

# Masterarbeit

## A Zoom-Based Specification Tool supporting Interdisciplinary User Interface Design Processes

von  
Florian Geyer

1. Gutachter: Prof. Dr. Harald Reiterer
2. Gutachter: Prof. Dr. Oliver Deussen

FB Informatik und Informationswissenschaft  
Studiengang Information Engineering  
Universität Konstanz

Mai 2008



# Abstract

Careful design of user interfaces is getting more and more important as computers increasingly determine the capabilities, limitations and organization of our work. Today, interfaces have to meet an extensive number of requirements on performance, functionality and usability. Additionally, the quality of user interfaces is also an economic factor, as it has a dramatical influence on customer satisfaction and brand value. Consequently, structured approaches to user interface design are characterized by a multitude of tasks and techniques that make interface development a complex process. However, current design practice is accompanied with a variety of employed tools and frequent transitions from abstract problem definitions to detailed specifications. Based on a detailed investigation of interface design theory and practice, this thesis introduces a novel interface specification tool that aims on addressing the interdisciplinary nature of the design process and gaps in communication and technology in order to ease work transitions. Nevertheless, it also aims on promoting creativity and innovation in design by offering informal means of expression and by allowing to relate design artifacts in context. It contributes to current design practice by respecting the need for iterations and continuous reviewing as well as collaboration among designers throughout the design process. This thesis explains the rationale behind the developed tool by a detailed examination of applied theoretical foundations, related research, practical design methods and utilized interface concepts, before presenting a conceptual model that led to the physical interface design. As a result of careful integration and implementation of innovative, zoom-based interaction and visualization techniques that closely resemble transitions in work practice, the introduced tool design extends the perception of interface specification tools to a broader, more comprehensive coverage of the design process. After describing the technical implementation, the thesis concludes with a review of contributions and remaining issues as well as implications for future investigations.





# Überblick

Die Gestaltung von Benutzeroberflächen gewinnt immer mehr an Bedeutung, da die Möglichkeiten, Grenzen und die Organisation unserer Arbeit zunehmend von Computern bestimmt werden. An sie wird eine umfangreiche Reihe von Anforderungen an Leistung, Funktionalität und Usability gestellt. Zusätzlich ist die Benutzerschnittstelle auch ein wirtschaftlicher Faktor, der Einfluss auf Kundenzufriedenheit und Markenwert hat. Strukturierte Ansätze der Benutzerschnittstellenentwicklung zeichnen sich daher durch eine Vielzahl von komplexen Methoden und Techniken aus. Prozessmodelle in der Praxis sind charakterisiert durch eine Vielzahl von eingesetzten Werkzeugen und häufigen Übergängen zwischen abstrakten Problemdefinitionen bis hin zu detaillierten Spezifikationen. Diese Masterarbeit stellt ein innovatives Werkzeug zur Spezifikation von Benutzerschnittstellen vor, das auf einer detaillierten Recherche von Theorie und Praxis basiert. Das entwickelte Werkzeug respektiert den interdisziplinären Charakter des Entwurfsprozesses und die Lücken in Kommunikation und Technik durch eine effektive Überbrückung von Werkzeugübergängen. Dennoch fördert das Werkzeug Kreativität und Innovation durch die Unterstützung von informellen Ausdrucksmitteln und durch die Möglichkeit erzeugte Artefakte im Kontext der Entwicklung zu betrachten. Durch Unterstützung von Iterationen und durchgängigen Evaluationsmöglichkeiten während des Gestaltungsprozesses, sowie der Kollaboration zwischen Designern wird der gesamte Designprozess abgedeckt. Diese Masterarbeit beschreibt die Hintergründe des entwickelten Werkzeugs durch eine ausführliche Beschreibung von angewandten theoretischen Grundlagen, verwandter Forschungsprojekte und praktischer Konzepte, bevor ein konzeptueller Entwurf und das finale Design vorgestellt werden. Durch eine bedachte Integration von innovativen, Zoom-basierten Visualisierung- und Interaktionstechniken, wird die Auslegung von Werkzeugen zur Unterstützung von Gestaltungsprozessen neu definiert. Nach Beschreibung der technischen Umsetzung, schließt die Arbeit mit einer Bewertung des Erreichten, offenen Fragen sowie Empfehlungen für zukünftige Entwicklungen ab.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation and Goals . . . . .	3
1.1.1	Need for a new Generation of UI Tools . . . . .	3
1.1.2	Incremental Innovation . . . . .	4
1.1.3	Proposed Features for effective Tool Support . . . . .	5
1.2	Research Approach . . . . .	6
1.2.1	Background . . . . .	6
1.2.2	Thesis Scope . . . . .	7
1.3	Thesis Structure . . . . .	7
<b>2</b>	<b>Theoretical Foundations</b>	<b>9</b>
2.1	User Interface Design and HCI . . . . .	9
2.2	General Approaches to UI Design . . . . .	11
2.3	Technology and Innovation . . . . .	12
2.4	Interdisciplinary Design . . . . .	13
2.4.1	Engineering and Design . . . . .	14
2.4.2	Separation of Concerns . . . . .	15
2.4.3	Bridging the Gap in Practice . . . . .	16
2.5	Treasuring Design Experience . . . . .	19
2.6	Summary . . . . .	22
2.6.1	Shortcomings . . . . .	22
2.6.2	Implications for Tool Support . . . . .	23
<b>3</b>	<b>Related Work</b>	<b>25</b>
3.1	A Lesson in History . . . . .	25
3.2	State-of-the-Art Tool Support . . . . .	26
3.3	Related Research . . . . .	28
3.3.1	Supporting early Design Phases by Sketching . . . . .	28
3.3.2	Hybrid Solutions to ease Work Transitions . . . . .	30
3.4	Research Gap . . . . .	33
<b>4</b>	<b>Analysis</b>	<b>37</b>
4.1	Structured Approaches to User Interface Design . . . . .	37
4.1.1	Integration with Software Development . . . . .	38
4.1.2	Widely adopted Structured Approaches . . . . .	38
4.2	Adapted Process Model . . . . .	45
4.2.1	Tasks, Techniques and Artifacts . . . . .	47
4.2.2	Interdisciplinary Selection of Artifacts . . . . .	48
4.3	Requirements for Tool Support . . . . .	50
4.3.1	General Guidelines . . . . .	50
4.3.2	Specific Requirements . . . . .	53
<b>5</b>	<b>Design</b>	<b>57</b>
5.1	Conceptual Design . . . . .	57
5.1.1	User Characteristics . . . . .	58
5.1.2	Contextual Work style . . . . .	61
5.1.3	The Design Room Metaphor . . . . .	67
5.2	Collecting Concepts - Interaction & Visualization . . . . .	71

5.2.1	Whiteboard Interaction . . . . .	71
5.2.2	Spatial Navigation . . . . .	76
5.2.3	Interactive Visualization . . . . .	84
5.2.4	Collaboration . . . . .	90
5.3	Physical Design . . . . .	94
5.3.1	Overview . . . . .	94
5.3.2	Interaction Concept . . . . .	95
5.3.3	Visualization & Navigation concept . . . . .	100
5.3.4	Collaboration . . . . .	106
<b>6</b>	<b>Implementation</b>	<b>109</b>
6.1	Technical Framework . . . . .	109
6.2	System Architecture . . . . .	110
6.2.1	System Components . . . . .	112
6.3	Lessons Learned . . . . .	116
<b>7</b>	<b>Conclusion</b>	<b>119</b>
7.1	Review . . . . .	119
7.1.1	Design Rationale . . . . .	119
7.1.2	Expert Evaluation & Field Study . . . . .	120
7.1.3	Reflection . . . . .	122
7.2	Future Work . . . . .	123
7.3	Summary . . . . .	126
<b>A</b>	<b>Methods and Tools</b>	<b>129</b>
<b>B</b>	<b>Conceptual Models</b>	<b>133</b>
<b>C</b>	<b>Screenshots</b>	<b>139</b>
<b>D</b>	<b>Evaluation Questionnaire</b>	<b>147</b>
	<b>Bibliography</b>	<b>153</b>

# List of Figures

2.1	HCI Design Curriculum . . . . .	10
3.1	SILK . . . . .	28
3.2	DENIM and DAMASK . . . . .	29
3.3	DiaMODL . . . . .	30
3.4	CanonSketch . . . . .	31
3.5	TaskSketch . . . . .	32
3.6	WinSketch . . . . .	33
3.7	Comparison of Selected Tools . . . . .	34
3.8	Comparison of Proposed Research Coverage to CanonSketch . . . . .	35
4.1	The STAR Lifecycle . . . . .	40
4.2	The Usability Engineering Lifecycle . . . . .	41
4.3	Usage-centered Design . . . . .	42
4.4	Scenario-based Design . . . . .	44
4.5	Adapted Process Model . . . . .	46
4.6	Interdisciplinary Visual Specification Framework . . . . .	48
4.7	A Framework for Creativity Support . . . . .	52
5.1	Role Map . . . . .	60
5.2	A Work Transition . . . . .	62
5.3	Transitions in Work Artifacts . . . . .	63
5.4	The Design Room . . . . .	64
5.5	Whiteboards in Practice . . . . .	69
5.6	Whiteboard Interfaces . . . . .	72
5.7	Direct Manipulation of Graphical Objects . . . . .	75
5.8	Spatial Navigation Options . . . . .	77
5.9	Panning & Zooming . . . . .	78
5.10	Semantic Zooming . . . . .	79
5.11	Examples of ZUIs . . . . .	82
5.12	Nested Containers . . . . .	86
5.13	Process Model Visualization . . . . .	87
5.14	Hierarchical Process Decomposition . . . . .	88
5.15	Nested Process Visualization . . . . .	90
5.16	Communication through Artifacts . . . . .	92
5.17	Collaboration Framework . . . . .	93
5.18	INSPECTOR's Main Window . . . . .	95
5.19	INSPECTOR's Zoomable Nested Structure . . . . .	97
5.20	Palette Tools . . . . .	98
5.21	Pointing and selecting . . . . .	99
5.22	Context Menus . . . . .	99
5.23	Toolbar Sections . . . . .	100
5.24	Scenario Level . . . . .	101
5.25	Storyboard Level . . . . .	102
5.26	Modeling Level . . . . .	103
5.27	UI Design Level . . . . .	104
5.28	Optional Views for Context & Detail . . . . .	105

---

5.29	Tracing & Linking Artifacts . . . . .	105
5.30	INSPECTOR on a wall-size Screen . . . . .	107
6.1	Model View Controller Architecture . . . . .	112
6.2	System Components . . . . .	113
6.3	Object Hierarchy (Data Model) . . . . .	114
6.4	Elements of the INSPECTOR Notation . . . . .	115
6.5	Interaction & Navigation Classes . . . . .	116
7.1	Proposed Spatial Visualization . . . . .	124
7.2	Proposed Dynamic Components . . . . .	125
B.1	Persona 1: Bill Jobs . . . . .	134
B.2	Persona 2: Donald Nielsen . . . . .	134
B.3	Persona 3: Ian Ambler . . . . .	135
B.4	A Design Room at IDEO . . . . .	136
C.1	INSPECTOR's main window . . . . .	140
C.2	INSPECTOR's zoomable nested structure . . . . .	141
C.3	Scenario Level . . . . .	142
C.4	Storyboard Level . . . . .	143
C.5	Modeling Level . . . . .	144
C.6	UI Design Level . . . . .	145
D.1	Evaluation questionnaire, p. 1/5 . . . . .	148
D.2	Evaluation questionnaire, p. 2/5 . . . . .	149
D.3	Evaluation questionnaire, p. 3/5 . . . . .	150
D.4	Evaluation questionnaire, p. 4/5 . . . . .	151
D.5	Evaluation questionnaire, p. 5/5 . . . . .	152

---

# List of Tables

3.1	Evaluation of state-of-the-art tool landscape . . . . .	27
4.1	Functional requirements (general UID support) . . . . .	54
4.2	Functional requirements (artifact support) . . . . .	54
4.3	Technical requirements . . . . .	55
4.4	Usability Goals . . . . .	55
7.1	Evaluation Results . . . . .	121
A.1	Process Artifacts (Analysis) . . . . .	130
A.2	Process Artifacts (Design) . . . . .	131
A.3	Process Artifacts (Evaluation) . . . . .	132
B.1	Features of the whiteboard metaphor . . . . .	135
B.2	Features of the room metaphor . . . . .	136
B.3	Features mapping of a combined metaphor . . . . .	137

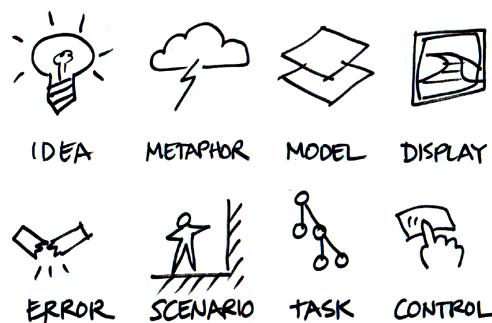




# Conventions

Throughout this thesis we use the following conventions.

- The plural "we" will be used throughout this thesis instead of the singular "I", even when referring to work that was primarily or solely done by the author.
- Unidentified third persons are always described in male form. This is only done for purposes of readability.
- Links to websites or homepages of mentioned products, applications or documents are shown in a footnote at the bottom of the corresponding page.
- A brief description of the main arguments is given in small margin notes at the beginning of each paragraph.
- References follow the Harvard citation format.
- The whole thesis is written in American English.
- The presentation of the rationale behind the presented design work is visually accompanied using Bill Verplank's four-column interaction design framework. Therefore, the below presented icons of the framework are utilized to guide through the parts of the thesis that are relevant to the rationale of the presented specification tool. Beginning with the identification of existing issues and possible solutions, implications for design are presented in a step-by-step structured manner.



(Verplank et al., 2001)

- A CD-ROM, attached to this thesis includes an electronic version of this document, an installer of the developed tool and a video presentation.



# Chapter 1

## Introduction

*“A picture is worth a thousand words.  
An interface is worth a thousand pictures.”*

—Ben Shneiderman

User Interfaces (UIs) are the face of almost all software applications. They determine the capabilities, limitations and organization of our work with computers. 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 and Plaisant, 2004, p. 16 ff): Life-critical systems require error-free interaction and rapid performance. Office and entertainment software requires ease of learning and subjective user satisfaction to compete on the market. Experimental systems or expert systems that focus on the enhancement or replacement of existing methodologies, like creative or cooperative interfaces, need to fulfill very high expectations to be adopted in practice. To achieve a high quality of the user interface, designers also have to respect the human diversity in terms of physical, cognitive and perceptual abilities to allow accessibility for all type of users. Especially with the rise of the World Wide Web (WWW) and public information displays the number of first-time users increased, which sets even higher demands on UI designs. Graphical interfaces also transport corporate values to end users and are therefore an important communication medium.

Why focus on User Interfaces?

Nevertheless, there are also strong business factors that require careful design of user interfaces. A good designed UI can save development costs, the need for updates and can enhance the success of a product on the market. Mayhew (2005, p. 17ff) gives some good examples, why usable user interfaces are worth their effort: Satisfaction of customers and higher productivity by providing good designed interfaces leads to customer trust and loyalty. This contributes to increased sales, cost savings and productivity. The usability of products also affects the public perception of a company, which influences market share and brand value. Software reviews in magazines, journals or newspapers often focus on user-friendliness and usability. While these factors regard the external return on investment (ROI) on the market, there are also internal benefits of employing advanced UI development methods. Dealing with UI issues and their requirements before actually implementing the UI avoids late changes in the development of a product. While early changes are easy to handle, late changes require even more additional changes and therefore increase development costs. If a product is already in use by the customer and the user interface is changed, updates are costly and may have influence on the customer’s satisfaction. Well-designed user interaction can also save support costs and documentation needs.

Good design is a business factor

*“As far as the customer is concerned, the interface is the product.”*  
(Raskin, 2000, p. 5)

Lack of attention to usability	While all these facts should make clear that user interface development requires extensive attention many products still lack good design. One reason is that the process of designing interfaces is not straightforward and involves many efforts. According to Myers and Rosson (1992) surveys show that on average 45 percent of the design and 50 percent of the implementation of new software is devoted to their user interface. While developers should determine appropriate UI techniques early in the product development cycle, they often lack budget and time to do this properly (Mayhew, 2005, p. 35). Since competing groups and market pressure is limiting resources, it is often hard to find justification for usability costs. Usability frequently remains a surplus and becomes less important than branding, user experience and other preferences. Eventually, usability surely is not the only key factor to effective ROI (Mayhew, 2005, p. 35).
User interface design is a complex task	However, even if companies employ usability methods early on in the development process, success is not guaranteed. UI design methods involve creativity, cooperation among different disciplines, decision making and various external - often domain-specific constraints - which make the process difficult and unpredictable. There is not one single approach to user interface design, which is applicable to all domains and requirements. During the 1980s and 1990s, a wide range of procedures and methods evolved (Benyon et al., 2005, p.1).
Variety of approaches and requirements	Some approaches are focused on engineering methods, which are regarded as heavyweight, while other lightweight approaches promote creativity in design as well as user experiences (UX). Each method has its qualification and choosing the right one for the target application and context is not easy (see Chapter 4.1—"Structured Approaches to User Interface Design"). Myers and Rosson (1992) give additional reasons why user interface development is inherently more difficult than creating other kinds of software: Iterative processes are required, which make using software engineering methods more complicated and more time-consuming. User interfaces depend highly on their context of use, user needs and domain-specific tasks and therefore require an extensive up-front requirement analysis. As Laurel (1990, p. 3) explains, the number of created requirements during the design process is extensive and originates from various sources. Since some usability methods involve user feedback into the development process, additional effort is created in the form of usability studies, focus groups or questionnaires.
Technological changes	Research in usability and user interface design approaches is an evolving discipline, which is often technology-driven. As technological changes occur, existing methods need to be adapted as new technologies introduce new possibilities. Preece et al. (1994, p.8) identifies the two most important challenges in human-computer interaction as the ability to keep abreast of changes in technology and to ensure that designs offer good interaction while harnessing the potential functionality of new technology. Since the user interface is the key to the underlying software functionality, cooperation between designers, interface experts and domain specialists as well as programmers is crucial to the success of development projects.
Interdisciplinary design	User interface design is by its nature an interdisciplinary process. Developing interfaces for a wide variety of users involves the participation of different perceptions into the design process. Human factors have to be combined with software engineering and creative design. other actors that might participate in the design process include marketing, information design, or graphical design. However, sometimes even future users are involved into design projects. Actors with different backgrounds and perceptions have to collaboratively work together to agree on one approach or decide between different design alternatives to finally create a user interface design. Interface designs are merely a compromise of different solutions:

"The difficulty of interface design is compounded by the fact that virtually all solutions are compromises. Solutions are shaped by a multitude of problems that are invisible to those outside of the design process"  
(Laurel, 1990, p. 3)

Frequently communication issues are hampering the development. Actors are used to their discipline-specific tools and methods, which are often not compatible to others and complicate sharing of information and decision-making. Additionally, the perception of design artifacts is inherently different between disciplines. Misunderstandings, confusion and lack of communication are results of these issues. Effective means of discussion, common understanding and decision-making therefore is an essential part of interface design.

Additionally, we identified a shift of responsibilities in our research for our industrial partners, Dr. Ing. h. c. F. Porsche AG and Daimler AG that affects specification of user interfaces (Memmel et al., 2007i,h; Memmel and Reiterer, 2007; Memmel and Heilig, 2007). Due to the strategic impact of interface products, industrial companies (clients) seek to increase their competence in UI design in order to reflect their corporate values in their products. Traditionally, interface development was completely outsourced to suppliers. Due to the increasing importance of good design, clients now aim at keeping actual design work in-house, while still outsourcing implementation. The role of the supplier is then limited to programming the final system based on a specification. While the client identifies timetable advantages and inherently better control over design, he also gains flexibility in choice of suppliers. Costly and time-consuming iterations between client and supplier are reduced to a clear frontier: Conceptual Design, prototyping and evaluation of interfaces is conducted by the client, implementation by the supplier. However, the demands on specification techniques that support this scenario now require novel approaches. Currently used text-based specifications or prototypes created with office applications seem awkward for this purpose. Due to the lack of programming expertise at the client side, novel approaches have to focus on providing visual methods of modeling and prototyping that eventually lead to an externalizable specification. Considering all these difficulties in interface design, it is obvious that it is a tough challenge for both research and business communities to make user interface design less complicated (Laurel, 1990, p. 4).

Shift of responsibilities

## 1.1 Motivation and Goals

In the following, we present the motivation for our research and briefly introduce the methodology and primary goals that accompany our efforts.

### 1.1.1 Need for a new Generation of UI Tools

According to Preece et al. (1994, p. 590) a major drawback in UI design practice is the use of tools that are not specific to user interface design tasks. As a result, user interface designers are used to general-purpose software, intended for use in various contexts. Common stand-alone software tools in use are drawing or graphics tools, modeling and diagramming software as well as Computer Aided Software Engineering (CASE) tools, visualization tools, prototyping software and user interface toolkits or GUI Builder (see Chapter 3.2—“State-of-the-Art Tool Support”) (Preece et al., 1994, p. 590). Current tools therefore usually support only one aspect of the design life cycle: the construction and prototyping of the user interface, while neglecting requirements, specification and evaluation (see Chapter 3—“Related Work”).

Current tool usage is insufficient

The need for a new generation of tool support in UI design practice is backed up by several authors in Human-Computer Interaction(HCI) research. Shneiderman (2003, p. 214ff) argues for more creativity support in user interface design tools and proposes his framework of “mega-creativity” for tool development (see Chapter 4.3.1—“Design Guidelines for Creativity Support”). Shneiderman (1992, p. 42) also describes the need for tools supporting HCI techniques and knowledge for commercial developers as one of the major goals of his profession in user interface design. As part of his research agenda on software tools, he claims following demands:

Need for tool support accepted in HCI

"Innovative methods of specification involving graphical constraints or visual programming seem to be a natural match for creating graphical user interfaces[...] Collaborative computing tools may provide powerful authoring tools that enable multiple designers to work together effectively on large projects.[...] Evaluation tools are still an open topic for user interface and web-site developers. Specification by demonstration is an appealing notion, but practical application remains elusive."

(Shneiderman, 1992, p. 206)

Lack of tool support  
harms creativity  
and traceability

Raskin (2000, p. 4) also argues for tools that facilitate design innovation. According to him, current interface building tools are lowering development costs and speed up implementation, but enshrine current paradigms and therefore do not promote creativity. Nielsen (1994, p. 264) inflames a discussion of, what he calls Computer-Aided Usability Engineering (CAUSE) tools as future development. He proposes that such tools should support prototyping, interactive construction and manipulation of screen layouts, specification techniques, hypermedia representations of design knowledge and guidelines, design rationale representations and feedback techniques for evaluation support (Nielsen, 1994, p. 264f). As tool support for creating conceptual models during requirement analysis is currently constrained to general-purpose software like diagramming tools or dedicated software engineering tools (CASE), creators of model-based approaches, like Constantine and Lockwood (2003, p. 6) also claim demand for dedicated tools:

"Perhaps most pressing is the need for tools that support usage-centered software engineering by incorporating its models and exploiting their interconnections for flexible concurrent modeling and systematic requirements tracing."

(Constantine and Lockwood, 2003, p. 6)

### 1.1.2 Incremental Innovation

Changing work  
practice by  
incremental  
innovation

Based on the preceding emphases on required tool-support, one may ask why it is not possible to address some issues with other means than the design of a novel supporting tool. However, we think that there might be other solutions as well, but tool-support is also a way of imposing new theoretical structures and methods for use in practice and therefore may change the minds of people in an effective way. The primary intention behind a tool-based approach lies in the fact that tools are an effective way of changing work practice. A prominent example of how our work is revolutionized by tools is the history of word processing software (Beyer and Holtzblatt, 1997, p. 8). Originally, everybody used typewriters and using them became a widespread practice. Early word processing software stayed close to the original functionality of real typewriters, but provided additional functionality like advanced typing support and correcting. When cut and paste functionality was introduced, it was instantly adopted in practice because it was a real life metaphor of cutting and pasting with scissors or glue. This procedure was something that everybody was familiar with in everyday life. As more and more features were added to word processors over the time, today's tools do not have very much in common with real typewriters, except the physical act of typing. A reason why users adapted to this new way of working was that it did change the work enough to make it more effective, but not too much to allow a transition. This short history of word processing also implicates how innovation can be successful:

"Then how can a design innovate successful? By taking one step at a time, always considering the interaction between the new ideas and the current work practice."

(Beyer and Holtzblatt, 1997, p. 8)

Therefore, an innovative tool approach for user interface specification should consider current work practice and make use of techniques and metaphors that are already established in practice. Applying new concepts and techniques in an iterative manner will more likely lead to adoption in practice. Eventually, adoption will lead to a change of minds and consequently a new definition of interface specification.

Iterative innovation is important

### 1.1.3 Proposed Features for effective Tool Support

“We humans are in love with our tools because they help us become more than we are, to overcome our limitations and extend the boundaries of what it is possible to do with our brains and bodies.[...]and we have a knack for inventing tools to help us overcome those limitations in pursuit of our goals”  
(Laurel, 1990, p. 225)

Based on our findings (see Chapter 4—“Analysis”) and that of others in shortcomings and drawbacks of current practice in UI development, we think that adequate tool support accompanied by a suitable theoretical framework is able to make user interface design less difficult. We believe that there is a lack of an integrating tool-based approach to interdisciplinary interface specification. Consequently, we propose the development of an innovative experimental tool that focuses on interdisciplinary issues, facilitates effective communication and promotes creativity in design as well as support for maintaining design experience. We therefore suggest following features for effective tool support:

Ideas for effective tool support in interdisciplinary specification

- **One Solution**

Integrate all required tasks of interdisciplinary user interface specification into one tool-based solution. This includes the conceptual requirements analysis, construction and prototyping of the user interface as well as specification and evaluation.

- **Bridge Disciplines**

Effectively address the gaps in technology and communication among disciplines that participate in the design process (see Chapter 2.4.2—“Separation of Concerns”).

- **Respect different Perceptions**

Respect the different perceptions to the subject matter, while still maintaining a common understanding of design artifacts (see Chapter 2.4—“Interdisciplinary Design”).

- **Support Creativity and Innovation**

Promote creativity in design and take advantage of innovative means of collaboration in interdisciplinary teams (see Chapter 2.4.1—“Engineering and Design”).

- **Iteration and Decision-Making**

Support the iterative nature of design processes and provide functionality for decision-making over alternatives, include support for evaluation and feedback throughout the design process.

- **Treasure Design Experience**

Keep track of developments and major changes during design to allow capturing of design knowledge for future projects (see Chapter 2.5—“Treasuring Design Experience”).

- **Externalize Specification**

Provide adequate means of design space visualization to enhance traceability and to exploit interconnections between related artifacts for specification purposes.

## 1.2 Research Approach

*“We become what we behold.  
We shape our tools, and thereafter our tools shape us.”*

—Marshall McLuhan

### 1.2.1 Background

BEST research  
cooperation

This work is based on previous and ongoing research at the HCI workgroup<sup>1</sup> at the University of Konstanz. It is settled within the research cooperation “Business Excellence in Software Usability and Design” (BEST) with Daimler AG Research Software Technologies. One of the goals within this cooperation is the investigation of links between principles and methods of Agile Modeling (AM) and Usability Engineering (UE).

Project setting and  
related work

While the described work is still an active research project, it incrementally developed since October 2006. Previous research within our group provided important fundamentals to our research. (Mommel and Heilig, 2007; Mommel et al., 2007f) introduced a visual specification method that is based on a model-driven chain of tools. While one tool was dedicated to information design, another tool was used to design interfaces. Eventually, a third tool connected both domains based on behavioral prototyping. While this approach provides efficient separation of concerns it is characterized by a high degree of formality. Consequently, it lacks creativity support by informal means of expression. In addition, the high degree of formalization enabled rapid functional prototyping with code generation but neglected flexibility and ease of use. However, experiences led to shift of focus on more informal and easier to use specification techniques as well as support for conceptual modeling. Some theoretical techniques and methods referenced in this work are also based on previous research in agile design approaches (Gundelsweiler et al., 2004; Mommel et al., 2007c,h), interdisciplinary design techniques (Mommel et al., 2007d), and visual specification as well as prototyping methods (Mommel et al., 2007e,j), which will be described in Chapter 4.2.2—“Interdisciplinary Selection of Artifacts” and 4.1.2—“Widely adopted Structured Approaches”. A major goal of this experimental tool therefore is also to provide a test environment to evaluate theoretical methods in bridging the gaps between disciplines. Other local related work also includes the investigation of state-of-the-art specification and prototyping tools as well as feedback techniques for evaluation (König, 2008). Publications related to the tool-design described in this work were already published by research group members Mommel and Reiterer (2008) and König (2008) and will appear in Rinn (2008). More information on this project, screenshots and video presentations are also available on the Project Website<sup>2</sup>.

Inspiration in  
visualization and  
interaction

This work was also inspired by expertise within our group in innovative visualization and interaction techniques, like post-desktop interface paradigms (Jetter, 2007) based on Zoomable User Interfaces (ZUIs) (König, 2006; Gerken, 2006) as well as exploration and interaction techniques for information spaces and displays (König et al., 2007). As of further examination, we think that innovative concepts and paradigms developed by Human-Computer Interaction and Information Visualization research may help to provide adequate means of visualization and interaction with multiple degrees of abstraction layers that accompany the design process and communication.

<sup>1</sup><http://hci.uni-konstanz.de/>

<sup>2</sup><http://hci.uni-konstanz.de/inspector>



## 1.2.2 Thesis Scope

On the one hand, the focus of this thesis lies on introducing a tool-based approach supporting interface design, on the other hand, this work is also strongly connected to parallel research in theoretical methods that support interdisciplinary design and advanced specification techniques. As a result, tool design and the corresponding interdisciplinary specification approach coevolve and incremental changes in theoretical methods eventually influence design and vice versa. Since there are many theoretical factors that influence this tool-based approach and its application during the design process, it is crucial to analyze the variety of connected aspects before presenting a design. Theoretical fundamentals of user interface design are as important as actual design approaches and work practice. Observations of current practice and recommendations within the research community will point out problems that are hampering interdisciplinary cooperation and may lead to solutions. The parallel designed theoretical framework to interdisciplinary specification then also proposes activities and tasks during process steps that have to be analyzed to understand user needs and implications for design. Therefore, this work includes a theoretical foundation and analysis of the problem domain before introducing a novel tool design. During this part of the thesis, the strongly connected methodical foundations are briefly described, ultimately this thesis is investigating new means of visualization and interaction that will be tested on theoretical methods that are developed at the same time. Consequently, the displayed models and artifacts are a result of parallel theoretical research that influenced design, but are not described in detail within this work. For a detailed rationale behind the theoretical framework, see Rinn (2008).

This work is accompanied by parallel research in a theoretical approach

## 1.3 Thesis Structure

As from our experience we think that an innovative tool is required to address issues found in user interface design practice. This work introduces an innovative tool design to support interdisciplinary teams in collaborative user interface specification. The rationale behind this tool is based on established and novel theoretical methods in bridging the gaps between the disciplines as well as a detailed analysis of work practice. By utilizing innovative visualization and interaction concepts to adapt to the problem domain, the tool design aims at leveraging creative thinking and innovation in design. Eventually innovative user interface design starts with employing innovative design tools. The thesis is split into seven chapters:

Thesis statement

The next chapter, 2—“Theoretical Foundations” reviews the main theories and fundamentals behind our approach, drawn from acknowledged research literature. It describes and analyzes the current state of interface design practice and important contributions that are relevant to the research at hand. Therefore, general philosophies and approaches to interface design are presented with an emphasis on creativity and innovation in design. Thereafter, issues of interdisciplinary design and implications for our research are presented. After describing techniques that are relevant to the issues of specification and capturing of design experiences, the chapter concludes with a summary on shortcomings in design practice as well as possible solutions to these issues. Readers that have a dedicated background in Human-Computer-Interaction may refer to the summary of this chapter.

Chapter Outlook

Chapter 3—“Related Work” describes most recent developments in the field of interface design and hybrid requirements modeling. Therefore, a brief overview on tool history and state-of-the-art tool support is given before resulting implications for investigation of innovative supporting concepts are described. Eventually, related research which focuses on exactly this field is presented. Finally, our research is presented within a comparative framework that reveals the research gap we are approaching.

Chapter 4—“Analysis” presents a detailed analysis of requirements that have to be met to efficiently support not only specification but also the overall design process. Therefore, integration of our approach to interface specification in practice is investigated by analyzing

commonly used structured approaches and acknowledged process models. Thereafter, an adapted process model is presented that respects the demands we identified in the previous chapters. Eventually, an interdisciplinary specification framework is presented that imposes a structure on integrated techniques, artifacts and resulting tasks. Finally, a summary on elaborated implications for design is presented with a detailed listing of general guidelines, specific requirements and goals for a novel design concept.

Chapter 5—“Design” explains both the conceptional fundamentals behind our proposed tool design and the result of our research as a final physical design. Consequently, the chapter starts off with introducing conceptual models that led to the basic design concept along with an analysis of contextual work style in practice. Thereafter, a blended metaphor that originated from contextual work style is presented that serves as the fundamental to physical design. Subsequently, interaction and visualization concepts are presented that were utilized to map the metaphor to an actual interface design. Eventually, the final physical design of our proposed tool is introduced and described along aspects of interaction, visualization and collaboration.

Chapter 6—“Implementation” briefly presents the technical implementation of the presented tool design. Therefore, it gives an overview on the technical framework before describing the utilized system architecture and system components. Eventually, experiences with implementation are presented along implications for future projects.

Finally, Chapter 7—“Conclusion” presents the outcome of our research by providing a review based on our experiences throughout the research process, evaluation results and a reflection of the final design against initial requirements. Thereafter, the thesis concludes with a brief summary that includes the contributions made and remaining issues.

## Chapter 2

# Theoretical Foundations

*“The important thing is not to stop questioning.”*

—Albert Einstein

In the following, the main theories and fundamentals behind our approach are reviewed, drawn from acknowledged research literature. Because we argue to respect overall design practice in a specification tool, the current state of interface design theory and important contributions that are relevant to the research at hand are briefly introduced. General philosophies on interface design are presented with an emphasis on creativity and innovation, interdisciplinary design, the capturing of design experience and specification. Along with the investigation, implications for our research are presented.

Outlook

### 2.1 User Interface Design and HCI

Before beginning to present origins and fundamentals of the research described in this work, it is essential to define referred terms and to relate them in context. Throughout this thesis, the term “Interface” or “User Interface” is used as a short term of a human-computer interface. There are many definitions for the term “Interface”. While some authors like Raskin (2000, p. 2) refer to the user interface as any form of interaction by the user with the computer, (e.g. speech input) the use of this term in the context of this work refers to the graphical user interface (GUI) used in common consumer software or web-based services. These interfaces are typically visual, window-based and controlled by mouse-driven inputs. Myers and Rosson (1992) gives an adequate definition:

User interface definition

“[A user interface is] the software component of an application that translates a user action into one or more requests for application functionality, and that provides to the user feedback about consequences of his or her action.”  
(Myers and Rosson, 1992)

Research puts many efforts into HCI to investigate new means of creating successful user interfaces. According to Benyon et al. (2005, p.1), it evolved as area of study during the 1980s and became more and more important ever since. With the rise of graphical user interfaces, the WWW and mobile phones the human-computer interface became more and more substantial in everyday life and style was becoming important as function. Technical developments, like personal digital assistants (PDAs), tablet computers, growing display space, wireless communication and new input devices as well as other emerging information appliances, created additional demands to the design and implementation of interfaces.

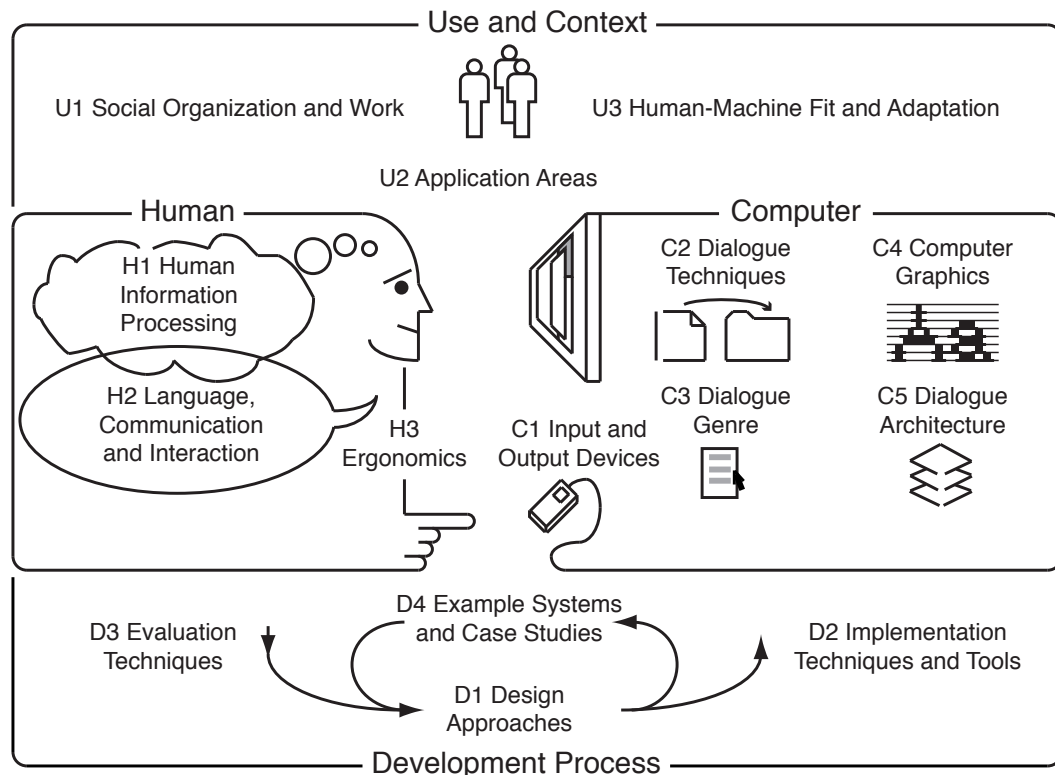
Human-Computer Interaction

Today there is a mix of ideas, approaches and technologies that are used by different actors in various contexts and application areas. HCI still is a dramatically emerging discipline and draws a lot of attention (Benyon et al., 2005, p.1). However, its scope of research is broader than just the design of the user interface. Moreover, it is concerned with all aspects that are related to computers and the interaction of humans with them (Preece et al., 1994, p.8).

“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.”  
 ACM SIGCHI Curricula for Human-Computer Interaction<sup>1</sup>

Areas of research in HCI

Consequently, many factors contribute to successful human-computer interaction development. A working group of the Special Interest Group for Human-Computer Interaction (SIGCHI) of the Association for Computing Machinery (ACM) has published a “HCI Design Curriculum”. Figure 2.1 gives an overview on this research field, which is meant to convey the scope and idea of HCI.



**Figure 2.1:** HCI Design Curriculum, taken from the the ACM SIGCHI Curricula for Human-Computer Interaction

User Interface Design and Interaction Design

The term “User Interface Design”, as referred to in this work, is concerned with the following fields regarding the development process: D1 - Design Approaches, D2 - Implementation Techniques and Tools, D3 - Evaluation Techniques and D4 - Example Systems and Case Studies. The term “Interaction Design” (IxD) is also often associated with those fields that are concerned with developing user interaction. While “User Interface Design” is focused on the development of the actual interface itself, “Interaction Design” has a broader understanding by looking at the design of system interfaces in a variety of media. Its focus is stronger on those aspects of an interface that defines its behavior over time and is much more concerned about practice, namely how to design user experiences (Sharp et al., 2007, p. 9). The user interface itself becomes a part of the interaction, explicitly that part

<sup>1</sup><http://www.sigchi.org/cdg/cdg2.html>

that represents the offered interaction choices or system responses. There are many different definitions of the term itself. Some interaction designers have their own perception of this field. While Cooper et al. (2007, p. 9) describes it as “goal-directed” and includes the fields “Form”, “Content” and “Behavior” as well as “experience design”, Sharp et al. (2007) defines Interaction Design much wider:

“[Interaction Design is] designing interactive products to support the way people communicate and interact in their everyday and working lives.”  
(Sharp et al., 2007, p. 8)

The scope of Interaction Design therefore includes that of “HCI” and “User Interface Design”. This work consequently interprets the term “Interaction Design” as applied methods that are concerned with the practice of designing user interface interactions in a creative manner that enhances user experiences.

## 2.2 General Approaches to UI Design

According to Wallace and Anderson (1993), there are four general approaches to interface design that developed due to discipline-specific perceptions: the craft approach, cognitive engineering, enhanced software engineering and the technologist approach.

Four general approaches

The craft approach looks at user interface design as a piece of art created by an artist. Designs are created based on the experience and knowledge of one or more designers. Each new design produced by the artists is unique and depends on the talent, innovation and creativity of the individual personalities. Design knowledge is communicated to other people by the general principle of a master and his apprentice. The apprentice learns by observing his master and tries to imitate his practice. The craft approach has no specific process model. As the design is driven solely by the knowledge of the artists, it often results in a lack of documentation for final designs. Initial requirements for tasks and users are often neglected and the lack of a process makes traceability of design decisions awkward. According to Le Peuple and Scane (2003, p. 26f), this approach to interface design was very common in early days of interface development and is still very widespread in small or medium website developments or small businesses. Today, these approaches are not adequate, since user interface projects trend to become more and more complex. In addition, rapid technological changes overburden single designers and require structured team-based processes.

The craft approach: creative but unstructured

Cognitive engineering approaches rely on the application of cognitive psychological theories as a fundament to user interfaces. They focus on the psychological study of human beings and their thinking, feelings and actions when doing work. Consequently, various approaches aim to understand how people process information in a specific context in order to establish a guide for the concerned actions in computer interaction. Therefore, these approaches involve early user feedback into their process steps. According to Le Peuple and Scane (2003, p. 26f), a major benefit of this procedure is that early evaluation tends to reduce late changes in development, which would raise costs dramatically. Exactly this is also one of the downsides of cognitive engineering approaches. Studying users is time-consuming and expensive. There is also not a guarantee of success, since designers also have to interpret and understand user needs based on their knowledge and domain-specific constraints.

Cognitive engineering: study users and actions

Enhanced software engineering methods aim on the introduction of HCI techniques into established software engineering processes, as these approaches often neglect the need for careful UI design. Therefore, bridging approaches are embedding user interface methods into several stages of the cycle, or into the overall software development process. As discussed later on (see Chapter 2.4—“Interdisciplinary Design”), the variety of requirements create a more complex process and requires interdisciplinary cooperation as well as expert

Enhanced software engineering: Complexity and effort

knowledge. Due to a lack of tool support and the expertise required to carry out such complex processes they are currently only sporadically adopted in practice.

Technologist approach:  
Separation of concerns

The technologist approach aims at automating the UI design process by employing software tools. Through easy-to-use tools for interface development and prototyping, designers should be enabled to create user interface specifications without having to learn complex programming languages. The basic principle here is to separate the concerns of user interface specialists or HCI designers and software professionals by creating a clear frontier between user interface design and application functionality. As more and more user interface tools appeared, they were widely adopted in practice. As of further examination (see Chapter 2.4.2—“Separation of Concerns”), many tools have a lack of functionality and neglect important aspects of user interface design, like requirements analysis and user needs. As most of the tools that became employed in practice are formal and enshrine current paradigms, they harm creativity and innovation in design.

Conflict in philosophies



Wallace and Anderson (1993) conclude that a perfect approach would effectively combine the benefits of each approach into one process model. This is a virtually impossible task, as many different philosophies have to be compromised. Today, there is not one single approach, which addresses all criteria proposed by Wallace and Anderson (1993) for an effective design process. It is therefore a major research objective to investigate new means of approaching interface design.

Implications for tool design



In respect to the research at hand, we think that it is a major issue to respect different perceptions on interface design. Nevertheless, we also believe that focusing on one single approach does not solve the problem. Instead, a combination of compatible methods and techniques or adaptability to changing demands may provide an effective solution. However, further investigation is necessary to understand the interconnections and conflicts between the described approaches.

## 2.3 Technology and Innovation

“Human-Computer Interaction is the kind of discipline which is neither the study of humans, nor the study of technology, but rather the bridging between those two.”

Interview with Terry Winograd in Preece et al. (1994, p. 53)

Study of requirements and technology is crucial

Interactive computer systems should be designed in a way, so that everyone is able to use them effectively. That does not mean that all computer systems should be designed for everybody, but for those people, for whom they are intended. Therefore, it is essential to find out about the human needs and requirements of the future users as well as the tasks involved before designing interfaces. Nevertheless, it is also crucial to have an eye open to the technological constraints and possibilities when designing interfaces. This brings up another critical aspect of interdisciplinary design: Engineering methods heavily rely on technical knowledge and innovations (Preece et al., 1994, p. 352), while human factors may constrain the use of new technology.

Evaluate acceptance of interface innovations

Technological developments might provide new means of visualization and interaction that fit the problem domain. HCI can benefit from these technologies, but also has to respect the acceptance of these innovations. According to Nielsen (1994, p. 266) it is a basic fact of human nature, that technological innovations are not immediately accepted by the majority of users. Concepts have to prove their eligibility and profit in practice. Consequently, it is a major concern in user interface design to make sure that innovative methods are employed the right way. There are certainly different types of innovations that determine the way in which technology is accepted. Incremental innovation is more likely to be accepted than revolutionary innovation. For example, the mouse - which is considered as a revolutionary innovation - was invented in 1964 by Engelbart, but was not used in practice until 20 years

later (Nielsen, 1994, p. 266). One way to speed up acceptance is to focus on incremental innovations that reuse or enhance existing methods and therefore do not constitute a large burden for the user.

The “Task-Artifact-Cycle”, described in Carroll (1991, p. 79f) demonstrates the principle that technology coevolves with HCI development itself. A task therefore sets requirements for the development of artifacts to support it. The resulting artifact then suggests possibilities and introduces constraints that often influence the task for which the artifact was originally created. The new task sets redefined requirements for a new design of the artifact and so on. This principle also manifests the need for iterative processes and flexible designs.

HCI is an innovative process itself

In respect to our investigation on tool design, we think that a specification tool should support iterative development in terms of functionality. Nevertheless, we also think that the tool design itself should be developed in an iterative manner. By respecting recent developments in technology and still having a look on adoptability in practice, we think that careful design is capable of imposing a change on practice. We therefore focus on evolutionary innovation instead of revolutionary innovation.

Implications for tool design



## 2.4 Interdisciplinary Design

“Not only does the sheer number of requirements increase the difficulty of interface design, but the variety of sources from which the requirements come requires that successful interface design be a multidisciplinary process.”  
(Laurel, 1990, p. 3)

User interface design is, by its nature, an interdisciplinary process. This is an obvious fact given that technology has become a substantial part of our everyday lives. Developing user interfaces therefore involves various disciplines that influence the use of interactive systems. HCI and Interaction Design literature, like Borchers (2001, p. 3ff), Sharp et al. (2007, p. 10), Laurel (1990, p. 31ff) and Benyon et al. (2005, p. 22), regard software engineering, human interface design, including human factors and the application domain as key disciplines. Other disciplines, which are frequently involved, include marketing, graphical design and information design. Some design approaches, like User-centered Design (Norman and Draper, 1986) or Participatory Design (Muller, 2003; Moggridge, 2006, p. 59ff), which originated from industrial design, even involve users in their requirement elicitation and evaluation techniques as well as actual design work (see Chapter 4.1—“Structured Approaches to User Interface Design”). Human interface designers or HCI experts often take a special role in interdisciplinary design. According to Borchers (2001, p. 4) a study showed that HCI actors usually deliver the highest ROI among any other members of a UI development team. When it comes to identifying problems in user interface designs HCI methods are up to three to four times more effective than software engineering techniques. Therefore, HCI experts often take a leading role in interdisciplinary design.

Involved disciplines

HCI substantially agrees on the demand for interdisciplinary communication to achieve good design (Borchers, 2001, p. 3ff). On the one hand, HCI designers are trained to understand the user’s needs, technologies and interactions to create effective user experiences, but are not specialized in implementing UI designs. On the other hand, software developers have sufficient background in implementing user interfaces, but also have to understand the business side, technological constraints and marketing interests. It is very difficult for one person to acquire all different kinds of knowledge necessary for successful interface development. Depending on the product, size and scope of a UI development project, the size of a company as well as its design philosophy, interdisciplinary teams are frequently employed in practice. Employed disciplines strongly depend on the application domain. Sharp et al. (2007, p. 11 ff) describes an additional benefit of interdisciplinary design. One of the positive aspects of bringing together people with different backgrounds and training is the potential of many more ideas being generated. Different perceptions promote new methods as well

Interdisciplinary Design is necessary

as more creative and original designs.

Communication  
Issues



Exactly this benefit is also one of the downsides of interdisciplinary design, besides the fact that having more people participate means more costs. The more people involved in a design team, the more difficult is it to communicate and progress forward as designs are being created (Sharp et al., 2007, p. 11 ff). People with different backgrounds have different perceptions of the things they are seeing and which they are talking about. What is important to one person may not even be seen by another person. These issues lead to misunderstandings, confusion and lack of communication. Team members make use of different terms or utilize the same terms to describe different things. The listed problems may lead to a complete failure of the project, but more often, the result is a badly designed user interface. Employing successful communication strategies and decision-making techniques is therefore an important aspect of interdisciplinary design.

Implications for tool  
design



We think that an innovative tool might help to bridge difficulties in communication and collaboration by providing a "common language" between participants in interdisciplinary contexts. However, a common language is not only determined by tool-usage but also by applied methods and techniques. Nevertheless, support for interdisciplinary collaboration may unleash the potential of creativity and innovation that is hidden between communication issues. In respect to the promoting specific disciplines in favor of others, we think that HCI should take a leading influence on tool design, as its ROI is proven to be most effective.

### 2.4.1 Engineering and Design

Two opposing  
philosophies in user  
interface  
development



One of the main challenges in HCI and interdisciplinary user interface design is the coherence of engineering and design philosophies. While engineering is an applied science that relies heavily on technology, models and testing, design contributes to creative skills and knowledge to achieve results. To understand differences in both philosophies, it helps to look at some definitions:

"[Engineering is...]the use of scientific principles, technical information and imagination in the definition of a mechanical structure, machine or system to perform pre-specified functions with the maximum economy and efficiency."  
J. C. Jones in Preece et al. (1994, p. 352)

"[Design is...]a creative activity - it involves bringing into being something new and useful that has not existed previously." and "simulating what we want to make (or do) before we make (or do) it as many times as may be necessary to feel confident in the final result."  
J. C. Jones in Preece et al. (1994, p. 352)

Both philosophies influence user interface development. In many respects, Software Engineering (SWE) and Usability Engineering (UE) contribute the most to current user interface design practice (Preece et al., 1994, p. 42ff). Nevertheless, design is also a well-established discipline and has great potential to contribute to HCI practice. A clear example is graphic design, which can enhance user experiences when it is applied in the right way. Enhancements in display size and quality as well as advanced graphics technology provide new means of expression on computer screens. High-quality two-dimensional graphics and three-dimensional virtual environments provide extensive interaction possibilities and allow rich user interfaces. Graphic design knows best how to utilize these emerging possibilities.



Considering these recent developments, it is clear that graphic design has gained importance in HCI. According to Laurel (1990, p. 3ff), Carroll (1991, p. 269ff), Preece et al. (1994, p. 42), Dix et al. (2003, p. 505ff), Beyer and Holtzblatt (1997, p. 264ff), Constantine and Lockwood (1999a, p. 185) and Cooper et al. (2007, p. 564 ff) creative design techniques, like sketching, initial brainstorming techniques, and the studio approach of putting design work on display in dedicated design rooms are increasingly getting popular in user interface design teams. Not only does design influence HCI, it is rather a two-way relationship. Engineering is influencing design practice as well. Today it is very common for designers to work with engineering tools like computer-aided design tools to design everything from everyday products to architecture or aircrafts.

Creative techniques are increasingly influencing HCI

One early enthusiastic advocate of creative techniques in interface design is Bill Verplank (Moggridge, 2006, p. 125ff). According to Verplank, "visual brainstorming" and sketching techniques can be used for exploring all kinds of design alternatives as well as to facilitate communication early in the design process. Buxton (2007) argues for extensive use of sketching for designing user experiences. He argues that bringing together two distinct philosophies is possible by employing simple means of expression. Sketching is familiar to all disciplines and widespread over a range of creative industries. It is used for early thoughts but also presentation and communication. According to him, especially conceptual design benefits from agile and informal sketches. In contrast, Constantine and Lockwood (1999a, p. 185) speaks of creativity in design as "creative engineering". According to Constantine, engineering can be both creative and artistic. However, when it comes to graphical art, he constrains its use to respect practical goals and performance required in engineering. He describes the idea in a real-life metaphor:

Examples in creative design approaches

"A bridge must be long enough to span the river and wide enough to carry the traffic.[...]However, a bridge...can also be an elegant and aesthetic work; a creative expression of functional beauty."

(Constantine and Lockwood, 1999a, p. 185)

Constantine's approach to interface design, "Usage-centered design" (Constantine and Lockwood, 1999a) (see Chapter 4.1—"Structured Approaches to User Interface Design") emphasizes on the use of abstract models, known from engineering as a fundamental to creative and original solutions. Every transition from a conceptual model to a physical design therefore is an opportunity for creative visual thinking and innovation (see Chapter 4.1—"Structured Approaches to User Interface Design").

However, with respect to our novel design tool, we think that informal means of creativity in design should be promoted in an attempt to bridge both philosophies. We therefore argue to support both sketching techniques and modeling support for leveraging the potential of innovation in design by still respecting the need for modeling requirements. Consequently, visual representations and brainstorming techniques that facilitate thinking and free association should be smoothly integrated into our tool design. Especially transitions from conceptual to physical design require additional attention.

Implications for tool design



## 2.4.2 Separation of Concerns

"An amazing world. All these people who study what is needed, and all these people who actually build the stuff - two different communities. They don't understand one another, and certainly don't talk to one another."

Interview with Donald Norman in (Laurel, 1990, p. 5)

Designers vs.  
Programmers



One of the most dramatic issues in interdisciplinary design is the communication between human factor specialists, or HCI designers and software professionals, namely software engineers or programmers. Both groups use inherently different jargons and means to describe what they think. Laurel (1990, p. 32) describes these different disciplines as two different cultures. For both cultures to work collaboratively it is necessary to learn and appreciate each other's language, traditions and values. When speaking of software professionals, the highest priority is to get the program running and efficient. Designers or human factor specialists have a different perception: They value the design and look of interfaces.

Bridging the  
disciplines by  
separation of  
concerns

Each perspective has its own right of existence and mixing both philosophies does not solve the problem (Laurel, 1990, p. 32), (Jose, 2003). That is why most research approaches that focus on the bridging of these disciplines center their attention on a separation of concerns. Programmers use their own methods and tools to build a software application, while human factor specialists and designers use dedicated tools to do their business. As described in Jose (2003), research has shown that allowing different disciplines to concentrate on their specific tasks is more efficient than having a continuous exchange. The problem is then reduced to the gap between design and implementation, which has to be bridged at some or several points of time during the development process with communication or specification. Therefore, one essential requirement to this practice is to establish a controlled interference and an appropriate frontier (Jose, 2003).

Shift of  
responsibilities in  
business

Through the separation of concerns, a shift of responsibilities is accompanied. HCI Designers can specify a user interface and then communicate it to developers that actually implement the underlying system. This is especially noteworthy since it opens up new possibilities for businesses. Companies do not necessarily have to employ both designers and programmers internally. Implementing the user interface can be outsourced to suppliers. This may enable smaller companies to focus more on HCI design than they could afford without outsourcing.

Implications for tool  
design



In respect to our investigation, a novel tool design has to support separation of concerns by employing a clear frontier between design and implementation. By respecting the responsibilities of both technical and creative communities, conflicts are avoided. We therefore argue to constrain the responsibilities by utilizing externalizable specifications as a well-defined frontier.

### 2.4.3 Bridging the Gap in Practice

Solutions have to  
focus on technology  
and communication

There are several attempts on bridging the gap between designers and programmers in work practice (Jose, 2003). The primary goal to promote cooperation between these two communities leads to basic requirements in software technology and means of communication. Software tools and programming languages have to be compatible with both disciplines to allow trouble-free exchange. A mixture of formats handicaps sharing of design artifacts in both directions. However, this technical exchange of work is not enough, since actors from both disciplines also have to understand the backgrounds. Without knowing why a specific artifact was designed the way it is, programmers might not understand the purpose and implementation will fail.

#### Technology and Tools

Separation of UI  
and code

One early effort on establishing supporting means of cooperation in technology is the research on User Interface Management Systems (UIMS) (Preece et al., 1994, p. 581 ff). These systems are aiming at providing an integrated development environment for HCI designers, which is separated from the application code itself. A UIMS therefore is an interactive software application that facilitates the separation of responsibilities between the user interface and the implementation code at runtime. The application is then reduced to the functional

“work”, while the UIMS handles user inputs. Deborah Hix, who spent over 10 years in UIMS research, describes such a system as follows:

“It should support all phases of the interface development process, beginning with activities like task analysis and functional analysis.[...] [it] would do all those things and then go all the way through, helping with the design, implementation, and even usability evaluation and refinement of the user interface. So probably, by my definition, a UIMS doesn’t exist today.”

Interview with Deborah Hix in Preece et al. (1994, p. 593)

UIMS implementations are rare today and do not nearly include such a wide range of functionality (Jose, 2003). Attempts on providing support for this kind of separation in application code, were for a long time reduced to implementation patterns, like the Model-View-Controller (MVC) design pattern (Goldberg and Robson, 1983) or the ARCH project (Sigchi, 1992).

Today’s software development tools offer advanced methods to separate the UI from the application code (see Chapter 3.2—“State-of-the-Art Tool Support”). Nearly all popular Integrated Development Environments (IDEs), like Microsoft’s (MS) Visual Studio<sup>2</sup> or the Eclipse Project<sup>3</sup> offer Graphical User Interface Builders, also known as “GUI Designers”. These tools support designers in creating user interfaces without having to implement code manually. Created designs can be shared or passed on to programmers. Yet, these tools are very formal and do not facilitate creativity and early activities, like task analysis, functional analysis as well as functionality like evaluation support (e.g through prototyping). Nevertheless, recent developments contribute to a better interoperability of design and implementation. UI description formats, which are based on the Extensible Markup Language (XML), like Microsoft’s “Extensible Application Markup Language” (XAML) (Januszewski and Rodriguez, 2007) or the “User Interface eXtensible Markup Language” (UsiXML) (Vanderdonck et al., 2004) enhance accessibility and iterations between design and code. Some of these description languages like the “User Interface Markup Language” (UIML) (Phanouriou, 2000) also allow platform and device independent development.

A new generation of tool support

Based on this brief review of current technology, we think that current UI technologies and formats provide effective means of separating implementation from design. Description formats for interfaces enhance accessibility and adaptability to device independent development. Providing a XML-based structure for specification of actual interface looks is an appealing notion for our specification demands. Consequently, we argue to support commonly utilized and widespread UI description languages as a part of our specification tool. However, the high degree of formal expression within traditional GUI designers has to be avoided in respect to demands on creativity.

Implications for tool design



## Communication & Specification

While technology helps to remove burdens in exchange of work and cooperation, facilitating communication is still essential. As mentioned before (see Chapter 1—“Introduction”), communication issues are one of the most important factors in interdisciplinary design. Misunderstanding is a threat to project goals and is therefore highly cost-intensive. Iterative user interface development approaches require repetitive decision-making over developed alternatives or advanced designs. Communication forms in practice are usually verbal, written or visual.

Three different forms of communication

The most widespread form of communication in business are team meetings (Preece et al., 1994; Bennett and Karat, 1994, p. 482). Team members from all disciplines meet to propose

Team meetings

<sup>2</sup><http://msdn2.microsoft.com/de-de/vstudio/aa700919.aspx>

<sup>3</sup><http://www.eclipse.org>

problems or changes. Proposals are also spread to all actors in writing before the meeting. This allows all team members to think about the problems before eventually discussing them. The proposed problems are then discussed for a while and conclusions are made if consensus is achieved. During meetings, creativity plays an immense role. The group tries to invent solutions to the proposed problems or changes. The dynamic of this process often leads to new and innovative concepts (Bennett and Karat, 1994). Results and decisions or potential sticky points are usually noted by one or more actors in the team for later reference.

Written communication and specifications are not adequate



Communication between designers and programmers can also be written. Especially, when the implementation is outsourced to external developers, written user interface specifications are frequently employed. Specifications are an established form of communication in engineering, manufacturing and business when it comes to agree on specific requirements between client and supplier. They are usually formal text-based documents that describe all aspects of a user interface. However, specification of user interfaces is not just a matter of defining the UIs presentation. A relevant feature is the description of its functional behavior (Hussey, 1996). The activity of creating this form of description is a highly anticipated topic in research. The main issues lie in the description of the interface and its behavior by maintaining a form of abstraction that makes it possible to understand it by suppliers, but does not require actual implementation by the client (Abowd, 1991). Early research in formal specification methods has been a major concern of the HCI Group at York University since the 1980s (Dix, 1987). Recent developments in this subject of research show a trend to using less formal methods, like models and prototypes for specifications (Dix et al., 2003, p. 313) (Hussey, 1996), (Jose, 2003). According to Hussey (1996) the main reason for this turn is that formally written specifications have significant disadvantages in readability and traceability due to a lack of abstraction.

Visual communication through models

To profit from the power of abstraction in communication, some approaches in interdisciplinary design are utilizing visual or text-based models of requirements as informal specification to describe the behavior of a user interface (see Chapter 4.1.2—“Widely adopted Structured Approaches”). User Interface Modeling - as this approach is frequently called - is also known as model-based<sup>4</sup> or model-driven<sup>5</sup> method and is well established in practice (Constantine et al., 2003, p. 6). Models focus inherently on the conceptual layer of user interface design (see Chapter 4.1—“Structured Approaches to User Interface Design”). They are a representation of how users interact with the software and how the system responds accordingly.

“A good model is accurate enough to reflect the features of the system being modeled, but simple enough to avoid confusion.”  
(Benyon et al., 1999, p. 6)

According to Benyon et al. (1999, p. 6ff) a major benefit of employing visual models in development processes is their ability of expressing different degrees of abstraction, which enhances interdisciplinary communication. While specifications are giving a detailed view on requirements, models are hiding some details, so that only the important aspects stand out. The use of models also especially fits interdisciplinary user interface development, since modeling is used in both of the most important disciplines, HCI and Software Engineering (Benyon et al., 1999, p. 6). However, choosing the right models in an interdisciplinary context is crucial to avoid misunderstandings (Constantine et al., 2003, p. 6). HCI research came up with a wide variety of models, ranging from scenarios, user models and activity models to task models (see Chapter 4.1.2—“Widely adopted Structured Approaches”). In Software Engineering, many models are graphical and have diagram notations, but may employ computer simulations or animations. In HCI various forms of media, like pictures and video might be used as models (Benyon et al., 1999, p. 6).

<sup>4</sup>Models are used primarily for problem decomposition and communication

<sup>5</sup>Computer-supported models are used to automatically generate code

Interactive prototypes may be considered as models, too. They offer a clear picture on how the users interact with the system, but still provide different levels of abstraction. The grades of abstraction in prototyping range from sketching techniques, over wireframes and mock-ups to high detailed interactive simulations (see Chapter 4.2.1—“Tasks, Techniques and Artifacts”).

Visual communication through interactive prototypes

“Prototypes are a useful aid when discussing ideas with stakeholders; they are a communication device among team members, and are an effective way to test out ideas for yourself.”  
(Sharp et al., 2007, p. 530)

Prototypes are primarily used to communicate designs to users and team members to evaluate their usability. According to Sharp et al. (2007, p. 530) the process of creating a prototype encourages reflection in design and therefore is an important aspect of interface design itself. While low-fidelity prototypes are employed early in the design process to evaluate alternatives, high-fidelity prototypes are often the last step of testing a final design before the writing of any code. Depending on the fidelity of a prototype, there is a wide range of prototyping tools that allow creating interactive simulations without having to write any code. Most available prototyping tools are specialized on high-fidelity prototypes (see Chapter 4.2.1—“Tasks, Techniques and Artifacts”), while low-fidelity prototypes are often created without tool support, like sketching techniques and storyboarding (see Chapter 4.2.1—“Tasks, Techniques and Artifacts”).

In reference to our research goals, we think that our design should especially focus on communication issues as it is dedicated to specification. Nevertheless, verbal communication within a team of designers is important to exchange thoughts and ideas during the design process. Therefore, our tool design has to support both forms of communication, namely “internal” communication and “external” specification. As internal communication is usually conducted in meeting rooms, we think that adequate presentation capabilities are required for our tool design. However, we think that written specifications for external communication have major drawbacks and hamper the overall information exchange. Instead, we argue to employ visual specification techniques, like visual models and interactive prototypes. Therefore, our tool has to support both visual forms of externalization as part of its specification. However, as we want to focus on separating implementation and design, model-driven approaches should be avoided as they are primarily used to generate code and therefore tend to impose formal means of expression.

Implications for tool design



## 2.5 Treasuring Design Experience

An important goal in corporate UI design projects is to keep track of decisions that were made throughout the process. As design is also based on knowledge, experiences from past developments may help in future operations. A major objective in this respect is to create a corporate memory of design knowledge. According to Borchers (2001, p. 5) such a repository can have following benefits: It helps to avoid repeating errors, which occurred during past projects and it can help introduce new members to a design context. Consequently, it is also a strong business case to keep knowledge as employees may leave the company and expertise is lost. In practice, there are several ways how to capture design experience: design patterns, guidelines or style guides and design rationale techniques.

Keeping track of design decisions

Capturing  
experience through  
design patterns

Just like models, design patterns are frequently used in engineering. They are general reusable solutions to commonly occurring problems in design. They can be regarded as a template on how to solve a problem. Borchers (2001) gives a brief definition:

“Put simply, a design pattern is a structured textual and graphical description of a proven solution to a recurring design problem.”  
(Borchers, 2001, p. 7)

The concept of patterns in software technology originated from Gamma et al. (1995) and is widely adopted in Software Engineering practice. Already Norman and Draper (1986) were influenced by the idea of patterns in architectural design as designing their approach to user interface design “User-centered design”. According to Borchers (2001, p. 8) these patterns are also applicable in HCI practice. He shows that they are suitable to capture design experience for future reference. In HCI they can take different forms, depending on the specific stages in user interface development processes (Dix et al., 2003, p. 285ff). While patterns in early stages may be text-based and manifest goals and tradeoffs or certain vocabulary, in later stages they may reach from interaction patterns (see Interaction Patterns in Chapter 5.2—“Collecting Concepts - Interaction & Visualization”) or prototypes to evaluation feedback and discovered solutions to problems (Borchers, 2001, p. 58ff). As patterns only capture solutions, or “lessons-learned”, they do not capture bad decisions, as a design rationale - which is described later - would do. Jose (2003) regards patterns also as a useful technique to interdisciplinary communication, since the basic concept is shared among different disciplines. According to Dix et al. (2003, p. 285) patterns are a recent addition to HCI and are not yet proven in practice.

Guidelines and  
Styleguides

User interface guidelines or styleguides are used as a textual form of describing interface design rules based on experience. They can be generally divided into two categories: abstract guidelines or concrete styleguides. Abstract guidelines, like the Eight Golden Rules of Interface Design (Shneiderman, 1992, p. 59ff), (Dix et al., 2003, p. 282f) or Nielsen’s heuristics (Nielsen, 1994, p. 91) or Norman’s seven principles (Norman, 2002, p. 45ff) are broad guidelines and may not be applicable to all domains. Concrete styleguides, as the Macintosh Human Interface Guideline<sup>6</sup> or the Microsoft Windows User Experience Guide<sup>7</sup> are tailored to the use of certain toolkits or programming languages. As a result, they potentially harm creativity in design (Borchers, 2001, p. 6). Concrete guidelines are also often employed in corporate environments to manifest corporate interaction styles and to maintain consistency among product families.

Design Rationale  
techniques facilitate  
knowledge

Another approach to capture experience in design is called “Design Rationale”. This term was first introduced by Carroll (1991, p. 80f) and was advanced in HCI research in various forms (Carroll, 1997, p. 73). According to Carroll, this principle is defined as follows:

“A design rationale is a detailed description of the history and meaning of an artifact. 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)

<sup>6</sup><http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/OSXHIGuidelines.pdf>

<sup>7</sup><http://msdn2.microsoft.com/en-us/library/aa511258.aspx>



Carroll (1991, p. 80ff) also gives reasons why employing a design rationale in user interface development is useful. Maintaining a comprehensive design rationale offers a chance to make HCI work more analytical and useful. Keeping track of many possible decisions and histories is an effective tool for evaluating tradeoffs in future designs. Carroll (1991) describes the level of detail in a design rationale as "expandable". This makes it especially suitable for constructing descriptions and abstractions within the context of evolution in design:

"Because design rationale is understood as a workspace for representation, a dynamic view of an artifact, it is also particularly well suited to the iterative nature of the design process."  
(Carroll, 1991, p. 81)

According to Carroll (1997, p. 73) and Moran and Carroll (1996, p. 267ff), design rationale is an experimental subject in HCI research and has not yet been efficiently adopted in practice. State-of-the-art research work is concerned with assessing and supporting its efficiency. According to Carroll (1997, p. 73) empirical studies are investigating how these techniques can be learned and used in project teams. Other work is focused on tool support in the creation and assessment of design rationales. Experimental tool-supported approaches mentioned in Rosson and Carroll (1995) and Moran and Carroll (1996, p. 267ff) usually integrate iterative model-based development and use structured or hierarchical methods to organize temporal design knowledge. Carroll (1997, p. 73) also proposes the use of a design rationale for interdisciplinary communication, but restrains its usefulness if language differences among stakeholders prevail. However, past research within our group on design rationale techniques (ProUse<sup>8</sup>) provides a strong fundamental to required functionality.

Lack of techniques  
and tools  
supporting design  
rationale

"Design rationale can be a language for stakeholders in the design, but these different stakeholders often speak different disciplinary languages, are motivated by different values, and see different technical issues when looking at the 'same' design problem."  
(Carroll, 1997, p. 73)

Primarily the lack of widespread tool support and the complexity of informations, artifacts and relations restrains HCI practice from employing design rationale techniques. However, Carroll (1997, p. 74) makes out great potential in new technologies, like advancements in communication and computer supported collaboration that may help to make efficient use of design rationale methods in future HCI development (see 5.2.4—"Collaboration"). Currently employed solutions focus on hypertext techniques to link artifacts with decisions or patterns.

However, we think that keeping track of experiences throughout design projects is an essential part of a specification. Eventually, an external actor that implements a design based on a specification might be interested in the rationale that led to this solution. Understanding solutions often implicates an understanding of the underlying problems. Nevertheless, interaction patterns and general guidelines provide useful "templates" on how to solve problems. Consequently, integration of both textual guidelines, interaction patterns as well as design rationale techniques have to be investigated for tool design. Hypertext techniques that allow linking and tracing design decisions may provide useful support.

Implications for tool  
design



<sup>8</sup><http://hci.uni-konstanz.de/index.php?a=research#>

## 2.6 Summary

The following lists summarize our findings from investigation of theoretical foundations in respect to the research at hand. After presenting this summary, the next chapter will investigate state-of-the-art tool support and interesting research projects that provided inspiration for later design. Thereafter, a detailed requirements analysis is presented.

### 2.6.1 Shortcomings

Referencing the preceding detailed descriptions, the following list summarizes identified shortcomings and drawbacks:



- User Interface Design involves multiple disciplines. Actors from various disciplines have specific backgrounds and perceptions. Since actors have different views to the subject matter, they are using diverse methods and tools during design processes. Supporting tools are also specific to certain actor's needs and modeling languages. The lack of a common course of action and the use of inappropriate, incompatible terminologies and modeling languages prevent transparency and traceability in design (see Chapter 2.4—"Interdisciplinary Design").



- Innovations and creativity in design are harmed by misunderstandings between the opposing philosophies Engineering and Design. Design promotes creativity, while engineering has its focus on functionality (see Chapter 2.4.1—"Engineering and Design"). Consequently, formal tools used in engineering are not interoperable with creative techniques. Separating concerns in development processes leads to a gap in technology and communication (see Chapter 2.4.3—"Bridging the Gap in Practice"). The lack of support for moving from conceptual design space (e.g. models) to physical design (UI design) hampers innovation and traceability (see Chapter 2.4.3—"Technology and Tools"). Insufficient means of communication and cooperation may lead to misunderstandings and wrong design decisions (see Chapter 2.4.3—"Communication & Specification").






- There is a lack of support in tracing design requirements, alternatives and iterative developments as well as decisions made during the design process (see Chapter 2.5—"Treasuring Design Experience"). To achieve a comprehensive design rationale, decisions and feedback on design alternatives have to be recorded for future reference. Tool support for visualization and communication of design experience in the form of networks of requirements and modeling artifacts is not yet adopted in practice. The difficulty in communicating abstract and detailed forms of design experience leads to the burial of critical information in documents that are difficult to browse (see Chapter 2.5—"Treasuring Design Experience"). The resulting misconceptions lead to costly change requests and iterations, which raise budgets and endanger project goals.



## 2.6.2 Implications for Tool Support

Based on the described shortcomings this list summarized ideas for the design of effective tool-support in user interface design.

- A tool-based approach should effectively address the difficulties in technology and communication among disciplines that participate in the design process. It has to respect the different perceptions to the subject matter, while still maintaining a common understanding of design artifacts. Nevertheless, a tool-based approach should favor HCI techniques given that their ROI in user interface design is proven more effective than that of other disciplines (see Chapter 2.4—“Interdisciplinary Design”). Furthermore, it should support technical developments that improve separation of concerns. Eventually, it should provide means to facilitate team-communication and decision-making as well as specifications in various forms of abstraction, like models and prototypes. 
- A new tool-approach should promote creativity in design by supporting informal methods. It should narrow the gap between conceptual modeling and physical design to facilitate innovation and traceability (see Chapter 2.4.1—“Engineering and Design”). Actors should also not be hindered to participate in design by having to adapt to a completely new work style. A novel tool design has to respect these issues by keeping interaction as simple as possible and by employing concepts that are, familiar to all actors and compatible to current work practice. 
- An advanced UI specification tool should allow the tracing of design requirements, alternatives and iterative developments as well as decisions made during the design process for future reference (see Chapter 2.5—“Treasuring Design Experience”) and for externalization (see Chapter 2.4.3—“Communication & Specification”). It has to offer adequate means of visualizing design experience in the form of visual requirement networks to enhance traceability and to allow actors to relate modeling artifacts in context (see Chapter 2.5—“Treasuring Design Experience” and Chapter 2.6.1—“Shortcomings”). Furthermore, integrated design rationale techniques should support abstractions to make design knowledge available to different interdisciplinary actors and new team members. 



## Chapter 3

# Related Work

*“If I have seen further it is  
by standing on the shoulders of giants.”*

—Isaac Newton

This chapter describes lessons from previous research before introducing a cross-sectional comparison of state-of-the-art software tools that are employed in practice. Thereafter, some innovative tools are described which are related to the research at hand. Eventually, a comparative framework is presented to clarify the need for a novel tool approach.

Outlook  
Outlook

### 3.1 A Lesson in History

According to Myers et al. (2000), research in user interface software had a dramatical impact on today’s design practice. Virtually all software applications today are built using some form of a user interface tool. Most of these tools have their origins in computer-science related research within the 1970s to the 1990s. Consequently, these tools were focusing on the technological constrains and requirements which were necessary to design user interfaces for the popular operating systems, like Microsoft Windows or MacOS (Macintosh Operating System). While there were quite different approaches in the beginning, today nearly all these applications offer the same functionality, which enables users to build user interfaces based on the WIMP (Windows, Icons, Menus, Pointing device) paradigm and the desktop metaphor.

History of UI Design  
Tool research

Whereas this uniformity imposed several benefits, it also led to stagnation. The consistency of user interface designs makes it easier for users to adapt to similar interfaces. An interface that looks familiar does not require many skills to be used efficiently in a short learning time. Nevertheless, this uniformity led to stagnation in leveraging new possibilities to create more useful and more innovative user interfaces. Additionally, the rise of new interface technologies in “Hypertext”, very small or very large screens in “ubiquitous computing” as well new input devices offer a wide range of possibilities, which are not addressed efficiently by common UI tools. Handheld applications, for example cannot employ the typical desktop metaphor based on their limited screen size. In contrast, wall-size displays are also not suitable to employ commonly used means of interaction.

Uniformity and  
rising need for  
innovation

"The implication of these changes is that we can expect a dramatic increase in the diversity of both the types of computing devices in use, and the task contexts in which they operate. This, in turn, implies that we are poised for a major change in user interfaces, and with it dramatic new needs for tools to build those interfaces."

(Myers et al., 2000, p. 5)

Diversity of tools  
and trends

These new requirements to interface development and tool usage are accompanied with an increasingly rising interest in HCI research. New theoretical methods to improve user interface development required novel supporting tools. Historically, the most successful UI tools focused on a particular part of the user interface or on specific phases within the development process (Nunes and Campos, 2004). While window managers and toolkits, scripting languages, hypertext editors, and other frameworks focus on the executable part of the interface and therefore formalize the interface, object-oriented programming languages facilitated efficient methods to separate the interface from the code. Model-driven interface approaches are even more constraining innovation in design. Nevertheless, tools that support early phases of the design process were also adapting to the demands for quick and iterative development. Tools with lower threshold provide informal and more abstract means of expression, which are not provided by formal tools. Prototyping tools, drawing and modeling support are increasingly reducing the amount of programming required to build interfaces and enable human factor specialists to actively participate in the design process. However, most tools were not successfully adopted as various transitions between tool-usage were emerging (Nunes and Campos, 2004). The need for informal expression (low-threshold) by still having powerful tools to design details (high-ceiling) within design processes is yet not effectively addressed by the diversity of tools and formats that can be found in practice.

### 3.2 State-of-the-Art Tool Support

Tool-support in  
practice

As previously described, a wide range of tools is employed in user interface design processes. Based on our findings in observing work practice (Memmel and Reiterer, 2008; Memmel et al., 2007e) and findings of Campos and Nunes (2007), Maguire (2001) and Ambler and Jeffries (2002) we present a cross-sectional overview on state-of-the-art tool usage in design practice. For a more detailed analysis of tool-usage in respect to specification, see König (2008). Frequently employed tools are therefore diagramming tools, graphics tools and dedicated UI development tools (Campos and Nunes, 2007). As a quite novel trend, dedicated prototyping software is emerging in design practice, especially in web development. These applications are focusing on supporting simplified development by providing graphical programming environments that allow building functional prototypes rapidly. Additionally, specification techniques are employed that allow to efficiently generate text-based specifications. In the following, these tools are briefly introduced and then analyzed in reference to their functionality along with common UI design tools.

Traditionally used  
tools

Table 3.1 presents an overview on currently employed tools in practice. The tools are plotted against categories of functionality and marked with indicators that represent a subjective estimation in "n/a (not applicable) or okay", "good" and "excellent". Categories of functionality are based on requirements that are discussed in detail in chapter 4—"Analysis". Dedicated UI development tools, like GUI Builders or XAML Editors are good in simple user interface design, but creativity is seriously harmed as existing concepts are heavily promoted. While development tools support basic prototyping, specification and collaboration, they are not excelling in these categories. Additionally a lack of modeling support, presentation functionality and process support demands for additional supplemental tools. Graphics tools, like Adobe Photoshop<sup>1</sup> are primarily employed to facilitate graphical design and presentation of designs. As means of expression are virtually unlimited, there are no serious constraints to creative design. Nevertheless, these tools lack support in the remaining cate-

<sup>1</sup><http://www.adobe.com/products/photoshop>

Category	Graphical Programming Axure / iRise	Diagramming Tools MS Visio	Graphics Tools Adobe Photoshop	Development Tools GUI Builder / XAML Editor
<b>Design</b> Creative UI Design				
<b>Modeling</b> Conceptual Design				
<b>Prototyping</b> UI Simulation				
<b>Presentation</b> Externalize Design				
<b>Specification</b> Communicate Design				
<b>Collaboration</b> Teamwork support				
<b>Process Support</b> Rationale, Artifact Management				

n/a or okay    good    excellent

**Table 3.1:** Evaluation of state-of-the-art tool landscape

gories. Diagramming tools, like MS Visio<sup>2</sup> are employed during conceptual design phases, but may also be used to model simple user interface designs. Arguably, they may also be used to externalize designs. Eventually, graphical UI programming environments, like Axure or iRise support design processes in many important categories.

A new generation of prototyping and UI specification tools is exemplified by iRise<sup>3</sup> and Axure<sup>4</sup>. König (2008) gives insight into various similar tools. They primarily focus on providing graphical programming functionality to enable designers to build functional prototypes. Nevertheless, they do not offer adequate conceptual design support as well as process support. Consequently, these applications tend to be used when the real design work is already done and it needs to be externalized. Features that are similar to those in development tools facilitate design that is a bit more flexible and offers some additional means of expression like simple graphics manipulation functionality. Graphical programming tools excel in prototyping, presentation and specification functionality. Collaboration is also partly supported, but restrained to asynchronous merging of development files, like in development tools. Nevertheless, this new generation of tools supports many aspects of the required functionality. However, efficient process support, like artifact management or design rationale techniques remain elusive. In addition, the formal means of expression as well as lack of support for conceptual modeling tasks hamper innovative interface design especially in early design phases.

A new generation in graphical programming

<sup>2</sup><http://www.visio.com/>

<sup>3</sup><http://www.irise.com>

<sup>4</sup><http://www.axure.com>

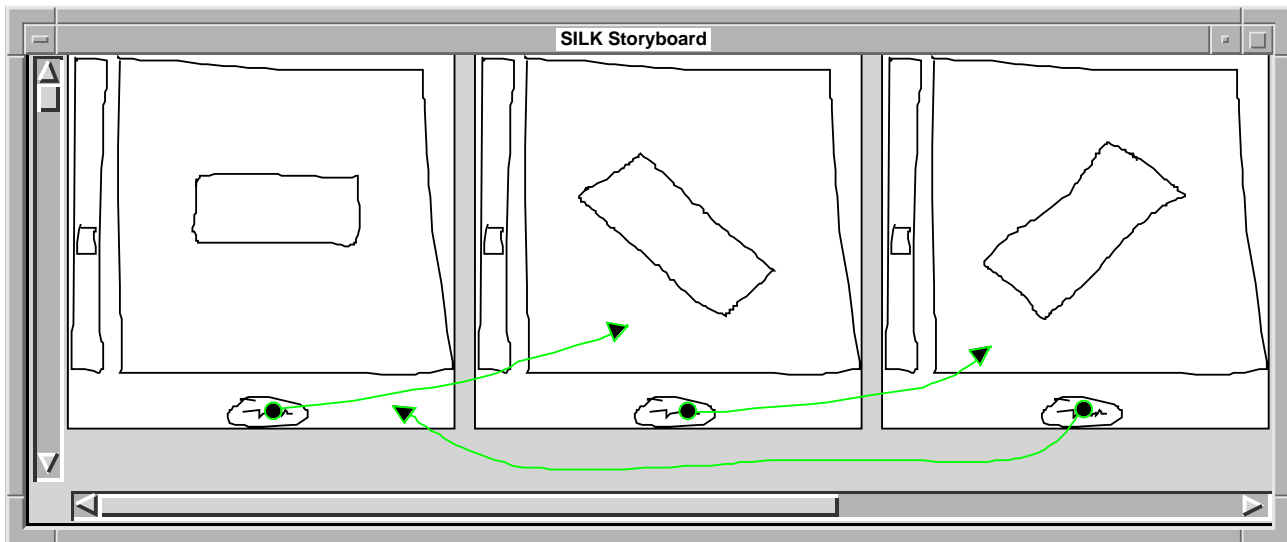


Figure 3.1: SILK (Sketching Interfaces Like Crazy)(Landay and Myers, 1995a, p. 3)

### 3.3 Related Research

Two trends in UI  
tool research

By looking at the current tool landscape and the lack of support within certain categories of Table 3.1, it is obvious that experimental research projects focus on the support of early design phases, like modeling and conceptual design as well as process support. Therefore, informal means of expression, which are essential in early process steps, are increasingly investigated. Experimental techniques and tools to support the nature of design processes in contrast focus on traceability of process steps and transitions by combining conceptual modeling with interface design or by offering novel ways of specification. As these two trends in research have significant impact on the research at hand, important and acknowledged related research projects are introduced in the following.

#### 3.3.1 Supporting early Design Phases by Sketching

Programming by  
sketching

Landay and Myers (1995a) introduced a design environment that focused on providing informal 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 Crazy) that allows designers to sketch interfaces using an electronic tablet and stylus. In contrast to real sketches, this electronic sketch is interactive, which allows specifying simple behaviors of the interface. While early experimental prototypes were limited to a single screen for sketching, the technique was improved by utilizing the concept of storyboards. A number of subsequent interface designs can then be linked with connecting lines, which are used to simulate interface flow. Designers are enabled to specify how the screens should change in respond to user actions.

Figure 3.1 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 behavior of rotating an object by pressing a button. Landay and Myers (1995a) argue that their approach preserves the benefits of pen and paper, namely that drawings can be produced very quickly and the medium of representation is still flexible. According to Landay and Myers (1995a), the informal "sketchy" look of the screens stimulates creativity and innovation in design. Landay and Myers (1995b, 2001), Landay (1996), Lin (1999) and present incremental advancements to the SILK concept by adding automatic recognition of sketches, conditional transitions and by improving performance and considerations of evaluation feedback from field studies.

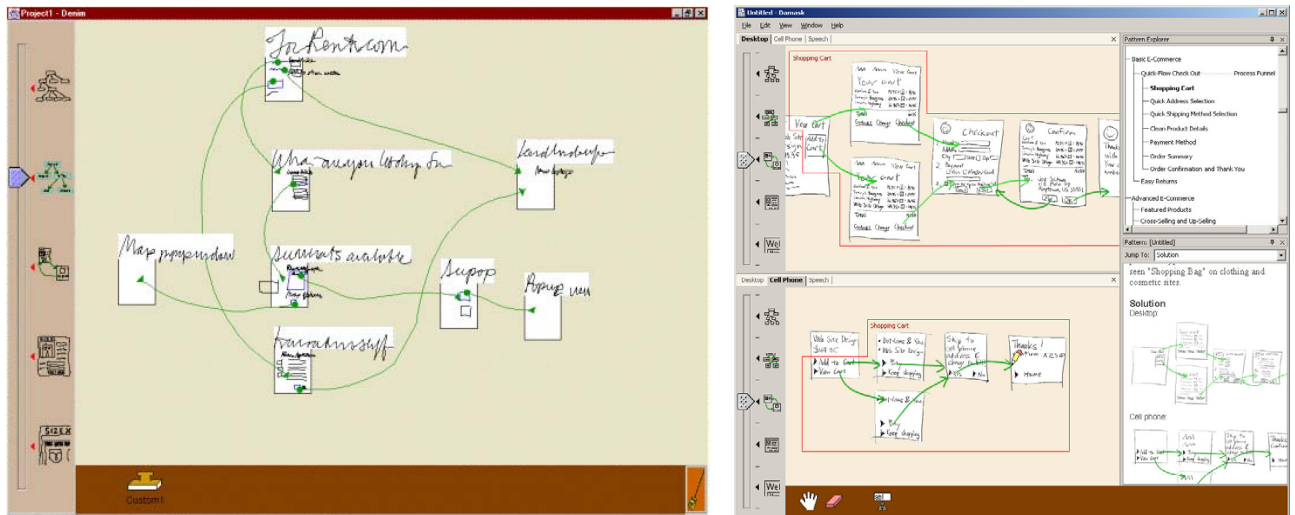


Figure 3.2: DENIM (Lin et al., 2001) (left) and DAMASK (Lin, 2003) (right)

The concept of simulation by electronic storyboarding is not entirely new. Some concepts of SILK are taken from HyperCard that was first released by Apple Computers in 1987. HyperCard is an easy-to-use programming environment that is based on a series of cards that are arranged into stacks. Cards can then be linked to each other, like hypertext links to specify transitions between cards and consequently behavior. HyperCard supports text, pictures, graphical shapes as well as audio and video, which gave it enough power to built real applications in a simple manner. Before HyperCard, programming was more or less exclusive for professional programmers. Landay and Myers (1995a) argue that HyperCard is not suitable for rapid prototyping in early design steps, as its direct manipulation features and scripting requirements hamper the fluidity of paper-based storyboarding. SILK improves this concept by employing sketching consequently throughout the interface.

The role of HyperCard

The concept of SILK was advanced in DENIM (Lin, 1999; Lin et al., 2001) and DAMASK (Lin, 2003). The improvements of DENIM in respect to SILK were based on an ethnographic 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. Additionally, pie menus were added to enhance interaction in the zoomable canvas and a dedicated "run mode" enables users to experience the interface flow in a separate browser window. DAMASK in contrast adds support for sketching interface representations for several devices, like PCs, cellphones or voice. Additionally, it introduced an interaction pattern browser that provides predefined behaviors for specific purposes. Figure 3.2 shows the interfaces of DENIM (left) and DAMASK (right). While the main part of the interface is a zoomable canvas, actual zooming is accomplished by a range slider on the left side of the interface. Multiple representations of cross-platform interfaces in DAMASK are stacked in tabs.

Improvements with zoom and patterns

The introduced sketching design tools provide informal means of expression in early process steps and promote creativity in design, but are limited to very abstract representations. In addition, the incremental nature of the design process is not considered, as new sketches have to be started from scratch. The lack of real-life look & feel of the interface makes evaluation under real-life conditions hard. Besides, these tools cover only a very small scope of the overall design process. Requirements, conceptual modeling and high-fidelity design are neglected. Therefore, additional transitions in tool-usage are necessary. Overall process complexity still remains or may even rise as yet another tool is added to the production chain.

Limitations of sketching design tools

A research project that is narrowing the work transition from early sketches to high fidelity designs is SketchiXML (Coyette et al., 2007b). Based on a trainable sketch recognizer (Coyette et al., 2007a), SketchiXML is turning scanned paper sketches of UI states automat-

SketchiXML

ically to XML documents that can be reused in graphical XML editors during subsequent development steps. By adding support for various interface domains, like portable devices and mobile phones, SketchiXML utilizes a novel XML specification format, similar to Microsoft's XAML that supports cross-platform and multiple device development, named UsiXML (Coyette, 2007). SketchiXML employs intermediate views to allow user interaction between certain work transitions. The respected steps are based on the CAMELEON framework (Calvary et al., 2003), which divides interface design for multiple devices in four layers: Tasks & Concept, Abstract User Interface, Concrete User Interface and Final User Interface. According to Coyette et al. (2007b), SketchiXML supports transitions from "Abstract User Interface" to "Final User Interface". Consequently, "Tasks & Concepts" are neglected and have to be bridged with other tools. Nevertheless, employing a XML based specification language for intermediate and final designs is an appealing option as it facilitates interoperability and accessibility throughout the process.

### 3.3.2 Hybrid Solutions to ease Work Transitions

Hybrid solutions to ease work transitions

In the face of these issues regarding work transitions and tool transitions, a range of experimental research projects aim on bridging these transitions in interface design by including multiple tasks into a single solution. Another major goal of these tools is also to ease the transitions in design practice by exploiting relationships among artifacts and consequently foster traceability. Examples of these "hybrid" solutions are DiaMODL (Traetteberg, 2004), CanonSketch<sup>5</sup> & TaskSketch<sup>6</sup> (Campos, 2005b) as well as WinSketch<sup>7</sup>. In the following, these tool approaches are described briefly, as they strongly influenced the research at hand.

DiaMODL

Traetteberg (2004) introduced a model based development environment, based on MS Visio, DiaMODL. He argues that model based development of user interfaces is not successfully adopted in practice, because of the neglected role of models and the negative results in design processes. Traetteberg (2004) explains this fact by the strict separation of UML tools, used for modeling and implementation and GUI Builders that are used to build actual interfaces. To bridge this separation, Traetteberg (2004) presents a hybrid solution named DiaMODL.

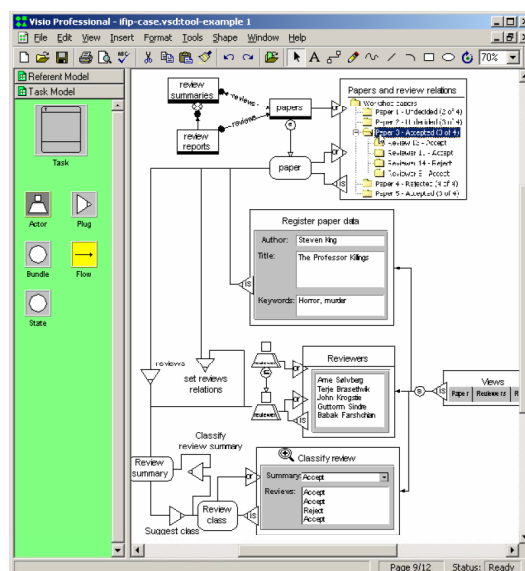


Figure 3.3: DiaMODL (Traetteberg, 2004)

<sup>5</sup><http://dme.uma.pt/projects/canonsketch/>

<sup>6</sup><http://dme.uma.pt/projects/canonsketch/>

<sup>7</sup><http://apus.uma.pt/winsketch>



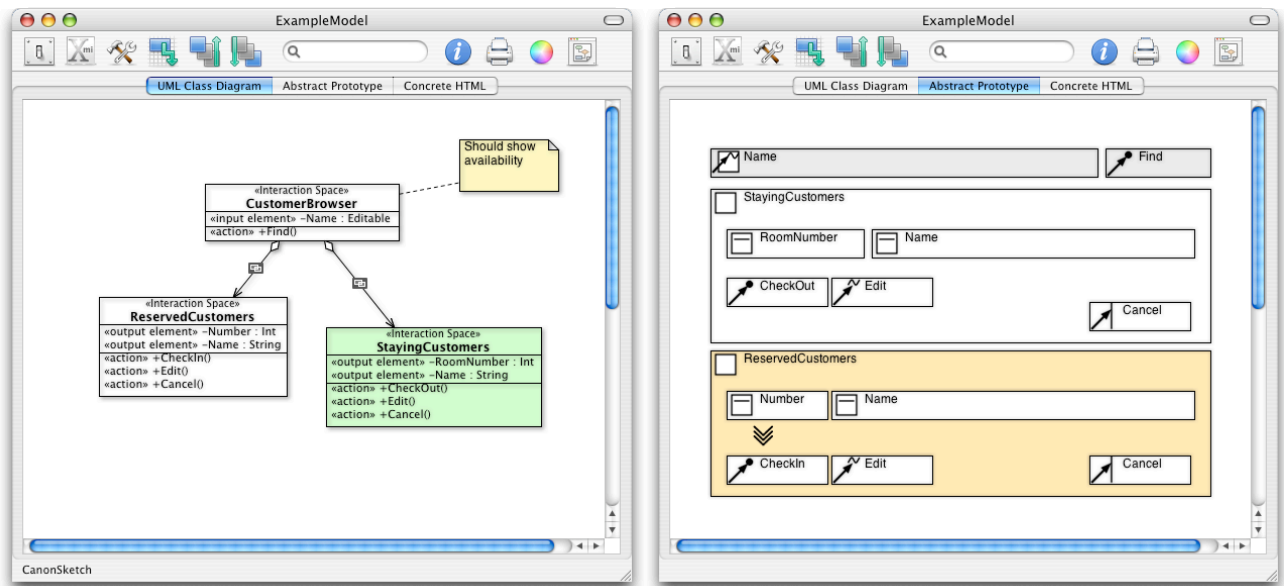


Figure 3.4: CanonSketch - UML view (left) and canonical abstract prototype (right) (Campos, 2005b)

Figure 3.3 shows the interface of this tool. The basic idea is to utilize mockups of interface components and then connecting them with flowcharts that reveal the underlying functionality. To achieve this, Traetteberg (2004) focuses on the logical structure of interactions that are displayed next to interface mockups. Relations and flows between these distinct states of the UI are also visualized with connecting lines. By using such a hybrid view, Traetteberg (2004) states that his notation may be used as a more abstract form of specification and argues that it is fully compatible with implementation methods that are found in development processes throughout different industries. As a replacement for detailed mockups, Traetteberg (2004) proposes to employ semi-formal more abstract prototypes, like Constantine's abstract canonical prototyping components (Constantine, 2003).

Campos and Nunes (2004) presented CanonSketch and TaskSketch, a tool-based approach that builds upon the idea of a hybrid view on user interface design as canonical prototypes and underlying models. By considering current work style in practice, interdisciplinary issues and communication issues, Campos (2005b) proposes a combination of a novel model based notation that characterizes the levels of abstraction as well as an interdisciplinary communication. Therefore, a work style model is used to identify transitions in work practice and implications for a notation that effectively addresses these. In respect to the nature of the design process and collaborative constrains, CanonSketch introduces three distinct levels of detail for its notation: UML (Unified Modeling Language) diagrams, abstract canonical prototypes as well as concrete HTML (HyperText Markup Language). As the UML in its original form is not compatible for user interface development, Campos (2005b) utilized the Wisdom method (Nunes, 2001), an extension to the UML for the design of interactive systems.

CanonSketch &  
TaskSketch

Figure 3.4 shows the interface displaying the same model at different abstraction views. While the UML diagram (left) is used to model properties and containment, the abstract prototype (right) displays these components in the form of an abstract prototype. As both notations are matched with a compromising notation, updates between views are synchronized as the user switches to a different view. As a conceptual model for the user interface has to exist before actually designing tasks, CanonSketch lacks support for early process phases.

Different views on  
abstractions

A closely related tool presented by Campos (2005a,b) is TaskSketch. In contrast to CanonSketch, this tool is focused on linking and tracing use cases for requirement analysis, before actually designing the user interface with CanonSketch. Again, the Wisdom approach is utilized to adapt use cases and activity diagrams to interface design.

Tracing use cases

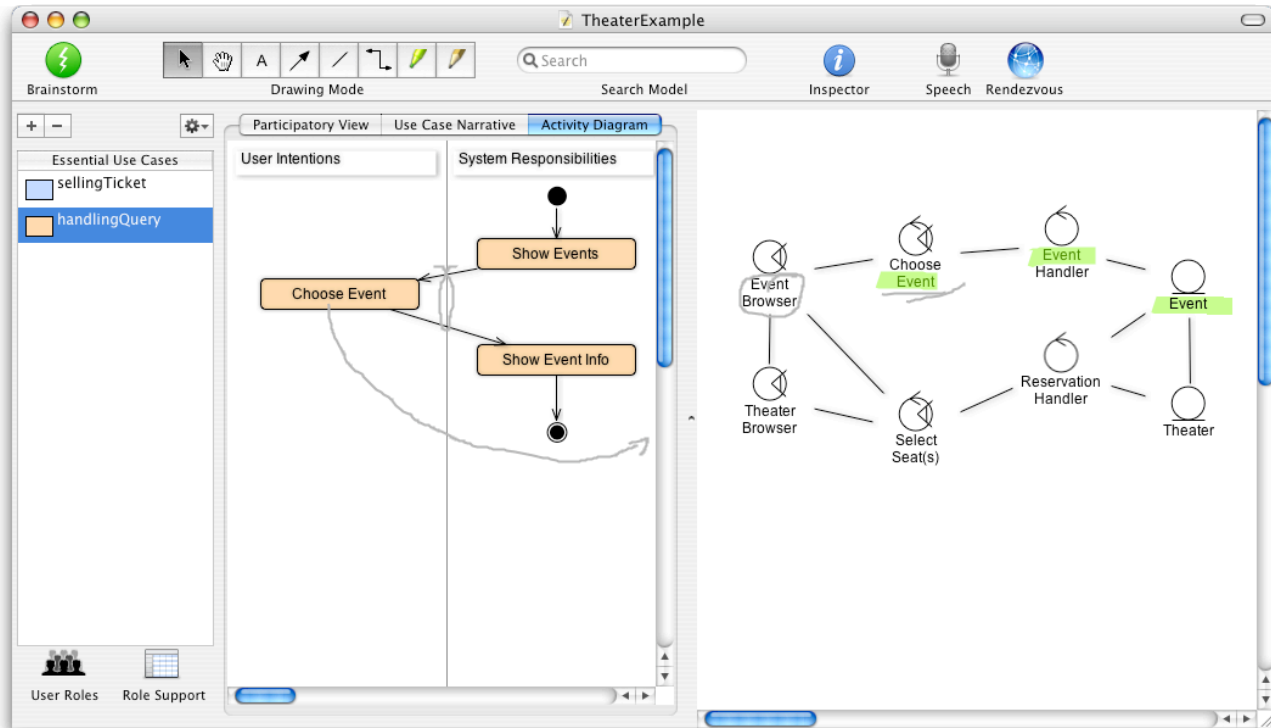


Figure 3.5: TaskSketch - tracing use-cases (right) and activities (left) in Wisdom notation (Campos, 2005b)

Brainstorming environment for collaboration

Figure 3.5 shows the interface of TaskSketch. Use-cases are displayed on the right, while activity diagrams, use case narratives and a participatory view are shown on the left side. Use cases are then highlighted to visualize relationships, to the corresponding models. Campos (2005a) additionally added support for speech input and group dynamics. To facilitate the latter, TaskSketch provides a Brainstorming environment that is synchronized with the use case contents. In this brainstorming environment, multiple stakeholders are entering ideas for concepts and tasks in a text field on the bottom. Accordingly, color-coded boxes appear on the top and slowly fall to the bottom of this shared display. Concepts and ideas are then clustered by dragging them close to other items. Campos (2005b) proposes this environment for stakeholder meetings or discussions in a team of designers. According to him, the playful character and the ideas from different backgrounds will foster collaboration among stakeholders.

Tool transitions prevail and conceptual design is neglected

As CanonSketch, which is dedicated to later process steps and TaskSketch, which is used for early requirement elicitation steps, are two separated tools, a transition between tool usages prevails during development processes. Consequently, a gap between early requirements and interface design is created that neglects the need for conceptual design. After the task is known, designers are forced to use additional modeling tools or whiteboard sketches to invent a solution. Accordingly, we think that CanonSketch and TaskSketch provide effective means of requirements tracing, but also believe that the lack of support in conceptual design harms creativity and innovation. The proposed brainstorming environment supports collaboration, but only in respect to idea generation in early process steps. Nevertheless, the novel way of hybrid representation of both interface designs and corresponding tasks adds significant value to rationale support. Due to the interactive nature, artifacts are flexible and moving between abstract and detailed representations is simplified.

WinSketch

The concept of CanonSketch and TaskSketch was further refined, which led to a redesign in WinSketch. WinSketch combines methods from CanonSketch and TaskSketch by utilizing the Wisdom method as well as canonical abstract prototypes. In addition, WinSketch employs models that originate from Usage-centered Design (Constantine, 1996). Figure 3.6 shows its simple interface, which features process navigation, whiteboard views and book-

marks (right). Users are guided through the usage-centered design process by following a horizontal hierarchical navigation that is structured into the basic entities "Requirements", "Analysis" and "Design". Each step is accompanied with one or multiple diagram representations that represent a model based view on 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.

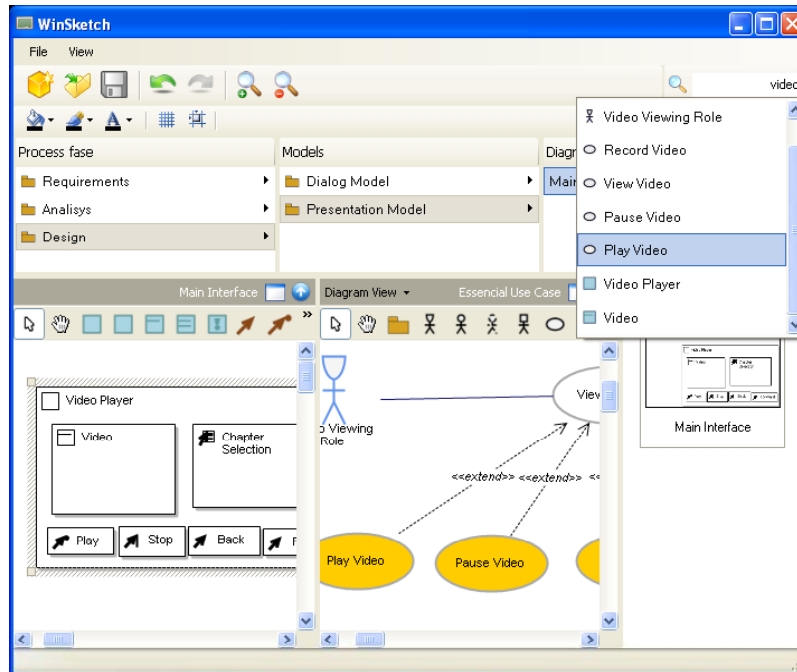


Figure 3.6: WinSketch

Again, we think that WinSketch provides effective support in tracing requirements. Additionally, the incorporated models of Usage-centered Design provide means to develop a conceptual design before approaching actual interface design. The guided process support improves traceability even more. Nevertheless, relationships between artifacts are cumbersome and hardly visible, as some of the different views are not synchronized. As diagrams are nearly always displayed in small frames, it is hard to gain overview and to switch between different representations. The rationale behind the interface representation is therefore hard to understand which makes this solution not suitable for specification purposes. Additionally, created designs cannot be exported in a reusable format.

Guided process and flexibility but lack of visualization

### 3.4 Research Gap

Based on the related research that was briefly described, we identified promising concepts and tradeoffs that may help to successfully support interdisciplinary specification. While sketching tools facilitate creative means of expression and rapid prototyping functionality in early process steps, produced artifacts are not properly integrated into the overall process. Additionally, initial requirements and conceptual modeling are not efficiently incorporated which retains work transitions. Nevertheless, rapid generation of prototypes with XML specification support is an appealing option for smoothing process step transitions between abstract and detailed interface representations. The interoperability of these XML specification languages enables to distribute created final designs for implementation purposes. In respect to the hybrid approaches that focus on bridging work transitions as well as supporting traceability, we think that integration of a wider scope of the development process, from initial requirements over conceptual design to final design is necessary. This will both im-

Evaluating concepts and tradeoffs

prove traceability, reduce tool transitions and may lead to a comprehensive design rationale. Nevertheless, flexibility of artifacts is still desired and improvements in making design decisions visible are clearly indispensable. In addition, we observed that collaborative aspects bear potential for improvement. Nevertheless, we think that innovative visualization concepts, like zooming may help to visualize the various degrees of abstraction found in design practice.

Identifying the research gap

To describe the research gap that we are approaching, a comparative framework may help to distinguish our proposed tool ideas from current state-of-the-art tool support and innovative research projects. Traditionally, UI tool comparison is determined primary by the process phases they support (Preim, 1999, p. 301). As the approach of hybrid tools and that of ours is to bridge process step transitions, these frameworks are not suitable for comparison. A simple table, which compares functionality based on categories, may provide a good overview but fails in visualizing the scope of differences. Campos and Nunes (2006) present a framework for work style transitions in interface design that addresses a wider scope. They classify UI tools based on three basic categories: "Collaboration", "Tool usage" and "Notation". Within these categories, specific dimensions are identified that reflect the degree of which this characteristic is present. "Collaboration" therefore consists of the axis "Asynchrony" (same time to different time) and "Distribution" (different places to same places). "Notation" is identified by "Perspective" (problem to solution), "Formality" (informal to formal) and "Detail" (abstract to concrete) while "Tool usage" is characterized by "Traceability" (independent to coherent), "Functionality" (not functional to fully functional) and "Stability" (modifiable to stable). We think that this framework is suitable to visualize differences in the approaches at hand by subjective qualitative estimation.

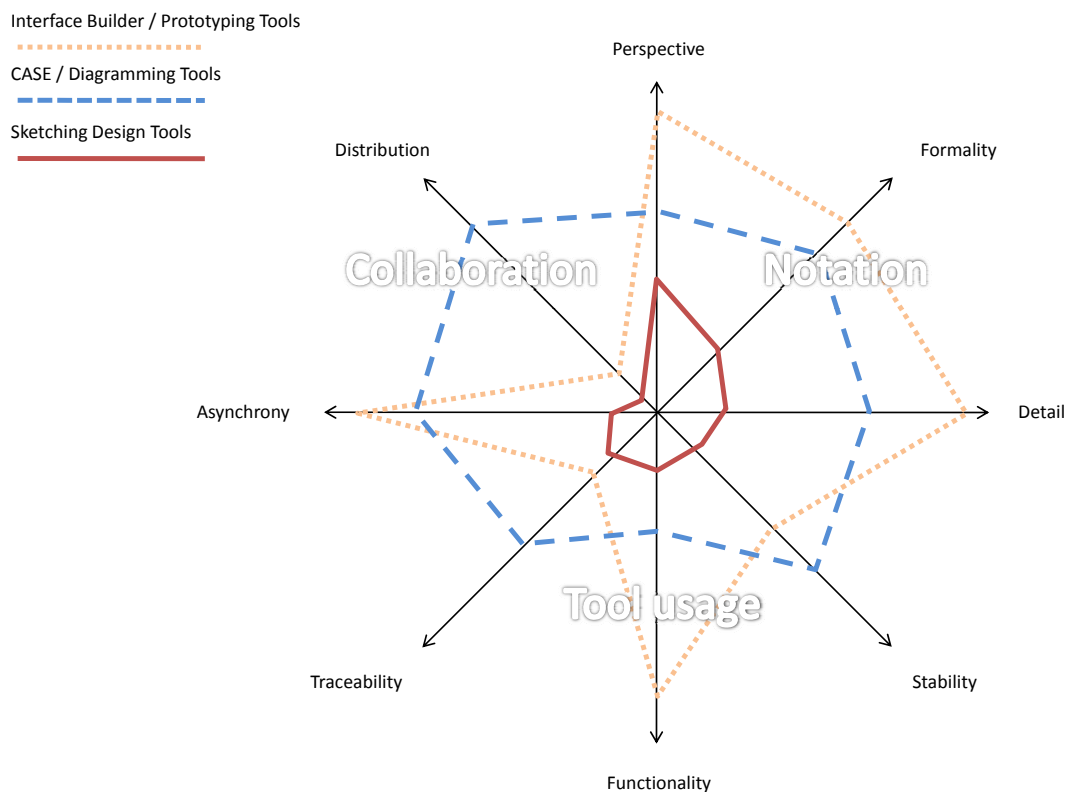
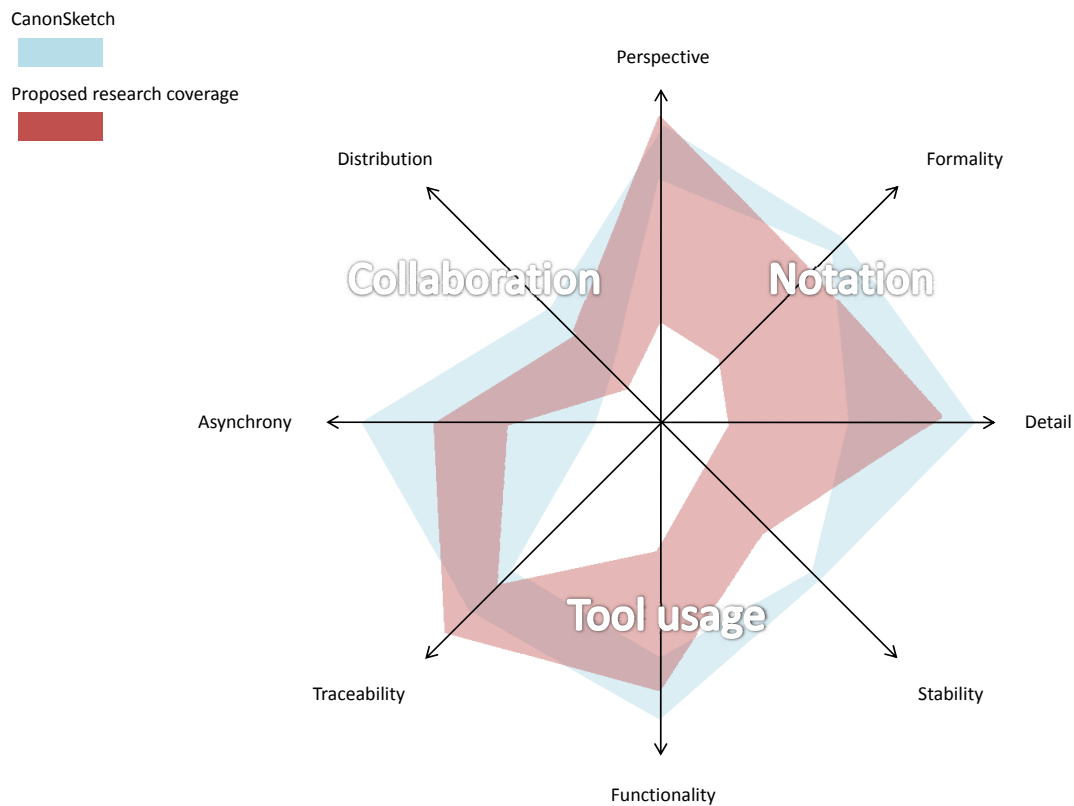


Figure 3.7: Comparison of selected tools, adapted from Campos and Nunes (2006)

Commonly used tools only cover distinct process phases

Figure 3.7 presents a comparison of selected tools, namely interface builders and prototyping tools, CASE and diagramming tools as well as sketching tools. Characteristics of tools are plotted against the eight dimensions of the work style model. While sketching tools are excellent in early design steps in terms of notation, they lack required support in collaboration and tool usage. Diagramming tools in contrast achieve a good average in most

dimensions, which makes them suitable for intermediate process steps. Finally, dedicated interface builders excel in functionality, formality and detail and are therefore positioned towards the end of the design process. The visualization reveals that currently multiple tools are required to leverage all dimensions of work style.



**Figure 3.8:** Comparison of proposed research coverage to CanonSketch, adapted from Campos and Nunes (2006)

Figure 3.8 shows CanonSketch displayed in work style dimensions along with the proposed research coverage within this work. In contrast to regular tools, both hybrid approaches cover a specific range on each dimension. Consequently, both tools are visualized as overlapping planes. The visualization reveals significant differences in both approaches. While CanonSketch focuses primarily on formal means of expression and is primarily employed towards a narrow solution space, our approach aims on providing a larger range of notation coverage by allowing informal to formal means of expression and by maximizing the range of available degrees of abstraction and detail. Consequently, our approach should cover more process steps, especially in conceptual design. In collaboration, we argue to keep design local and to reduce asynchrony, as we think that face-to-face collaboration is mandatory for good design and should not be replaced. In terms of tool usage, we want to focus on traceability as well as functionality. Stability of artifacts should retain a certain amount of flexibility, as the iterative nature of the design process requires constant changes. Consequently, in reference to CanonSketch we argue for a broader scope, wider means of expression and less formality to support creativity and innovation in design by still respecting required functionality and traceability for efficient process support.

Comparison against  
CanonSketch

Overall, our approach to a specification tool that respects the design process itself is a novel contribution to design practice. While dedicated specification tools focus on specification of interfaces when actual design work is done, we argue for integration of design rationale in such a specification. Nevertheless, as an interactive tangible specification our approach also aims on utilizing the potentials of collaboration in interdisciplinary contexts. By using a "common language" in design and modeling as well as tool-usage, we aim on minimizing work transitions that hamper development. While CanonSketch and TaskSketch define

A novel contribution  
to design practice



themselves as "Usage-centered" CASE tools, our approach aims on offering similar traceability by respecting a wider scope and additionally informal means of expression. By offering a single solution that integrates various design steps, the overall design process is simplified and design work will eventually lead to a specification without the need to explicitly compile such a document.

## Chapter 4

# Analysis

*“Knowing how people will use something is essential”*

—Donald Norman

Based on the previously described investigation of general implications for tool design and concepts from related work, this chapter presents a detailed analysis of how our ideas can be integrated into current practice. Therefore, commonly employed structured approaches to interface design and widespread process models are analyzed in reference to our research goals. We will propose an integrative framework by providing an adapted process model and an interdisciplinary selection of employed means of expression. Eventually, the resulting techniques, artifacts and tasks are presented. Finally, elaborated requirements are presented along general guidelines that may provide assistance to our design.

Outlook

### 4.1 Structured Approaches to User Interface Design

A result of different discipline-specific perceptions in interdisciplinary design is the variety of structured approaches to interface design. While engineers tend to use structured and formal processes, designers employ creative means and human factor specialists focus on cognitive approaches to interaction. To achieve successful cooperation between actors it is therefore crucial to create a common approach that respects discipline-specific priorities. According to Jose (2003) the first step to successful cooperation is to make sure that misunderstandings are eliminated by creating a common language for communication, or as Erickson (2000) calls it, a “lingua franca” for design.

Common approach is crucial for cooperation and communication

Many of the previously described implications for effective tool-support rely heavily on the use of a common approach. When models are used during the design process, they have to be understood by all participants. Even when communication between actors is limited to clear frontiers (see Chapter 2.4.3—“Communication & Specification”), e.g. as specification, and an effective separation of concern is employed, a unified process model is essential to avoid interference (Jose, 2003). Design rationale techniques are also only successful when they agree on one common form of description for all actors. As common approaches also determine the general influence of engineering and design philosophies, they are an effective instrument to create a collective course of action that facilitates certain priorities like creativity in design. This section gives an overview on widespread approaches and processes and introduces a novel method to interdisciplinary specification that is the fundamental of the later introduced integrating tool. Furthermore, essential tasks, techniques and tool-usage during design processes are analyzed, before implications for tool-design are presented.

Effective tool-support has to employ an interdisciplinary approach

### 4.1.1 Integration with Software Development

UI design process is part of certain stages in software development

The process of user interface development is an integrative part of the software development cycle. The most widespread model for general process steps within software development is the system lifecycle. The following steps summarize the well-known waterfall model of a generic system lifecycle (Dix et al., 2003, p. 227ff), (Sharp et al., 2007, p. 414ff):

- Requirements analysis and functional analysis
- Preliminary design or architectural design
- Detailed design
- Implementation and testing
- Integration and evaluation
- Maintenance

Usually software development is considered as cyclic. A new version of software products or added functionality requires starting the cycle from the beginning after completing it. As some of the activities may lead to the discovery of unforeseen problems, there is also often the need to go back to a previous stage, which makes most of the process steps iterative (Dix et al., 2003, p. 227ff).

User interface design moved to the early stages of the cycle

Traditionally the user interface tended to be built at the end of the software development cycle, where almost all functionality was already implemented (Dix et al., 2003, p. 235). This procedure is naturally, given that simple user interfaces, as command-line interfaces did not require much attention, but were only designed to invoke underlying functionality. The waterfall model does work quite well if initial requirements do not change during system development. As user interfaces gained in importance during the 1980s it was common, that requirements for them did change during later phases within the development cycle, which led to costly iterations. This issue was raised because many requirements for UIs cannot be determined from the beginning, but have to respect evaluated designs in order to make them more useful (Dix et al., 2003, p. 235). Accordingly, user interface tasks were moved to the early stages of the development cycle. Today there are various approaches to the integration of user interface design in software development, which will be analyzed in the following sections.

### 4.1.2 Widely adopted Structured Approaches

Variety of different structured approaches

While most researchers and practitioners agree on a basic iterative and phase-based cycle (see Chapter 4.1—“Structured Approaches to User Interface Design”), they still propose a variety of different development cycles. These structured, iterative approaches differ in scope, general philosophies and applied methods or models. Whereas some processes tend to employ engineering methods (Usability Engineering), other approaches focus on user needs (User-centered Design) combined with tasks (Usage-Centered Design) or active user involvement (Participatory Design). Specific process descriptions provide particular techniques for all phases, which determine the characteristics of the overall process. Some approaches are considered as “heavyweight”, while other approaches are called “lightweight”. On the one hand, “heavyweight” processes often focus on formal techniques and activities to achieve predictable results but lack in support for creativity. On the other hand, informal activities in “lightweight” processes may enhance creativity in design but restrain traceability and predictable results. Therefore, employed methods and techniques guide the overall development and the success in terms of usable results. In the following paragraphs, some of



the most widespread and acknowledged structured approaches are presented, because they are an essential fundamental of a requirements analysis for tool-support. While there are an extensive number of structured approaches, only the most relevant ones that influenced the methodology employed in the proposed interdisciplinary tool-design are described in detail. Important aspects of these approaches are highlighted for later reference.

Focus on selected approaches

One of the earliest efforts in providing a structured approach to user interface design is "User-centered System Design" by Norman and Draper (1986).

"User-centered design emphasizes that the purpose of the system is to serve the user[...]The needs of the users should dominate the design of the interface, and the needs of the interface should dominate the design of the rest of the system."  
(Norman and Draper, 1986, p. 32)

Norman and Draper (1986) do not specify a concrete process or propose specific steps but merely provide an abstract framework of alternative ways of doing things in interface design without much practical grounding (Carroll, 1991, p. 227ff). Nevertheless, their approach "User-centered design" is a landmark in HCI research and heavily inspired today's design processes as well as the ISO 13407 (see Chapter 4.2—"Adapted Process Model"). Norman argues that behavioral science may help to improve user interfaces. He coined the term "Cognitive Engineering" for this methodology, which is now found in most of today's structured approaches within the "Analysis" or "Conceptual Design" phase. In Norman (2002) he presents a structured approach, called "seven stages of action". This process model can be considered as a design aid, which leads to a basic checklist of methods that help to bridge the "gulf of execution" and the "gulf of evaluation". The "gulf of execution" stands for the inconsistency of what users want to do, and what they are allowed to do, while "the gulf of evaluation" stands for the user's efforts in interpreting the state of the system based on their expectations. As a result, Norman (2002) introduced the requirement of conceptual modeling and the existence of gaps between the analysis phase, conceptual model and physical system design, which are now a major concern in most structured approaches and reflected in three-step process designs (Sears and Jacko, 2007, p. 10). The term "User-centered design" currently tends to be used as a general design philosophy, rather than a concrete approach.

Influence of User-centered Design

In contrast to these rather vague descriptions, an early concrete approach to a structured process is the STAR lifecycle, presented by Hartson and Hix (1989) (see Figure 4.1). It provides an evaluation-centered iterative process around five basic activities: task analysis, requirements specification, conceptual design, prototyping and implementation. Hartson and Hix (1989) came about this methodology by studying how design takes place in practice. They found that evaluation is crucial before going on to the next stage. They argue that going both "bottom-up" and "top-down" is required in iterative waves. However, they do not explicitly define a start point in their cycle, but propose that the iteration may start in any phase within the process. They conclude, that it is essential to do both structure and detail at the same time to achieve effective results. In Hix and Hartson (1993, p. 117ff) they present a systematic procedure with detailed descriptions of activities within all phases of their lifecycle model reaching from early requirements to implementation.

The STAR lifecycle

Later on, Nielsen (1992) presented a solid process model, which he calls "The Usability Engineering Lifecycle". According to Nielsen, a comprehensive lifecycle that corrects the quality of the design at every single step is necessary to create usable systems. He recommends certain activities, divided across three main stages of development: "pre-design phase", "design phase" and "post-design phase".

Nielsen's Usability Engineering Lifecycle

"A usability engineering process to ensure good user interfaces includes elements to be considered before the design, during the design, and after field installation of a software product."  
(Nielsen, 1992, p. 1)

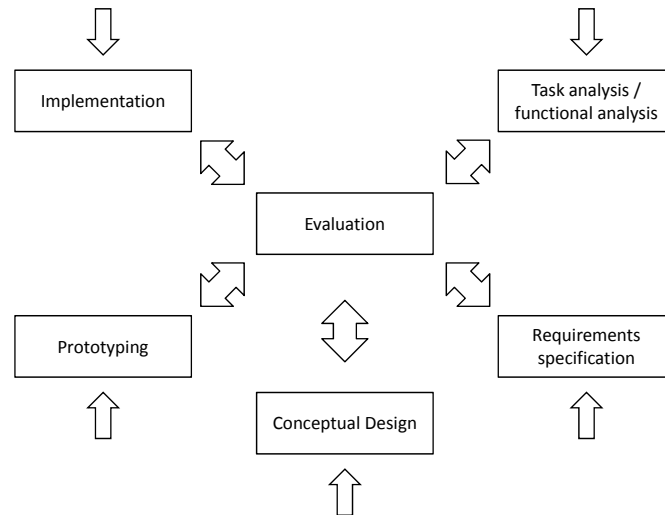


Figure 4.1: The STAR Lifecycle, adopted from Hix and Hartson (1993)

Within the “pre-design stage”, Nielsen lists field studies, usability tests of existing systems and competitive studies of other solutions as the main activities. In his second phase, the “design phase” he recommends to start with parallel design exercises. After deciding for one basic design approach, he argues to evolve the chosen design in an iterative refining manner by using prototypes of different fidelity. During the “post-design phase”, collecting feedback and usage statistics provide implications for redesign or a refresh of existing designs. Nielsen regards his process as lightweight and estimates the effort of each step within the phases to about one or two days of work.

Implications for tool design



We think that Norman’s and Nielsen’s contributions are an important fundamental to our tool design, because they clarify the basic gap between analysis, conceptual design and physical design as the major transitions that have to be bridged. Therefore, our efforts have to respect these gaps in process flow. Narrowing these will eventually improve design practice. However, actual examples for support for these demands remain an open issue. Additionally, we may learn from Hix and Hartson’s contributions that constant evaluation is crucial during all parts of the design process. Therefore, we argue to integrate evaluation functionality in the overall solution.

Mayhew’s Usability Engineering Lifecycle

A quite different user interface design process model with the same name, “The Usability Engineering Lifecycle”, was presented by Mayhew (1999). Unlike Nielsen (1992), Mayhew provides a holistic view of usability engineering and links user interface design activities to software engineering techniques, like Object-oriented Software Engineering (OOSE) methods developed by (Jacobson, 1992). Mayhew’s approach is considered the first real attempt to merge Software Engineering with UI design.

“The Usability Engineering Lifecycle documents a structured and systematic approach to addressing usability within the product development process. It consists of a set of usability engineering tasks applied in a particular order at specific points in an overall software development lifecycle.”  
(Mayhew, 2005, p. 42f)

Figure 4.2 presents, in summary, a visual representation of Mayhew’s lifecycle. The overall process is divided into three phases: “Requirements Analysis”, “Design/Testing/Development”, and “Installation”. Within each phase, a flowchart presents specific usability engineering tasks, their order of application and possible iterations. May-

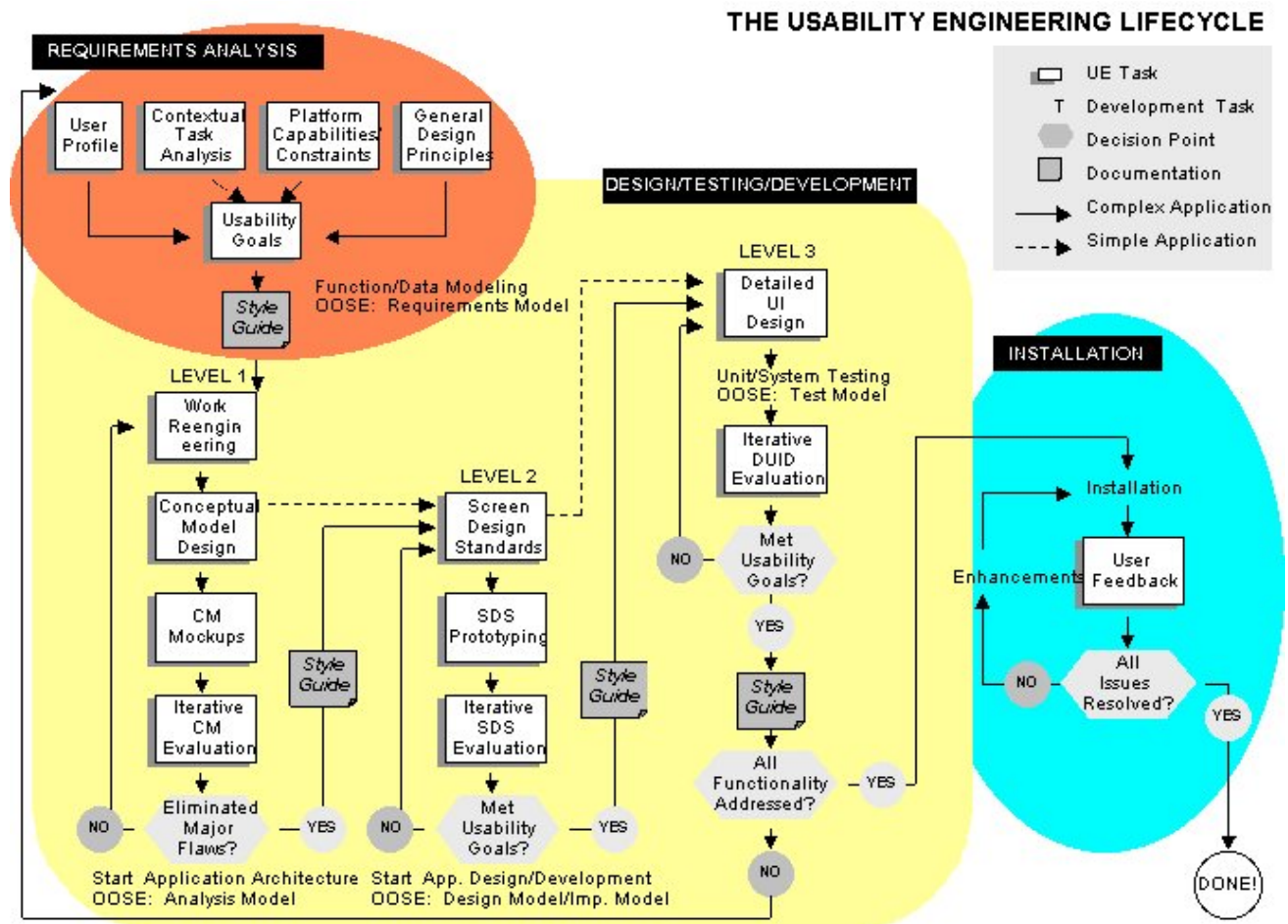


Figure 4.2: The Usability Engineering Lifecycle (Mayhew, 1999)

new links certain stages within phases to corresponding OOSE stages in order to synchronize development. A usability engineering "task" can be defined as an activity that produces a concrete artifact, which is the prerequisite for subsequent process steps. Mayhew does not explicitly define one single technique for accomplishing a task, but rather lists a set of techniques to choose from that originate from both "User-centered Design" (Norman and Draper, 1986) and "Usage-centered Design" (Constantine, 1996). This flexibility of choice in techniques makes the overall process adaptable to various problem domains and project sizes. Therefore the lifecycle may be considered both "lightweight" and "heavyweight". Mayhew's first phase, the "Requirements Analysis" comprises user profiles, task analysis, general principles, goal settings and constraints. The second phase, "Design/Testing/Development" is spread into three levels and starts with conceptual modeling in the first level. Level 2 applies prototyping methods while Level 3 evaluates final designs. Results within stages are kept within a "styleguide", which is the overall documentation for use throughout the process and for later reference. Depending on the scope and detail of this documentation, it may be considered as a "Design Rationale". Mayhew's last phase, "Installation" provides user feedback from application in practice for future improvements. The described techniques, which go along with the cycle, are very much focused on usability instead on design.

Constantine and Lockwood (1999a) initially coined the term "Usage-centered Design" to describe their approach to user interface development. Nowadays this term tends to be used as a general philosophy, like "User-centered Design". Although similar in name, both approaches differ inherently in their basic philosophy.

Usage-centered design and software engineering

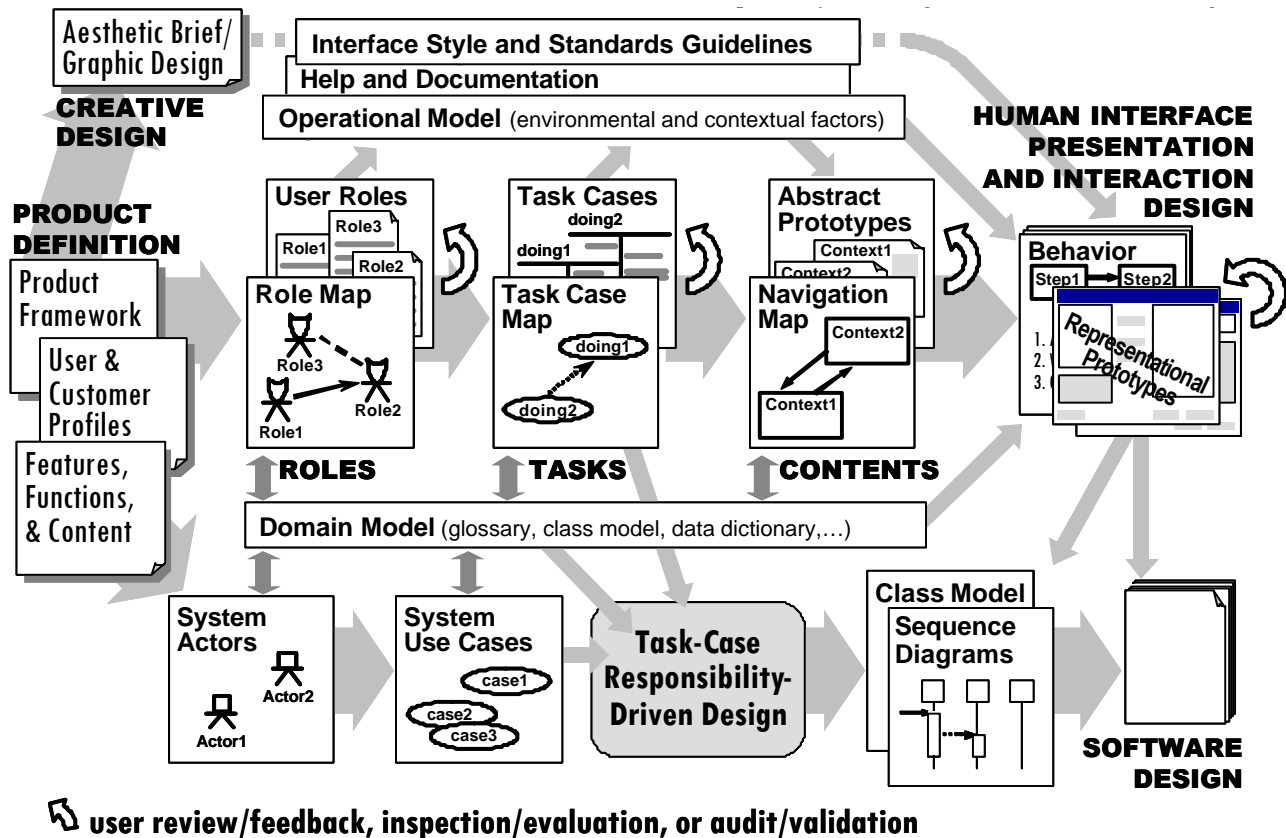


Figure 4.3: Usage-centered Design Process (Constantine and Lockwood, 2003)

"To design dramatically more usable tools, it is not users who must be understood, but usage - how and for what ends software tools will be employed[...]Usage-centered design focuses on the work that users are trying to accomplish and on what the software will need to supply via the user interface to help them accomplish it."

(Constantine and Lockwood, 1999a, p. 23)

In contrast to Mayhew (1999), Constantine and Lockwood (1999a) propose the exclusive use of models and abstract representations in analysis and design to facilitate innovation and to make the overall process more agile. Constantine and Lockwood (1999a) argue that strict guidelines and extensive text-based techniques are hampering creative designs and may lead to more costs as well as to unsatisfiable user experiences (Constantine and Lockwood, 1999a, p. 23). Therefore, they present a "lightweight" set of simple, abstract models and corresponding techniques, pragmatic guidelines as well as an organizational framework to guide the UI development process just enough to make it successful, but not too rigid to constrain creativity. Whereas these model-based techniques are the key elements to the underlying process, they can also be considered separately as general methods for improving software usability. The general approach to employ models in UI design is also often referred to as "model-based design".

The Usage-centered process

The process itself, outlined in Figure 4.3 is based on concurrent engineering. Although the core models are connected in a sequence, they are developed concurrently in practice, as designers are moving from model to model. Additionally, the process is divided into two threads. On the one hand, the primary thread is focused on interface design, while on the other hand a secondary, parallel thread is focused on system implementation issues. According to Constantine and Lockwood (2003) this separation allows to integrate UI design into existing Software Engineering processes in an efficiently manner. By using abstract models within all steps of the development cycle, communication is facilitated between hu-

man factor specialists, designing within the primary thread, and implementation specialists, guided by the second thread (see Chapter 2.4.2—“Separation of Concerns”). Constantine and Lockwood (1999a,b) and Constantine (1996, 2003) give detailed guidance on employed models and techniques for practitioners. The most important abstract models within the conceptual UI modeling phase are “User Roles” and “Role Maps”, “Task Cases” and “Task Maps” as well as “Essential Use Cases”. Physical design is assisted by prototypes of different fidelity, like “Canonical Abstract Prototypes” and “Navigation Maps”. Concurrently employed Software Engineering models may range from adapted models like “System Actors” and “System Use Cases” to general Unified Modeling Language (UML) models, like class diagrams or sequence diagrams. According to Constantine and Lockwood (2003, p. 6) the usage-centered design process is a well established “industrial strength” process that has proven successful in numerous projects in a variety of businesses and organizations around the world.

In reference to our research emphasis we believe that the flexible choice of tasks and techniques within Mayhew’s lifecycle is appealing as it imposes a flexibility on the overall process. Therefore, we will consider adaptability and choice for tool design to provide adaption to various problem domains. Based on Constantine’s contributions, we think that modeling, creativity support and innovation are smoothly integrated. Additionally, we believe that the employed models provide a fundamental for interdisciplinary design processes that effectively separates concerns in design.

Implications for tool design



While Mayhew (1999) employs detailed and text-intensive methods, Constantine and Lockwood (1999a) propose to use models as a substitute to make the overall process more agile and flexible. An alternative solution to introduce both abstraction and detail into a “lightweight” user interface development process was introduced by Rosson and Carroll (2001). Their structured approach, “Scenario-based Design” utilizes scenarios, or “real-life stories” to describe the context of use all the way through process steps. Scenario-based design uses concretization to avoid the unmanaged splitting of design problems by abstraction. Instead of starting off with listing user needs, requirements and tasks, the designers focuses on activities early on as the fundamental to subsequent design steps.

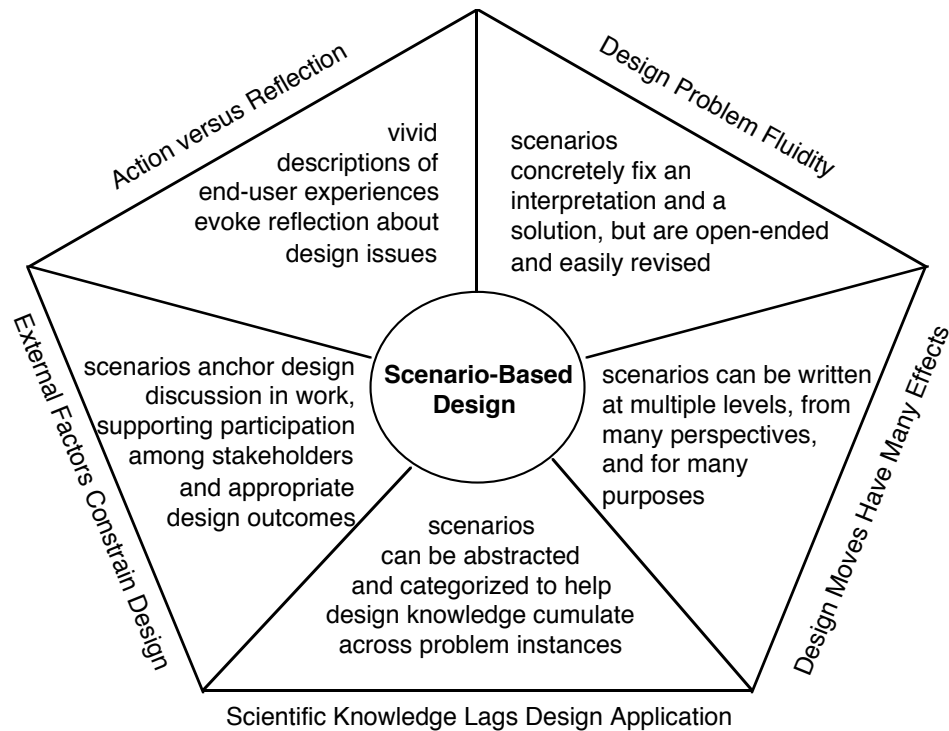
Scenario-based Design

“Scenarios are stories. They are stories about people and their activities[...]. In scenario-based design, descriptions of how people accomplish tasks are a primary working design representation[...]. Scenarios highlight goals suggested by the appearance and behavior of the system, what people try to do with the system, what procedures are adopted, not adopted, carried out successfully or erroneously, and what interpretations people make of what happens to them.”  
(Carroll, 2000a, p. 1f)

According to Carroll (2000a), scenarios may have various forms of representation: text-based stories, “action-event” spreadsheets, storyboards, sequence diagrams, flowcharts or even prototype simulations. As presented in Figure 4.4, scenario-based design may improve UI design in five areas (Carroll, 2000a): Scenarios invoke reflection in the content of the design work, by coordinating action and reflection. They are concrete and flexible at the same time, which improves fluidity of design situations. They provide multiple views of an interaction - both abstract and detailed - to help developers manage consequences of design moves. They can easily be captured, recorded and categorized. They facilitate communication among stakeholders, helping to make design activities more accessible. Especially this feature makes scenarios a proper technique for communication in interdisciplinary design. As described in Carroll (2000a, p. 2), these benefits also make scenarios suitable for design rationale representations.

“They (scenarios) are the minimal contexts for developing user-oriented design rationale: a given design decision can be evaluated and documented in terms of its specific consequences within particular scenarios.” (Carroll, 2000a, p. 2)





**Figure 4.4:** Aspects of Scenario-based Design (Carroll, 2000a, p. 10)

Variety of participatory and cooperative approaches

While Constantine and Lockwood (1999a) and Rosson and Carroll (2001) propose an alternative to “User-centered Design” and Mayhew (1999) combines existing methods in one approach, they still agree on the general process model. Similarly, there are a number of structured cycles that promote different philosophies and provide detailed descriptions for integration into the general process model. One of these alternative philosophies is called “Participatory Design” (PD) (Bodker et al., 2004) or “Cooperative Design”. In contrast to engineering cycles or usage-centered approaches, these structured processes are a subset of the user-centered design philosophy. Their major characteristic is the active participation of real users throughout the process. As the field of PD is extraordinary diverse (Sears and Jacko, 2007, p. 1052), only one popular example is described.

Contextual Design

Beyer and Holtzblatt (1997) present a “Customer-centered Design” approach to user interface design called “Contextual Design”. Similar to user-centered approaches, Beyer and Holtzblatt (1997) utilize detailed customer data within all phases of development. They incorporate ethnographic methods, field studies and customer participation in their top-level steps. Similar to Constantine and Lockwood (1999a), Beyer and Holtzblatt (1997) also rely on model-based methods for several of their process steps, like flow-models, sequence models or affinity diagrams. Since the process of gathering extensive user-data is often time consuming and costly, this approach is considered as “heavyweight” (Holtzblatt et al., 2004, p. 21). Consequently, Holtzblatt et al. (2004) present a more lightweight approach to contextual design for use in various contexts by skipping some of the process steps and by utilizing tool-support.

Implications for tool design



In respect to our investigation, we think that scenarios provide effective means for interdisciplinary communication and reflection in design. As they are understood by all actors no matter what their backgrounds are, they can be utilized as a common language in design. Additionally, they are applicable to virtually all parts of the design process. Therefore, they may be employed for problem decomposition and as a general guiding structure. In respect to contributions from “Customer-centered Design”, we believe that the collection of extensive user data might collide with our needs for a lightweight approach to facilitate agile freedom.

## 4.2 Adapted Process Model

As there are a quite extensive number of variations of the described approaches that were adopted in practice, this chapter introduces a customized process model as fundamental for an integrating specification tool. This procedure seems natural, since the majority of approaches are adapted to a custom process model for application. Therefore, we now present an adapted process model that combines previously explained implications for tool design and the described client-supplier situation for specification. How specific tasks during all phases are accomplished and which tools are employed within certain steps will be described in Chapter 4.2.1—“Tasks, Techniques and Artifacts”.

Customized  
process model

Just like software development processes, user interface development can be considered as a stage model. There are certain activities that have to be completed within a stage, before the next stage can be approached (Heim, 2007, p. 97). Consequently, some approaches to interface design are similar linear as the waterfall lifecycle, but there are also approaches that are more user-centered and do not employ a specific order of development steps. Nevertheless, all of them have definite phases and some degree of iteration. Jokela et al. (2003), Sharp et al. (2007) and Maguire (2001) reference the ISO 13407 “Human-centered design processes for interactive systems” (Jokela et al., 2003) as a standard that provides general guidance for human centered design processes. ISO 13407 can be regarded as an important supplement to specific HCI literature. First, as a standard, it is based on the consensus of a wide international board of researchers and practitioners of the field. Secondly, it approaches design from a higher level of abstraction than most methodology books. Rather than describing different specific usability techniques, it describes usability at a level of principles, process flow and activities. It is therefore a good source to describe user interface design processes in general.

Interface design  
approaches have  
definite phases

The following list, based on findings of Heim (2007, p. 97), Sharp et al. (2007, p. 414ff) and Moggridge (2006, p. 726ff) as well as the general process model in ISO 13407 summarizes the most common phases found in interface design approaches:

- Cost and risk analysis
- Observation
- Task analysis
- Requirements assessment
- Conceptual Design
- Physical Design
- Prototyping
- Evaluation
- Usability testing
- Documentation
- Maintenance

Common phases  
during UI  
development

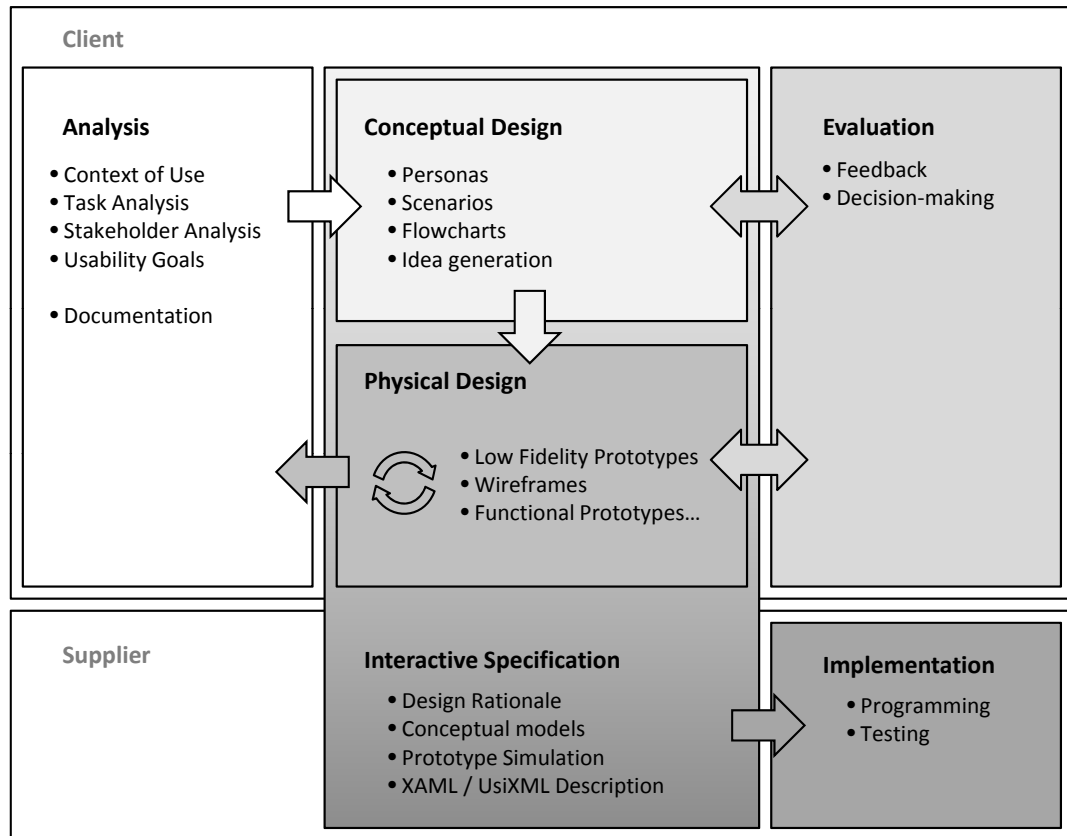
These discrete steps, together with descriptions in ISO 13407 and Heim (2007, p. 97) as well as Sharp et al. (2007, p. 414ff) can be turned into a more abstract process model. Combined with our demands and perceptions to the subject matter, Figure 4.5 presents a simplified view on the employed UI design process, its activities and iterations for later reference. The adapted process model is divided into two main areas of responsibility: client & supplier. While the client focuses on design questions, the supplier is responsible for implementing a specified design. However, the supplier is limited to one task, namely implementation

An abstract model  
for user interface  
specification

from specification, while the client is responsible for a series of tasks that are described as the following phases.

#### Analysis phase

The initial step, the "Analysis" phase, answers questions about the basic components of the interface design project, like involved tasks and current work practice as well as user profiles. Ideas on how to solve a problem are generated during brainstorming-like sessions. This process part is also frequently called "Requirements Analysis" or "Requirements Engineering" phase. It concludes with keeping all external constraints and goals in some form of requirements documentation, before moving on to the next stage, the "Design" phase.



**Figure 4.5:** Adapted Process Model based on Memmel et al. (2007h), Heim (2007, p. 97), Sharp et al. (2007, p. 414ff), ISO 13407 and Moggridge (2006, p. 726ff)

#### Design phase

The design stage consists of two subsequent steps: the conceptual design and the physical design of the interface. During conceptual design, different perceptions of how to address the needs generated in the previous phase by restructuring work are explored. Usually this process is documented with various abstract models of the problem space, like personas, scenarios, task cases or flowcharts. In the physical design phase, many possible ways of realizing the conceptual design in the real world are designed. Initial methods of exploring alternative designs are low-fidelity prototypes, like paper sketching and wireframes. Informal methods as drawings of user interfaces on papers are inexpensive and creative means of early exploration. As the solution space is narrowed towards a final design, high fidelity and functional prototypes are employed to evaluate look & feel of the interface. The physical design phase itself applies iterative methods to generate advanced or alternative designs. For the "Design" phase, we will later introduce an interdisciplinary selection of models and means of expression that are characterized by different levels of abstraction (see Chapter 4.2.2—"Interdisciplinary Selection of Artifacts").

#### Evaluation phase and overall iteration

During both design steps, evaluation methods are used to measure the success of the applied design steps or proposed interface designs. Early informal feedback (e.g. user feedback) in design phases may provide important clues on how to achieve a better design and lead to the



elimination of expensive iterations. Consequently, the "Evaluation" phase is not a discrete phase, but accompanies the entire design phase from the beginning to the end. However, none of the described phases is mutually exclusive. While moving through the process steps one may learn that previously unrelated aspects are influencing initial requirements (see Chapter 2.3—"Technology and Innovation"). These issues require iterating between all three phases "Analysis", "Design" and "Evaluation".

Eventually, a shift of responsibilities is initiated as the overall design reaches a final solution. Consequently, the client has to communicate the developed solution to the supplier. Therefore, the overall rationale behind the designed interface should be made accessible to the supplier. Based on our investigation, we argue that artifacts that resulted from both conceptual and physical design should be compiled to an interactive specification that includes the Design Rationale, conceptual models, a prototype simulation and XML descriptions formats. The supplier will then deliver the final application after implementing and testing it based on the supplied interactive, browsable specification.

Interactive  
specification

### 4.2.1 Tasks, Techniques and Artifacts

Based on our findings in observing work practice (Mommel and Reiterer, 2008; Mommel et al., 2007e) and findings of (Maguire, 2001) and Ambler and Jeffries (2002) we present a cross-sectional overview on User Interface Design work practice. Tool-usage is also well observed by Ambler and Jeffries (2002) and AgileModeling<sup>1</sup>. By aligning general process steps to our adapted process model, we present a summary of involved tasks, techniques and current tool-usage that accompany the design process.

Process steps and  
artifacts

Table A.1 in A—"Methods and Tools" presents common steps during the "Analysis" phase. As early exploration of the problem space tends to be experimental and broad, resulting initial artifacts are rather informal, like paper sketches and whiteboard drawings. As tasks are analyzed and specified more precise, subsequent artifacts range from presentation slides, diagrams, spreadsheets to rather formal means of expression, like word processing and CASE tools. Similarly, requirements specification, which concludes this development phase results in documents that are extensive in content and awkward to browse.

Artifacts within the  
analysis phase

Process steps within the "Design" phase are described briefly in Table A.2 (A—"Methods and Tools"). Again, initial conceptual design is accompanied with informal means of expression, like drawings and sketches. As idea generation leads to more concrete and tangible ideas, representations that are more formal are employed, like modeling diagrams. The physical design stage again starts with informal means of expression to explore many different solutions in an efficient manner. As the solution space is narrowed towards a final design, a wide range of artifacts in different abstraction levels is employed, like lo-fi to hi-fi prototypes to facilitate effective evaluation methods. Results are then presented in specification documents that are again extensive in content and characterized by awkward traceability.

Artifacts within the  
design phase

Artifacts, produced within the final phase, "Evaluation" are presented in Table A.3 (A—"Methods and Tools"). When feedback on requirement models and designs are collected, it may originate from various sources. Evaluation meetings of stakeholders are rather informal and result in hand-written notes or transcript documents. Feedback collected in the application field or resulting from user surveys is usually integrated in documents or hypertext. Our findings reveal that current work practice entails following crucial software require-

Artifacts within the  
evaluation phase

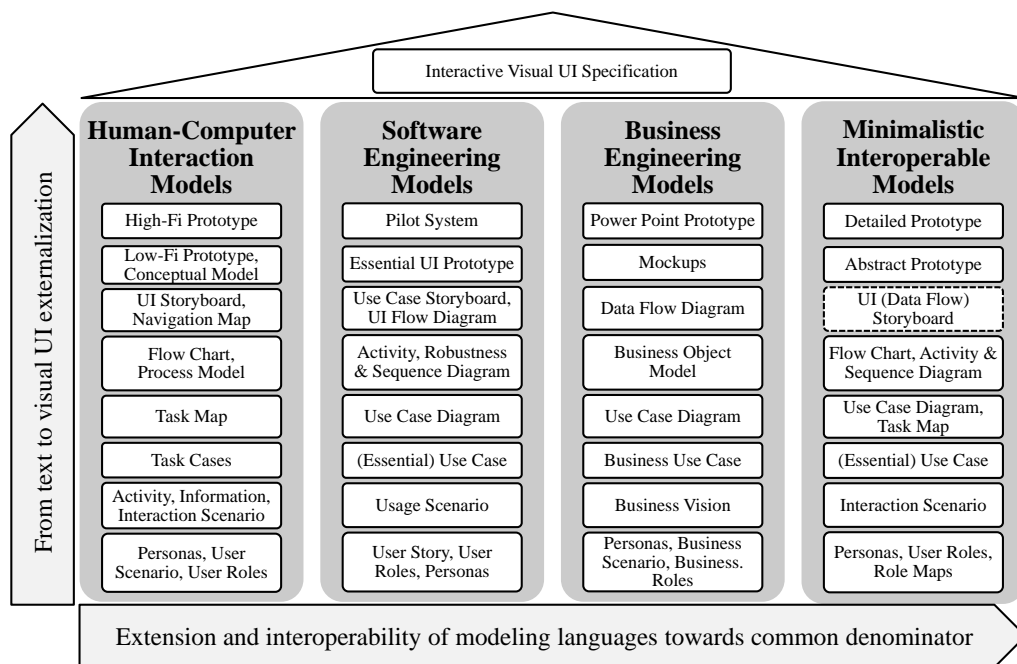
<sup>1</sup><http://www.agilemodeling.com/artifacts/>

Demands for artifact support

ments regarding the support of techniques and resulting artifacts within the user interface design process:

- Informal means of expression (Drawings) are crucial for early expression of ideas and brainstorming.
- Existing documents and feedback require word processing functionality.
- Modeling abstractions requires diagramming software or CASE tools.
- User Interface design tasks demand for prototyping tools supporting a range of fidelity levels.
- Meeting and Communication support is essential for collaboration (Presentation software).
- Hypertext techniques are obligatory for documentation, specification and artifact management.

### 4.2.2 Interdisciplinary Selection of Artifacts



**Figure 4.6:** Interdisciplinary Visual Specification Framework (Mommel et al., 2007h)

Interdisciplinary modeling and specification

As the number of previously described techniques and artifacts is extensive, we will introduce an interdisciplinary selection of artifacts that compromises various means of expressions based on their interoperability towards a common denominator. One the one hand, a major goal of this framework is to facilitate common understanding and communication between different disciplines that participate in the design process (internal communication) (see Chapter 2.4—“Interdisciplinary Design”). On the other hand, the framework also aims on bridging the gap between client and supplier with interactive specifications (external specification)(see Chapter 2.4.2—“Separation of Concerns”). The selection then forms our “common language” for specification that aims on effectively communicating both, design rationale and physical design solutions to all participating actors. The developed specification framework is then integrated into the previously described adapted process model

within the conceptual and physical design phases. Based on our findings presented in Chapter 2—“Theoretical Foundations” and Chapter 4.1—“Structured Approaches to User Interface Design”, we additionally focus on providing a step-by-step approach to interdisciplinary modeling and specification of user interfaces (Mommel et al., 2007h,j; Mommel and Reiterer, 2008). Inspired by work transitions (see Chapter 5.1—“Conceptual Design”) and various degrees of abstraction found in work practice, we therefore introduce a concept of making user interface designs externalizable as interactive visual specifications that are browsable from abstract into detail. Figure 4.6 conveys an overview on the framework, based on the extension and interoperability of interdisciplinary modeling languages toward a common denominator.

According to Mommel et al. (2007h), modeling languages used in the disciplines HCI, SE and Business Engineering (BE) can be compromised into one modeling language that all stakeholders understand. Employed models within the identified common denominator are adopted from “Usage-centered Design” (Constantine and Lockwood, 1999a), “Scenario-based Design” (Carroll, 2000b) and Agile Modeling (Ambler and Jeffries, 2002). Models that are too difficult to understand by all actors are filtered out. Models that are too close to actual implementation and are considered unnecessary by Agile Modeling are also not considered. The remaining models offer an agile freedom in terms of formality. The horizontal axis in Figure 4.6 shows the results of our filtering of models. We also identify different levels of abstraction within the discipline specific modeling languages to support visualization of process flow from initial text-based requirements to a detailed interactive prototype simulation. On the vertical axis of Figure 4.6, models are distinguished by their level of abstraction. While on the bottom, models are abstract they keep getting more precise toward the top. The sum of all created artifacts then resembles an interactive visual specification of the developed user interface, which can be explored either bottom-up or top-down. For a closer look on the rationale and integration of the presented selection, see Rinn (2008).

Common denominator in design and modeling

The presented specification framework is integrated into the adapted UI design process (see Chapter 4.2—“Adapted Process Model”) within the “Design” phase (see Figure 4.5). Conceptual Design therefore starts at the highest level of abstraction, “Personas, User Roles, Role Maps” and continues over “Interaction Scenarios”, “Essential Use Cases” and “Task Maps” to “Flow Charts”. The “Physical Design” phase is initiated by “UI storyboards” and continues over “Abstract prototypes” to “Detailed prototypes”. Throughout this process, four discrete levels of abstraction can be identified: “User Modeling”, “Scenario Modeling”, “Task Modeling” and “User Interface Design & Prototyping” (see Chapter 5.2.3—“Process Visualization”).

Process integration and vertical abstraction levels

## 4.3 Requirements for Tool Support

*“Everything should be made as simple as possible,  
but not simpler.”*

—Albert Einstein

After a detailed elaboration of requirements to a novel tool approach within this chapter and Chapter 2—“Theoretical Foundations”, we present the results of our findings in a summarized form. Nevertheless, we also present findings of others that relate to the research at hand. Therefore, general guidelines for UI tools and creativity support are examined. Thereafter, a compilation of requirements based on the categories of functional, technical and usability aspects is presented.

### 4.3.1 General Guidelines

General guidelines  
for design tools

Within the large HCI research community, we are not the first to investigate tool improvements. Therefore, it is crucial to look at general principles defined by previous research in next generation design tools as well as general principles for supporting creative processes. Respecting these guidelines is essential before eventually specifying concrete requirements.

#### Design Principles for Design Tools

General guidelines  
for UI tools

Campos and Nunes (2006) present general guidelines for tool support of user interface design processes, which are based on their research during a two-year period and development of the UI tools CanonSketch and TaskSketch. According to Campos and Nunes (2006, p. 14f) the most important guidelines that UI tools should follow are:

- **Explorability**

Design tools should effectively support the explorative nature of design processes. As the iterative and alternating nature of user interface design processes leads to a variety of artifacts, it is necessary to provide means to explore this space. Campos and Nunes (2006) regard common undo and redo mechanisms as inappropriate to support this requirement. Instead, a design tool should actively invite creativity in early process steps and force focus as the design progresses.

- **Expressiveness**

As modeling and design heavily relies on visual presentation the means of expression should not be limited. The designer needs to be supported in expressing his ideas with a variety of informal and formal tools. Design tools have to support this requirement to become adopted in practice.

- **Guidance**

A design tool should guide the overall design process just enough to make it successful, but not too stiff to constrain the actions of the designer. Campos and Nunes (2006) regard the employment of selected notations as efficient guidance.

- **Desirability**

As design tools are used by designers, they have to look engaging and attractive to become adopted in practice. As designers have a special sense for design, visual design and look & feel of both, employed models and design tools, needs special attention.

The guidelines "Explorability" and "Expressiveness" imply that UI design tools should be both "thinking tools" and "building tools". Most available UI tools focus on efficient support for most building tasks, at least in formal or semiformal ways. Tools that support the actual "thinking" task are rarely and hard to find in practice. We therefore look for principles for supporting creativity in thinking tools.

Implications for tool support



### Design Guidelines for Creativity Support

Shneiderman (2003, p. 209ff) describes a practical framework for designing user interfaces that support creative thinking, called "mega-creativity". According to Shneiderman (2000), there are three different perspectives on creativity: the inspirational, the structural and the situational model.

Three different perspectives on creativity

- **The inspirational model**

This perspective focuses on promoting techniques like brainstorming, free association and imaginative thinking. A mandatory requirement for creativity within the inspirational model is to break away from the existing mind set by allowing to perceive the problem with "new eyes". According to Shneiderman (2000) this technique can be supported by visual techniques that present loose relationships between artifacts, like mind maps.

- **The structural model**

This model emphasizes on analytical frameworks for creativity on the basis of previous work. The structural model favors visual information and models as key to understand a problem. By visualizing "how things currently work" with systematic exploration of models like flow charts or structured trees, understanding of the problem domain is promoted. According to Shneiderman (2000) this technique is supported by methods that support "going back", "making changes" to models and further exploration.

- **The situational model**

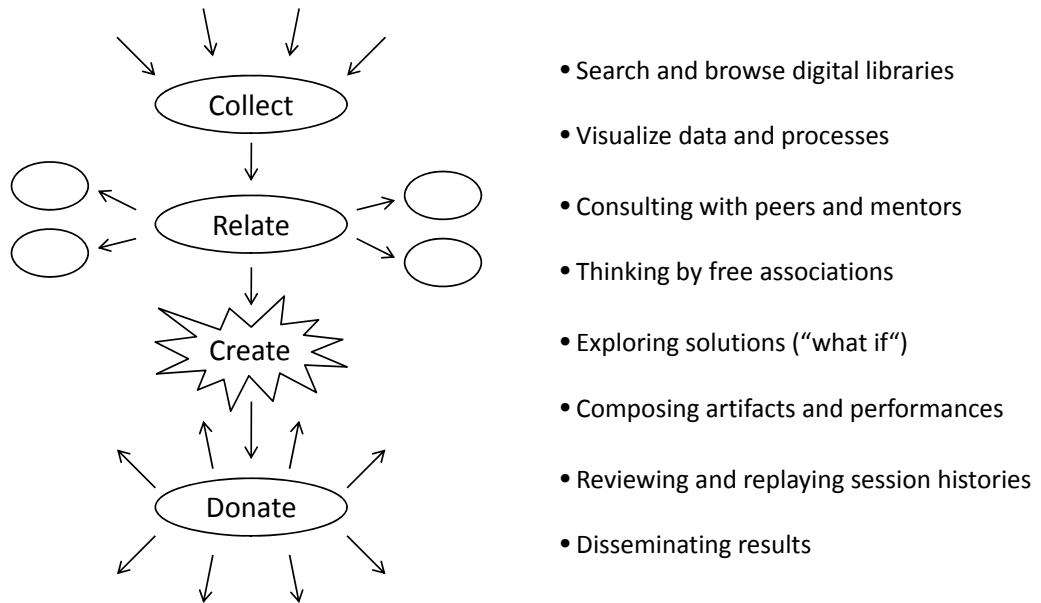
This perspective regards the social context as a key part of the creative process. Accordingly, the influence of social challenges, mentors and peers create a strong desire to innovate in pursuit for recognition. According to Shneiderman (2000), this model can be supported by interfaces that support access to previous work as well as consultation and discussion with members of the field.

Shneiderman (2003, p. 214ff) combines these three perspectives to a framework, which consists of the four steps: collect, relate, create and donate. Shneiderman (2000) also suggests eight concrete activities to integrate creativity in tool designs, which are shown in Figure 4.7. (Shneiderman, 2003, p. 214ff) argues for better integration of the proposed activities into creativity-supporting software instead of using separate tools for all eight tasks.

A framework for creativity

"The creativity framework will work only if there is integration of multiple creativity support tools.[...]However, the main challenge for users and designers is to ensure smooth integration across these novel tools and with existing tools such as word processors, presentation graphics[...]The first aspect of integration is data sharing and it can be accomplished by providing compatible data types and file formats.[...]A third aspect of integration is the smooth coordination across windows."

(Shneiderman, 2000, p. 122)



**Figure 4.7:** “Mega-creativity” Framework for Creativity Support in Software Tools, adapted from (Shneiderman, 2003, p. 221)

#### Creativity support guidelines

Based on this creativity framework, the Workshop on Creativity Support Tools 2005<sup>2</sup> Shneiderman et al. (2006) summarize concrete design principles for the development of tools that support creative processes from a number of acknowledged researchers. While some of these guidelines explicitly address user interface tools, they also aim on improving general creative processes that require a composition of novel artifacts in the domains of computer programs, scientific writing, engineering diagrams or artwork. Following list summarized the proposed principles (Shneiderman et al., 2006):

- Support Exploration
- Low Threshold<sup>3</sup>, High Ceiling<sup>4</sup>, and Wide Walls<sup>5</sup>
- Support Many Paths and Many Styles
- Support Collaboration
- Support Open Interchange
- Make It As Simple As Possible - and Maybe Even Simpler
- Choose Black Boxes Carefully<sup>6</sup>
- Invent Things That You Would Want To Use Yourself
- Balance user suggestions, with observation and participatory processes
- Iterate, Iterate - Then Iterate Again
- Design for Designers
- Evaluation of Tools

#### Implications for tool support

In reference to our demands on effective tool support in interdisciplinary specification, we

<sup>2</sup><http://www.cs.umd.edu/hcil/CST/>

<sup>3</sup>Ease-of-learning

<sup>4</sup>Support sophisticated projects

<sup>5</sup>Avoid predefined templates

<sup>6</sup>Avoid complex representation of objects but don't hide all details



think that creativity support should not suffer from the use of formal means of expression. Based on the above described guidelines, we are affirmed in our perception of both proposed process support and artifact support, as we already cover several guidelines, like variety of expression, relating of artifacts, collaboration, accessibility and iteration. However, we discovered an additional need for free association, exploration and simplicity for our tool design in order to more effectively support creative processes.

### 4.3.2 Specific Requirements

When looking back on the chapters 2—“Theoretical Foundations” and 4—“Analysis”, a wide range of ideas and implications for effective tool support were elaborated. In order to enhance overview and to maintain context within this work, the following tables present a summarized overview on elaborated aspects for tool-support and eventual implications for design. Consequently, the tables represent the constraints for tool-design, which will be presented in the next chapter. They list specific requirements which are prioritized with numbers to reflect their relative importance. Accordingly, requirements with priority-level “1” should get the highest attention in design, while requirements with lower priority levels (“2”, “3”, “4”) may have to retreat in important design decisions. All requirements are grouped into three categories: functional requirements, technical requirements and usability goals. By listing requirements separately and in a structured form, they also become a “checklist” for later reflection of design decisions and final evaluation of the system.

Summary of elaborated requirements for later reference

#### Functional Requirements

Functional requirements were elaborated within chapters 2—“Theoretical Foundations” and 4—“Analysis”. While chapter 2—“Theoretical Foundations” focused on the fundamentals for general User Interface Design support, Chapter 4—“Analysis” analyzed work practice and requirements for artifact support within common design processes. Accordingly, resulting functional requirements are divided into the subcategories: “General UID support” and “Artifact support”. Table 4.1 presents elaborated requirements for general support of interdisciplinary user interface design, while Table 4.2 presents requirements for artifact support within design processes.

Functional requirements: general support and artifact support

#### Technical Requirements

Technical requirements are implicit results of previously described functional requirements. Functional requirements therefore set constraints and general requirements to technical implementation. Table 4.3 presents the prioritized technical requirements which have to be respected in later tool-design.

Technical requirements are a result of functional requirements

#### Usability Goals

Table 4.4 presents usability goals that result from the target domain described in chapter 2—“Theoretical Foundations” and the previously described functional requirements. The table is divided into qualitative usability goals, derived from user characteristics and tasks, and quantitative goals, defining the acceptable user performance, based on a subset of high-priority requirements. In order to respect the different stakeholder characteristics, ease of use and ease of learning receive highest priority. Eventually, these issues determine whether the tool-based approach will be adopted in practice or not. Since hypertext-methods characterize an important part of the requirements for tool-design, ease of navigation and context awareness are also important usability aspects. Finally, qualitative usability measures will also have a critical impact on adoption in practice.

Usability issues determine adoption in practice

Priority	Required support	Design effort
1	Artifact management	Compile artifacts into a structured framework that reflects the design process. Relating artifacts should be grouped based on their level of abstraction. Visualize relations between artifacts.
1	Creativity & Innovation	Facilitate innovation and creativity by effectively narrowing the gap between conceptual design and physical design. Provide informal means of expression to promote creative thinking.
2	Specification & Design Rationale	Keep track of all artifacts during the design process and allow structured organization. Keep design decisions for later reference and extend traceability. Effectively communicate final designs and their rationale.
3	Communication & Collaboration	Bridge difficulties in communication among actors. Facilitate a common understanding in design by employing abstractions. Allow collaborative work by bridging the gap in technology.
4	Iteration & Alternation	Support the iterative nature of the overall development process. Allow flexible traveling from problem to solution space throughout the process.
4	Adaptability	Keep the process flexible and adaptable to various problem domains. Allow flexible choice of models and means of expression.

**Table 4.1:** Functional requirements regarding general UID support (based on Chapter 2—“Theoretical Foundations” and 4—“Analysis”)

Priority	Required functionality	Design effort
1	Diagramming	Facilitate graphical creation and manipulation of diagrams for modeling purposes. Integrate all required diagram elements.
1	Prototyping	Facilitate various forms of expression, from lo-fi to hi-fi representations. Support the iterative and alternating nature of prototypes.
2	Sketching	Support informal means of expression, like hand drawings, writings and markings.
3	Hypertext	Include hypertext features to enhance traceability and specification throughout the variety of artifacts.
4	Word processing	Facilitate word processing input for text-based artifacts.
4	Presentation	Include comprehensive presentation functionality to enable cooperation among actors.

**Table 4.2:** Functional requirements regarding artifact support (based on Chapter 4—“Analysis”)



Priority	Technical requirement	Implementation effort
1	Accessibility	Respect technical settings in the application domain. Take profit from technical opportunities and respect restrictions. Ease tool-usage and installation for all actors.
1	Separation of Concerns	Effectively separate concerns in design and implementation by creating a clear frontier in specification.
2	Support current Standards and Formats	Support commonly used standards and file formats to allow integration of existing work.
2	Facilitate teamwork and collaboration	Allow sharing and distribution of design artifacts among team members. Support asynchronous work as well as synchronous work.

**Table 4.3:** Technical requirements (based on Chapter 2—“Theoretical Foundations”)

Priority	Usability Goal (qualitative)	Design effort
1	Ease of Use	Respect discipline-specific tool usage and knowledge. Make interaction as simple as possible to allow all actors to actively participate in the design process. Employ compulsory functionality only.
1	Ease of Learning	Employ interaction concepts that are familiar to all actors and compatible to current work practice. Make use of metaphors that are already in use.
2	Ease of Navigation	Guide through the process and artifacts by offering navigation aids. Make progress within the process visible.
3	Context awareness	Make sure that actors are still aware of the “big picture” while allowing a focus on details within the design space.
Priority	Usability Goal (quantitative)	Design effort
1	Task efficiency (relative)	Make all supported tasks more efficient than they would be in real life without tool support. Make overall task achievement faster in comparison to employing a variety of different tools instead.
2	Task efficiency (absolute)	Make individual tasks-completion and response times as fast as possible.

**Table 4.4:** Usability Goals (based on Chapter 2—“Theoretical Foundations”)



## Chapter 5

# Design

*“Where you innovate, how you innovate,  
and what you innovate are design problems.”*

—Tim Brown

In the following, we will present the rationale behind the design of our specification tool based on the presented requirements. Therefore, a conceptual model is introduced by describing user characteristics and contextual work style. Thereafter, a metaphor is introduced that is utilized to map the conceptual model to a physical interface design. After investigation of appropriate interaction and visualization concepts, we will eventually present the physical design of our tool.

Outlook

### 5.1 Conceptual Design

The first step in creating a user interface design from a set of requirements is conceptual design (see Chapter 4.1—“Structured Approaches to User Interface Design”). As the elaborated requirements for tool-support are extensive and detailed it is nearly impossible to simply “invent” a proper physical solution that respects all functional requirements, technical requirements and usability goals. Instead, it is widespread UI design practice to employ a more abstract representation of how to realize the required actions in practice, a “conceptual model”. This abstract form of representation then reveals implications for physical tool-design. The developed “design concept” eventually resembles a user-centered and usage-centered foundation to the involved tasks.

Why conceptual design is important

The term “conceptual model” was first used by Norman (2002) to describe how a system is designed based on the designer’s mental model. According to Norman, the user of a system develops a mental model of how the system works through interpreting, similar to reading a text. This model is then used by the user to understand the system and its behavior. In practice, the user’s mental model is often different from the designer’s mental model. A major goal for the designer therefore is to understand the user’s mental model. Successful interaction requires creating a conceptual model of a system that is close to the user’s mental model.

What is a conceptual model?

Conceptual design is probably the most difficult task within the design phase. It requires understanding of users, work practice, creative thinking and a sense for adequate innovations. A clear and simple concept that seems natural to the intended users makes it easier for them to learn and use the product. Since a major usability goal of the research at hand is to revolutionize work practice by employing concepts that are familiar to all actors, current

A design concept that is close to practice is key to successful innovation

work practice has to be respected to the greatest extent. Eventually, this design step aims on the explicit construction of the mental concept that users have to adapt to when employing the tool. Therefore, a good conceptual design is the key to the adoption of a tool-design in work practice.

Personas,  
Metaphor and  
Scenarios

In the following, constraints and factors that led to the central UI design concept are described. Starting from describing and understanding users with personas, contextual work styles are described. Thereafter, a work metaphor is introduced, which resembles the central design concept. Consequently, the metaphor is evaluated against the constraints that result from previous examined guidelines, requirements and goals.

### 5.1.1 User Characteristics

Describing users  
with personas

Understanding the user's characteristics is not a straightforward process. Based on observations in work practice, questionnaires, interviews or focus groups, big piles of user data may arise. Each person that is being observed might differ slightly from other persons. At the same time, different users may have many things in common. To avoid an extensive description of users in detailed profiles, Cooper et al. (2007) provides a powerful concept of user models, namely "Personas". This chapter presents three different personas that resemble the main stakeholders within the context of this work. Thereafter, the personas are related within a "Role Map" (Constantine and Lockwood, 1999a).

#### Personas

"Personas provide us with a precise way of thinking and communicating about how users behave, how they think, what they wish to accomplish, and why. Personas are not real people, but they are based on the behaviors and motivations of real people we have observed and represent them throughout the design process." (Cooper et al., 2007, p. 75)

Identified personas  
are reduced to the  
main stakeholder  
groups

Personas are merely an "archetype" based on behavioral data. They exemplify behavioral patterns observed during observation. By utilizing personas, an understanding of project specific user goals in their specific contexts can be developed. In the following, three personas are presented which represent abstractions from our observations in work practice. This does not mean that the developed system is strictly focused on three different user types. Instead, they represent the three basic disciplines that participate in interdisciplinary design projects: Human-Computer Interaction, Software Engineering and domain expertise, in the following case exemplified by Business Engineering (see Chapter 2.4—"Interdisciplinary Design"). This choice of personas is based on our priorities in requirements in interdisciplinary design. As there is a wide range of disciplines that may influence interdisciplinary design, other personas may exist as well. For the sake of readability, these are not described further in this work, as their implications on design are less crucial.

Analysis: Persona  
Bill Jobs



Figure B.1 in B—"Conceptual Models" presents persona "Bill Jobs". He represents the Business Engineering domain, which is also a representation of the client in separated developments (see Chapter 2.4.2—"Separation of Concerns"). His focus and interests during interface design projects lie primarily in:

- Business goals
- Functional requirements
- Corporate values and marketing
- Communication

- Personnel and Financing
- Process progress and costs

“Bill Jobs” is not a central character in respect to interdisciplinary user interface development. He is rather an “observer” of the overall process. His major intentions are to communicate business goals and requirements, to understand the progress of the development, to steer the direction in terms of corporate interests and to keep abreast of costs and timeframes. Therefore, he has to communicate with HCI designers and software practitioners. Nevertheless, he is responsible for the overall outcome of the software development process. Consequently, he makes decisions about iterations and advancements throughout the process.

Figure B.2 in B—“Conceptual Models” shows persona “Donald Nielsen”. As HCI expert, he represents the central character in interdisciplinary development (see Chapter 2.4—“Interdisciplinary Design”). He is responsible for actual user interface design. His focus and interests lie in:

Analysis: Persona  
Donald Nielsen



- Requirements Analysis
- Conceptual Design
- Physical Design
- Team collaboration
- Evaluation & Design Decisions
- Specification

Nevertheless, he is also a crucial hub in information exchange between all stakeholders. He reports to “Bill Jobs” and communicates results to external suppliers. His concern is also to make his design efforts understandable to other stakeholders that do not have expertise in HCI. Donald and his team are the actual “craftsmen” that work on designing the user interface with all its details and constrains within a structured process (see Chapter 4.1—“Structured Approaches to User Interface Design”). They utilize a wide range of tools for modeling, design and evaluation (see Chapter 4.2.1—“Tasks, Techniques and Artifacts”). As additional challenge, they have to communicate their designs to external suppliers in specification documents. “Donald Nielsen” and its team are the main users of the system developed in this work. Therefore, they should influence tool-design the most.

Figure B.3 in B—“Conceptual Models” introduces the third and final persona, “Ian Ambler”. He represents the software engineering stakeholder in interdisciplinary development. His concern is to understand the specification of a user interface and its functionality. As head of actual implementation, he is responsible for programming the design into a tangible software system. His concerns therefore are:

Analysis: Persona  
Ian Ambler



- Communication
- Specification
- Implementation

A critical issue for communication is his lack of knowledge on HCI methods. As software practitioner, he is proficient in employing abstract models in software development but is not familiar to HCI models. As misunderstandings in specification may lead to increasing costs and exceeding timeframes, he prefers a comprehensive specification, executable prototypes and ideally a design rationale to understand the rationale behind the specified system.

## Role Map

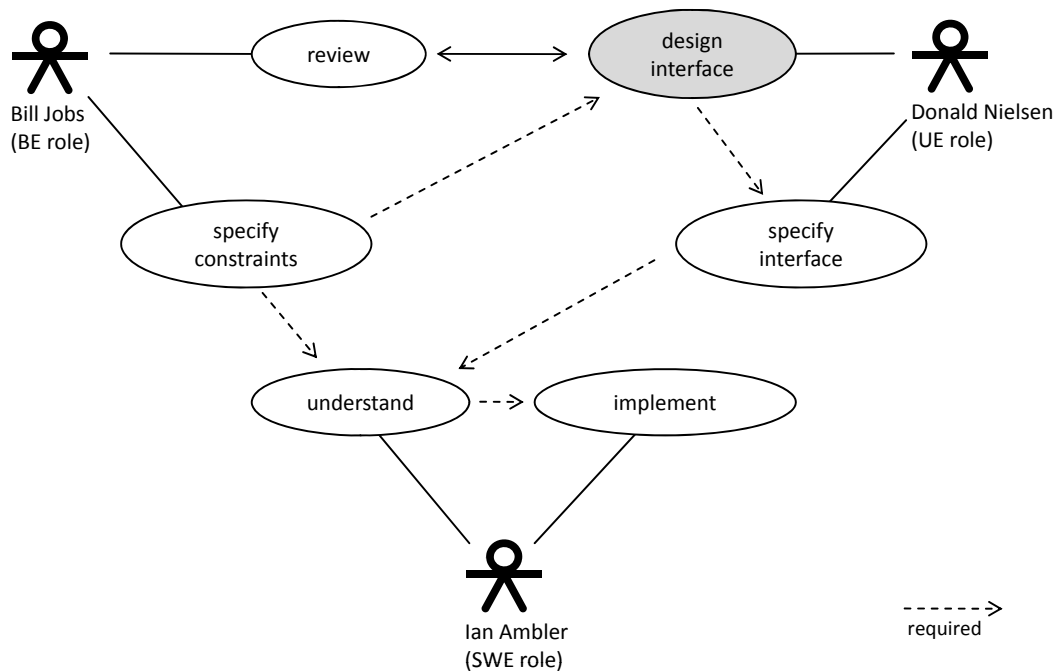


Figure 5.1: Role Map

## Collaboration

Figure 5.1 shows a role map (Constantine and Lockwood, 1999a) which is based on the previously described personas. This simplified model conveys a view on the main responsibilities of each actor and their dependencies. Responsibilities of all personas can be reduced to two abstract main tasks. Additionally, the overall process progress is visualized top-down. The "design interface" task is highlighted since it is the crucial part of the collaboration chain in respect to the work presented in this thesis. Additionally, this task is collaborative itself, as "Donald Nielsen" works with a team of designers. The role map also reveals the three frontiers of collaboration. While the frontier between specification tasks and the "understand" task is principally one-way, the frontier between "design interface" and "review" is two-way. Consequently, a tool approach for interdisciplinary user interface design has not only to support the main task "design interface" but also the strongly connected tasks "specify interface", "review" and "specify constraints". In respect to the interdisciplinary nature of the design process, we define two levels of collaboration: Internal and external collaboration. For the sake of readability and consistency, we will distinguish both levels by calling them "Collaboration" and "Communication":

- **Collaboration** (internal collaboration)  
is defined as the collaboration of designers within the main task, "design interface".
- **Communication** (external collaboration)  
is defined as the collaboration of all stakeholders within the overall interdisciplinary development process.

## 5.1.2 Contextual Work style

After identifying and analyzing user characteristics, it is now essential to investigate the methodology and the physical setting in which interdisciplinary design takes place. Similarly to the variety of described approaches to user interface development (see Chapter 4.1—“Structured Approaches to User Interface Design”), there are also virtually unlimited numbers of actual work styles in practice. Nevertheless, they can also be turned into a common work model by employing scenarios and corresponding tasks. As one major goal of this research is to focus on innovation and creativity support, work styles that facilitate these factors are examined further for tool-design. From assessment of recent research literature, like Shneiderman (2003), Campos and Nunes (2006), Cherubini et al. (2007), Ju et al. (2007) and Guimbretiere et al. (2001), a clear trend to creative and informal work styles in UI design approaches and practice can be found. The following work style analysis starts with the most important scenarios of collaboration, “incremental refinement” and “places for collaboration”. Consequently, it is analyzed from a HCI perspective, e.g. “Donald Nielsen’s” work practice. Thereafter, the analysis focuses on aspects related to communication, the tasks “review”, “specify constraints” and “specify interface”.

Investigating  
contextual work  
style

### Incremental Refinement and Transitions

As described in chapter 4.1—“Structured Approaches to User Interface Design”, the “design interface” task follows a process of incremental refinement. The incremental nature of the design process is based on the subsequent process steps as well as iterations between design artifacts. Refinement is not restricted to the later phases of interface design but can be found in all process steps. As iterations are necessary, initial constraints and models are frequently refined or changed. Therefore, the process of user interface design is accompanied with a constant movement from less detail to greater detail. At the same time, artifact representations move from coarse granularity to fine granularity. While early artifacts usually have a higher grade of abstraction, designers also tend to look at the big picture early on in the process. Lin et al. (2000) give a valuable insight into the work style of interface designers. They usually keep the big picture in their minds while they travel on through the process. As early process steps feature the highest grade of abstraction, they are usually realized with the most informal tools, like sketching, even though they will eventually lead to computer-supported artifacts or designs. Some designers even work with paper throughout the overall process or utilize sketches before they are actually using computerized tools to realize their ideas. Consequently, hand-sketched artifacts can be observed throughout all process steps and artifacts. This fact can be explained by the explorative nature of paper sketches.

Process defines  
incremental  
refinement



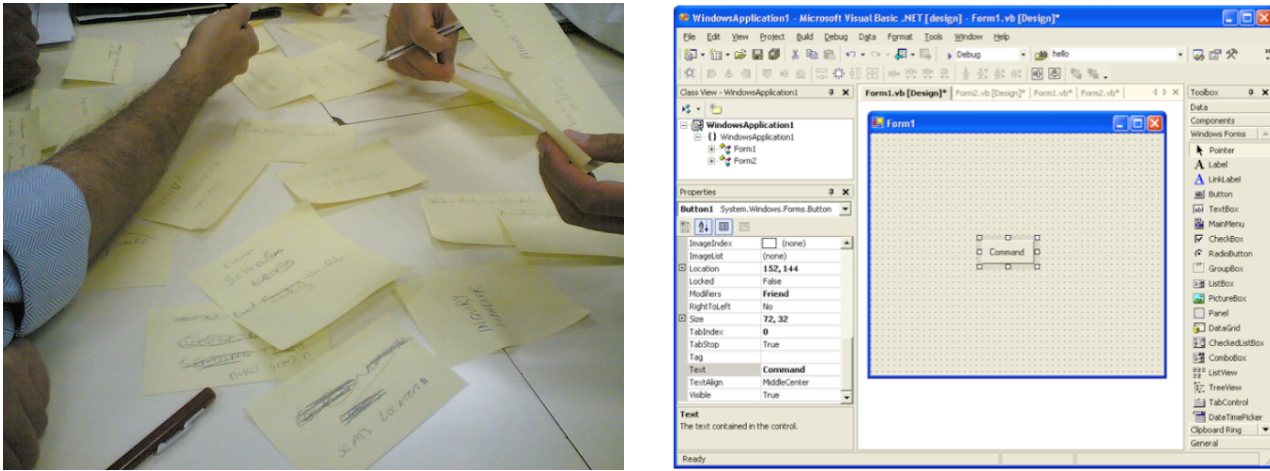
“Sketch ideas, in contrast, assist visual thinking and they are especially well suited to preliminary thinking and brainstorming. The coming of ideas is one of the important things about visual representations. They allow side-by-side comparison.”

Bill Verplank in Preece et al. (1994, p. 467)

Sketches are an efficient expression to “work through” ideas before using more formal and precise tools, which also bring significant more effort. They are an effective tool to try different things out, or evaluate alternative ideas or designs. As papers can easily be compared for evaluation, they are primarily employed to explore the space of possibilities. Nevertheless, eventually there is no way around switching to software tools in more advanced process steps as the result of the process is an interactive software system. One reason for an early switch to software tools is the ability to incrementally modify artifacts.

Why sketching is  
important





**Figure 5.2:** A work transition from paper to a software tool (Campos and Nunes, 2006)

“The beginning of each step I’ll do on paper. As soon as I feel like I’m going to be doing any design revisions, I’ll move to [an electronic tool] because it’s easier to make changes to these things.”

Interview with a designer in Lin et al. (2000, p. 512)

There is no way around software tools

Another benefit of employing software tools is the ability of replication and distribution. Electronic artifacts are also more precise and express a higher level of detail. However, exactly this precision is the major reason why designers sketch ideas on paper. By using software tools instead of sketching in early process steps, the results often bring a higher degree of formality. As early process steps resemble unfinished ideas, a formal representation is not well suited. Using formal tools make representations look like the real thing, which makes it harder to further develop or discard these ideas. Formal representations show too much detail, like specific fonts or colors that might distract evaluation of designs from the important focus, the larger concept. Especially when designs have to be communicated to external stakeholders, overspecified representations that look too formal may lead to confusions (see Chapter 2.4.3—“Communication & Specification”). Precisely because of the wide range of abstractions that can be found in the design process these transitions from informal to formal and back can be observed frequently throughout the process. Sketches of artifacts and models are only one example, but transitions can also be found from low fidelity prototypes to medium and high fidelity prototypes. Therefore, just like the transition from paper to software tool (see Figure 5.2), other transitions have to be bridged throughout the process.

“There is no orderly progression from high-level, abstract overview toward low-level, concrete detail. Designers bounce between abstraction and detail, from analysis to implementation, almost on a moment-to-moment basis.” (Constantine and Lockwood, 1999a, p. 38)

Five frequent transitions in practice

Transitions between artifacts during design steps are exceptionally diverse and frequent in user interface design. This fact can be explained by the concurrent use of various forms of abstraction and expression combined with the iterative nature of the process. According to Campos and Nunes (2006) the most common transitions in design practice are:

- **Problem space to solution space**

Resembles the transition from problems in the real world represented as models to a interface system as a solution to these problems (e.g. Conceptual Model to Screen Design)



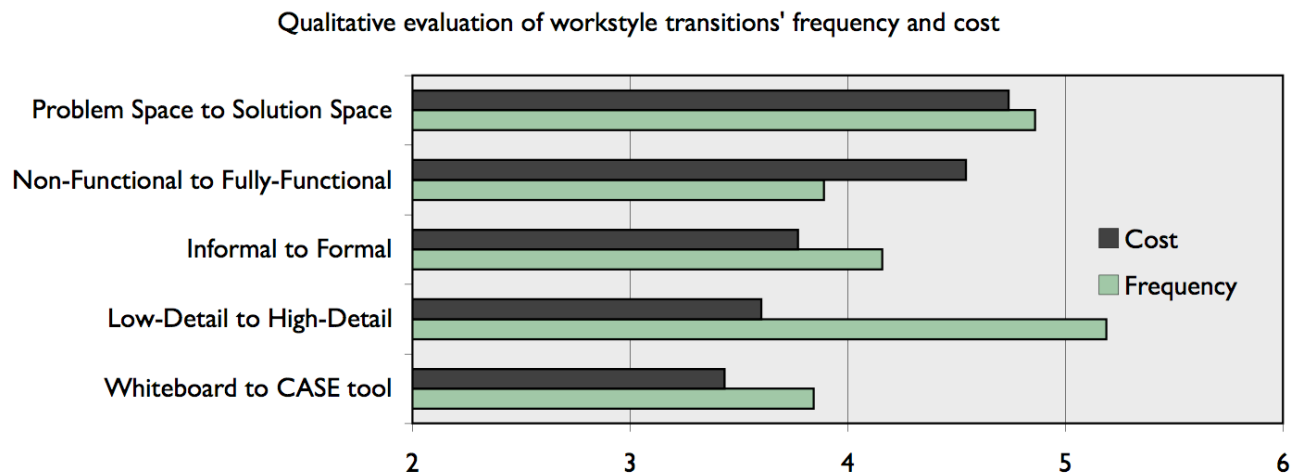


Figure 5.3: Transitions in work artifacts within UI design processes (Campos and Nunes, 2006)

- **Non-functional to fully functional artifacts**

As in early stages of the process artifacts are rather static, they tend to be more functional toward the the end (e.g. mockups vs. functional prototypes)

- **Informal to formal artifacts**

Early informal means of expression, like sketches of screen layouts, are transferred into more precise digital representation with basically the same level of detail (e.g. sketches vs. wireframes)

- **Low-detail to high-detail**

Iterative development leads to transitions from early experimental setups to more detailed designs towards the final solution (e.g. storyboard vs. prototype)

- **Whiteboard to CASE tool**

Hand writings or sketches of models on whiteboards or paper are transferred into dedicated CASE tools (e.g. use cases)



The chart shown in Figure 5.3 shows transitions, their frequency and costs found in current practice. The provided data is based on qualitative questionnaires of designers in practice. The designers were evaluating work transitions by selecting a value from a 7-point Likert scale, ranging from 1 - low to 7 - high.

The averaged results show that the most frequent transition is from "Low-Detail to High-Detail". The second most frequent is discovered as moving from "Problem Space to Solution Space". The latter is also identified as the most difficult one. Accordingly, support for these two transitions is considered as top priority. "Non-Functional to Fully-Functional" has higher costs than frequency. Consequently, this transition makes up the most difficult one in cost-frequency ratio.

Important transitions

### Places for Collaboration

Beyer and Holtzblatt (1997, p. 13f) describe the physical work environment for cross-functional teams as one of the most important issues in cooperative design. The major concern in cooperation can be formulated as two simple questions: Where are such teams supposed to meet? How are they exchanging information? According to Beyer and Holtzblatt (1997, p. 13), the most common work environment in design practice is the cubicle (office room). As office rooms are not suited for one ore more people working together, meeting rooms are utilized for team meetings. However, meeting rooms do not facilitate efficient

Most work environments do not support creative thinking

support for displaying physical design artifacts, as it would be required to spread them out before each meeting and eventually to collect them for the next meeting. Because these constraints make collaborative meetings difficult, communication in cross-functional teams is seriously harmed. Nevertheless, according to Beyer and Holtzblatt (1997, p. 14), face-to-face design is crucial to successful collaboration.

“Working together effectively means having workplaces where real work, done by multiple people working face-to-face, can happen.”  
(Beyer and Holtzblatt, 1997, p. 14)

Dedicated design rooms facilitate creative cooperation



One way of providing such a workspace is to employ a dedicated “Design Room” where cross-functional teams meet and cooperatively work together. These rooms stand for themselves as an approach to collaborative working in a design team. Design rooms support cross-functional teams by providing an environment that allows exploring ideas during early stages of design and by keeping work artifacts together throughout all development phases. They have been proven to successfully facilitate a human-centered overview on design and in promoting communication, discussion and generation of alternatives (Bennett and Karat, 1994). Detailed descriptions of appropriate room settings can be found in various works, like Preece et al. (1994, 2002, p. 480, p. 294ff), Beyer and Holtzblatt (1997, p. 203f), Carroll (1991, p. 269ff), Bodker et al. (2004), Bodker and Buur (2002) and Bennett and Karat (1994). Figure 5.4 shows an exemplified layout of a design room described in Karat and Bennett (1990).

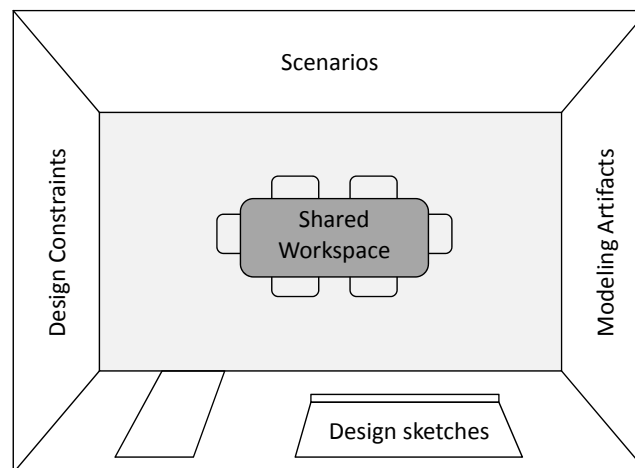


Figure 5.4: The Design Room, adapted from Preece et al. (1994, p. 480)

Touring the design room



During design processes, a variety of design artifacts are posted on walls or are written on whiteboards, like requirements, goals, guidelines, ideas, drawings, models, pictures, storyboards, sample scenarios or prototypes. These artifacts may be structured and rearranged in order to maintain overview. In the following, we will see that informal expression and spatial relations between artifacts are crucial for creative collaboration.

“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 up an idea up into something larger.”  
(Beyer and Holtzblatt, 1997, p. 202)

While artifacts are usually produced outside the design room, they are brought in for discussion in design meetings. The team then works collaboratively on a table in the center of

the room. Actors can point at artifacts on the wall or simply take them from the wall for discussion. On making decisions, sticky notes can be attached to the artifacts to mark them for later reference.

According to Beyer and Holtzblatt (1997, p. 203f), design rooms bring various benefits: A dedicated design room means that work artifacts remain between design sessions, which enhances traceability within the process. The shared workspace enables actors to efficiently communicate and discuss design work. Transitions between paper-based process artifacts can be bridged easily by simply arranging them on a wall in a way to make them comparable. Subsequent artifacts or related artifacts may also be connected by simply drawing lines between them. The immense wall space allows making the overall design process visible. Iterative process artifacts are as easy to compare as alternative designs.

Benefits of design rooms: creativity and communication

"I think one of the first things you learn in design is to put forward a number of alternatives so that you can then compare them. Having a lot of display space is important for doing this because you can make them visible. One of the things you can do with visual things is superimpose them, or put them side by side and quite often when you start doing this, you like one better than another."

Bill Verplank in Preece et al. (1994, p. 467)

Throughout the process, an extensive number of artifacts are produced. Transitions between artifacts are narrowed by a spatial organization of design artifacts. By facilitating both overview on the context and detailed views on artifacts at the same time, creative thinking and decision-making are effectively supported. As the paper-based display space is quite large, interface designs can be much bigger than on a screen, which allows more people to look at them at the same time. The collection of artifacts on the wall does not only facilitate communication but is also the "public memory and conscience" (Beyer and Holtzblatt, 1997, p. 203) of the design team. It is too hard for people to keep every aspect of work in their head at once. Artifacts on the wall remain and are instantly available for reference. The simple fact that artifacts are always there to remind people enhances overall process understanding. Beyer and Holtzblatt (1997, p. 204) emphasizes the role of the human perception abilities in the success of design rooms:

Design rooms facilitate spatial organization of information

"What's more, people have a spatial sense that helps keep the data organized. It's common for someone referring to customer data to back up a claim by pointing at part of an affinity that covers all four walls. They nearly always point to the right place."

(Beyer and Holtzblatt, 1997, p. 204)



Design rooms are currently not widespread in practice and can nearly exclusively be found in professional design settings. Their employment is quite expensive, as they have to be dedicated to a specific design project and therefore take a lot of space that is frequently unused when actors work separately. However, this is not the only reason. While design rooms are well suited for collaborative and creative design, there are also some downsides. Some of these can be revealed by looking at a design room in practice. Figure B.4 in B—"Conceptual Models" shows a design studio at IDEO<sup>1</sup>, which is one of the most innovative companies in the world<sup>2</sup>. A quite large diversity of information is posted on the wall. Artifacts range from sketches to photographs over sticky notes to large quantities of printouts from digital media.

Downsides of design rooms

Design rooms primarily support paper-based artifacts. As digital information is an essential ingredient of user interface design, it needs to be printed out first before it can be attached to a wall. This major drawback leads to the problem that artifacts cannot easily be updated as

Enhancing the design room

<sup>1</sup><http://www.ideo.com>

<sup>2</sup>[http://www.fastcompany.com/fast50\\_08/ideo.html](http://www.fastcompany.com/fast50_08/ideo.html)

the process iterates. While actors tend to employ informal means of expression during brainstorming or idea generation phases, they need to sort out ideas during post-brainstorming phases. During these phases, generated ideas have to be transcribed and archived for later reference. As handwritten artifacts are hard to transfer into permanent storage, they are even harder to transfer into searchable digital archives. Informal expressions, like scribbles or handwritings may also lead to misunderstandings within the team as they imply a certain degree of interpretation. Moreover, actors, which have a background in non-design related disciplines, like marketing experts or software engineers, may simply be overwhelmed by the sheer number of visible artifacts. Similarly, new team members that enter a design process, may have issues in keeping up with the design rationale, as outdated artifacts are being replaced throughout the process. However, a well-maintained design room, which employs some way of imposing a structure on posted process artifacts, may handle some of these issues. It may even be suitable to communicate designs and process progress to external stakeholders. To cope with some of the described issues, experimental computer supported collaborative environments are focusing on using technology to support this kind of group work (see Chapter 5.2.4—“Collaboration”).



## Communication

Common language  
crucial

Common understanding is crucial for external collaboration, which is facilitated mainly by communication. Effective communication requires the sharing of process artifacts. Therefore, all artifacts produced within the overall process should be accessible to all actors. Nevertheless, not all communication tasks require the same kind of work style and focus. On the one hand, communication within the “review” task is two-way and focuses on artifacts within the design process as well as the “big picture”. On the other hand, communication in “specify” tasks is mainly one-way and builds up a clear frontier, which focuses more on actual results. However, both styles of communication require a common language to avoid misunderstandings.



“Effective communication requires access to each other and each other’s ideas and partial designs, use of a common ‘language’, for instance, a formal specification language, dataflow diagrams or ER diagrams, a way of tracking and recording ideas and decisions, and so on.”  
(Preece et al., 1994, p. 479)

Visual specification  
techniques



As previously described (see Chapter 2.4.3—“Communication & Specification”), the most common forms of communication in user interface design practice are meetings, written documentations and formal specifications. As formal notations have significant disadvantages (see Chapter 2.4.3—“Communication & Specification”), ways of providing a common language by employing models as primary medium for communication are increasingly getting popular in practice. As they are abstract representations, they are easier to understand than formal notations, but also leave room for interpretation. Consequently, these approaches to communication are accompanied with “graphical” or “visual” specifications in form of prototypes or interactive dialog flows, which make up for the lack of formality. The process of building up such a visual specification gained popularity as advanced prototyping tools arised, that enable designers to program actual prototypes without having to code them manually.

“This technique can be referred to as ‘programming by representation’ since the programmer is building up the interaction dialog directly in terms of the actual graphical interaction objects that the user will see, instead of indirectly by means of some textual specification language that must still be linked with the presentation objects.”  
(Dix et al., 2003, p. 313)

Connecting the created interactive prototypes with the rationale behind the interface, represented by models is one of the major concerns of the interdisciplinary specification framework introduced in Chapter 4.2.2—“Interdisciplinary Selection of Artifacts”. Utilizing this novel form of specification, we argue that communication between stakeholders can be significantly enhanced. Nevertheless, the specification has to be made accessible to the actors during the process for reviewing and as bridge between the frontiers to implementation (e.g. as XAML document). As communication is primarily concluded within meetings and via various forms of portable media, like hypertext, a supporting tool should allow presenting the specification and the rationale in these contexts. We think that a combination of this method with visual presentation techniques that are already used in practice, like presentation software (e.g. MS PowerPoint) promises successful adoption. An interactive form of representation that allows browsing contents based on the focus of communication and desired abstraction seems a natural fit to these problems.

Communicating specification and rationale



### 5.1.3 The Design Room Metaphor

We think that the “Design Room” is an adequate metaphor on how to effectively address contextual work style with tool-support, as the basic concept of the design room is proven successful in practice. Actors that may work effectively in design room environments are likely to adapt to a conceptual model that employs this familiar metaphor. We therefore define the basic conceptual model for our innovative tool-support as the design room metaphor. In the following, we will introduce metaphors as interaction design concept and describe the chosen metaphor in detail, before we evaluate its application to the target domain.

The design room as metaphor for work style support

#### Metaphors as Foundation to Interaction Design

Employing metaphors as a foundation to physical interface design is an arguably acknowledged technique to narrow the gap between the user’s mental model and the designer’s mental model. It is also a quite popular tool for designers to achieve consistency and ease of learning in their designs. Metaphors originate from the world of linguistics and literature. Their fundamental idea is to transport concepts from one domain to another. A metaphor therefore is not just a literary thing, but a fundamental to the way we think (Benyon et al., 2005, p. 596). Metaphors are based on cognitive semantics. According to Benyon et al. (2005, p. 596), our thinking starts from the metaphorical use of basic concepts, or “image schemas”. In our minds, we are persistently referring to concepts we know from the real world as we approach a new domain. This basic concept of our minds can be applied to the domain of user interfaces. Interface metaphors aim at transporting a mental model of the system to the user. The user perceives this mental model simply by observation of the interface and prediction of its response to his actions based on his experiences in real-life (Preim, 1999, p. 166).

Why employ metaphors?



“In the development of interactive systems we are constantly trying to describe a new domain... to people. So we have to use metaphor to describe this new domain in terms of something that is more familiar.” (Benyon et al., 2005, p. 596)

Metaphors, or interface metaphors in interaction design are primarily employed to facilitate ease of learning. A popular example of a metaphor is the desktop metaphor that is found in the popular operation systems such as Windows XP and MacOS. This metaphor carries the well-known concepts of a real office, the desktop, files and folders to the world of computing. According to Moggridge (2006, p. 53), the desktop metaphor was an extension to the already employed typewriting metaphor (see Chapter 1—“Introduction”) used for word processing. There were already attempts to implement this kind of metaphor before by using complex, three dimensional simulations (Moggridge, 2006, p. 54). Nevertheless, the breakthrough to

A lesson in metaphors: the desktop metaphor

a successful application was the simplicity of Mott's two-dimensional iconic representation. This reveals the fundamental to applying metaphors: simplicity. When applying metaphors the designer always has to realize that metaphors in design are really "blends".

Successful metaphors are blends

A blend takes input from at least two domains: The characteristics of real life, described by the source, and characteristics of the target, the computer. Blended metaphors will eventually have more features on a computer than they had in the original domain, but will still work for the user. Consequently, for a metaphor to work, it is not essential to reflect an exact copy of the source, but to find an adequate mapping to the target. Therefore, metaphors are not easy to transfer to an actual user interface. They should not be forced and still have to be consistent with real-world expectations, even if they have to bring some degree of abstraction to be successful. Cooper et al. (2007) advises as one of his design principles:

"Use metaphors if you can find them, but don't bend your interface to fit some arbitrary metaphoric standard."  
(Cooper et al., 2007, p. 279)

Careful application of metaphors is important

Therefore, it is essential to carefully design a metaphor and evaluate all aspects of it before looking for ways of how to apply it to a user interface. If a metaphor does not fit, a variety of problems arises. The implicated meanings and associations of metaphors are their strength, but also their weakness. This controversy in the use of metaphors is what Hudson (2000) describes as "a double-edged sword". There are many examples of failed mappings of metaphors, like the "wallpaper" on a "desktop" or the awkward feature of the Macintosh, that allows to drag a floppy disk to the wastebasket in order to eject it (Hudson, 2000, p. 13). As metaphors are bent to fit functionality, the original power of the metaphor is lost due to a lack of consistency. However, Hudson (2000, p. 14-15) and (Benyon et al., 2005, p. 598) provide some guidelines for good metaphor design:

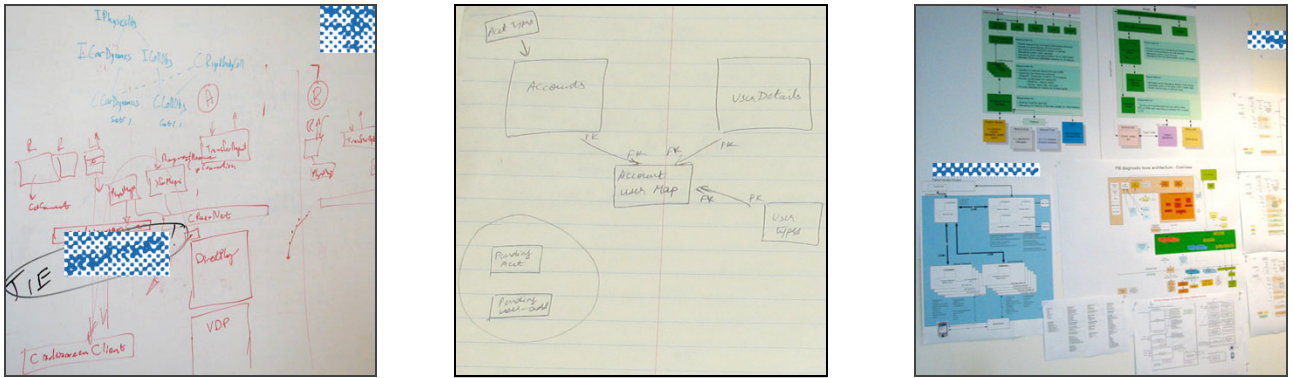
- Apply metaphors only for a good reason.
- Metaphors do not have to be complete, but interfaces need to provide adequate clues.
- Metaphors operate on systems of relationships; do not use concepts out of context.
- Focus on metaphors in objects and actions; do not force realistic visual representations.
- A metaphor should not rely on mere appearance; it should behave like one, too.
- Try to get as close to the original domain as possible.
- Mix metaphors only when they originate from the same context.

### Composing the Metaphor

Combination of Metaphors is common in complex systems

According to Preim (1999, p. 177), it is not possible to create a complex interactive system by describing it with just one metaphor. Instead, a combination is frequently employed. By closely looking at the desktop metaphor, a range of combined metaphors can be found (Preim, 1999, p. 177): Windows are themselves a metaphor for panes that display a part of a document. Menus in windows were inspired by menus in restaurants even if they look inherently different. Experiences revealed that combinations of metaphors are possible and do not confuse the user if they are applied in the right way. Nevertheless, an important requirement in combining metaphors is that they should have some content-related relationship (Preim, 1999, p. 177). As one essential ingredient of design rooms is one or multiple





**Figure 5.5:** Various forms of the whiteboard concept found in design practice (Cherubini et al., 2007)

whiteboards, we look upon the “Design Room Metaphor” as a combination of the already proven “whiteboard” metaphor with the “room” metaphor. While the whiteboard metaphor is already widespread in design practice and can be found in various software tools, like diagramming, CASE tools, presentation and drawing software, the room metaphor is primarily employed in experimental collaborative interfaces. In the following, the characteristics of both metaphors are briefly described.

The referred concept of a whiteboard may have different appearances, as shown in Figure 5.5. While early informal drawings are usually created on office whiteboards with non-permanent pens, others might be sketched on paper. Pinning printouts of electronic artifacts on walls can be seen as an extension to the original idea of the whiteboard. As many popular metaphors in interface design - like the desktop - the whiteboard originated from the first widespread interface metaphor, the typewriter (Shneiderman and Plaisant, 2004, p. 216). In modern WYSIWYG<sup>3</sup> word processing software, the user looks upon a “page” of text. The virtual document is shown, as it will appear when printed. Actions of the user are immediately visible, like they would be on a real typewriter. This metaphor started a fundamental interaction concept of what is known as “direct manipulation” (see Chapter 5.2.1—“Whiteboard Interaction”). A wide range of software interfaces applied this concept on other domains, like slide presentations (e.g. PowerPoint), desktop publishing (e.g. Adobe Pagemaker) and general-purpose drawing or diagramming software (e.g. MS Paint, MS Visio). As drawings and diagrams are primarily sketched on whiteboards in real-life, these applications apply a whiteboard metaphor in their user interfaces. Table B.1 in B—“Conceptual Models” presents the basic features of this metaphor, based on descriptions of Mynatt (1999) and Cherubini et al. (2007).

Whiteboards are also used for collaboration in real-life, as they are visible to multiple persons at the same time. Consequently, some experimental collaborative computer systems utilize the whiteboard interface metaphor not only to manipulate and organize drawings, but also in order to facilitate collaboration. Popular examples for such systems are the Tivoli system (Moran and van Melle, 2000), the Range whiteboard (Ju et al., 2007), the SUMLOW UML design tool (Chen et al., 2003) and Liveboard (Elrod et al., 1992).

In contrast to the whiteboard, the room metaphor is rarely found in practice. As there are inherently less implementations to this interface metaphor, it is crucial to describe its concept. Rooms are primarily used for collaboration. Depending on the purpose, there are different interpretations to this metaphor. While some implementations utilize this metaphor for navigation (multiple room setup), in context of collaborative applications it usually consists of one room. Card and Henderson (1987) presented an early design a multiple, virtual-workspace interface to support user task switching by utilizing room concepts. It was initially introduced as a solution to a lack of screen space. Today, similar concepts are found in desktop managers, which align screens based on spatial organization, using virtual doors

Design room as a combination of existing metaphors



The whiteboard metaphor



The room metaphor



<sup>3</sup>What You See Is What You Get

to move from space to space. Single room metaphors for collaboration follow a different approach. Table B.2 in B—“Conceptual Models” presents features of the room metaphor, based on research of Greenberg and Roseman (2003).

Successful implementations of the room metaphor can be found in TeamRooms (Roseman and Greenberg, 1996, 1997), which is today known as TeamWave and was commercially available as the TeamWave Workspace<sup>4</sup> product. Other acknowledged research, which is utilizing the room metaphor, is COLAB and Cognoter (Foster and Stefik, 1986).

How to combine? We seek to combine some of the features of both metaphors to build up a “Design Room”. While the focus of the design process lies in the creation, manipulation and relation of artifacts on the whiteboard, collaborative features of the room metaphor may also turn out helpful in physical design. Accordingly, the following chapter focuses on the evaluation of a combined metaphor against previously described guidelines and requirements for tool-support.

## Evaluation

Evaluation of feature support in both metaphors Before attempting to implement a metaphor, its features have to be evaluated in reference to the initial requirements. Therefore, this chapter examines the benefits and drawbacks resulting from utilization of the proposed “Design Room” metaphor. As a first step to evaluation, suitability of features provided by both metaphors are examined against requirements, which were defined by general guidelines for user interface tools (see Chapter 4.3.1—“Design Principles for Design Tools”), the creativity framework (see Chapter 4.3.1—“Design Guidelines for Creativity Support”) and functional requirements elaborated within the analysis (see Chapter 4.3.2—“Functional Requirements”). Thereafter, implications for design and necessary improvements are described briefly.

Mapping of features to requirements Table B.3 in B—“Conceptual Models” shows a mapping of functional requirements to the features that characterize both metaphors. The table is divided into the four categories of requirements that influence functionality of tool-support (see Chapter 4.3—“Requirements for Tool Support”). The requirements identified within each category are listed in the second column. Features of both metaphors are mapped to their related requirements based on categories identified in Tables B.1 and B.2. Missing feature support is highlighted, as these requirements demand for additional consideration in design.

Evaluation of whiteboard features The whiteboard metaphor primarily offers features that support a creative design process. It efficiently facilitates informal means of expression and allows integrating nearly any paper-based artifact. Sharing of a limited number of these artifacts is also supported, as multiple actors have access to the whiteboard, either asynchronous or synchronous. The whiteboard also promotes creative thinking, as it provides an informal space for collection and expression of ideas. Clustering and organization features support group discussions and externalize a common vision. Whiteboards provide some limited features in arranging and relating artifacts, which facilitates decision-making by comparing and evaluating iterative or alternative artifacts. As sticky notes can be attached to all artifacts, whiteboards also allow collecting feedback from actors. Nevertheless, the whiteboard lacks some important features. It does not provide features that support efficient guidance on the overall design process. Its limited space and limited flexibility in terms of organization and sharing is restraining long time collaboration. As whiteboards only support static artifacts, expression and flexibility throughout the iterations within design processes are constrained.

Evaluation of room features The room metaphor enhances the whiteboard metaphor primarily in collaborative aspects. It resembles a meeting place for collaboration among actors. It provides more space for collaboration and advanced features for organizing content. Artifacts can be focused for discussion by navigation between spatial locations in the room. Areas of interest can be separated within the design process as dedicated whiteboards or walls. Therefore, it is more

<sup>4</sup><http://www.markroseman.com/teamwave>



suitable to support guidance on the design process. The spatial organization features present in a room enable actors to relate process artifacts in a more comprehensive manner. Work transitions can be bridged more efficiently within this shared workspace. The persistence of artifacts and tools within the room facilitates the public team memory and represents a collection for later reference. The inhabitation of the room by one or more actors from different disciplines contributes to the adaptability. Accessibility of artifacts for all actors also supports presentation demands.

## 5.2 Collecting Concepts - Interaction & Visualization

in the following, we will introduce and discuss relevant user interface concepts regarding interaction and visualization. Because the whiteboard metaphor is the fundamental for creating and manipulating process artifacts, interface implementation concepts are presented as a first step of investigating suitable interaction concepts. In further steps, spatial navigation concepts, which support transitions between artifacts, visualization concepts for the design process and interface concepts for collaboration, are presented and examined for physical design.

Stepwise  
investigation

In the previous chapter, a conceptual model was developed upon user characteristics, work style and a metaphor, which supports mapping contemporary work style to a new domain. While user characteristics demand for communication support and clear frontiers in collaboration, the work style analysis revealed that various work style transitions have to be bridged throughout the process. The proposed "Design Room" metaphor aims at combining features from the whiteboard metaphor and the room metaphor in a manner that efficiently supports creativity, communication and work style transitions. As of the described evaluation, some requirements do not have direct mappings to metaphor support, namely "Hypertext" and "Word processing". Nevertheless, integration of these features in tool design has to be supported. Eventually, the strength of the whiteboard metaphor in supporting creative design and creation of artifacts is to be combined with the collaborative features that characterize the room, by employing innovative interaction and visualization concepts. We therefore set the main tasks of investigation for proper physical design of an interface as:

Investigation  
concepts for proper  
mapping of the  
conceptual model



- **Whiteboard Interaction**

Build up upon the whiteboard metaphor for artifact creation and manipulation as well as integration of dynamic artifacts into the whiteboard concept.

- **Spatial navigation**

Harness spatial features of both metaphors for organization and thinking support. Ease transitions between artifacts by smooth switching with navigational aids.

- **Visualization**

Visualize spatial relations to support traceability, creativity and thinking processes. Provide additional measures of visualization to guide the design process as well as overview on the overall process while allowing detailed views on demand.

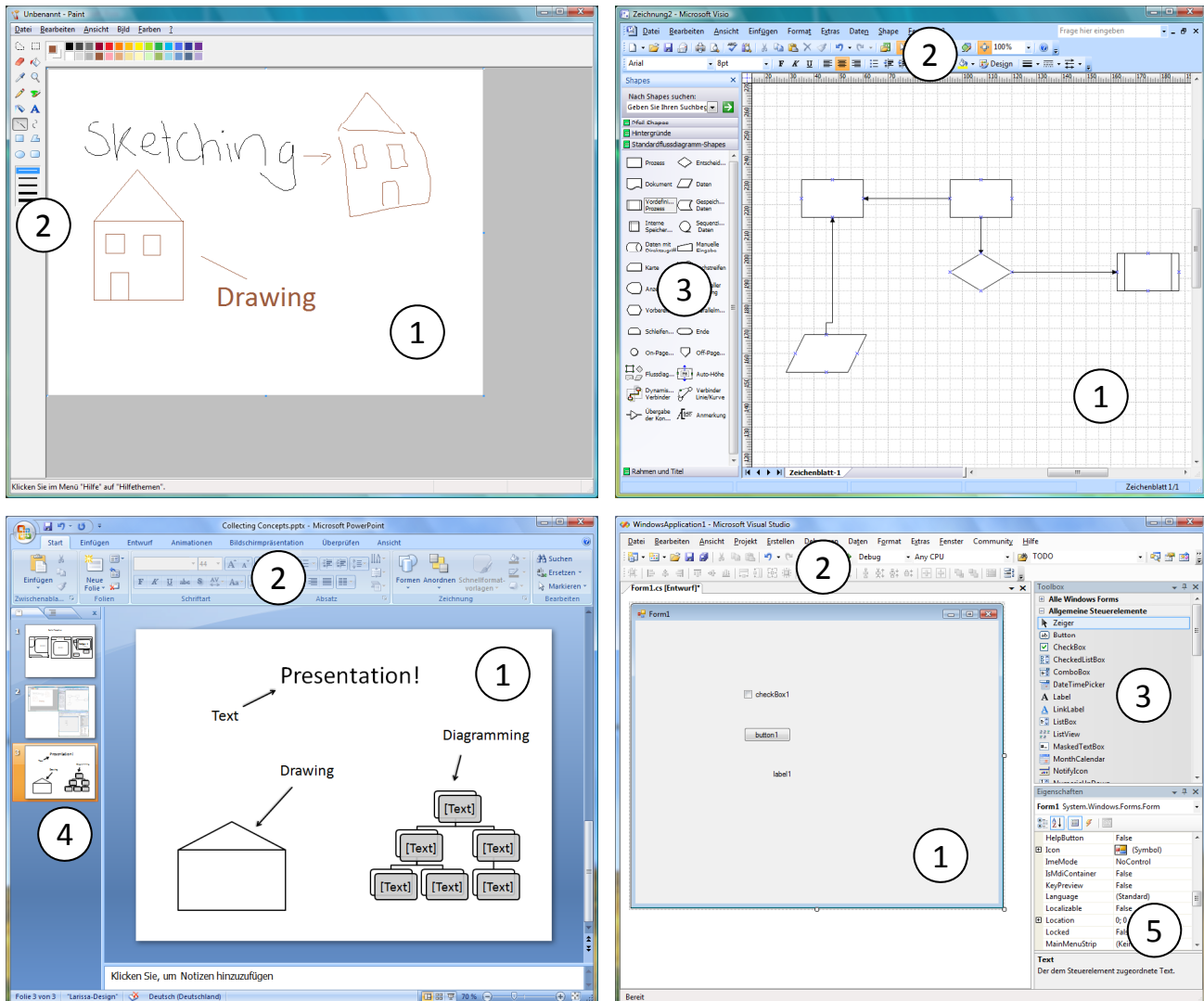
- **Collaboration**

Extend the whiteboard to facilitate asynchronous and synchronous work. Consider container and inhabitation features to support specification, communication and design rationale.

### 5.2.1 Whiteboard Interaction

Figure 5.6 shows four widespread software implementations of the whiteboard metaphor, which are frequently employed in user interface design practice. Actors often find themselves moving from one tool to another as they move forward through the process. Each tool

Widespread  
whiteboard  
interfaces



**Figure 5.6:** The Whiteboard Metaphor in interfaces: MS Paint (top-left), MS Visio (top-right), MS PowerPoint (bottom-left), MS Visual Studio (bottom-right)

is used for different purposes within different process steps, but they all have the whiteboard metaphor in common. Typical whiteboard interfaces are characterized by a whiteboard presentation, or canvas (see Figure 5.6 (1)), tools (see Figure 5.6 (2)), palettes (see Figure 5.6 (3)) and detail & context interface components (see Figure 5.6 (4)(5)).

#### Drawing software

Drawing software, like MS Paint (see Figure 5.6 top-left) might be used for simple sketching and drawings during early design steps. It resembles the closest mapping of the real whiteboard to a graphical interface. Some available tools to manipulate the canvas (see Figure 5.6 (2)), like pencils, rubbers and scissors are close to real tools, while other tools are blends of the metaphor to the computing domain, like text typing, shapes and pipette.

#### Diagramming software

Diagramming software, like MS Visio (see Figure 5.6 top-right) adapts the whiteboard metaphor by offering more blends to enhance diagramming purposes. It provides a canvas with grid-overlay, a set of domain-specific shapes in palettes (see Figure 5.6 (3)) and connection tools (see Figure 5.6 (2)) to ease the creation and manipulation of diagrams, which are frequently employed in modeling phases.

#### Presentation software

Presentation software, like MS PowerPoint (see Figure 5.6 bottom-left) combines diagramming, drawing and presentation functionality in a whiteboard-based interface. Available tools (see Figure 5.6 (2)) are extensive and make heavy use of metaphor blends. As presenta-

tion software is based on presenting information in a timely manner, it is not limited to one whiteboard, but provides a context control (see Figure 5.6 (4)) to enable the users to create and manipulate multiple whiteboards by still maintaining context. Presentation software might be used during design processes to present designs or models for discussion, but is also frequently employed for prototyping purposes, as it is capable of visualizing dialog flows.

Finally, interface development software, like MS Visual Studio (see Figure 5.6 bottom-right), provides an abstraction of a whiteboard to facilitate the design and manipulation of graphical user interfaces. Its components are blends of the previously described concepts of tools (see Figure 5.6 (2)), palettes (see Figure 5.6 (3)) and detail & context interfaces (see Figure 5.6 (5)) to the domain of graphical user interfaces. Its detail interfaces enable users to manipulate detailed properties of visual objects by still maintaining context on the primary task. Similar software is employed towards the end of user interface design processes or during implementation, but less frequently for prototyping even though it provides suitable functionality.

User Interface  
Design software

Actors utilize the presented tools within the design process, even if they are originally dedicated for other purposes. We think that a combination of different aspects of whiteboard interfaces and adequate metaphor blends will effectively support user interface design. We aim to create a whiteboard interaction design that is especially designed for user interface tasks, which were presented in Chapter 4—“Analysis”. By combining the required features in one single interface, we believe that work transitions can be bridged more easily. Improved performance in bridging transitions between tool-usage will enhance overall process efficiency and traceability. Additionally, as the whiteboard metaphor is already well established and familiar to all actors within the target domain, ease of learning is at its best. Therefore, we will investigate typical interaction patterns and interface components.

Bridge work  
transitions in tool  
usage by blends

## Interface Components

Figure 5.6 reveals the basic interface components that build up a whiteboard interface: The whiteboard or canvas (1), tools (toolbars or menus) (2), palettes (3) and context (4) & detail (5) components. All these interface components fall into the category of “direct manipulation” interfaces.

Direct manipulation  
interface  
components

“Direct manipulation” is a term that was coined by Ben Shneiderman in 1974 to describe visual interface designs that consist of three essential principles:

“1. Continuous representations of the objects and actions of interest with meaningful visual metaphors. 2. Physical actions or presses of labeled buttons, instead of complex syntax. 3. Rapid, incremental, reversible actions whose effects on the objects of interest are visible immediately.”

(Shneiderman and Plaisant, 2004, p. 234)

According to Shneiderman and Plaisant (2004, p. 234f), it is possible to design interfaces that have a wide range of benefits by applying these basic principles. New users will quickly learn about the basic functionality by demonstration or exploration. Error messages are rarely and users will immediately see if their actions are productive or counterproductive and can adjust their actions accordingly. User experience is at its best, because direct visual feedback encourages users and imposes a feeling of confidence. When users experience that they are in control of the interface they can predict the interface’s response. One of the most important benefits of direct manipulation in context of the work at hand is that it facilitates exploration. As actions on objects are straightforward and predictable, they promote a playful exploration of possibilities. Playing with these possibilities supports creative thinking and innovations. However, direct manipulation interfaces are not applicable to all domains and may bring some disadvantages, like increased system resources, necessity of advanced

Direct manipulation  
paradigm



programming skills for visual techniques and cumbersome options for automation of tasks Shneiderman and Plaisant (2004, p. 259).

Employed direct manipulation techniques

In the following, previously identified elements of the whiteboard metaphor (see Figure 5.6) are described by referencing the direct manipulation features they employ. According to Cooper et al. (2007, p. 377), most direct manipulation actions fall into one of seven categories, which can also be found throughout the different elements of whiteboard interfaces: Pointing, Selection, Drag & Drop, Control manipulation, Palette tools, Object manipulation and Object connection.

Manipulating whiteboard content

The dominating element of whiteboard interfaces is the whiteboard or canvas itself (see Figure 5.6 (1)). It is the central presentational component of the interface and its primary responsibilities are displaying content and objects as well as relations between these objects. As screen space is limited, the whiteboard representation is usually scalable. Zooming functionality enables the user to focus on parts of the whiteboard in order to manipulate details, which are not displayed adequately in full scale. Whiteboards are frequently extended by layout grids that provide aids for object positioning and manipulation. Interaction with presented objects is primarily achieved by employing the direct manipulation features pointing, selecting and drag&drop. These mouse-based interactions are supported by tools (toolbars and menus) and palettes. In the following, characteristics of these interactions will be further described in detail.



### Interaction Patterns

Patterns in whiteboard interaction

Whiteboard interaction is primarily implemented by pointing, clicking and dragging with the mouse. Cooper et al. (2007, p. 382f) gives a detailed description of whiteboard interaction styles and widespread patterns. Following these patterns in physical design is highly recommended, as they can be found in various whiteboard-like software applications. Ease of use and ease of learning requires paying attention to widespread interaction patterns, as users have experiences in use of similar interfaces and consequently expectations on similar-looking interfaces. In the following, interaction patterns for whiteboard interfaces are presented in reference to Cooper et al. (2007, p. 382f).

- **Pointing and clicking with the mouse**

As the mouse cursor is moved over objects of interest on the whiteboard, a change of cursor shape indicates possible operations on this object. While this feature is well-known as "cursor-hinting", Cooper et al. (2007, p. 383) calls this property "pliancy". A single mouse-click usually selects data or an object or changes the state of controls. Double-clicking objects or controls means single-click plus action. Clicking & dragging the mouse is a versatile operation, which is employed for several purposes, which will be described later. Other frequently employed mouse interactions are chord-clicking, which means to trigger functionality by using a combination of two mouse buttons. As this technique is not widespread, it is considered an expert-only idiom.



- **Selection and command ordering**

The act of choosing an object or control is referred to as selection. This simple interaction pattern is typically accomplished by pointing and clicking. Selection of multiple objects is typically implemented by dragging so-called "selection marquees" or "drag-rectangles" with the mouse or by keyboard- and button-actuated means of doing this. The subset of objects are then visually marked with specific colors. Cooper et al. (2007, p. 392) distinguishes between mutual exclusion, additive and group selection. Mutual selection is usually the standard-behavior when selecting objects based on single mouse-clicks. Previously marked objects are then unmarked as a new object is clicked. Consequently, one object can be selected at once. Additive selection is usually triggered by keyboard commands and clicking of objects at the same time. Group selections are based on the click-and-drag mouse operation. They are primarily used to extend existing subsets. Command ordering functionality is usually implemented



by selections and triggering of tools on these selections. Functionality is then usually invoked by using menus, toolbars or context-menus. Typical actions are insert, replace, copy and cut.

- **Drag & Drop**

Dragging and dropping objects is usually achieved by clicking and holding the mouse button, while moving the mouse across the screen and then releasing it in a meaningful location. In its most common form, it is used to reposition objects, select objects and reshape objects. In combination with tools or palettes, this operation is employed to draw on the whiteboard or to drag and drop objects from palettes. Its interaction can be described as "clicking on some object and moving it to imply a transformation" (Cooper et al., 2007, p. 397). An important aspect of drag&drop implementations is visual feedback. Drop candidates must visually indicate their receptivity while the drag cursor must visually identify the source object. Drag and drop from palettes is frequently employed in diagramming-style whiteboard interfaces to add objects.



- **Object Manipulation**

Objects on the whiteboard are usually manipulated by clicking and dragging. According to Cooper et al. (2007, p. 411f), object manipulation can be divided into three categories: repositioning, resizing and reshaping. Repositioning is the simple act of moving an object by dragging it to a new location. Resizing and reshaping is usually implemented by so-called "resize handles" or simply "handles", shown in Figure 5.7. A handle consists of little squares or circles, positioned at the boundaries of an object. Clicking and dragging those leads to direct manipulation of the boundaries. The handles indicate selection and offer resizing and reshaping functionality at the same time.

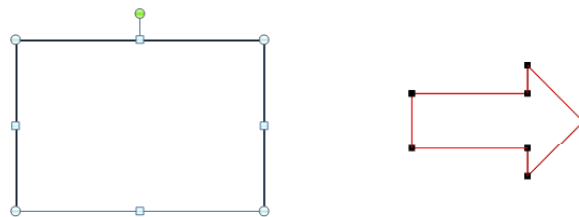


Figure 5.7: Direct graphical object manipulation with handles in MS PowerPoint

### Toolbars, Menus and Detail & Context

Toolbars or menus (see Figure 5.6 (2)) offer additional manipulating functionality that exceeds possibilities available through direct mouse interaction. Special variations of menus are context menus. They provide context-specific functions which can usually be triggered with the right mouse button directly on the concerning objects. While dropdown menus can be found in nearly any graphical user interface, toolbars are a more recent addition to make access to functions more visible to the user. According to Cooper et al. (2007, p. 493f), toolbars are an efficient mechanism for providing persistent, direct access to commonly used functions and therefore are a good extension to menus that provide complete toolsets. As toolbars are permanently visible and can be triggered with a single mouse click, they are especially helping new users. According to Shneiderman and Plaisant (2004, p. 236), special attention has to be spent on iconic presentation when designing toolbars, as they have to make sure that the underlying functionality is recognized by the user. A good way to achieve consistency of functional mapping therefore is to employ familiar icons that are also used in other popular interfaces.

Toolbars and menus as supporting tools





Palette tools for content creation and manipulation



Palettes, also known as toolboxes (see Figure 5.6 (3)) are a popular interface concept to make predefined objects available to the user. According to Cooper et al. (2007, p. 409f), there are two basic variations of palettes: Modal palettes and charged cursor tools. Modal tools will change the cursor's appearance to indicate an active tool. These mechanisms are primarily employed in drawing tools, where cursor movement will produce lines on the whiteboard. Although they are an effective and natural tool, they will only work for actions on object that can be drawn, like lines, erasers or shapes. Charged cursor tools, in contrast load specific predefined objects on the cursor. Instances of these objects are then created on the whiteboard, where a user clicks or drops them.

Detail & Context components



Detail & Context interface components are employed to facilitate views on detailed properties or overviews on multiple whiteboards. Navigation between different whiteboards is often realized with thumbnails (see Figure 5.6 (4)) that represent a small preview of whiteboard contents. Other contextual components may utilize trees or lists to visualize structures within whiteboard objects or between several whiteboards. Manipulation of certain details of whiteboard objects is often provided by modeless dialogs, like property panes that are attached to one side of the whiteboard or float on the whiteboard. These dialogs allow displaying contextual information without the constraining consequence to interrupt the application like modal dialogs. Cooper et al. (2007, p. 510) recommends to employ modeless dialogs to replace otherwise necessary modal dialogs as they do not hinder application flow. Nevertheless, modeless dialog design are challenging to design and implementation, as they require dynamic manipulation of data without freezing the application and advanced concerns in input termination.

## 5.2.2 Spatial Navigation

Navigation between artifacts within the design room



A single whiteboard will eventually be too small to provide adequate screen space for the extensive numbers of artifacts created throughout the user interface design process. As the amount of available screen space, or "screen real estate" is limited on computer screens, it has to be extended with interaction concepts that facilitate spatial navigation. Nevertheless, relations between artifacts and whiteboard contents have to remain visible to support the essential requirements of explorability and traceability. As the following analysis will show, commonly used detail & context concepts are not adequate for our demands. We therefore investigate alternative means of providing spatial navigation that fits our demands in whiteboard- and design-room navigation.

### Focus & Context Interfaces

General options for spatial navigation interfaces

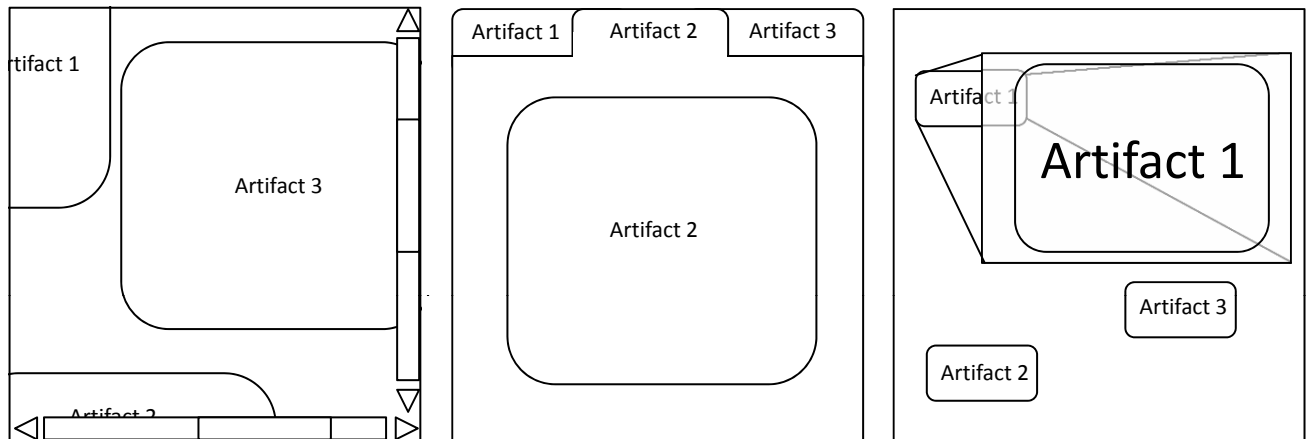
As of examination of Mynatt (1999, p. 7), Cockburn et al. (2006), Baudisch et al. (2002) and Pook et al. (2000), there are three general options for managing virtual two-dimensional whiteboard space: Scrollable space, segmented space and zoomable space. Three-dimensional virtual spaces are disregarded at this place, as they tend to complicate interaction with two-dimensional artifacts (Heim, 2007, p. 73). Figure 5.8 displays three simplified representatziions of these interface concepts in order of their employment in practice.

Scrollable display space

According to Mynatt (1999, p. 7), users show a slight preference for a scrollable space, as this is the most common form of virtual space and is well distributed among commonly used desktop applications (see Figure 5.8 left). Nevertheless, users are concerned about losing track of artifacts 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 artifacts are spread along a wide range of screen space.

Detail & context display space

As a solution to these issues, some whiteboard implementations employ detail & context mechanisms to gain whitespace while maintaining the visibility of other artifacts on the whiteboard. A popular approach therefore is to separate artifacts on several whiteboards



**Figure 5.8:** Options for spatial navigation between artifacts on the whiteboard (scrolling, switching, zooming)

and to offer visual references or thumbnails to provide access to them via “switching” operations. Examples for such implementations are “tabbed windows” (see Figure 5.8 middle) and thumbnail previews for navigation (e.g. MS PowerPoint). As separation of whiteboard contents hides relations between artifacts that are spread over multiple whiteboards, overview, consistency and traceability are seriously harmed. Thumbnail navigations or tabbed windows are only capable of visualizing a sequence of whiteboards. As an essential demand to visualization in this work is to reveal relations between artifacts within the design process, interface concepts based on separation are not suitable.

Finally, zoomable space (see Figure 5.8 right) provides detail and context by opening up a third dimension: time. Users are able to display detailed views on artifacts by triggering one or more zooming operations. While contextual information can be provided by visualizing artifacts in smaller scale, manipulation of objects is realized by zooming into these small representations. Zooming in and out in various degrees of scale facilitates advanced visualization of detail and context. As zooming is continuously, a wide range of abstractions can be displayed. A very simple example of a zooming interface can be found in MS Word. It allows zooming out and looking at all pages of a document tiled across the primary window. At this level, the text is not readable, but layout configuration, pictures and other page elements are still easily identified. Zooming into a desired page then reveals the content and possibilities for manipulation. This simple example explains the basic concept, but is a very restricted version of a zooming interface, as it neglects multiple scales and spatial location of its documents. True zooming interfaces actually represent a new interface paradigm, by harnessing the potential of spatial location, multiple scales and navigation aids to support users in exploring the virtual space. In the following, we therefore investigate zooming interface concepts as supporting supplement for extending screen space of the whiteboard.

Zooming display space



### Zoomable User Interfaces

Zoomable User Interfaces (ZUIs) are primarily considered as UI interaction techniques, but are also regarded as an emerging interface paradigm as they change user interaction dramatically and impose new concepts for allowing interfaces to harness the human perception capabilities. According to Raskin (2000, p. 152ff), regular graphical user interfaces based on the WIMP (Windows, Icons, Menus and Pointing device) paradigm equate to navigating a maze, while zoomable user interfaces are analogous to flying. His concept of a zooming interface paradigm (ZIP) aims on utilizing the human capability of remembering landmarks and the sense of relative positioning for interface interaction. A zooming interface that is consistent with Raskin’s paradigm should employ organizational clues, such as proportion, color, patterns, proximity and other visual stimuli instead of buttons, toolbars or menus. His

ZUIs are both techniques and a novel interface paradigm

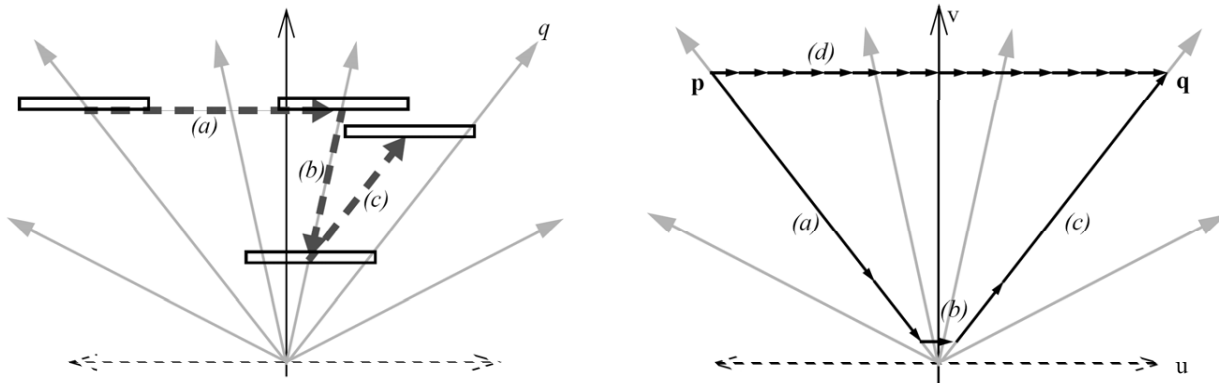


Figure 5.9: Panning & Zooming illustrated in space-scale diagrams (Furnas and Bederson, 1995)

experimental implementation ZoomWorld<sup>5</sup> demonstrates this concept in context of document exploration. In comparison to the previously described zooming in MS Word, document content is scaled in multiple dimensions and is organized in a structured hierarchical manner based on spatial positioning. ZoomWorld is not the only paradigm for zoomable user interfaces (König, 2006), but provides extensive documentation<sup>6</sup> and is highly anticipated in the research community. The following analysis is focused on ZUI techniques that are practically applicable instead on novel interface paradigms that aim on revolutionizing the current interface paradigms.

#### Fundamentals of ZUIs

The zoomable user interface as an interaction concept was already introduced by Sutherland (1964), adopted by Perlin and Fox (1993) and advanced by Bederson and Hollan (1994) as a new way of interacting with graphical interfaces. It presents the user a canvas that is larger than the viewing area of the computer screen and objects that are placed on the canvas determined by the user, the system or both. In contrast to scrollable virtual spaces, ZUIs employ zooming tools that are much like a telescopic lens or a zooming camera. According to Bederson and Hollan (1995), this zoomable canvas has a virtually unlimited resolution. Based on system limitations, a user may zoom in and out to an infinite degree. This characteristic makes ZUIs similar to 3D interfaces with three dimensions of movement: x, y and z-axis. Instead of navigating freely in all directions, ZUIs allow to move forward only in one direction along the z-axis. Bederson and Boltman (1999) provide a general definition of ZUIs:

“Zoomable User Interfaces (ZUIs) are a visualization technique that provides access to spatially organized information. A ZUI lets users zoom in and out, or pan around to view much more information than can normally fit on a single screen” (Bederson and Boltman, 1999, p. 2)

#### Key features of ZUIs

In the following, some of the essential features of ZUIs, namely “Panning & Zooming”, “Semantic Zooming” and “Animations” are briefly described by referencing Furnas and Bederson (1995).

#### Panning & Zooming

Navigation within ZUIs is mainly facilitated by “Panning & Zooming”. Figure 5.9 schematically visualize this technique in space-scale diagrams by projecting the zoomable space onto two axes (space), and looking onto the z-axis (scale) from the top.

“Panning” therefore is regarded as moving the view across the space axis, while “zooming” moves the view along trajectories of the scale axis. Figure 5.9 (left), (a) shows a pure pan, (b) a zoom-out operation and (c) a zoom-around (the point q) operation. The navigation



<sup>5</sup><http://rchi.raskincenter.org/>

<sup>6</sup><http://rchi.raskincenter.org/index.php?title=ZUI.Specification>



strategy of zooming out, panning and then zooming in is one of the most important benefits in ZUI navigation:

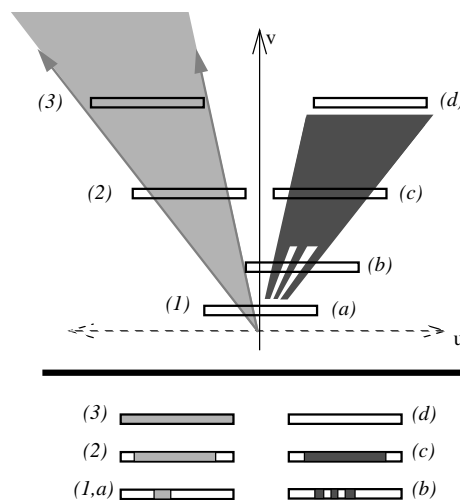
“Paradoxically, in scale-space the shortest path between two points is usually not a straight line. This is in fact one of the great advantages of zoomable interfaces for navigation and results from the fact that zoom provides a kind of exponential accelerator for moving around a very large space.” (Furnas and Bederson, 1995, p. 4)

Figure 5.9 (right) visualizes this concept. Great traversal distances in ZUIs (d) may be reached efficiently by first zooming out (a), making a small pan (b) and then zooming in by focusing a point of interest (c), which is then closer to the original position as before. Therefore, ZUIs make navigation in three dimensions logarithmic in contrast to direct traversions (d), which is the common navigation style in scrolling virtual spaces (Furnas and Bederson, 1995, p. 5).

Efficient navigation  
in large spaces

“Semantic Zooming” describes an improvement to regular geometric zooming. According to Furnas and Bederson (1995, p. 6), geometric zooming only changes the size of objects within the ZUI, whereas semantic zooming allows objects to change their appearance based on the scale they have in the current view.

“For example, an object could just appear as a point when small. As it grows, it could then in turn appear as a solid rectangle, then a labeled rectangle, then a page of text, etc.” (Furnas and Bederson, 1995, p. 6)



**Figure 5.10:** Semantic zooming illustrated in scale-space (Furnas and Bederson, 1995, p. 6)

Figure 5.10 illustrates this concept with different viewports within a scale-space diagram. The object on the left corresponds to a simple line that just appears larger as the user zooms in (1), (2), (3). The line on the right changes its appearance as the user zooms in. In position (a) it is not visible at all, in (b) it appears as a dotted line segment, while in (c) it is displayed as a solid line. In position (d) it disappears again, as its appearance would be larger than the actual viewport. Semantic zooming therefore enables to display only the relevant information of objects when they are looked at from overview distance, while all details are revealed once the user focuses (zooms in) on this object. Consequently, it is a powerful tool for information visualization.

Semantic zooming



Animation improves subjective user satisfaction



Bederson and Boltman (1999) evaluated zoomable user interfaces and found that animation of zooming operations is beneficial to help users building up a mental image of the zoomable information space. According to them, animation helps users to maintain object constancy and therefore facilitates better understanding of the relations between two states of a system. Bederson and Boltman (1999) also revealed that animated navigation could improve subjective user satisfaction. Nevertheless, smooth animation takes time, which brings up a fundamental tradeoff between the time spent for animating and the time spent actually using the interface. Bederson and Boltman (1999) recommend an animation timespan of 0.5 seconds to 1 second as adequate balance for zooming operations.

Benefits and drawbacks

Zoomable User Interfaces are a highly anticipated research subject, as they bear potential for improvement in managing large spaces of information. Efficient means of navigation with zooming & panning, semantic zooming and smooth animations are key benefits of ZUIs in comparison to spatial navigation in traditional WIMP interfaces. Relations between objects and system states are easier to perceive as interface components are not separated. Nevertheless, ZUIs also brings a significant drawback, namely "desert fog" (Jul and Furnas, 1998). This term describes a state where the user has navigated in the ZUI to a place, where no landmarks are visible. According to Jul and Furnas (1998), this could appear in one of three ways: The user may zoom in to a place where objects are not visible, e.g. based on semantic zoom, the user may zoom out to a point where the scale of object is too small to display, or users may zoom into areas that do not contain any information. In such situations, the user is left without any cues on how to get back to viewing information without wandering around randomly. This situation may be even worse than in traditional scrolling virtual spaces. However, research came up with several solutions to some problems that focus primarily on navigational aids and advanced zooming techniques. Consequently, contributions made by researchers to improve ZUI performances have to be respected when developing zoomable interfaces.

Navigational aids to avoid desert fog



Furnas (1997) extended the view-navigation model to include visible structures on the zoomable space. Such a structure should provide traversal requirements (a small number of paths between views) and navigation requirements (cues of navigation options within each view). Similar concepts were proposed by Good et al. (2004) who also stated that ZUIs are an efficient technique to visualize abstractions and hierarchies based on spatial organization and scale on various levels of navigation. Good et al. (2004) argues to employ structures hierarchies in ZUIs to unleash the real potential of semantic zooming. Nevertheless, Furnas (1997) found that ZUIs may still lead to views in desert fog. To resolve this issue, they proposed to limit zooming operations if the zoom scale is beyond visible objects. Plaisant et al. (1996) came up with another solution that allows users temporarily return to an overview position, which displays their zoomed-in position with adequate cues. This tradeoff may be seen as a compromising step away to the conventional graphical user interfaces. Pook et al. (2000) and Pook (2001), in contrast propose to employ in-place contextual aids, which he calls context layer, that transparently overlaps the user's view along with a hierarchical tree in a separate display which makes overall structure and position within the ZUI visible and accessible. Similar concepts were developed with separate or superimposed overview windows. Pook (2001) also proposed a history mechanism, which is similar to history functionality in web browsers. Hornbaek et al. (2002) compared performance of ZUIs with and without overview mechanisms and found that switching between different views of detail and overview costs time and decreases efficiency, but enhance the user's ability to recall their position in the virtual space. Other experimental navigation aids that are not considered as they are focused on very large information spaces are variations of fisheye-views (Furnas, 1986, 2006), considered and improved by Bartram et al. (1995) and Schaffer et al. (1996).

Improved zooming mechanisms simplify interaction

Advanced zooming techniques aim on improving the original panning & zooming interaction mechanism. "Goal-directed zoom" proposed by Woodruff et al. (1998), supports zooming operations with menus, previews and "automatic zoom". Menus therefore present the user with a selection of possible zooming actions based on their current focus. Previews graphically predict the outcome of a zooming operation before actually committing to it. "Automatic zoom" zooms to an adequate representation of an object that is appropriate to manipulate it. "Automatic zoom" operations are then triggered through selection. Similar

zooming techniques were proposed by Furnas and Zhang (1998) and Ware (2004, p. 377ff). Usually, selected objects are then maximized and centered in the middle of the zooming interface. Igarashi and Hinckley (2000) provide additional detailed techniques to improve automatic zooming with "combined zooming & panning". In contrast to the above described zooming techniques, which are all based on smooth animation, Bederson and Hollan (1994) and Pook et al. (2000) propose to employ jump zooming for some applications. Jump zooming skips animation between zooming operations that are time intensive. Therefore, they aim on making interaction more efficient, e.g. when transitions between spatial locations are very large.



### Evaluation for Tool Design

As of the described examination of ZUI features and constrains, we think that a zoomable interaction approach can effectively address issues in spatial navigation present in the research at hand. ZUIs organize information in space and scale, which allows efficient artifact management, even when graphical artifacts found in design processes are extensive. Additionally, they fit our metaphor as overall ZUI interaction is similar to navigating in a room or in front of a whiteboard. They also provide beneficial means of imposing an explorative structure on the interface, which may help to visualize and guide through the process of user interface design. As ZUIs are suited for hierarchical representations they support visualization of multiple degrees of abstraction that accompany the design process. Nevertheless, semantic zooming techniques additionally improve the presentation of abstractions within the design process and may help to bridge work transitions. Animated zooming can be utilized to narrow the work transitions frequently found between artifacts produced in many process phases of interface design. Goal-directed zooming techniques are helpful to support the iterative nature of the design process as they efficiently allow to navigate between widespread parts of an interface. ZUIs enable to perceive spatial relations between artifacts and harness the human perception capabilities in landmark navigation, which is beneficial for creative thinking and free association. Navigational aids improve overview and detail which are unavoidable to facilitate awareness of the "big picture" as well as artifact details throughout the process. Eventually, advanced zooming techniques can be used to simplify interaction and reduce complexity of user interaction while animation may enhance subjective user satisfaction, which contributes to the desirability of our design.

ZUI techniques fit demands for spatial navigation

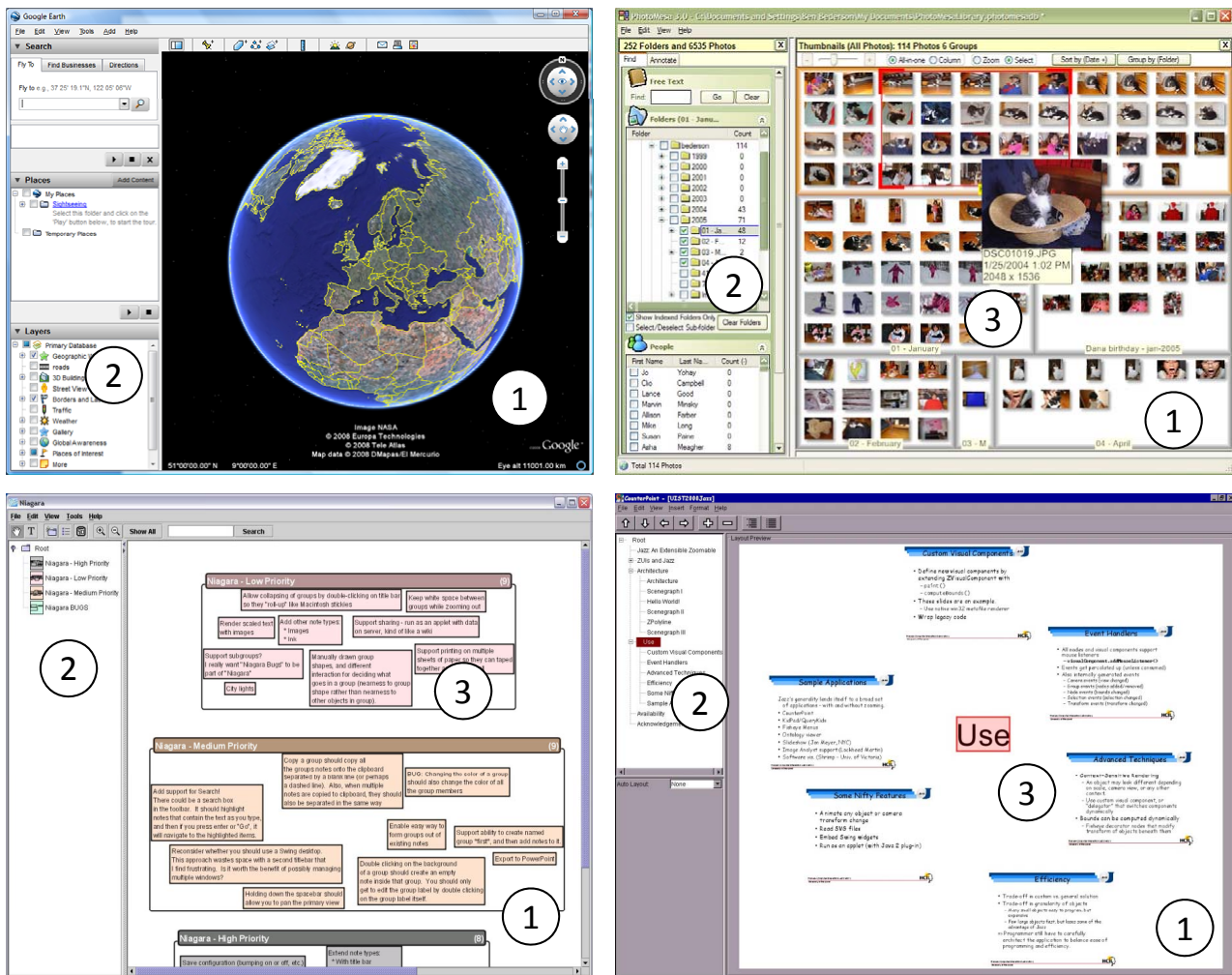
Nevertheless, we also think that employing zooming techniques may bring some disadvantages and issues that have to be dealt with. ZUIs are regarded as a new interface paradigm as they dramatically change interaction. Consequently, users might have issues with adapting to this novel interaction style. Desert fog issues may lead to frustrating user experiences and harm efficient use. As there are no established interaction patterns in ZUIs, ease of learning and ease of use is significantly harmed. ZUIs make the overall interface more complex and are currently regarded as expert-systems, which imposes additional demands to users. Imposing a fixed structure on the zoomable information space to enhance navigation, may restrain creative thinking (Guidance vs. Creativity). Eventually, ZUI implementations and evaluations primarily exist within the domains of document or image management and geographical information systems (GIS). They are not proven to work for interfaces based on whiteboard or room metaphors.

Identified drawbacks

Consequently, based on the identified drawbacks, we propose guidelines to make employment of ZUI techniques successful concerning the research at hand:

Implications for design

- Avoid a dramatical change of the interaction paradigm as ease of use and ease of learning are top priorities; consider a blend of WIMP and ZUI techniques for the sake of adaptability.
- Utilize adequate zooming techniques and navigation aids to improve spatial navigation on the whiteboard, but avoid too complex interaction which are hard to understand for all actors.



**Figure 5.11:** Popular ZUI implementations, Google Earth (top-left), PhotoMesa (top-right), Niagara (bottom-left), CounterPoint (bottom-right)

- Efficiently combine metaphor blends with the zooming interface paradigm.
- Utilize appropriate visualization of the design process with supporting zoom techniques.
- Make visualization flexible enough to facilitate creative thinking, but not too loose to constrain guidance on the process.
- Employ familiar interaction patterns based on widespread zoomable user interface implementations to improve adoption among actors.

### Examples & Interaction Patterns

Investigating interaction patterns by examples

As ZUIs are not widespread in practice, it is hard to find consistent interaction patterns that are familiar to all actors. Nevertheless, ZoomWorld provides a specification of its interaction that is regarded as a consensus of ZUI developers. Patterns can also be identified by looking at a range of successful ZUI implementations.

<p>Actual successful ZUI implementations (see Figure 5.11) are mostly experimental research projects that can be found in the domains of geographical information systems (GIS), image viewing and management, note taking, reading, presentation and visualization or graph tools. As illustrated in Figure 5.11, most successful ZUI implementations are blends of ZUI techniques with the common WIMP paradigm. Usually, these interfaces consist of a zoomable canvas (1), navigation aids (2) and structured visualization of contents (3). In the following, some most acknowledged implementations are presented that are related to the research at hand.</p>	<p>Example implementations are blends of WIMP and ZUI techniques</p>
<p>Google Earth<sup>7</sup> (see Figure 5.11 top-left), Microsoft Virtual Earth<sup>8</sup> as well as NASA World Wind<sup>9</sup> utilize zooming techniques as navigational tool for their map displays. Employed techniques are overview displays and various navigation aids as well as variations of goal-directed zooming and semantic zooming. Content editing remains elusive as most content is supplied by the system as maps or satellite images. Map-related systems are arguably the most proven and widespread zooming interface implementations and interaction patterns are well distributed.</p>	<p>GIS systems</p>
<p>PhotoMesa<sup>10</sup> (see Figure 5.11 top-right) (Bederson, 2001), PhotoFinder<sup>11</sup> (Kang et al., 2000) and SAPHARI<sup>12</sup> (Semi-Automatic Photo Annotation and Recognition Interface) (Kang et al., 2007) utilize zooming interaction to visualize and interact with pictures in a structured manner. They specialize on enhancing image collection browsing based on smooth zooming and hierarchical structures. PhotoMesa utilizes zoomable quantum treemaps (Shneiderman, 1998; Bederson et al., 2003; Blanch and Lecolinet, 2007) to display the content of photo collections based on spatial location and scale. Content creation is constrained to attaching annotations on pictures, as the visualization is created from the existing file structure.</p>	<p>Image management tools</p>
<p>Niagara<sup>13</sup> (see Figure 5.11 bottom-left) (Good, 2002, 2003) is a zoomable organization tool for thoughts, documents and presentations. It offers content creation functionality, like typing text and moving container objects as well as structuring information. It features a whiteboard-like interface, grouping and containers. Its semantic zooming techniques introduced automatic text reduction (Good et al., 2002b). Similarly, NoteLens<sup>14</sup> (Good, 2002) is a note taking software that employs detail and context interface components as well as rapid visual feedback.</p>	<p>Note taking tools</p>
<p>CounterPoint<sup>15</sup> (see Figure 5.11 bottom-right) (Good et al., 2002a; Good, 2003) is a zoomable presentation tool for organizing presentations slides based on spatial relationships. Authoring functionality is constrained to aligning and structuring presentations. It does not include actual editing of slides as they are imported from MS PowerPoint. Similarly, the Fly Presentation Tool (Holman et al., 2006) utilizes a mind-map like spatial structure on a zoomable canvas to structure slide show presentations. Content creation is again not facilitated as it currently utilizes existing pictures as slides. KidPad<sup>16</sup> Druin et al. (1997) utilizes sketching and linking of sketches for drawing storyboards on a zoomable canvas. Content creation is limited to sketches of different colors.</p>	<p>Presentation &amp; Sketching</p>

<sup>7</sup><http://earth.google.com/>

<sup>8</sup><http://www.microsoft.com/virtualearth/>

<sup>9</sup><http://worldwind.arc.nasa.gov/>

<sup>10</sup><http://www.cs.umd.edu/hcil/photomesa/>

<sup>11</sup><http://www.cs.umd.edu/hcil/photolib/>

<sup>12</sup><http://www.cs.umd.edu/hcil/saphari/>

<sup>13</sup><http://goodle.org/niagara.html>

<sup>14</sup><http://www.windsorinterfases.com/notelens.shtml>

<sup>15</sup><http://www.cs.umd.edu/hcil/counterpoint/>

<sup>16</sup><http://www.cs.umd.edu/hcil/kiddesign/kidpad.shtml>



Common patterns  
in zoom interaction

Based on the provided documentation and examples of ZUI implementations some ZUI interaction patterns for spatial navigation can be identified, which present a consensus of ZUI developers that may help to ease adoption in practice. The following list presents most common interaction patterns in ZUI implementations:

- **Selection and command ordering**

A single mouse-click usually selects an object on the zoomable canvas. Double-clicking objects means single-click plus navigation. Triggered navigation actions may include goal-directed zooming operations, like automatic zooming. Command ordering is diverse and cannot be unified to a simple pattern. Most implementations utilize contextual menus, shortcuts, experimental pie-menus, or gesture recognition.



- **Panning**

Panning interactions are also quite diverse. ZoomWorld recommends pressing down the mousewheel and dragging to pan the window. The inverse direction of mouse movement then corresponds to the panning direction. This interaction pattern is taken from the "hand cursor" which is common in whiteboard interfaces and feels like grabbing & dragging the view. Releasing mousewheel stops the panning immediately.



- **Manual Zooming**

Most manual zooming operations are initiated by scrolling the mouse wheel. This pattern, also called real-time zoom, translates the view based on the amount of scroll-ticks. It supports both directions, whereas negative scrolls zoom out and positive scrolls zoom in. Zooming in is translated around the mouse pointer which is required to navigate efficiently. The viewport is then automatically panned to align the center of the view around the pointer. Zooming out does not follow the pointer, but decreases zoomfactor constantly, which feels like moving away in a straight line. Field-of-view boxes, that are frequently employed in regular whiteboard zooming are not very popular in ZUIs and are mostly found in regular virtual scrolling spaces.



- **Goal-directed zooming**

Goal-directed zooming operations are primarily triggered by navigation aids, like overviews, structure views or other menu controls or even hyperlinks (hyperzoom). These commands are usually triggered by double-clicking, which is considered as committing to an action in respect to navigation.



None of the examples and available ZUI specifications, like that of ZoomWorld, presents interaction patterns that concern content creation. As this functionality is a crucial demand for whiteboard interfaces, a mechanisms has to be identified which bridges ZUI techniques and whiteboard interaction patterns that are familiar to all actors.

### 5.2.3 Interactive Visualization

*"Overview first, zoom and filter, then details on demand"*

—Ben Shneiderman

ZUIs facilitate  
means for  
advanced  
interactive  
visualization

As previously mentioned, Zoomable User Interfaces provide advanced possibilities for interactive visualizations. According to Benyon et al. (2005, p. 605) information design and visualization are one of the most important factors concerning efficiency, when designing an interface. Card et al. (1999) and Shneiderman and Plaisant (2004) argue for novel visualization techniques that harness the power of novel interaction techniques to effectively present and interact with large quantities of data. The main goals in designing effective visualizations are therefore to provide the user with a good overview in the first place, then allowing zooming in and filtering the data that is required to focus on the task (Shneiderman, 1996). Card et al. (1999) provides a number of examples that feature techniques like



"dynamic queries" Ahlberg and Shneiderman (1994), ConeTrees (Robertson et al., 1991) and TreeMaps (Shneiderman, 1998) among other novel visualization techniques presented and discussed in Sears and Jacko (2007, p. 509ff).

Shneiderman (1996, 1992, p. 570-603) present a task by data taxonomy which aims on providing a structured framework which supports choosing the right visualization for specific types of data. He distinguishes between seven types of data (one-, two-, three-dimensional data, temporal and multi-dimensional data as well as hierarchical and network data) and seven tasks (overview, zoom, filter, details-on-demand, relate, history and extracts). In reference to Shneiderman's taxonomy, we identify the type of data associated with interface design process artifacts as two-dimensional. According to the goal setting and requirements within this research (see Chapter 4.3—"Requirements for Tool Support"), required tasks are overview, zoom, details-on-demand, relate and history. Based on Shneiderman's task framework we consequently present main requirements for visualization:

A structured framework for visualization

- **Overview**

Gain an overview of the entire collection. Strategies include zoomed out views, additional detail-views, overview windows with movable field-of-view boxes and intermediate views.

- **Zoom**

Zoom in items of interest. Strategies include smooth zooming, zoom focus, zoom controls and pointing and zooming commands (goal-directed zoom).

- **Details-on-demand**

Select an item or group to get details when needed. The usual approach is to simply click on an item to get dialog windows with additional information.

- **Relate**

View relationships among items. Relations are usually displayed by connecting lines that are either persistent visible or displayed on selection.

- **History**

Keep a history of actions to support progressive refinement. Information exploration is a process with many steps. Allowing users to retrace their steps is important.

### Space-Scale Visualization

According to Sears and Jacko (2007, p. 509), the key decision in any visualization is to choose the right attributes to spatially organize the data. Once this has been decided, there are only a few visual distinctions that can be made to mark different types of data. Benyon et al. (2005, p. 608) describes the most effective options:

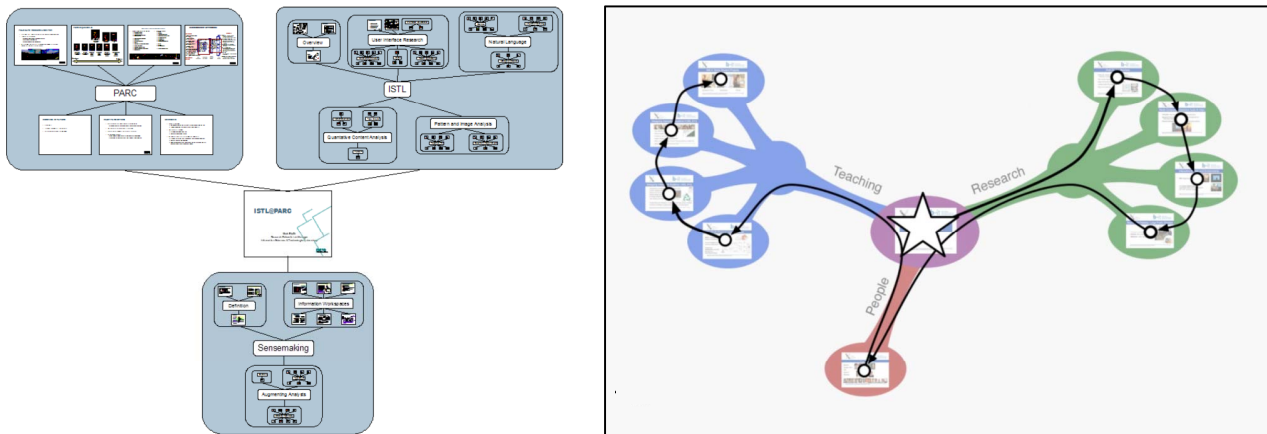
The key to visualization are data attributes

"The designer can use points, lines, areas or volumes to mark different types of data. Objects can be connected with lines or enclosed inside containers. Objects can be distinguished in terms of color, shape, texture, position, size and orientation."

(Benyon et al., 2005, p. 608)

Zoomable User Interfaces support all of these options of visual attributes to present data. Nevertheless, the most effective are spatial location and scale. Popular examples of such space-scale visualizations are TreeMaps (Shneiderman, 1998) and zoomable TreeMaps (Blanch and Lecolinet, 2007). Advancements of these visualization techniques are Quantum TreeMaps and Ordered TreeMaps (Bederson et al., 2003). These visualization techniques utilize spatial location and scale in order to present a very large number of data sets. Nested TreeMaps (Shneiderman, 1998) in contrast focus on the visualization of hierarchies and various degrees of abstraction. As these examples demonstrate that space-scale visualization

ZUI visualization techniques



**Figure 5.12:** Tree of nested containers in CounterPoint (left) (Good, 2003, p. 181) and spatial mindmap-like visualization in Fly (right) (Holman et al., 2006)

is powerful, they are not suitable for the artifact data that is fundamental to the research described in this work.

Space-scale visualization suitable for abstractions and relations

When looking at the ZUI implementations, which were previously presented (see Chapter 5.2.2—“Zoomable User Interfaces”), many interfaces take advantage of space and scale by either employing nested objects in tree-structures (scale, location & connections) or graphs (location & connections). Figure 5.12 (left) shows a clustered, tree-like visualization proposed by Good (2003, p. 181) that visualizes relations between objects based on hierarchies. According to Good (2003, p. 179ff), clustered objects provide overview and enable efficient exploration by zooming. Figure 5.12 (right) shows an alternative way to visualize relations between objects, namely connections that are designed, as they would appear in mindmaps. Good (2003, p. 179ff) argues that the inherent spatial structure of ZUIs helps to increase the withholding of presentation content by the audience. Similarly, Holman et al. (2006) believe that their representation incorporates Gestalt principles. Its structure represents informations associatively by utilizing color, weighted edges and spatial location.

Nested trees facilitate exploration by zooming



While the nested tree layout in CounterPoint is automatically generated from a given hierarchical structure and is therefore rigid and inflexible, the structure in Fly is dynamic and created manually by dropping items on the zoomable canvas in the desired location. Interaction is facilitated by zooming operations in both of the described interfaces. In respect to Shneiderman’s visualization mantra (Shneiderman, 1996, p. 340) we think that these visualizations provide adequate overviews and promote understanding in relations among artifacts. Zooming operations allow to easily explore the visual structure and additional navigational aids might support history functionality. Nevertheless, the visualizations lack detail-on-demand features.

### Process Visualization

Abstractions and relations in process steps and artifacts

We think that the introduced hierarchical visualization options in ZUIs may be utilized to reflect the characteristics of the user interface design process. As there are multiple discrete process steps as well as transitions between artifacts, we think that two combined ways of visualizing structure are adequate. On the one hand, process steps have to be visualized in sequential relation to each other. On the other hand, process steps include multiple artifacts, which are again characterized by work transitions, which can be considered as mutual relations. We therefore investigate nested trees as structure-providing fundamental to visualization. Nevertheless, visual affinities between artifacts and process steps should be utilized to promote reflection and thinking in design by free association. Consequently, the structured visualization should be flexible enough to allow manual placement of artifacts.



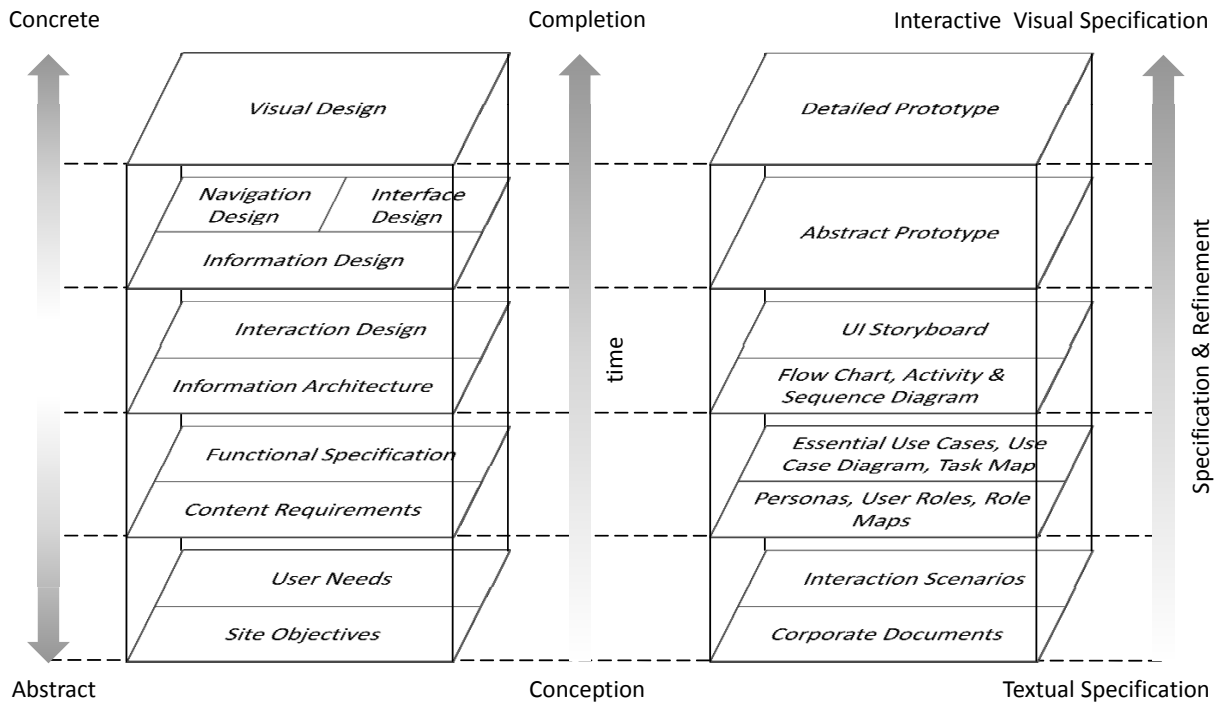


Figure 5.13: Abstraction Levels in UI Design Processes, adapted from Garrett (2002)

Figure 5.13 (left) shows a conceptual model of the interface design process by Garrett (2002). He conceptualizes the development of user interfaces in terms of five levels, which build up the final interface based on discrete steps: strategy, scope, structure, skeleton and surface. The model presents a classification of the development process from conception to completion as well as from abstract to concrete. The bottom layer, with the highest degree of abstraction is the "strategy plane". It is concerned with understanding fundamental constraints and objectives of the interface. The "scope" plane emphasizes on functionality and on content, while the "structure" plane covers typical interaction design tasks and concludes with a conceptual model. Eventually, the next two planes, "skeleton" and "surface" cover navigational design and the aesthetics of actual interface design. Garrett (2002) proposes to employ a simple graphical language to map out informations within the "structure" and "skeleton" planes. Recommended techniques are wireframes and storyboards.

A five-level model of the design process

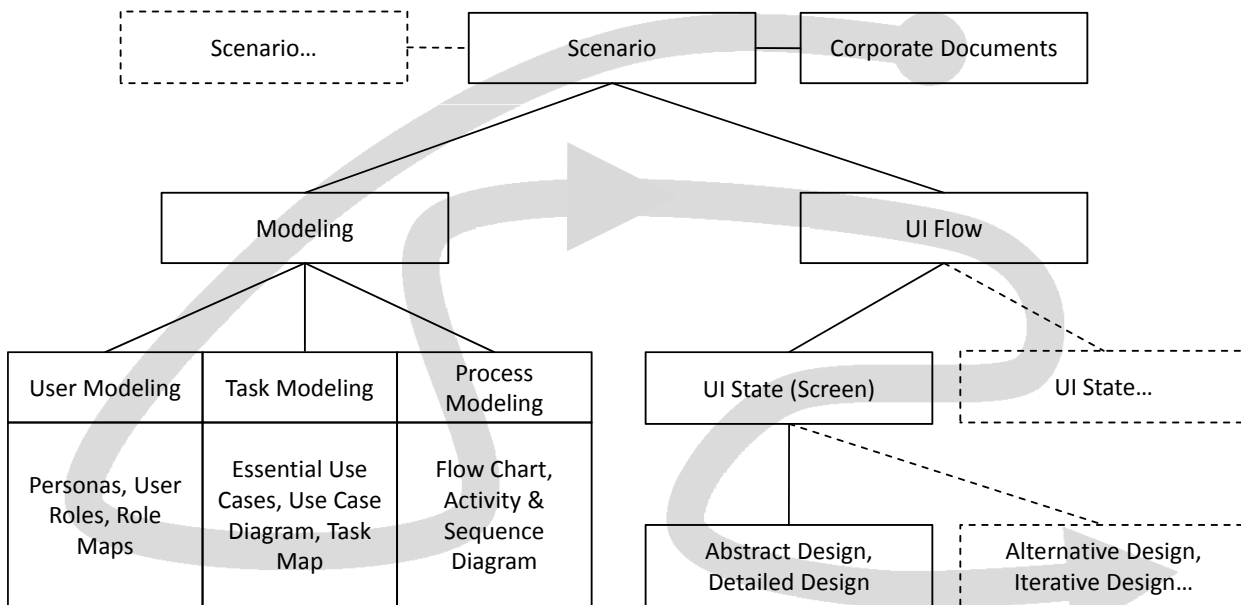


Figure 5.13 (right) aligns Garrett's model to the artifacts employed in the visual specification framework that was introduced in chapter 4.2.2—"Interdisciplinary Selection of Artifacts". Each step from the initial textual means of specification to the final interactive specification can be mapped to the corresponding planes of Garrett's model. The mapping reveals that a multiple number of artifacts within the "scope" and "structure" plane coexist on the same level. Consequently, we think that not only relations between the discrete planes prevail, but also within these planes between related artifacts. The overall process is therefore characterized by a certain amount of hierarchy that has to be investigated further.

Mapping of artifacts to process levels

In terms of visualization and interaction and in reference to Shneiderman's mantra, we want to provide a feeling of zooming into details of the design process from an initially abstract point of view (overview). Zooming therefore starts at the bottom plane and continues to the top plane. Because of the number of artifacts within some planes, a continuous zoom seems not adequate. In contrast, several decisions have to be made when moving between artifacts in process steps. Additionally, some artifacts like scenarios are frequently used for initial problem decomposition and therefore multiple scenarios may exist in parallel. Accordingly, we aim on investigating possibilities in a hierarchical layout of the process steps that combines the different levels of abstraction with decomposition tasks throughout

Decomposition of tasks imposes hierarchical structure



**Figure 5.14:** Hierarchical task decomposition of the UI design process

the process.

Process  
decomposition and  
artifacts



Figure 5.14 shows a hierarchical mapping of process steps, artifacts and decompositions within the process flow in from of a task-decomposition tree. The hierarchical structure is derived from characteristics of the tasks and artifacts within our Interdisciplinary Specification Framework. In our experimental setup, we regard Scenarios and Corporate documents as entry point for the overall process. As corporate documents may include detailed descriptions of the project objectives, contextual informations about the target domain and corporate guidelines, they are usually decomposed in several scenarios of use and eventually define a number of interaction scenarios. Therefore, several scenarios may exist depending on the scope and size of the developed interface. Scenarios are considered as an element, which makes sure that all aspects of an interaction are respected. Scenarios are applicable throughout the overall process (see Chapter 4.1.2—“Widely adopted Structured Approaches”) and take various forms. Consequently, we think that they are a suitable tool to partition the complexity of the design process into smaller pieces that are easier to handle. We therefore argue that scenarios are an early method to split the process in order to make it more tangible and accordingly propose an initial layer, composed of scenarios and corporate documents (see Figure 5.14 (1)). Each scenario is then a fundamental to building up the interface with the remaining process artifacts. As designers enter the design phase, they are confronted with a duality of conceptual modeling and actual physical design of interfaces (see Chapter 4.1.1—“Integration with Software Development” Figure 5.14 (2), (3)). By employing this kind of structure, we think that the sequential nature of these two tasks is respected, without neglecting the relations between models and their corresponding interface designs. Based on the assumption that concrete user interface designs are a subsequent product of a well-defined conceptual model, we think that visualizing relations between modeling and design is essential. Modeling tasks are then again divided based on their sequential appearance within the process flow (see Figure 5.14 (4)). Garrett (2002) proposes storyboards and wireframes as tools for presenting interface designs in a higher degree of abstraction. Similarly, we propose to employ flow diagrams or storyboards as entry-point into the design phase. Storyboards are characterized by a multiple number of connected user interface states. We therefore decompose the design task into multiple UI states (see Figure 5.14 (5)). Eventually, based on the iterative nature of the design process, a certain UI state may have various representations, like abstract or detailed designs as well as iterative or alternative designs (see Figure 5.14 (6)).

## Nested Containers and Connections

Based on the five levels of abstraction, presented in Figure 5.13 and the developed hierarchical structure (see Figure 5.14), a nested tree visualization can be created which can be utilized to make the process structure and artifact relations visible and explorable by zooming interactions. Figure 5.15 shows a simplified mapping of both visualization concepts into a nested tree visualization, similar of that in Figure 5.12 (left). Hierarchical aspects are visualized by nested rectangles that may include subordinate rectangles and so on. We will introduce the term "Container" for these rectangles in reference to a real-life metaphor. Relations between containers are visualized with simple connecting lines.

Nested tree visualization with artifact containers and connections

"A container has an inside and an outside and you can put things in and take things out. This is such a fundamental concept that it is the basis of the way that we conceptualize the world."  
(Benyon et al., 2005, p. 596)

Containers therefore provide a hierarchical structure that is familiar to users because its concept is derived from a real-life metaphor. When referring to the proposed room-metaphor, containers are objects in a room that inherit other objects, like boxes for example. In the case of the "Design Room", these correspond to walls or whiteboards within the room. A room may contain at least four walls; walls again may contain several whiteboard; whiteboards once more may include one or more spatial segments that designers created with lines. A container therefore simply represents a hierarchical spatial structuring unit. Based on the spatial location of containers, affinities between these objects can be identified. Even better though is to explicitly visualize relations with lines. On a real whiteboard, this connection may simply be drawn between two segments. In the proposed nested tree visualization, these relations may be explicitly presented by connecting lines. The overall visualization concept therefore is inherently different from common nested tree visualizations, as containers are dynamically movable within their constraints and therefore allow visualizing affinities, based on spatial location and on explicit connections.

Containers for flexible relations and hierarchical structure



Figure 5.15 presents the hierarchy of containers within the nested visualization. The initial document container (b) inherits document instances (b) that may be arranged and connected (h) based on affinity. Similarly, scenario containers (c) contain model containers (d) as well as screen containers (c). By connecting screen containers, a UI storyboard, which represents relations and UI-flow, can be created easily. Model containers provide space for model instances (e) that represent the three different subsequent forms of modeling tasks. Again, relations between models can be visualized by connections. Finally, screen containers include screen instances (g), which are used for iterative and alternative designs. The temporal relations between screens instances can be visualized based on spatial location and overlapping. In overlapping screen instances, the screen in front can be regarded as an iterative improvement of the underlying screen, while aligning screen instances spatially separated can be considered as an alternative design.

Visualization notation



We think that the proposed visualization provides an adequate visual "drill down" from context to detail in combination with zoomable interaction techniques. We identify abstraction and relations as the main constraints in visualization. Consequently, the various degrees of abstraction are visualized with scale. Detailed representations are smaller and need to be explored, while artifacts with higher degrees of abstraction are visible from overview position. Abstractions are distinguished by process steps and work transitions. Therefore, process steps or abstraction levels are clustered according to their order in the process progress based on the container visualization. Affinities are explicitly presented within certain process steps and between work transitions as connecting lines and in addition by spatial clustering with containers.

Clustering of artifacts according to abstraction and affinity

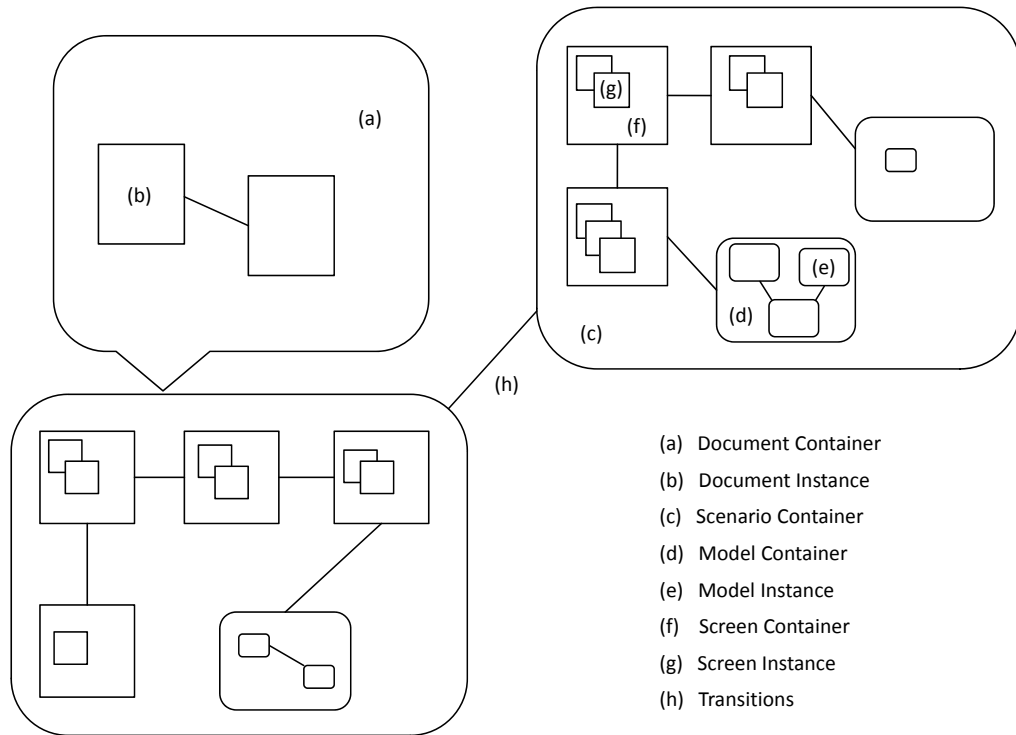


Figure 5.15: Nested Process Visualization

### 5.2.4 Collaboration

Essential needs for collaboration are to understand and to generate ideas

In Chapter 5.1.2—“Communication”, collaborational work style was briefly discussed. As a result, the essential needs in communication support were elaborated, namely: A common language in communicating design, sharing of artifacts, visual specification techniques and design rationale support. As in any conversations, collaboration requires the participants to establish a common understanding of the tasks they have to perform and the artifacts they are looking at. While it is a secondary activity in many domains, in design tasks, in research environments and in brainstorming sessions, it is essential to generate ideas. Designers typically gather to discuss work and then collaboratively generate ideas that support this work. Therefore, the main jobs in collaboration are to generate ideas and to understand these. In the following, tool support for these activities is discussed.

#### Meeting and Decision Support

Three types of collaboration support

According to Dix et al. (2003, p. 679ff), there are three types of computer supported systems, where the generation of ideas and decisions is the primary focus: Argumentation tools, meeting rooms and shared workspaces.

Argumentation support

Argumentation tools are software systems that support the recording of design decisions. Similar to the concept of “design rationale” (see Chapter 2.5—“Treasuring Design Experience”) these methods aim on recording decisions and the arguments that led to those decisions. As described before, such a design rationale may help improving traceability throughout the process and clarifying future design options. Both of these supporting tasks affect a potential groupware system. On the one hand, these tasks primarily concern one-way communication. A future designer may learn from decisions that were possibly made years ago. Therefore, it can be considered as a communication task rather than a collaboration task. On the other hand, it can also be two-way, as design rationale may also be employed to commu-

nicate decisions between groups or individual actors. Nevertheless, both options focus on "asynchronous communication". Argumentation support systems often employ hypertext techniques like Wikis that are used to support design teams as well as individual actors instead of co-authored documents. As the design of interfaces is a complex process that may cover periods of several months to several years, it is unlikely that several actors are manipulating the same artifacts at the same time. Synchronous communication is therefore mainly conducted in meetings. According to Dix et al. (2003, p. 680), most argumentation tools are typically used within the same office or on individual computers. Consequently, they are classified as "asynchronous co-located" systems.



In contrast, meeting rooms are utilized for face-to-face communication. Meeting room support systems provide a similar accessibility of artifacts as in design rooms by making them available in meeting rooms as shared documents and files. According to Heim (2007, p. 24f), practical implementations have various forms. While networks provide the fundamental infrastructure for collaboration, computer-supported cooperative work (CSCW) describes the way in which computers are used to support team communication. Current CSCW environments involve large projected displays, like digital projectors, wall-size displays or "smart screens". Smart screens facilitate additional means of input, like handwritings on interactive whiteboards or multiple user input. However, employment of these rather new methods in practice remains elusive. Nevertheless, the rising of large, high-resolution displays is paving new ways to leverage the powerful design room metaphor. As computer displays move away from the desktop and become walls, they are offering new means to interact with a wide variety of artifacts. The shared screen then takes the form of an electronic whiteboard, where actors can bring up artifacts for discussion that they created asynchronously. In principle, there are two ways of interacting with the artifacts when discussing in a group: The screen is controlled by multiple persons at the same time or by a single person that acts as a meeting facilitator. The first way of interaction raises numerous problems, as several actors may decide to interact on the screen at the same time. Some systems therefore employ "locking" mechanisms to synchronize input. In practice, these systems are not efficient, as they interrupt the meeting flow. As an alternative way, a facilitating actor may control the screen and negotiate control options with meeting participants. According to Dix et al. (2003, p. 682), these techniques bring various benefits as they utilize the "social protocol" that is characterized by the fact that all actors are in the same room and are able to talk to each other. If the participants talk while they control the screen, all others are aware of who is acting on what. Participants may also interfere with each other's actions. Dix et al. (2003, p. 682) calls this beneficial characteristic "deixis".

Meeting support



"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."

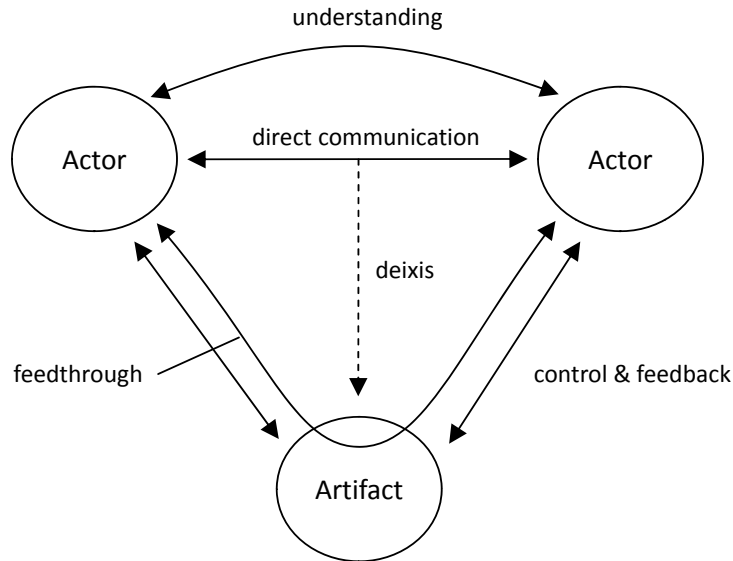
(Dix et al., 2003, p. 682)



Communication in meeting rooms or design rooms is not only supported by direct communication and common understanding, but also determined by artifacts that accompany the design process. Figure 5.16 shows a diagram that visualizes this relationship. Direct communication therefore is supported by the "deixis" that utilizes shared artifacts for communication. Actors may refer to artifacts as part of the communication task by simply pointing at them. Additionally, artifacts are not only supplemental to direct communication tasks, but may also be a mean of communication themselves. As actors control and manipulate artifacts, others may observe these actions and respond with feedback. According to Dix et al. (2003, p. 699), communication through artifacts can be as important as direct communication between actors. Therefore, computer supported collaboration has to make artifacts accessible, not only for discussion, but also for manipulation.

Communication through artifacts





**Figure 5.16:** Control and feedback from shared artifacts, adapted from (Dix et al., 2003, p. 699)

**Shared workspaces** In contrast to large screens in meeting rooms, shared workspaces are often similar to the concept of the room metaphor. They extend the shared screen, like on electronic whiteboards and utilize shared workspaces as medium for facilitating co-located collaboration. Actors may bring in artifacts and leave them for others or work on them at the same time. Additionally, they may attach notes on artifacts in order to communicate their ideas. Shared workspaces may also be used to conduct meetings when all actors are permanently co-located but require synchronous communication. Each participant is usually assigned to a dedicated work area within the shared workspace. By exploring the different workspaces, they may then link their work to each other’s or refer to another’s artifacts. Successful implementations of these kind of systems that are primarily based on the room metaphor can be found in TeamRooms (Roseman and Greenberg, 1996, 1997) as well as COLAB and Cognoter (Foster and Stefik, 1986).

**Lack of awareness in computer supported collaboration** According to Gutwin (1997) and Gutwin and Greenberg (2002), all interfaces that support collaborative work should also facilitate “collaborative awareness”. This means that a system should allow actors to build up knowledge of the common work progress. Actors should know about what others are doing and what they are doing. In practice, this is a major bottleneck of effective collaboration. Gutwin (1997) argues that real face-to-face work cannot be realized with a computer based system, based on three constrains that limit the effectiveness of computer-facilitated collaboration: Reduction of the user’s perception of the workspace, suppressing the user’s expressivity and limiting the ways in which artifacts are used.

**ZUIs to support collaboration awareness** Therefore, Gutwin (1997) examined a collaborative system based on a visual interface, the “radar view”. Although it is not a ZUI, it contains an overview window of a shared canvas as well as a larger scrollable frame in which actual work is performed. In contrast to a ZUI, this interface has a limited resolution and no zooming functionality, which is a major drawback. Therefore, collaboration awareness may be improved by employing a shared zoomable canvas. Zoomable User Interfaces were previously introduced concerning single user interaction. Nevertheless, according to Yang (1999) they may also be suitable to support collaboration tasks. Nevertheless, research in ZUIs as collaborational tool mainly focuses on real-time remote work, which is not the focus of this work. However, we think that a zoomable whiteboard application may be used for other collaborational tasks, like face-to-face interactions in meetings and asynchronous interactions as shared workspace.

	same place	different place		same place	different place
Same time	<b>Face-to-face interaction</b> Meeting rooms, single shared displays, wall displays		Same time	<b>Face-to-face interaction</b> Shared zoomable whiteboard on large display	
different time	<b>Remote interaction</b> Argumentation tools, shared workspaces	<b>Communication &amp; Coordination</b> Email, networks	different time	<b>Remote interaction</b> Linked artifacts and feedback in shared workspace	<b>Communication &amp; Coordination</b> Specification file (artifacts & prototype)

**Figure 5.17:** Time/space Groupware Matrix (left), Collaboration Framework (right)

### Collaboration Framework

Figure 5.17 (left) presents common groupware systems in a time/space matrix. Based on the available options for supporting meetings and decisions and the characteristics of the target domain, we think that efficient tool support should facilitate co-located (same place) and synchronous (same time) collaboration in the first place. Consequently, face-to-face interaction in meeting rooms should be supported, as well as shared displays or wall displays. As most design work is done in the same office and because we think that remote meetings harm creativity, we do not consider synchronous remote meetings. In contrast, remote interaction (different time / same place) needs to be supported to communicate design rationale to project members. Therefore, shared workspaces and argumentation functionality, as hypertext, needs to be supported. Eventually, a tool design has to support external communication (different time / different place), which is facilitated with specification functionality.

Classification matrix for collaboration needs



Figure 5.17 (right) shows a mapping of required groupware functionality to a ZUI implementation. Meeting rooms (same time / same place), which are usually equipped with large shared displays, may be utilized to present a shared view on all process artifacts during meetings. The zooming mechanism is then utilized to focus on specific artifacts for group discussion. In respect to synchronizing input, we think that one single facilitating actor should manipulate the zoomable whiteboard and the artifacts. As decisions are being made by face-to-face communication, they can be noted on the whiteboard for future reference. To facilitate co-located asynchronous collaboration (different time / same place) we utilize the design room metaphor. As the zoomable whiteboard imposes a structure, containers and persistence, artifacts do not have to disappear after meetings. As the process continues, all elaborated artifacts persist, as they would do in a regular room. By linking related or subsequent artifacts and by adding feedback, the design rationale becomes traceable. Individuals might explore the various created artifacts by zooming from abstract into detailed views, which facilitates a design rationale. Nevertheless, the functionality of asynchronous collaboration demands for an accessibility of the shared workspace for all actors throughout the process. Therefore, the workspace content has to be available as a file, database or other shared data representation. Similarly, external communication (different time / different place) requires to communicate artifacts as a part of a specification document. Consequently, artifacts and interface designs should also be made available to remote actors.

Mapping of collaboration support to a zoomable whiteboard





## 5.3 Physical Design

Overview,  
interaction,  
visualization &  
collaboration  
concepts

In the following, the final design of the developed interdisciplinary specification tool is presented in reference to the previously elaborated concepts. The descriptions are kept brief and details are only provided if they are necessary to understand the basic concept. Overall, this chapter focuses on key features and the rationale behind them. The tool, called INSPECTOR (Interdisciplinary Specification Tool) is introduced by guiding through its interface components before interaction concepts and navigation features are described. The presentation is accompanied with references to previously described conceptual foundations as well as investigated interaction and navigation concepts. Accordingly, the rationale behind the interface design is to be revealed by referencing previous described fundamentals. Thereafter, the integrated visualization levels are described in a top-down manner. Eventually, features are described that aim on supporting efficient collaboration in design before all features are evaluated against the initial requirements.

### 5.3.1 Overview

INSPECTOR at a  
glance

INSPECTOR's interface is based upon the whiteboard metaphor and aims on taking advantage of design room features. Therefore, it extends this metaphor based on a zoomable canvas to support spatial relation between artifacts. Consequently, all artifacts that are created throughout the design process are placed onto a single canvas in a structured manner. In order to effectively address the characteristics of design processes, INSPECTOR employs a hierarchical notation of artifacts that reveals various degrees of abstraction and relations among artifacts. Combined with advanced zooming techniques, exploration of the problem and design space is efficiently integrated. INSPECTOR's interface design respects currently dominant interaction patterns to allow adoption in practice but also integrates innovative visualization and navigation concepts to provide the feeling of a tangible UI rationale specification.

Interface  
components

Figure 5.18 shows the main window of INSPECTOR in overview mode, which means that the view is zoomed out to display all contents of the whiteboard. The interface consists of a zoomable canvas (1) that displays a hierarchical zoomable structure, palette tools (2), a toolbar (3) and optional non-modal floating windows (4),(5),(6) that serve context & detail purposes. The palette (2) is aligned to the top-right of the whiteboard only in overview-mode. In zoomed-in views, the palette is attached to focused container shapes. The zoomable canvas, which displays all modeling and design content in a nested container notation (see Chapter 5.2.3—"Interactive Visualization") dominates the interface and nearly takes the whole screen. Floating detail & context windows are displayed as semi-transparent overlay windows and can be activated or deactivated as well as freely positioned by the user.

Whiteboard  
features

Overall appearance of the interface is similar to that of other whiteboard interfaces (see Chapter 5.2.1—"Whiteboard Interaction"). As content creation and manipulation has to be intuitive, existing interface patterns provide beneficial assistance. The zoomable view on a whiteboard (1) closely resembles interface concepts that can be identified in currently employed interface tools. This similarity therefore facilitates ease-of learning and ease-of-use and consequently promotes adoptability in practice. By employing familiar interaction patterns, introduced in Chapter 5.2.1—"Interaction Patterns" whiteboard, for whiteboard interaction users will instantly recognize key features and their functionality. As adoptability is top-priority in design, recognizable patterns are promoted throughout all parts of the interface. For example, palettes (2) and toolbars (3) are widely spread in currently used tools in UI design practice. The palette in INSPECTOR is utilized exclusively to facilitate content creation. Whiteboard containers, as well as artifacts, like shapes, pictures or documents are simply dragged & dropped onto the zoomable canvas. Direct manipulation features, like pointing, selecting, resizing and dragging are employed to manipulate the content afterward. A toolbar offers additional access to tools like connections, scribble pens, navigation features and view properties.



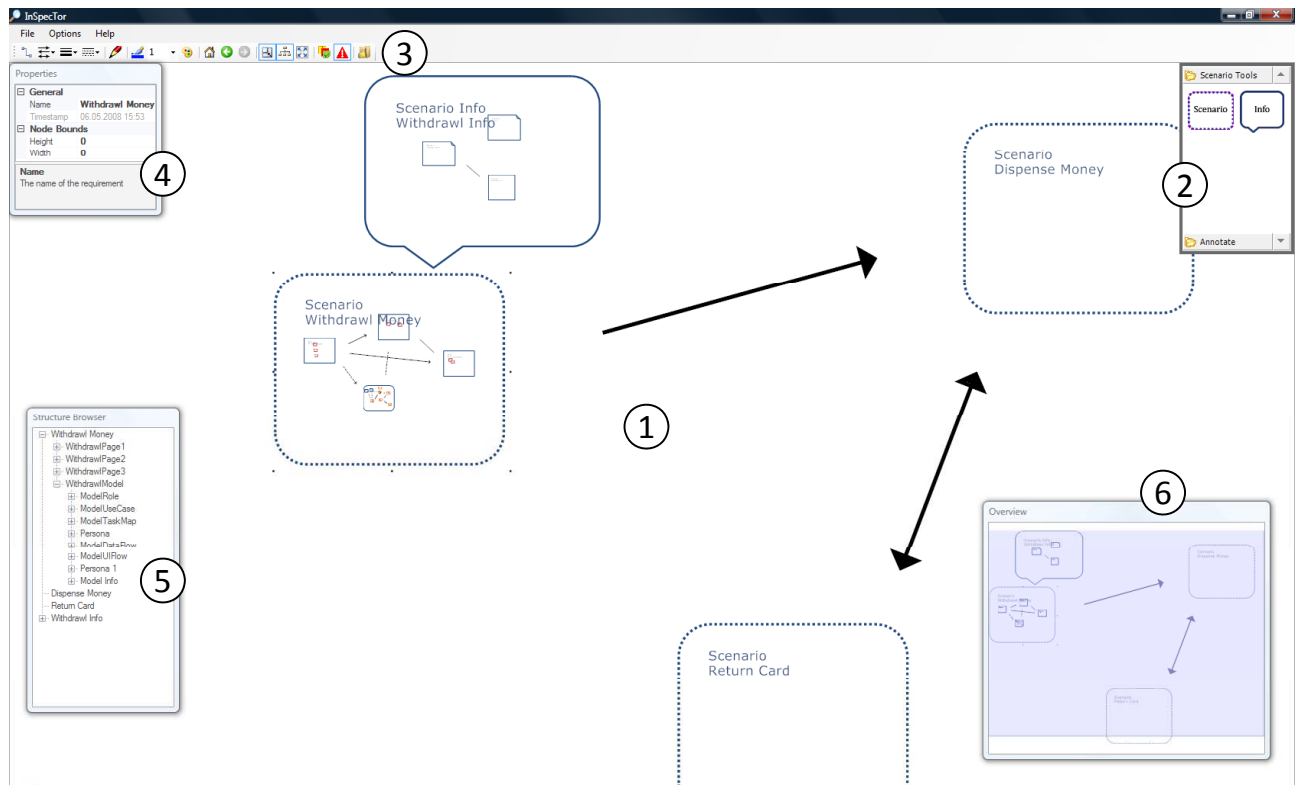


Figure 5.18: INSPECTOR's main window (see C—"Screenshots", Figure C.1 for a full scale version)

However, a major interaction concept that distinguishes the INSPECTOR interface from common whiteboard interfaces and therefore stands for the main innovation is the zoomable canvas in combination with a structured notation. Instead of a scrollable space, INSPECTOR utilizes a zoomable space that displays all process artifacts positioned in relation in one single view. Based on zooming & panning techniques as well as goal-directed zooming methods that are heavily oriented at widespread state-of-the-art zooming interface techniques (see Chapter 5.2.2—"Examples & Interaction Patterns"), the view on the whiteboard is altered to focus on specific artifacts within the design space. The latter are integrated into a nested tree visualization, described in Chapter 5.2.3—"Nested Containers and Connections". Therefore, the canvas displays a hierarchical structure of containers and links between these containers. When referencing the previously introduced metaphors (see Chapter 5.1.3—"The Design Room Metaphor"), containers can be seen as separate whiteboards with a specific purpose that are spatially arranged and connected based on the design process within a design room. Consequently, the zooming functionality blends the whiteboard metaphor with the room metaphor. Overall interaction and navigation aims on leveraging the benefits of the design room as common workspace by combining its features with the already adopted whiteboard interface concept. By utilizing spatial relations, container properties, connections and free positioning, we aim at facilitating creative thinking by free association (see Chapter 4.3.1—"General Guidelines"). Some features of the design room metaphor are not instantly visible, as the interface appearance is heavily oriented towards the whiteboard. Nevertheless, room features are blended in and utilized in collaboration (see Chapter 5.2.4—"Collaboration").

Design room features

### 5.3.2 Interaction Concept

In the following, the interaction concept of INSPECTOR is described by guiding through the most important interface components. Their logical structure, functionality and purpose in relation to the overall tool design is explained and reflected in figures. The following

Focus on interaction concepts and interface components

descriptions focus on describing general interaction concepts that are consistent in all parts of the interface. Nevertheless, means of expression are inherently different in the various parts of the zoomable structure. Therefore, the different levels of expression that resemble the various degrees of abstraction found in design practice are described along with the visualization concept later on in the next chapter.

### Zoomable Structured Containers

Guided content creation with containers and palettes

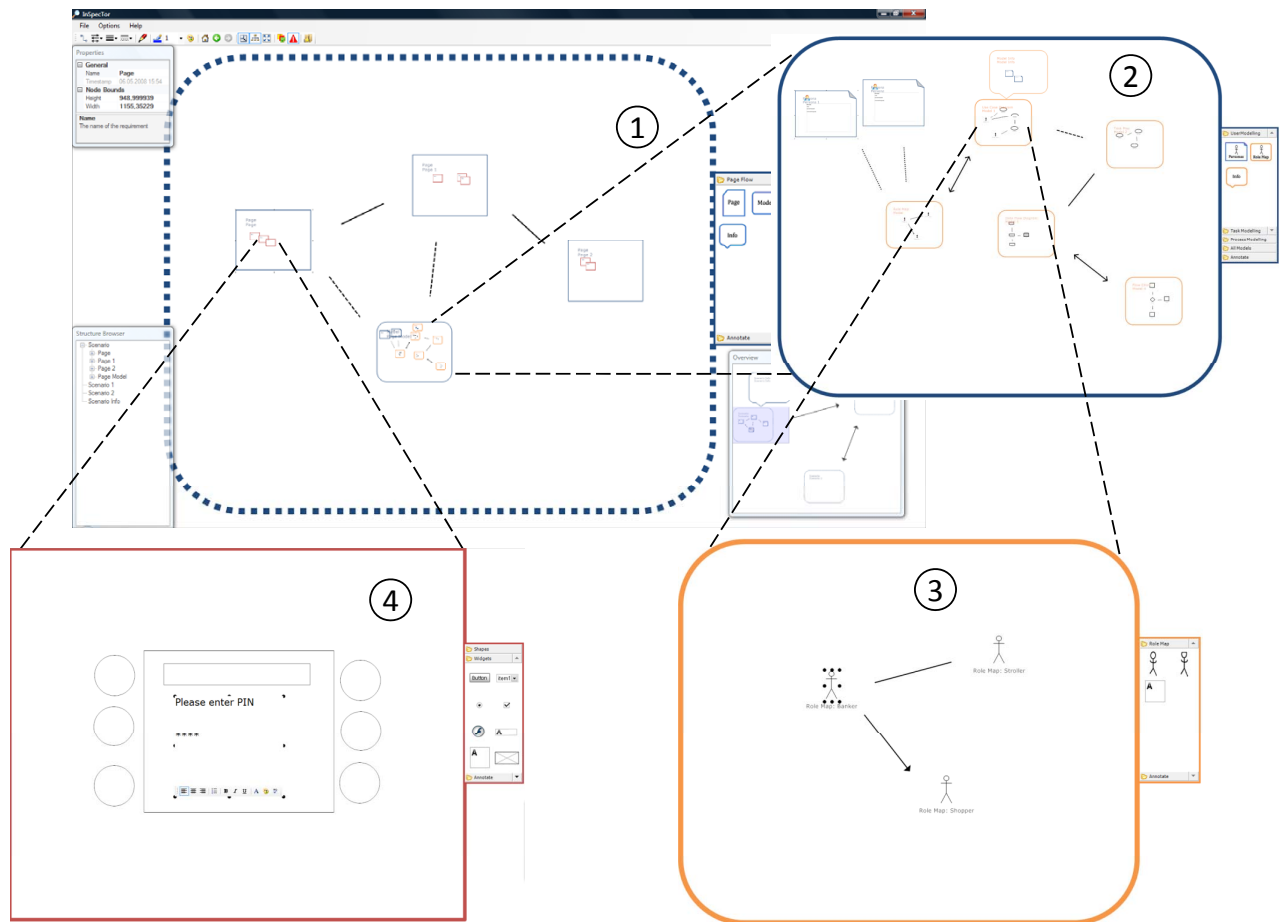
Figure 5.19 shows an illustration of the zoomable canvas and zoomed in views. Parts of the nested structure are magnified to visualize the change of view when the user zooms in on specific containers within the nested structure. In the background of figure 5.19, INSPECTOR's main window is displayed. A container (1) is centered in the zoomable viewport. Within this container, shapes can be placed by dragging & dropping from an attached palette. Each shape stands for a specific artifact within the overall design process. The shapes are then placed and freely arranged according to their implicated meanings. In case of (1), the whiteboard container provides artifacts within its palette that allow the user to create a storyboard notation. Therefore, artifacts like pages, corresponding models and information bubbles can be arranged and connected to resemble a UI flow storyboard (see Chapter 5.2.3—"Process Visualization" for a detailed explanation of the employed process artifacts). The shapes themselves are again containers that provide palettes that are specific to the containers contents and so on. When zooming into a container shape (2), subsequent containers can be created which are then manipulated by zooming into their shapes. Again, the contents of the attached palettes change according to the purpose of the containers (3)(4). Once a certain degree of detail is reached by zooming in, concrete diagramming shapes that are specific to the artifact container are attached in the palette. Container shapes within INSPECTOR are color-coded to instantly reveal the intended content to the user. As a result, content creation in INSPECTOR's nested structure is guided through context-specific palettes and colors. This interaction concept is utilized throughout all visualization levels (see Chapter 5.3.3—"Visualization & Navigation concept"). A major goal of this methodology is to guide the user through the levels of abstraction that accompany the design process by explorative zooming.

Manual zooming techniques facilitate exploration

Actual zooming itself is realized by employing widespread interaction patterns for zoomable user interfaces that were introduced in Chapter 5.2.2—"Spatial Navigation". Available zooming techniques therefore are manual panning & zooming, goal-directed zooming and semantic zooming. All zooming techniques in INSPECTOR are also animated to improve context awareness and to harness the human perception for spatial navigation. Zoom durations and timings are again aligned to implications from ZUI research (see Chapter 5.2.2—"Zoomable User Interfaces"). INSPECTOR's zooming & panning techniques are designed in a way to provide the feeling of standing in front of a large whiteboard. Scrolling the mouse wheel up and down zooms the view in and out. The view is zoomed in along the mouse position when scrolling the wheel up. Scrolling the wheel back is implemented as a straight zooming back. This methodology is one of the most important features in efficient ZUI navigation (see Chapter 5.2.2—"Zoomable User Interfaces"). In respect to a real whiteboard or a design room, this corresponds to moving towards a whiteboard section to focus on specific artifacts and moving back to gain overview on all whiteboard contents. Panning is facilitated by dragging the view in one or more directions while the mouse wheel is hold pressed. This action corresponds to moving the real whiteboard in a direction or to pushing and pulling away paper sketches. While these basic zooming techniques allow exploring the hierarchical structure in a natural way, goal-directed zooming techniques are mandatory to avoid desert fog issues (see Chapter 5.2.2—"Zoomable User Interfaces").

Goal-directed zooming facilitates efficient guidance

Therefore, goal-directed zooming is facilitated by mouse selections as well as command triggering supported by floating windows and other features that are introduced later on. Basic goal-directed zooming, namely automated zooming, is achieved by simply double-clicking a shape on the zoomable canvas (see Chapter 5.2.2—"Examples & Interaction Patterns"). Single click is reserved for selecting shapes. Automatic zooming is integrated for a stepwise



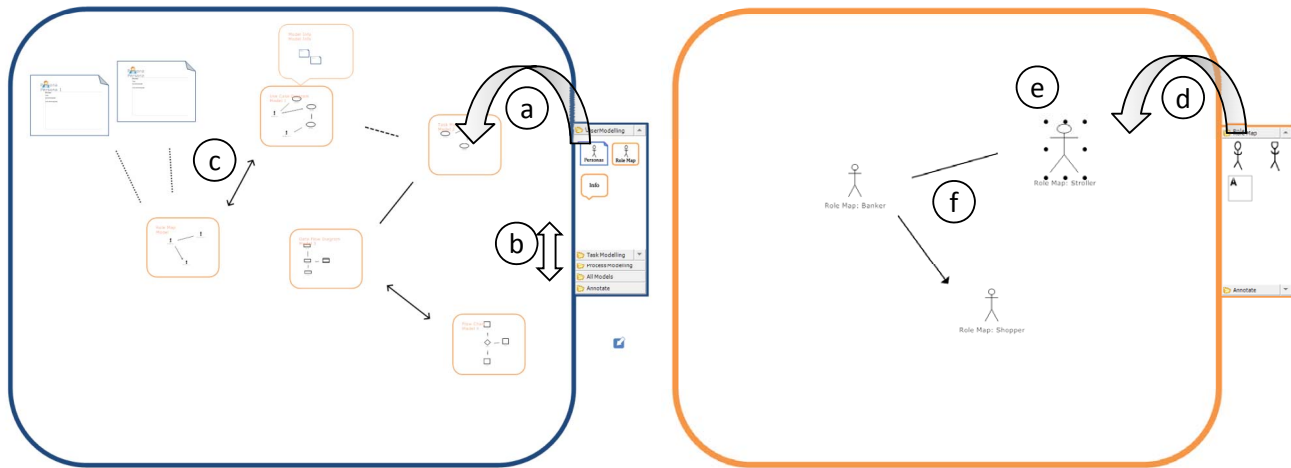
**Figure 5.19:** INSPECTOR's Zoomable Nested Structure (see C—"Screenshots", Figure C.2 for a full scale version)

exploration of the zoomable structure. Consequently, double-clicking a shape that is embedded into a container zooms in on the container. The view is then automatically zoomed in and centered onto the selected container. Again, zooming is smoothly animated and timed accordingly to allow efficient exploration. This procedure is then repeatedly applied as stepwise exploration continues. Zooming out is facilitated by double-clicking into the white space around container shapes. Again, zooming is stepwise and therefore centers the view on the container that embeds the currently focused shape. In combination with the later described navigation aids, we think that goal-directed zooming improves navigation capabilities of the user dramatically.

### Content Creation

As previously described, each container on the zoomable canvas can be regarded as an instance of a whiteboard in real life. Because containers can actually inherit other containers, each whiteboard container is therefore a specific spatial section in a larger context. In reference to the design room metaphor, there may exist several whiteboards within one room. Usually, whiteboards in real rooms are dedicated to a specific purpose. If a whiteboard facilitates multiple purposes like user needs, tasks and flow models, designers may again structure this whiteboard according to the affinity of the artifacts they contain. In INSPECTOR, this characteristic is reflected by the structured container visualization and the contextual palette that is attached to these containers. Based on the type of container, available artifacts are presented for use. Figure 5.20 shows two different examples, meta-containers (left) and

Meta-containers  
and whiteboard  
containers



**Figure 5.20:** Palette tools for content creation - container shapes (left), artifact shapes (right)

whiteboard containers (right).

#### Meta-containers

The palette on the left is attached to a container that inherits other containers. It can be regarded as a meta-container for specific modeling containers. Containers are then dragged & dropped into the meta-container (a). These meta-containers therefore serve the purpose of relating different models to each other while facilitating effective guidance on the grouping of artifacts that have relations to each other. Available artifacts within meta-containers may exceed the amount of screen space available in the palette. Therefore, the palette features an accordion mechanism (b) to allow the display of a variety of artifacts in a structured manner. As whiteboard containers are dropped onto the meta-container, they can be freely arranged by the user and connected with lines (c) to visualize relations among them. Containers are then again explored by zooming operations. Artifacts within sub-containers are grouped which means that transformations on these containers also affect their members. This methodology enables constant rearranging throughout the design process, which is a crucial requirement to creative thinking by free association.

#### Whiteboard containers

In contrast, whiteboard containers, like that shown in figure 5.20 (right) do not inherit other containers. Instead, they resemble a simple whiteboard instance for a specific purpose. In this example, the container provides means of expression for a “Role-Map” diagram. Therefore, actor shapes are available in the palette and can be dragged & dropped onto the whiteboard container (d). Moreover, resize handles that were described in Chapter 5.2.1—“Whiteboard Interaction” allow simple resizing and translation of the artifact shapes (e). Again, shapes can also be freely arranged and connected via different lines (f). Overall interaction with these containers therefore is very similar to regular whiteboard interfaces and consequently should be familiar to all actors. As there are many different whiteboard containers and available artifacts, they are described in detail later on along the various levels of visualization.

### Pointing, Selecting & Command Ordering

#### Direct manipulation features

Within whiteboard containers, various graphical artifacts are placed, arranged and manipulated to create diagrams or interface designs. While actual content creation is facilitated by drag & drop functionality, manipulation is achieved by pointing, selecting and command ordering patterns that are familiar from other whiteboard interfaces (see Chapter 5.2.1—“Whiteboard Interaction”). Selecting artifacts is therefore realized by single-clicks. Once an object is selected, it is highlighted by handles that are added to its bounds. The object can then be resized by dragging these resize handles. Artifacts can be moved by dragging the artifact across the whiteboard. Multiple selections are achieved by dragging a selection

marquee (see Figure 5.21) or by selecting artifacts sequentially while holding the "Shift" key down. Overall interaction is exactly as actors would expect it from other design applications or drawing packages. However, these interaction patterns are applicable to all artifacts except for resizing containers because they have a fixed size.

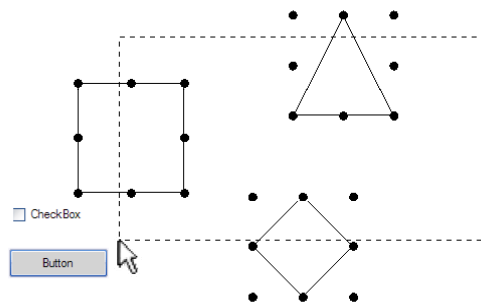


Figure 5.21: Pointing and Selecting

Context menus (see Figure 5.22) are employed to manipulate properties that are not visible and therefore cannot be manipulated with direct manipulation features. Figure 5.22 (right) shows a context menu for manipulating connections that allows selecting the appearance of connection arrows and lines. Figure 5.22 (left) shows a context menu on a shape artifact. Available options are typical for whiteboard interfaces. Besides delete and copy controls, the vertical order of appearance can be altered with four typical buttons. Additional options allow to add notes to an artifact, delete referencing links (see Chapter 5.3.3—"Linking & Tracing Artifacts") or to create a template for further use. Templates therefore represent a collection of selected artifacts for further use. Displayed in an overlay window that contains a list of created templates, artifact collections can be dragged & dropped onto the canvas. Especially in interface design, where specific interface elements are repeatedly used, templates ease workflow dramatically.

Context menus and templates

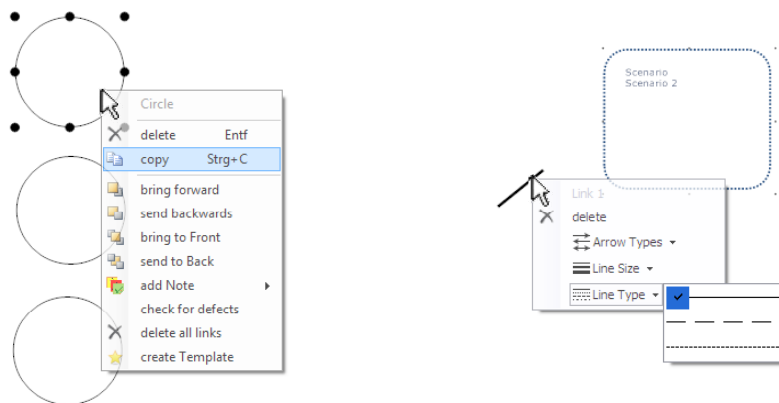


Figure 5.22: Context menus - shape tools (left), connection tools (right)

Content manipulation within whiteboard containers is also supported by a toolbar that provides tool-based manipulation features, navigation features and view options (see Figure 5.23) that are applicable to all containers. Because the toolbar is always visible no matter in which container the view is currently zoomed in, tools are universally available. The toolbar is divided into five sections according to functionality: connection tools (1), a sketching tool (2), color chooser (3), navigation tools (4) and view options (5). Connection tools (1) are similar to connection features that can be found in a variety of whiteboard interfaces. The user selects the type of arrows, line size and line appearance with a series of drop-down menus. By toggling the connection button, which is the first button in the row, the mouse cursor is charged with linking functionality. The connecting line is then created by sequen-

Manipulation features and tool options of the toolbar

tially selecting two shapes within a container. Sketching tools (2) allow sketching simple lines onto whiteboard containers. Again, the mouse cursor is changed by toggling the pencil button. Dragging the mouse over the canvas with the left mouse button pressed creates instant sketches while the pencil tool is activated. The third section allows specifying color attributes and pen sizes via drop-down menus and color chooser dialogs (3). The settings made in this section are not only applied to sketches, but to graphical shapes as well. Therefore, the user can separately select stroke color and fill color. The navigation section (4) displays navigation history tools that are familiar from browser applications. They enable the user to move back and forth between goal-directed zoom steps as well as to return to an overview position. Therefore, these buttons contribute to goal-directed zooming support. Eventually, view options (5) allow toggling floating windows that are used for detail & context features. Another button triggers fullscreen mode, which extends INSPECTOR's canvas onto the full computer screen. Additional view options are a review mode and a template browser, which will be described later on.

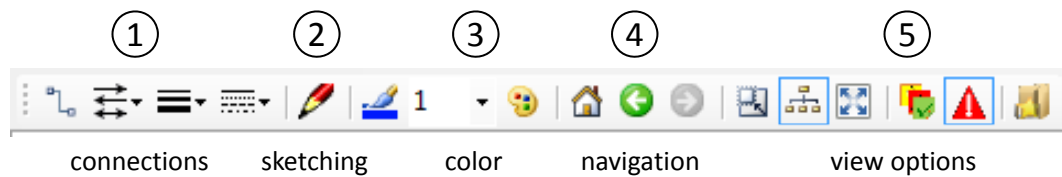


Figure 5.23: Toolbar Sections

### 5.3.3 Visualization & Navigation concept

Artifact levels,  
navigation aids and  
traceability support

After introducing the basic interaction concept, the overall connections between the nested visualization and the whiteboard interaction styles are to be revealed by presenting the distinct levels within the INSPECTOR notation. Therefore, this chapter will introduce four levels that result form the nested hierarchy developed in Chapter 5.2.3—“Interactive Visualization”. Thereafter, detail & context features are introduced that support navigation through the employed visualization. Eventually, traceability support is described that effectively visualizes the rationale and relations behind created artifacts. Artifact linking functionality additionally supports navigation purposes and prototyping features.

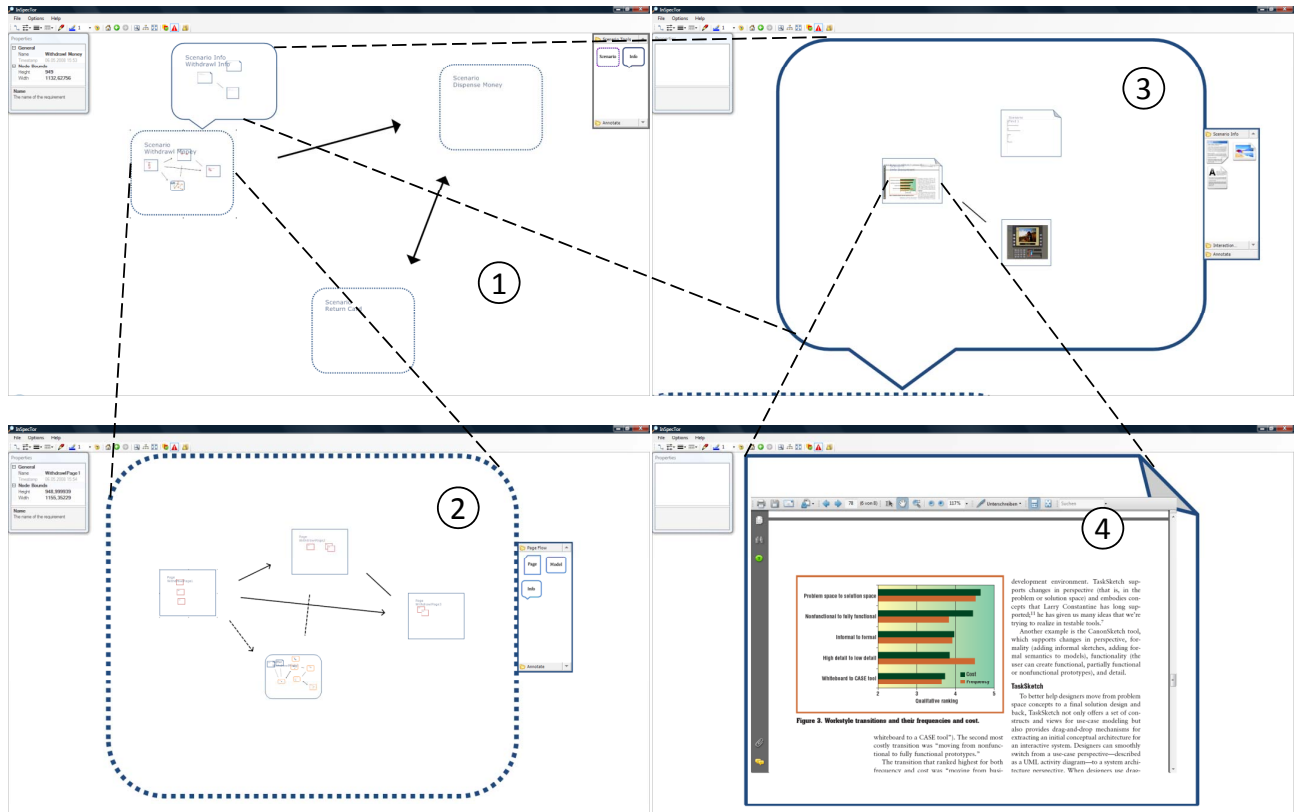
#### Artifact Level Structure

Four discrete levels  
of process  
visualization

As previously described, INSPECTOR utilizes a nested tree visualization that employs metacontainers to structure related containers and whiteboard containers that contain concrete artifacts like diagrams or UI designs. Within this hierarchy, containers are hierarchically structured according to the degree of abstraction they represent during the design process (see Chapter 5.2.3—“Process Visualization”). To reveal the overall cohesion of these structures, discrete levels are introduced that reflect different process phases as well as the distinct degrees of abstraction that accompany the process. However, these levels do not correspond to sequential zooming operations. Instead, one level is accompanied by a series of zooming operations that eventually lead to a transition into the next level. The visualization levels in INSPECTOR are in order of their appearance within the design process:

- Scenario Level
- Storyboard Level
- Modeling Level
- UI Design Level





**Figure 5.24:** Scenario Level - Scenario Map (top-left), Storyboard-Layer (bottom-left), document bubble (top-right), PDF document container (bottom-right) (see C—“Screenshots”, Figure C.3 for a full scale version)

In the following, these discrete levels are described along with figures that visualize the zooming operations that allow exploring related containers and artifacts. The example screens show a simplified interface design project for an ATM (Automatic Teller Machine), which was modeled within INSPECTOR. This example facilitates understanding relations between the design process and the way artifacts are integrated into the zoomable structure.

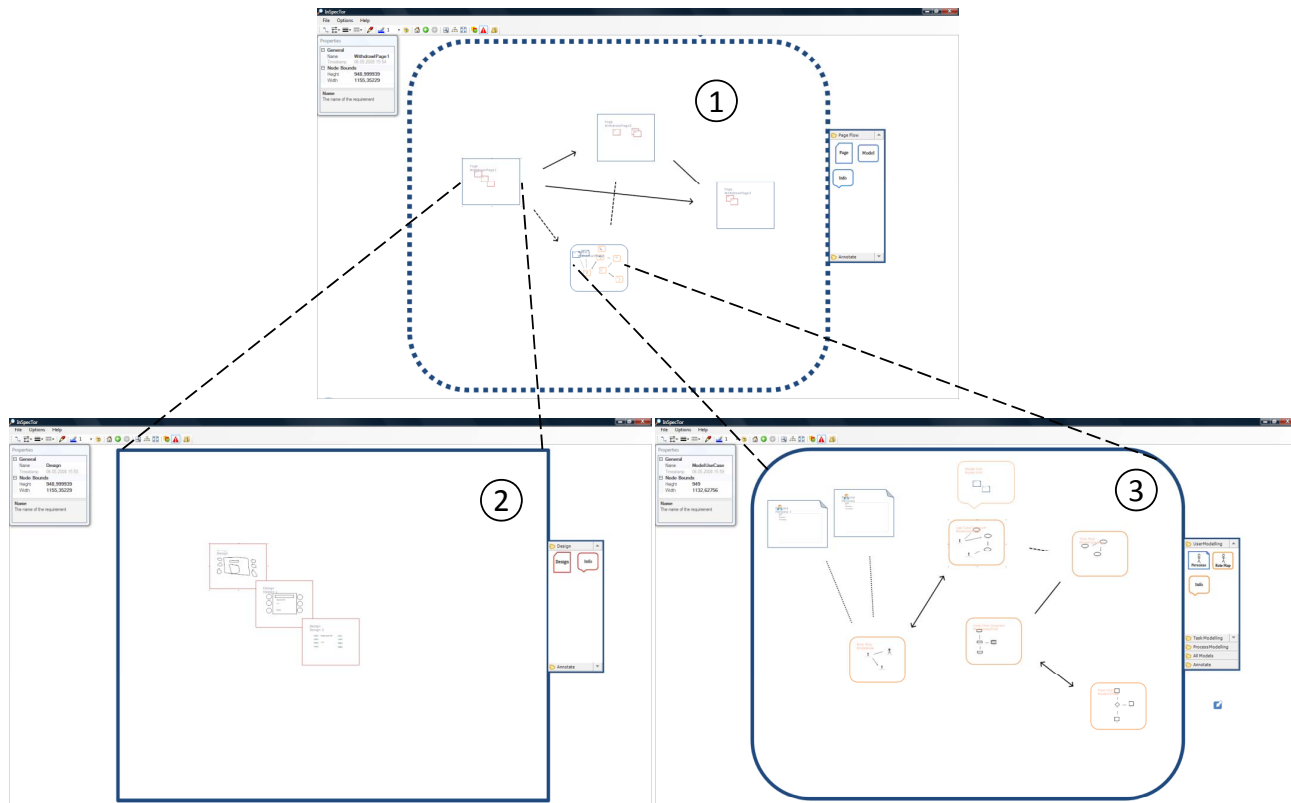
Description guided by example screenshots

Figure 5.24 shows the initial visualization level, the “Scenario Level” (see Chapter 5.2.3—“Process Visualization”). Figure 5.24 (1) shows the scenario-level in overview position. The canvas contains a number of shapes that represent “Scenarios” and “Document Bubbles”. Shapes are arranged and connected based on their affinities. Scenarios in context of an ATM are for example “Withdraw Money”, “Enter PIN” and “Return Card”. Connections allow visualizing the flow and dependencies between these scenarios. By zooming into scenario shapes, the user will approach the “Storyboard Layer” (2) to describe the scenario in terms of conceptual modeling and interface flow. Each scenario may also be supplementary described with document bubbles that are attached to a scenario shape. Document bubbles (3) may contain different corporate documents that are related to the scenario, for example statistics (4), user data, questionnaires or pictures and other media. Again, related documents may be grouped spatially or connected by lines. When zooming into documents, an embedded viewer is displayed that allows to view and browser documents (4). Supported formats include PDF documents, word-processing formats (e.g. MS Word), presentation slides (e.g. MS PowerPoint) as well as web pages or simple text files. Document support also includes an embedded internet browser that allows integrating various media, like Marcomedia Flash, streaming videos or audio files. Therefore, INSPECTOR aims on integrating corporate files in a structured manner. Document bubbles, like that displayed in Figure 5.24 are available on all levels and for all containers. Therefore, existing documents may be attached to the artifacts they are related to within the process.

Scenario Level

After entering the Storyboard Level (see Figure 5.25) by zooming into a scenario shape, the user is presented with a meta-container that allows creating a special notation that is de-

Storyboard Level



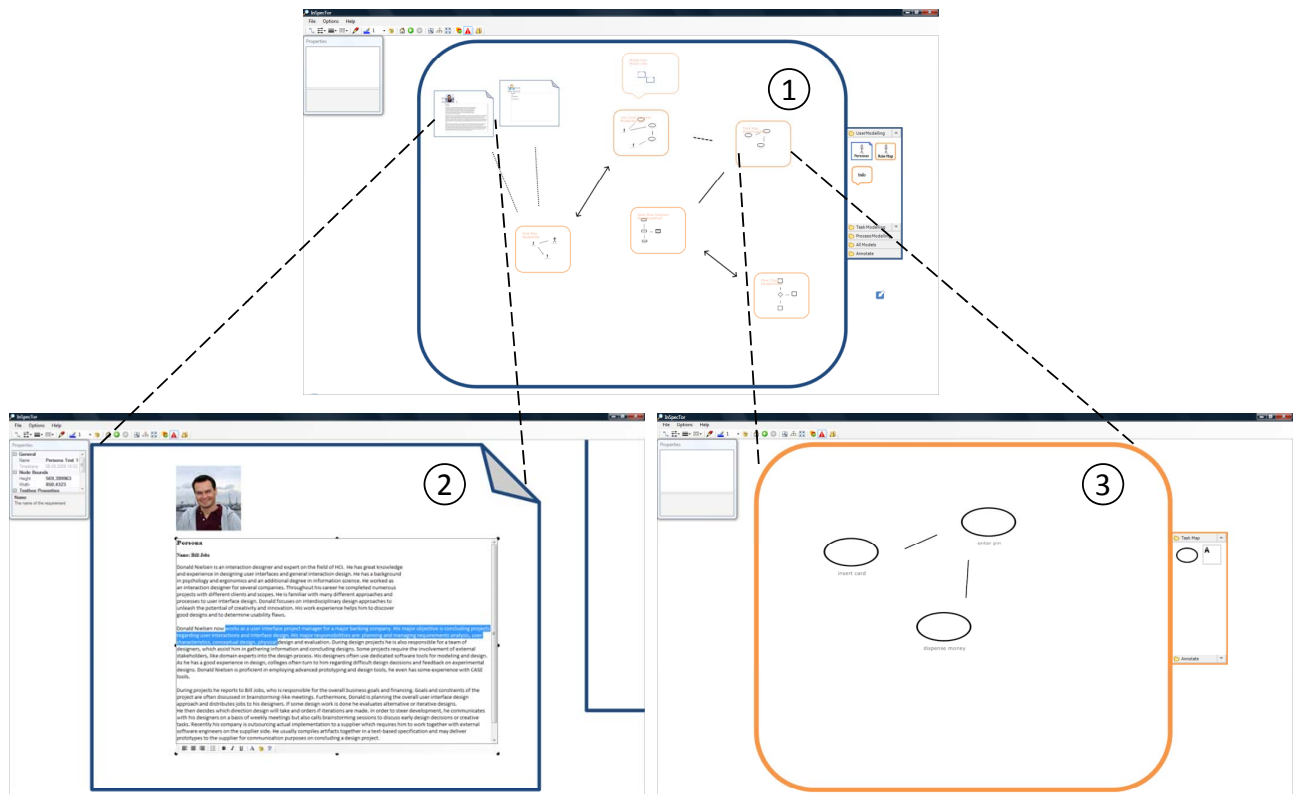
**Figure 5.25:** Storyboard Level - Storyboard Map (top), UI design layer (bottom-left), modeling layer (bottom-right) (see C—“Screenshots”, Figure C.4 for a full scale version)

scribed as a “Storyboard Map”. Utilized containers are “Pages” and “Models”. Again, document bubbles are available to attach related documents, like styleguides to pages or textual descriptions to models. Pages are utilized to specify certain stages of an interface, while modeling containers provide space for conceptual modeling, which is related to the scenario. Page containers and modeling containers are arranged within the storyboard map similar to UI flow diagrams. Modeling containers are then connected with lines to one or more pages they are related to. Different types of affinities may be visualized with different connecting lines and arrows. Once the user zooms in on the modeling container shape (3) he will enter the “Modeling Level”. A series of sequential screens that are connected with lines represent a dialog flow. In context of an ATM these pages represent certain states of the UI that are necessary for interaction (e.g. “Welcome screen”, “Choose Options screen” etc.). Pages are then specified in detail by zooming into the page shape (2), which leads to the “UI Design Level”. Consequently, the Storyboard Level resembles the transition between conceptual modeling (3) and actual interface design (2). Switching between modeling and design therefore takes two zooming operations: Zooming out to the storyboard map, zooming back in on models or pages. The Storyboard Level aims on effectively narrowing the gap between these inherently distinct tasks during design processes. When comparing this transition (problem space to solution space) to currently employed solutions in available tools or even tool transitions, we think that the Storyboard Layer represents a major contribution to design practice.

#### Modeling Level

Figure 5.26 shows the Modeling Level the user enters when zooming into a modeling container within the Storyboard Map. The Modeling Level is initiated by a Modeling Map (1) that allows creating a number of models that are frequently used in practice. While some models originate from “Usage-centered Design”, other models originate from Agile Modeling or UML. Interconnections and relations between models are again visualized by connection lines. For example, a Persona might be related to an actor within a Role Map, while actors within Role Maps might also be connected to a Use Case Diagram. Use Cases then have specific relations to Task Maps and so on. For a detailed description of the employed



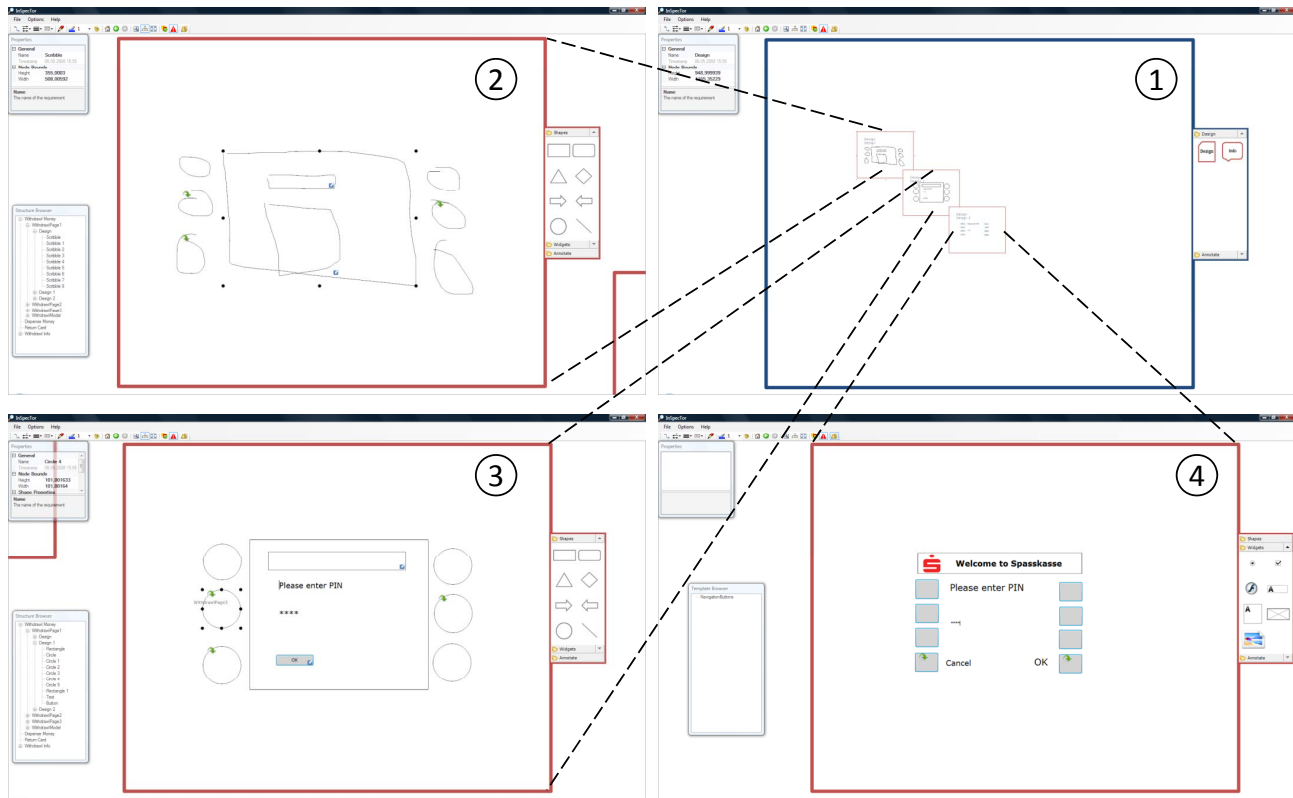


**Figure 5.26:** Modeling Level - Modeling Map (top), Persona model (bottom-left), task map model (bottom-right) (see C—“Screenshots”, Figure C.5 for a full scale version)

models and their interconnections see Rinn (2008). Therefore, the Modeling Level aims on exploiting the interconnections between artifacts within conceptual modeling. Again, transitions between tools and their artifacts are minimized. Modeling artifacts are specified by zooming into the modeling shapes. Again, depending on the purpose of the model, different tools are offered. Figure 5.26 (2) shows a Persona model, which offers a predefined template for persona modeling. Therefore, a picture can be chosen by double-clicking a placeholder image and text can be entered in an integrated editor that also allows to format text. Diagramming tasks, like a Task Map (3) are simply facilitated by shapes, text fields and connecting lines.

Eventually, the user will enter the UI Design Level by zooming into a page shape within the Storyboard Map. Within this page container, page instances can be created and aligned to a UI Design Map that represents iterative or alternative designs. Figure 5.27 (1) shows three overlapping page designs. By overlapping the designs, the iterative nature of designs is revealed. Vertical order of page designs can be manipulated by previously described tools in context menus. In contrast, placing UI designs next to each other marks them as alternative design versions. By zooming into the page shapes, the user can specify different versions of this distinct UI state. Figure 5.27 (2) shows an early, sketched interface design of an ATM welcome screen, while (3) shows an abstract version of the same design employing shapes and typed text. Eventually, (4) presents a medium fidelity design that contains actual buttons, text, shapes and pictures. Other available interface components include a selection of popular widgets, text fields, pictures, wireframes and even Macromedia Flash support for interactive content and animations. Consequently, the UI Design Level offers various means of expression that are necessary to efficiently support creativity and innovation in design. While informal means of expression, like sketching facilitate a rapid exploration of design alternatives, more formal means of expression, over shapes and pictures up to actual real interface components provide effective means for narrowing the solution space. As a result, we think that the UI Design Level bridges several work transitions that are found in the design process, like whiteboard to diagramming or diagramming to GUI Builder. By

UI Design Level



**Figure 5.27:** UI Design Level - UI Design Map (top-right), abstract UI design (top-left), abstract sketched UI design (bottom-left), medium-fidelity UI design (bottom-right) (see C—“Screenshots”, Figure C.6 for a full scale version)

integrating these design steps into a single UI Design Map, the development of design alternatives and iterative designs is always traceable and contributes to a comprehensive design rationale. We also believe that the overview on created designs may efficiently support decision making over alternative designs.

**Detail & Context Components**

Supporting navigation and context awareness

INSPECTOR employs a range of detail & context interface components that are used to support efficient navigation. By providing overview and structure, the user is always aware of the context in which he navigates. Additionally, detailed properties of artifacts can be displayed and manipulated with an inspector window. All these tools are integrated in optional floating windows that can be activated and deactivated on demand. In creative sessions and agile drawings, these tools might not be useful and hinder the interaction flow. Therefore, they can be toggled and resized on demand to avoid interference with the canvas contents. Their translucent appearance allows maintaining a view on the canvas while navigating or designing.

Three optional floating windows provide context & detail

Figure 5.28 shows three floating windows that are integrated as non-modal dialogs: Property inspector (left), structure browser (middle) and overview window (right). The properties inspector displays properties of artifacts that are currently selected on the zoomable canvas in a browsable property sheet that is similar to the property dialog in Interface Designers or GUI Builders. The user can then manipulate these properties by changing values that are displayed in the property sheet. Manipulation of name properties, sizes and other items is instantly applied to the corresponding object on the canvas. In contrast to modal dialogs that are invoked via context menus, this methodology does not interrupt the interaction flow. The structure browser, displayed in Figure 5.28 (middle) visualized the nested tree

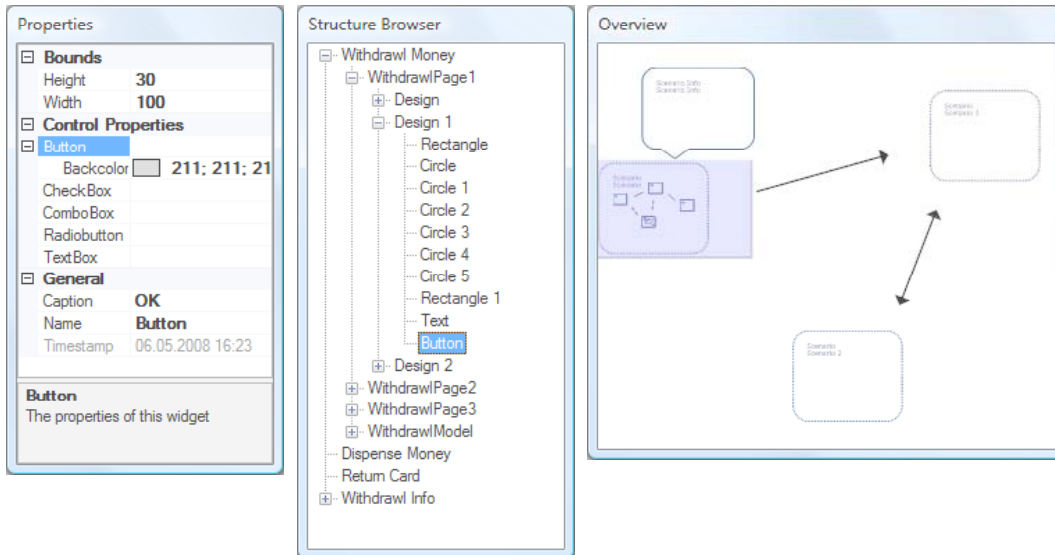


Figure 5.28: Optional views for Context & Detail - properties dialog (left), Structure Browser (middle), Overview Window (right)

structure on the zoomable canvas in a tree control. As the notation is explored, the tree is automatically expanded to display currently viewed artifacts. The structure browser therefore offers overview on the overall structure, while also providing selection and navigation functionality. Single-clicking selects artifacts on the canvas and double-clicking artifacts within the tree structure navigates the zoomable view to the corresponding container that inherits it. Therefore, it resembles a powerful navigation aid. Similar navigation aid concepts were introduced in Chapter 5.2.2—“Zoomable User Interfaces”. Eventually, the overview window allows gaining overview on the zoomable canvas and the currently displayed viewport while navigating. A blue rectangle marks the current view and moving it translates the viewport accordingly, which corresponds to manual panning operations. Overall, INSPECTOR’s detail & context components provide efficient means of navigation in multiscale environments by maintaining context awareness.

Linking & Tracing Artifacts

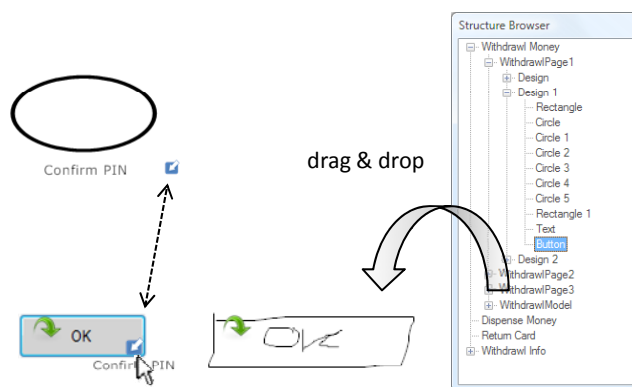


Figure 5.29: Tracing & Linking Artifacts

While artifacts can be arranged based on spatial affinity and connected by lines, some artifacts may relate to other artifacts that are widespread on the canvas in terms of spatial distance and therefore cannot be connected directly. Therefore, INSPECTOR employs links

Supporting traceability and hyperzoom navigation

and references to allow tracing these artifact relations (see Figure 5.29) that can be regarded as “hyperzoom links”. For example, a Use Case may relate to a button on a specific page design. References that can be attached to both artifacts in form of small icons allow instant zoom navigation between them. Similarly, the transition between certain subsequent pages within the UI Design Level is visualized with connecting lines on the storyboard map, but the action that specifies the transition is not. The links and references are created by simply dragging & dropping an artifact from the structure view onto an artifact on the canvas. INSPECTOR utilizes two small icons that facilitate two different linking types. A small green arrow visualizes links that represent transitions to other pages. A small blue reference icon is utilized to display references to other artifacts. References may have multiple destinations and artifacts are always cross-referenced for the sake of consistency. The destination of the reference or link is displayed as the user moves the mouse over these icons. Double-clicking then navigates to the destination. In the case of a reference, zooming is smoothly animated and temporarily zooms back to maintain overview, while navigation with links results in an instant change of view to simulate hypertext behavior. Principally, all artifacts in INSPECTOR can be traced by references, but only page artifacts have linking properties. As artifacts are linked or referenced, connecting lines are automatically created if possible and if they do not already exist between the artifacts. For example, if two pages are linked by a button, the link becomes also visible in the Storyboard Map. If a Use Case references a button, a line between its modeling container and the corresponding page is added automatically.

Linking facilitates simple prototyping

By employing linking and tracing throughout the specification, even the most difficult relations between artifacts can be revealed. The visualization of, references and links as well as connecting lines at the same time reveals overall relations and helps to integrate a comprehensive design rationale. The linking functionality gives INSPECTOR the ability to prototype UI behavior in a simple manner. As these techniques can be applied to all artifacts that may have different degrees of abstraction, traceability is facilitated throughout all steps of the process.

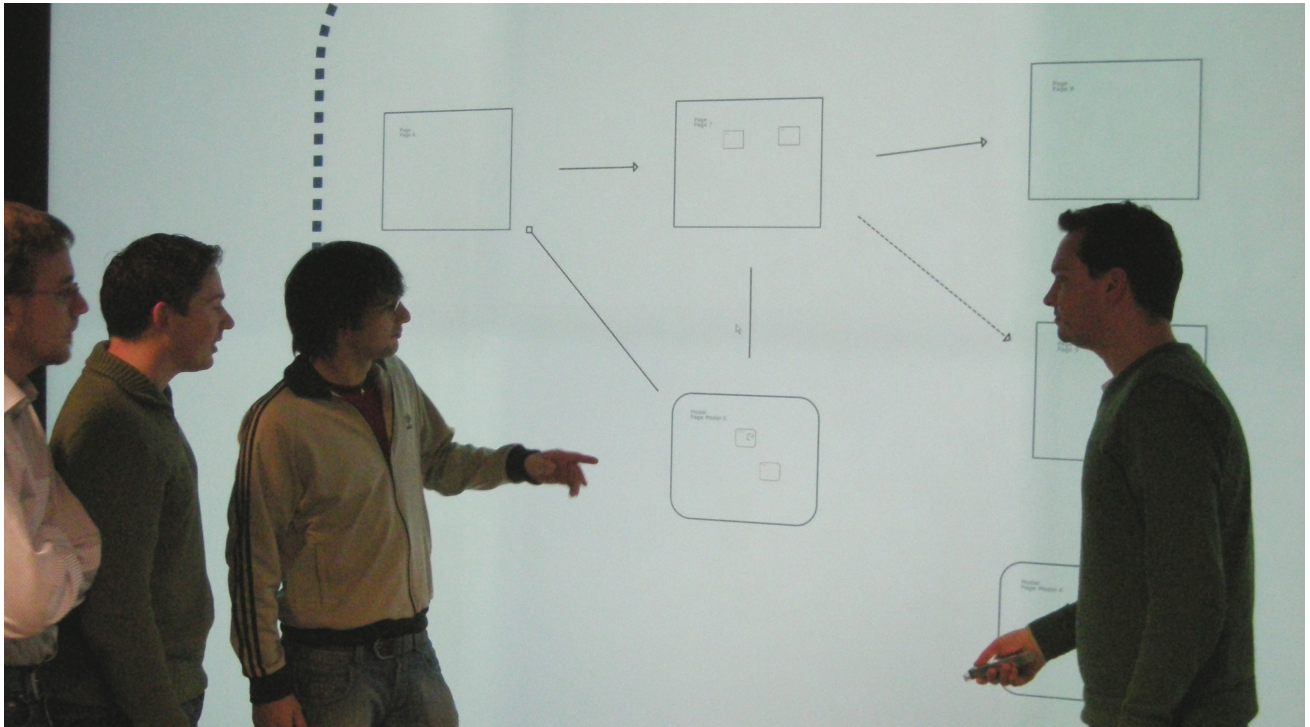
### 5.3.4 Collaboration

Shared workspace, presentation and feedback

In the following, INSPECTOR’s collaboration support is introduced. Based on investigated collaboration support (see Chapter 5.2.4—“Collaboration”), INSPECTOR integrates techniques and methods that facilitate asynchronous collaboration as a shared workspace as well as synchronous collaboration with presentation functionality in meetings. Additionally, feedback components allow placing textual feedback on artifacts, which can be efficiently used in design meetings or in synchronous or asynchronous review sessions. Overall collaboration support is aligned to the design room metaphor, which supports similar means of collaboration (see Chapter 5.1.3—“The Design Room Metaphor”).

Shared workspace

INSPECTOR’s zoomable canvas provides virtually unlimited screen space. Therefore, a large number of artifacts can be created on the canvas. Because INSPECTOR utilizes an XML (.vspec) format for persistent storage of its artifacts, these files can easily be exchanged within a team of designers or can be sent to external actors via email. Additionally, multiple files can be merged and imported into one single canvas. Merging can also be achieved by storing the XML files in a code versioning system (CVS). However, the zoomable canvas within INSPECTOR can be utilized as a shared workspace in order to collaborate in design. For example, the basic structure of the interface design, namely scenario-containers and some models are created during a synchronous brainstorming session. Afterwards, design work may be assigned to different designers that work on separate or similar design issues. After some asynchronous work is done, results can be merged into INSPECTOR’s canvas, which then allows to collaboratively review the design work. The overall process is very similar to behavior in real design rooms. Actors may bring in artifacts into a design room and leave them for others. At the same time, artifacts that were developed by different designers are spatially laid out next to each other for review and decision-making. We think that exactly this feature support in INSPECTOR is effectively promoting collaboration and decision making in meetings throughout the process.



**Figure 5.30:** INSPECTOR displayed on a large, wall-sized display with laserpointer interaction during design meetings

Design meetings are usually conducted in meeting rooms that do not have room for paper-based artifacts (see Chapter 5.2.4—“Collaboration”). INSPECTOR utilizes the room features, presented in Chapter 5.1.3—“The Design Room Metaphor” to cope with these limitations. INSPECTOR, displayed on a large screen (see Figure 5.30) in a meeting room provides necessary presentation needs as well as artifact persistence. The meeting facilitator controls INSPECTOR’s canvas and zooms in to focus on artifacts for discussion. Communication among meeting participants is then conducted via the displayed artifact and by direct verbal communication (see Chapter 5.2.4—“Collaboration”). As the zooming interaction utilizes the human perception for spatial relations, we think that the context of the discussion can be controlled in a simple manner. While common presentation tools only allow displaying artifacts in a sequence, INSPECTOR exploits the interconnections and relations between the varieties of artifacts.

Presentation features

As an additional collaboration support, INSPECTOR integrates sticky notes that can be attached to artifacts on the canvas. These small color-coded textfields can be dropped from the palette onto any artifact on the canvas to leave notes. Available colors mark artifacts as “critical” (red), “noteworthy” (yellow) or as “okay” (green). Sticky notes are then displayed as small icons, but are magnified by a zoom animation as the user activates them by double-clicking. An additional floating window is utilized to gain overview on notes that were left on the canvas. A more detailed description of this feature is provided by König (2008). We think that this simple feedback feature is able to address review and evaluation needs that accompany the design process. Whether notes are attached asynchronously or in meetings, they provide informal means of expressing feedback on specific artifacts within the design space. Again, the sticky note feature is taken from real life situations in design rooms, where sticky notes are efficiently used.

Feedback via Sticky Notes

Eventually, INSPECTOR’s notation can be regarded as an interactive visual specification of interface designs. By respecting the rationale behind the developed interface with interconnections and links, a cross-discipline specification framework and innovative browsing features, we think that it can be used in replacement for a textual specification that strictly employs formal means of expression. INSPECTOR documents can be distributed to external actors as a communication medium that facilitates common understanding in design. Exter-

Specification and UI XML export

nal actors that focus on implementation of the specified interface can browse this interactive tangible specification with INSPECTOR and explore the rationale and functionality that is necessary for implementation. Specification support is additionally supported by an UI export feature that allows exporting final UI designs in separate XML documents. Supported formats are Microsoft's XAML and UsiXML (see Chapter 2.4.3—"Bridging the Gap in Practice"). This feature bridges another tool transition that would be required when employing textual specifications, namely remodeling of the specified interface for implementation in dedicated GUI Designers. Overall, we think that INSPECTOR's specification features exceed those of traditionally used specification tools.

Design Rationale  
support

Finally, INSPECTOR files can also be distributed to new team members as a form of a design rationale. Design knowledge is preserved as artifacts, feedback and interconnections persistently remain on the zoomable canvas. Therefore, INSPECTOR can also be used as a corporate memory that keeps track of developments throughout the process. Decisions and implications for future projects can be traced and browsed by adding sticky notes. Additionally, successful patterns and templates stored in INSPECTOR's template browser can be utilized in future projects. The interactive nature of INSPECTOR's visualization features, like links and references replaces commonly used hypertext methods (see Chapter 2.5—"Treasuring Design Experience") for design rationale specification.



## Chapter 6

# Implementation

*“Most software today is very much like an Egyptian pyramid with millions of bricks piled on top of each other, with no structural integrity, but just done by brute force and thousands of slaves.”*

—Alan Kay

This chapter briefly describes the technical implementation of the presented tool design. Due to the scope of this work, detailed descriptions of implementational aspects and code samples are not provided. Instead, this chapter focuses on the fundamental architecture and system components to convey an overview on the overall implementation efforts. Nevertheless, detailed technical documentation is available in Memmel et al. (2007a) and code documentation. In the following, the basic framework is introduced before the system architecture is described. Thereafter, important system components are briefly described in reference to this architecture. Finally, a summary on implementational issues and limitations regarding system architecture and ZUI framework is presented.

Outlook

### 6.1 Technical Framework

INSPECTOR is based on the .NET Framework and the C# programming language by Microsoft. It was developed using the integrated development environment Visual Studio 2005<sup>1</sup>. The utilized .NET Framework in version 2.0 provides an up to date, proven and comprehensive software platform for development of rich and performance optimized applications for Microsoft Windows. Microsoft .NET includes a runtime environment and a set of class libraries (API). This includes the Windows Forms classes for implementation of user interfaces. Visual Studio 2005 provides a graphical designer for user interfaces (GUI) in addition to advanced programming support. By utilizing external libraries (DLLs) and external interface components (widgets), the basic functionality of Windows Forms can be extended easily. 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 employed using external libraries.

C# & .NET Framework

In order to employ a scalable canvas INSPECTOR utilizes the ZUI Framework Piccolo .NET (Bederson et al., 2004). The Piccolo Framework is added to the programming repository as an external library. Piccolo .NET is maintained by the HCI Group at the University of Maryland which developed several ZUI Frameworks, like Pad++ (Perlin and Fox, 1993), JAZZ (Bederson et al., 2000) and may be seen as the pioneers of “real-life” ZUI development. Piccolo of-

Piccolo .NET ZUI Framework

<sup>1</sup><http://msdn.microsoft.com/VStudio/>

fers a scalable canvas, an object hierarchy, zooming functionality and interaction techniques that perfectly suit technical requirements for INSPECTOR. The construction and manipulation of objects on the canvas is object-oriented. Piccolo follows the well-known scene-graph concept, which is used to hierarchically arrange and manipulate graphical objects. Similar concepts can also be found in three-dimensional graphical environments.

GDI+ support

Piccolo .NET utilizes the .NET GDI+ library within the .NET Framework for graphical presentation of content on the canvas. Offering a wide range of commands, GDI+ enables to efficiently and easily draw and display custom content on the scalable canvas. Utilizing this support, graphics and text could be integrated and scaled in a simple manner. Besides generic graphics, GDI+ smoothly integrates images and vector graphics. With the employment of the WMF/EMF format from Microsoft, which stores vector graphics in GDI+ commands, an efficient representation of vector graphics could be realized. Therefore, vector graphics originating from external sources, like MS PowerPoint or MS Visio could be utilized for content design. Piccolo .NET also facilitates presentation and interaction with controls, or widgets on the canvas. The Piccolo .NET Framework classes utilized within INSPECTOR will only be described briefly in the following. For a deeper understanding of Piccolo and its architecture, we reference the official documentation<sup>2</sup>.

## 6.2 System Architecture

Model View  
Controller &  
Observer Pattern

INSPECTOR utilizes the widespread Model View Controller (MVC)(Goldberg and Robson, 1983) architecture, in combination with the Observer software pattern. The MVC approach is generally used for the separation of content, presentation and logic. A major goal in employing this concept is to enable flexible program design, even when system components are constantly changed or updated. It allows altering or extending visible system components based on interface definitions and therefore facilitates reusability of individual components. In addition, the software pattern is proven in a wide variety of large software projects as it provides overview in code and structure by reducing complexity. The Observer pattern is a design pattern, which originated from the concept of object-oriented programming. It serves as a central control object, which handles and coordinates updates and synchronization between multiple views.

Model View  
Controller

The MVC pattern architecture consists of three system components that depend on each other (see Figure 6.1): The control (controller), the presentation (views) and the data model (model). Depending on the implementation, these components have distinct characteristics. The model object contains the data that the application will display in one or more views, which represent independent presentational forms. The model utilizes neither the controller itself, nor the presentation. It is not aware of how often or in what form it is presented. Changes in some parts of the model may originate from the presentation, which then adjusts the model accordingly. In the context of aspect-oriented programming (AOP) this interaction is usually supplemented by the Observer design pattern (Observer Pattern). The presentation object provides several forms of visualization on the data that is stored in the model. It is not responsible for manipulation of the data, but is primarily passive concerning manipulation operations. The presentation is controlled by the model, and updates accordingly as changes occur. Depending on the implementation, the presentation usually only forwards user input (events) to the control, in order to manipulate data. The control eventually manages user input from the presentation and manipulation of data on the model. The control also updates the presentation and is responsible for synchronizing multiple views as changes occur.

MVC provides  
necessary  
separation and  
structure

As a result, the MVC architecture pattern offers a clear separation of the system in modules and dedicated objects based on their functionality. In object-oriented software development, this pattern is utilized by employing interface definitions for the respective components. The individual interfaces then represent one application tier. Instances of corresponding

<sup>2</sup><http://www.cs.umd.edu/hcil/jazz/learn/piccolo.net/doc-1.2/api/>



interfaces are finally mapped using classes. This formalization of MVC components enables adding or replacing views and models in a simple manner. Because of these advantages the MVC principle is the de-facto standard in the development of large software projects.

The previously mentioned Observer design pattern is a software pattern within the category of behavior. It is utilized to dynamically transfer updates and changes to one or more objects. The pattern is part of the acknowledged GoF (Gang of Four) (Gamma et al., 1995) pattern collection. It is also known as the Publish-Subscribe pattern. The observer pattern provides a mechanism that fits the MVC scenario: One or more components represent the state of an object graphically. As changes in the status of one ore more objects occur, all components are informed of that change. Nevertheless, the individual objects of the components themselves remain independent from the observer. The Observer pattern provides a solution to these issues by using a mechanism that allows components to subscribe and unsubscribe to a central managing object. Any modifications to the observed object will eventually lead to notifications over a previously defined interface. This way, the observer object itself does not need to know anything about his observers. Accordingly, the observers implement a part of the observing object made available through interfaces to react to any occurring changes.

Observer pattern

The subject, also known as publisher defines the observed object. It implements a list of observers, from which it knows that they have certain functionality, based on the defined interface. It then offers methods to allow observers to connect and disconnect, as well as interfaces for the notification of observers (`notify()`). The observer, also called subscriber defines the interface for updates. A concrete subject ultimately saves his condition, notifies all subscribers when changes occur and provides an interface to inquire the current condition. Each observer object has a specific reference to the object that is observed and saves their states. By implementing upgrades to the interface, which may have different demands on interface definitions, the various states are still consistently kept separate. The loose coupling between subject and observer can be varied independently. A dependent object receives the changes automatically. When employing only a number of observers, the Observer Pattern is an effective and simple implementation of multicasts. In combination with the above-described MVC architecture, the Observer Pattern facilitates updates between the model and presentation tiers. In this special case, the model represents the observed object (subject / publisher) and the presentation the observers (subscribers).

Observer facilitates synchronizing of multiple views with data

In the following, the main components of INSPECTOR are presented within the MVC architecture (see Figure 6.1). According to the MVC structure INSPECTOR's components are divided into model, view and controller tiers. This strict separation is applied to all objects except for the main window (Main method), which is located globally above these tiers. The separation is reflected in code by using separate namespaces. The structure of namespaces is also reflected in the folder structure of the source code. All objects are exclusively assigned to one namespace to enhance consistency. If objects refer to external objects from other namespaces they must explicitly define an import statement. Communication between different modules is generalized on unified interfaces. Components within the same namespace are able to directly reference each other. In exceptional cases, the transfer of user input to graphical objects may be communicated directly for the sake of performance.

Main components

The namespace `INSPECTOR.Model` contains the main interface to the model, the model-specific object itself, and all the objects necessary for internal representation of data. These are namely the different types of data, the data structure and an XML interface for import and export of the data model. The namespace `INSPECTOR.Control` provides an interface for data management, the actual controller object, and all related objects that are needed for control tasks. These objects are primarily responsible for user interaction, generation and manipulation of objects and for triggering updates. The namespace `INSPECTOR.View` provides an interface for the presentation tier. It enables to view the actual graphical objects as well as alternative views of these objects. INSPECTOR includes different views: A scalable canvas and their objects, an overview-window, a structure view of the data and a property panel (detail inspector). Since the scalable canvas is compiled from a multitude of different objects, these have a dedicated subordinate namespace `INSPECTOR.View.Nodes`.

Namespaces

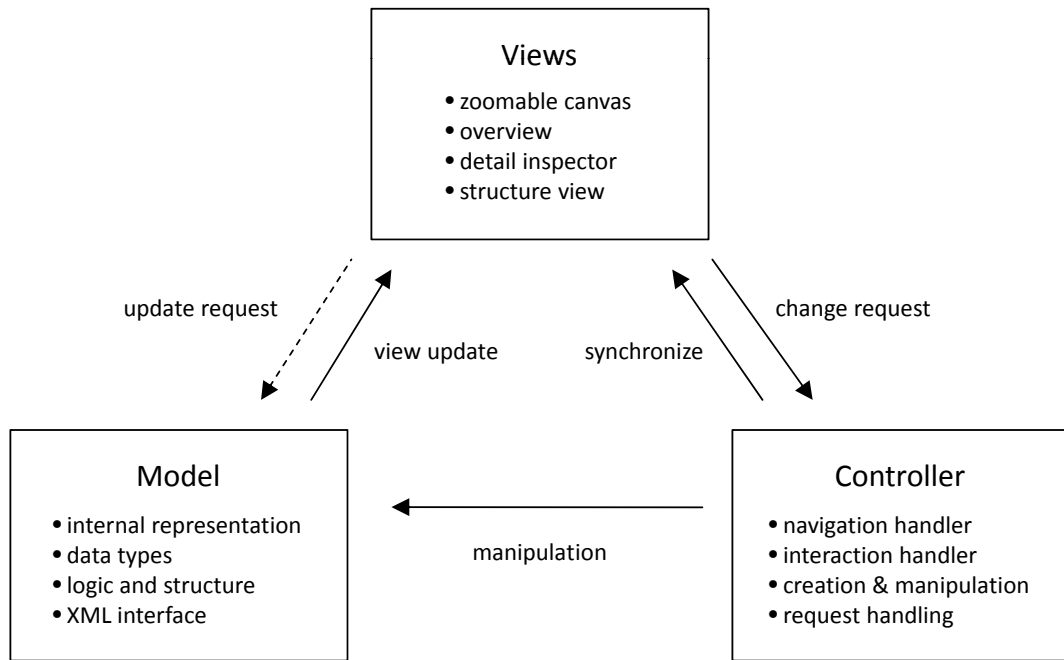


Figure 6.1: Model View Controller Architecture

Relations between components

Figure 6.1 reveals the relations between the main components of INSPECTOR. As top tier, the views are handling visualization and interaction with the user. Requests are then forwarded to the underlying tiers. Simple update-requests are directly sent to the model for the sake of performance while manipulation requests are forwarded to the Controller tier. Actual manipulation of the data is then handled by the Controller object, which is connected to the model for this purpose. As changes within the model occur it automatically updates registered views accordingly. As views are switched or tools are selected, the Controller keeps the different views synchronized regarding aspects that are not related to the data-model but are required for interaction.

### 6.2.1 System Components

Main implementational components

In the following, the most important system components of INSPECTOR, their properties and implementation are briefly described. Therefore, some simple UML class diagrams are presented that reflect the structure. Instead of providing detailed documentation of code, only core classes are introduced. First, the core components of the application are described. Figure 6.2 shows the core classes of INSPECTOR and their relations. These can be divided into three areas: the application core (1), graphical elements of the main window (2) and the MVC mechanism (3).

Core, Controls and MVC

The core of the application forms the static class "Program". This object contains the application process, calls the main window ("INSPECTORMain") and provides access to local resources and property settings. All of these objects are nested in the namespace INSPECTOR. The object "Resources" provides a container for local resources, such as images and icons. The generally defined application settings are centrally managed in the "Settings" object. Both objects can be manipulated in Visual Studio using a graphical editor. The application core utilizes objects that can be clustered according to their functionality, namely "Controls" (2) and "MVC" (3). "Controls" are graphically objects that are used to display tools and menus for interaction. These control objects communicate with the rest of the application through the Controller object. Eventually, the MVC implementation is initialized by the application core. After initializing the Model and the Controller, multiple views are registered to the Model before they are actually displayed. In the following, the characteris-

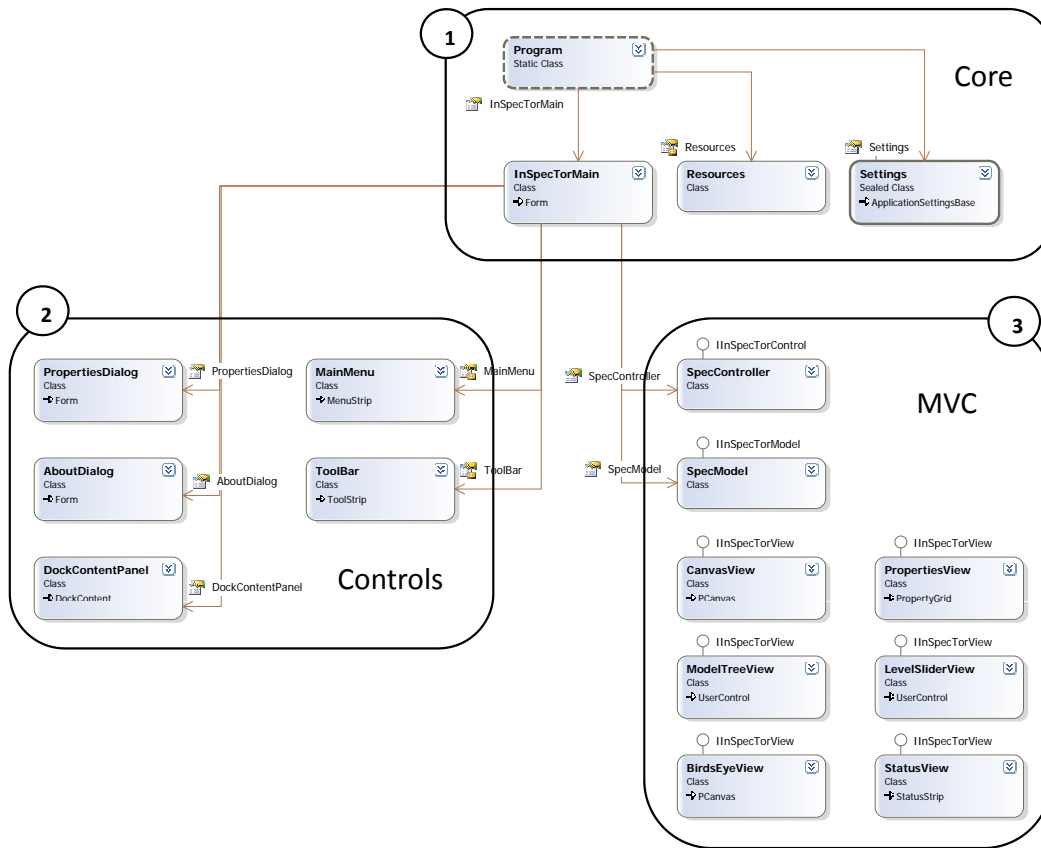


Figure 6.2: System Components

tics of the MVC components are described briefly.

### Data Model

The data model in INSPECTOR is implemented as the specific object "SpecModel" which implements the interface IINSPECTORModel. The data model consists of a hierarchical structure of elements that resemble the INSPECTOR notation and related functions, such as insert, delete and search operations. For each element of the nested tree visualization, there is a representation in the form of a class. Based on the characteristics of the different classes the graphical representations are generated differently for each attached view. Therefore, all elements of the model include the necessary information for representation in all different views in one synchronized object.

Organizing the data and notation

Figure 6.3 shows the object hierarchy within the data model. The root of the model is the class "ProjectInfo". This object represents an instance of an INSPECTOR project. A project therefore contains elements and groups, which in turn contain elements and groups. This structure enables a recursive nesting, like in balanced trees, or searchable tree graphs. Each element of the data structure, except the root "ProjectInfo", is derived from the base class "ElementInfo". This object contains informations that all elements of the notation have in common. Graphical elements of notation, with exception of links ("LinkInfo"), are derived from the base class "NodeInfo". Accordingly, the basic properties for visual presentation in all views are incorporated into this class. The respective specific notation elements, including the groups ("GroupInfo"), are then finally derived from the "NodeInfo" class. Eventually, the hierarchy is divided into three levels: project elements, intermediate property-container and discrete notation elements. The tree is implemented in C# by using ordered lists, known

Object Hierarchy

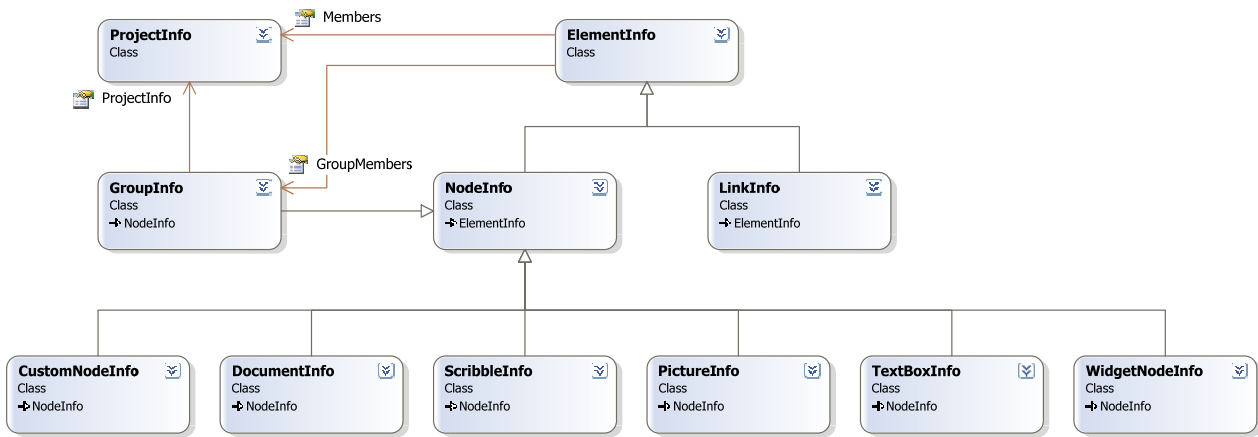


Figure 6.3: Object hierarchy within the data model

as "ArrayLists" in the group objects and the project object.

XML Export

The entire data structure can be exported in an XML file. Likewise, a recovery of the data structure is provided from such a XML file. This feature allows the storage, transfer and the sharing of INSPECTOR projects. This functionality was implemented in C# by utilizing serialization attributes ("Iserializable") interfaces. Each element of the data structure tree therefore contains corresponding attributes. Additionally, a base class has to specify possible subclasses for a successful serialization. Serialization (export) and deserialization (import) of the tree, starting from the root "ProjectInfo", is then realized with the "XmlSerializer" object of the .NET Framework. Due to the continuous use of objects with corresponding serializable attributes ("Iserializable"), as well as compatible ArrayLists, the entire data model is automatically converted into an XML representation.

Primary Presentation

Presenting the notation

The presentations (views) provide visualizations of the data model and facilitate user interactions. In addition to the primary presentation, where the INSPECTOR notation is visualized on a scalable canvas, there are a number of secondary presentations. Each presentation in INSPECTOR implements the interface "IINSPECTORView" and therefore receives update notifications from the data model and the controller object. The interface defines a number of different notifications that are either updates on added or removed items or overall update requests. For the sake of readability, only the primary presentation is described briefly in the following. Other presentations follow similar concepts and properties are accordingly.

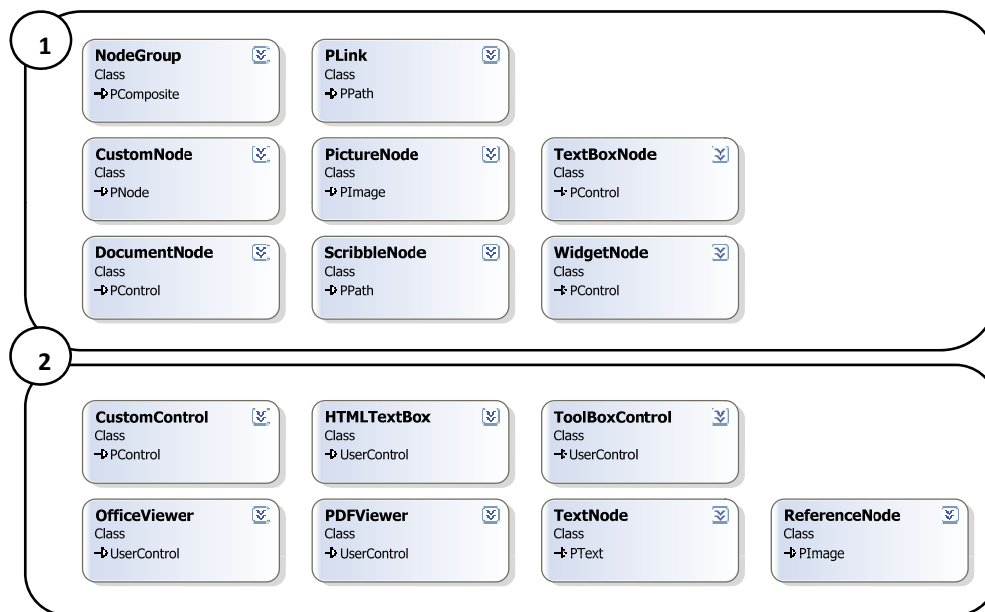
Primary presentation

The scalable canvas, "CanvasView" is the primary presentational interface for visualization and manipulation of the data model. It visualizes the INSPECTOR notation according to the structure of the model in a nested tree visualization. The scalable canvas itself also provides tools available to the user at run time to build and manipulate the graphical notation. Most of this functionality is implemented by utilizing core classes of the Piccolo .NET Framework. In the following, the elements of the notation and auxiliary objects are described briefly.

Visualizing the notation

The notation displayed in the primary presentation is based on the principle of the scenegraph concept, which resembles hierarchically structured interactive graphics. The notation consists of groups ("NodeGroup"), and their grouped objects ("Nodes") as well as the links between these elements ("PLink"). Figure 6.4 shows an overview of the available types of objects that are used to represent the INSPECTOR notation. A number of objects, that are themselves a visual element of the INSPECTOR notation are directly integrated as dedicated objects and support direct user interaction (1). They are also represented in the model as dedicated objects and are therefore generated exclusively by update notifications. Another group of objects provides helper objects for visualization of elements from the first group

(2). Accordingly, elements from the first group are composed from one or more elements of the second group.



**Figure 6.4:** Elements of the INSPECTOR Notation

Figure 6.5 shows the classes that are used for interaction handling and navigation on the canvas. These objects are split into three categories according to their functionality. The navigation on the scalable canvas and interaction handling with notation elements is mainly realized by the “CustomInteractionHandler” object (1). This object controls the selection and scaling of items, the goal-directed zoom, jump-zooming between elements, free panning and zooming with the mouse and the logic behind those operations, like selecting, deselecting and grouping of objects. There are several concrete levels of zoom-magnifications that are employed in order to enable copy & paste as well as linking functionality. As user interactions within the canvas lead to a manipulation of objects, these operations are forwarded to the Controller object that takes care of manipulation within the model. In addition to this central navigation and interaction object, some interaction options are realized by employing events (2). Events are triggered and handled accordingly when specific properties of elements are changed. The occasion of such events is handled by a number of handler classes. Figure 6.5 shows external event handlers for selection, reference-triggering and property changes in the second group. A third group (3) contains auxiliary objects that are used to manipulate notation elements, like a contextual menu, graphical handles and layers.

Interaction &  
Navigation

## Controller

The main task of the controller object is to synchronize user interactions within all views and to manipulate the model based on user requests. The controller contains a list of all presentations and all auxiliary interaction objects to facilitate a common interface to interaction handling. The controller consists only of the actual controller object “SpecController” which implements the interface “IINSPECTORControl”. Available methods are eventually triggered by user actions and lead to manipulation of data objects or to synchronization of the registered views. The controller therefore synchronizes command requests from the views that require manipulation of the data model and forwards them to the data model. Commands of the menus or key combinations, which have an impact on other interactive elements, are synchronized. This redirection is necessary to allow a loose coupling of views in respect to the MVC architecture. Additionally, the controller offers a range of methods

Synchronizing and  
forwarding of  
manipulation  
requests

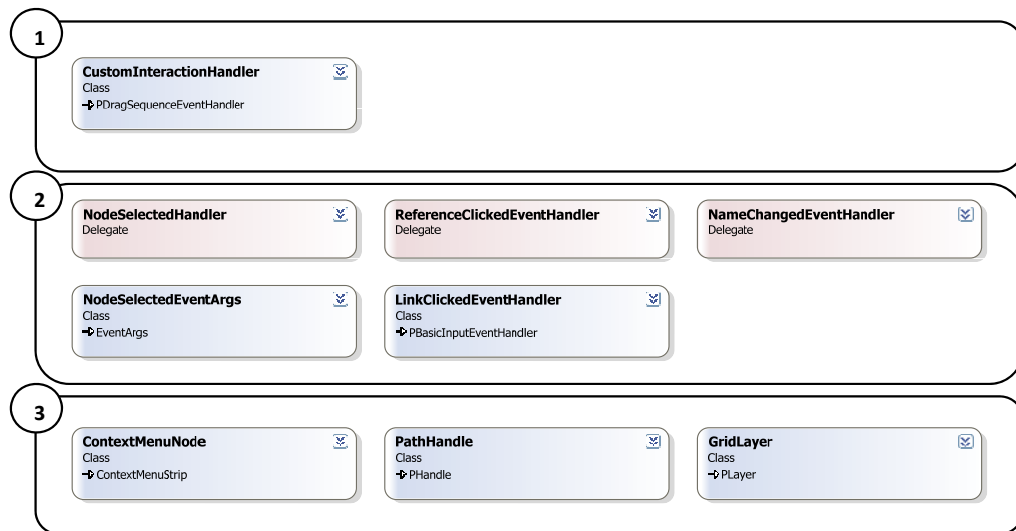


Figure 6.5: Interaction & Navigation Classes

that facilitate the creation of notation elements (factory) and includes a range of methods that allow retrieving the state of system components, like scaling and level position.

### 6.3 Lessons Learned

Implementational Lessons

After describing the architecture and components, we want to conclude our implementational efforts with a summary of experiences that accompanied the implementation. Therefore, our lessons in implementation are divided into architectural lessons and experiences with the ZUI framework *Piccolo .NET*.

Lessons in architecture

When the project started, the initial implementation was experimental and prototypical. Implementation focus was on exploring the possibilities and constrains in design. Therefore, code was unstructured and traceability was awkward. While this early prototyping was necessary to investigate concepts, demands were narrowed towards a general design concept. Because *INSPECTOR* is based on content creation and manipulation, focus shifted on making design artifacts externalizable and persistent. At the same time, the scope and demands for functionality and quality of the interface grew. We quickly realized that a structured architecture is necessary to advance the project in a reasonable manner. Additionally, the characteristics of the development setting, like team members and their focus in development made it necessary to separate system components. Consequently, early prototypical implementations were refactored to fit the popular MVC design pattern and the observer mechanism. Both concepts were proven successful in implementation but refactoring efforts were huge. However, the basic knowledge of patterns, interface concepts and namespaces among team members provided efficient means of communication in meetings. Nevertheless, implementational efforts were increasingly rising as functionality was added. *INSPECTOR* now consists of a complex network of event-based interactions that are heavily influencing each other. As functionality is added, influence on existing implementations of related objects is dramatically. Therefore, adding supplementary functionality leads to a number of required redesigns in existing implementations. Eventually this fact requires team members to understand all parts of the system instead of just a small amount that concerns their objectives. Especially the dynamic navigation on the zoomable canvas, grouping of graphical objects and connections as well as content creation features were revealed as crucial implementational issues. Consequently, constant bug-tracing and testing was mandatory. In retrospect, many issues may not have existed if the overall functionality of *INSPEC-*

TOR was available in the beginning. Instead, constant adding of functionality led to instabilities. Nevertheless, we think that the iterative style of development advanced creativity and innovation in design. In fact, many design concepts did arise from the iterative development of experimental features.

The ZUI framework Piccolo can be regarded as a two-edged sword. On the one hand it offers a strong fundamental and a large number of predefined objects that ease development dramatically, on the other hand, some technical limitations hamper the freedom in ZUI design. Nevertheless, Piccolo is the de-facto leader in real-life ZUI support and without Piccolo, implementation of multiscale interfaces requires extensive knowledge of coordinate systems, transformations and computer graphics. It seems that exactly these features are the focus of Piccolo's architecture. Visualization and navigation in zoomable spaces is efficiently integrated in Piccolo's framework. Even with a large amount of graphical objects, performance of the zoomable canvas in Piccolo .NET is outstanding. The scene-graph structure and available graphical nodes were sufficient for implementation of all notation elements and ease understanding of hierarchies. Manipulation and navigation features of Piccolo are adequate and provide support for most common navigation styles and whiteboard interaction patterns. A large collection of examples supplied with Piccolo helped with early exploration and provided implementation patterns (low ceiling). Even so, graphical components like vector graphics, static text, pictures and shapes are smoothly integrated into the scalable canvas when using the right formats. However, the options for dynamic components are limited. Exactly this is the major limitation in current ZUI implementations. Controls and embedded objects, like PDF viewers, are only active at 100% scale. This means that these controls can only be used when they are visible in full scale and constrains the viewport on the zoomable canvas to a fixed resolution. This limit originates in overall system architecture of today's user interfaces that are usually not scalable rather than the design of Piccolo. As an implication of this drawback, navigation and interaction on the zoomable canvas is hindered. Zooming operations have to make sure that controls are displayed in full scale, which restrains free zooming & panning. When a limited amount of dynamical objects is used this is no major drawback, but in our case, it limited navigation and interaction possibilities dramatically. To cope with some of these issues, goal-directed zooming techniques were promoted. Another drawback related to dynamic content is the manipulation of canvas contents during runtime. Dynamical grouping of notation elements in groups seems to be straightforward in respect to the scene-graph structure, but actual performance is awkward. Additionally, content creation during runtime lacks performance and raises serious issues in multithreading. As of our observations in ZUI implementations (see Chapter 5.2.2—"Zoomable User Interfaces"), practical implementation of dynamical content in multiscale interfaces remains elusive. Most zoomable user interfaces focus on visualization and exploring of existing data spaces rather than using ZUIs as a workspace that allows extensive means of manipulation. From our experiences, this fact is due to the technical limitations in dynamical content of today's ZUI frameworks.

Benefits and  
drawbacks of  
employing Piccolo  
.NET





## Chapter 7

# Conclusion

*“If we knew what we were doing,  
it wouldn’t be called research, would it?”*

—Albert Einstein

In the following conclusion, we present the outcome of our research. Therefore, a review is given that briefly reflects the rationale behind the developed solution, conducted evaluation measures, related developments and a reflection of the final design against initial defined requirements. Thereafter, a proposal of future work is given before this thesis concludes with a brief summary on achieved contributions and remaining issues.

Outlook

### 7.1 Review

After presenting the final design and implementation of INSPECTOR and describing its features and applications, the following review analyzes the process and outcome of our research in respect to the experimental setting in which it was developed. Therefore, results from continuous expert review, field studies and evaluation questionnaires as well as resulting evolution in design are briefly described. Thereafter, a subjective discussion along initially defined requirements is presented.

Analysis of design evolution, tradeoffs and outcomes

#### 7.1.1 Design Rationale

INSPECTOR was designed and implemented at the Human-Computer Interaction workgroup at the University of Konstanz by a team of graduate students and a PhD candidate over a period of 18 months. Initially, the major goal was to develop a prototype of an innovative specification tool that respects interdisciplinary issues and a specification framework. As fundamental to the design concept, current workstyle and employed models were analyzed (Mommel et al., 2007c,e,a,b). Based on experiences from previous projects Mommel et al. (2007f); Mommel and Heilig (2007); Mommel et al. (2007g), where a tool chain based on model-driven development was designed, a major goal was also to bridge remaining work transitions instead of strictly separating process steps during development. Therefore, clear frontiers in communication and specification were desired that reduce transitions to a minimum requirement. We identified that communication is both internal and external. Additionally, we realized that expert tools with high ceiling, but also high threshold are not suitable for important creative expression in design. As the design concept developed, initial requirements were refined but also neglected. The shift of design focus went from formal

Tradeoffs in Design Concept and their rationale

specification to a more informal approach that also respects the actual design process and its iterative nature. HCI concepts were promoted against influences of other disciplines. As actual implementation was initiated, zoomable user interfaces were investigated as visualization concept to allow displaying the extensive number of artifacts but also to provide a visualization of the various degrees of abstraction that accompany the process. After experimenting with various visualizations and interface layouts, the solution space was narrowed. As the interface design matured, additional functionality was implemented incrementally. Development was constantly reviewed based on a two-week schedule in local focus groups. As implementation reached a status that allowed distributing the tool, field studies and expert reviews were conducted in order to gain evaluation feedback on our approach. Results from both evaluation measures were considered in development and advanced the stability of INSPECTOR but also led to improvements in interface design.

### 7.1.2 Expert Evaluation & Field Study

Evaluation  
questionnaires

Evaluation of INSPECTOR was - and still is - conducted by expert reviews in the form of questionnaires. The participants of a first feedback round were UI specification experts from Daimler AG (n=6). Participants were introduced to INSPECTOR by a short textual presentation of its concept, the motivation behind our approach and a short introductory video presentation. Along with these informations, INSPECTOR was distributed to the participants. Therefore, installation packages were compiled to efficiently provide an installation of INSPECTOR via email. Participants were then asked to evaluate the tool and return feedback based on a questionnaire within two weeks. The questionnaire (see D—“Evaluation Questionnaire”), consists of five parts. The first part aims on identifying the activities of participants, commonly used tools and methods, and difficulties that typically accompany the design process. The remaining parts are dedicated to reveal INSPECTOR’s contribution to design practice. Therefore, the following four categories are presented in which participants note their feedback based on a 5-point Likert scale and optional subjective textual comments:

- Application of employed modeling notations and overall understanding of the notation
- Abilities of integrated means of expression for UI design and evaluation
- Assessment of general usability of INSPECTOR’s interface
- General contribution of INSPECTOR to current design practice

Questionnaire  
results and  
implications

Table 7.1 shows the result of a first evaluation round (n=6) with an early version of INSPECTOR. The averaged results revealed positive aspects as well as areas of improvement. Foremost and most important, the general concept of combining modeling and UI design to ease work transitions was highly anticipated (average value 4.83) in practice. Consequently, we think that the general idea of bridging work transitions and its solution is accepted and promising. Overall contribution to existing specification practice was moderate (average value 3.83). Based on comments supplied with this question, we think that this value was due to the immature status of INSPECTOR at the time of this evaluation. As participants were quite satisfied with the means of expression in conceptual modeling (average value 3.79), we identified room for improvement in prototyping capabilities (average value 3.28) as well as overall interaction (average value 3.33). Consequently, we focused on improving zoom interaction, traceability techniques and prototyping functionality, which resulted in cross-referencing functionality with hyperzoom links. Additionally, Sticky Notes were introduced to better support usability evaluation (average value 3.33) in meetings and review sessions. Prototyping capabilities were also extended by implementing copy & paste mechanisms as well as a template repository. We also proposed a contextual layer, which is not yet implemented to improve context & detail features (average value 3.33) and switching between artifacts (average value 3.50). This contextual layer aims on providing a quick

Questionnaire topic	Avg.
Ability to integrate documents and logic with INSPECTOR	3.66
Chance to capture conceptual and schematic ideas	3.83
Support for user, task and interaction modelling	4.00
Possibility to link models and to thereby increase the traceability and transparency	3.66
<b>Text-based and graphical requirements modelling (aggregated)</b>	<b>3.79</b>
Accessibility of the prototyping features	<b>3.16</b>
Provided functionality at the UI design layer	<b>3.40</b>
Applicability of the UI designs for usability evaluations	<b>3.33</b>
Possibility to link UI designs in order to create a simulation	3.25
<b>Overall UI prototyping capabilities (aggregated)</b>	<b>3.28</b>
Chance to get both overview and detail on the zoom-based 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
<b>Overall rating of the interaction with INSPECTOR (aggregated)</b>	<b>3.33</b>
<b>The overall contribution of INSPECTOR to existing UI specification practice</b>	<b>3.83</b>
<b>The improvement of work style through a combination of different models with multi-fidelity UI design</b>	<b>4.83</b>

**Table 7.1:** Average evaluation results from questionnaires (n=6) based on a 5-point Likert scale (Memmel et al., 2008)

temporary view on artifacts overlaid on the screen on demand without the need to navigate to the related container on the canvas. However, based on the feedback, we think that improvements are possible in all areas as well as on the overall notation.

To further evaluate INSPECTOR, we concluded a field study during a lecture project. This field study included three group of students (n=8) that were asked to employ INSPECTOR during an interaction design project for a rear-seat entertainment system. Along this period of three weeks, the participants wrote down experiences with tool usage based on a weekly diary, which was structured into five categories: which type of models were employed in modeling; which additional tools were applied to cope with missing functionality; problems and bugs that occurred during usage; subjective ratings of user experiences and eventually general opinions about the tool approach. The reason for such a diary-based evaluation was that we wanted to investigate how users adapt to the novel interaction style over a longer time period. As INSPECTOR is regarded as an expert tool, we think that this evaluation style fits our needs better than classical usability tests. In weekly meetings, the students also gave presentations using INSPECTOR on a large projector. Therefore, we could investigate INSPECTOR's presentation capabilities in real-life. After the first results of our diary study, we found that artifacts on the canvas were behaving awkward after several hours of usage. We discovered that the grouping mechanism was interfered with some goal-directed zooming operations. Additionally, problems with integration of documents were increasing. Therefore, we immediately investigated the problems and could solve some of them quickly. Student also reported that they preferred real paper & pencil instead of the sketching mechanism provided within INSPECTOR. It was proven that sketching input with pointing and selecting is not efficient. Consequently, we will investigate tablet pen support for early sketching of ideas. Eventually, students were initially not familiar with the employed modeling languages. This issue was then addressed by providing a help feature that supplies information on employed models directly from the palette via hypertext links. Finally, as we concluded the diary study, we found that overall subjective user experience improved during the three weeks of application. While initial ratings on user experience after the first week were averaged with 1.75 points on a 5-point Likert scale, it improved to 3.00 points after the second week. At the end of the study, participants rated user experience with an average of 4.25 points. We found that INSPECTOR was significantly improved ( $F(2,14)=105.00, p;0.001$ ) by our instant fixing measures.

Field Study results

### 7.1.3 Reflection

Reflection of functionality against requirements	When looking back on the initial presented requirements (see Chapter 4.3—“Requirements for Tool Support”), INSPECTOR’s interface features provide characteristics that address most requirements in some extent but also raises doubt on effective support in some areas. Requirements were presented in four categories: process support, artifact support, technical requirements and usability goals. In the following, INSPECTOR’s features are mapped to these requirements to reflect design against requirements.
Process support	Requirements for process support were identified as “Artifact management”, “Creativity & Innovation”, “Specification & Design Rationale”, “Communication & Collaboration”, “Iteration & Alternation” as well as “Adaptability”. We think that INSPECTOR’s notation effectively addresses management, specification and design rationale demands by offering a structured and guided visualization that allows relating and tracing artifacts in context. Additionally integrated features like copy & paste as well as templates improve management capabilities. Nevertheless, the notation provides informal expressions and room for creative thinking at the same time. Additionally, a range of informal to formal means of expression facilitates iterative development. By offering an interdisciplinary selection of models, presentation features and feedback support, collaboration and communication support are smoothly integrated. However, we experienced that the structured containers notation is not instantly understood by all users. We think that this may be due to the “vertically deep” hierarchy that requires zooming in multiple steps. Adaptability is restrained to the use of a predefined selection of different models. Content within palettes is fixed and is not dynamically adaptable to other processes without additional implementational efforts. Therefore, the scope of application is limited to settings, where agile methods are promoted. Informal artifacts also tend to be interpreted differently by actors that have diverse backgrounds. This can be a benefit but also a drawback. Room for interpretation facilitates idea generation but also restrains specification qualities. Specification abilities were also not evaluated in detail. Additional investigation if INSPECTOR’s notation is applicable as an interactive specification document in real-life settings is necessary, just like the interoperability of employed models in practice.
Artifact support	Required functionality in artifact support was identified as “Diagramming”, “Prototyping”, “Sketching”, “Hypertext”, “Word processing” and “Presentation”. As described in the following, INSPECTOR provides limited support for all these artifacts. Integrated forms of expression include shapes, sketches, interface components and hypertext links. Therefore, diagrams and interface designs can be created in different forms of abstraction. However, overall formality of artifacts is restrained in order to facilitate agile ease of use. The guided content creation structures diagrams in containers that can be related afterwards. Simple hypertext features are integrated in order to trace artifacts and to facilitate simple behavioral prototyping. Word processing functionality is integrated in embedded viewer components, while presentation functionality is achieved by navigation features. However, INSPECTOR focuses on informal means of expression. While this is no major drawback in diagramming and sketching, it is in prototyping, word processing and presentation. In corporate presentations for example, informal content might come out awkward, especially when communicating to external clients. However, informal means of expression in presentations is at least critical but may end up fatal in specifications. Prototyping in INSPECTOR can be integrated in early steps as well as in later high-fidelity designs. Nevertheless, the lack of formal specification of a final design leaves room for interpretation. Additionally, INSPECTOR lacks support for fully functional prototypes. Again, implications for adoption in practice depend on the development setting. Finally, word processing and hypertext functionality is not smoothly integrated due to technical limitations. As documents are integrated in embedded viewers, linking and referencing text passages to artifacts on the canvas remains elusive.
Technical requirements	Technical requirements for INSPECTOR were identified as “Accessibility”, “Separation of Concerns”, “Support current Formats and Standards” as well as “Facilitate teamwork and collaboration”. In terms of accessibility and collaboration, INSPECTOR aims at providing a shared workspace that is based on the principle of the design room metaphor for both arti-

fact sharing and communication. Sticky Notes provide simple means of feedback support while communication is integrated with presentational features that allow communication through artifacts during discussions. However, we have discovered that integration of identified design room features for accessibility is not efficiently addressed by INSPECTOR. Because the shared canvas is based on a file structure, merging and comparing asynchronous work requires additional efforts by the user. When referencing the metaphor, this means that actors always have to “bring in” their artifacts into the room before discussion is initiated. Just leaving artifacts for others or exploring artifacts that were elaborated by others is complicated. In addition, collaboration is only provided if actors agree on a basic structure to avoid interferences. Separation of concerns in INSPECTOR is facilitated by communication features and specification export for state-of-the-art UI specification formats (XAML, UsiXML). Yet, due to constant implementational changes within the design level, functionality is currently restrained and practicability has to be proven. Consequently, evaluation of this feature is an open issue.

Achievement of usability goals was a top priority in respect to adoption of the tool in practice. Defined qualitative goals were “Ease of Use”, “Ease of Learning”, “Ease of Navigation” and “Context Awareness”, while quantitative goals were “Relative task efficiency” and “Absolute task efficiency”. Ease of use is listed as top-priority, because it determines user experiences and consequently adoption. While INSPECTOR employs straightforward and ease-to-use direct manipulation features that are familiar to all users, we also think that this simplicity restrains functionality. From results our evaluations we revealed that some users would like to have more detailed means of expression. However, this would complicate interaction and would require additional affordances. Nevertheless, we still think that the ease of use within INSPECTOR contributes to agile and creative design settings. During our field study, we also found that prevailing implementational issues restrain efficient use. In respect to ease of learning, we were investigating and implementing widespread interaction patterns for both whiteboard interaction as well as zoom interaction. Therefore, we think that we respected ease of learning as good as we could in terms of interaction. However, we found that the notation within INSPECTOR and the nested hierarchical structure is not instantly recognized and fully understood by all users. For both ease of navigation and context awareness, we integrated various navigation aids, goal-directed zooming and context & detail components. However, we have not conducted a formal evaluation of their usefulness. We think there is an advantage in task efficiency relative to current workstyle that is accompanied with various transitions. Nevertheless, both relative and absolute task efficiency has to be proven.

Usability goals

## 7.2 Future Work

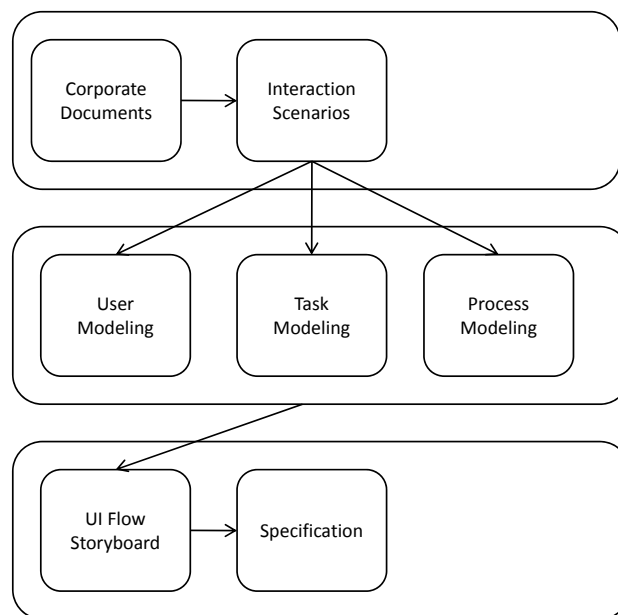
In the following, we describe implications of our research for future development. Based on feedback and evaluation results, we think that overall performance of INSPECTOR needs improvement. Usability studies and formal tests on task efficiency are currently not applicable due to remaining implementation issues. Therefore, fixing remaining problems is top-priority before adding more features. As we are confident in the innovative fundamental to our design, we aim on promoting it further to make it more usable and adoptable in practice. Consequently, we propose following future improvements and redesigns that regard the notation, navigation & contextual aids, general artifact support, specification tasks as well as collaboration support. Eventually, a broader view on INSPECTOR’s interaction and navigation concept is provided by investigating application to other domains.

General improvements and specific redesigns

We discovered that INSPECTOR’s notation is not instantly recognized and understood by some users. Therefore, a redesign of its structure and visualization has to be considered. As the notation is created entirely from scratch, guidance can possibly be improved by providing a predefined template to start with. Due to the deep alignment of the hierarchical containers in terms of scale, exploration by zooming takes several steps. Additionally, artifacts that are deeply embedded into the structure, like UI designs are impossible to recognize

Notation redesign

in overview position. We think that a spatial layout of the notation is easier to understand and is capable of utilizing screen space in a more effective way. It would harness the power of overview capabilities and spatial relations in a better way, as all artifacts are then visible from overview position. Nevertheless, the feeling of diving into the specification from abstract to detail would be replaced to moving through the process by panning instead of zooming. However, a combination of both redesigns could be realized by presenting the user with a predefined process model with horizontally aligned containers that are already displayed on the canvas as the user starts the application (see Figure 7.1). Widespread process models (see Chapter 4.1—“Structured Approaches to User Interface Design”) usually provide spatial layouts of their artifacts that can be utilized. Zooming into the spatially aligned containers would then allow specifying the different process steps. Overview position is subsequently utilized to gain insight into both process progress and artifacts. A predefined spatial template would also allow adapting the notation to various purposes. In combination with dynamic palettes, a choice of process model and utilized artifacts would be possible. We also discovered inconsistencies in our notation. For example, personas currently have to be created separately for each scenario but are logically dedicated to the overall design. This inconsistency also applies to some other conceptual models. Therefore, the scenario approach to structure the design process is questionable. As result from our evaluation, we found that interconnections between artifacts are currently not obvious. In addition, the consistence and completeness of the developed models is not instantly visible and is not checked automatically, due to the freedom of choice in modeling. A predefined process model would also allow clarifying these issues.



**Figure 7.1:** Proposed process model template visualization based on spatial arrangement instead on a hierarchy of scales

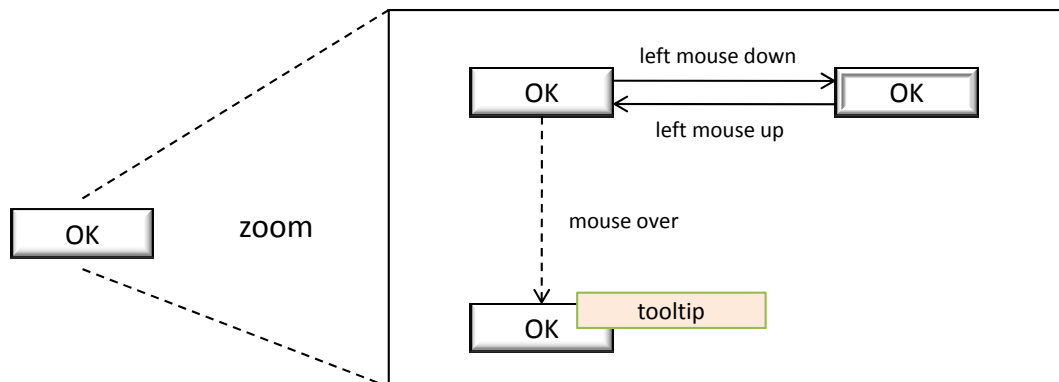
Improve navigation  
and context  
awareness

Navigational aids in INSPECTOR were implemented to avoid getting lost in the zoomable specification space. However, we identified context & detail issues that still prevail. Expert users have stated that switching from design to models is awkward. A first attempt to cope with these issues is being approached by the development of a contextual layer that superimposes containers on the canvas without the need for navigation. However, additional techniques may be considered to improve context awareness. The overview window that is currently employed is not effective when the user navigates into deep structures. The increasingly smaller scale of the viewport visualization on the overview harms perception of the spatial position and navigation functionality. Concepts, like fisheye views, filters, layers or portals may improve navigation and context awareness additionally. Nevertheless, application of these techniques has to be analyzed in respect to this context before considering implementation. In addition, we propose to include searching functionality, as current

functionality is constrained to browsing.

Artifact support within INSPECTOR is extensive. Nevertheless, informal means of expression dominate. In respect to improving prototyping capabilities, we think the additional functionality needs to include expressions that are more formal. For example, grid alignments extended formatting options and design assistants could improve the UI design level. As an early attempt to include such techniques, grids were experimentally implemented as well as a component that assists physical interface design by checking color distances of selected design elements. Document integration in INSPECTOR is currently realized by embedded viewer components. This is merely a tradeoff solution, as the viewers do not allow to link text passages to other artifacts on the canvas. Investigation of alternative means of word-processing integration as well as smooth integration of text-based documents is an open issue. A zoomable appearance of text documents, e.g. spatially aligned pages would provide better visualization features. As an additional improvement to the prototyping capabilities of INSPECTOR, we argue to improve both formality as well as visualization. Functional prototypes may be integrated by employing dynamic components in interface design. For example, zooming into a shape or component on a UI design could display the dynamic states of this object by semantic zooming (see Figure 7.2). Possible visualization for this could be a notation that is similar to a flow chart. After designing the discrete states of an object, connections could visualize the actions that change the state of this object. For example, events like mouse-over, mouse-click etc. then lead to a dynamic change of appearance. This functionality would make designs more dynamic and prototyping more powerful. Due to experiences within our field study, we also found that currently employed sketching functionality is not integrated smoothly. Students were still preferring pen & paper instead of INSPECTOR's sketching techniques. Consequently, we want to investigate more tangible ways of providing sketching functionality supplemented by pen tablets or electronic whiteboards. We also consider extending artifact integration to more media, like videos or speech input for feedback purposes.

Improve artifact support & prototyping



**Figure 7.2:** Specification of dynamic components by semantic zooming - button (left), button states (right)

When looking on the future of specification capabilities within INSPECTOR, we still have to prove feasibility. Exploration of the zoomable structure by actors with different backgrounds in interaction styles may impose new problems. Currently, we regard INSPECTOR as an expert tool. Nevertheless, adoption in practice is essential and should not be constrained by complicated navigation techniques. Therefore, formal usability tests with users that do not have technical backgrounds is still an open issue and has to be investigated. Frontiers of the visual specification, like that in XAML / UsiXML export should be made visible in the overall visualization. Currently, final designs cannot be explicitly distinguished from iterative designs. In addition, the burial of final interface designs into the deep hierarchy makes it hard for novel actors to instantly view the specified interface. Instead, they have to travel through the specification space to "find" the final solution. As mentioned earlier, informal means of expression that are utilized within INSPECTOR still have to be proven in specification practice. However, we think that final designs have to be visually marked to

Improve specification capabilities

make them stand out from the intermediate results. Accordingly, a versioning system could provide effective support.

Improve  
collaboration  
support

Eventually, collaboration features were accompanied with many tradeoffs due to metaphor mapping in favor of the whiteboard metaphor. We therefore think that the concept of a shared workspace needs to be refined in a possible redesign of the data structure of INSPECTOR. Feedback in Sticky Notes should be separated from artifacts, which would allow independent import and export. Additionally, making the data structure available in a database or via web-based access, for example as a browser applet, would improve overall collaboration support. Extending INSPECTOR into a multi-user system would allow marking artifacts that originate from different designers. Visual distinction could be realized by colors or icons. During our experimental setups with large, wall-size displays, we also realized that efficient use of INSPECTOR requires suitable input devices. We therefore propose to investigate voice input, tangible navigation devices and interactive whiteboard support with pen input.

Application to other  
domains

Finally, we think that the tool design we developed is also applicable to other domains than interface design and specification. The combination of the whiteboard metaphor with room features by employing zooming visualizations and simple direct manipulation features may also prove beneficial in other creative domains. As our approach respects principles of creativity frameworks, it is a tool for creating, relating and presenting artifacts that accompany creative processes. Therefore, it can be used as a general brainstorming environment and may replace currently employed text-based brainstorming tools. The hierarchical structure and spatial relations then allow harnessing our perception capabilities to leverage creative thinking by free association. All it takes is to define the content of the employed palettes to a specific domain. For example, the integration of virtually all kinds of media into the zoomable canvas allows collecting and relating information as a general research tool. Eventually, INSPECTOR may also be used as a prototyping tool for ZUIs themselves as content creation can be adapted for this purpose.

### 7.3 Summary

The quest for  
adoption

Within this thesis, an innovative tool design was presented that supports interdisciplinary user interface specification. After presenting the fundamental theory behind the tool approach, related works were investigated. Thereafter, the identified lacks in tool support were reflected in a detailed analysis of structured approaches in practice. During investigation in theory and the following analysis, we found that a new generation of UI tools has to respect the interdisciplinary nature of the design process, gaps in communication and technology as well as specification and prototyping techniques. Nevertheless, creativity and innovation in design should be promoted by supporting informal means of expression. Due to the iterative nature and constant decision making within the design process, we also found that design rationale techniques, the tracing of decisions, collaboration among designers and persistent evaluation is crucial for the success of design projects. In a detailed analysis of work style in practice, we identified the need to bridge a number of work transitions. Thereafter, a conceptual design was developed that respects both analyzed requirements and work style transitions. By utilizing a metaphor, the problem domain was mapped to an interface solution. After investigating concepts for interaction and visualization, a physical design was presented. By carefully designing physical interface components and respecting discipline-specific backgrounds in interaction design, we also aimed at making our novel contribution adoptable in practice. In expert evaluations and field studies, we discovered that the basic idea behind our approach is promising but also identified room for improvements. We recognized a clear demand for a broader scope of UI design tools and received positive feedback on our efforts in bridging work transitions. However, overall performance of our solution does not yet allow conducting formal usability studies and tests. Especially the integration of zoom-navigation and detail & context components requires additional attention. Furthermore, additional formal means of expression were desired by some experts. During a field



study that was settled within a lecture project, we were investigating the collaborative aspects of our solution by testing presentation capabilities and by inquiring user experiences in real-life settings. We also discovered that ease of learning in practice is hampered by misunderstandings of the employed notation. However, we are confident in our contributions and think that further redesigns and improvements may prove that our tool is adoptable in practice. However, we think that adoption is not only determined by our efforts, but also accompanied with a change of minds in practice, or as Larry Constantine puts it:

“Ultimately, the true pace of change is not dictated by the evolution of science or technology or of ideas, but by the capacities of humans and human social systems to accommodate change. 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)



## **Appendix A**

# **Methods and Tools**

<b>Task</b>	<b>Summary</b>	<b>Techniques</b>	<b>Artifacts &amp; tool support</b>
Context of use analysis	Study background information (context) and problem space, which is relevant to the design project.	Stakeholder meeting Observation of work practice Field studies Affinity diagrams	Paper sketches / whiteboard drawings Presentation software (e.g. MS Powerpoint) Video recordings Word processing software (e.g. MS Word) Diagramming software (e.g. MS Visio)
Task analysis	Study required processes and actions in terms of cognitive processes to achieve a task. Organize and structure required tasks.	Stakeholder meeting Task decomposition Task flow diagrams Essential Use Cases Scenarios of Use Object-Action Table	Paper sketches / whiteboard drawings Presentation software Mind mapping software (e.g. MindManager) Diagramming software Spreadsheet software (e.g. MS Excel) Word processing software CASE tools
Stakeholder analysis	List all stakeholders and potential users of the system. Identify human constraints and preferences. Analyze responsibilities and task goals.	User surveys Interviews Focus groups Observation	Questionnaires (e.g. MS Word) Transcript documents (e.g. MS Word) Video recordings
Usability goals	Describe required usability requirements in order to set objectives and priorities in design work. Refer to guidelines.	Usability goal setting Corporate guidelines Standards	Word processing software Portable Documents (e.g. Adobe Acrobat) Hypertext
Idea generation	Generate ideas by considering any suggestion, thus allowing creativity and innovation. Establish a general feel for the solution space.	Group brainstorming Focus groups	Paper sketches / whiteboard drawings Mind mapping software Transcript documents Video recordings
Documentation	Keep track of requirements for later reference.	Requirements specification Style guides	Word processing software Portable Documents Hypertext

**Table A.1:** Tasks, techniques and resulting artifacts within the "Analysis" phase

<b>Task</b>	<b>Summary</b>	<b>Techniques</b>	<b>Artifacts &amp; tool support</b>
Conceptual Design	Create a vision of the solution space. Express abstract vision in models and diagrams. Facilitate creative thinking and innovation.	Personas Scenarios Flowcharts Use Cases Sequence / Activity Diagrams	Word processing software Paper sketches / whiteboard drawings Presentation software Diagramming software CASE tools
Idea generation	Invent metaphors, dialog flow and design ideas by considering any suggestion. Establish a general feeling for the design space.	Group brainstorming Focus groups Storyboards Flowcharts Scenarios	Transcript documents Paper sketches / whiteboard drawings Presentation software Diagramming software CASE tools
Physical Design	Create concrete tangible designs and dialog flows. Experiment with alternative designs and refine the design space by iteration.	Paper prototyping Low fidelity prototypes Wireframes Canonical prototypes High fidelity prototypes	Paper sketches / whiteboard drawings Drawing software (e.g. Adobe Photoshop) Presentation software Dedicated Prototyping software (e.g. iRise)
Specification	Compile design artifacts and solutions into one browseable document. Communicate final design to supplier.	User Interface Specification Visual Specification	Hypertext Word processing software Portable Documents Interactive Prototypes

**Table A.2:** Tasks, techniques and resulting artifacts within the "Design" phase

<b>Task</b>	<b>Summary</b>	<b>Techniques</b>	<b>Artifacts &amp; tool support</b>
Feedback	Collect feedback from all stakeholders. Evaluate design artifacts throughout the process. Make decisions over iterative and alternative designs.	Focus groups Meetings (local / remote) User surveys Interviews	Paper sketches / whiteboard drawings Feedback notes Transcript documents Questionnaires (e.g. MS Word)
Design Rationale	Document all steps that led to the final design for later reference. Include both, positive and negative decisions.	Design Rationale documents Extended Style Guide	Hypertext Word processing software Portable Documents (e.g. Adobe Acrobat)

**Table A.3:** Tasks, techniques and resulting artifacts within the "Evaluation" phase

## **Appendix B**

# **Conceptual Models**

## Bill Jobs

Bill Jobs is a marketing expert and project manager for corporate software development. He has great knowledge and experience in software development process management and software marketing. He studied computer science, but has an additional degree in business administration (MBA). He gained work experience by managing several major projects within his career. He is aware of time-critical and financial aspects in software development and understands the value of good user interfaces from a market perspective. His work experience helps him to manage software projects of different scope and to deal with critical issues that arise throughout the development process.



Bill Jobs now works as a project manager for a major banking company. In his own department, he is responsible for managing software projects and website development. His major responsibilities include: functional specification, corporate identity, transmitting corporate values as part of a comprehensive marketing strategy, personnel planning and communication, financial planning and process management. He is responsible for communicating project goals and to manage project progress by delegating assignments to software developer teams and user interface design teams. He works exclusively with general purpose office software.

During projects, he plans the overall project approach, functionality, constraints, goals and marketing aspects as well as financing and personnel. Recently, based on corporate outsourcing strategy, he increasingly hires external contractors or suppliers for implementation of software applications. Nevertheless, he aims on keeping user interface development in-house, because he thinks that corporate values are a major factor to product success. Therefore, he discusses overall goals of the project in early project phases in brainstorming-like meetings with Donald Nielsen, the project leader of user-interface development. In later development stages he includes software practitioners from a hired supplier company or contractors. He then decides on functionality, financial issues and timeframes for the overall project. On a basis of monthly meetings he communicates with Donald Nielsen and external contractors. He decides whether iterations or alternative ways in development are conforming to corporate strategy and if project budgets meet the timeframe. When a user interface design project is completed, he calls in both parties, designers and software supplier. He negotiates financing of implementation with the external supplier on a basis of a user specification documents. On completion of the project by the supplier, he checks conformity of implementation based on the specification.

**Figure B.1:** Persona 1: Bill Jobs

## Donald Nielsen

Donald Nielsen is an interaction designer and expert on the field of HCI. He has great knowledge and experience in designing user interfaces and general interaction design. He has a background in psychology and ergonomics and an additional degree in information science. He worked as an interaction designer for several companies. Throughout his career he completed numerous projects with different clients and scopes. He is familiar with many different approaches and processes to user interface design. Donald focuses on interdisciplinary design approaches to unleash the potential of creativity and innovation. His work experience helps him to discover good designs and to determine usability flaws.



Donald Nielsen now works as a user interface project manager for a major banking company. His major objective is concluding projects regarding user interactions and interface design. His major responsibilities are: planning and managing requirements analysis, user characteristics, conceptual design, physical design and evaluation. During design projects he is also responsible for a team of designers, which assist him in gathering information and concluding designs. Some projects require the involvement of external stakeholders, like domain experts into the design process. His designers often use dedicated software tools for modeling and design. As he has a good experience in design, colleagues often turn to him regarding difficult design decisions and feedback on experimental designs. Donald Nielsen is proficient in employing advanced prototyping and design tools, he even has some experience with CASE tools.

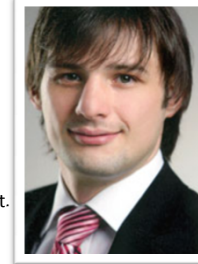
During projects he reports to Bill Jobs, who is responsible for the overall business goals and financing. Goals and constraints of the project are often discussed in brainstorming-like meetings. Furthermore, Donald is planning the overall user interface design approach and distributes jobs to his designers. If some design work is done he evaluates alternative or iterative designs. He then decides which direction design will take and orders if iterations are made. In order to steer development, he communicates with his designers on a basis of weekly meetings but also calls brainstorming sessions to discuss early design decisions or creative tasks. Recently his company is outsourcing actual implementation to a supplier which requires him to work together with external software engineers on the supplier side. He usually compiles artifacts together in a text-based specification and may deliver prototypes to the supplier for communication purposes on concluding a design project.

**Figure B.2:** Persona 2: Donald Nielsen



## Ian Ambler

Ian Ambler is a hard working software engineer and programmer. He has great knowledge and experience in software development as well as programming techniques and languages. He studied computer science, with emphasis on software engineering methods. Ian worked as a programmer for several years before he recently got promoted to project manager. He gained work experience in object oriented software development as well as website development and deployment during his career. He is always up-to date on new technologies and developments within software development. His work experience enables him to efficiently architect and plan software structures, implementations and testing.



Ian now works as a project manager for a medium-sized software company. His main objective is managing project assignments from clients. His major responsibilities are: client communication, project goal setting, software architecture planning, programming and process management. He delegates a team of programmers with software planning tasks, distributes programming assignments, code management jobs and code testing. He usually works with CASE tools, integrated programming environments and GUI Builders as well as general office software for client communication.

Recently his company got a major assignment to complete a software project. As the user interface of this application was already designed by the client the major objective lies in implementing functionality described in specification documents. In early project stages Ian negotiates project assignments and financing with clients on a basis of specification documents. He then plans software architecture, functionality, technical constraints with support of his team. During the development, he communicates project development issues, like progress, timeframes and costs to the client. Usually he decides on implementational priorities and issues for the overall project. He communicates with external clients on a basis of monthly meetings, reports project progress and test results. On completion, he presents the final software to the client by presenting implemented functionality to specified functionality.

**Figure B.3:** Persona 3: Ian Ambler

Category	Features	Description
<b>Expression</b>	Drawings	Actors use pens to create drawings of objects, diagrams or other artifacts
	Artifacts	Not only drawings, but a variety of paper-based artifacts are supported
	Text	Drawings may be annotated with brief textual descriptions
	Connections	Associations between objects are represented as lines
	Abstraction	Drawings may be incomplete and informal
<b>Thinking</b>	Thoughts	Whiteboards show incomplete content, which facilitates thinking
	Relations	Display of multiple objects conveys overview and suggests relations
	Discussion	Incomplete content facilitates discussion among actors
<b>Organization</b>	Spatial organization	Drawings on a whiteboard are organized based on spatial relations
	Clustering	Actors cluster their drawings by segmentation lines
	Erase and Clean	Drawings are discarded by erasing them from the whiteboard
	Reminders	Actors can quickly attach reminders such as sticky-notes
<b>Sharing</b>	Real-time collaboration	Actors notice contributions made by them or others
	Perception	Drawings are abstractions visible to all actors, interpretation is unlocked

**Table B.1:** Features of the whiteboard metaphor, adapted from (Mynatt, 1999, p. 6f) and (Cherubini et al., 2007)



Figure B.4: A Design Room at IDEO (courtesy of IDEO)(Guimbretiere et al., 2001)

Category	Features	Description
<b>Bounded space</b>	Partitioning	Rooms are bounded spaces, separated by walls
	Containment	Rooms contain people, tools and artifacts
	Permeability	Actors enter and leave rooms, bring things in and out
<b>Container</b>	Persistence	Objects left in a room persist over time in the same spatial location
	Customization	Actors customize a room by bringing in artifacts and by arranging them
	Ownership	A room is owned by one or more actors at the same time
<b>Spatial location</b>	Spatial relations	Objects in a room are organized based on spatial relations
	Proximity	Actors can interpret their actions based on how close they are to other's
	Reference and orientation	Actors perceive and reference the room from a common orientation
	Reciprocity	Actors are aware of the fact that others can see their actions
<b>Inhabitation</b>	Presence and awareness	Actors notice contributions made by them or others
	Encounters	Rooms are a meeting place, actors may meet occasionally or on invitation
	Real-time meeting	Meetings may occur in real time with multiple people
	Asynchronous definition	Actors can leave things in the room for others for collaboration

Table B.2: Features of the room metaphor, adapted from (Greenberg and Roseman, 2003, p. 5f)

Category	Requirement	Whiteboard features (category)	Room features (category)
<b>UI Tool Guidelines</b>	Explorability	Thinking	Spatial location
	Expressiveness	Expression	-
	Guidance	-	Spatial location, Inhabitation
<b>Creativity Framework</b>	Collect	Organization	Bounded space, Spatial location
	Relate	Thinking	Bounded space, Spatial location
	Create	Expression	Container
	Donate	Sharing	Container
<b>General UI Design support</b>	Artifact management	Organization	Bounded space
	Creativity & Innovation	Expression, Thinking, Organization	Spatial location
	Specification & Design Rationale	Organization	Container, Spatial location
	Communication & Collaboration	Sharing, Expression	Container, Inhabitation
	Iteration & Alternation	Organization	Container, Spatial location
	Adaptability	Expression	Inhabitation
<b>Artifact support</b>	Diagramming	Expression, Organization	-
	Prototyping	Expression, Organization	-
	Sketching	Expression	-
	Hypertext	-	-
	Word processing	-	-
	Presentation	Expression, Sharing	Inhabitation

**Table B.3:** Features mapping of a combined metaphor to tool requirements



## Appendix C

### Screenshots

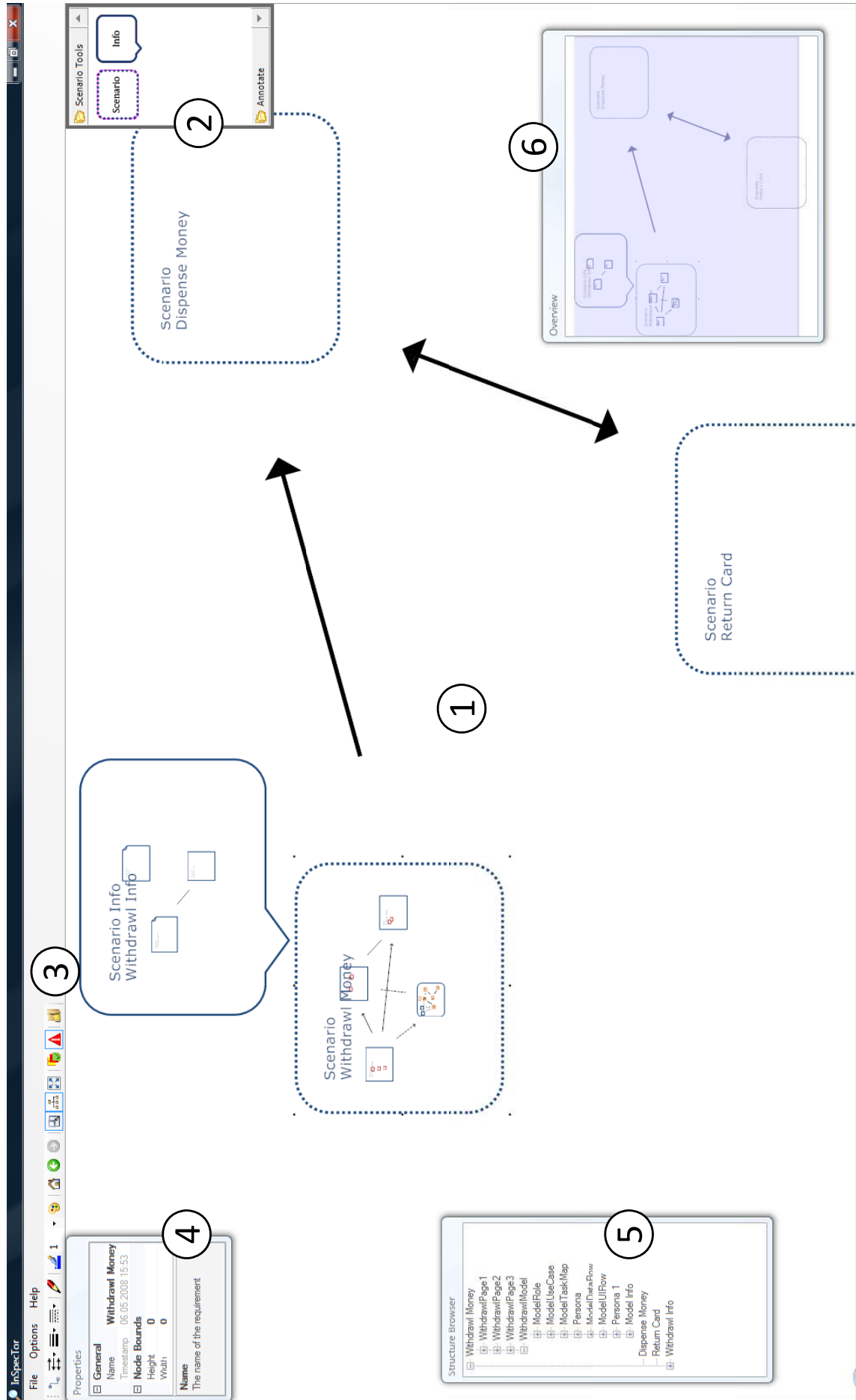


Figure C.1: INSPECTOR's main window

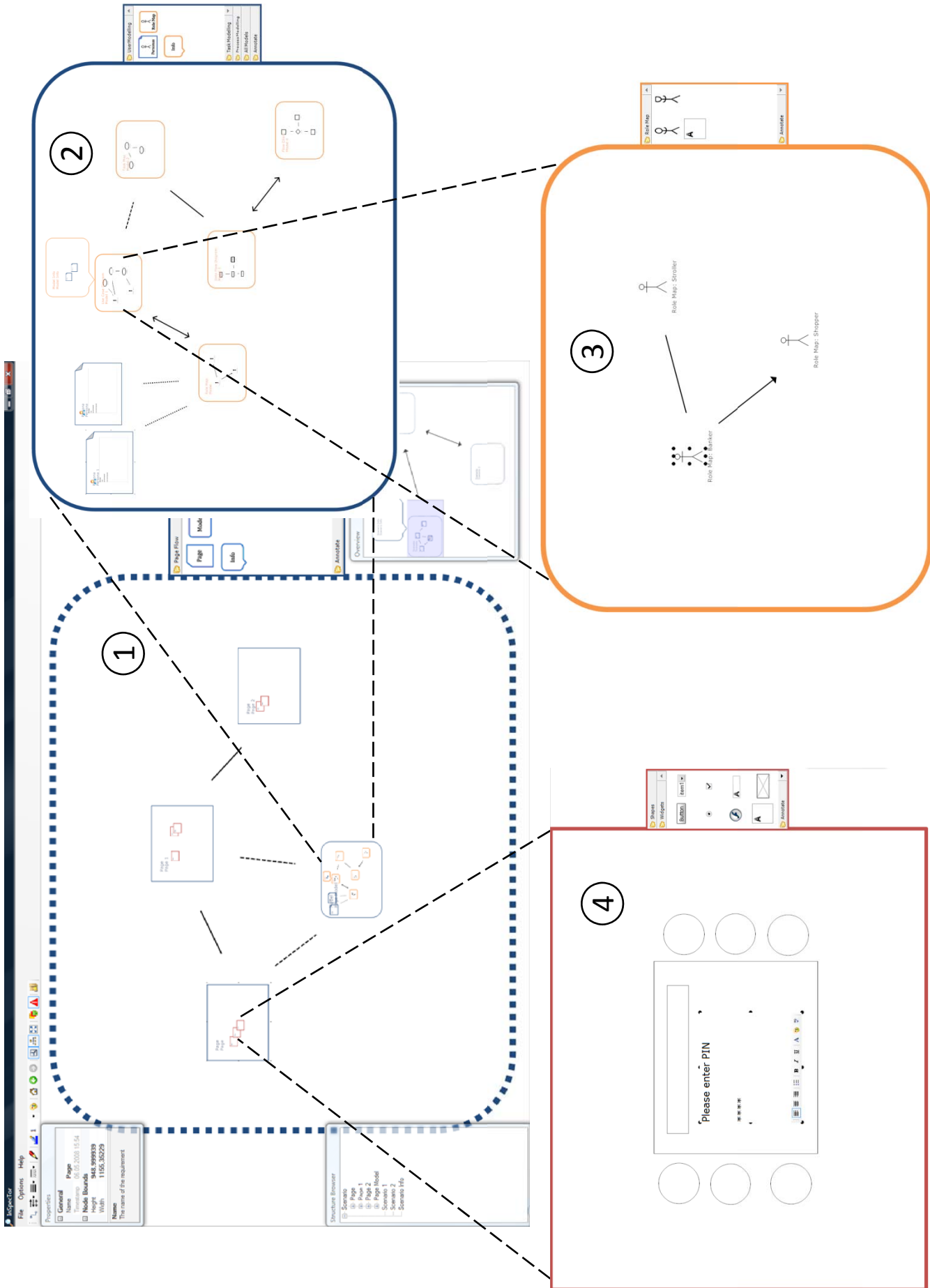


Figure C.2: INSPECTOR's zoomable nested structure (see Appendix A for a full scale version)

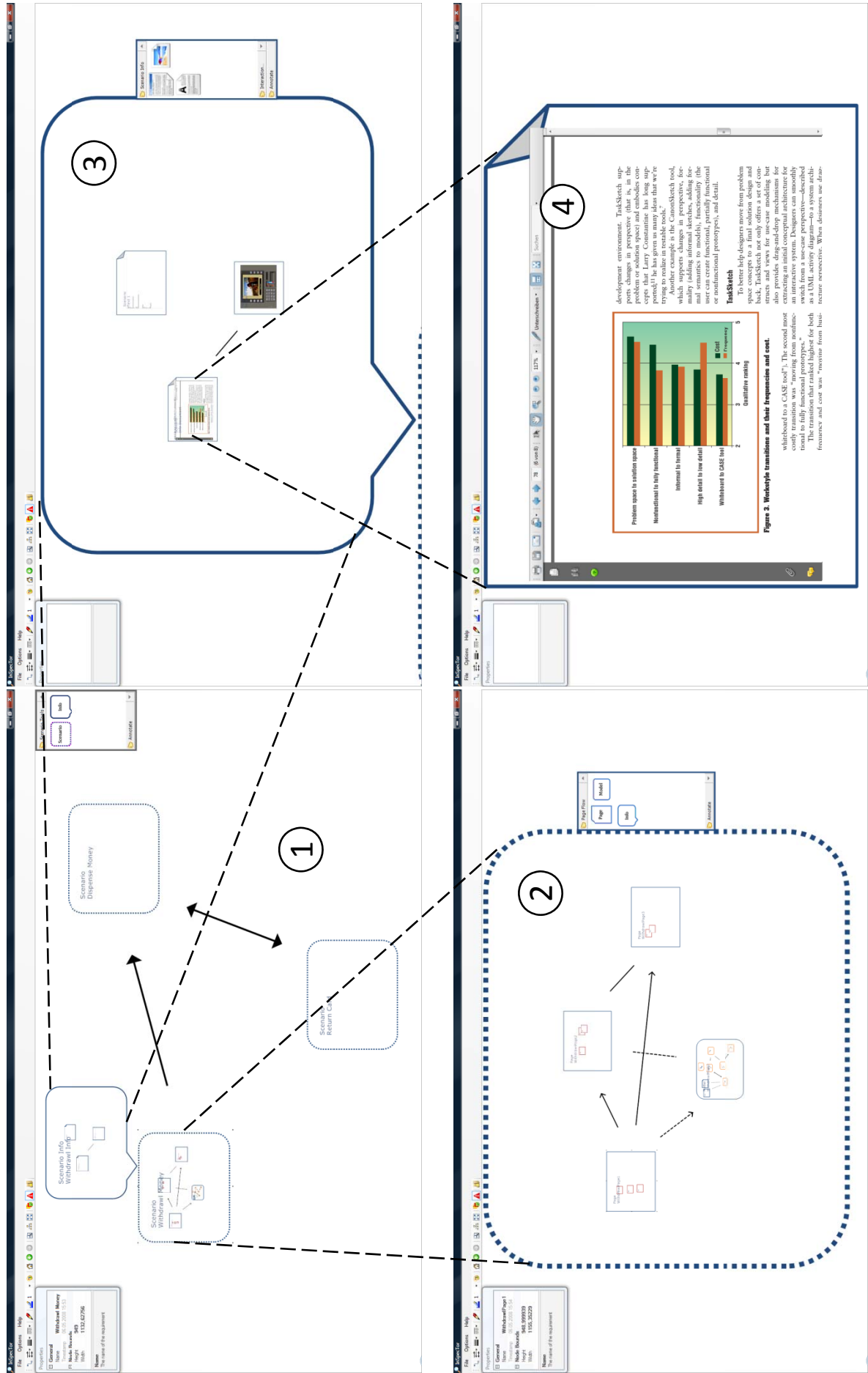


Figure 3.3: Scenario Level - Scenario Map (top-left), Storyboard-Layer (bottom-left), document bubble (top-right), PDF document container (bottom-right) (see Appendix A for a full scale version)



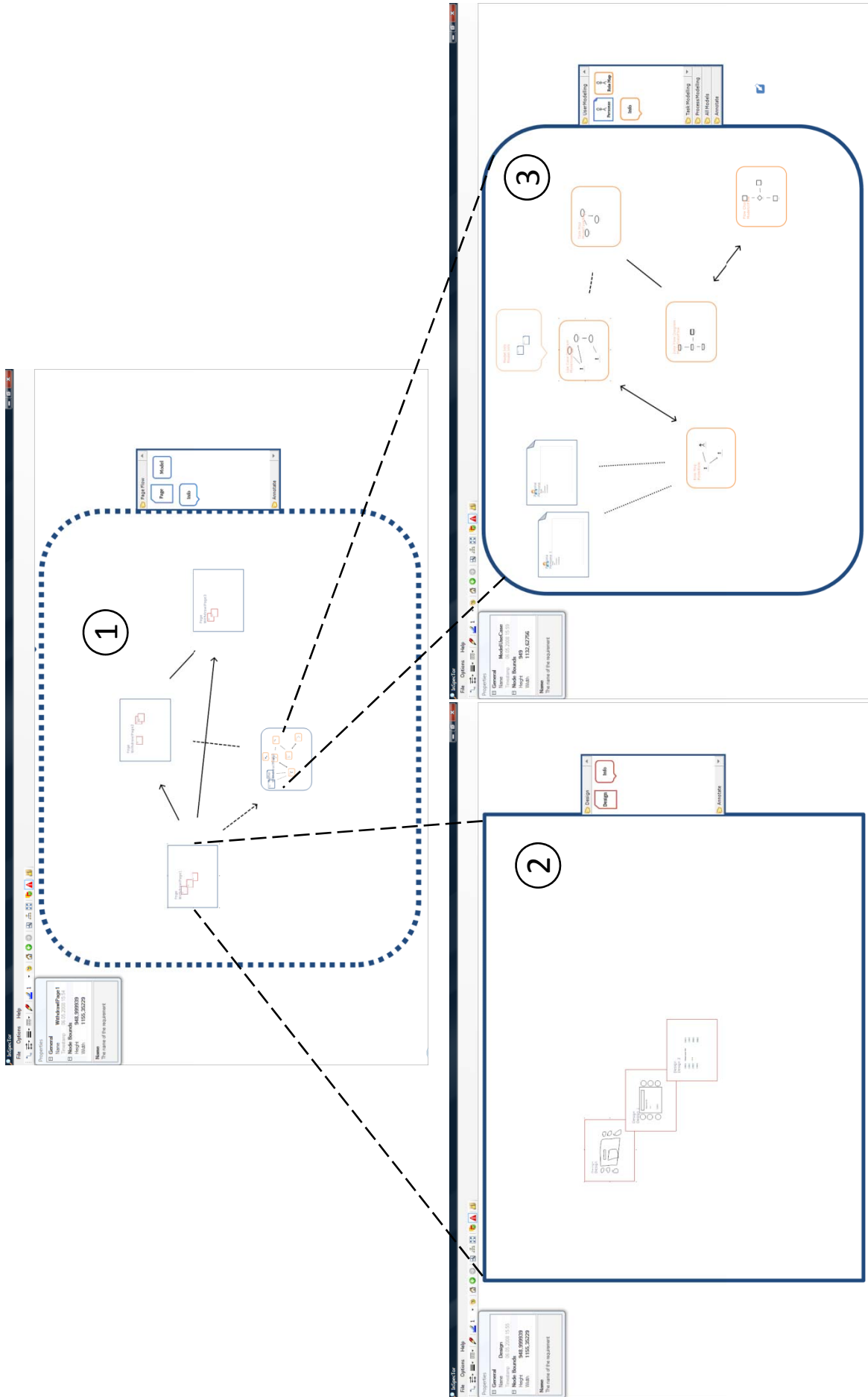


Figure C.4: Storyboard Level - Storyboard Map (top), UI design layer (bottom-left), modeling layer (bottom-right) (see Appendix A for a full scale version)

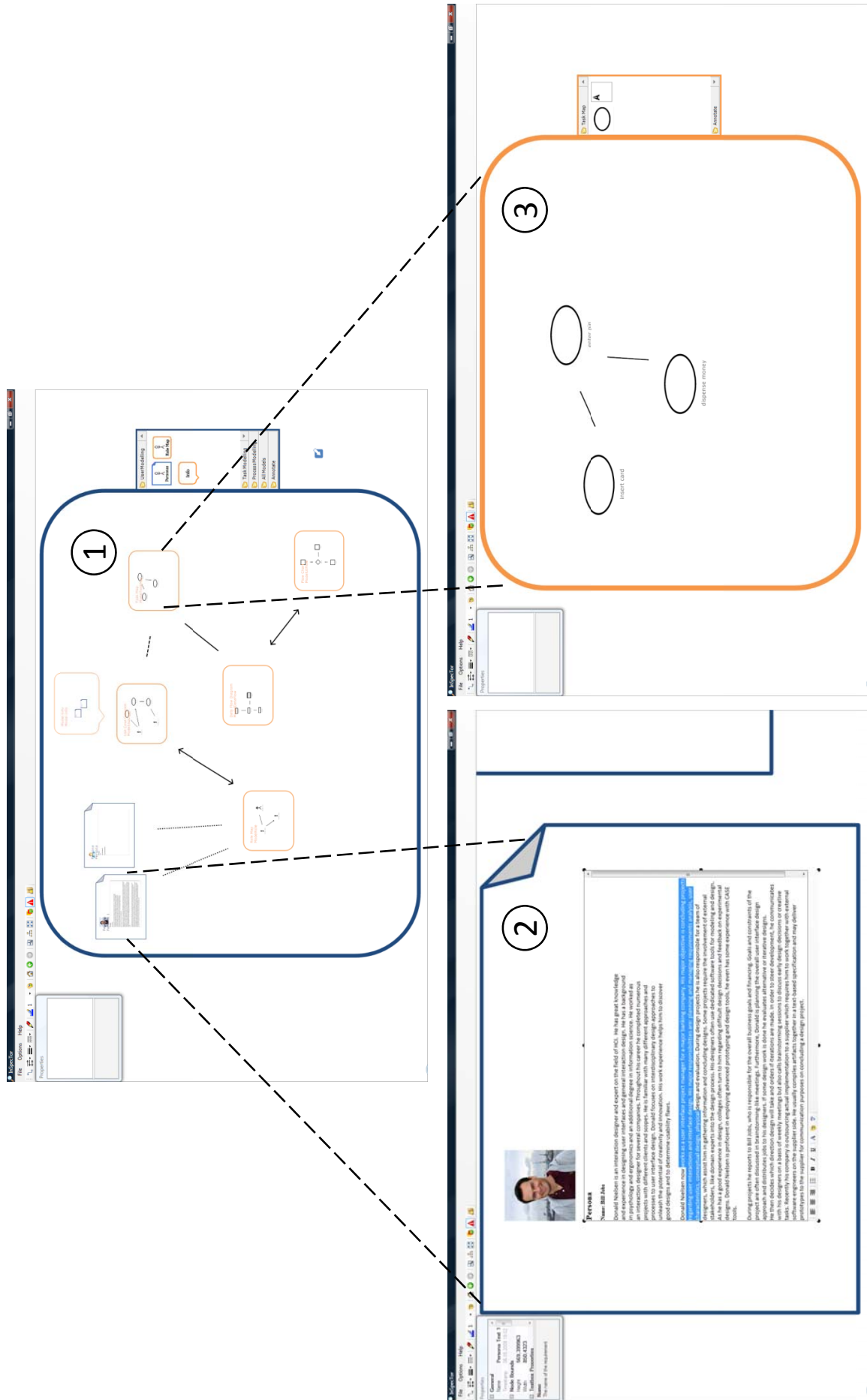


Figure C.5: Modeling Level - Modeling Map (top), Persona model (bottom-left), task map model (bottom-right) (see Appendix A for a full scale version)

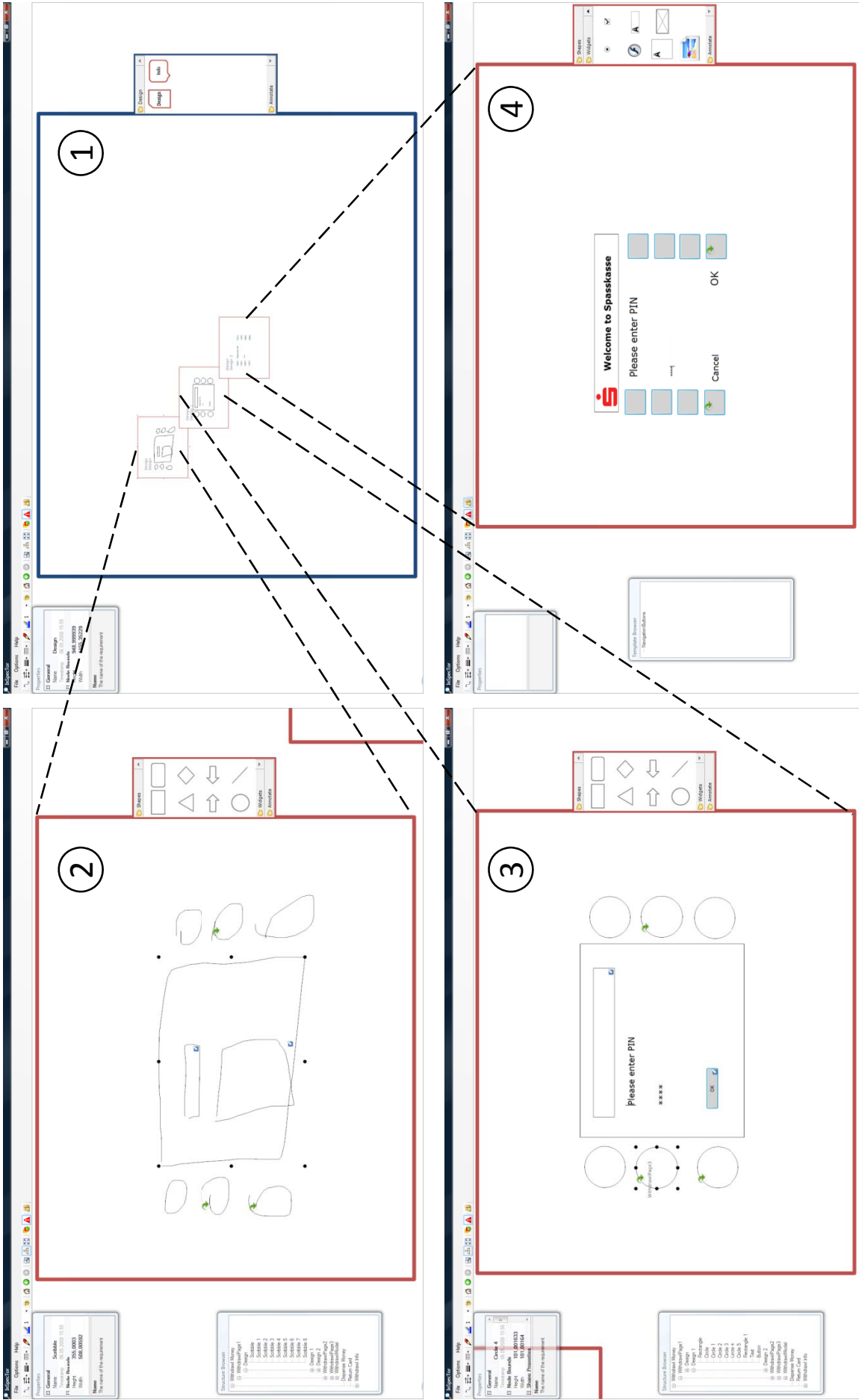


Figure C.6: UI Design Level - UI Design Map (top-right), abstract UI design (top-left), abstract sketched UI design (bottom-left), medium-fidelity UI design (bottom-right) (see Appendix A for a full scale version)



## **Appendix D**

# **Evaluation Questionnaire**



## Fragebogen

### Inspector - Interdisciplinary Specification Tool

#### Die Motivation

In vielen User Interface<sup>1</sup> Entwicklungsprozessen werden vor allem textbasierte Dokumente dazu verwendet, Aussehen und Verhalten von interaktiven Systemen zu beschreiben. Gerade bei der Entwicklung von User Interfaces für komplexe Anwendungsdomänen, sagen Bilder jedoch meist mehr als Tausend Worte. Usability Experten müssen für eine erfolgreiche Spezifikation eines interaktiven Systems in der Regel sowohl grafische Modellierungssprachen verwenden, also auch Prototypen bauen. Der Usability Experte muss mit den ihm zur Verfügung stehenden Mitteln mit Experten anderer Disziplinen kommunizieren und Lösungen erarbeiten (etwa Software Entwickler oder Business Modellierer). Dabei sehen sich viele Usability Experten in einem schwierigen Spannungsfeld zwischen formalen Modellen (etwa UML) und informalen Prototypen (etwa Papierprototypen oder Mockups). Meist stehen Ihnen dabei nur unzureichende Werkzeuge zur Verfügung oder die Werkzeugunterstützung beschränkt sich vor allem auf (zu) späte Phase im Entwicklungsprozess.

#### Das Werkzeug

Aufgrund unserer Beobachtungen haben wir ein experimentelles Werkzeug namens Inspector entwickelt. Das Werkzeug Inspector stellt dem Usability Experten zahlreiche Modellierungssprachen zur Verfügung. Die einzelnen Modelle sind so gewählt, dass die jeweilige Notation auch außerhalb der Disziplin Usability Engineering bekannt ist. Umgekehrt werden die Ausdrucksmöglichkeiten des Usability Experten durch die Agilisierung und Vereinfachung von Modellen erweitert. Für unterschiedliche Aktivitäten werden entsprechende Modelle zur Verfügung gestellt, etwa Personas für die Modellierung von Benutzern, oder Use Case Diagramme zur Modellierung von Benutzeraufgaben. Alle Modelle können untereinander verbunden werden, so dass der Usability Experte die spezifizierten Modelle von abstrakten bis hin zu detaillierten Darstellungen explorieren kann. Modelle werden schließlich mit Prototypen des User Interface verbunden, die ebenfalls mit dem Werkzeug erzeugt werden können. Durch ein Zoom-basiertes Konzept soll das mögliche Eintauchen in die Spezifikation verdeutlicht werden, indem der Usability Experte beispielsweise vom Prototypen aus die zugrundeliegenden Modelle näher betrachtet. Insgesamt entsteht durch diesen Verbund ein interaktiver Speicher für wichtige Dokumente, die ansonsten als Text oder isolierte Bilder in Datenbanken (etwa Doors) eingepflegt werden. Es entsteht so mit eine visuell erlebbare User Interface Spezifikation.

#### Die Umfrage

Im Rahmen dieser Umfrage möchten wir Sie bitten, die Modellierungs- und Prototyping-Funktionalitäten des Werkzeugs Inspector auszuprobieren und zu bewerten. Dabei sollen Sie auch Urteil über die das Zoom-basierte Interaktionskonzept abgeben. Da es sich bei Inspector noch um ein experimentelles Werkzeug handelt, möchten wir Sie bitten Ihre Bewertung vor allem auf Idee und Konzept des Werkzeugs zu fokussieren. Schließlich sind wir vor allem daran interessiert, wie Sie derzeit User Interfaces spezifizieren und ob Sie sich von einem Werkzeug wie Inspector eine Unterstützung vorstellen könnten.

**Für eventuelle Rückfragen ich Ihnen gerne telefonisch oder per E-Mail zur Verfügung**  
Thomas Memmel ([memmel@inf.uni-konstanz.de](mailto:memmel@inf.uni-konstanz.de))

**Wir danken Ihnen für Ihre Mitarbeit bei unserem Forschungsprojekt!**

<sup>1</sup> Benutzungsschnittstelle, Bedienoberfläche



**A. Zu Ihrem Tätigkeitsbereich**

1. Bitte Beschreiben Sie bitte kurz Ihren Aufgabenbereich im User Interface Entwicklungsprozess:

---



---

2. Wie viele Jahre Erfahrung haben Sie in dem von Ihnen genannten Aufgabenbereich?

---

3. Welche Werkzeuge verwenden Sie momentan im User Interface Entwicklungsprozess und warum?  
(Mehrfachnennungen möglich. Falls Ihr Werkzeug nicht gelistet ist, bitte den Produktnamen mit angeben)

Programm		Einsatzbereich, Art der Verwendung	Vor- und Nachteile bei der Verwendung
MS Word	<input type="checkbox"/>		
MS Excel	<input type="checkbox"/>		
MS PowerPoint	<input type="checkbox"/>		
MS Visio	<input type="checkbox"/>		
MindManager	<input type="checkbox"/>		
Bildbearbeitungsprogramme	<input type="checkbox"/>		
Telelogic Doors	<input type="checkbox"/>		
Sonstige (bitte nennen)	<input type="checkbox"/>		

4. Wie schätzen Sie Ihre Kenntnisse im Umgang mit graphischen Modellierungssprachen ein?

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	keine
----------	--------------------------	--------------------------	--------------------------	--------------------------	--------------------------	-------

5. Wie häufig verwenden Sie graphische Notationen in im User Interface Entwicklungsprozess?

Sehr oft	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	nie
----------	--------------------------	--------------------------	--------------------------	--------------------------	--------------------------	-----

6. Wenn Sie graphische Notationen verwenden, zu welchem Zweck verwenden Sie diese?

- Anforderungsermittlung       Formale Darstellung       Darstellung von Beziehungen
- Code-Generierung (etwas aus UML)       Zur Spezifikation von Konzepten

Figure D.2: Evaluation questionnaire, Page 2 of 5



7. Würden Sie graphische Modellierungssprachen verwenden, wenn diese einfach zu gestalten wären?

Ja	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Nein
----	--------------------------	--------------------------	--------------------------	--------------------------	--------------------------	------

8. Wie schätzen Sie Ihre Kenntnisse im Umgang mit Prototyping Werkzeugen ein?

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	keine
----------	--------------------------	--------------------------	--------------------------	--------------------------	--------------------------	-------

9. Wie häufig verwenden Sie Prototypen im User Interface Entwicklungsprozess?

Sehr oft	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	nie
----------	--------------------------	--------------------------	--------------------------	--------------------------	--------------------------	-----

10. Wenn Sie Prototypen verwenden, zu welchem Zweck verwenden Sie diese? Mehrfachnennung mgl.

Anforderungsermittlung     Evaluation     Diskussion von Konzepten     Zur Spezifikation

11. Welche Aufgaben sind im User Interface Entwicklungsprozess am Schwierigsten?

Aufgabe	Sehr einfach	Einfach	Neutral	Schwierig	Sehr schwierig
Dokumentation von Anforderungen	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Pflege von Anforderungen	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Nachvollziehbarkeit von Anforderungen	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Zusammenhänge zwischen Anforderungen und UI Design	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Designentscheidungen speichern	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Umschalten zwischen abstrakten Modellen und visuellen Darstellungen	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

**B. Inspector: Textbasierte und graphische Modellierung**

1. Können wichtige Informationen, Dokumente und Logiken erfasst werden?

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sehr schlecht
Kommentar						

2. Können Sie mit dem Werkzeug konzeptuelle Ideen schnell und schemenhaft erfassen?

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sehr schlecht
Kommentar						

3. Können Sie mit den Modellierungssprachen eine Beschreibung von Benutzern, Aufgaben und Dialogabläufen vornehmen?

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sehr schlecht
Kommentar						

Figure D.3: Evaluation questionnaire, Page 3 of 5



**4. Können Sie die erzeugten Modelle miteinander verbinden und Zusammenhänge besser erfassen?**

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sehr schlecht
Kommentar						

**C. Inspector: User Interface Prototyping**

**1. Ist es Ihnen möglich die Prototyping-Funktionalitäten sinnvoll und effektiv zu integrieren?**

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sehr schlecht
Kommentar						

**2. Ist der Funktionsumfang der Benutzeroberflächengestaltung ausreichend?**

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sehr schlecht
Kommentar						

**3. Können Sie sich vorstellen mittels der erstellten Prototypen Evaluationen durchzuführen?**

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sehr schlecht
Kommentar						

**4. Können Sie unterschiedliche Designs miteinander verbinden, so dass eine Simulation der Dialogabfolge entsteht?**

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sehr schlecht
Kommentar						

**D. Inspector: Interaktion Generell**

**1. Ist es Ihnen einerseits möglich Gesamtüberblick und Detailinformationen einzusehen?**

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sehr schlecht
Kommentar						

**2. Unterstützt Sie die Art und Weise der Interaktion bei der Arbeit mit Modellen und Design?**

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sehr schlecht
Kommentar						

**3. Wird Ihnen durch die Interaktion der Wechsel zwischen Modellen und UI-Design erleichtert?**

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sehr schlecht
Kommentar						

Figure D.4: Evaluation questionnaire, Page 4 of 5



4. Sind die Informationen, die zur Erledigung bei Ihren typischen Aufgaben notwendig sind, auf dem Bildschirm übersichtlich verfügbar?

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sehr schlecht
Kommentar						

5. Individuelle Anmerkungen:

Hier ist Platz für weitere Kritik an dem Werkzeug Inspector oder für Probleme, die Sie bei Beantwortung der Fragen nicht losgeworden sind.

---



---



---



---



---



---



---



---



---



---

**E. Abschließende Bewertung**

1. Wie beurteilen Sie den möglichen Wert eines Einsatzes von Werkzeugen wie Inspector in Ihrem User Interface Entwicklungsprozess?

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sehr schlecht
Kommentar						

2. Halten Sie die Integration von Modellen und UI-Designs in einem gemeinsamen Werkzeug für Usability-Experten für sinnvoll?

Sehr gut	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sehr schlecht
Kommentar						

Figure D.5: Evaluation questionnaire, Page 5 of 5

# Bibliography

- Abowd, G. D. (1991). *Formal descriptions of user interfaces*. In *Theory in Human-Computer Interaction (HCI)*, IEE Colloquium on, pages 7/1–7/3.
- Ahlberg, Christopher and Shneiderman, Ben (1994). *Visual information seeking: tight coupling of dynamic query filters with starfield displays*. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 313–317. ACM Press, New York, NY, USA. ISBN 0897916506. doi:10.1145/191666.191775.
- Ambler, Scott W. and Jeffries, Ron (2002). *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. Wiley. ISBN 0471202827.
- Bartram, Lyn, Ho, Albert, Dill, John, and Henigman, Frank (1995). *The continuous zoom: a constrained fisheye technique for viewing and navigating large information spaces*. In *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 207–215. ACM Press, New York, NY, USA. ISBN 089791709X. doi:10.1145/215585.215977.
- Baudisch, Patrick, Good, Nathaniel, Bellotti, Victoria, and Schraedley, Pamela (2002). *Keeping things in context: a comparative evaluation of focus plus context screens, overviews, and zooming*. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 259–266. ACM Press, New York, NY, USA. ISBN 1581134533. doi:10.1145/503376.503423.
- Bederson, B. B. and Hollan, J. D. (1994). *Pad++: a zooming graphical interface for exploring alternate interface physics*. *Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 17–26.
- Bederson, B. B. and Hollan, J. D. (1995). *Pad++: a zoomable graphical interface system*. *Conference on Human Factors in Computing Systems*, pages 23–24.
- Bederson, B. B., Shneiderman, B. E. N., and Wattenberg, M. (2003). *Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies*. *The Craft of Information Visualization: Readings and Reflections*.
- Bederson, Benjamin B. (2001). *Photomesa: a zoomable image browser using quantum treemaps and bubblemaps*. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 71–80. ACM Press, New York, NY, USA. ISBN 158113438X. doi:10.1145/502348.502359.
- Bederson, Benjamin B. and Boltman, Angela (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, Washington, DC, USA. doi:http://doi.ieeecomputersociety.org/10.1109/INFVIS.1999.801854.
- Bederson, Benjamin B., Meyer, Jon, and Good, Lance (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*, pages 171–180. ACM, New York, NY, USA. ISBN 1581132123. doi:10.1145/354401.354754.
- Bederson, Benjamin B., Grosjean, Jesse, and Meyer, Jon (2004). *Toolkit design for interactive structured graphics*. *IEEE Trans. Softw. Eng.*, 30(8):535–546. ISSN 0098-5589. doi:10.1109/TSE.2004.44.

- Bennett, John L. and Karat, John (1994). *Facilitating effective hci design meetings*. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 198–204. ACM, New York, NY, USA. ISBN 0897916506. doi:10.1145/191666.191743.
- Benyon, David, Bental, Diana, and Green, Thomas (1999). *Conceptual Modeling for User Interface Development (Practitioner Series (Springer-Verlag))*. Springer-Verlag. ISBN 1852330090.
- Benyon, David, Turner, Phil, and Turner, Susan (2005). *Designing Interactive Systems: People, Activities, Contexts, Technologies*. Addison Wesley. ISBN 0321116291.
- Beyer, Hugh and Holtzblatt, Karen (1997). *Contextual Design : A Customer-Centered Approach to Systems Designs (Morgan Kaufmann Series in Interactive Technologies)*. Morgan Kaufmann. ISBN 1558604111.
- Blanch, Renaud and Lecolinet, Éric (2007). *Browsing zoomable treemaps: Structure-aware multi-scale navigation techniques*. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1248–1253.
- Bodker, Keld, Kensing, Finn, and Simonsen, Jesper (2004). *Participatory IT Design: Designing for Business and Workplace Realities*. The MIT Press. ISBN 026202568X.
- Bodker, Susanne and Buur, Jacob (2002). *The design collaboratorium: a place for usability design*. *ACM Trans. Comput.-Hum. Interact.*, 9(2):152–169. ISSN 1073-0516. doi:10.1145/513665.513670.
- Borchers, Jan (2001). *A Pattern Approach to Interaction Design*. John Wiley & Sons. ISBN 0471498289.
- Buxton, Bill (2007). *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann. ISBN 0123740371.
- 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*, 15(3):289–308.
- Campos, P. and Nunes, N. J. (2007). *Practitioner tools and workstyles for user-interface design*. *Software, IEEE*, 24(1):73–80. doi:10.1109/MS.2007.24.
- Campos, Pedro (2005a). *Task-driven tools for requirements engineering*. In *13th International Requirements Engineering Conference (RE'05), Doctoral Consortium*. Paris, France.
- Campos, Pedro (2005b). *User-centered case tools for user-centered information systems*. In *12th Conference on Advanced Information Systems Engineering*. Porto, Portugal.
- Campos, Pedro and Nunes, Nuno (2006). *Principles and practice of work style modeling: Sketching design tools*. In *In Proceedings of HWID 06 - Human-Work Interaction Design, IFIP International Federation for Information Processing*, Springer-Verlag, pages 203–219.
- Campos, Pedro F. and Nunes, Nuno J. (2004). *Canonsketch: A user-centered tool for canonical abstract prototyping*. In *EHCI/DS-VIS*, pages 146–163.
- Card, Stuart K. and Henderson, Austin (1987). *A multiple, virtual-workspace interface to support user task switching*. In *CHI '87: Proceedings of the SIGCHI/GI conference on Human factors in computing systems and graphics interface*, pages 53–59. ACM Press, New York, NY, USA. ISSN 0736-6906. doi:10.1145/29933.30860.
- Card, Stuart K., Mackinlay, Jock, and Shneiderman, Ben (1999). *Readings in Information Visualization : Using Vision to Think (The Morgan Kaufmann Series in Interactive Technologies)*. Morgan Kaufmann. ISBN 1558605339.
- Carroll, J. M. (1997). *Human-computer interaction: Psychology as a science of design*. *Annual Review of Psychology*, 48(1):61–83.
- Carroll, J. M. (2000a). *Five reasons for scenario-based design*. *Interacting with Computers*, 13(1):43–60.

- Carroll, John M. (1991). *Designing Interaction: Psychology at the Human-Computer Interface (Cambridge Series on Human-Computer Interaction)*. Cambridge University Press. ISBN 0521409217.
- Carroll, John M. (2000b). *Making Use: Scenario-Based Design of Human-Computer Interactions*. The MIT Press. ISBN 0262032791.
- Chen, Q., Grundy, J., and Hosking, J. (2003). *An e-whiteboard application to support early design-stage sketching of uml diagrams*. *Human Centric Computing Languages and Environments, 2003. Proceedings. 2003 IEEE Symposium on*, pages 219–226.
- Cherubini, Mauro, Venolia, Gina, Deline, Rob, and Ko, Andrew J. (2007). *Let's go to the whiteboard: how and why software developers use drawings*. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 557–566. ACM Press, New York, NY, USA. ISBN 9781595935939. doi:10.1145/1240624.1240714.
- Cockburn, A., Karlson, A. M. Y., and Bederson, B. B. (2006). *A review of focus and context interfaces*. Technical report, University of Maryland, College Park.
- Constantine, L., Biddle, R., and Noble, J. (2003). *Usage-centered design and software engineering: Models for integration*. *Proceedings, International Conference on Software Engineering, 2003*, pages 3–9.
- Constantine, L. L. (1996). *Usage-centered software engineering: new models, methods, and metrics*. In *Software Engineering: Education and Practice, 1996. Proceedings. International Conference*, pages 2–9. doi:10.1109/SEEP.1996.533974.
- Constantine, Larry L. (2001). *Back to the future*. *Commun. ACM*, 44(3):126–129. ISSN 0001-0782. doi:10.1145/365181.365239.
- Constantine, Larry L. (2003). *Canonical abstract prototypes for abstract visual and interaction design*. *Interactive Systems. Design, Specification, and Verification*, pages 1–15.
- Constantine, Larry L. and Lockwood, Lucy A. D. (1999a). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design (ACM Press)*. Addison-Wesley Professional. ISBN 0201924781.
- Constantine, Larry L. and Lockwood, Lucy A. D. (1999b). *Use cases in task modeling and user interface design*. In *CHI '99: CHI '99 extended abstracts on Human factors in computing systems*, pages 352–352. ACM, New York, NY, USA. ISBN 1581131585. doi:10.1145/632716.632935.
- Constantine, Larry L. and Lockwood, Lucy A. D. (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*, pages 746–747. IEEE Computer Society, Washington, DC, USA. doi:10.1109/52.991331.
- Cooper, Alan, Reimann, Robert, and Cronin, David (2007). *About Face 3: The Essentials of Interaction Design*. Wiley. ISBN 0470084111.
- Coyette, Adrien (2007). *A Methodological Framework for Multi-Fidelity Sketching of User Interfaces*. Ph.D. thesis, Université catholique de Louvain, France.
- Coyette, Adrien, Schimke, Sascha, Vanderdonckt, Jean, and Vielhauer, Claus (2007a). *Trainable sketch recognizer for graphical user interface design*. *Human-Computer Interaction INTERACT 2007*, pages 124–135. doi:10.1007/978-3-540-74796-3\_14.
- Coyette, Adrien, Vanderdonckt, Jean, and Limbourg, Quentin (2007b). *Sketchixml: A design tool for informal user interface rapid prototyping*. *Rapid Integration of Software Engineering Techniques*, pages 160–176. doi:10.1007/978-3-540-71876-5\_11.
- Dix, A. J. (1987). *Formal Methods and Interactive Systems: Principles and Practice*. Ph.D. thesis, University of York.
- Dix, Alan, Finlay, Janet E., Abowd, Gregory D., and Beale, Russell (2003). *Human-Computer Interaction (3rd Edition)*. Prentice Hall. ISBN 0130461091.

- Druin, Allison, Stewart, Jason, Proft, David, Bederson, Ben, and Hollan, Jim (1997). *Kidpad: a design collaboration between children, technologists, and educators*. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 463–470. ACM, New York, NY, USA. ISBN 0897918029. doi:10.1145/258549.258866.
- Elrod, Scott, Bruce, Richard, Gold, Rich, Goldberg, David, Halasz, Frank, Janssen, William, Lee, David, Mccall, Kim, Pedersen, Elin, Pier, Ken, Tang, John, and Welch, Brent (1992). *Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration*. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 599–607. ACM Press, New York, NY, USA. ISBN 0897915135. doi:10.1145/142750.143052.
- Erickson, Thomas (2000). *Lingua francas for design: sacred places and pattern languages*. In *DIS '00: Proceedings of the 3rd conference on Designing interactive systems*, pages 357–368. ACM, New York, NY, USA. ISBN 1581132190. doi:10.1145/347642.347794.
- Foster, Gregg and Stefik, Mark (1986). *Cognoter: theory and practice of a colab-orative tool*. In *CSCW '86: Proceedings of the 1986 ACM conference on Computer-supported cooperative work*, pages 7–15. ACM Press, New York, NY, USA. ISBN 1234567890. doi:10.1145/637069.637072.
- Furnas, G. W. (1986). *Generalized fisheye views*. In *CHI '86: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 16–23. ACM Press, New York, NY, USA. ISSN 0736-6906. doi:10.1145/22627.22342.
- Furnas, George W. (1997). *Effective view navigation*. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 367–374. ACM Press, New York, NY, USA. ISBN 0897918029. doi:10.1145/258549.258800.
- Furnas, George W. (2006). *A fisheye follow-up: further reflections on focus + context*. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 999–1008. ACM Press, New York, NY, USA. ISBN 1595933727. doi:10.1145/1124772.1124921.
- Furnas, George W. and Bederson, Benjamin B. (1995). *Space-scale diagrams: understanding multiscale interfaces*. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 234–241. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. ISBN 0201847051. doi:10.1145/223904.223934.
- Furnas, George W. and Zhang, Xiaolong (1998). *Muse: a multiscale editor*. In *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 107–116. ACM, New York, NY, USA. ISBN 1581130341. doi:10.1145/288392.288579.
- Gamma, Erich, Helm, Richard, Johnson, Ralph, and Vlissides, John (1995). *Design Patterns*. Addison-Wesley Professional. ISBN 0201633612.
- Garrett, Jesse J. (2002). *The Elements of User Experience: User-Centered Design for the Web*. New Riders Press. ISBN 0735712026.
- Gerken, Jens (2006). *Orientierung und Navigation in zoombaren Benutzerschnittstellen unter besonderer Berücksichtigung kognitions-psychologischer Erkenntnisse*. Master's thesis, University of Konstanz.
- Goldberg, A. and Robson, D. (1983). *Smalltalk-80: the language and its implementation*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
- Good, L., Bederson, B. B., and Others (2002a). *Zoomable user interfaces as a medium for slide show presentations*. *Information Visualization*, 1(1):35–49.
- Good, L., Stefik, M., Baudisch, P., and Bederson, B. B. (2002b). *Automatic text reduction for changing size constraints*. *Conference on Human Factors in Computing Systems*, pages 798–799.
- Good, Lance (2002). *Staying in the flow with zoomable user interfaces*. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 548–549. ACM, New York, NY, USA. ISBN 1581134541. doi:10.1145/506443.506475.

- Good, Lance, Stefik, Mark J., and Bederson, Benjamin B. (2004). *A comparison of zoomable user interfaces and folders for grouping visual objects*. Technical report, Palo Alto Research Center and The Human Computer Interaction Lab, University of Maryland.
- Good, Lance E. (2003). *Zoomable User Interfaces For The Authoring And Delivery Of Slide Presentations*. Ph.D. thesis, University of Maryland, College Park.
- Greenberg, S. and Roseman, M. (2003). *Using a room metaphor to ease transitions in groupware. Sharing Expertise: Beyond Knowledge Management*.
- Guimbretiere, Francois, Stone, Maureen, and Winograd, Terry (2001). *Fluid interaction with high-resolution wall-size displays*. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 21–30. ACM Press, New York, NY, USA. ISBN 158113438X. doi:10.1145/502348.502353.
- Gundelsweiler, Fredrik, Memmel, Thomas, and Harald, Reiterer (2004). *Agile usability engineering*. In *Proceedings of the 4th Mensch & Computer conference (MCI 2007, Paderborn, Germany)*, pages 33–42. Oldenbourg Verlag.
- Gutwin, C. and Greenberg, S. (2002). *A descriptive framework of workspace awareness for real-time groupware*. *Computer Supported Cooperative Work*, 11:411–446.
- Gutwin, Carl (1997). *Workspace Awareness in Real-Time Distributed Groupware*. Ph.D. thesis, Department of Computer Science, University of Calgary, Canada.
- Hartson, Rex H. and Hix, Deborah (1989). *Human-computer interface development: concepts and systems for its management*. *ACM Comput. Surv.*, 21(1):5–92. ISSN 0360-0300. doi: 10.1145/62029.62031.
- Heim, Steven (2007). *The Resonant Interface: HCI Foundations for Interaction Design*. Addison Wesley. ISBN 0321375963.
- Hix, Deborah and Hartson, Rex H. (1993). *Developing User Interfaces: Ensuring Usability Through Product & Process (Wiley Professional Computing)*. Wiley. ISBN 0471578134.
- Holman, David, Stojadinović, Predrag, Karrer, Thorsten, and Borchers, Jan (2006). *Fly: an organic presentation tool*. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 863–868. ACM, New York, NY, USA. ISBN 1595932984. doi: 10.1145/1125451.1125620.
- Holtzblatt, Karen, Wendell, Jessamyn B., and Wood, Shelley (2004). *Rapid Contextual Design: A How-to Guide to Key Techniques for User-Centered Design (Interactive Technologies)*. Morgan Kaufmann. ISBN 0123540518.
- Hornbaek, Kasper, Bederson, Benjamin B., and Plaisant, Catherine (2002). *Navigation patterns and usability of zoomable user interfaces with and without an overview*. *ACM Trans. Comput.-Hum. Interact.*, 9(4):362–389. ISSN 1073-0516. doi:10.1145/586081.586086.
- Hudson, William (2000). *The whiteboard: metaphor: a double-edged sword*. *interactions*, 7(3):11–15. ISSN 1072-5520. doi:10.1145/334216.334221.
- Hussey, A. (1996). *Object-oriented specification and design of user-interfaces*. In *Computer-Human Interaction, 1996. Proceedings., Sixth Australian Conference on*, pages 336–337. doi:10.1109/OZCHI.1996.560157.
- Igarashi, Takeo and Hinckley, Ken (2000). *Speed-dependent automatic zooming for browsing large documents*. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 139–148. ACM, New York, NY, USA. ISBN 1581132123. doi: 10.1145/354401.354435.
- Jacobson, I. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Professional. ISBN 0201544350.
- Januszewski, Karsten and Rodriguez, Jaime (2007). *The new iteration - how xaml transforms the collaboration between designers and developers in windows presentation foundation*. Technical report, Microsoft Corporation.

- Jetter, Hans C. (2007). *Informationsarchitektur und Informationsvisualisierung für die Post-WIMP Ära*. Master's thesis, University of Konstanz.
- Jokela, Timo, Iivari, Netta, Matero, Juha, and Karukka, Minna (2003). *The standard of user-centered design and the standard definition of usability: analyzing ISO 13407 against ISO 9241-11*. In *CLIH '03: Proceedings of the Latin American conference on Human-computer interaction*, pages 53–60. ACM Press, New York, NY, USA.
- Jose, Jorge B. (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*. Zurich, Switzerland.
- Ju, Wendy G., Lee, Brian A., and Klemmer, Scott R. (2007). *Range: exploring proxemics in collaborative whiteboard interaction*. In *CHI '07: CHI '07 extended abstracts on Human factors in computing systems*, pages 2483–2488. ACM, New York, NY, USA. ISBN 9781595936424. doi:10.1145/1240866.1241028.
- Jul, Susanne and Furnas, George W. (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*, pages 97–106. ACM, New York, NY, USA. ISBN 1581130341. doi:10.1145/288392.288578.
- Kang, H., Shneiderman, B., and Human (2000). *Visualization Methods for Personal Photo Collections: Browsing and Searching in the PhotoFinder*. Human-Computer Interaction Laboratory, Institute for Advanced Computer Studies.
- Kang, H., Bederson, B. B., and Suh, B. (2007). *Capture, annotate, browse, find, share: Novel interfaces for personal photo management*. *International Journal of Human-Computer Interaction*, 23(3):315–337.
- Karat, John and Bennett, John (1990). *Supporting effective and efficient design meetings*. In *INTERACT '90: Proceedings of the IFIP TC13 Third International Conference on Human-Computer Interaction*, pages 365–370. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands. ISBN 0444888179.
- König, Daniel (2008). *Anforderungen an Requirements Engineering Werkzeuge für die visuelle Spezifikation*. Master's thesis, University of Konstanz.
- König, Werner A. (2006). *Referenzmodell und Machbarkeitsstudie für ein neues Zoomable User Interface Paradigma*. Master's thesis, University of Konstanz.
- König, Werner A., Bieg, Hans J., Schmidt, Toni, and Reiterer, Harald (2007). *Position-independent interaction for large high-resolution displays*. In *IHCI'07: Proceedings of IADIS International Conference on Interfaces and Human Computer Interaction 2007*, pages 117–125. IADIS Press.
- Landay, J. and Myers, B. (1995a). *Just draw it! programming by sketching storyboards*.
- Landay, J. A. and Myers, B. A. (2001). *Sketching interfaces: toward more human interface design*. *Computer*, 34(3):56–64. doi:10.1109/2.910894.
- Landay, James A. (1996). *Silk: sketching interfaces like crazy*. In *CHI '96: Conference companion on Human factors in computing systems*, pages 398–399. ACM, New York, NY, USA. ISBN 0897918320. doi:10.1145/257089.257396.
- Landay, James A. and Myers, Brad A. (1995b). *Interactive sketching for the early stages of user interface design*. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 43–50. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. ISBN 0201847051. doi:10.1145/223904.223910.
- Laurel, Brenda (1990). *The Art of Human-Computer Interface Design*. Addison-Wesley Professional. ISBN 0201517973.
- Le Peuple, Jenny and Scane, Robert (2003). *User Interface Design (Crucial Study Texts for Computing Degree Courses)*. Crucial. ISBN 1903337194.



- Lin, J. (2003). *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.
- Lin, James (1999). *A visual language for a sketch-based ui prototyping tool*. In CHI '99: CHI '99 extended abstracts on Human factors in computing systems, pages 298–299. ACM, New York, NY, USA. ISBN 1581131585. doi:10.1145/632716.632899.
- Lin, James, Newman, Mark W., Hong, Jason I., and Landay, James A. (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, pages 510–517. ACM Press, New York, NY, USA. ISBN 1581132166. doi:10.1145/332040.332486.
- Lin, James, Newman, Mark W., Hong, Jason I., and Landay, James A. (2001). *Denim: an informal tool for early stage web site design*. In CHI '01: CHI '01 extended abstracts on Human factors in computing systems, pages 205–206. ACM Press, New York, NY, USA. ISBN 1581133405. doi:10.1145/634067.634190.
- Maguire, M. (2001). *Methods to support human-centred design*. *International Journal of Human-Computer Studies*, 55(4):587–634.
- Mayhew, Deborah J. (1999). *The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design (The Morgan Kaufmann Series in Interactive Technologies)*. Morgan Kaufmann. ISBN 1558605614.
- Mayhew, Deborah J. (2005). *Cost-Justifying Usability, Second Edition: An Update for the Internet Age, Second Edition (Interactive Technologies)*. Morgan Kaufmann. ISBN 0120958112.
- Mommel, T., Geyer, F., Rinn, J., and Reiterer, H. (2007a). *Interdisciplinary visual user interface specification*. Technical report, University of Konstanz.
- Mommel, T., Geyer, F., Rinn, J., and Reiterer, H. (2008). *Tool-support for interdisciplinary and collaborative user interface specification*. In *To be published in: Proceedings of the IADIS International Conference on Interfaces and Human Computer Interaction (IHCI 2008, Amsterdam, The Netherlands)*.
- Mommel, Thomas and Heilig, Mathias (2007). *Model-based visual software specification*. In *Proceedings of the IADIS International Conference on Interfaces and Human Computer Interaction (IHCI 2007, Lisbon, Portugal)*, pages 3–10. IADIS Press, Lisbon, Portugal.
- Mommel, Thomas and Reiterer, Harald (2007). *Visuelle spezifikation interaktiver systeme mit modell- und xml-basierten prototyping-werkzeugen und werkzeugketten*. In *Proceedings of the 1st conference on Software Engineering Essentials (SEE 2007, Munich, Germany)*, pages 78–92. IfI Technical Report Series IfI-07-07.
- Mommel, Thomas and Reiterer, Harald (2008). *Inspector: Method and tool for visual ui specification*. In *Proceedings of the 3rd IASTED International Conference on Human Computer Interaction (IASTED-HCI '08)*. Acta Press, Canada (to be published).
- Mommel, Thomas, Bock, C., and Reiterer, Harald (2007b). *Model-driven prototyping for corporate software specification*. In *Proceedings of the EHCI-HCSE-DSVIS'07*. available online: <http://www.se-hci.org/ehci-hcse-dsvis07/accepted-papers.html>.
- Mommel, Thomas, Gundelsweiler, Fredrik, and Reiterer, Harald (2007c). *Agile human-centered software engineering*. In *Proceedings of 21st BCS HCI Group conference (HCI 2007, University of Lancaster, UK)*, pages 167–175. British Computer Society.
- Mommel, Thomas, Gundelsweiler, Fredrik, and Reiterer, Harald (2007d). *Cruiser: a cross-discipline user interface & software engineering lifecycle*. In *Proceedings of the 12th International Conference on Human-Computer Interaction (HCII 2007, Beijing, China)*. Springer-Verlag, Berlin, Heidelberg.
- Mommel, Thomas, Gundelsweiler, Fredrik, and Reiterer, Harald (2007e). *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), Chamonix, France: Proceedings*, pages 177–182. Acta Press, Canada.

- Memmel, Thomas, Heilig, Mathias, Schwarz, Tobias, and Reiterer, Harald (2007f). *Visuelle softwarespezifikation*. Technical report, University of Konstanz.
- Memmel, Thomas, Heilig, Mathias, Schwarz, Tobias, and Reiterer, Harald (2007g). *Visuelle spezifikation interaktiver softwaresysteme*. In *Proceedings of the 7th Mensch & Computer conference (MCI 2007, Weimar, Germany)*, pages 307–310. Oldenbourg Verlag.
- Memmel, Thomas, Reiterer, Harald, and Holzinger, A. (2007h). *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, Beijing, China)*. Springer-Verlag, Berlin, Heidelberg 2007.
- Memmel, Thomas, Reiterer, Harald, Ziegler, Heiko, and Oed, Richard (2007i). *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 e.V.*, pages 99–104. Fraunhofer IRB Verlag, Stuttgart.
- Memmel, Thomas, Reiterer, Harald, Ziegler, Heiko, and Oed, Richard (2007j). *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 e.V.*, pages 99–104. Fraunhofer IRB Verlag, Stuttgart.
- Moggridge, Bill (2006). *Designing Interactions*. The MIT Press. ISBN 0262134748.
- Moran, Thomas P. and Carroll, John M., editors (1996). *Design Rationale: Concepts, Techniques, and Use (Computers, Cognition, and Work)*. CRC. ISBN 0805815678.
- Moran, Thomas P. and van Melle, William (2000). *Tivoli: integrating structured domain objects into a freeform whiteboard environment*. In *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*, pages 20–21. ACM, New York, NY, USA. ISBN 1581132484. doi:10.1145/633292.633306.
- Muller, Michael J. (2003). *Participatory design: the third space in hci*. *Handbook of HCI*. Mahway NJ USA: Erlbaum, pages 1051–1068.
- Myers, Brad, Hudson, Scott E., and Pausch, Randy (2000). *Past, present, and future of user interface software tools*. *ACM Trans. Comput.-Hum. Interact.*, 7(1):3–28. ISSN 1073-0516. doi:10.1145/344949.344959.
- Myers, Brad A. and Rosson, Mary B. (1992). *Survey on user interface programming*. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 195–202. ACM Press, New York, NY, USA. ISBN 0897915135. doi:10.1145/142750.142789.
- Mynatt, E. D. (1999). *The Writing on the Wall*. *Human-computer Interaction, INTERACT'99*.
- Nielsen, J. (1992). *The usability engineering life cycle*. *Computer*, 25(3):12–22. doi:10.1109/2.121503.
- Nielsen, Jakob (1994). *Usability Engineering (The Morgan Kaufmann Series in Interactive Technologies)*. Morgan Kaufmann. ISBN 0125184069.
- Norman, Donald and Draper, Stephen (1986). *User Centered System Design: New Perspectives on Human-computer Interaction*. TF-CRC. ISBN 0898598729.
- Norman, Donald A. (2002). *The Design of Everyday Things*. Basic Books. ISBN 0465067107.
- Nunes, D. N. J. (2001). *Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach*. Ph.D. thesis, Universidade da Madeira.
- Nunes, Nuno J. and Campos, Pedro (2004). *Towards usable analysis, design and modeling tools*. In *Proceedings of the Workshop on Making model-based User Interface Design practical: usable and open methods*. Funchal, Portugal.
- Perlin, Ken and Fox, David (1993). *Pad: an alternative approach to the computer interface*. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 57–64. ACM Press, New York, NY, USA. ISBN 0897916018. doi:10.1145/166117.166125.

- Phanouriou, C. (2000). *UIML: A Device-Independent User Interface Markup Language*. Ph.D. thesis, Virginia Polytechnic Institute and State University.
- Plaisant, Catherine, Milash, Brett, Rose, Anne, Widoff, Seth, and Shneiderman, Ben (1996). *Lifelines: visualizing personal histories*. In *CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press, New York, NY, USA. ISBN 0897917774. doi:10.1145/238386.238493.
- Pook, Stuart (2001). *Interaction and Context in Zoomable User Interfaces*. Ph.D. thesis, École Nationale Supérieure des Télécommunications, Paris, France.
- Pook, Stuart, Lecolinet, Eric, Vaysseix, Guy, and Barillot, Emmanuel (2000). *Context and interaction in zoomable user interfaces*. In *AVI '00: Proceedings of the working conference on Advanced visual interfaces*, pages 227–231. ACM, New York, NY, USA. ISBN 1581132522. doi:10.1145/345513.345323.
- Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., and Carey, T. (1994). *Human-Computer Interaction: Concepts And Design (ICS)*. Addison Wesley. ISBN 0201627698.
- Preece, Jennifer, Rogers, Yvonne, and Sharp, Helen (2002). *Interaction Design*. Wiley. ISBN 0471492787.
- Preim, Bernhard (1999). *Entwicklung interaktiver Systeme: Grundlagen, Fallbeispiele und innovative Anwendungsfelder (Springer-Lehrbuch)*. Springer. ISBN 3540656480.
- Raskin, Jef (2000). *The Humane Interface: New Directions for Designing Interactive Systems*. Addison-Wesley Professional. ISBN 0201379376.
- Rinn, Johannes (2008). *Methods and tool-support for interdisciplinary requirements modeling and user interface specification*. Master's thesis, University of Konstanz.
- Robertson, George G., Mackinlay, Jock D., and Card, Stuart K. (1991). *Cone trees: animated 3d visualizations of hierarchical information*. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 189–194. ACM Press, New York, NY, USA. ISBN 0897913833. doi:10.1145/108844.108883.
- Roseman, Mark and Greenberg, Saul (1996). *Teamrooms: network places for collaboration*. In *CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 325–333. ACM, New York, NY, USA. ISBN 0897917650. doi:10.1145/240080.240319.
- Roseman, Mark and Greenberg, Saul (1997). *A tour of teamrooms*. In *CHI '97: CHI '97 extended abstracts on Human factors in computing systems*, pages 208–209. ACM, New York, NY, USA. ISBN 0897919262. doi:10.1145/1120212.1120346.
- Rosson, Mary B. and Carroll, John M. (1995). *Narrowing the specification-implementation gap in scenario-based design*. *Scenario-Based Design: Envisioning Work and Technology in System Development*, pages 247–278.
- Rosson, Mary B. and Carroll, John M. (2001). *Usability Engineering: Scenario-Based Development of Human Computer Interaction (Interactive Technologies)*. Morgan Kaufmann. ISBN 1558607129.
- Schaffer, D., Zuo, Z., Greenberg, S., Bartram, L., Dill, J., Dubs, S., and Roseman, M. (1996). *Navigating hierarchically clustered networks through fisheye and full-zoom methods*. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(2):162–188.
- Sears, Andrew and Jacko, Julie A., editors (2007). *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications, Second Edition (Human Factors and Ergonomics)*. CRC. ISBN 0805858709.
- Sharp, Helen, Rogers, Yvonne, and Preece, Jenny (2007). *Interaction Design: Beyond Human Computer Interaction*. Wiley. ISBN 0470018666.
- Shneiderman, B. (1996). *The eyes have it: a task by data type taxonomy for information visualizations*. *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. doi:10.1109/VL.1996.545307.

- Shneiderman, B. (1998). *Treemaps for space-constrained visualization of hierarchies*. *Human-Computer Interaction Lab, University of Maryland, Dec, 26*.
- Shneiderman, B., Fischer, G., Czerwinski, M., Resnick, M., Myers, B., Candy, L., Edmonds, E., Eisenberg, M., Giaccardi, E., Hewett, T., and Others (2006). *Creativity support tools: Report from a us national science foundation sponsored workshop*. *International Journal of Human-Computer Interaction*, 20(2):61–77.
- Shneiderman, Ben (1992). *Designing the user interface (2nd ed.): strategies for effective human-computer interaction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0201572869.
- Shneiderman, Ben (2000). *Creating creativity: user interfaces for supporting innovation*. *ACM Trans. Comput.-Hum. Interact.*, 7(1):114–138. ISSN 1073-0516. doi:10.1145/344949.345077.
- Shneiderman, Ben (2003). *Leonardo's Laptop : Human Needs and the New Computing Technologies*. The MIT Press. ISBN 0262692996.
- Shneiderman, Ben and Plaisant, Catherine (2004). *Designing the User Interface : Strategies for Effective Human-Computer Interaction (4th Edition)*. Addison Wesley. ISBN 0321197860.
- Sigchi (1992). *A metamodel for the runtime architecture of an interactive system*. *The UIIMS Workshop Tool developers*, pages 32–37.
- Sutherland, Ivan E. (1964). *Sketch pad a man-machine graphical communication system*. In *DAC '64: Proceedings of the SHARE design automation workshop*. ACM Press, New York, NY, USA. doi:10.1145/800265.810742.
- Traetteberg, H. (2004). *Integrating dialog modelling and application development*. *Proc. of the First International Workshop on Making Model-based User Interface Design Practical MBUI*.
- 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*, pages 19–20.
- Verplank, Bill, Sapp, Craig, and Mathews, Max (2001). *A course on controllers*. In *NIME '01: Proceedings of the 2001 conference on New interfaces for musical expression*, pages 1–4. National University of Singapore, Singapore, Singapore.
- Wallace, M. D. and Anderson, T. J. (1993). *Approaches to interface design*. *Interacting with Computers*, 5(3):259–278.
- Ware, Colin (2004). *Information Visualization, Second Edition: Perception for Design (Interactive Technologies)*. Morgan Kaufmann. ISBN 1558608192.
- Woodruff, Allison, Landay, James, and Stonebraker, Michael (1998). *Goal-directed zoom*. In *CHI '98: CHI 98 conference summary on Human factors in computing systems*, pages 305–306. ACM, New York, NY, USA. ISBN 1581130287. doi:10.1145/286498.286781.
- Yang, H. (1999). *Collaborative applications of zoomable user interface*. Technical report, Collaboratory for Research on Electronic Work, School of Information, University of Michigan.