Erschienen in: Proceedings of the 21st British HCI Group Annual Conference on People and Computers : HCI...but not as we know it : volume 1 / Ball, Linden J. et al. (Hrsg.). - New York, NY : ACM, 2007. - S. 167-175. - ISBN 978-1-902505-94-7

Agile Human-Centered Software Engineering

Thomas Memmel
Human-Computer Interaction Lab
Universitätsstrasse 10, Box D73
78457 Konstanz, Germany
+49 7531 88 3547
memmel@acm.org

Fredrik Gundelsweiler
Human-Computer Interaction Lab
Universitätsstrasse 10, Box D73
78457 Konstanz, Germany
+49 7531 88 3547
gundelsw@inf.uni-konstanz.de

Harald Reiterer
Human-Computer Interaction Lab
Universitätsstrasse 10, Box D73
78457 Konstanz, Germany
+49 7531 88 3547
reiterer@inf.uni-konstanz.de

ABSTRACT
We seek to close the gap between software engineering (SE) and human-computer interaction (HCI) by indicating interdisciplinary interfaces throughout the different phases of SE and HCI lifecycles. As agile representatives of SE, Extreme Programming (XP) and Agile Modeling (AM) contribute helpful principles and practices for a common engineering approach. We present a cross-discipline user interface design lifecycle that integrates SE and HCI under the umbrella of agile development. Melting IT budgets, pressure of time and the demand to build better software in less time must be supported by traveling as light as possible. We did, therefore, choose not just to mediate both disciplines. Following our surveys, a rather radical approach best fits the demands of engineering organizations.

General Terms
Design, Human Factors, Theory.

Keywords
Agile development, software engineering, usability engineering, human-centered design, agile usability engineering.

1. INTRODUCTION
From its birth in the 1980s, 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 [30]. HCI professionals are known as interaction designers, usability experts, graphic designers, user experience experts, etc. [5]. In this article we will use the term user interface designer as a synonym for a professional who combines knowledge of usability, graphics and interaction design.

Whereas HCI focuses on user interface design (UID) issues such as ease of use, ease of learning, user performance, user satisfaction or aesthetics, SE considers how functional requirements are translated into a running system. Software engineers are generally trained in topics such as algorithms, data structures, and system architecture or database design [30]. HCI and SE are recognized as professions made up of very distinct populations. Each skill set is essential for the production of quality products, but no one set is sufficient on its own [3]. As software engineers are also responsible for implementing UID specifications as running code, there is a need to communicate with HCI personnel. The interaction layer - as interface between system and user - 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 user interface (UI) usability, software engineering 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 explicit integration of HCI methods and processes" [23, 33].

1.1 Integration Issues
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.

With modeling languages like UML [16], SE tends to be more formal and model-based. UML "is very strong at specifying the structure and the functionality of the application, (but) it is seldom used to (…) specify usability-related information" [36]. Various UML-based languages (e.g., [1, 8]) have been developed in order to overcome this limitation, but the extensions do also lead to more formalism and decreased understanding for business people [26, 40].

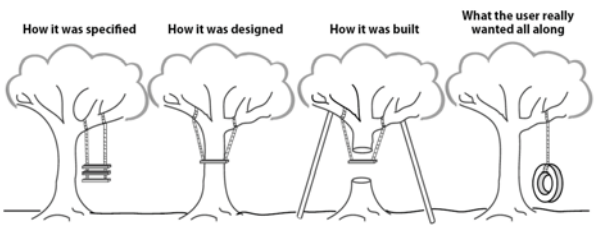

Figure 1. The understanding between HCI, SE and Business personnel [40].

For specifying backend system capabilities, UML is undoubtedly one of the most powerful modeling languages for SE. For UID issues, UML can only describe interactions formally and is unable to visualize real UI behavior. "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" [40] (see Figure 1).

HCI also recognizes some formal models such as use-cases, task models or role models [10]. With regard to the usage of informal artifacts like storyboards, scenarios [31] or paper prototypes, however, HCI is generally more informal and therefore more open to collaborative design with stakeholders.

Table 1. Methods for integrating SE and UE, based on [34].

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 and conceptualization	Use artifacts known by both professions and adjust their expressiveness
Enhancing object-oriented software engineering notations and models	Only use notations and models that can be understood by all stakeholders
Extending RE methods for collecting information about users and usability	Include principles, practice and light- to medium-weight methods from HCI into the RE up-front
Developing new processes for interactive systems design	Reconsider applied methods, keeping in mind shorter time-to-market and business demands
Representing design artifacts including prototypes using different formalisms	Apply prototyping as a method of participatory design, enabling all stakeholders to gather requirements and test systems without coding
Conveying user-centered design (UCD) attitudes, not just tools and methods, to support UCD activities	Integrate tools that support both professions in system implementation by transporting patterns, design knowledge, design principles, etc.

As there is a need for both professions to talk the same language, both disciplines need to agree on a certain degree of formality for describing components of the UI. Several conference workshops highlighted major integration issues, which are summarized in Table 1. For each integration issue indicated by [34], we suggest methods of application in a cross-discipline engineering process.

1.2 Common Modeling Languages

Very formal or complex models are an inappropriate base for communication, especially so for collaborative design processes with high user- and business-stakeholder participation. As an interdisciplinary modeling language for RE, research suggests scenarios [22, 31, 35, 36] - known as user stories (light-weight scenarios) in agile development [4] - and prototypes [36] as bridging techniques for HCI and SE. In SE, scenarios – as a sequence of events triggered by the user – are generally used for requirements gathering and for model checking. Such a (use-case) scenario "is used to identify a thread of usage for the system to be constructed (and) provide a description of how the system will be used" [29]. In contrast, HCI applies scenarios to describe software context, users, user roles, tasks and interaction [31]. Altogether, scenarios can be used as a vision of the system by explaining functionality and interactive behavior, as well as being a description of users and their tasks.

Prototypes are also applied by both disciplines, although their role in SE was less important until agile approaches refocused attention on them as vehicles for inspections and testing. Prototypes in SE can also be used to verify functional specifications and models as well as for understanding problems by doing user inspections. AM already recognizes prototypes as a type of small release, which can continuously be changed and fine-tuned [6]. HCI mainly recognizes them as an artifact for iterative user interface design and distinguishes different dimensions of prototypes [32]. The bottom-line is that some informal methods of agile development are close to HCI practice and therefore the pathfinder for a common course of action.

1.3 Bridging the Gaps

Just as with prototyping and scenarios as common vehicles, HCI and (agile) SE can converge on other shared principles and practices by using more methods known by both professions and by speaking the same language.

Faced with the demands of today's project environments, a reliance on heavy-weight methods such as formal models (SE), or massive documentation such as style guides (HCI) are far too expensive for the design of interactive products in any case. After all, greater cost-effectiveness and flexibility were among the main arguments for lighter, agile methods. In contrast, cheap and light-weight methods such as paper prototypes (HCI) or essential use cases (SE) are too abstract to be understood by everybody and too unstructured to guide RE towards a system specification. We therefore believe that cross-discipline agile methods are the optimum, and workable, compromise. As agile approaches already exist in both SE [4, 9] and HCI [10, 19], they can be the interface for a common and balanced software lifecycle [6].

Shared agile models such as prototypes can include and convey the attitudes of both SE and HCI. Built with adequate toolkits (see Section 3), they can be generated and changed quickly (rapid prototyping), but still be sufficiently detailed to act "as a medium of communication" [6] with stakeholders. They can preserve design experience in a visual language understood by everybody. Otherwise, "scenarios can support the design process (…) as they probe to test assumptions and stimulate creation. (They) can form the common ground between the disciplines, and furthermore need to be integrated (…) throughout the life cycle" [31,36].

In Section 2 we will therefore show how SE and HCI can be integrated under the umbrella of popular agile development paradigms, namely Extreme Programming (XP) [4] and Agile Modeling (AM) [2]. We summarize existing HCI approaches to UID and indicate their ports to XP and AM. As a result of our research, we present our agile cross-discipline user interface and software engineering lifecycle known as *CRUISER*. The lifecycle is based on the application of scenarios and prototypes as well as on the use of other methods of agile SE and agile HCI.

2. CROSS-DISCIPLINE DEVELOPMENT

In contrast to classic, heavy-weight software engineering processes like the V-Model, agile methods begin coding at a very early stage while having a shorter up-front RE phase and less documentation. Following the paradigm of XP,

implementation of code takes place in small increments and iterations, and the customer is supplied with small releases after each development cycle. During a short requirement analysis called the exploration phase, the development team writes user stories in an attempt to describe user needs and roles; the people interviewed need not necessarily be the real users of the eventual software product. In [19] we explained that XP therefore often fails to collect real user data and starts coding based only on assumptions about user needs.

Agile methods are less rigid than XP and take more care over initial requirements-gathering as they provide room for low-fidelity (essential) prototyping, activity diagrams or use-case diagrams [2, 9]. 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 user interface (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. Thus, agile development does not really qualify as user-centered design (UCD), but can function as one pillar on the way to an integrated approach.

When designing UIs, continual changes to the UI due to fast iterative design may give rise to conflicts with user expectations and learnability, cause inconsistency and finally lead to user dissatisfaction [14]. Evaluation of small releases with stakeholder participation does not ensure that the whole system provides a consistent conceptual, navigational or content model [19]. Nevertheless, the discussions about agile approaches to UID led to a movement in the HCI community, which began to reconsider its user-centered heavy-weight lifecycles [19, 14, 21, 17]. Both SE and UID have to cope with a shorter time-to-market, in which the quality of the delivered software must not suffer. The continuous shortening of development lifecycles is therefore a great challenge both for management and the methods and tools applied. Many experts, who are likely to have a home in both SE and HCI, consequently faced up to the issue by developing approaches to light-weight or so called agile human-computer interaction, for example, *eXtreme Usability* [21]. Agile approaches to HCI are the second necessary pillar for bridging the gap between SE and HCI.

The 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 [6]. In order to identify interfaces between agile SE and agile HCI, we have to highlight different approaches to UID, analyze their agile potential (Section 2.1) and their different contributions to a cross-discipline process (Section 2.2).

2.1 User Interface Design Approaches

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 such as Scenario-Based Usability Engineering [31] or the Usability Engineering Lifecycle [27] 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, etc., right from the beginning.

Constantine [14] argues that UCD produces design ideas and visual prototypes out of the assembled requirements data by a rather magical process in which the transformation from claims to design is neither comprehensible nor traceable. Such a "Black Box Designer" produces creative solutions without being able to explain or illustrate what goes on in the process. Furthermore, 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 suggests a concentration on tasks instead of on users and presents methods for an adequate requirements analysis. His usage-centered design approach takes up the basic philosophy of AM and concentrates on essential and easy to understand models such as the role model, task model or content model. With the application of these medium-weight methods HCI becomes more formal, but the simplicity of their syntax still enables collaborative design with stakeholders. The models of usage-centered design can be applied by both HCI and SE personnel. Their thoughtful and common application can be an effective commitment to a UID by engineering rather than by trial and error [14]. Constantine also suggests prototypes as a tool for requirements elicitation, but argues that filling in detail too early often leads to premature decisions. He recommends abstract prototypes [11], which can be associated with low-fidelity prototyping (see Section 2.2).

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

This is where Donald Norman's recently proposed activity-centered design approach (ACD) [28] comes in. Like Constantine, he argues that by concentrating on the user instead of on their tasks, UCD may be misleading and a time-consuming detour. But he also argues that task completion time is probably not always the silver-bullet factor that causes software to be successful. Products causing a high joy of use can 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. Based on findings in the agile-usability group, Constantine is currently reworking his usage-centered design approach and is discussing with Donald Norman: "…I am working to make activity theory more systematic and accessible to designers: to enable them to model and understand activity as quickly and agilely as practical so they can collaborate better with agile developers (…)" [39].

In Lowgren and Stolterman's book about thoughtful interaction design (TID) [26], 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, for example, captured in mock-ups or elaborated interactive (high-fidelity) prototypes. It enables manipulation, stimulation, visualization and decision making for the most promising design. The operative image is eventually transformed into a (visual) specification of the final design if it is sufficiently detailed. In contrast to UCD or usage-centered design, Lowgren and Stolterman describe design as neither a linear nor an iterative process. They identify no clear division between design and construction, but rather a continuous interaction between vision, operative image and specification. The designer wants to learn as much about the design space as possible, keeping the number of possible solutions high and narrowing the design towards the best solution as late as possible.

2.2 Integrating XP, AM and HCI

Table 2 shows a comparison of the design approaches under discussion. During development, a common design process should be able to address usability goals as well as user experience and user performance goals.

Table 2. Comparison of approaches, adapted from [13].

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

The development lifecycle is to be set up on the core methods of all the approaches presented, such as, for example, selective user involvement and participatory design (UCD, ACD), prototyping and visual thinking (TID), as well as medium-weight modeling with (use-case) scenarios or task maps (usage-centered design). 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, an aspect to which XP in particular gives top priority. Informal communication across organizational borders should be easy, and teams should share offices and common spaces such as an XP design room. As in agile SE, the writing of documentation should be kept to a minimum. Since reaching agreement on abstract notions solely through the use of text is difficult, one of the most important skills for a user interface designer is therefore the ability to make ideas visible, allowing participants to look at, feel, analyze and evaluate them. Basically, both XP and AM also attempt to visualize task cases as early as possible.

Table 3. Similarities between XP and HCI (excerpt).

XP Practice	HCI Practice
Iteration, Small Increments	Prototyping
Planning Game	Focus Groups
Story Cards, Task Cards, User Stories	Scenarios, User Role & Task Model
On-Site Customer	User-Centered Design, User Participation
Testing, Test Story	Evaluation, Usability Inspections
Metaphor	Conceptual & Mental Model, UI Metaphor

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 [4, 9]. 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" [34], 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.

Table 4. Similarities between AM and HCI (excerpt).

Agile Modeling Practice	HCI Practice
Prove It With Code	Prototyping
Create Several Models in Parallel, Model With Others	Concurrent Modeling (User Role-, Task-, Content-Model), Brainstorming
Active Stakeholder Participation	Usage-Centered Design, User Participation, Focus Groups
Consider Testability	Evaluation, Usability Inspections
Display Models Publicly	Design Room

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. The contribution of TID and ACD is the additional consideration of hedonic quality [20].

Table 5. Overall comparison of agile SE, usual HCI and special interaction design principles and practice (excerpt).

AM & XP Practice	HCI Practice	TID & ACD Practice
Document only what is regularly updated and easy to maintain	Documentation through comprehensible models and graphs	Minimalist documentation, interactive representations
Show results early, small releases	Lo-/Hi-Fi prototyping, Reuse	Make ideas visible, abstract and detailed prototypes
Communication in small teams, design rooms	Design rooms, styles guides	Informal communication, discussion
On-site customer, active stakeholder participation	User participation, collaborative design	Understanding visions through externalization
User performance	User performance, user experience	User performance, user experience, hedonic quality

3. THE CRUISER 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 [18, 24]. Although CRUISER is based on XP, which was formerly restricted to small teams with a maximum of 30 programmers [4], 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 [15] and within large organizations [25].

For the following explanation of CRUISER, we concentrate on those issues that need to be worked out collaboratively by HCI and SE experts. Parts of SE practice that are independent from UID are not mentioned in detail.

CRUISER starts with the initial requirements up-front (IRUP, see Figure 2), 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 Tables 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 authoritive design (TID, ACD).

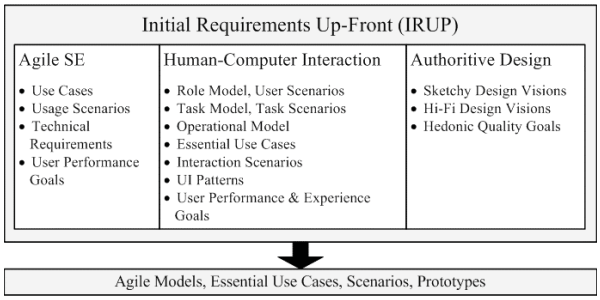

Figure 2. CRUISER initial requirements up-front.

As discussed in Section 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 [10] focuses on surveying essential information and satisfies an agile course of action due to the use of index cards (see Table 6). 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. Important aspects of the user workplace and context are documented in the operational model [10]. 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).

Table 6. UE methods for RE rated as agile.

Methods	Description
Card Sorting	Fast method to gather requirements on index cards (TCD)
Brainstorming	Similar to "Model with others" in AM
Focus Groups	Related to what XP calls the "Planning Game". Agile, if the number of participants is kept low and documentation reduced to the essentials
Task / Usage Scenarios	Established by Constantine & Lockwood [10]

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 [7, 37]. HCI patterns can be applied in correlation with the AM principles "Apply Design Standards" and "Use Existing Resources" [2]. They ease the design process by providing design knowledge and previous experience. During all IRUP assignments, users, HCI, SE and business personnel support and finalize the requirements analysis 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., [27]), 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 user interface designer to decide very early about the degree of user involvement and the necessity of more innovative solutions. The user interface

designer can have a considerable influence on balancing user performance goals, user experience goals and hedonic quality demands and can guide the IRUP accordingly.

Altogether, the outcome of the IRUP are agile models that describe user needs and task goals. Through the application of scenarios and methods for early prototyping, first design visions are created that will be refined during the next stage.

The second phase of the development process is the initial conceptual phase (ICP, see Figure 3). 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 [7] facilitates this procedure.

The development of UI and system architecture can take place in parallel as soon as a minimalist, common UI specification [12] 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 in the IRUP phase. It is very likely that highly interactive UIs will have greater impact on the design of the system architecture.

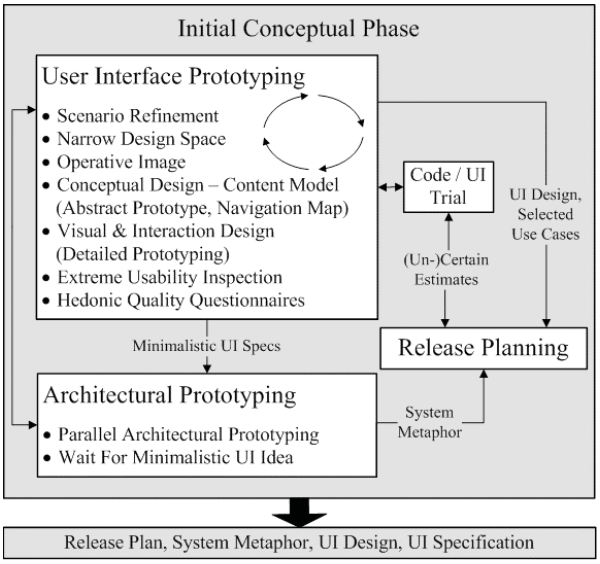

Figure 3. CRUISER initial conceptual phase.

As discussed in Sections 1 and 2, 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 [40]. 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, easy to maintain and embrace iteration. 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). Only detailed simulations allow an assessment of real UI behavior and are therefore especially necessary to cross-check the *feel* of the UI with requirements. 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 therefore also provide methods and tools for high-fidelity prototyping that overcomes most of the disadvantages mentioned in Table 7.

Table 7. Low- & High-Fidelity Prototyping, based on [32].

Type	Advantages	Disadvantages
Low-Fidelity	less time & lower cost evaluate multiple design 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 user-driven clearly defines navigational scheme 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

Hence, we recommend prototyping tools such as Microsoft Expression Interactive Designer, Macromedia Flash, iRise Studio or Axure Pro [40] that can be applied in agile environments in accordance with the demands described. 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.

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 that can be used in the ensuing construction phase. Especially when system implementation is done offshore, textual specifications for interactive systems are inappropriate [26, 40]. Visual specifications such as high-fidelity prototypes are unambiguous and can guarantee the final system matches stakeholder expectations about UID and behavior. Forwarded to the construction phase (see Figure 4), the programmer can pop-up the visual specification whenever usual specification documents do not describe the requirements coherently. Prototyping therefore minimizes the risk of making the wrong design decisions during programming and leads the way towards a winning design solution.

Through the well-balanced and thoughtful application of selected methods of RE such abstract modeling, detailed prototyping and agile usability evaluations (see Table 8), CRUISER avoids a design by trial-and-error and makes the design process move forward in a traceable and rational 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 [17]. In order to give due regard to the UI's hedonic qualities (see Table 8), which are, for example, the ability to stimulate or the ability to

express identity, we envision a design review with the help of easy to use questionnaires like AttrakDiff [20].

Table 8. Example of lightweight evaluation methods.

Method	Description
User and Expert Reviews	These review methods can be scaled and the effort can be adapted to the situation. A good educated user interface designer or usability engineer should propose a suitable scale for the method to be utilized. Reviews are less time consuming and more cost-effective than tests. Design / usability experts should be found in every agile UE design team. Results are noted on defect cards.
Collaborative Usability Inspection	Structured review of site usability. Designers, developers, end users, graphics designers and usability specialists collaborate. Cheaper and faster than usability testing. Can be conducted at almost every phase of the design process. Can be used repeatedly for iterative refinement. Finds many usability defects as efficiently as possible.
AttrakDiff	With the AttrakDiff questionnaire, a product can be rated for its attractiveness with respect to usability and design. While most evaluation studies concentrate on pragmatic quality (usability), the AttrakDiff questionnaire also considers the hedonic quality of interactive products. It can therefore be used to ask for user experience and the emotional admissibility of a product [20].

On entering the construction and test phase (CTP), coding starts (see Figure 4). 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 [17]). 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 (see Section 2.2).

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 construction phase ends with the deployment of a small release. During the first iterations, the small releases will most probably have the character of horizontal prototypes with an incrementally increasing depth of functionality.

Before the next iteration starts, each small release can again be evaluated using cheap and fast methods [17]. 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 or usage scenario. Just like user roles or task cases, 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, together with project stakeholders, can decide on the necessary measures.

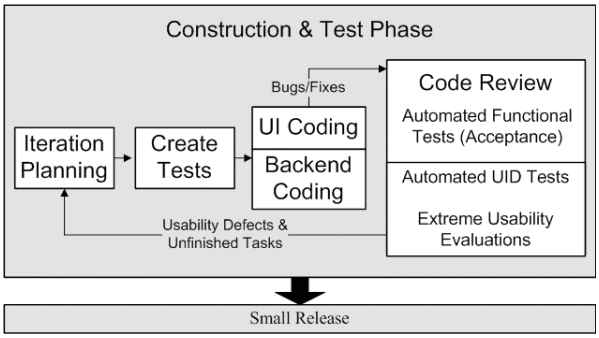

Figure 4. CRUISER construction and test phase.

The last step in the CRUISER lifecycle is the deployment and production phase. While users are working with the system, new functionality may be requested, or usability and emotional 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.

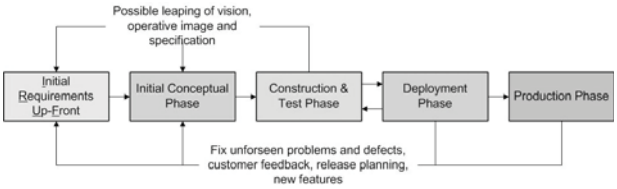

Figure 5. Overall phases of CRUISER.

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 try to 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 the XP lifecycle of Kent Beck [4], 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.

Our lifecycle is built upon industrial and research experience. Due to the defiances project managers have to face, we found it is most appropriate to join HCI and SE, having an agile perspective. Melting IT budgets, pressure of time and the demand to build better software in less time must be supported by traveling as light as possible. We did therefore not choose to just mediate both disciplines. Following our surveys, a rather radical approach best fits the demands of engineering organizations. Our next step will be to make use of this lifecycle in practical projects. Because we see great potential in our approach, we decided to publish it without evaluation in practice. We believe that by its application, software with high functional, usability and hedonic quality can be developed in less time and with significantly lower costs. We hope our

approach inspires other agile developers, encourages making use of it and motivates to share experience.

5. REFERENCES

[1] Abrams, M., and Phanouriou, C. UIML: An XML language for building device-independent user interfaces. In *Proceedings of the XML 1999*. (Philadelphia, 1999).

[2] Ambler, W. S. *Agile Modeling*, John Wiley & Sons, New York, 2002.

[3] Baxton, B. Performance by design: The role of design in software product development. In *Proceedings of the second international conference on Usage-centered design*. (Portsmouth, NH, October 2003). 2003, 26-29.

[4] Beck, K. *Extreme Programming Explained*. Addison-Wesley, 1999.

[5] Belenguer, J., Parra, J., Torres, I., and Molina, P. J. HCI designers and engineers: Is it possible to work together? In *Proceedings of the IFIP INTERACT 2003 workshop on Bridging the gap between software engineering and human-computer interaction*. (Portland, Oregon, 2003). 2003.

[6] Blomkvist, S. Towards a model for bridging agile development and user-centered design. In A. Seffah, J. Gulliksen, and M. C. Desmarais (eds.), *Human-centered software engineering: Integrating usability in the development process*. Springer, Dordrecht, Netherlands, 2005, 219-244.

[7] Borchers, J. A *Pattern Approach to Interaction Design*. John Wiley and Sons, New York, 2001.

[8] Carter, J. A., Liu, J., Schneider, K., and Fourney, D. Transforming usability engineering requirements into software engineering specifications: From PUF to UML. In A. Seffah, J. Gulliksen, and M. C. Desmarais (eds.), *Human-centered software engineering: Integrating usability in the development process*. Springer, Dordrecht, Netherlands, 2005, 147-169.

[9] Cockburn, A *Agile Software Development*. Addison-Wesley, 2001.

[10] Constantine, L. L., and Lockwood, L. A. D., *Software for Use: A Practical Guide to Models and Methods of Usage-Centered Design*. Addison-Wesley, Reading, MA, 1999.

[11] Constantine, L. L. Rapid abstract prototyping. Software development. Reprinted in S. Ambler, and L. Constantine (eds.), *The unified process elaboration phase: Best practices in implementing the UP*. CMP, Lawrence, KS, 2000.

[12] Constantine, L. L., and Lockwood, L. A. D. Usage-centered engineering for web applications. *IEEE Software, 19*, 2 (2002), 42-50.

[13] Constantine, L. L. Process agility and software usability: Toward lightweight usage-centered design. *Information Age, 8*, 8 (2002), 1-10.

[14] Constantine, L. L. Beyond user-centered design and user experience: Designing for user performance. *Cutter IT Journal, 17*, 2 (February 2004).

[15] Eckstein, J. *Agile Software Development in the Large: Diving Into the Deep*. Dorset House Publishing Co., Inc., New York, 2004.

[16] Fowler, M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd edition)*. Addison-Wesley, 2003.

[17] Gellner, M., and Forbig, P. Extreme evaluations: Lightweight evaluations for software developers. In *Proceedings of the IFIP INTERACT 2003 workshop on Bridging the gap between software engineering and human-computer interaction*. (Portland, Oregon, 2003). 2003.

[18] Grün, C., Gerken, J., Jetter, H. C., König, W., and Reiterer, H. MedioVis: A user-centred library metadata browser. In *Research and advanced technology for digital libraries 2005: Proceedings of the 9th European Conference (ECDL 2005)*. Springer-Verlag, Berlin, 2005.

[19] Gundelsweiler, F., Memmel, T., and Reiterer, H. Agile usability engineering. In R. Keil-Slawik, H. Selke, and G. Szwillus (eds.), *Mensch & Computer 2004: Allgegenwärtige Interaktion*. München, Oldenbourg Verlag, 2004, 33-42.

[20] Hassenzahl, M., Platz, A., Burmester, M., and Lehner, K. Hedonic and ergonomic quality aspects determine a software's appeal. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI 2000)*. ACM Press, New York, NY, 2000, 201-208.

[21] Holzinger, A., and Slany, W. XP + UE ® XU Praktische Erfahrungen mit eXtreme Usability. *Informatik Spektrum, 29*, 1 (2006).

[22] Jarke, M. Scenarios for modeling. *Communications of the ACM, 42*, 1 (1999), 47-48.

[23] Kazman, R., Bassl, L., and Bosch J. Workshop overviews: Bridging the gaps between software engineering and human-computer interaction. In *Proceedings of the 25th international conference on Software engineering*. IEEE Computer Society, 2003.

[24] Limbach, T., Reiterer, H., Klein, P., and Müller, F. VisMeB: A visual metadata browser. In *Proceedings of the IFIP conference on Human-computer interaction (INTERACT 2003)*. IOS Press, Amsterdam, 2003, 993-996.

[25] Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., May J., and Kahkonen, T. Agile software development in large organizations. *Computer, 37*, 12 (2004), 26-34.

[26] Lowgren, J., and Stolterman, E. *Thoughtful Interaction Design: A Design Perspective on Information Technology*. MIT Press, Cambridge, MA, 2004.

[27] Mayhew, D. J. *The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design*. Morgan Kaufmann, San Francisco, 1999.

[28] Norman, D. Human-centered design considered harmful. *Interactions, 12*, 4 (July/August, 2005), 14-19.

[29] Pressman, R. S. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 2001.

[30] Pyla, P. S., Pérez-Quiñones, M. A., Arthur, J. D., and Hartson, H. R. Towards a model-based framework for integrating usability and software engineering life cycles. In *Proceedings of the IFIP INTERACT 2003 workshop on Bridging the gap between software engineering and human-computer interaction*. (Portland, Oregon, 2003). 2003.

[31] Rosson, M. B., and Carroll, J. M., *Usability Engineering: Scenario-Based Development of Human Computer Interaction* Morgan Kaufmann, San Francisco, 2002.

[32] Rudd, J., Stern, K., and Isensee, S. Low vs. high fidelity prototyping debate. *Interactions, 3*, 1 (1996), 76-85.

[33] Seffah, A., and Metzker, E.. The obstacles and myths of usability and software engineering. *Communications of the ACM, 47*, 12 (2004), 71-76

[34] Seffah, A., Gulliksen, J., and Desmarais, M. C. (eds.), *Human-centered software engineering: Integrating usability in the development process*. Springer, Dordrecht, Netherlands, 2005.

[35] Seffah, A., Desmarais, M. C., and Metzker, E. HCI, usability and software engineering integration: present and future. In A. Seffah, J. Gulliksen, and M. C. Desmarais (eds.), *Human-centered software engineering: Integrating usability in the development process*. Springer, Dordrecht, Netherlands, 2005, 35-57.

[36] Sutcliffe, A. G. Convergence or competition between software engineering and human computer interaction. In A. Seffah, J. Gulliksen, and M. C. Desmarais (eds.), *Human-centered software engineering: Integrating usability in the development process*. Springer, Dordrecht, Netherlands, 2005, 71-84.

[37] van Welie, M., and Hallvard, T. Interaction patterns in user interfaces. In *Proceedings of the 7th Pattern languages of programs conference*. (Monticello Illinois, 2000). 2000.

[38] Willshir, M. Where SE and HCI meet. A position paper. In *Proceedings of the IFIP INTERACT 2003 workshop on Bridging the gap between software engineering and human-computer interaction*. (Portland, Oregon, 2003). 2003, 57-60.

[39] Yahoo Agile-Usability Group. Larry L. Constantine comments on 'Don Norman on Agile Development', online: http://groups.yahoo.com/group/agile-usability/message/2021, last visited 19th of March, 2007.

[40] Zetie, C. *Show, Don't Tell: How High-Fidelity Prototyping Tools Improve Requirements Gathering*. Forrester Research Inc., 2005.