

CRUISER: A Cross-Discipline User Interface and Software Engineering Lifecycle

Thomas Memmel, Fredrik Gundelsweiler, and Harald Reiterer

Human-Computer Interaction Lab
University of Konstanz, D-78457 Konstanz, Germany
{mommel, gundelsw, reiterer}@inf.uni-konstanz.de

Abstract. This article seeks to close the gap between software engineering and human-computer interaction by indicating interdisciplinary interfaces of SE and HCI lifecycles. We present a cross-discipline user interface design lifecycle that integrates SE and HCI under the umbrella of agile development.

Keywords: Human-Computer Interaction, Usability Engineering, Extreme Programming, Agile Modeling, User-centered Design & Development (UCD).

1 Human-Computer Interaction and Software Engineering

From its birth in the 1980's, the field of human-computer interaction (HCI) has been defined as a multidisciplinary subject. To design usable systems, experts in the HCI arena are required to have distinct skills, ranging from an understanding of human psychology, to requirements modeling and user interface design (UID) [1]. In this article we will use the term user interface (UI) designer as a synonym for a professional who combines knowledge of usability, graphics and interaction design.

Table 1. Methods for integrating SE and UE, based on [2] (excerpt)

Integration issue	Method of application
Mediating and improving the communication lines between users, usability experts and developers	Use medium-weight artifacts, work with toolkits appropriate for collaborative design, talk the same language, work in pairs
Extending software engineering artifacts for UI specification & conceptualization	Use artifacts known by both professions and adjust their expressiveness
Extending RE methods for collecting information about users and usability	Include principles, practice and light- to medium-weight methods from HCI into RE
Representing design artifacts including prototypes using different formalisms	Apply prototyping as a method of participatory design; all stakeholders gather requirements

Whereas HCI focuses on UID issues such as ease of use, ease of learning, user performance, user satisfaction or aesthetics, software engineering (SE) considers how

functional requirements are translated into a running system. HCI and SE are recognized as professions made up of very distinct populations. Each skill set is essential for the production of quality software, but no one set is sufficient on its own. The interaction layer is the area where HCI and SE are required to work together, in order to ensure that the resulting software product behaves as specified in the initial requirements engineering (RE). To provide a high level of UI usability, software SE has to work with people with a background in HCI, but the course of collaboration is mostly unclear. It is therefore true that classic and agile SE methods still lack integration of HCI methods and processes (see Table 1).

Bearing these two different engineering disciplines in mind, each software design process can be characterized in terms of its dependency on its engineering orientation, ranging from a formal and model-based methodology to an informal explanatory design. SE tends to be more formal and “consequently, the business user and IT analyst may think that they both agree on a design, only to discover down the line that they had very different detailed implementations and behaviors in mind” [3].

Very formal or complex models are an inappropriate base for communication, especially so for collaborative design processes with high user- and business-stakeholder participation. Scenarios [4] - known as user stories in Extreme Programming (XP) [5] - and prototypes are recognized as interdisciplinary modeling language for RE and as bridging techniques for HCI and SE [6]. In SE, scenarios – as a sequence of events triggered by the user – are generally used for requirements gathering and for model checking. HCI applies scenarios to describe software context, users, user roles, tasks and interaction [4]. Prototypes in SE are used to verify functional specifications and models. Agile Modeling (AM) and XP recognize prototypes as a type of small release [5,7], whereas HCI mainly employs them for iterative UID [8]. The bottom-line is that some informal methods of XP and AM are close to HCI practice and therefore the pathfinder for a common course of action. While heavy-weight methods such as style guides (HCI) are far too expensive, light-weight methods such as essential use cases (SE) are in contrast too abstract for system specification. Cross-discipline agile methods are the optimum, and workable, compromise. Agile approaches of both SE [5] and HCI [9,10] are therefore the interface for our common and balanced software lifecycle known as *CRUISER*.

2 From XP to Agile Cross-Discipline Software Engineering

In contrast to classic, heavy-weight SE processes like the V-Model, agile methods begin coding at a very early stage while having a shorter up-front RE phase. Following the paradigm of XP, implementation of code takes place in small increments and iterations, and the customer is supplied with small releases after each development cycle. During the exploration phase, teams write user stories in an attempt to describe user needs and roles. But the people interviewed need not necessarily be the end-users of the eventual software. XP therefore often starts coding based only on assumptions about end-user needs [10].

AM is less rigid than XP and takes more care over initial RE as it provides room for low-fi prototyping, activity diagrams or use-case diagrams [11]. Nevertheless, the analysis phase is finished as soon as requirements have been declared on a horizontal

level, because the iterative process assumes that missing information will be filled in at later stages. Development in small increments may work properly as long as the software is not focused on the UI. Changes to software architecture usually have no impact on what the user sees and interacts with. With the UI, however, it is a different story. When designing UIs, continual changes to the UI may give rise to conflicts with user expectations and learnability, cause inconsistency and finally lead to user dissatisfaction. Thus, agile development does not really qualify as user-centered design (UCD), but can function as one pillar for an integrated approach [10].

Both SE and UID have to cope with a shorter time-to-market, in which the quality of the delivered software must not suffer. This therefore is a great challenge both for management and the methods and tools applied. Our idea is a balanced hybrid process, which is both agile SE and agile UCD, and which is consistent with the principles and practices of both disciplines. In order to identify interfaces between agile SE and agile HCI, we have to highlight different approaches to UID, analyze their agile potential and their different contributions to a cross-discipline process.

Like XP, original UCD is a highly iterative process. It differs from agile methods, however, since real users are taken into account and the development team tries to understand user needs and tasks before any line of code is written. The lifecycles of usability engineering processes [4,12] provide numerous methods and tools that should support the designer in gathering all of the required information. Most of these methods are rated as heavy-weighted, due to their claim to analyze and document as much as possible about users, work flows, context, etcetera right from the beginning.

Constantine [9] argues that UCD produces design ideas in a rather magical process in which the transformation from claims to design is neither comprehensible nor traceable. Such a “Black Box Designer” produces creative solutions without being able to explain or illustrate what goes on in the process. Furthermore, UCD tries to converge the resulting, often diverse, design alternatives into a single solution, which is then continuously evaluated and refined. UCD may therefore take a long time, or even fail, if too many users are involved and narrowing the design space is difficult. Iteration may create the illusion of progress, although the design actually goes round in circles and solutions remain elusive. Altogether, a one-to-one integration of UCD processes and methods is in general inappropriate for an agile course of action. Constantine’s usage-centered design approach takes up the basic philosophy of AM and concentrates on essential and easy to understand models. Through their application, HCI becomes more formal, but the simplicity of their syntax still enables collaborative design by engineering rather than by trial and error [9] (see Table 2). Although the list of usage-centered design success stories is creditable, the products praised tend to support user performance rather than user experience. This cannot be the only aspiration of a modern design approach, however. This is where Donald Norman’s recently proposed activity-centered design approach (ACD) [13] comes in. Products causing a high joy of use can reach great user acceptance even when they lack usability. Norman therefore votes for the integration of emotional design issues and the stronger consideration of user satisfaction. In Lowgren and Stolterman’s book about thoughtful interaction design (TID) [14], the designer, in order to design such highly usable and aesthetic systems, switches between 3 levels of abstraction: vision, operative image and specification. If the designer is confronted with a design situation, at first an often sketchy and diffuse vision emerges. Frequently, several visions are promising and are therefore competing to be implemented, eventually resulting in a chaos of conflicting visions. The initial

version of the operative image is the first externalization of the vision, e.g. captured in mock-ups or elaborated interactive (hi-fi) prototypes. It enables manipulation, stimulation, visualization and decision making for the most promising design. The designer wants to learn as much about the design space as possible, narrowing the design towards the best solution as late as possible. The operative image is transformed into a (visual) specification of the final design if it is sufficiently detailed.

Table 2 shows a comparison of the design approaches under discussion. Our development lifecycle is set up on the core methods of all the approaches presented, such as e.g. selective user involvement (UCD, ACD), prototyping for visual thinking (TID), as well as modeling with scenarios or task maps (usage-centered design).

Table 2. Comparison of user interface design approaches, adapted from [9]

User-Centered Design	Usage-Centered Design	Activity-Centered Design	Thoughtful Interaction Design
Focus is on users	Focus is on usage	Focus is on activities	Focus is on design
Substantial user involvement	Selective user involvement	Authoritative user involvement	Thoughtful user involvement
User studies Particip. Design User testing	Explorative modeling Model validation Usability inspections	Sequential activity / Task analysis Emotional design	Visual Thinking Particip. Prototyping Visual Specification
Iterative prototyping	Modeling & abstract prototyping	Understanding activities	Switching btw. abstract and detail
Informal, black box process	Systematic, specified white box process	Rule-breaking black box process	Depends on guidance and authority
Design by trial-and-error	Design by engineering	Design by authority	Design by visual thinking

All designers in a project need to have a similar understanding of the vision and the wholeness of the system (TID). Thus continuous and lively discussion is necessary (XP). Informal communication across organizational borders should be easy, and teams should have common spaces (XP). Since reaching agreement on abstract notions (text) is difficult, ideas have to be made visible, allowing participants to look at, feel, analyze and evaluate them as early as possible (XP, AM).

The process should be controlled by an authoritative person who must have a deep understanding of both SE and HCI. With our demand for such highly capable personnel, we concur with what XP and AM announced as one of their most important criteria for project success [5]. The leader navigates through the development process, proposes solutions to critical design issues and applies the appropriate design, engineering and development methods. Since the gap between SE and HCI becomes less significant “when the (HCI) specialist is also a strong programmer and analyst” [2], we chose XP as fundamental to our thoughts on bonding SE and HCI. Its principle of pair programming allows people with different fields of expertise, but common capabilities, to design a system together.

The basis of our cross-discipline lifecycle is therefore the identification of similarities between XP and HCI (see Table 3), AM and HCI (see Table 4), as well as ACD and TCD when compared to HCI, AM and XP (see Table 5). We outline some major similarities, although our comparison highlighted many more interfaces of these disciplines. Although different in their wording, agile principles and practices are comparable and show a significant overlap, such as in iterative design, small releases and prototyping, story cards of active stakeholder participation and scenarios, or testing and evaluation. Modern UID approaches do not oppose collaboration with SE; on the contrary, they underline the commonalities.

Table 3. Similarities between XP and HCI (excerpt)

XP Practice	HCI Practice
Iteration, Small Increments, Adaptivity	Prototyping
Planning Game	Focus Groups
Story Cards, Task Cards, User Stories	Scenarios, User Profiles, Task Model

Table 4. Similarities between AM and HCI (excerpt)

Agile Modeling Practice	Usability Engineering Practice
Prove It With Code	Prototyping
Create Several Models in Parallel	Concurrent Modeling
Active Stakeholder Participation	Usage-Centered Design, User Participation
Consider Testability	Evaluation, Usability Inspections

Table 5. Overall comparison of agile SE, usual HCI and other practice (excerpt)

AM & XP Practice	HCI Practice	TID & ACD Practice
Minimalist documentation	Comprehensible models	Interactive representations
Show results early	Lo-/Hi-Fi prototyping	Make ideas visible asap
Small teams, design rooms	Design rooms, styles guides	Informal communication
Active stakeholder part.	Collaborative design	externalization of visions
User performance	User performance, user experience	User performance, user experience, hedonic quality

3 Agile Cross-Discipline User Interface Design and Software Engineering Lifecycle

Our agile cross-discipline user interface and software engineering lifecycle, called *CRUISER*, originates in our experience of developing various kinds of interactive

software systems in teams with up to 20 members [16]. Although CRUISER is based on XP, we firmly believe in a scaling of our lifecycle for larger teams, bearing in mind success stories of agile development with several hundred team members [17] and within large organizations [18]. For the following explanation of CRUISER, we concentrate on those issues that need to be worked out collaboratively by HCI and SE experts. SE practice that is independent from UID are not mentioned in detail.

CRUISER starts with the initial requirements up-front (IRUP, see Table 6), which must not take longer than the claims analysis in XP. The agile timeframe can be preserved if the methods employed can be rated as agile (see Table 3, 4, 5) and interdisciplinary. Concerning the design of the UI, XP and AM practice is not sufficient and has to be endorsed by UID practice and authoritative design (TID, ACD).

Table 6. CRUISER initial requirements up-front; contributions of disciplines

Initial Requirements Up-Front (IRUP)		
Agile SE	Human-Computer Interaction	Authoritive Design
Use Cases, Usage Scenarios	Role & Task Model	Sketchy Design Visions
Technical Requirements	User-, Task-, Interaction Scenarios	Hi-Fi Prototypes
User Performance Goals	Essential Use Cases	Hedonic Quality Goals
	UI Patterns	
	User Experience Goals	

As discussed in Chapter 2, the real users have to be taken into account rather than just stakeholders of any kind. Appropriate cross-discipline methods for analyzing user needs are role models and task models. The model-based RE proposed by [9] focuses on surveying essential information and satisfies an agile course of action due to the use of index cards. The user roles are prioritized (Focal User Roles) and sorted in relation to their impact on product success. Finally, essential use cases describe user tasks and enable the building of task model and task map. Like user roles, task cases are sorted in accordance with Kent Beck’s proposal, which is “required - do first, desired - do if time, deferred - do next time”, whenever the necessary scenarios are established for understanding and communication. For a shared understanding of developers and for communication with stakeholders, all models are translated into scenarios, which can focus on different aspects of UID (users, tasks, interactions).

Since agile methods do not consider the UI in detail, they do not recognize extensive style guides as used in HCI practice. We therefore suggest light-weight style guides that are shorter, more relevant and contain UI patterns [19]. They ease the design process by providing design knowledge and experience (AM: Apply Design Standards, Use Existing Resources). During all IRUP assignments, users, HCI, SE and business personnel support and finalize RE with initial discussions about scenarios and design alternatives. This alone will result in various outline visions such as mockups or prototypes that make up the initial project design space.

In contrast to other HCI lifecycles (e.g. [12]), CRUISER envisions the externalization of design visions even before the requirements analysis is finished. In our opinion, this break with common HCI practice enables the UI designer to decide

very early about the degree of user involvement and the necessity of more innovative solutions. He can have a considerable influence on balancing user performance, user experience and hedonic quality demands and can guide the IRUP accordingly.

The second phase of the development process is the initial conceptual phase (ICP, see Figure 1). In the ICP we envisage a separation of ongoing UI prototyping from architectural prototyping whenever possible to speed up the process. The conscientious application of software patterns [19] facilitates this procedure.

The development of UI and system architecture can take place in parallel as soon as a minimalist, common UI specification [13] is generated and the necessary interfaces are identified. Dependencies between UI and system architecture can be found with the help of task cases and scenarios established during IRUP. It is very likely that highly interactive UIs will have greater impact on the system architecture.

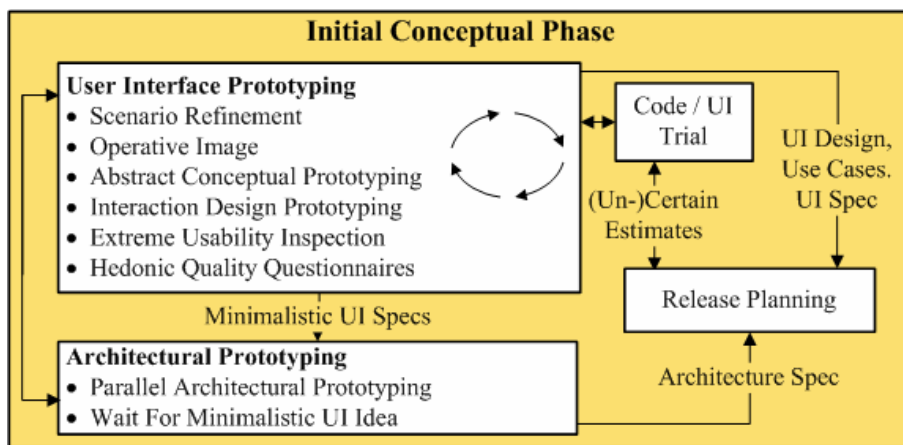


Fig. 1. CRUISER initial conceptual phase

As discussed, prototypes are common practice in HCI and SE. The overall purpose of the ICP is therefore the generation of more detailed and interactive prototypes for narrowing the design space towards a single solution through discussion with stakeholders and through scenario refinement [3]. For this assignment, the designer must leap between abstract and detailed levels of prototyping, always considering a timeframe and expressivity suitable for an agile project environment (see Table 7).

Bearing in mind the claims of agile methods, prototypes should be easy to work with and, above all, quick to produce and easy to maintain. With more interactive and complex external representations, the designer conducts a dialogue about design solutions and ideas. Prototypes that are visually more detailed help us to overcome the limitations of our cognitive abilities to process, develop, and maintain complex ideas and to produce a detailed operative image (TID). As long as the prototype can be modified using simple direct manipulation techniques, the users can be proactively involved in the participatory process. In addition to low-fi prototyping for e.g. conceptual design, a modern UID approach must also provide methods and tools for hi-fi prototyping that overcomes most of the disadvantages mentioned in Table 7. We

recommend prototyping tools such as Macromedia Flash and iRise Studio. They are easy to use for all stakeholders due to the absence of coding, they allow reuse of components through the application of patterns or templates, and they produce running interactive simulations that can be enhanced to small releases.

Table 7. Low- and High-Fidelity Prototyping, based on [8] (excerpt)

Type	Advantages	Disadvantages
Low-Fidelity	less time & lower cost evaluate multiple concepts communication device address screen layout issues	limited usefulness for usability tests navigational and flow limitations facilitator-driven poor specification
High-Fidelity	partial/complete functionality interactive use for exploration and test marketing & sales tool	time-consuming to create inefficient for proof-of-concept designs blinds users to major representational flaws management may think it is real

Interactive prototypes can also run as “Spike Solutions”, which are used to evaluate and prove the functionality and interoperability of UI concepts and system architecture. More importantly, they can be applied as visual, interactive UI specifications in the ensuing construction phase. Visual specifications are unambiguous and can guarantee the final system matches stakeholder expectations about UID and behavior. The prototyping-based process minimizes the risk of making wrong design decisions and leads the way towards a winning design solution.

Through the well-balanced and thoughtful application of selected methods of RE such abstract modeling or detailed prototyping, CRUISER avoids a design by trial-and-error and makes the design process move forward in a traceable manner. The process of identifying the most promising design solution is guided by UI evaluations, which can be kept at low complexity if the UE methods applied are agile [20]. In order to give due regard to the UI's hedonic qualities, which are e.g. the ability to stimulate or to express identity, we envision a design review with AttrakDiff [15].

On entering the construction and test phase (CTP), coding starts (see Figure 2). At this phase, the CRUISER lifecycle closely resembles the incremental and iterative manner of XP. CTP therefore begins with iteration planning and the creation of unit- and acceptance-tests, which are later used to evaluate parts of the system architecture (e.g. automatically) and the UI (e.g. with extreme evaluations [20]). The latter guarantees that the previously defined usability or hedonic quality goals are properly taken into account. They are only to be executed if a usability expert on the team identifies a need for it. We therefore recommend the integration of HCI personnel in the pair programming development.

As with the construction of prototypes, the actual coding of UI and system architecture again takes place in parallel, and components of the UI that have great impact may be developed faster initially and then later refined during the following iterations. As in XP, the CTP ends with the deployment of a small release. Before the next iteration starts, each small release can again be evaluated using cheap and fast methods [20]. If usability or hedonic quality issues are identified, they can also be

documented on index cards (“defect cards”). Each defect is assigned to its corresponding task case. The usability defects may be sorted and prioritized and thus reviewed during earlier or later iterations. If usability or design catastrophes occur, HCI and SE experts and stakeholders can decide on the necessary measures.

The last step in the CRUISER lifecycle is the deployment phase. While users are working with the system, new functionality may be requested, or usability and design issues that were underrated during the iterations may be raised. The lifecycle therefore allows for a return to earlier phases to cater for such new requirements.

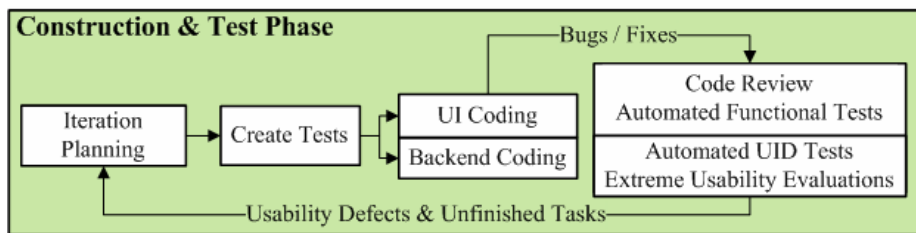


Fig. 2. CRUISER construction and test phase

4 Summary

Our motivation was to take a step towards a cross-discipline procedure for software design with respect to agile movements. With the CRUISER lifecycle, we bridge HCI and SE based on the commonalities of both fields. Similarities can be found in basic principles and practices as well as among the methods and tools that are typically applied. CRUISER has important links to XP [5], but differs from it in many important aspects related to AM, HCI and beyond. For integrating all critical disciplines under the umbrella of one common lifecycle, we concur with the findings of interdisciplinary researchers and use scenarios and prototypes as fundamental artifacts propelling a design process with high involvement of users and stakeholders.

References

1. Pyla, P.S., Pérez-Quñones, M.A., Arthur, J.D., Hartson, H.R.: Towards a Model-Based Framework for Integrating Usability and Software Engineering Life Cycles. In: Proceedings of Interact 2003, Zurich, Switzerland, September 1-3, IOS Press, Amsterdam (2003)
2. Seffah, A., Gulliksen, J., Desmarais, M.C. (eds.): Human-centered software engineering – integrating usability in the development process, pp. 3–14. Springer, Heidelberg (2005)
3. Zetie, C.: Show, Don’t tell - How High-Fidelity Prototyping Tools Improve Requirements Gathering, Forrester Research Inc. (2005)
4. Rosson, M.B., Carroll, J.M.: Usability engineering: scenario-based development of human computer interaction. Morgan Kaufmann, San Francisco (2002)
5. Beck, K.: Extreme Programming Explained. Addison-Wesley, London, UK (1999)

6. Sutcliffe, A.G.: Convergence or competition between software engineering and human computer interaction. In: Seffah, A., Gulliksen, J., Desmarais, M.C. (eds.) Human-centered software engineering – integrating usability in the development process, pp. 71–84. Springer, Heidelberg (2005)
7. Blomkvist, S.: Towards a model for bridging agile development and user-centered design. In: Seffah, A., Gulliksen, J., Desmarais, M.C. (eds.) Human-centered software engineering – integrating usability in the development process, pp. 219–244. Springer, Heidelberg (2005)
8. Rudd, J., Stern, K., Isensee, S.: Low vs. high fidelity prototyping debate, *Interactions*, vol. 3(1), pp. 76–85. ACM Press, New York (1996)
9. Constantine, L.L.: Process agility and software usability: Toward lightweight usage-centered design, *Information Age*, vol. 8(8) (August 2002)
10. Gundelsweiler, F., Memmel, T., Reiterer, H.: Agile Usability Engineering. In: Keil-Slawik, R., Selke, H., Szwillus, G. (Hrsg.) *Mensch & Computer 2004: Allgegenwärtige Interaktion*, pp. 33–42. Oldenbourg Verlag, München (2004)
11. Ambler, W.S.: *Agile Modeling*. John Wiley & Sons, New York (2002)
12. Mayhew, Deborah, J.: *The usability engineering lifecycle - A Practicioners Handbook for User Interface Design*. Morgan Kaufmann, San Francisco (1999)
13. Norman, D.: Human-Centered Design Considered Harmful. *Interactions* 12(4), 14–19 (2005)
14. Lowgren, J., Stolterman, E.: *Thoughtful Interaction Design: A Design Perspective on Information Technology*. MIT Press, Cambridge, MA (2004)
15. Hassenzahl, M., Platz, A., Burmester, M., Lehner, K.: Hedonic and Ergonomic Quality Aspects Determine a Software’s Appeal, In: *Proceedings of the CHI 2000, Conference on Human Factors in Computing*, The Hague, NL, pp. 201–208 (2000)
16. Limbach, T., Reiterer, H., Klein, P., Müller, F.: VisMeB: A visual Metadata Browser. In: Rauterberg, M. pp. 993–996. IOS Press, Amsterdam (2003)
17. Eckstein, J.: *Agile Software Development in the Large: Diving Into the Deep*. Dorset House Publishing Co., Inc. New York (2004)
18. Lindvall, M., Muthig, D., Dagnino, A.: Agile Software Development in Large Organizations. *Computer* 37(12), 26–34 (2004)
19. Borchers, J.: *A Pattern Approach to Interaction Design*. John Wiley & Sons, New York (2001)
20. Gellner, M., Forbrig, P.: Extreme Evaluations – Lightweight Evaluations for Software Developers, In: *IFIP Working Group 2.7/13.4, editor, INTERACT 2003 Workshop on Bridging the Gap Between Software Engineering and Human-Computer Interaction* (2003)