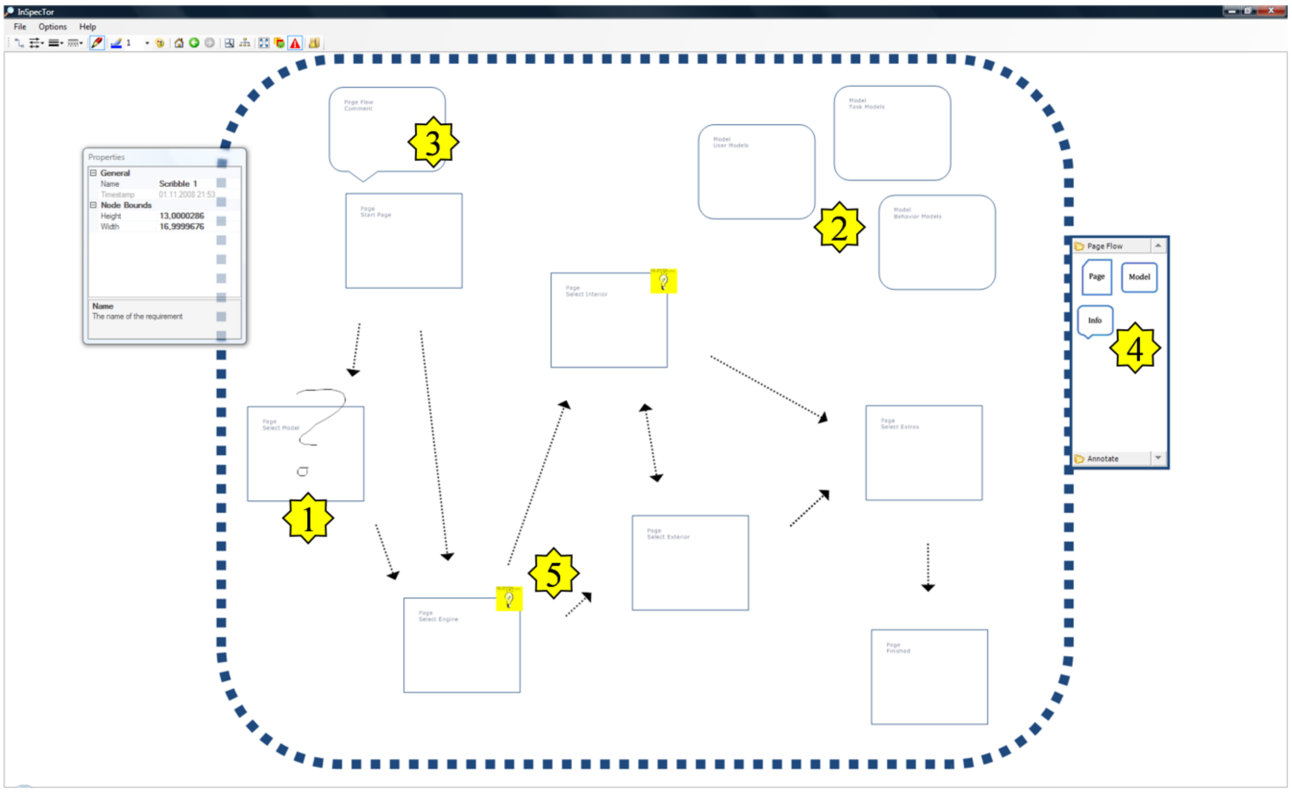
General	
Name	Info Document
Timestamp	01.11.2008 21:01
Node Bounds	
Height	1099
Width	1406,93237

Name
The name of the requirement

Figure 103: Integrating textual artefacts to specify business goals, design vision and narratives

Storyboarding

After Dr. Z has finished shaping the problem-domain in the necessary detail, he wants to draw a first, approximate frame for the UI that has to be developed. In order to assemble the different screens that will probably make up the car configurator with the underlying requirements, Dr. Z switches to the storyboard layer (the principle: e.g. multiple models, perspective). There, he will drill-down into the detail of the scenario shapes, which actually functions as a placeholder for the storyboard. The storyboard layer combines UI states and placeholders for modelling in one mediating view (the principle: e.g. formality and detail, perspective). The UI states are represented by rectangles that will later contain UI designs with different levels of fidelity (see Figure 104, 1). User, task and behaviour models, which will specify the requirements for the sketched storyboard, will be assembled in the placeholders represented by rounded rectangles (see Figure 104, 2). As on most levels, information bubbles can now be used to add additional information relevant to either the design or the requirements-modelling stage (see Figure 104, 3). The shapes that represent all three kinds of artefacts can again be inserted by a simple drag'n'drop operation from the toolbox (see Figure 104, 4). Because Dr. Z is still unsure about the content of some UI states, i.e. web pages, that he added as a first guess, he drops some annotations on the canvas that will inform him or a different stakeholder about his sketchy design thoughts (see Figure 104, 5) (the principle: e.g. support collaboration). The small annotations can also be magnified on demand. Dr. Z also uses the electronic pen to hand-draw a question mark on one of the pages. Naturally, the different UI states can be connected with different types of lines or arrows. From an analysis of the scenarios and of some existing corporate documents on car configurators from both his company and some competitors, Dr. Z already has an approximate idea of the most important pages that will make up the configuration process.



Properties

⊟ General	
Name	Scribble 1
Timestamp	01.11.2008 21:53
⊟ Node Bounds	
Height	13,0000286
Width	16,9999676

Name
The name of the requirement

Figure 104: A storyboard layer that combines UI states and placeholders for modelling in one view

6.4.2 User Modelling

Using INSPECTOR during user analysis

Very soon Dr. Z recognizes that it is too difficult to develop a detailed storyboard without having a concrete idea about users and their tasks. He starts an extensive requirements analysis and meets with potential end-users as well as domain experts in interviews and focus groups. While some of his colleagues moderate the workshops and ask the participants some structured question, Dr. Z uses INSPECTOR to capture the most important findings on the fly (the principle: e.g. low threshold). Having zoomed into the placeholder for user modelling on the storyboard level, he uses the built-in text editor and an information bubble to safekeep and structure the user needs thus found (see Figure 105, 2+3).

Modelling personas

Later, Dr. Z starts to develop personas from the collected information. As explained in Chapter 4, several characters recognized in the user analysis can be merged to one persona, which then acts as a proxy for a group of users. The persona therefore provides a means to talk and reason about the group through the characteristics of one fictional individual, the persona. For personas modelling, INSPECTOR provides a predefined template (see Figure 105, 5) that guides Dr. Z in structuring the personas as proposed, for example, by (Cooper 1999; Cooper 2004). In this way, Dr. Z models several personas on the canvas (see Figure 105, 1).

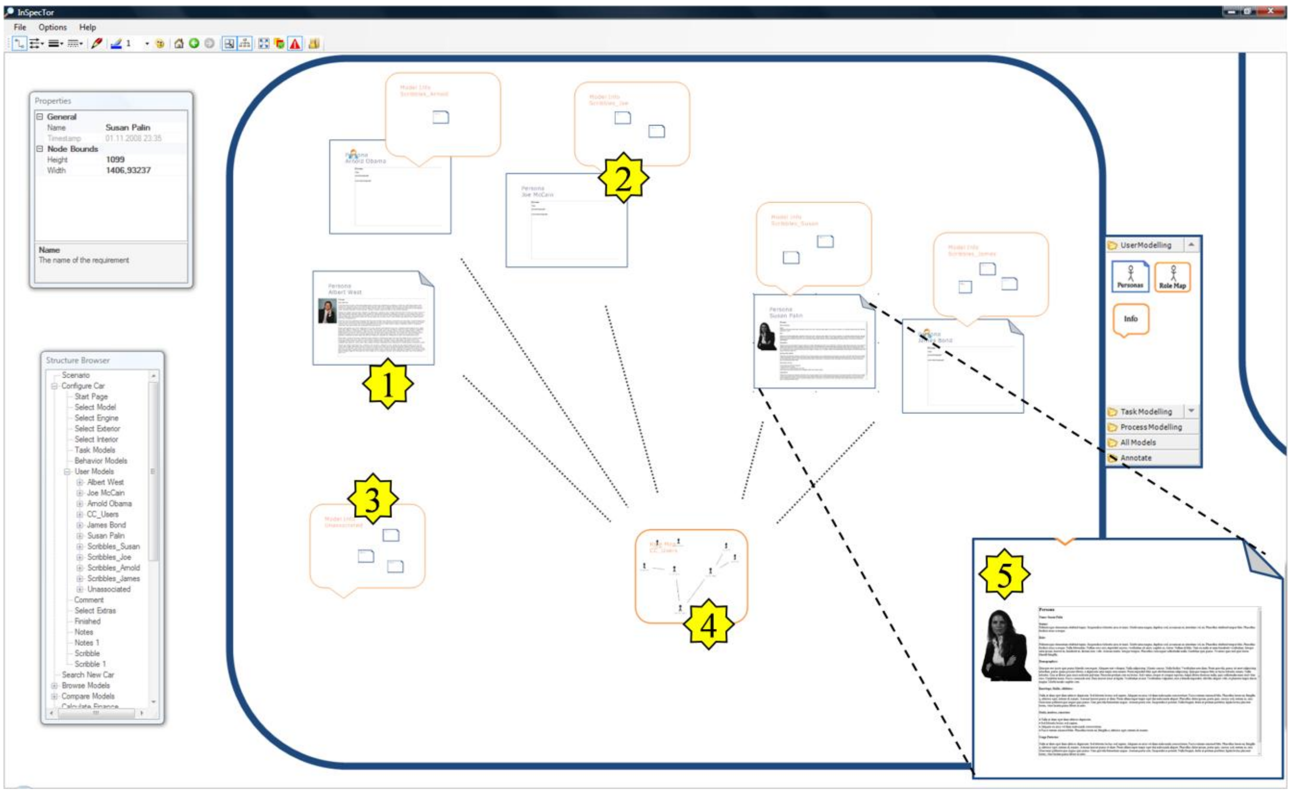


Figure 105: A placeholder for user modelling with personas and user- role map

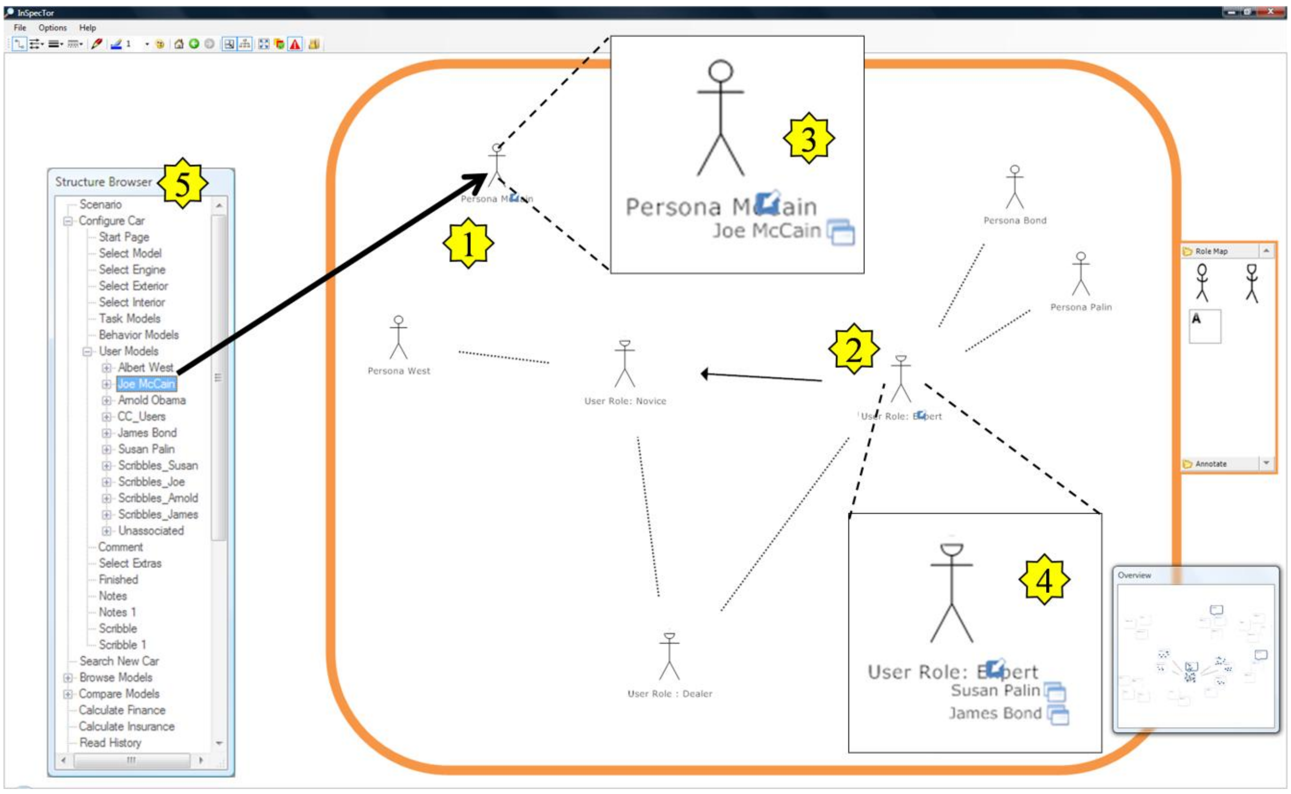


Figure 106: Modelling a user- role map with links from role to personas (names fictitious)

Relating personas in role maps

During personas modelling, Dr. Z realizes that the requirements analysis shed light on a variety of different users, as is typical for complex projects. He therefore decides to combine the personas with a role map (see Figure 105, 4) to outline relationships and dependencies between the types of users he will have to take into account (the principle: e.g. multiple models). Before Dr. Z models the role map, he draws associating lines from the personas onto the role-map shape to visualize the relationship (the principle: e.g. traceability).

He then zooms into the placeholder for the role-map model. Eventually, Dr. Z begins to gain a valuable overview of the types of users and their relationships. He interrelates the various personas (see Figure 106, 1) or proxy-roles (see Figure 106, 2) with different connectors. In this context, both shapes can be linked to one or more associated models. The connection is visualized with a link symbol and an optional pop-up list of connections to other artefacts (see Figure 106, 3+4). Dr. Z is able to develop these links by dropping the labels of target elements from the structure browser (see Figure 106, 5) onto the source element, for example a user-role shape. The target element, in turn, receives an outgoing link to the associated role shape (the principle: e.g. traceability; support many paths; iterate to another artefact).

6.4.3 Task Modelling

Discussing modelled artefacts

After user modelling, Dr. Z meets with stakeholders and reviews the personas he has just created (the principle: e.g. model with others, support collaboration). Some of his colleagues are unsure whether the personas really match the findings of the focus groups and interviews. Dr. Z therefore needs to trace the models back to the original scribbles, which luckily are safekept in the UI specification space (the principle: e.g. guidance, traceability). Because he is in a scheduled team meeting and time is short, he quickly navigates to the respective artefacts (i.e. information bubbles, see Figure 105, 2) by selecting and double-clicking the name of the item in the structure browser (see Figure 106, 5) in order to present the recorded data. INSPECTOR then initiates an automatic goal-directed jump zoom and navigates the user through the UI specification space, directly targeting towards the selected artefacts. In this way, the user is able to learn the path to the artefact in case he wants to navigate to it manually at a later stage.

Relating personas and use-cases

Dr. Z decides to relate the identified personas and roles to the use-cases that were identified during the requirements analysis and from reading the scenarios. He therefore uses INSPECTOR to develop some use-case diagrams (see Figure 107, 1). In addition, Dr. Z and other stakeholders from the team, in particular some product managers, have some new and interesting ideas for designing a highly innovative car configurator. At this stage, however, Dr. Z cannot assign these use-cases to any of the personas, but needs to structure them in some way to gain an overview. For this purpose, INSPECTOR offers the possibility of modelling task maps (see Figure 107, 2), as proposed in the common denominator (the principle: multiple models, model with purpose). Dr. Z also prepares some placeholders for writing down some essential use-cases (see Figure 107, 3 + 4), which he will use to describe the detail of a single case (i.e. task). He has already associated some of the essential use-cases with the previously created use-case diagrams and the task map (the principle: e.g. traceability). As with the other levels of development in the interactive UI specification, all artefacts are available from the toolbox on the right side of the modelling placeholder (see Figure 107, 5). In fact, the toolbox contains all notations applicable to the common denominator and Dr. Z can therefore use and mix them freely.

Developing use-case diagrams

Dr. Z starts to develop a use-case diagram by arranging the shapes concerned on the zoom-canvas. Again, he connects the representatives to the underlying detailed artefacts. For example, Dr. Z again links a personas model with the role shapes he dropped on the canvas (see Figure 108, 1). The shape consequently illustrates the modelled connection with an icon (see Figure 108, 2). Dr. Z also creates connections between the case shapes and the associated essential use-cases, which need to be accessible for obtaining the details of user intentions and system behaviour (see Figure 108, 4+6). Because the user roles or the personas involved in the use-case diagram are also tightly coupled to the requirements addressed in the essential use-case, Dr.

Z also creates links between them (see Figure 108, 4+5). Dr. Z creates similar links between the case shapes and essential use-cases within the task map, which otherwise employs similar shapes (see Chapter 4).

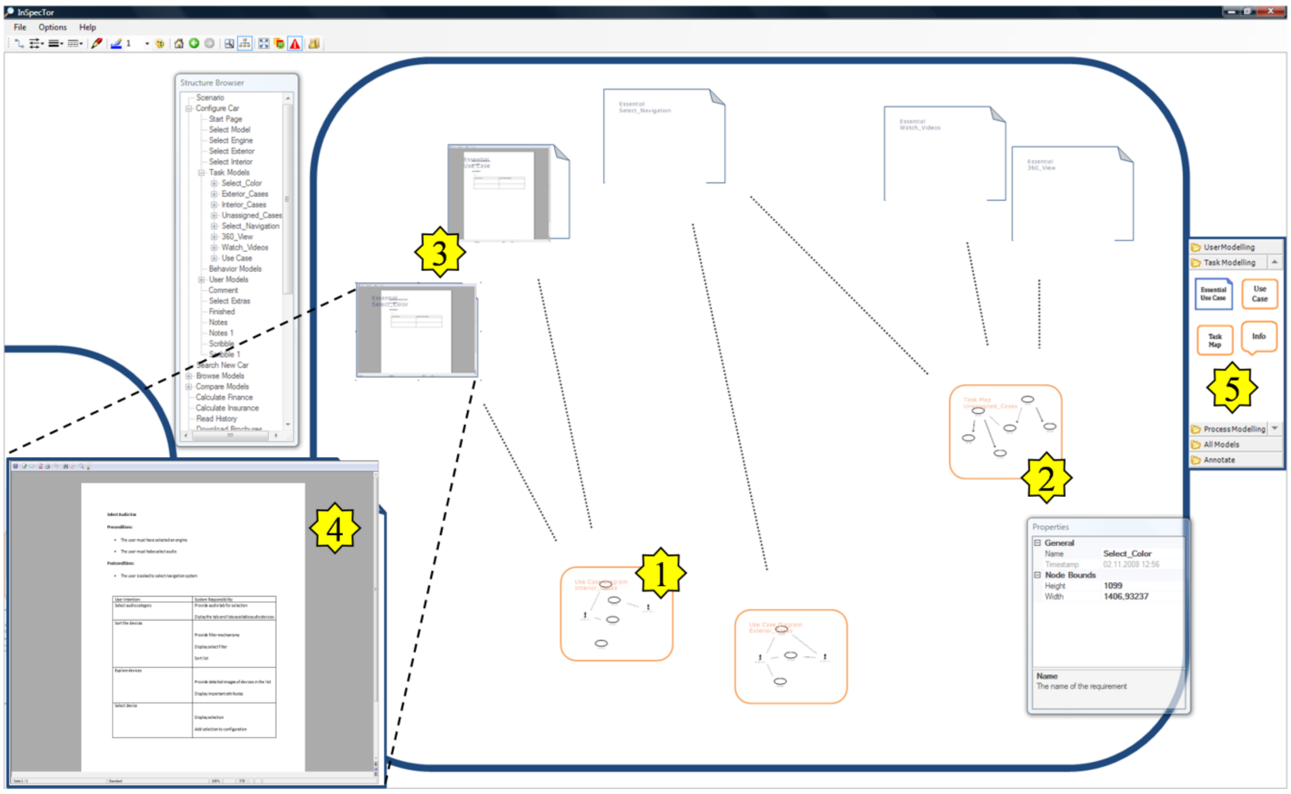


Figure 107: Modelling use-cases, essential use-cases and task maps with INSPECTOR

The context layer

To make sure that the link he created points to the right personas model, Dr. Z clicks in the preview icon just below the link in order to open the context layer (see Figure 108, 3). The context layer provides a preview of artefacts that have been linked to the source shape (the principle: e.g. explorability, guidance).

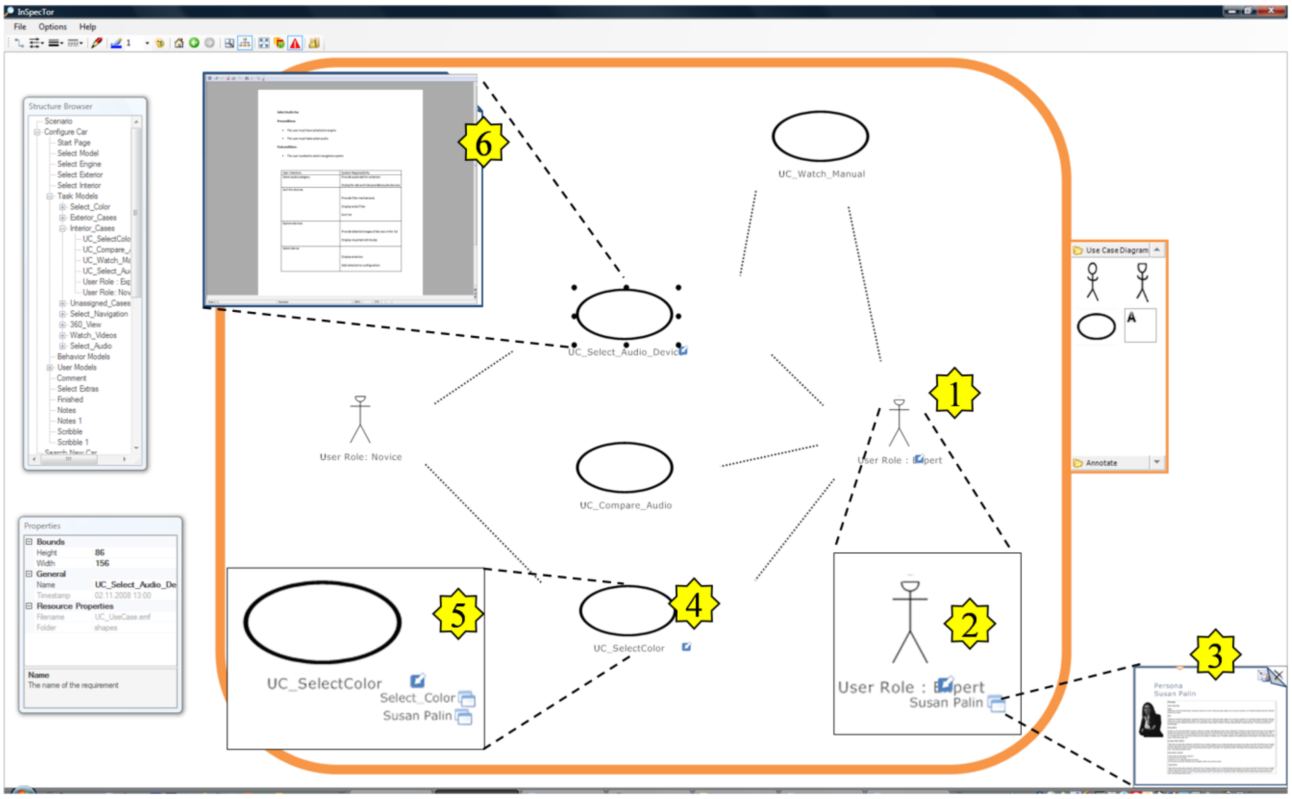
Due to the zoom paradigm of INSPECTOR, frequent navigation between artefacts can be an onerous task, especially if objects are widespread i.e. some distance apart along the three dimensions (x, y and z-axes) of the ZUI canvas. In order to be able to crosscheck two artefacts, for example, the context layer functions as a navigation aid and reduces the number of required zoom operations (the principle: e.g. guidance). This feature can be compared to the situation in a design room in which a person temporarily removes an artefact from the wall of wonders and walks towards another artefact to which it is related or connected (the principle: e.g. support many paths and many styles, support exploration). When the context layer is opened, it is shown in a large and freely relocatable overlay-window on the canvas. Because Dr. Z does not need the preview of the related artefact permanently, he minimizes it (see Figure 108, 3). The artefact visualized in the context layer cannot be manipulated. For this purpose, Dr. Z would have to activate the link and make INSPECTOR zoom to the connected personas model.

Emerging network of artefacts

After Dr. Z has finished these steps, several artefacts are already extensively interconnected in the UI specification space. For example, the personas model that was referenced twice from a variety of other artefacts is already traceable from a use-case diagram, an essential use-case and a user-role map, and vice versa (the principle: high ceiling, model with purpose). In this way, Dr. Z can work with different kinds of models, every single one having its own strengths in expressing certain aspects of the UI (the principle: e.g. traceability, multiple models, apply the right artefacts).

Linking task models to design

During task modelling, Dr. Z already begins to develop in his mind a conceptual model of the eventual UI. He realizes that he needs to safekeep his ideas in order not to forget his design visions.

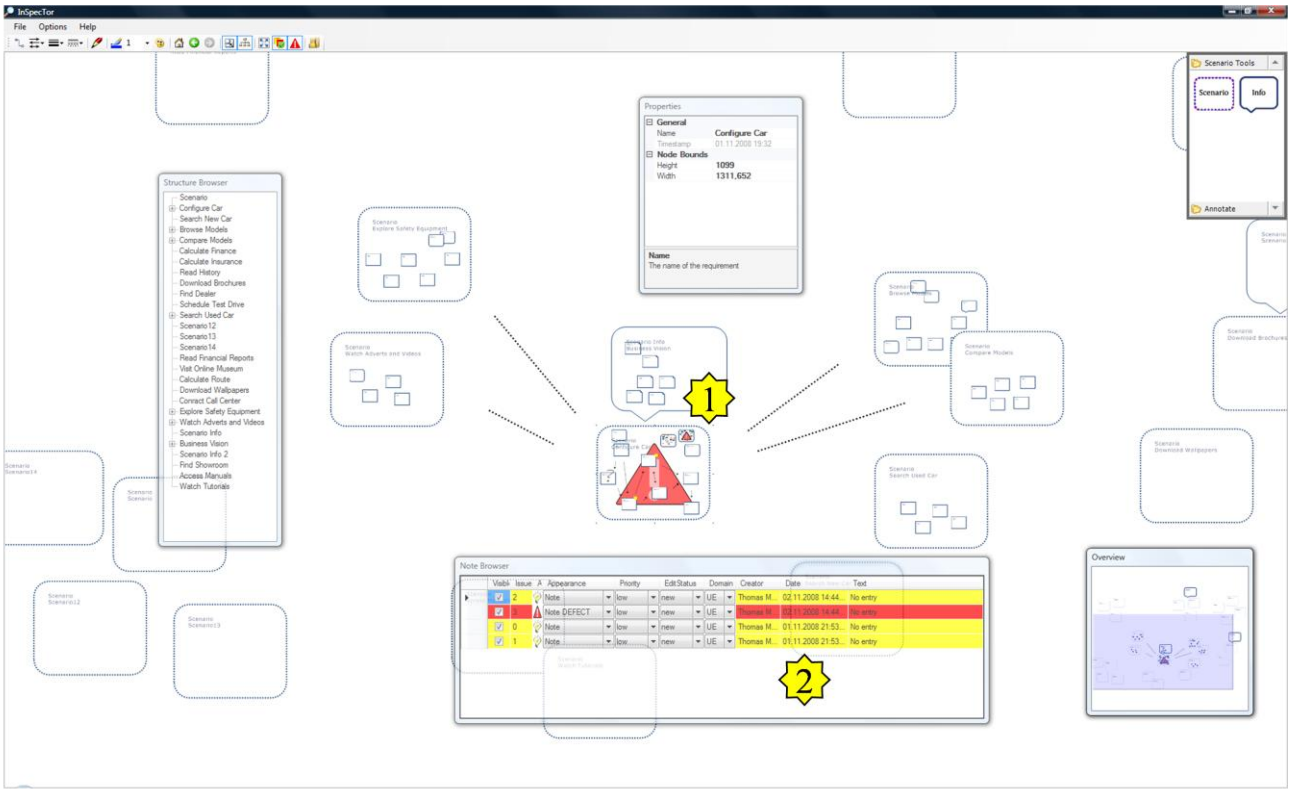


Properties

Bounds	
Height	86
Width	156
General	
Name	UC_Select_Audio_De
Timestamp	02.11.2008 13:00
Resource Properties	
Filename	UC_UseCase.emf
Folder	shapes

Name
The name of the requirement

Figure 108: Developing use-case diagrams with links to essential use-cases and personas



Properties

General	
Name	Configure Car
Timestamp	01.11.2008 19:32
Node Bounds	
Height	1099
Width	1311,652

Name
The name of the requirement

Note Browser

Visibl	Issue	A	Appearance	Priority	Edit Status	Domain	Creator	Date	Text
✓	2		Note	low	new	UE	Thomas M...	02.11.2008 14:44...	No entry
✓	3		Note DEFECT	low	new	UE	Thomas M...	02.11.2008 14:44...	No entry
✓	0		Note	low	new	UE	Thomas M...	01.11.2008 21:53...	No entry
✓	1		Note	low	new	UE	Thomas M...	01.11.2008 21:53...	No entry

Figure 109: INSPECTOR visualizing comments and the corresponding annotation-management console

At several stages during task modelling, and during other modelling stages as well, Dr. Z wants to sketch or prototype his design thoughts (the principle: e.g. apply the right artefacts, perspective, formality and detail). While INSPECTOR's support for UI design is discussed in Chapter 6.4.5, it must be stated here that all kinds of artefacts can, of course, also be connected to UI designs, or even to single elements of a developed UI.

Managing feedback

After Dr. Z has modelled the most important task diagrams and essential use-cases, he moderates several workshops to review his results. Moreover, he saves to disk the interactive UI specification space developed so far and forwards it as an archive (including the embedded documents, images, etc.) to other stakeholders, whom he invites to model on their own any user tasks they have in mind (the principle: e.g. support open interchange, support collaboration, model with others). Some days later, Dr. Z analyzes the feedback using the annotation manager (see Figure 109, 2), which displays, for example, simple comments (yellow) and critical defects of the specification (red). Besides presenting these in the management console, INSPECTOR also visualizes defects within the shapes on the canvas (see Figure 109, 2). The annotation manager can be used to change the content or status of the annotations. Moreover, Dr. Z can jump-zoom directly to the critical area or the annotated artefacts in order to review the feedback and to implement changes (the principle: e.g. maximize stakeholder ROI, embrace change). If Dr. Z zooms into the specification space manually, the management console filters the displayed annotations with respect to the level on which they occur. This means that the annotation manager will only display entries that are available on the current layer of the specification landscape or on levels deeper in the hierarchy. Conversely, comments made on superior levels reappear as soon as Dr. Z zooms out.

6.4.4 Behaviour Modelling

Modelling UI behaviour

After Dr. Z has finished user and task modelling, he is confident that he can plan the UI flow in detail. Although he has already developed an approximate UI storyboard, the different screens of the car configurator might change due to relationships to external entities. From modelling the use-cases, Dr. Z knows that the car configurator relies heavily on huge databases that provide information about the car's equipment or pricing, for example. Moreover, these databases identify conflicts in the configuration of the car. The car configurator must display warnings and options to the user if combinations assembled by the user cannot be manufactured.

Developing data-flow diagrams

In accordance with the common denominator, INSPECTOR provides the shapes required to model agile data-flow diagrams (the principle: e.g. multiple models) in the notation proposed by (Ambler 2006c). In addition to shapes designating data stores (see Figure 110, 1), Dr. Z adds a shape to represent the source of the interaction sequence, which is the expert user in this example (see Figure 110, 4). Both kinds of shapes are connected to the actual steps in this data-flow diagram (see Figure 110, for example 2+4). Again, all the shapes involved can be connected to supporting information such as (essential) use-cases, a user-role map or a sketchy vision of the UI design (the principle: e.g. traceability, explorability). In this example, Dr. Z wants to outline the dependencies of the 'select audio device' use-case from databases that provide detailed information about the devices available and their allowable combinations with other additional equipment.

Modelling the complexity of UI flow

During later stages of the UI specification process, Dr. Z realizes that the simple UI storyboard he modelled at the beginning is far from being sufficiently detailed to handle all the possible ways to interact with the car configurator and to travel through the configuration process (the principle: e.g. expressiveness). In many UI states, the user can make choices that then determine the subsequent steps in the dialogue flow. In other cases, some UI states must be travelled through by all users, disregarding previous decisions in the configuration process. The flow of steps depends strongly on (1) the kind of user that interacts with the application and, consequently, (2) the variety of possible use-cases.

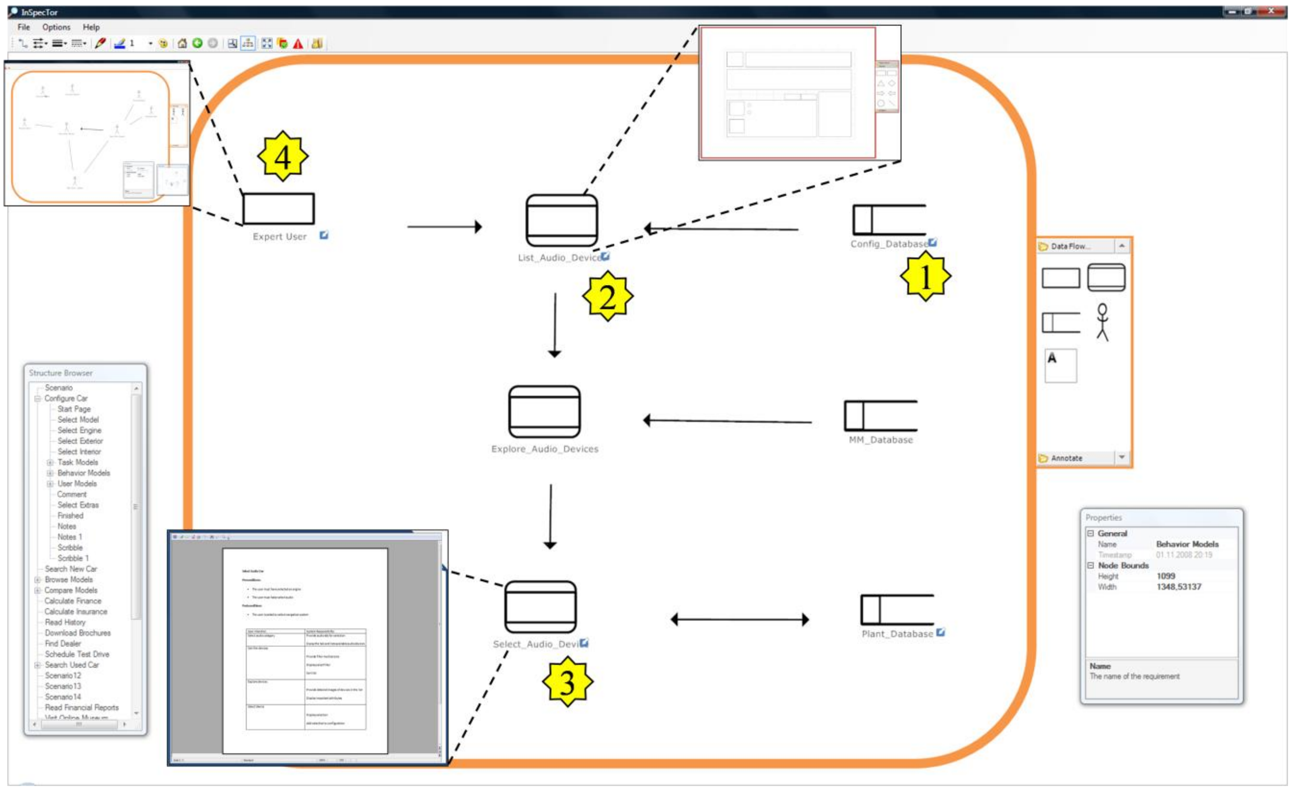


Figure 110: Developing a data-flow diagram with links to essential use-cases, role maps and design

Developing activity diagrams

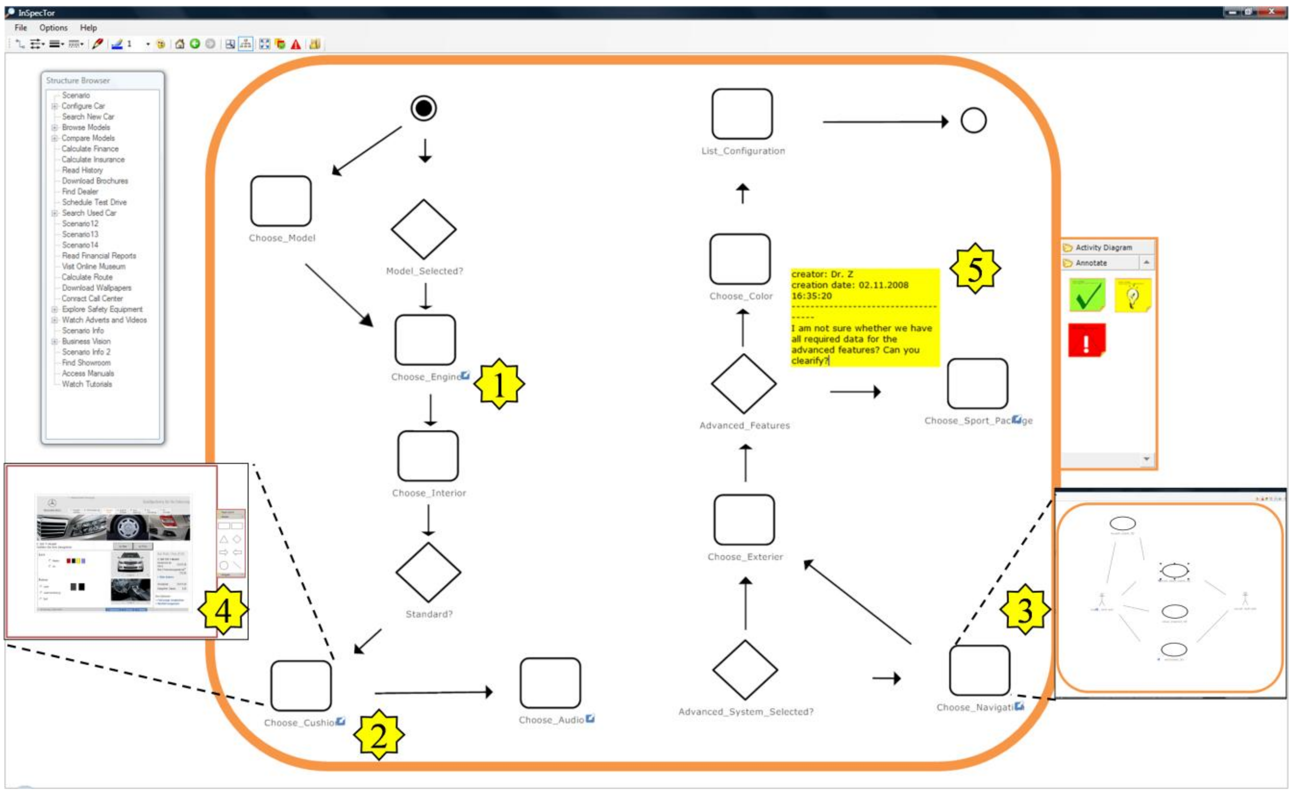
At the end of the overall modelling activities, Dr. Z therefore decides to visualize the complexity of the whole configuration process with a huge activity diagram. The activity diagram will also help Dr. Z to discuss his model with technicians who are likely to provide him with valuable feedback on his specified UI flow (the principle: e.g. formality and detail, perspective). Afterwards, Dr. Z updates his UI storyboard in some areas to make it match the discoveries resulting from the activity diagram.

Linking the activity diagrams to a mature network of artefacts

The activity diagram, which is based on the notation of (Ambler 2006j), is usually extensively interconnected (see Figure 111, 1-3) to other artefacts because most other models, such as use-case diagrams (see Figure 111, 3), already exist and refer to the activity diagram (and vice versa). Dr. Z links his activity diagram to detailed UI designs (see Figure 111, 2+4) that he has already created (see Chapter 6.4.5). This allows the context layer to be used for switching between the abstract diagram and the detail screens on demand. In this respect, Dr. Z relies on the context layer as a mechanism that supports him in making the right arrangement of dialogues in the activity diagram (the principle: e.g. design for designers). Dr. Z also leaves a small comment in the diagram in order to inform other stakeholders about the latest uncertainties or other issues he still has with this model (see Figure 111, 5).

Travelling through the UI specification space

With regards to the example of modelling and design throughput shown in Figure 98, the stage of behaviour modelling will usually mark the upper end of the UI specification process that Dr. Z has travelled along. Through an extensive network of links between the artefacts he has created, Dr. Z is able to explore the interactive UI specification from any point on the zoom landscape (the principle: e.g. traceability, support exploration). Before a supplier is assigned to implement the UI specified by Dr. Z and the contributing stakeholders, the team will conduct several meetings to consolidate the modelled interactive application. The stakeholders that participate in the process will use the annotation tools to mark models or designs that need to be revised (the principle: e.g. embrace change). Other stakeholders might decide to change some artefacts themselves and just indicate the update by adding a comment (the principle: e.g. model with others, active stakeholder participation). Due to the network linking the artefacts, stakeholders are able to trace their changes to related artefacts that might also be affected by their changes. Finally, Dr. Z will use the interactive UI specification to hold a usability study with some of the users interviewed in the user analysis (the principle: maximize stakeholder ROI).



creator: Dr. Z
creation date: 02.11.2008
16:35:20

I am not sure whether we have all required data for the advanced features? Can you clearify?

Figure 111: Modelling an activity diagram with e.g. links to designs and annotations

6.4.5 Sketching and Specifying UI Design

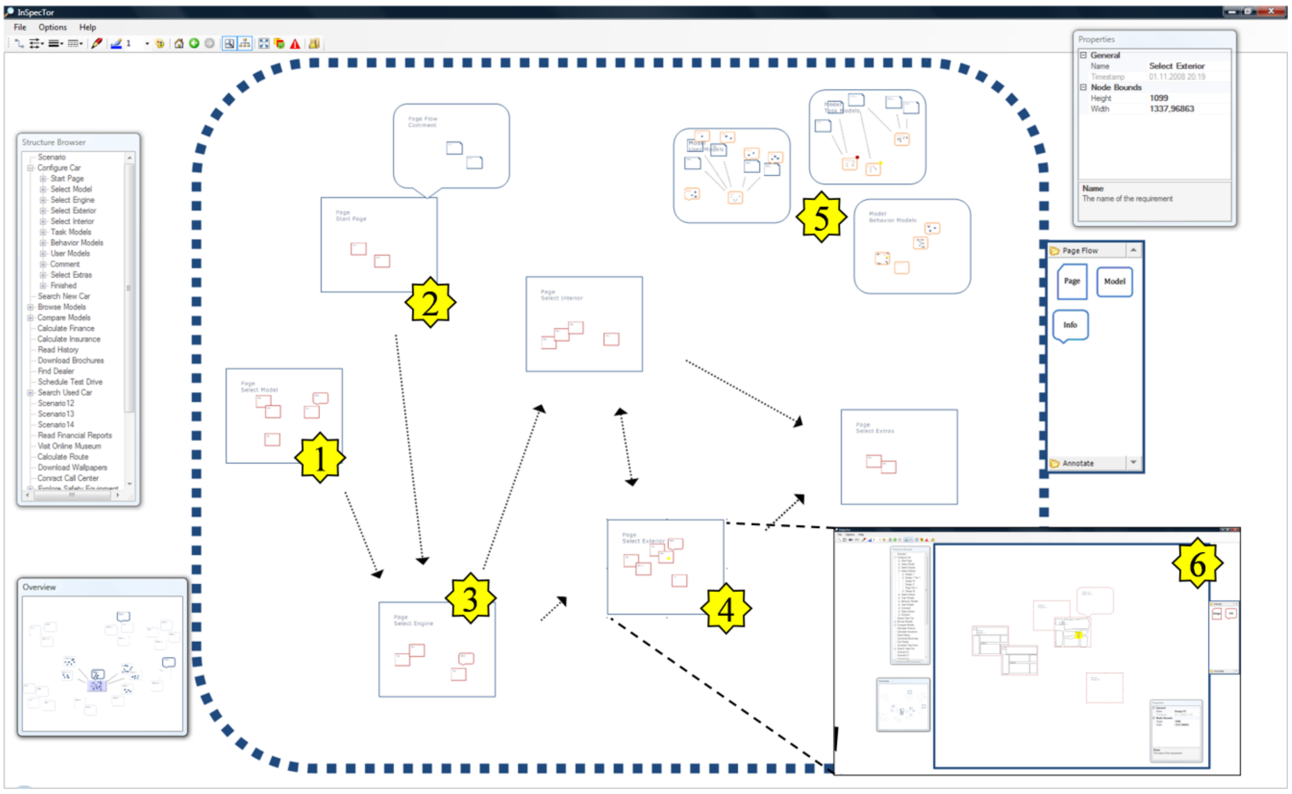
Developing UI design with INSPECTOR

At several points in time, Dr. Z wants to externalize his design visions (the principle: e.g. iterate to another artefact). INSPECTOR supports a range of levels of UI design fidelity (the principle: e.g. explorability). The UI design layer is available from the UI storyboard level. The design shapes are represented as sharp-cornered rectangles (see Figure 112, 1-4) to distinguish them from the placeholders for requirements modelling (see Figure 112, 5). For every UI state, Dr. Z has to develop different kinds of UI designs in respect of functionality, interactivity, resolution or scope, for example (see Chapter 4, Table 61).

Designing the UI and managing design versions and alternatives

The shapes on the UI storyboard therefore do not directly contain drawings or UI elements, but are instead placeholders for safekeeping different versions or UI designs (see Figure 112, 6). In this way, trade-offs, decisions and related comments can be stored in the UI specification space for later reference and to provide information for subsequent steps (or projects) as well as for people not involved in the design process (the principle: e.g. guidance). Due to the infinite space available on the zoom-canvas, Dr. Z and his team can externalize all their design ideas and employ INSPECTOR as a computer-supported extension of their minds (the principle: e.g. explorability, support exploration, wide walls, design for designers). INSPECTOR supports the development of multiple versions, i.e. alternatives of a design, with an easy-to-use versioning feature. For example, the versioning system enables Dr. Z to create new designs on the basis of an existing sketch or prototype (the principle: e.g. embrace change). This saves a lot of time, especially during participatory design sessions or workshops. Moreover, INSPECTOR provides the opportunity to group several UI elements or whole layouts and to save them as a reusable template. The template can then be dragged on the design canvas, resulting in a reallocation of the embedded UI elements. Thus, INSPECTOR eases and speeds up

UI design activities with some simple, but very effective features (the principle. e.g. explorability, expressiveness). Naturally, Dr. Z can at any time mark the design version that effectively represents the status quo and that is therefore part of the UI simulation.



Properties

General	
Name	Select Exterior
Timestamp	01.11.2008 20:19
Node Bounds	
Height	1099
Width	1337,96863

Name
The name of the requirement

Figure 112: Accessing UI designs from the UI storyboard layer

The UI simulation

The UI simulation, which is actually the spokesman of the interactive UI specification, is created by linking the different UI states. Dr. Z can create such links by dropping the name of a UI state from the structure browser onto a UI element. In this way, all UI states relevant to running through a scenario can be arranged in a way that best supports the interaction with the system. Because Dr. Z. is still unsure about the optimal combination of design solutions, he can easily try different alternatives, for example by designating different versions of a design to be part of the simulation (the principle: explorability; iterate, iterate). Due to the nature of the interactive specification, UI designs and UI elements are also linked to the underlying requirements (the principle: e.g. model with purpose). This means that at any time, Dr. Z can switch between the abstract and the detail, as well as between different representations of the design (the principle: e.g. formality and detail).

UI design with different fidelity and scope

As outlined in Chapter 4, Dr. Z will usually develop a great variety of different kinds of prototypes in the UI specification process. During earlier stages of the process, Dr. Z wants to sketch designs, for example with an electronic pen and a suitable input device (see Chapter 7+8), in order to quickly document design visions or to capture ideas gathered in sessions with end-users and other stakeholders (see Figure 113). In this context, INSPECTOR has been developed to provide an important feature for fostering creative processes. The sketching mechanism implemented has no constraints and can be employed at any time and in every level of the interactive UI specification (the principle: e.g. low threshold). This means that even user, task or behaviour models could be partly developed using the sketching mechanism.

Elements of the UI design can be linked to other UI states (see Figure 113, 1; Figure 114, 1), this being indicated by a green arrow. UI designs can also be connected to underlying or related requirements (see Figure 113, 2) in order to develop the necessary correlation of designs and models.

Specification-level UI design

Having arrived at the later stages of the UI specification process, Dr. Z is able to design the UI on a mature, precise and detailed level that is able to guide programmers in the supplier's organization in implementing the UI in accordance with the requirements of the client organization. For developing a sophisticated design, Dr. Z can make use of a variety of predefined abstract and concrete widgets (see Figure 114, 2). In addition, it is possible to include ActiveX components and AdobeFlash clips to specify particularly interactive behaviour in detail.

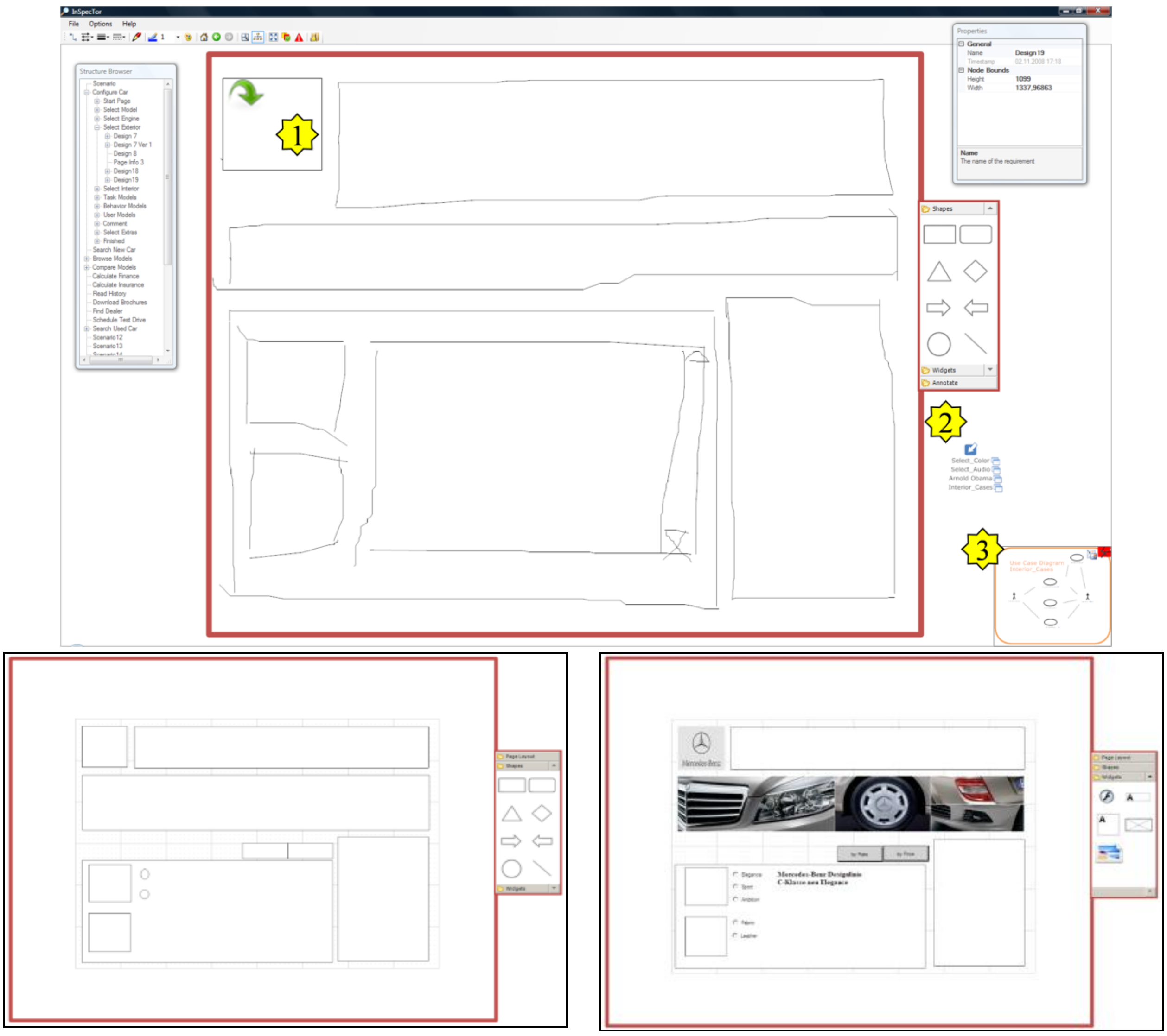


Figure 113: Sketch, abstract and mixed-fidelity UI design made with INSPECTOR (Memmel & Reiterer 2008; Memmel et al. 2008f)

Exporting and sharing the interactive UI specification

At the end of the specification process, Dr. Z forwards the interactive repository to the supplier (the principle: e.g. support open interchange, software is your primary goal). The supplier is able to click through the scenarios and the corresponding UI simulations, and he can drill-down into the electronic wall of wonders, which captures the requirements and all the artefacts that contributed to the design rationale of the UI that has to be implemented. In order to ease the process of actually coding the UI, the designs of the interactive UI specification can be saved separately from the rest of the specification archive. Although INSPECTOR also supports, and exports in, the usiXML language (Vanderdonckt et al. 2004; Vanderdonckt 2008), its compatibility with Microsoft XAML is more important.

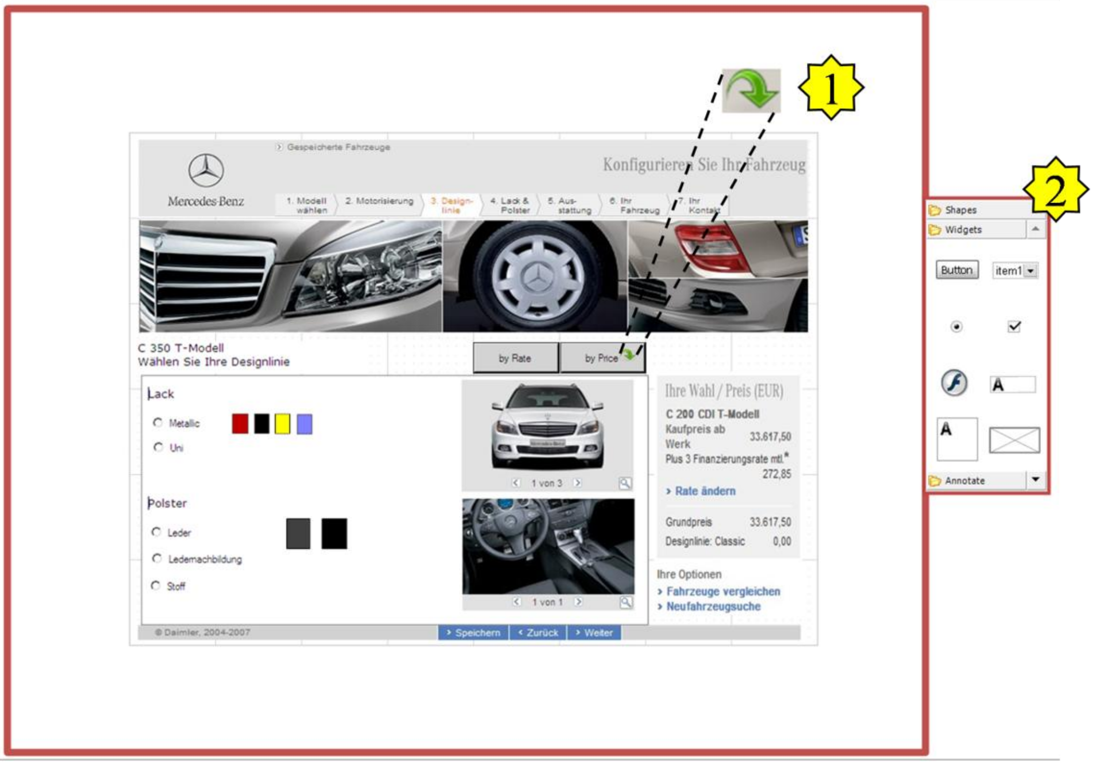


Figure 114: A specification-level UI design in INSPECTOR; as published in (Memmel & Reiterer 2008; Memmel et al. 2008f)

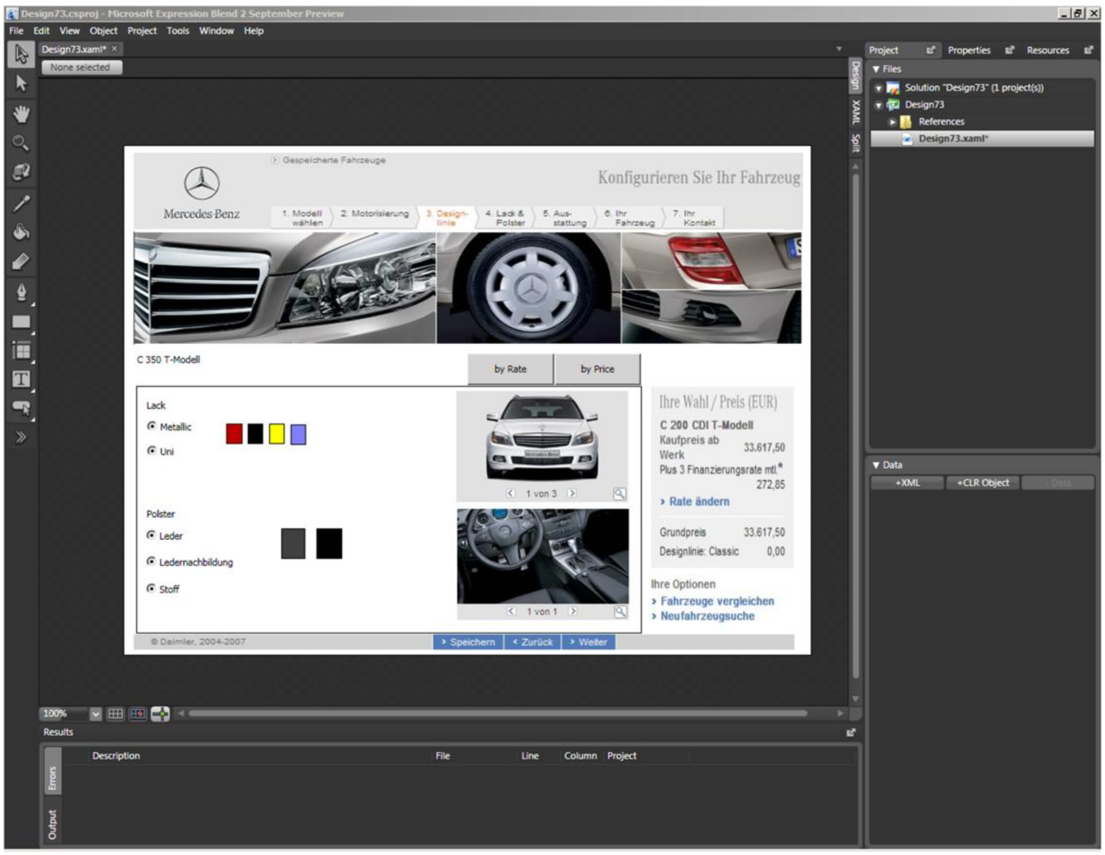


Figure 115: A specification-level UI design opened in MS Expression Blend 2

Because the programmer can open up the specification-level designs in Microsoft Expression Blend (see Figure 115), the final UI can be developed on the basis of the preliminary work on the client side. This saves time and money, and both parties can make sure the coded interactive application aligns with the specification. Due to XML, productivity is enhanced by seamlessly sharing projects, Microsoft Silverlight designs, Microsoft WPF designs, as well as in-progress XAML designs with Microsoft Visual Studio. This enables designers and developers to quickly build and test iterative revisions of the application's functionality and UI design, as well as allowing enhanced development capabilities without impacting the design freedom required during corporate UI specification.

6.5 Key Points

- INSPECTOR is based on important requirements for UI specification tool support, which were derived from an analysis of corporate UI development practice and several scientific criteria related to creativity, work style and agile processes, for example.
- INSPECTOR employs a zoom-based interaction style to optimally map the concept of design rooms and whiteboards to the computer-aided virtual tool-landscape. The ZUI approach is combined with other interaction concepts that stakeholders are familiar with from the WIMP interfaces and Office applications they use in their everyday work.
- INSPECTOR provides different zoom techniques to support a system of navigation that is suitable for the problem. The navigation operates through the UI specification space, which extends throughout the network of connected artefacts. Zooming and panning particularly simplify frequent and important activities such as switching between abstract and detailed representations (between models and UI designs, for example), as well as changing the perspective from the problem-space to the solution-space.
- INSPECTOR can be applied in different kinds of UI specification processes. However, its support for creativity, explorability and multiple perspectives of the specification and its artefacts, for example, distinctly favours iterative and incremental processes.
- The case study demonstrates the capabilities of INSPECTOR and underlines its claim to be able to replace primarily text-based Office applications in designing interactive systems in non-IT client organizations. The method and tool presented here instead support interactive UI specifications, as defined in Chapter 3, and therefore allow UI simulation and the traceability of requirements at any time.

Chapter 7 Empirical Studies

"A product, a service, a practice, or a perspective – however new and innovative – can have no impact without acceptance; no significance without change in people and their institutions"(Constantine 2001), p. 128

This chapter summarizes the results of three different evaluation studies conducted to test and enhance INSPECTOR and the underlying common denominator for interactive UI specifications. Chapter 7.1 outlines the results of an initial questionnaire-based study that confirmed at an early stage the contribution of INSPECTOR to corporate UI specification processes. Chapter 7.2 presents the results of a long-term usability study that was used to evaluate the usability and applicability of INSPECTOR over an extended period. The study revealed important findings about the understanding of the tool's conceptual model and highlighted some essential enhancements that would have been difficult to find in a lab-based test setting. Chapter 7.3 explains the consolidated findings from expert interviews conducted in the industry. The detailed study again underlined the requirements in UI specification processes and successfully pointed out the advantages of tools such as INSPECTOR. All usability studies were exploited to develop and upgrade INSPECTOR in the context of this thesis. Accordingly, many of the improvements that could be inferred from the results of the studies had already been incorporated in the tool as presented in Chapter 6. The overall results of the empirical studies are summarized in some key points in Chapter 7.4.

7.1 Questionnaire Study

Shortly after the first release of INSPECTOR, software and UI specification experts (n=6) were interviewed in a questionnaire-based usability study in order to assess the status quo of the experimental tool approach. The participants were introduced to INSPECTOR through a short demonstration, a video and a supplementary text explaining the motives for our approach. Each expert was provided with an installation of the tool and had two weeks to return their feedback by means of a questionnaire that was divided into 5 parts.

The first part was designed to (1) identify the field of activities of each respondent, (2) obtain an overview of the models and tools typically applied, and (3) obtain an assessment of difficulties along the supply chain. The second to fourth parts asked about INSPECTOR in respect of (1) the applicability of the modelling notations, (2) the completeness of the UI design capabilities and their practicability for UI evaluation, and (3) the assessment of the tool's general usability and the user experience provided. The fifth part asked if INSPECTOR could, in general, improve UI specification practice.

Results of the interviews

With regards to the results of the survey (see Table 3), all respondents stated that INSPECTOR, as a tool that combines models with UI design, contributed great value to their work style (average 4.83 pts; on a 5-point Likert scale). The added value was particularly identified in terms of an increased coherence of models and design artefacts, whereby INSPECTOR enhances traceability and transparency. Even the very early version of INSPECTOR was therefore expected to be able to improve existing UI specification practice (average 3.83 pts). The participants in the study were quite satisfied with INSPECTOR's support for text-based and graphical requirements modelling (average 4.00 pts).

Nevertheless, the feedback pointed to the necessity for a better linking functionality between the modelling artefacts. Some experts stated that while creating a UI design, the interaction with INSPECTOR could be enhanced by a contextual layer. This would give the expert the opportunity to easily crosscheck the design with underlying models. Instead of frequently jumping back and forth on the canvas, it would then be possible to temporarily visualize models and UI concurrently.

Consequently, we implemented a visualization that highlights all outgoing and incoming links of a model in order to enhance traceability. The experts regretted the absence of some important features such as master components and templates, which was due to the experimental nature of INSPECTOR's design and prototyping facilities. These features are needed to enable rapid prototyping and quick generic changes. In addition to a copy & paste mechanism that was required for the UI design layer, we therefore also implemented support for grouping UI elements and storing them in a template repository. In order to improve the utility of INSPECTOR during usability evaluations of modelling and design artefacts, we also developed an annotation component. During meetings, discussions and feedback sessions, sticky notes can now be attached to all artefacts on the specification canvas. This allows the recording of feedback and design decisions for later consideration during subsequent specification tasks. The notes can be accessed in a spreadsheet component that allows sorting and filtering, as well as jump navigation towards them.

Table 70 Overview of feedback; average points based on a 5-point Likert scale

Questionnaire topic	**Avg.**
Ability to integrate documents and logic with INSPECTOR	3.66
Opportunity to capture conceptual and schematic ideas	3.83
Support for user, task and interaction modelling	4.00
Linking models and thereby increasing traceability and transparency	3.66
Text-based and graphical requirements modelling (aggregated)	3.79
Accessibility of the prototyping features	3.16
Functionality provided at the UI design layer	3.40
Applicability of the UI designs for usability evaluations	3.33
Possibility to link UI designs in order to create a simulation	3.25
Overall UI prototyping capabilities (aggregated)	3.28
Opportunity to get both overview and detail on the specification space	3.33
Helpfulness of the zoom-interaction style during prototyping and modelling	3.00
Support for switching between created artefacts	3.50
Accessibility of all necessary information on the zoom canvas	3.50
Overall rating of the interaction with INSPECTOR (aggregated)	3.33
Overall contribution of INSPECTOR to existing UI specification practice	3.83
Improvement of work style through a combination of different models with multi-fidelity UI design	4.83

7.2 Diary Study

Scope of the study

In the first interviews (see Chapter 7.1), several usability issues concerning the general interaction with the tool were found. In order to address these issues and to be able to enhance the tool's quality, INSPECTOR was applied during an interaction design lecture in the context of a long-term diary study. Three groups of computer science and HCI students (n=8) were asked to use the tool during a Volkswagen AG use-case study on the specification of rear-seat entertainment systems. For a period of three weeks, every student wrote their own diary to give insight into (1) the kind of models created, (2) additional tools that were used, (3) problems that occurred, (4) the ratings of the user experience, (5) general issues and opinions about the tool.

Advantages of the diary study

The diary study was chosen as a usability method in order to able to evaluate INSPECTOR over a longer period of time. Because the question of how the empirical results change with the duration and intensity of usage was of great interest, a long-term study was more appropriate than a classical usability test. In weekly workshops, students and advisors discussed the intermediate results and recorded the issues for subsequent correction.

By means of the diary study it was found, for example, that objects on the ZUI canvas occasionally behaved inconsistently after the tool had been used for several hours and a large number of zoom operations had been performed. Students also reported issues with INSPECTOR-integrated external documents (PDF, Word, etc.) when these were repeatedly opened and saved. This resulted in a disarrangement of the XML structure in saved project files and was significant in preventing a fluent and continuous work style. To have been able to identify these problems in a much shorter lab-based usability study would have been a matter of pure chance. Thanks to the diary study, we were able to solve these issues quickly. Moreover, it was recognized that some participants preferred to create the first abstract prototypes with paper and pencil initially. Accordingly, the use of the built-in sketching mechanism increased as soon as a pen tablet was provided as an input device.

In addition, it proved to be very difficult to rapidly prototype UIs with point-and-click interaction on the canvas. It is therefore now necessary to evaluate different pen-tablet technologies that can be permanently combined with INSPECTOR. This will significantly increase performance during design sessions. A further point is that students were initially not comfortable with all the notations provided and required assistance for their proper application. This issue could be addressed by beginning work on a help feature that explains notations as well as their scope of application. Lastly, the affordance of templates for personas or essential-use-cases, for example, is to be enhanced to ease the understanding of the artefacts.

Summarized results

Ultimately, the diary study and the upgrades resulted in an improvement of the feedback on the tool's usability. Rated with an average of 1.75pts (std. 0.46) (on a 5-point Likert scale) after the first week and 3pts (std. 0.00) after the second, participants assessed INSPECTOR with an average of 4.25pts (std. 0.46) at the end of the study. A repeated-measure ANOVA revealed a significant main effect for the rating across the weeks ($F(2,14)=105.00$, $p<0.001$).

Furthermore, the differences between each week are also very significant statistically (week 1 vs. week 2: $F(1,7)=58.33$, $p<0.001$; week 2 vs. week 3: $F(1,7)=58.33$, $p<0.001$). The corresponding inter-rater agreement was assessed by calculating the intraclass correlation. This revealed a significant correlation of 0.99 ($p=0.000$), indicating a high homogeneity in the subjects' ratings of the system across the three weeks.

The overall results were a motivation to further enhance INSPECTOR, taking into account the proposed changes and add-ons. The diary study helped in preparing INSPECTOR for a demonstration in the industry, this being necessary to obtain an assessment from those experts and stakeholders who are the potential users of both a method and a tool support that are based on the idea of interactive UI specifications.

7.3 Expert Interview

Purpose of the study

After the promising feedback received at the later stages of the diary study and the corresponding enhancements to INSPECTOR, another evaluation study with HCI experts was conducted to obtain consolidated feedback from experienced people who are likely to be potential users of both the interactive UI specification method and tool support that come with Inspector. By means of an interview study, the completeness and applicability of the UI specification method and the experimental tool support was to be assessed. Experts at the research facilities of Daimler AG (n=3) and the department for UI design and consulting at Siemens AG (n=5) were invited to participate in the study. The interviews were semi-structured in order to preserve the opportunity to drill-down into detail on interesting issues related to typical UI specification practice, while still maintaining replicability. In a 30-minute group presentation, all participants were introduced to both method and tool approach. Afterwards, each interviewee was given a schedule for meeting the interviewers in the usability lab of the respective institution.

Structure of the interview

The warm-up phase was designed to ask specific questions on the interviewee's typical assignments during UI-related projects. Another question asked which tools are typically employed during requirements gathering, modelling, and design. In addition, the participants were asked to reflect on tasks they perceive as being difficult to execute. The main part of the interviews was divided into two sections.

Analysis of UI specification method

The first section aimed to analyze the kind of UI specification process that already exists in the interviewee's organization. At this stage, questions were related to the kind of artefacts used and the communication with other disciplines. One aspect of interest, therefore, was to find out about specific artefacts intended to ease interdisciplinary collaboration. Other questions were targeted towards the formality of the artefacts employed and the connectivity between the artefacts, especially between those related to modelling on the one hand and design on the other. Finally, this section of the main part of the interviews discussed the proposed 'common denominator' for UI specification with the participant. With regards to its component parts, the interviewees were asked to comment on its completeness and soundness.

Analysis of UI specification tool

The second section of the main part was reserved for discussing the modelling and UI prototyping capabilities of Inspector that are based on the common denominator. For this purpose, the tool was either demonstrated by the interviewer or used by the interviewee himself to discover its functionality. Concurrently, questions about the usability of the zoom interaction used in INSPECTOR were posed.

Analysis of added value

The cool-off phase of the interview was designed to address general questions about the applicability of INSPECTOR during UI specification processes in the interviewee's organization. Accordingly, the interviewee was asked to make statements about weaknesses in both method and tool. With regards to the support for collaboration that INSPECTOR provides, the interview ended by asking for the interviewee's opinion on combining INSPECTOR with the Powerwall and multimodal input.

The dominance of Office applications

The warm-up phase again provided interesting insight into the work style of interaction designers in the industry (see Table 71). The results underline that most experts are typically involved during many stages of UI specifications. This includes the early stages of requirements gathering as well as the later stage of detailed UI design. The work style of all experts interviewed is very much determined by the use of Office applications, especially Microsoft Word, Microsoft PowerPoint and Microsoft Visio. Other tools such as Photoshop or MindMap are also used frequently. This result corresponds with the findings of the studies of (Bock & Zühlke 2006; Memmel et al. 2007a) presented in Chapter 1. Not surprisingly, these tools are used across all phases of UI specification to develop both models and designs. Here, the experts all have their own individual preferences and employ the tools that they prefer. Usually, no organizational guidelines constrain the tool usage, which ultimately leads to the media discontinuities we have already identified in previous studies (see Chapter 2).

Lost in specification space

Accordingly, experts stated that they were uncomfortable with the task of finding information in the created artefacts. The lack of role-specific access requires frequent scanning of extensive text-based documents. Hence, experts are overwhelmed by the artefacts that they themselves create. This phenomenon can be described as the 'lost in specification space' (LOST) problem. The problem is caused by simply having too many different artefacts that lack connectivity.

The wood for the trees problem

The phenomenon is therefore directly associated with a lack of traceability and patency. After raw requirements data has been gathered, it is likely that it will no longer be updated. Once requirements have been modelled, they will be used just once to work out first designs. Although it was stated by interviewees that a premature focus on design should be prevented, the absence of means for roundtrip engineering will indirectly lead to a trial-and-error UI specification process, rather than a design by engineering. As experts therefore tend to lose sight of the requirements they once carefully gathered, this phenomenon can be called the 'not seeing the wood for the trees' (NOTE) problem.

Table 71: General questions and answers on UI specification practice (warm-up phase)

Question	**Answers (aggregated from n=8 recorded replies)**
What are your typical assignments in UI specification projects?	Conducting user and (contextual) task analysis; development of style guides; creating mock-ups; assembling the specification sheet; use-case design; developing personas; evaluation; creating low-fidelity and high-fidelity designs
What are the typical artefacts, methods and tools you apply for UI specification?	Modelling dialogue flow and UI design with Microsoft Visio; writing session protocols with Microsoft Word; creating use-cases in Visio; creating task models in Visio; recording current dialogue flow in flow charts; prioritisation of requirements; drawing navigation maps in PowerPoint; discussing concepts using flipcharts; creating paper mock-ups for communication; abstract prototypes in PowerPoint; detailed design in Photoshop; task modelling in MindMap
Which UI specification tasks do you rate as being difficult?	Searching information in a text-based specification document; lack of role-specific access to specification documents and embedded information (lost in spec. space); related information is hidden in overwhelming textual descriptions (wood-trees problem); too many text-based documents; lack of patency and traceability; media discontinuities hamper spec. process; maintaining traceability; translating raw data into requirements; preventing a premature focus on design; handling change requests; maintaining the timeliness of artefacts over time
What does the UI specification you develop look like?	Numeric values on pixel-level; extensive text documents including annotated images; often incorporates detailed description of context of use; word documents supplemented with mock-ups made in Photoshop; Word documents with many, many images;
How do you balance models and design?	Prototype is part of the UI spec., developed in parallel; no specific prototypes applied to prove completeness and consistency of task cases; text and models contain what the prototype cannot express;
Do you have general comments?	The UI prototype is more determining than any text document; patency could be reached through integrating tool or well defined interfaces of a tool-chain; software developers do not request formal diagrams from HCI experts; the supplier is usually not interested in requirements, but just in what has to be built; specifications describe everything in detail, while style guides determine design at a higher, rather general level;

Prototypes as important specification artefacts

Both the LOST and the NOTE problem might be symptomatic of the strong dominance of textual documents. Because experts lose the overview of what they want to specify, they tend to describe everything in a verbose style. By the nature of excessive texts, a certain degree of redundancy and repetition comes with the kind of UI specifications that are produced in this way. As stated by the interviewees, the usual outcomes of their UI specification efforts are therefore enormous Word documents that are supplemented with images of UI elements, layouts or detailed UI designs. If the specification process allows for concurrent development of UI prototypes, for example with the help of graphics designers, these prototypes are often more important than what was determined in the narratives. This is in line with the need to use prototypes as a means of participatory design and interdisciplinary UI specification. In the experience of many interviewees, the IT supplier in charge of coding the final system is primarily focused on prototypes.

Getting away from text

While the warm-up phase of the interview provided a first insight into the high-level concerns of UI specification, the main phase served as pathfinder in understanding the day-to-day tools of the HCI expert's business in detail. In this process, it was very interesting to find out the specification artefacts preferred by the interviewees (see Table 72). Consistent with their experience with text, experts stated that they do not like working with scenarios and other text-based artefacts. Conversely, they prefer to produce more structured artefacts such as personas, user profiles, use-case diagrams or dialogue-flow models.

Traceability through linking

Creating and maintaining links between the artefacts created was frequently stated to be among the most important issues in UI specification. But with regards to the kind of tools used in practice, interdependencies between models are usually not developed. This is called the LINK problem throughout the remainder of this thesis. The lack of connectivity of artefacts is especially critical when it comes to the relationship of models and UI design. Links between the UI and requirements are accepted as an important vehicle for validating consistency and completeness. It is also argued that the traceability of the design process is improved by links.

Table 72: Questions and answers on applied models and ways of bridging the gaps during UI specification (first section of main phase)

Question	**Answers (aggregated from n=8 recorded replies)**
Which models and artefacts do you employ when working with experts from other disciplines?	Paper-based documents (text); mock-ups; click-through prototypes; user roles; user profiles; personas; use-case diagrams can be part of the specification document; use-cases (in various levels of formality); models that externalize dialogue flow are very important; UI storyboards are useful, if created 'agile'; style guides (add-on to spec. document)
Which artefacts would you rate as inappropriate for interdisciplinary UI specification?	Scenarios, as they are too verbose; text-based documents (too abstract, ambiguity); personas are developed rarely (prefer user profiles or user roles)
How important are links and transitions between different artefacts? How are the links and transitions developed?	Transitions between UI states are most important; links often do not exist (difficult to create and maintain in text documents such as Word); transitions are important, but costs for keeping requirements up to date during the design stage are too high
Is it important to link graphical models and UI designs?	Important to link use-case diagrams to UI designs as use-cases lack information about properties of UI elements; relationship very important; prototype used to crosscheck with user task; linked artefacts give the expert more power when arguing for design decisions; linking is very important
What is your feeling about the proposed common denominator for UI-related modelling and design?	Abstract prototypes only used internally (spec. team), as too few details for discussions with decision makers; abstract prototypes essential to understand in team; storyboards often too playful; flow charts preferred over storyboards (more technical); use-cases and use-case diagrams very important (pushed by organization); user roles very important; flow charts often too expensive (rather switching to design); the business vision is very important and often ignored – but it's important to document what the IT system is designed for
Which grade of formality do you consider useful during interdisciplinary UI specification processes?	Formality of common denominator is very appropriate; employed artefacts rather informal, but process is formal; the closer the project is to releasing final design, the more formal (i.e. precise) the artefacts should be; more formal work style would harm UI specification process; formality with respect to writing personas or use-cases is adequate; in very technical projects (e.g. medical apparatus), strict UML diagrams are sometimes requested
What general comments to you have?	HCI expert has to learn and adopt technical notations; projects that employ different modelling languages and prototypes are known to be more successful; incorporating new means of UI specification faces the challenge of changing well-established and often intractable mind sets

Upgrading the role of the interaction designer

Interaction designers are in a much more comfortable position when they can easily demonstrate how they worked out a design from the requirements gathered. Hence, being able to turn UI specification processes into a traceable white-box process is an opportunity to become a more important person in the design team – an aspect critical for the success of UE processes as well as the related personal career, and at the same time probably one of the most compelling issues for interaction designers in the industry.

Evaluation of the common denominator

One of the main ideas behind the common denominator is the association of models and designs to form a sound and complete UI specification. Obtaining an assessment of the proposed set of artefacts from experienced practitioners was therefore one of the most interesting aspects of the expert interviews. With regards to the design layer, several experts stated that abstract prototypes are a very important means for discussing design alternatives. However, abstract prototypes are mostly used internally. As part of an interactive UI specification, they will therefore usually have the role of design artefacts that inform others about decision-making and trade-offs. The integration of (essential) use-cases and use-case diagrams was rated as being very important due to the significance of the models in most organizations. Flow charts, in turn, tend to be produced infrequently. In simple projects, most of the experts prefer to visualize dialogue flow directly with prototypical designs and screen flows. It is, however, recognized that flow charts are necessary in more complex design situations.

The formality of the UI specification process

Taking into account the artefacts of the common denominator and the discussion of the appropriate formality in interdisciplinary UI specification processes (as outlined in Chapter 3), the participants of the interview study were asked to give their opinion on the formality required for their projects to be successful. Most experts argued that too much formality would particularly harm collaboration with other disciplines. When asked how to define formality, it was stated that the structure of personas is already perceived as formal. This degree of formality is well accepted in project teams. However, experts explained that, with progress towards the final system, the employees' means of expression become more formal. With regards to the artefacts that were mentioned in the interview, this confirms, for example, the selection and integration of semi-formal diagrams derived from SE (see Chapter 4). One participant also stated that in very technical projects, the client often demands strict UML diagrams. For example, this is the case when designing the UI of medical apparatus, which has to be constructed with regard to ISO standards.

Table 73: Questions and answers on INSPECTOR (second section of main phase)

Question	**Answers (aggregated from n=8 recorded replies)**
What is your opinion on the zoom-approach of INSPECTOR?	Switching between abstract and detail very well supported, especially travelling from the detail to the abstract; animated zooming eases understanding of spec. space
Does the zoom-based interaction support switching artefacts?	Contextual layer helps modelling and design; zooming helps to make UI specification a white-box process
Can INSPECTOR increase transparency and traceability?	History and versioning help to understand design decisions; links between models considerably help traceability and transparency
Do the prototyping capabilities satisfy your requirements?	Generic changes are not well supported (consider master components); design capabilities should be similar to PowerPoint and Visio; mixed-fidelity detail is often required in design; some UI states must be described textually (prototyping everything is too much), e.g. in tables that contain information about behaviour
Do you have general comments?	Provide role-specific access to INSPECTOR (e.g. filter out some information); dissolve hierarchy of notations in order to reduce complexity; increase the functionality of the overview window to prevent disorientation; notations should preferably be organized as a network (no hierarchy); overview is important to maintain orientation; hierarchy is a good way to organize design artefacts; different starting points would increase flexibility

The contribution of INSPECTOR

Some interviewees mentioned that interaction designers need to have an understanding of technical terminologies and modelling languages if they wish to successfully participate in various UI-related projects. This accords with the definition of the job profile of the interaction designer provided in Chapter 3. However, many interaction designers may have an intractable work style that makes it difficult for them to adapt to new methods and tools. It was therefore important to discuss the general contribution of INSPECTOR (see Table 73) in the second part of the interview's main phase, and the opportunity for acceptance of INSPECTOR in the interviewee's organization in the cool-down phase of the interview (see Table 73).

Switching between abstract and detail

The zoom-based interaction style of INSPECTOR is consistently recognized as an ideal means to overcome the LINK and LOST problems, In particular. Animated zooming visualizes the topology of the specification space, while geometric and semantic zooming support switching between abstract and detailed representations. All told, zooming is recognized as a means to make UI specification a white-box undertaking. The newly developed contextual layer successfully decreases the number of zoom operations during modelling and design activities and correspondingly increases the usability of the tool.

Traceability and transparency

History and versioning features additionally support the work style of experts as these features help to preserve design decisions and help to guide design in the right direction. In this area, INSPECTOR's functionalities therefore help to overcome the well-known NOTE problem. As the repository of all important information items, INSPECTOR provides excellent traceability and transparency. With regards to decision-making and discussion with stakeholders, it is therefore recognized as a UI specification workbench that strengthens the role of the interaction designer.

Opportunities for enhancement

Concerning the opportunities for future enhancements, the interviewees proposed role-specific access to the specification space. Filtering out parts of the UI specification is expected to increase efficiency and effectiveness when locating relevant information items, especially in very complex projects. Together with more sophisticated versions of the embedded overview window, orientation in the specification space would be further simplified. With regards to the complexity of UI specification tasks and the usability of INSPECTOR, experts were at variance over the hierarchy of artefacts that is imposed by the common denominator and its mapping on the zoom-canvas. Some experts regard the hierarchy as an adequate means of organizing accumulated work artefacts. However, they expect that the hierarchy should be flexible in terms of letting the tool-user decide on the starting artefacts in the subsequent layers. This is in contrast to the currently fixed hierarchy of notations in INSPECTOR and is an important aspect for future work (see Chapter 8). Other experts explained that they regard developed artefacts as a network of related information, which comes close to the need for linking and association. Hence, if INSPECTOR could let the user decide on how the artefacts should be organized on the canvas, the user experience would be likely to increase.

Collaborative work style

However, the conversations in the cool-down phase (see Table 74) of the interviews proved that INSPECTOR, in its current version, is already well recognized as a valuable contribution to the existing tool landscape. As suspected from earlier findings, the experts interviewed explained that valuable information is currently lost along the supply chain of the UI specification processes in which they participate. The opportunity to save important work artefacts, together with INSPECTOR's prototyping capabilities, therefore supports the experts very well in their everyday work. While experts from Daimler AG are often confronted with situations that demand close collaboration, interaction designers from SIEMENS AG usually run a UI specification project on their own. Accordingly, the requirements for collaborative workbenches differ. All the experts recognize the added value of fruitful discussions with stakeholders, but although their organizations provide collaborative IT systems such as digital whiteboards, they are not often used due to the high threshold in setting them up. Hence, increasing INSPECTOR's support for collaborative modelling and design has to focus on two aspects concurrently. Firstly, collaboration must be possible not just on gigapixel displays but also on smaller screens. Hence, input devices and screen space must be flexible in order to be available in any circumstance.

Secondly, the tool support must be extremely easy to use and easy to learn. Stakeholders must be able to use the tool as easily as they use paper for sketching out models or designs. If the threshold of using the electronic tool support is higher, the added value must in turn be significantly increased as well.

Table 74: Question and answers of the cool-down phase

Question	**Answers (aggregated from n=8 recorded replies)**
How would you rate the added value of using INSPECTOR in the UI specification process of your organization?	Information from many meetings and focus groups is currently lost, could well be saved with INSPECTOR; the tools enables a situation-dependent definition of a design process (what to do first, what do later); INSPECTOR's functionality together with Flash make up the 'dream team'
What in your opinion is the potential for increasing INSPECTOR's abilities to support collaboration?	Collaboration mostly takes place at the UI layer; collaboration is important (typically many actors start sketching during discussions); having a shared means of communication drives the design process; most consultants work at the client's site – powerwall would not be available; using the tool at the powerwall must be easy (low threshold, high ceiling)
What are your general concerns with method and tool?	Functionality to produce reports is missing (e.g. Word document); need for an extensive help feature providing guidance on UI design and modelling, and how the UI specification is to be developed; the dominance of Office-like applications is difficult to overcome
What enhancements would you propose?	Provide role-specific access to be able to cross-reference work artefacts with the tool-users that developed them; provide facilities to integrate a wider range of documents (not just Office, PDF, etc)

Possibilities for successful establishment

In general, the expert interviews confirmed the usefulness of the method and tool support proposed in this thesis. If the prototyping capabilities become more sophisticated and allow for generic changes, the possibility of competing successfully with dominant Office applications is high. Overcoming the domination of Word, PowerPoint and other Office applications is a success-critical factor for changing existing UI specification cultures and mindsets. Some kinds of reports must, of course, still be possible in the form of Word documents, for example. INSPECTOR must support their creation as text-based documents, as they will continue to be important for contract documentation.

Summary of most important findings

The most important outcome of the expert interviews is the confirmation that the core concepts of INSPECTOR are the kind of support that is urgently required by interaction designers. The LOST, NOTE and LINK problems were shown to be successfully addressed by INSPECTOR. This confirmation also included our choice of modelling notations and design layers, which proved to be the most important vehicles for the interaction designers in the industry.

7.4 Key Points

- Early tests of INSPECTOR showed that combining UI modelling and UI design in a single tool is a vital requirement for supporting stakeholders with different backgrounds in corporate UI specification processes. In this context, traceability and transparency are among the constantly cited issues in UI specification.
- INSPECTOR's zoom-based concept supports the work style of modellers and designers. But in order to reduce zooming and panning operations, travelling through the specification space must also be accompanied by contextual visualizations of artefacts related to the focus. For crosschecking models and design or vice versa, the contextual layer introduced after the usability studies proved to be very helpful. While zooming is acknowledged to facilitate the activities along the UI specification supply chain, features such as versioning or contextual layering of artefacts are believed to strengthen the role of the expert in the overall process.
- For sketch-based prototyping, which proved to be extremely important for experts in early stages of UI specification, mouse and keyboard are not the perfect input devices. INSPECTOR should therefore be combined with well-suited input devices to foster creativity and agility along the supply chain. Multi-modal input is therefore among the most interesting enhancements for collaborative and interdisciplinary UI specification tool support.
- Templates were found to be very important for stakeholders, because they ease the development of new artefacts and guide stakeholders in making the right choices or filling in the correct information. In addition, INSPECTOR should be upgraded by incorporating a contextual support system that guides its users through the whole UI specification process; for example, suggesting a certain sequence in modelling. Moreover, templates could be combined with UI or HCI patterns (Borchers 2001).
- The expert interviews again underlined the need for innovative UI specification tools that enable a departure from the use of Office-like applications such as Microsoft Word or Excel. When using these kinds of tools in UI specification processes, experts are frequently overwhelmed by the vast number of artefacts (LOST problem) and are nearly always unable to keep track of relationships between artefacts and the underlying requirements (LINK problem). This is why they lose sight of artefacts in complex UI specification spaces (NOTE problem). INSPECTOR was acknowledged as being a very suitable approach to solving these issues.
- The common denominator proved to be a quite realistic and appropriate set of models and UI design artefacts. Because it is accepted that IxD has to become open-minded with regards to the application of more formal means of UI modelling, the set of artefacts assembled for UI specification was welcomed and believed to be quite complete. However, the hierarchy of diagrammatic notations and prototypes does not need to be mapped one-to-one to the ZUI of INSPECTOR. An analysis is required as to whether the decomposition of the hierarchy in favour of a rather planar arrangement of artefacts in the specification space would leverage the usability of the tool. The trade-off between organization of artefacts and understanding the links between them is to be evaluated in future work.
- In order to extend the applicability of INSPECTOR, the tool should be able to run on screens ranging in size from small to very large. A major challenge for future work is to upgrade the scalability of INSPECTOR's interface. Supporting various screen resolutions relates well to the subject of multi-modal input styles.

Chapter 8 Summary and Conclusion

Overview of this chapter

This chapter summarizes the contribution of the method and tool support presented in this thesis. For this purpose, the following sections reconsider the problems of corporate UI specification and the compiled requirements for improving the overall process. With regards to future work, as presented in Chapter 8.2, the advantages of interactive UI specifications and the proposed experimental tool support are tempered by issues that must be resolved in order to make the package complete and mature for application in the field.

8.1 Summary

Focus of this thesis

This thesis analyzed the challenges of corporate UI specification processes. The most important aspects that contribute to the successful design of corporate software systems are (1) adequate means for communication and collaboration, as well as common means for externalizing design thoughts, (2) modelling properties of the UI in expressive but easy-to-use notations, and (3) different means for UI prototyping that narrow the specification space towards a detailed specification-level design that will guide actual coding of the UI.

UI quality is success-critical

In UI specification, structured approaches are required that allow the gaps between the disciplines of SE, IxD, and BPM to be bridged. A shared course of action for all stakeholders is required in order to develop high UI quality, which today is a factor that is critical for economic success. Style guides and the safekeeping of design rationale and important design artefacts are very relevant to the process of corporate UI specification.

Towards interactive UI specifications

In this context, this thesis has analyzed the shortcomings of the text-based UI specification processes that are today common in the industry. Due to the focus of this work on non-IT organizations, a special aspect of corporate UI specification is the ease of use of the corresponding UI tools and the modelling notations provided, their purpose being to support requirements expression and design vision. In Chapter 3, the need for new and completely different means for UI specification was inferred from a survey of common practice. The concept of interactive UI specifications supports all stakeholders in driving the overall process with simulations of the look and feel of the UI. Because the systematic prototyping-driven UI specification process also relies heavily on the models that document and visualize the requirements of the UI, interactive UI specifications extend the usual prototypes by additionally carrying models below the UI layer.

The common denominator

The detailed ingredients of interactive UI specifications and the nature of the models to be included was discussed and determined in Chapter 4. Agile, semi-formal notations are used to assemble the resulting common denominator for UI specification. The models are pragmatic, in the sense that they are capable of supporting the requirements of corporate stakeholders, and approximate, in the sense that they are just formal enough to express important aspects of the UI, while not overwhelming people. UI prototypes on a specification-level of detail are the living part of the interactive specification. In addition, sketch-based and canonical abstract prototypes supplement the design process by fostering creativity and design thinking with mixed-fidelity detail. In the end, the common denominator allows different disciplines to be united during UI specification processes without introducing new modelling languages. It is completely based on existing notations and design techniques, but assembles them into a traceable chain of models and designs, which in turn makes the UI specification process more transparent and appropriate with respect to the work style of the stakeholders.

Differentiation from related work

Previous research has shed light on the various necessary stages of UI design that need to be taken into account by specification methods and the corresponding tool support. The common denominator, as well as the INSPECTOR tool presented in

Chapter 6, takes the levels of the CAMELEON reference framework (Calvary et al. 2003) into proper account. It integrates all relevant levels of UI modelling, including problem-domain modelling, user modelling, task modelling and behaviour modelling. In addition to some text-based, but primarily diagrammatic notations, the common denominator recognizes several levels of UI design. In order to propel creative processes, which are necessary to innovate and deliver the strategic demand for high UI quality, the sketching and modelling of interrelated requirements is combined in a way that is new in the field of UI specification. In summary, the UI specification method and tool support proposed in this chapter is differentiated from previous work in terms of patency and coverage of all stages in designing corporate interactive software systems. Moreover, the proposed common denominator and the corresponding study of the ingredients required in sound (interactive) UI specifications opposes the hypothesis of software companies such as iRise, which tries to dismiss the undeniable necessity of requirements modelling in conjunction with UI prototyping. The common denominator clearly argues that UI simulation alone is not enough to build interactive software systems. In this context, the proposed solution is also focused on a specific target audience. Consequently, both the common denominator and the INSPECTOR tool are designed to take the demands and work style of non-IT stakeholders and people with a rather informal work style into account much better than other solutions currently on the market (see Chapter 5).

The INSPECTOR tool: putting theory into practice

The INSPECTOR tool is able to outpace existing solutions by its integrating nature and an interaction style that perfectly matches the most frequent and important activities in UI specification. The zoom-based UI specification space provides means for sketching, modelling, designing and annotating in a way similar to whiteboards and the design-room setting in the real world. It is therefore a powerful repository for design artefacts and functions not only as an electronic interface for mature models and prototypes, but also as a safekeeper of design decisions and design rationale. The interaction via panning and zooming is based on natural work with physical artefacts and gives very good support to maintaining overview (i.e. stepping back in order to see more and orientate), while being able to drill-down into detail (i.e. getting closer to a physical artefact and bringing it into focus). Because INSPECTOR can integrate all the important artefacts, including existing documents (for example Word documents, PowerPoint presentations, etc.), it can be utilized along the whole UI specification supply chain. Furthermore, it bridges the media discontinuities between client and supplier by providing an executable specification and the opportunity to reuse specification-level UI designs in subsequently applied development tools.

Assessment of the solution

Towards the end of this thesis, detailed reports on three different evaluation studies shed light on the contribution of both the method and the experimental tool support. The necessity of having informal means for UI design were underlined by early criticism regarding the absence of input devices for hand-drawing sketches. Experts consistently stated that the proposed tool support is a very good match for the requirements of experts who frequently take part in interdisciplinary UI specification projects. The evaluation studies concluding this thesis therefore successfully confirmed both the challenges in corporate UI specification and the framework of appropriate tool support, for which INSPECTOR is a good match. Naturally, the studies also revealed that INSPECTOR still has room for improvement. For example, integrating support for building templates and master components that ease and speed-up UI design has already helped to increase tool usability. Although the zoom interaction with INSPECTOR was well understood, the consequent hierarchical nesting of artefacts may unnecessarily hamper ease of use and ease of learning. Besides a few less urgent findings, the evaluation highlighted several issues that must be addressed in the future, as discussed in Chapter 8.2.

Setting the stage for change

INSPECTOR was developed based on the definition of a common denominator for corporate UI specification processes. Due to bilateral cooperation with software development companies such as Axure and iRise, the findings and solutions proposed in this thesis are likely to contribute to future releases of UI development suites. Because the proposed solution was overwhelmingly acknowledged in the industry, the UI specification process may be subject to revision. Naturally, the opportunity to implement a suitable prototyping-driven specification process depends on various factors, such as corporate culture, human mindsets, and the openness of both to new methods in UI development. In this context, success stories such as that of introducing a solid UI specification tool chain for modelling in-car information systems at Dr. Ing. h. c. F. Porsche AG (VDI/VDE 2008), which is partly based on (Memmel et al. 2007a), highlight the opportunity for promising changes in UI development culture, adjusted for the particular goal and target audience, of course.

8.2 Future Work

ZUIs and ZOIL

After almost three years of development, INSPECTOR has reached a mature state and can convincingly demonstrate its main features and ideas. However, the tool is still not stable enough to be employed in practice. Consequently, future work will focus on consolidating the existing implementation. In this context, the technical framework of INSPECTOR may be replaced by a new platform, namely the ZOIL framework for developing ZUIS, presented by (Jetter 2007; Jetter et al. 2008a; Jetter et al. 2008b), for example.

Developing ZUIs with a ZUI-based tool

Moreover, the focus of INSPECTOR on UI specification for innovative interactive systems could take the development of ZUIs into particular account. Because there is no UI specification for ZUIs, the zoom-based concept would be able to address their development in an ideal manner. Accordingly, the partial decomposition of the modelling hierarchy that was suggested in the usability studies could be implemented in conjunction with an extension or an adjustment of the common denominator to take the special requirements of modelling ZUIs into account. Ultimately, the experimental tool support should be enhanced in terms of more flexibility with regards to the domain of application and to the kind of UI that has to be built.

Important areas of future work

Independently of these modifications, both method and tool should be upgraded in respect of three dimensions (see Table 75) that are especially important for corporate UI specification in general, and the development of interactive UI specifications in particular.

Table 75 Overview of future work on INSPECTOR

Aspect	**Detailed information**
Simulation	Simulations propel the design process. An easy-to-use UI design layer must provide support for prototyping-driven development. UI patterns must be incorporated to ease design for stakeholders with no design background; support animated UI elements and provide means to make UI elements zoomable (especially for developing ZUIs)
Creativity	By providing interfaces to multi-modal input devices, such as laser-pointers or table-tops, creativity can be supported well. Depending on the situation, suitable input devices help to model and design that which is required
Collaboration	Simulation and support for creativity help to establish a collaborative UI specification style. By using different input devices and large high-resolution displays, actors can work in a computer-aided design room, equipped with an artefact repository. For the latter, we envision a database-supported versioning system that allows management and comparison of the stored artefacts.

Simulation

The current capabilities of INSPECTOR with respect to UI simulation are satisfactory for the specification of most kinds of common UIs. But, as discussed in the previous section, the means provided for modelling and design will have to undergo radical expansion if the development of ZUIs is to be possible.

Creativity and collaboration

The other two dimensions are very much interrelated, and can be particularly addressed by making INSPECTOR's design room and whiteboard metaphor more dominant with regards to the appearance of the tool and the interaction with it. In this context, creativity and collaboration in UI specification can be fostered by providing a variety of input and output devices.

Gigapixel PowerWall setup

In meeting or media-room set-ups, INSPECTOR could enable users to cooperatively work on requirements models or UI designs during brainstorming sessions. Utilized as a gigapixel-wide electronic whiteboard, INSPECTOR records all the created artefacts in a structured manner. Actors can also work asynchronously using their own workspace, for example on a desktop installation. Modelled artefacts are then exported into XML documents and re-imported into a shared workspace, which resembles the common design rationale. Initial experimental setups with our high-resolution powerwall installation (4640 x 1920 pixels) at the University of Konstanz allowed a comprehensive view of our zoomable specification space (see Figure 116). This high-resolution display supports our ZUI approach by displaying a wide range of artefacts and relationships (overview) to all actors.

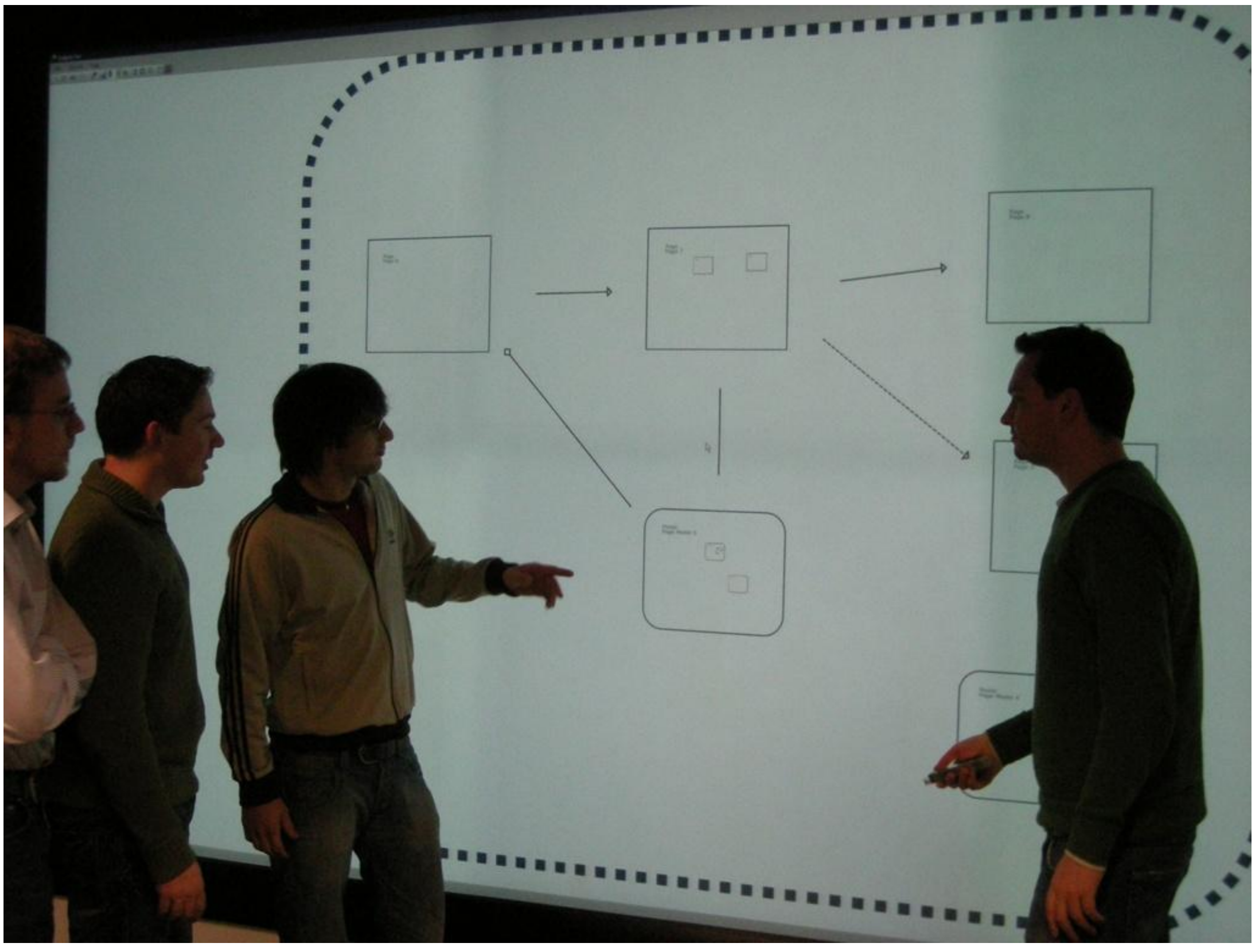


Figure 116: Evaluating INSPECTOR at the PowerWall installation at the University of Konstanz (Memmel et al. 2008c; Memmel et al. 2008d)

Laser-pointer interaction (König et al. 2007) enables an easy-to-use cooperative and more collaborative interaction style (see Table 75). Through point-and-click operations, actors explore and manipulate the UI specification space. In order to ease the collaboration with multi-modal input devices, INSPECTOR should follow a more straightforward zoom approach. In this context, abandoning the nesting of modelling and design objects on the specification canvas in favour of a large zoomable object-information landscape is also helpful. The handling of complexity will then be possible through search and filter functions, which provide quick access to artefacts.

In order to provide the whole functionality of INSPECTOR at the high-resolution display, multi-modal input devices are also necessary to foster creativity and collaboration. In addition to the laser-pointer as an input device, we are therefore considering PDAs or iPhones for text-input (e.g. writing text or annotations) and optionally for panning and zooming as well. Moreover, several stakeholders, using multiple devices, such as the laser-pointer or table-top, could work with INSPECTOR at the same time. Naturally, the INSPECTOR UI would need to be adapted to some extent to be usable without mouse and keyboard in every situation. Because browsing menu structures using a laser-pointer is a thankless activity, interaction with INSPECTOR could also be supported by speech-recognition. A media room, such as the one at the University of Konstanz (see Figure 117), provides a research environment for the design, development, and evaluation of novel interaction techniques and input devices, as well as a simulation facility for future collaborative work environments. It therefore offers different input devices (e.g. laserpointer, hand-gestures, multitouch, eye-gaze, speech) and output devices (Table-Top, HD-Cubes, audio & tactile feedback), which can be used simultaneously and in combination, creating a new dimension of multimodal interaction. Ultimately, a thoughtful combination of modalities for interacting with INSPECTOR will be a major part of our future research.

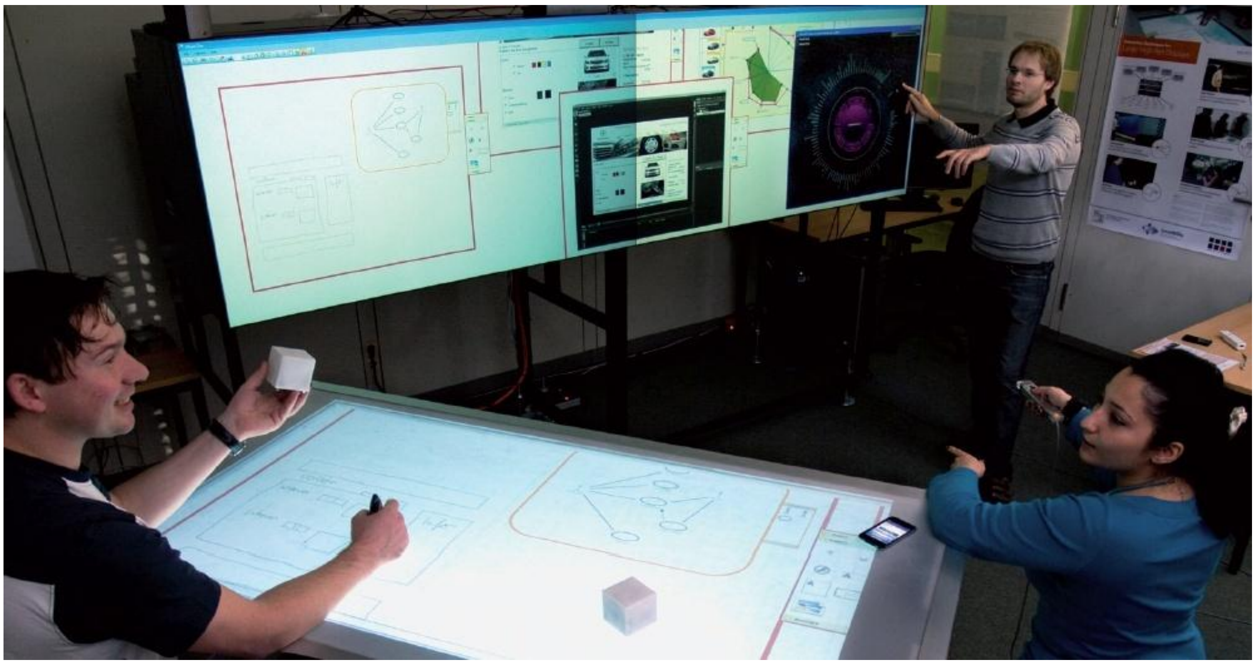


Figure 117: Using INSPECTOR in the media room at the University of Konstanz

Literature

Abrams, M. and Phanouriou, C. (1999), 'UIML: An XML Language for Building Device-Independent User Interfaces', in *Proceedings of the XML '99*, Philadelphia, USA.

Alexander, I. F., Maiden, N. and Wiley, J. P. (2004), *Scenarios, Stories, Use Cases Through the System Development Life-Cycle*, John Wiley & Sons, New York.

Alfonseca, J. N., Gouveia, A. C. and Nunes, N. J. (2006, 22.6.2006), *WinSketch*, viewed 9.10.2008, from http://apus.uma.pt/~winsketch.

Ambler, S. and Jeffries, R. (2002), *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*, Wiley.

Ambler, S. (2004a, May 6th, 2004), *Use Cases of Mass Destruction*, viewed 14.9.2008, from http://www.ddj.com/architect/184415814.

Ambler, S. (2004b), *The Object Primer - Agile Model-Driven Development with UML 2*, Cambridge University Press.

Ambler, S. (2006a, 2006), *Introduction to System Use Cases*, viewed 23.8.2008, from http://www.agilemodeling.com/artifacts/systemUseCase.htm.

Ambler, S. (2006b, 2006), *Agile Modeling (AM) Practices v2*, viewed 1.11.2008, from http://www.agilemodeling.com/practices.htm.

Ambler, S. (2006c, 2006), *Introduction to Data Flow Diagrams*, viewed 25.9.2008, from http://www.agilemodeling.com/artifacts/dataFlowDiagram.htm.

Ambler, S. (2006d), *Usage Scenarios*, viewed 23.8.2008, from http://www.agilemodeling.com/artifacts/usageScenario.htm.

Ambler, S. (2006e, 2006), *Introduction to Essential (Abstract) Use Cases*, viewed 14.9.2008, from http://www.agilemodeling.com/artifacts/essentialUseCase.htm.

Ambler, S. (2006f, 2006), *Introduction to UML 2 Use Case Diagrams*, viewed 14.9.2008, from http://www.agilemodeling.com/artifacts/useCaseDiagram.htm.

Ambler, S. (2006g, 2006), *Introduction to Flow Charts*, viewed 25.9.2008, from http://www.agilemodeling.com/artifacts/flowChart.htm.

Ambler, S. (2006h, 2006), *Introduction to UI flow diagrams*, viewed 25.9.2008, from http://www.agilemodeling.com/artifacts/uiFlowDiagram.htm.

Ambler, S. (2006i, 2006), *Introduction to UML Sequence Diagrams*, viewed 26.9.2008, from http://www.agilemodeling.com/artifacts/sequenceDiagram.htm.

Ambler, S. (2006j, 2006), *Introduction to UML Activity Diagrams*, viewed 26.9.2008, from http://www.agilemodeling.com/artifacts/activityDiagram.htm.

Anderson, J. R., Matessa, M. and Lebiere, C. (1997), 'ACT-R: A theory of higher level cognition and its relation to visual attention', *Human Computer Interaction*, vol. 12, no. 4, pp. 439-462.

Balzert, H. (2000), *Lehrbuch der Software-Technik*, Spektrum Akademischer Verlag, Heidelberg.

Barbosa, S. D. J. and Paula, M. G. (2003), 'Interaction Modelling as a Binding Thread in the Software Development Process', in *Proceedings of the ICSE'2003 Workshop on bridging the gaps between software engineering and human-computer interaction SEHCI'2003*, pp. 84-91.

Baudisch, P., Good, N. and Stewart, P. (2001), 'Focus plus context screens: combining display technology with visualization techniques', in *Proceedings of the 14th annual ACM symposium on User interface software and technology (UIST '01)*, ACM Press, pp. 31-40.

Baudisch, P., Good, N., Bellotti, V. and Schraedley, P. (2002), 'Keeping things in context: a comparative evaluation of focus plus context screens, overviews, and zooming', in *Proceedings of the SIGCHI conference on human factors in computing systems (CHI '02)*, ACM Press New York, NY, USA, pp. 259-266.

Bäumer, D., Bischofberger, W. R., Lichter, H. and Züllighoven, H. (1996), 'User Interface Prototyping - Concepts, Tools, and Experience', in *Proceedings of the 18th International Conference on Software Engineering (ICSE 1996)*, Berlin, Germany, pp. 532-541.

Beck, K. (1999), *Extreme Programming Explained*, Addison-Wesley.

Bederson, B. and Boltman, A. (1998), 'Does Animation Help Users Build Mental Maps of Spatial Information? ', Computer Science Department, University of Maryland, College Park, MD, Tech Report CS-TR-3964.

Bederson, B. and Boltman, A. (1999), 'Does Animation Help Users Build Mental Maps of Spatial Information?', in *INFOVIS '99: Proceedings of the 1999 IEEE Symposium on Information Visualization*, IEEE Computer Society, p 28.

Bederson, B., Meyer, J. and Good, L. (2000), 'Jazz: an extensible zoomable user interface graphics toolkit in Java', in *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, ACM, pp. 171-180.

Bederson, B., Shneiderman, B. and Wattenberg, M. (2003), 'Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies'. in *The Craft of Information Visualization: Readings and Reflections*, Bederson, B. B. and Shneiderman, B. (eds.), Morgan Kaufmann.

Bederson, B., Grosjean, J. and Meyer, J. (2004), 'Toolkit Design for Interactive Structured Graphics', *IEEE Trans. Softw. Eng.*, vol. 30, no. 8, pp. 535-546.

Belenguer, J., Parra, J., Torres, I. and Molina, P. J. (2003), 'HCI Designers and Engineers: Is it possible to work together?', in *In IFIP Working Group 2.7/13.4, INTERACT 2003 Workshop on Bridging the Gap Between Software Engineering and Human-Computer Interaction*.

Bengtsson, P. O. (2002), 'Architecture-Level Modifiability Analysis', Department of Software Engineering and Computer Science, Blekinge Institute of Technology, Technical Report.

Benyon, D., Turner, P. and Turner, S. (2005), *Designing Interactive Systems: People, Activities, Contexts, Technologies*, Addison Wesley.

Berkun, S. (2000), *The art of UI prototyping.*, viewed 6.8.2008, from http://www.scottberkun.com/essays/12-the-art-of-ui-prototyping/.

Berkun, S. (2002), *The list of reasons ease of use doesn't happen on engineering projects*, from http://www.scottberkun.com/essays/22-the-list-of-reasons-ease-of-use-doesnt-happen-on-engineering-projects/.

Bevan, N. (1995), 'Measuring usability as quality of use ', *Software Quality Journal*, vol. 4, no. 2, pp. 115-130.

Beyer, H. and Holtzblatt, K. (1997), *Contextual Design : A Customer-Centered Approach to Systems Designs (Morgan Kaufmann Series in Interactive Technologies)*, Morgan Kaufmann.

Bias, R. and Mayhew, D. (1994), *Cost Justifying Usability*, Morgan Kaufman Publishers.

Björk, S., Redström, J., Ljungstr, P. and Holmquist, L. E. (2000), 'PowerView: Using information links and information views to navigate and visualize information on small displays', in *Proceedings of Handheld and Ubiquitous Computing 2000 (HUC2k)*, Springer, Bristol, UK, pp. 46-62.

Blanch, R. and Lecolinet, E. (2007), 'Browsing Zoomable Treemaps: Structure-Aware Multi-Scale Navigation Techniques', *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1248-1253.

Blankenhorn, K. and Jeckle, M. (2004), 'A UML Profile for GUI Layout', in *Proceedings of the 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts and Applications for a Networked World, Net.ObjectDays 2004*, Erfurt, Germany, pp. 110-121.

Blomkvist, S. (2005), 'Towards a model for bridging agile development and user-centered design'. in *Human-centered software engineering – Integrating usability in the development process*, Seffah, A., Gulliksen, J. and Desmarais, M. C. (eds.), Springer, Berlin, pp. 219–244.

Blythe, M. A., Overbeeke, K., Monk, A. F. and C. Wright, P. (2004), *Funology: From Usability to Enjoyment*, Springer.

BMW (2008), *BMW Connected Drive*, viewed 16.10.2008, from http://www.bmw.com/com/de/insights/technology/connecteddrive/open_internet.html.

Bock, C. and Zühlke, D. (2006), 'Model-driven HMI development – Can Meta-CASE tools relieve the pain?', in *Proceedings of the First International Workshop on Metamodelling – Utilization in Software Engineering (MUSE)*, Portugal, pp. 312-319.

Bock, C. (2007a), 'Model-Driven HMI Development: Can Meta-CASE Tools do the Job?', in *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, Waikoloa, USA, p 287b.

Bock, C. (2007b), 'Einsatz formaler Spezifikationen im Entwicklungsprozess von Mensch-Maschine-Schnittstellen ', University of Kaiserslautern

Boehm, B. (1981), *Software Engineering Economics*, Prentice-Hall Inc, New Jersey.

Boehm, B., Gray, T. and Seewaldt, T. (1984), 'Prototyping vs. Specifying: A Multiproject Experiment', *IEEE Transactions on Software Engineering*, pp. 290-320.

Boehm, B. (1988), 'A Spiral Model of Software Development and Enhancement', *IEEE Computer*, vol. 21, no. 5, pp. 61-72.

Boehm, B. (1991), 'Software risk management', *IEEE Software*, vol. 8, no. 1, pp. 32-41.

Borchers, J. (2001), *A Pattern Approach to Interaction Design*, John Wiley & Sons.

Boyarski, D. and Buchanan, R. (1994), 'Computers and communication design: Exploring the rhetoric of HCI', *Interactions*, vol. 1, no. 2, pp. 24-35.

Buering, T. (2007), 'Zoomable User Interfaces on Small Screens - Presentation & Interaction Design for Pen-Operated Mobile Devices ', PhD thesis, University of Konstanz.

Buering, T., Gerken, J. and Reiterer, H. (2008), 'Interaction design for zooming maps on pen-operated devices', *International Journal of Human-Computer Studies*, vol. 66, no. 8, pp. 605-627.

Business Process Management Initiative (2008), *Business Process Modeling Notation* viewed 22.6, from http://bpmi.org/.

Buxton, B. (2003), 'Performance by design: The role of design in software product development', in *Proceedings of the Second International Conference on Usage-Centered Design*, Portsmouth, NH, pp. 26-29

Buxton, B. (2007), *Sketching User Experiences: Getting the Design Right and the Right Design*, Morgan Kaufmann.

Caetano, A., Goulart, N., Fonseca, M. and Jorge, J. (2002), 'JavaSketchIt: Issues in Sketching the Look of User Interfaces', in *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, Palo Alto, USA, pp. 9-14.

Calabria, T. (2004), *An introduction to personas and how to create them*, viewed 22.8.2008, from http://www.steptwo.com.au/papers/kmc_personas/.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. and Vanderdonckt, J. (2003), 'A Unifying Reference Framework for Multi-Target User Interfaces', *Interacting with Computers* vol. 15, no. 3, pp. 289–308.

Campos, J. C. (2004), 'The modelling gap between software engineering and human-computer interaction', in *Proceedings of the ICSE 2004 Workshop on Bridging the Gaps*, Kazman, I. R., Bass, L. and John, B. (eds.), pp. 54-61.

Campos, P. and Nunes, N. J. (2004a), 'CanonSketch: A User-Centered Tool for Canonical Abstract Prototyping', in *EHCI/DS-VIS*, pp. 146-163.

Campos, P. (2005a), 'Task-Driven Tools for Requirements Engineering', in *13th International Requirements Engineering Conference (RE'05), Doctoral Consortium*.

Campos, P. (2005b), 'User-Centered CASE Tools for User-Centered Information Systems', in *12th Conference on Advanced Information Systems Engineering*.

Campos, P. and Nunes, N. J. (2005a), 'Galactic Dimensions: A Unifying Workstyle Model for User-Centered Design', *Lecture Notes in Computer Science*, vol. 3585, pp. 158-169.

Campos, P. and Nunes, N. J. (2005b, 2.5.2005), *TaskSketch*, viewed 6.10.2008, from http://dme.uma.pt/projects/tasksketch/index.html.

Campos, P. and Nunes, N. J. (2006), 'Principles and Practice of Work Style Modeling: Sketching Design Tools'. in *Human Work Interaction Design: Designing for Human Work*, Springer, Boston, pp. 203-219.

Campos, P. and Nunes, N. J. (2007), 'Towards useful and usable interaction design tools: CanonSketch', *Interacting with Computers*, vol. 19, no. 5-6, pp. 597–613.

Campos, P. F. and Nunes, N. J. (2004b), *CanonSketch*, viewed 6.10.2008, from http://dme.uma.pt/projects/canonsketch/index.html.

Card, S., Mackinlay, J. and Shneiderman, B. (1999), *Readings in Information Visualization: Using Vision to Think*, Morgan Kaufmann.

Carroll, J. (1991), *Designing Interaction: Psychology at the Human-Computer Interface (Cambridge Series on Human-Computer Interaction)*, Cambridge University Press.

Carroll, J. (2000a), *Making Use: Scenario-Based Design of Human-Computer Interactions*, The MIT Press.

Carroll, J. M. (1995), 'Introduction: The Scenario Perspective on System Development'. in *Scenario-Based Design: Envisioning Work and Technology in System Development*, New York: John Wiley and Sons, pp. 1-17.

Carroll, J. M. (1997), 'HUMAN-COMPUTER INTERACTION: Psychology as a Science of Design', *Annual Review of Psychology*, vol. 48, no. 1, pp. 61-83.

Carroll, J. M. (2000b), 'Five reasons for scenario-based design', *Interacting with Computers*, vol. 13, no. 1, pp. 43-60.

Carter, J. A., Liu, J., Schneider, K. and Fourney, D. (2005), 'Transforming usability engineering requirments into software engineering specifications: from PUF to UML'. in *Human-centered software engineering – integrating usability in the development process*, Seffah, A., Gulliksen, J. and Desmarais, M. C. (eds.), Springer, Dordrecht, Netherlands, pp. 147-169.

Charlton, G. (2007), *Hidden charges and poor usability deter online shoppers* from http://www.e-consultancy.com/news-blog/362475/hidden-charges-and-poor-usability-deter-online-shoppers.html.

Clark, A. (2001), 'Natural-Born Cyborgs?', in *Proceedings of Cognitive Technology: Instruments of Mind: 4th International Conference.*, Springer, Warwick, UK, pp. 17-24.

Cockburn, A. (2000), *Writing Effective Use Cases*, Addison-Wesley Professional.

Cockburn, A. (2001), *Agile Software Development*, Addison-Wesley.

Cockburn, A. (2002, 23.8.2007), *Use cases, ten years later*, viewed 2008.

Cockburn, A. and McKenzie, B. (2002), 'Evaluating the effectiveness of spatial memory in 2D and 3D physical and virtual environments', in *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, pp. 203-210.

Cockburn, A., Karlson, A. M. Y. and Bederson, B. B. (2006), 'A Review of Focus and Context Interfaces', Department of Computer Science, University of Maryland, HCIL Tech Report 2006-09.

Cockburn, A. (2008, 1.1.2008), *Resources for writing use cases*, viewed 14.9.2008, from http://alistair.cockburn.us/index.php/Resources_for_writing_use_cases.

Cole, M. (1996), *Cultural Psychology: A Once and Future Discipline*, The Belknap Press of Harvard University Press, Cambridge, U.S. and London, UK.

Collins, J. C. and Porras, J. I. (1996), 'Building Your Company's Vision', *Harvard Business Review Article*, vol. Sept-Oct 1996, no. 96501, pp. 64-77.

Constantine, L. and Lockwood, L. (1999a), 'Use cases in task modeling and user interface design', in *CHI '99 extended abstracts on Human factors in computing systems*, ACM, pp. 352-352.

Constantine, L. (2001), 'Back to the future', *Commun. ACM*, vol. 44, no. 3, pp. 126-129.

Constantine, L. and Lockwood, L. (2001), *Structure and Style in Use Cases for User Interface Design*, viewed 15.7.2008, from http://www.foruse.com/Files/Papers/structurestyle2.pdf.

Constantine, L. (2003), 'Canonical Abstract Prototypes for Abstract Visual and Interaction Design', *Interactive Systems. Design, Specification, and Verification*, pp. 1-15.

Constantine, L., Biddle, R. and Noble, J. (2003), 'Usage-Centered Design and Software Engineering: Models for Integration', in *Proceedings, International Conference on Software Engineering 2003*, pp. 3-9.

Constantine, L. and Lockwood, L. (2003), 'Usage-centered software engineering: an agile approach to integrating users, user interfaces, and usability into software engineering practice', in *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, IEEE Computer Society, pp. 746-747.

Constantine, L. (2005), *Users, Roles, and Personas*, from http://www.foruse.com/articles/rolespersonas.pdf.

Constantine, L. and Campos, P. (2005), 'CanonSketch and TaskSketch: innovative modeling tools for usage-centered design', in *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, ACM, pp. 162-163.

Constantine, L. (2007), *Yahoo Agile-Usability Group - Comments on "Don Norman on Agile Development"*, viewed 19.3, from http://groups.yahoo.com/group/agile-usability/message/2021.

Constantine, L. L. (1996), 'Usage-centered software engineering: new models, methods, and metrics', in *Proceedings of the International Conference on Software Engineering: Education and Practice, 1996*, pp. 2-9.

Constantine, L. L. (1998), 'Rapid Abstract Prototyping', *Software Development*, vol. 6, no. 11.

Constantine, L. L. and Lockwood, L. A. D. (1999b), *Software for Use: A Practical Guide to Models and Methods of Usage-Centered Design*, Addison-Wesley.

Constantine, L. L. (2002), 'Process agility and software usability: Toward lightweight usage-centered design', *Information Age*, vol. 8, no. 8.

Constantine, L. L. and Lockwood, L. A. D. (2002), 'Usage-Centered Engineering for Web Applications', *IEEE Software*, vol. 19, no. 2, pp. 42-50

Constantine, L. L. (2004), 'Beyond User-Centered Design and User Experience: Designing for User Performance', *Cutter IT Journal*, vol. 17, no. 2.

Cooper, A. (1995), *About Face: The Essentials of User Interface Design*, IDG Books, Foster City, CA.

Cooper, A. (1999), *The Inmates are Running the Asylum*, Sams.

Cooper, A. (2004), *The Inmates Are Running the Asylum : Why High Tech Products Drive Us Crazy and How to Restore the Sanity (2nd Edition)*, Sams.

Cooper, A., Reimann, R. and Cronin, D. (2007), *About Face 3.0: The Essentials of Interaction Design*, Wiley.

Coyette, A., Vanderdonckt, J. and Limbourg, Q. (2006), 'SketchiXML: A Design Tool for Informal User Interface Rapid Prototyping', in *Proceedings of International Workshop on Rapid Integration of Software Engineering techniques RISE'2006*, Guelfi, N. and Buchs, D. (eds.), Springer, Geneva, pp. 160-176.

Coyette, A., Schimke, S., Vanderdonckt, J. and Vielhauer, C. (2007), 'Trainable Sketch Recognizer for Graphical User Interface Design', in *Proceedings of the International Conference on Human-Computer Interaction (INTERACT 2007)*, Rio de Janeiro, Brazil, pp. 124-135.

D. Fitton, Cheverst, K., Kray, C., Dix, A., Rouncefield, M. and Saslis-Lagoudakis, G. (2005), 'Rapid Prototyping and User-Centered Design of Interactive Display-Based Systems', *IEEE Pervasive Computing*, vol. 4, no. 4, pp. 58-66.

Dijkstra, E. W. (1976), *A Discipline of Programming*, Englewood Cliffs, NJ: Prentice Hall.

DIN EN ISO 8402 (1994), 'Quality Vocabulary'.

DIN EN ISO 9001 (1987), 'Quality systems - Model for quality assurance in design/development, production, installation and servicing'.

DIN EN ISO 9126 (1991), 'Software product evaluation - Quality characteristics and guidelines for their use'.

DIN EN ISO 9241-11 (1998), 'Anforderungen an die Gebrauchstauglichkeit; Leitsätze'.

DIN EN ISO 13407 (1999), 'ISO/IEC 13407: 1999 (E) Human-Centred Design Processes for Interactive Systems'.

Dix, A. (2003), 'Upside down As and algorithms - computational formalisms and theory'. in *HCI Models, Theories, and Frameworks. Toward a Multidisciplinary Science*, Carroll, J. M. (ed.), Elsevier LTD, Oxford, pp. 381-430.

Dix, A., Finlay, J., Abowd, G. and Beale, R. (2003), *Human-Computer Interaction (3rd Edition)*, Prentice Hall.

Dix, A. J. (1987), 'Formal Methods and Interactive Systems: Principles and Practice'.

Dix, A. J. (1995), 'Formal methods an introduction to and overview of the use of formal methods within HCI'. in *Formal Methods' in Perspectives on HCI: Diverse Approaches*, Monk, A. and Gilbert, N. (eds.), Academic Press, London, pp. 9-43.

Donahue, G. M. (2001), 'Usability and The Bottom Line', *IEEE Software*, vol. 18, pp. 31-37.

Elnaffar, S., Graham, N.C. (1999), 'Semi-Automated Linking of User Interface Design Artifacts.', in *Proceedings of the 3rd International Conference on the Computer-Aided Design of User Interfaces (CADUI 1999)*, Kluwer Academic Pub, pp. 127–138.

Excelsoftware (2008), *Gane & Sarson DFD*, viewed 25.9.2008, from http://www.excelsoftware.com/.

Faulkner, X. (2000), *Usability Engineering (Grassroots)*, Palgrave Macmillan.

Florida, R. (2002), *The Rise of the Creative Class and How It's Transforming Work, Leisure, Community and Everyday Life*, Basic Books, New York.

Florida, R. (2005), *The Flight of the Creative Class*, HarperCollings, New York.

Folmer, E. and Bosch, J. (2004), 'Cost Effective Development of Usable Systems: Gaps between HCI and SE', in *ICSE 2004: Proceedings of the Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction*, Scotland.

Folmer, E. and Bosch, J. (2005), 'Cost Effective Development of Usable Systems; Gaps between HCI and Software Architecture Design', in *Proceedings of the conference on Advances in Information Systems Development: Bridging the Gap between Academia and Industry (ISD'2005)*, Karslstad, Sweden.

Forbrig, P. (2001), 'Beziehungen zwischen Aufgabenmodellierung und objektorientierter Softwareentwicklung', in *Proceedings of the Mensch & Computer 2001, Workshop UML und Aufgabenmodellierung*, Bad Honnef, Germany.

Fowler, M. (2003), *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley, Longman, Amsterdam.

Furnas, G. and Bederson, B. (1995), 'Space-scale diagrams: understanding multiscale interfaces', in *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press/Addison-Wesley Publishing Co., pp. 234-241.

Furnas, G. and Zhang, X. (1998), 'MuSE: a multiscale editor', in *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, ACM, pp. 107-116.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995), *Design Patterns*, Addison-Wesley Professional.

Gane, C. P. and Sarson, T. (1979), *Structured Systems Analysis: Tools and Techniques*, Prentice Hall Professional Technical Reference Englewood Cliffs.

Garrett, J. J. (2000), *The Elements of User Experience*, viewed 16.8.2008, from http://www.jjg.net/elements/pdf/elements.pdf.

Garrett, J. J. (2002), *The Elements of User Experience: User-Centered Design for the Web*, New Riders Press.

Garvin, D. A. (1984), 'What does "product quality" really mean?', *Sloane Management Review*, vol. 26, no. 1, pp. 25-43.

Gellner, M. and Forbrig, P. (2003), 'Extreme Evaluations – Lightweight Evaluations for Software Developers', in *Proceedings of IFIP Working Group 2.7/13.4 INTERACT 2003 Workshop on Bridging the Gap Between Software Engineering and Human-Computer Interaction*, Harning, M. B. and Vanderdonckt, J. (eds.), Zürich, Switzerland, pp. 75-80.

Geyer, F. (2008), 'A Zoom-Based Specification Tool supporting Interdisciplinary User Interface Design Processes', Master thesis, University of Konstanz.

Ghosh, G. (2004), *Agile, multidisciplinary teamwork*, viewed 26.8.2008, from http://www.userminded.no/archives/Agile_Multidisciplinary_Teamwork.pdf.

Gold, K. (2007), *Tackling the Shopping Cart Abandonment Rate* from www.searchmarketingstandard.com/articles/2007/05/tackling-the-shopping-cart-abandonment-rate.html.

Good, L. E. (2003), 'Zoomable User Interfaces for the Authoring and Delivery of Slide Presentation', PhD thesis, University of Maryland.

Good, M., Spine, T. M., Whiteside, J. and George, P. (1986), 'User-derived impact analysis as a tool for usability engineering', in *Proceedings of Human Factors in Computing Systems (CHI 1986)*, Mantei, M. and Oberton, P. (eds.), New York, ACM, pp. 241-246.

Gotel, O. C. Z. and Finkelstein, A. C. W. (1994), 'An Analysis of the Requirements Traceability Problem', in *Proceedings of ICRE'94,*, IEEE Computer Society Press, Los Alamitos, pp. 94-101.

Gottesdiener, E. (2002), *Requirements by Collaboration: Workshops for Defining Needs*, Addison-Wesley Longman, Amsterdam.

Gray, W. D., John, B. E. and Atwood, M. E. (1993), 'Project Ernestine: Validating a GOMS analysis for predicting and explaining real-world performance', *Human-Computer Interaction*, vol. 8, no. 3, pp. 237-309.

Green, M. (1986), 'A survey of three dialogue models', *Transactions on Graphics*, vol. 5, no. 3, pp. 244-275.

Gross, M. D. and Yi-Luen Do, E. (1996), 'Ambiguous Intentions: A Paper-LikeInterface for Creative Design', in *Proceedings of UIST'96*, pp. 183-192.

Grudin, J. and Pruitt, J. (2002), 'Personas, Participatory Design and Product Development: An Infrastructure for Engagement', in *Proceedings of Participation and Design Conference (PDC2002)*, Sweden pp. 144-161.

Grün, C., Gerken, J., Jetter, H. C., König, W. and Reiterer, H. (2005), 'MedioVis - a User-Centred Library Metadata Browser', in *Proceedings of the 9th European Conference on Research and Advanced Technology for Digital Libraries*, Springer, Wien, Austria, pp. 174-185.

Gruhn, V. and Wellen, U. (1998), 'From business process models to distributed software architecture', in *Proceedings of the 3rd international workshop on software architecture*, New York, NY, USA, pp. 53-56.

Gunaratne, J., Hwong, B., Nelson, C. and Rudorfer, A. (2004), 'Using Evolutionary Prototypes to Formalize Product Requirements', in *Proceedings of ICSE 2004 Bridging the Gaps Between Software Engineering and HCI*, Edinburgh, Scotland, pp. 17-20.

Gundelsweiler, F., Memmel, T. and Harald, R. (2004), 'Agile Usability Engineering ', in *Proceedings of the 4th Mensch & Computer Conference (MCI 2007)*, Keil-Slawik, R., Selke, H. and Szwillus, G. (eds.), Oldenbourg Verlag, München, Paderborn, Germany, pp. 33-42.

Gundelsweiler, F., Memmel, T. and Reiterer, H. (2007a), 'ZEUS Zoomable Explorative User Interface for Searching and Object Presentation ', in *Proceedings of the 12th International Conference on Human-Computer Interaction, Human Interface and the Management of Information. Methods, Techniques and Tools in Information Design*, Smith, M. J. and Salvendy, G. (eds.), Springer, Berlin/Heidelberg, Beijing, China, pp. 288-297.

Gundelsweiler, F., Memmel, T. and Reiterer, H. (2007b), 'ZUI Konzepte für Navigation und Suche in komplexen Informationsräumen ', *i-com - Zeitschrift für interaktive und kooperative Medien*, vol. 6, no. 01/2007, pp. 38-48.

Gutierrez, O. (1989), 'Prototyping techniques for different problem contexts', in *Proceedings of the SIGCHI conference on Human factors in computing systems: Wings for the mind*, ACM press, pp. 259 - 264.

Gutwin, C. and Fedak, C. (2004), 'Interacting with Big Interfaces on Small Screens: A Comparison of Fisheye, Zoom, and Panning Techniques', in *Proceedings of Graphics Interface 2004*, pp. 145-152.

Hackmann, J. (2007, 12.3.2007), *Jedes fünfte Projekt ist ein Totalausfall* viewed 2.6.2008, from http://www.computerwoche.de/index.cfm?pid=255&pk=589879.

Harel, D. (1987), 'Statecharts: A Visual Formalism for Complex Systems', *Science of. Computer Programming*, vol. 8, no. 3, pp. 231-274.

Hasdogan, G. (1996), 'The Role of User Models in Product Design for Assessment of User Needs', *Design Studies*, vol. 17, no. 1, pp. 19-33.

Hassenzahl, M., Platz, A., Burmester, M. and Lehner, K. (2000), 'Hedonic and Ergonomic Quality Aspects Determine a Software's Appeal.', in *In: Proceedings of the CHI 2000 Conference on Human Factors in Computing*, The Hague, NL, pp. 201-208.

Heim, S. (2007), *The Resonant Interface: HCI Foundations for Interaction Design*, Addison Wesley.

Hewett, T. T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G. and Verplank, W. (1992, 11.4.2008), *Curricula for Human-Computer Interaction*, from http://www.sigchi.org/cdg/index.html.

Hix, D. and Hartson, H. R. (1993), *Developing User Interfaces: Ensuring Usability Through Product and Process*, John Wiley and Sons.

Hoffmann, M., Kühn, N., Weber, M. and Bittner, M. (2004), 'Requirements for Requirements Management Tools', in *Proceedings of the 13th IEEE International Conference on Requirements Engineering, 2004*, Berlin, Germany, pp. 301-308.

Holt, J. (2005), *A pragmatic guide to business process modelling*, The British Computer Society, Swindon, UK.

Holtzblatt, K. (2002), *Personas and contextual design*, viewed 3.7.2008, from http://www.incent.com/community/design_corner/02_0913.html.

Holzinger, A. (2004), 'Rapid Prototyping for a Virtual Medical Campus Interface', *IEEE Software*, vol. 21, no. 1, pp. 92-99.

Holzinger, A. and Slany, W. (2006), 'XP + UE = XU. Praktische Erfahrungen mit eXtreme Usability', *Informatik Spektrum*, vol. 29, no. 1.

Hornbæk, K., Bederson, B. and Plaisant, C. (2002), 'Navigation patterns and usability of zoomable user interfaces with and without an overview', *ACM Transactions on Human-Computer Interaction*, vol. 9, no. 4, pp. 362-389.

Horrocks, I. (1999), *Constructing the user interface with statecharts*, Addison-Wesley, Harlow.

Hudson, W. (2000), 'The whiteboard: metaphor: a double-edged sword', *Interactions*, vol. 7, no. 3, pp. 11-15.

Hudson, W. (2003), 'Adopting User-Centered Design Within an Agile Process: A Conversation', *Cutter IT Journal*, vol. October 2003.

Hussey, A. (1996), 'Object-oriented specification and design of user-interfaces', in *Proceedings of the 6th Australian Conference on Computer-Human Interaction 1996*, pp. 336-337.

IDS Scheer AG (2008a), *BPMN allocation diagram in ARIS Business Architect*, viewed 21.10.2008, from http://www.ids-scheer.com/en/ARIS/Modeling_Standards/BPMN/79746.html.

IDS Scheer AG (2008b), *ARIS UML Designer*, viewed 16.10.2008, from http://www.ids-scheer.de/de/ARIS/ARIS_Software/ARIS_UML_Designer/105142.html.

iRise (2008), 'iRise - Visualize your business', Corporate Document / Marketing Presentation.

Isazadeh, H. and Lamb, D. A. (2006), *CASE Environments and MetaCASE Tools. Technical Report No. 1997-403*, viewed 25.10, from http://www.cs.queensu.ca/TechReports/reports1997.html.

Jacobson, I. (1992), *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley Professional.

Jarke, M. (1999), 'Scenarios for modeling', *Communcations of the ACM*, vol. 42, no. 1, pp. 47-48.

Jarzabek, S. and Huang, R. (1998), 'The case for user-centered CASE tools', *Communications of the ACM*, vol. 41, no. 8, pp. 93-99.

Jeffries, R. (2001, 30.8.2001), *Essential XP: Card, Conversation, Confirmation* viewed 20.8.20008, from http://www.xprogramming.com/xpmag/expCardConversationConfirmation.htm.

Jetter, H.-C., Engl, A., Schubert, S. and Reiterer, H. (2008a), 'Zooming not Zapping: Demonstrating the ZOIL User Interface Paradigm for ITV Applications ', in *Adjunct Proceedings of European Interactive TV Conference*, July 3-4, Salzburg, Austria.

Jetter, H.-C., König, W. A., Gerken, J. and Reiterer, H. (2008b), 'ZOIL - A Cross-Platform User Interface Paradigm for Personal Information Management', in *Proceedings of Personal Information Management 2008: The disappearing desktop (a CHI 2008 Workshop)*, April 5-6, Florence, Italy.

Jetter, H. (2007), 'Informationsarchitektur und Informationsvisualisierung für die Post-WIMP Ära', Master thesis, University of Konstanz.

Johnson, S. (2003), 'Buyer and User Personas', *ProductMarketing.com*, vol. 1, no. 4, pp. 8-9.

Jones, S. and Sapsford, J. (1998), 'The role of informal representations in early design', in *Proceedings of the DSV-IS'98*, pp. 117-133.

Jones, T. O. and Sasser, W. E. (1995), 'Why satisfied customers defect', *Harvard Business Review*, vol. 6, no. 1, pp. 88-99.

Jordan, P. W. (2000), *Designing Pleasurable Products: An Introduction to the New Human Factors*, Taylor & Francis, London, New York.

Jose, J. (2003), 'HCI Designers and Engineers: It is possible to work together?', in *INTERACT 2003 Workshop on Closing the Gaps: Software Engineering and Human-Computer Interaction*, Harning, M. B. and Vanderdonckt, J. (eds.), Université catholique de Louvain, Institut d'Administration et de Gestion (IAG) on behalf of the International Federation for Information Processing (IFIP), pp. 14-19.

Jul, S. and Furnas, G. (1998), 'Critical zones in desert fog: aids to multiscale navigation', in *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, ACM, pp. 97-106.

Kaptelinin, V. (1995), 'A comparison of four navigation techniques in a 2D browsing task', in *CHI '95: Conference companion on Human factors in computing systems*, ACM Press, New York, NY, USA, pp. 282-283.

Karat, C. (1993a), 'Cost-benefit and business case analysis of usability engineering', in *Proceedings of the SIGCHI'93*, Amsterdam.

Karat, C. (1993b), 'Usability Engineering in dollars and cents', *IEEE Software*, vol. 10, no. 3, pp. 88-89.

Karat, C. (1997), 'Cost-justifying usability engineering in the software life cycle'. in *Handbook of Human-Computer Interaction*, Helander, M., Landauer, T. and Prabhu, P. (eds.), Elsevier Science, Amsterdam.

Karat, J. and Bennett, J. (1990), 'Supporting effective and efficient design meetings', in *INTERACT '90: Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction*, North-Holland Publishing Co., pp. 365-370.

Karat, J. and Bennett, J. (1991), 'Using scenarios in design meetings - a case study example', pp. 63-94.

Kazman, R., Bass, L. and Bosch, J. (2003), 'Workshop overviews: Bridging the gaps between software engineering and human-computer interaction', in *Proceedings of 25th international conference on software engineering*, IEEE Computer Society, pp. 777-778

Klinkhammer, D., Memmel, T., Gundelsweiler, F. and Reiterer, H. (2007), 'Interaktionskonzepte und Visualisierungen zum Online-Fahrzeugvergleich', *Zeitschrift für Automobilwirtschaft (ZfAw)*, vol. 3/07, pp. 62-71.

König, D. (2008), 'Anforderungen an Requirements Engineering Werkzeuge für die visuelle Spezifikation', Master thesis, University of Konstanz.

König, W., Bieg, H., Schmidt, T. and Reiterer, H. (2007), 'Position-independent interaction for large high-resolution displays', in *IHCI'07: Proceedings of IADIS International Conference on Interfaces and Human Computer Interaction 2007*, IADIS Press, pp. 117-125.

Kurosu, M. and Kashimura, K. (1995), 'Apparent Usability vs. Inherent Usability. Experimental Analysis on the Determinants of Apparent Usability', in *In Proceeedings of CHI 1995*, pp. 292-293.

Landauer, T. K. (1995), *The trouble with computers*, MIT Press.

Landay, J. and Myers, B. (1995a), 'Interactive sketching for the early stages of user interface design', in *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, pp. 43-50.

Landay, J. and Myers, B. (1995b, 27.11.1995), *Just draw it! Programming by sketching storyboards*, viewed 9.10.2008, from http://www.cs.cmu.edu/~landay/research/publications/storyboard-tr/storyboard.html.

Landay, J. (1996), 'SILK: sketching interfaces like krazy', in *CHI '96: Conference companion on Human factors in computing systems*, ACM Press, pp. 398-399.

Landay, J. A. and Myers, B. A. (2001), 'Sketching interfaces: toward more human interface design', *Computer*, vol. 34, no. 3, pp. 56-64.

Lederer, A. L. and Prasad, J. (1992), 'Nine Management Guidelines for Better Cost Estimating', *Communications of the ACM*, vol. 35, no. 2, pp. 51–59.

Lepreux, S., Vanderdonckt, J. and Michotte, B. (2007), 'Visual Design of User Interfaces by (De)composition'. in *Interactive Systems. Design, Specification, and Verification*, pp. 157-170.

Lichter, H., Schneider-Hufschmidt, M. and Züllighoven, H. (1993), 'Prototyping in industrial software projects-bridging the gap between theory and practice', in *Proceedings of the 15th International Conference on Software Engineering*, IEEE Computer Society, Baltimore, MD., pp. 221-229.

Lieberman, B. (2004, 29.4.2004), *UML Activity Diagrams: Detailing User Interface Navigation*, viewed 28.9.2008, from http://www.ibm.com/developerworks/rational/library/4697.html.

Lim, Y.-K., Stolterman, E. and Tenenberg, J. (2008), 'The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas', *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 15, no. 2, pp. 1-27.

Lin, J. (1999), 'A visual language for a sketch-based UI prototyping tool', in *CHI '99 extended abstracts on Human factors in computing systems*, ACM, pp. 298-299.

Lin, J., Newman, M., Hong, J. and Landay, J. (2000), 'DENIM: finding a tighter fit between tools and practice for Web site design', in *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, pp. 510-517.

Lin, J., Newman, M., Hong, J. and Landay, J. (2001), 'DENIM: an informal tool for early stage web site design', in *CHI '01: CHI '01 extended abstracts on Human factors in computing systems*, ACM Press, pp. 205-206.

Lin, J., Thomsen, M. and Landay, J. A. (2002), 'A Visual Language for Sketching Large and Complex Interactive Designs', *CHI Letters: Proceedings of Human Factors in Computing Systems: CHI 2002*, vol. 4, no. 1, pp. 307-314.

Lin, J. (2003a), 'Damask: A Tool for Early-Stage Design and Prototyping of Cross-Device User Interfaces', in *In Conference Supplement of UIST 2003: ACM Symposium on User Interface Software and Technology*, Vancouver, BC, Canada, pp. 13-16.

Lin, J. (2003b), 'Damask: A Tool for Early-Stage Design and Prototyping of Cross-Device User Interfaces', *CHI 2003 workshop on HCI Patterns: Concepts and Tools, Fort Lauderdale, Florida*.

Löwgren, J. and Stolterman, E. (2004), *Thoughtful Interaction Design: A Design Perspective on Information Technology*, The MIT Press.

Löwgren, J. (2008, 10.6.2008), *Interaction Design*, viewed 14.10.2008, from http://www.interaction-design.org/encyclopedia/interaction_design.html.

MacIntyre, F., Estep, K. W. and Sieburth, J. M. (1990), 'Cost of user-friendly programming', *Journal of Forth Application and Research*, vol. 6, no. 2, pp. 103-115.

Malhotra, Y. (1998), 'Business Process Redesign: An Overview', *IEEE Engineering Management Review*, vol. 25, no. 3.

Manning, H., McCarthy, J. C. and Souz, R. K. (1998), 'Why Most Web Sites Fail', *Interactive Technology Series (Forrester Research)*, vol. 3, no. 7.

Marcus, A. (2002), 'Return on Investment for Usable User-Interface Design: Examples and Statistics', *User Experience Magazine* vol. 1, no. 3, pp. 25-31.

Marion, C. (1999), *What is Interaction Design and What Does It Mean to Information Designers?*, viewed 20.6, from http://mysite.verizon.net/resnx4g7/PCD/WhatIsInteractionDesign.html.

Markopoulos, P. and Marijnissen, P. (2000), 'UML as a representation for Interaction Design', in *Proceedings of the OZCHI 2000*, pp. 240–249.

Mayhew, D. J. (1999), *The Usability Engineering Lifecycle: A Practioner's Handbook for User Interface Design*, Morgan Kaufman Publishers.

McCurdy, M., Connors, C., Pyrzak, G., Kanefsky, B. and Vera, A. (2006), 'Breaking the Fidelity Barrier: An Examination of our Current Characterization of Prototypes and an Example of a Mixed-Fidelity Success', in *Proceedings of CHI'06*, ACM Press, New York, pp. 1233-1242.

Memmel, T. (2005), 'An innovative navigation concept for complex information spaces exemplified by DaimlerChrysler's digital sales channel', Master thesis, University of Konstanz.

Memmel, T., Bock, C. and Reiterer, H. (2007a), 'Model-driven prototyping for corporate software specification ', in *Proceedings of the 1st Conference on Engineering Interactive Systems 2007 (EHCI-HCSE-DSVIS'07)*, Springer, Salamanca, Spain, pp. 158-174

Memmel, T., Geyer, F., Rinn, J. and Reiterer, H. (2007b), 'Interdisciplinary Visual User Interface Specification', University of Konstanz, Dept. Computer Science, Technical Report.

Memmel, T., Gundelsweiler, F. and Reiterer, H. (2007c), 'Prototyping Corporate User Interfaces - Towards A Visual Specification Of Interactive Systems ', in *Proceedings of the 2nd IASTED International Conference on Human Computer Interaction (IASTED-HCI '07)*, Acta Press, Canada, Chamonix, France, pp. 177-182.

Memmel, T., Gundelsweiler, F. and Reiterer, H. (2007d), 'CRUISER: a Cross-Discipline User Interface & Software Engineering Lifecycle ', in *Proceedings of the 12th International Conference on Human-Computer Interaction (HCII 2007), Human-Computer Interaction - Interaction Design and Usability (Part I)*, Jacko, J. (ed.), Springer, Beijing, China, pp. 174–183.

Memmel, T., Gundelsweiler, F. and Reiterer, H. (2007e), 'Agile Human-Centered Software Engineering', in *Proceedings oft the 21st BCS HCI Group conference, HCI...but not as we know it (HCI 2007)*, Ball, L. J., Sasse, M. A., Sas, C., Ormerod, T. C., Dix, A., Bagnall, P. and Ewan, T. M. (eds.), British Computer Society, Lancaster, UK, pp. 167-175.

Memmel, T., Reiterer, H. and Holzinger, A. (2007f), 'Agile Methods and Visual Specification in Software Development: a chance to ensure Universal Access ', in *Proceedings of the 12th International Conference on Human-Computer Interaction (HCII 2007), Universal Access in Human-Computer Interaction - Coping with Diversity (Part I)*, Stephanidis, C. (ed.), Springer, Beijing, China, pp. 453–462.

Memmel, T., Reiterer, H., Ziegler, H. and Oed, R. (2007g), 'Visuelle Spezifikation zur Stärkung der Auftraggeberkompetenz bei der Gestaltung interaktiver Systeme ', in *Proceedings of 5th Workshop of the German Chapter of the Usability Professionals Association (GC-UPA)*, Roese, K. and Brau, H. (eds.), Frauenhofer IRB Verlag, Stuttgart, pp. 99-104.

Memmel, T., Brau, H. and Zimmermann, D. (2008a), 'Agile nutzerzentrierte Softwareentwicklung mit leichtgewichtigen Usability Methoden – Mythos oder strategischer Erfolgsfaktor?', in *Usability Professionals 2008*, Brau, H., Diefenbach, S., Hassenzahl, M., Koller, F., Peissner, M. and Röse, K. (eds.), Fraunhofer IRB Verlag: Stuttgart, Lübeck, Germany, pp. 223-227.

Memmel, T., Geis, T. and Reiterer, H. (2008b), 'Methoden, Notationen und Werkzeuge zur Übersetzung von Anforderungen in User Interface Spezifikationen', in *Usability Professionals 2008*, Brau, H., Diefenbach, S., Hassenzahl, M., Koller, F., Peissner, M. and Röse, K. (eds.), Fraunhofer IRB Verlag: Stuttgart, Lübeck, Germany, pp. 45-48.

Memmel, T., Geyer, F., Rinn, J. and Reiterer, H. (2008c), 'A Zoom-Based Specification Tool for Corporate User Interface Development ', in *Proceedings of the IADIS International Conference on Interfaces and Human Computer Interaction (IHCI 2008)*, Amsterdam, The Netherlands, pp. 368-370.

Memmel, T., Geyer, F., Rinn, J. and Reiterer, H. (2008d), 'Tool-Support for Interdisciplinary and Collaborative User Interface Specification', in *Proceedings of the IADIS International Conference on Interfaces and Human Computer Interaction (IHCI 2008)*, Amsterdam, The Netherlands, pp. 51-60.

Memmel, T. and Reiterer, H. (2008), 'Inspector: Interactive UI Specification Tool ', in *Proceedings of the 7th International Conference On Computer Aided Design of User Interfaces (CADUI 2008)*, Springer, Albacete, Spain, pp. 161-174.

Memmel, T., Reiterer, H., Ziegler, H. and Oed, R. (2008e), 'User Interface Specification In Complex Web-Based Information Spaces ', in *Proceedings of the 3rd IASTED International Conference on Human Computer Interaction (IASTED-HCI '08)*, Cunliffe, D. (ed.), Acta Press, Canada Innsbruck, Austria, pp. 180-185.

Memmel, T., Vanderdonckt, J. and Reiterer, H. (2008f), 'Multi-fidelity User Interface Specifications', in *Proceedings of the 15th International Workshop on the Design, Verification and Specification of Interactive Systems (DSV-IS 2008)* Springer, Kingston, Canada, pp. 43-57.

Metzker, E. (2005), 'Adoption-Centric Usability Engineering: Systematic Deployment, Evaluation and Improvement of Usability Engineering Methods in the Software Engineering Lifecycle', PhD thesis, University of Ulm.

Metzker, E., Reiterer, H. (2002), 'Evidence-Based Usability Engineering', in *Proceedings of the 4th Int. Conf. on Computer-Aided Design of UIs (CADUI 2002)*, Kluwer Academic Publishers, pp. 323-336.

Michotte, B. and Vanderdonckt, J. (2008), 'GrafiXML, a Multi-target User Interface Builder Based on UsiXML', in *Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems (ICAS 2008)*, IEEE Computer Society Press, Los Alamitos, pp. 15-22.

Microsoft Research (2008), *Piccolo2D*, viewed 27.10.2008, from http://www.piccolo2d.org/.

Mikkelson, N. and Lee, W. O. (2000), 'Incorporating user archetypes into scenario-based design', in *Proceedings of the UPA 2000*.

Miller, G. and Williams, L. (2006), 'Personas: Moving Beyond Role-Based Requirements Engineering ', Technical Reports, NC State University, TR-2006-24.

Moran, T. and Carroll, J. (1996), *Design Rationale: Concepts, Techniques, and Use (Computers, Cognition, and Work)*, CRC.

Mori, G., Paterno, F. and Santoro, C. (2002), 'CTTE: Support for Developing and Analyzing Task Models for Interactive System Design', *IEEE Transactions on software engineering*, vol. 28, no. 8, pp. 797-813.

Mrdalj, S. and Jovanovic, V. (2002), 'User Interface Driven System Design', *Issues in Information Systems (IIS)*, vol. 3.

Myers, B. and Rosson, M. (1992), 'Survey on user interface programming', in *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, pp. 195-202.

Mynatt, E. D. (1999), 'The Writing on the Wall', in *Proceeding of the 7th IFIP Conf. on Human-Computer Interaction*, Sasse, M. A. and Johnson, C. (eds.), IOS Press, Edinburgh, UK, pp. 196-204.

Newman, N. W., Jason, J.L., Hong, I., Landay, J.A. (2003), 'DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice', *Human-Computer Interaction*, vol. 18, no. 3, pp. 259-324.

Nielsen, J. (1993), *Usability Engineering*, Academic Press, Inc., Boston, MA.

Nielsen, J. (1994), *Usability Engineering*, Morgan Kaufmann.

Nielsen, J. (1998, 1998), *Failure of Corporate Websites*, viewed 2.6, from http://www.useit.com/alertbox/981018.html.

Nielsen, J. (2002), *Customers as Designers*, viewed 2.6., from http://www.useit.com/alertbox/20000611.html.

Nielsen, J. (2004a, 15.3.2004), *Why Consumer Products Have Inferior User Experience*, viewed 8.6., from http://www.useit.com/alertbox/20040315.html.

Nielsen, L. (2004b), 'Engaging Personas and Narrative Scenarios', PhD thesis, Department of Informatics, Copenhagen Business School.

Nielsen Norman Group (2008), *Paper Prototyping: Stills from the Video*, viewed 10.8.2008, from http://www.nngroup.com/reports/prototyping/video_stills.html.

NIST (2002), 'NIST Planning Report 02-3: The Economic Impacts of Inadequate Infrastructure for Software Testing'.

Nóbrega, L., Nunes, N. J. and Coelho, H. (2007), 'The Meta Sketch Editor - A Reflexive Modeling Editor '. in *Computer-Aided Design Of User Interfaces V*, Calvary, G., Pribeanu, C., Santucci, G. and Vanderdonckt, J. (eds.), Springer, pp. 201-214.

Nóbrega, L. (2008, 24.10.2008), *MetaSketch Editor*, viewed 27.10.2008, from http://www.labuse.org/metasketch/.

Norman, D. and Draper, S. (1986), *User Centered System Design: New Perspectives on Human-computer Interaction*, L. Erlbaum Associates Inc, Hillsdale, NJ, USA.

Norman, D. (2002), *The Design of Everyday Things*, Basic Books.

Norman, D. (2005), 'Human-Centered Design Considered Harmful', *Interactions*, vol. 12, no. 4, pp. 14-19.

Norman, D. A. (2004), *Emotional Design: Why we love (or hate) everyday things*, Basic Books, New York.

Nunes, N. J. and Cunha, J. F. e. (2000), 'Wisdom - A UML-based architecture for interactive systems', in *Proceedings of the 7th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS2000)*, Paterno, F., Palanque, P. (ed.), Springer, New Tork, U.S., pp. 191-205.

Nunes, N. J. (2001), 'Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach', PhD thesis, Universidade da Madeira.

Nunes, N. J. and Cunha, J. F. e. (2001), 'Wisdom-Whitewater interactive system development with object models'. in *Object modeling and user interface design: designing interactive systems*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, pp. 197 - 243

Oberquelle, H. (1984), 'On Models and Modelling in Human-Computer Cooperation.', in *Proceedings of the 2nd European Conference on Readings on Cognitive Ergonomics - Mind and Computers*, Springer, London, UK, pp. 26-43

OMG (2007, 11.9.2007), *Introduction to OMG's Unified Modeling Language*, viewed 21.6, from http://www.omg.org/gettingstarted/what_is_uml.htm.

Österle, H. (1994), *Business Engineering - Prozess- und Systementwicklung, Band 1: Entwurfstechniken*, Springer, Heidelberg.

Paterno, F., Mancini, C. and Meniconi, S. (1997), 'ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models', in *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction (Interact 1997)*, Chapman & Hall, Sydney, Australia, pp. 362-369.

Paternò, F., Santoro, C. and Tahmassebi, S. (1998), 'Formal Models for Cooperative Tasks: Concepts and an Application for En-Route Air Traffic Control', in *Proceedings of the DSV-IS'98*, Springer Verlag, Abingdon, pp. 71-86.

Paternò, F. (2000a), *Model-Based Design and Evaluation of Interactive Applications (Tutorial held at HCI2000, Pisa, Italy)*, viewed 16.10.2008, from http://giove.cnuce.cnr.it/~fabio/mbde.html.

Paternò, F. (2000b), *Model-Based Design and Evaluation of Interactive Applications*, Springer.

Paternò, F. (2008a, 19.12.2006), *ConcurTaskTreesEnvironment - CTTE*, viewed 16.10.2008, from http://giove.cnuce.cnr.it/ctte.html.

Paternò, F. (2008b), *ConcurTaskTrees*, viewed 16.10.2008, from http://giove.cnuce.cnr.it/concurtasktrees.html.

Perlin, K. and Fox, D. (1993), 'Pad: an alternative approach to the computer interface', in *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM Press, pp. 57-64.

Petrie, J. N., Schneider, K.A. (2006), 'Mixed-fidelity Prototyping of User Interfaces', in *Proceedings of the DSV-IS'2006*, Springer, pp. 199-212.

Plumlee, M. and Ware, C. (2006), 'Zooming versus multiple window interfaces: Cognitive costs of visual comparisons', *ACM Trans. Comput.-Hum. Interact.*, vol. 13, no. 2, pp. 179-209.

Pook, S., Lecolinet, E., Vaysseix, G. and Barillot, E. (2000), 'Context and interaction in zoomable user interfaces', in *AVI '00: Proceedings of the working conference on Advanced visual interfaces*, ACM, pp. 227-231.

Pook, S. (2001), 'Interaction and Context in Zoomable User Interfaces', PhD thesis, École Nationale Supérieure des Télécommunications.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S. and Carey, T. (1994), *Human-Computer Interaction: Concepts And Design (ICS)*, Addison Wesley.

Preece, J., Rogers, Y. and Sharp, H. (2002), *Interaction Design: Beyond Human-Computer Interaction*, John Wiley & Sons.

Preim, B. (1999), *Entwicklung interaktiver Systeme: Grundlagen, Fallbeispiele und innovative Anwendungsfelder (Springer-Lehrbuch)*, Springer.

Pressman, R. S. (1992), *Software Engineering: A Practitioner's Approach*, McGraw Hill New York.

Pruitt, J. and Grudin, J. (2003), 'Personas: Practice and Theory', in *Proceedings of the Conference on Designing for User Experiences*, San Franciso, CA, U.S., pp. 1-15.

Pruitt, J. and Adlin, T. T. (2006), *The Persona Lifecycle : Keeping People in Mind Throughout Product Design*, Morgan Kaufmann.

Puerta, A. R. (1997), 'A model-based interface development environment', *IEEE Computer*, vol. 14, no. 4, pp. 41-47.

Pyla, P. S., Pérez-Quiñones, M. A., Arthur, J. D. and Hartson, H. R. (2003), 'Towards a Model-Based Framework for Integrating Usability and Software Engineering Life Cycles', in *Proceedings of the IFIP Working Group 2.7/13.4 INTERACT 2003 Workshop on Bridging the Gap Between Software Engineering and Human-Computer Interaction*, pp. 67-74.

QuickMBA.com (2008), *The Business Vision and Company Mission Statemen*, viewed 23.8.2008, from http://www.quickmba.com/strategy/vision/.

Rashid, A., Meder, D., Wiesenberger, J. and Behm, A. (2006), 'Visual requirements specification in end-user participation', in *Proceedings of the first international workshop on Multimedia Requirements Engineering (MERE)*, IEEE Computer Society, p 6.

Rational Software Corporation (2008), *Guidelines - Business Vision*, viewed 23.8.2008, from http://www.ts.mah.se/RUP/RationalUnifiedProcess/process/modguide/md_bvsio.htm.

Rauterberg, M., Strohm, O. and Kirsch, C. (1995), 'Benefits of user-oriented software development based on an iterative cyclic process model for simultaneous engineering', *International Journal of Industrial Ergonomics*, vol. 16, no. 4-6, pp. 391-410.

Reeps, I. E. (2006), *Joy-of-Use: Ästhetik, Emotion und User Experience für interaktive Produkte*, Vdm Verlag Dr. Müller.

Reiterer, H. (2000), 'Tools for Working with Guidelines in Different Interface Design Approaches', in *Annual Workshop of the Special Interest Group on Tools for Working with Guidelines*, Vanderdonckt, J. and Farenc, C. (eds.), Springer, Biarritz, France, pp. 225-236.

Richter, M. and Flückiger, M. (2007), *Usability Engineering Kompakt - Benutzbare Software gezielt entwicklen*, Elsevier GmbH, München.

Robertson, S. and Robertson, J. C. (1999), *Mastering the Requirements Process*, Addison-Wesley Professional.

Rolland, C., Grosz, G. and Kla, R. (1999), 'Experience With Goal-Scenario Coupling in Requirements Engineering', in *Proceedings of the 4th IEEE International Symposium on Requirements Engineering*, IEEE Computer Society, Limerick, Ireland, pp. 74-81.

Rosson, M. and Carroll, J. (2002), *Usability Engineering: Scenario-Based Development of Human Computer Interaction (Interactive Technologies)*, Morgan Kaufmann.

Royce, W. (1970), 'Managing the Development of Large Software Systems', in *Proceedings of IEEE WESCON 26*, pp. 1-9.

Rudd, J., Stern, K. and Isensee, S. (1996), 'Low vs. high fidelity prototyping debate', *Interactions, ACM Press*, vol. 3, no. 1, pp. 76-85.

Rudin-Brown, C. (2005), *Strategies for Reducing Driver Distraction from In-Vehicle Telematics Devices: Report on Industry and Public Consultations*, viewed May 31st, from www.tc.gc.ca/roadsafety/tp/tp14409/menu.htm

Scheer, A.-W. (1998), *ARIS - Modellierungsmethoden, Metamodelle, Anwendungen*, Springer, Berlin-Heidelberg.

Schindler, E. (2008, 11.1.2008), *Success Factors for Corporate Intranets*, viewed 2.6.2008, from http://www.cio.com/article/171400/Success_Factors_for_Corporate_Intranets.

Schneider, K. (1996), 'Prototypes as assets, not toys: why and how to extract knowledge from prototypes', in *Proceedings of the 18th international conference on Software engineering*, IEEE Computer Society Washington, DC, USA, pp. 522-531.

Schön, D. A. (1987), *Educating the Reflective Practitioner: Toward a New Design for Teaching and Learning in the Professions*, Jossey-Bass, San Francisco, CA.

Schrage, M. (1999), *Serious Play: How the World's Best Companies Simulate to Innovate* Harvard Business School Press.

Sefelin, R., Tscheligi, M. and Giller, V. (2003), 'Paper prototyping-what is it good for?: a comparison of paper-and computer-based low-fidelity prototyping', in *Proceedings of the Conference on Human Factors in Computing Systems (CHI 2003)*, ACM Press New York, NY, USA, pp. 778-779.

Seffah, A. and Metzker, E. (2004), 'The obstacles and myths of usability and software engineering', *Commun. ACM*, vol. 47, no. 12, pp. 71-76.

Seffah, A., Desmarais, M. C. and E., M. (2005a), 'HCI, usability and software engineering integration: present and future'. in *Human-centered software engineering – integrating usability in the development process*, Seffah, A., Gulliksen, J. and Desmarais, M. C. (eds.), Springer, Dordrecht, Netherlands, pp. 35-57.

Seffah, A., Gulliksen, J. and Desmarais, M. C. (2005b), *Human-Centered Software Engineering - Integrating Usability In The Software Development Lifecycle*, Springer.

Sharp, H., Rogers, Y. and Preece, J. (2007), *Interaction Design: Beyond Human Computer Interaction*, Wiley.

Shearstone, P. (2008), *Goal Setting - Got Goals?*, viewed 23.8.2008, from http://sbinfocanada.about.com/cs/management/a/gotgoalsps.htm.

Shneiderman, B. (1992), *Designing the user interface (2nd ed.): strategies for effective human-computer interaction*, Addison-Wesley Longman Publishing Co., Inc.

Shneiderman, B. (1998), 'Treemaps for space-constrained visualization of hierarchies', *Human-Computer Interaction Lab, University of Maryland*, vol. 26, no. Dec.

Shneiderman, B. (2000), 'Creating creativity: user interfaces for supporting innovation', *ACM Trans. Comput.-Hum. Interact.*, vol. 7, no. 1, pp. 114-138.

Shneiderman, B. (2003), *Leonardo's Laptop : Human Needs and the New Computing Technologies*, The MIT Press.

Shneiderman, B. and Plaisant, C. (2004), *Designing the User Interface : Strategies for Effective Human-Computer Interaction (4th Edition)*, Addison Wesley.

Shneiderman, B., Fischer, G., Czerwinski, M., Resnick, M., Myers, B., Candy, L., Edmonds, E., Eisenberg, M., Giaccardi, E., Hewett, T., Jennings, P., Kules, B., Nakakoji , K., Nunamaker, J., Pausch, R., Selker, T., Sylvan, E. and Terry, M. (2006a), 'Creativity Support Tools: Report From a US National Science Foundation Sponsored Workshop', *International Journal of Human-Computer Interaction*, vol. 20, no. 2, pp. 61-77.

Shneiderman, B., Fischer, G., Czerwinski, M., Resnick, M., Myers, B., Candy, L., Edmonds, E., Eisenberg, M., Giaccardi, E., Hewett, T. and Others (2006b), 'Creativity Support Tools: Report From a US National Science Foundation Sponsored Workshop', *International Journal of Human-Computer Interaction*, vol. 20, no. 2, pp. 61-77.

Silva, P. P. d. and Paton, N. W. (2000), 'User Interface Modelling with UML', in *Proceedings of the 10th European-Japanese Conference on Information Modelling and Knowledge Representation*, Saariselkä, Finland, pp. 203-217.

Silva, P. P. d. and Paton, N. W. (2003), 'User Interface Modeling in UMLi', *IEEE Software*, vol. 20, no. 4, pp. 62-69.

Sommerville, I. (2004), *Software Engineering (7th Edition)*, Addison Wesley.

Sousa, K. S., Mendonca, H. and Vanderdonckt, J. (2008a), 'Addressing the Impact of Business Process Changes on Software User Interfaces', in *Proceedings of 3rd IEEE/IFIP International Workshop on Business-Driven IT Management (BDIM 2008)*, pp. 11-20.

Sousa, K. S., Mendonca, H. and Vanderdonckt, J. (2008b), 'User Interface Development Lifecycle for Business-Driven Enterprise Applications', in *Proceedings of the 7th International Conference on Computer-Aided Design of User Interfaces CADUI 2008*, Lopez Jaquero, V., Montero Simarro, F., Molina Masso, J. P. and Vanderdonckt, J. (eds.), Springer, Albacete, Spain.

Sousa, K. S., Mendonca, H., Vanderdonckt, J., Rogier, E. and Vandermeulen, J. (2008c), 'User Interface Derivation from Business Processes: A Model-Driven Approach for Organizational Engineering', in *Proceedings of 23rd Annual ACM Symposium on Applied Computing SAC'2008*, ACM Press, New York, pp. 553-560.

Spool, J. M. (1997, 1.9.1997), *Why On-Site Searching Stinks*, viewed 2.6, from http://www.uie.com/articles/search_stinks/.

Stachowiak, H. (1973), *Allgemeine Modelltheorie*, Springer, Wien

Stevens, W., Myers, G. and Constantine, L. (1974), 'Structured Design', *IBM Systems Journal*, vol. 13, no. 2, pp. 115-139.

Sukaviriya, N., Sinha, V., Ramachandra, T., Mani, S. and Stolze, M. (2007), 'User-Centered Design and Business Process Modeling: Cross Road in Rapid Prototyping Tools', in *INTERACT 2007*, al., C. B. e. (ed.), Springer LNCS, pp. 165-178.

Sutcliffe, A. (2000), 'On the effective use and resuse of HCI knowledge', *ACM Transactions on Computer-Human Interact.*, vol. 7, no. 2, pp. 197-221.

Sutcliffe, A. G. (2005), 'Convergence or competition between software engineering and human computer interaction'. in *Human-centered software engineering – integrating usability in the development process*, Seffah, A., Gulliksen, J. and Desmarais, M. C. (eds.), Springer, pp. 71-84.

The Standish Group (2003), 'CHAOS Report 2003: The Laws of CHAOS'.

The Standish Group (2006), 'CHAOS Report 2006: The Laws of CHAOS'.

Trætteberg, H. (1998), 'Dialog modelling with interactors and UML Statecharts - A hybrid approach'. in *Proceedings of the 10th International Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS,98)*, Markopoulos, P. and Johnson, P. (eds.), Springer, Wien, Austria, pp. 346-361.

Trætteberg, H. (2003), 'Dialog modelling with interactors and UML Statecharts - A hybrid approach.', in *Proceedings of the DSVIS 2003*, Springer, Funchal, Madeira, pp. 289-301.

Trætteberg, H. (2004), 'Integrating dialog modelling and application development', in *Proceedings of the First International Workshop on Making Model-based User Interface Design Practical MBUI*, Trætteberg, H., Molina, P. J. and Nunes, N. J. (eds.), January 13, 2004, Funchal, Madeira, Portugal.

Trætteberg, H. (2008), 'A Hybrid Tool For User Interface Modeling And Prototyping'. in *Computer-Aided Design Of User Interfaces V*, Calvary, G., Pribeanu, C., Santucci, G. and Vanderdonckt, J. (eds.), Springer, pp. 215-230.

Traetteberg, H. (2002), 'Model-based User Interface Design', Norwegian University of Science and Technology.

UIDesign (2003, 2003), *Cover Summary Pattern*, viewed 16.9.2008, from http://www.uidesign.net/1999/papers/CoverSummaryPattern.html.

Usability First (2005), *Usability Glossary: interaction design*, viewed 20.6.2008, from http://www.usabilityfirst.com/glossary/term_204.txl.

Usability First (2008), *Usability Glossary: user profile*, viewed 20.8.2008, from http://www.usabilityfirst.com/glossary/term_710.txl.

van der Veer, G. C. and van Welie, M. (2000), 'Task Based Groupware Design -Putting theory into practice', in *Proceedings of the conference Designing Interactive Systems (DIS 2000)*, New York, U.S.

Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D. and Florins, M. (2004), 'USIXML: a User Interface Description Language for Specifying Multimodal User Interfaces', *W3C Workshop on Multimodal Interaction. Sophia Antipolis*, pp. 19-20.

Vanderdonckt, J. (2005), 'A MDA-Compliant Environment for Developing User Interfaces of Information Systems', in *Proceedings of 17th Conf. on Advanced Information Systems Engineering CAiSE'05*, Pastor, O. and Cunha, J. F. e. (eds.), Springer Lecture Notes in Computer Science, Porto, pp. 16-31.

Vanderdonckt, J. (2008, 9.9.2008), *UsiXML - Home of the USer Interface eXtensible Markup Language*, viewed 10.10.2008, from http://www.usixml.org.

VDI/VDE (2008, 16.10.2008), *Hightech muss bedienbar bleiben*, viewed 28.10.2008, from http://www.vdi.de/6930.0.html?&tx_ttnews[tt_news]=46452&tx_ttnews[backPid]=6481&cHash=dd439aed31.

Virzi, R. A., Sokolov, J. L. and Karis, D. (1996), 'Usability problem identification using both low-and high-fidelity prototypes', in *Proceedings of the SIGCHI conference on Human factors in computing systems: Common Ground*, ACM Press New York, NY, USA, pp. 236-243.

von der Beeck, M., Braun, P., Rappl, M. and Schroder, C. (2002), 'Model based requirements engineering for embedded software', in *Proceedings of IEEE Joint International Conference on Requirements Engineering 2002*, p 92.

von Hippel, E. (2005), *Democratizing Innovation*, MIT Press, Cambridge, MA.

Walker, M., Takayama, L. and Landay, J. A. (2002), 'High-fidelity or low-fidelity, paper or computer? Choosing attributes when testing web prototypes', in *Proceedings of the Human Factors and Ergonomics Society 46th Annual Meeting*, pp. 661-665.

Wallace, M. D. and Anderson, T. J. (1993), 'Approaches to Interface Design', *Interacting with Computers*, vol. 5, no. 3, pp. 259-278.

Ware, C. (2004), *Information Visualization, Second Edition: Perception for Design*, Morgan Kaufmann.

Wells, D. (1999a), *eXtreme Programming - Story Cards for a Coffee Maker*, viewed 20.8.2008, from http://www.extremeprogramming.org/example/coffeestories.html.

Wells, D. (1999b), *eXtreme Programming - User Stories*, viewed 20.8.2008, from http://www.extremeprogramming.org/rules/userstories.html.

Winograd, T. (1996), *Bringing design to software*, Addison Wesley.

Winograd, T. (1997), *From Computing Machinery to Interaction Design*, viewed 20.6, from http://hci.stanford.edu/~winograd/acm97.html.

Wixon, D. and Jones, S. (1995), 'Usability for fun and profit: A case study of the re-design of the VAX RALLY'. in *Human-Computer Interface Design: Success Stories, Emerging Methods, and Real-World Context*, Rudisill, M., Lewis, C., Polson, P. G. and McKay, T. (eds.), Morgan Kaufmann Publishers.

Woodruff, A., Landay, J. and Stonebraker, M. (1998), 'Goal-directed zoom', in *CHI '98: CHI 98 conference summary on Human factors in computing systems*, ACM Press, pp. 305-306.

Zave, P. and Jackson, M. (1997), 'Four Dark Corners of Requirements Engineering', *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 1, pp. 1–30.

Zetie, C. (2005), 'Show, Don't tell - How High-Fidelity Prototyping Tools Improve Requirements Gathering', Forrester Research Inc.