Scenario-Based Usability Engineering Techniques in Agile Development Processes

Hartmut Obendorf
Software Architect
C1 WPS
hartmut@obendorf.de

Matthias Finck
CEO
Effective Webwork
finck@effective-webwork.de

CHI 2008, April 5–10, 2008, Florence, Italy.
ACM 978-1-60558-012-8/08/04.

Abstract
Improving the users' experience is a common goal of both software engineering and usability engineering. However, although practitioners of both disciplines collaborate in practice, development processes often rely on a sequential division of labor, and thus limit the effectiveness of a meeting of different perspectives. In this paper, we report on experiences we made in both academia and industry as we put an agile development process pattern to the test – combining Extreme Programming and Scenario-Based Usability Engineering, based on a blend of perspectives on equal terms.

Keywords
Usability engineering, software engineering, contextual design, scenario-based design, extreme programming

ACM Classification Keywords
H.5.2 User-centered design
D.2.2 Design Tools and Techniques

Introduction
Usability is a key quality and a prime selling point of interactive software, and often an important part of the initial design vision. However, preserving this vision until the stage of implementation remains one of the vital challenges for HCI as a discipline.

This report assumes that software engineering and usability engineering alike aim to provide the best possible user experience, yet acknowledges that in software engineering, a coherent design vision can be difficult to maintain as functionality is hierarchically decomposed in features, and that usability engineering risks to bear pipe dreams as vital technical constraints cannot be incorporated in non-technical sketches or prototypes.

An *integrated process*, with usability techniques playing hand in hand with the software engineering process is thus necessary in cases where both the technical constraints and the wholeness of the design are equally important; whether due to financial or organizational restrictions, many projects match this description.

This paper reports on our efforts to develop, teach and implement an integrated process that combines agility with the overview of work practices that UE aims to provide. A central aspect in the following pages is the use of scenarios as a focal point to connect a design vision with the more technical tasks of the programmer.

We first sketch the gap we address and argue for scenarios as tools for an integrated SW & UE approach before we describe a process pattern combining XP and Usability Engineering and reflect on the experiences we made teaching and adapting it to real projects.

Closing the Gap: Integrated U/S Engineering

Agile Software Development

The Agile Movement was formed as it became obvious that real-world projects required software developers to react to changes in requirements for organizational software [16] with high flexibility without harming the quality of the software. A central element of agile processes is their light weight [6], and their focus on the participating stakeholders and the software itself in favor of the formal definition of requirements and a strict application of well-defined processes. Code, and thus the functionality of the software become the defining element for a project. This concentration on required or implemented functionality is furthered by the negotiation processes that agile methods define: instead of processing a pre-defined list of requirements, agile methods try to convince customers to prioritize functionality, and thus to synchronize design decisions with the customer. End users, however, are not necessarily part of these negotiations, and although agile methods often mention user participation, their manner is not defined, or projected in unrealistic form (e.g. the on-site customer of XP) – with possible consequences for Usability.

Development "feature by feature" can hinder a consistent design as the development process does not define the activity of designing a whole vision of the future system ("big up-front designs" are a not seen favorably, see e.g. [1]) and feature is thus added to feature without a development scheme.

Scenario Techniques

Scenarios have a long tradition in what is nowadays called Usability Engineering, even before that term was coined – Kyng employed scenarios intensively as a communications medium for developing software [8][11]. Carroll and Rosson proposed scenarios enriched with claims documenting design rationale as a method to speed up the task-artifact cycle [5] and developed their method into a full process called Scenario-Based Development [12]. Although this process exists for some years now, no industrial application has

been documented so far. Yet, scenarios are widely used in industry as a guide for development – often in a less elaborate form and not relying on textual representation for refinement of the interaction design, nonetheless with rather promising results [18][9].

Scenarios promise to close the gap that exists in agile methodology when it comes to analyze the work context and define a consistent design vision: 1. Scenarios describing the existing state focus on problems in their context and provide a documentation of requirements that is both complete and inexact enough to remain flexible regarding details in the design. This allows the development of an almost holistic vision of a future system at an early point in development, which can then be used as a basis for the reflected negotiation of core requirements. 2. Scenarios that describe the future design include concrete activities of its users; this identifies the core functionality of the design and allows extending the description simply through writing further scenarios.

To support the negotiation process, both types of scenarios should be written with participation of the developers, and evaluated with participation of the end users. Comparing scenarios as a method with complete Usability Engineering processes, an agile approach is promoted without losing the holistic perspective.

Prerequisites for an integrated agile process
The integration of usability techniques in an agile development process first requires maintaining its agility. Both approaches we report upon try to satisfy the following prerequisites that we identified as necessary for an integrated agile process:

- Initial momentum is critical for the success of a project; to create a 'whole' picture, a coherent vision must be formed early on while design for individual features can often follow later.
- While a vision can never be complete, it must allow the extrapolation of behavior of parts of the system that are not detailed in the vision.
- Immediate feedback is vital to avoid surplus design work; systematic speculation feeds interaction with users; throw-away prototyping helps to narrow down the design space.
- Paper prototypes, informal drawings and static design stills can be more useful than a running prototype as they allow users to discuss as equals.

Experience Report

A postgraduate course at the Applied and Social Informatics Group (ASI) and the Software Engineering Group (SWT) at the University of Hamburg provided the first testing ground for the integration of scenario techniques in agile development methods [12][14]. We implemented a new process in order to test how well the techniques imported from different methodologies fit together in practice and what influence these changes would have regarding the agility of the process.

We decided to use Extreme Programming (XP) as a basis for the integrated process model. XP as proposed by Beck [1][2] is a typical agile approach that tries to address both 'inner' and 'outer' quality of software, the former through a lean development practice based on informal communication where appropriate and a rigid test-first approach, the latter through propagation of continuous negotiation with a representative of the

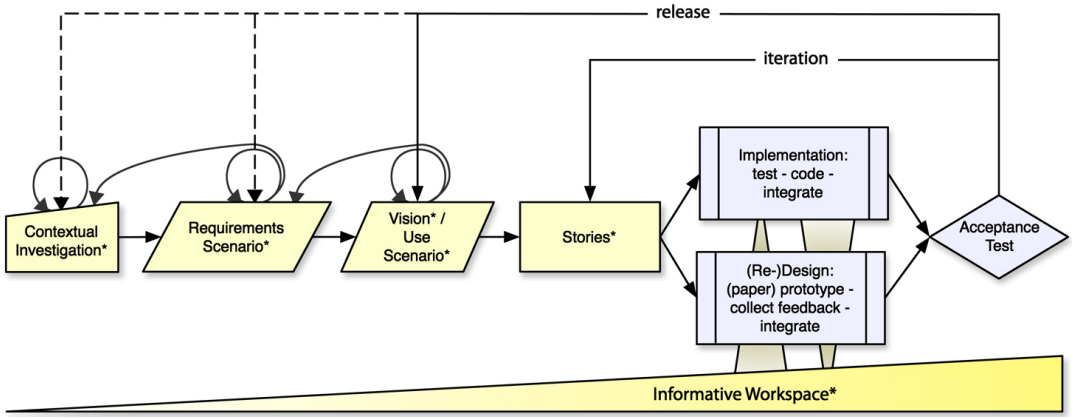

Figure 1: The XPnUE process pattern incorporates standardized XP & UE design techniques, but introduces new forms of representation (asterisks).

customer. However, requirements engineering as an activity within XP was never explicitly defined, and so the XP community has failed to establish a common culture for gathering the requirements for a software system. While XP demonstrates its strengths in iterative software development where requirements may change as a system is incrementally put into use, the question of how to devise an initial design is largely unanswered.

To address these weaknesses of XP, we adopted techniques from Scenario-Based-Engineering (SBD) [12] and Rapid Contextual Design (RCD) [10], which we changed in order to fit them into the process: We draw on Contextual Inquiry [3] as an effective method of gathering information about the use context and responsibilities and relationships of end users. Our *Requirements Scenario* leans on SBD's Problem Scenario, and the *Vision* or *Use Scenario* is a less strict version of SBD's Activity Scenario, mixing activities with sketches of partial solutions within the interface.

The XP process itself also had to be modified, e.g. our notion of *Stories* (defined as inexact requirements and "promise for conversation" by XP) includes storyboards as possible – and often necessary – elements and *Tasks* can also lead to further contextual investigation, throw-away prototyping or the advancement of the *Vision* that is used to communicate about the proposed solution with customers and end users. In addition, we used informal and incomplete information stored on notes on pin boards; this *Informative Workspace* worked both like its related XP technique (organizing and visualizing progress), and like the CD's Affinity Diagram (part of the workspace visualized the team's understanding of the problem domain and the projected solutions).

The XPnUE process pattern (see Fig. 1 and [12][13]) was implemented during the postgraduate course in a software design project, a distributed calendar system. The project team needed to understand the role that an electronic calendar system played for different university personnel (secretaries and researchers) in their daily work. With this realistic setting, we experienced a large number of realistic difficulties for an academic course, ranging from dramatic schedule difficulties to feature-demanding users to capacity shortages, and some more. With some exceptions, however, we were able to follow

the planned activities, run through two development cycles and deliver a working prototype in the end.

Industrial Experience

Motivated by our positive experiences with using the process pattern in an academic course, we consequently tried to install a similar procedure in real-world settings. We report here on impressions collected in a project that took place over a period of 6 months within an insurance company and aimed to replace an existing terminal server information system with a Web-based application for some of its users, part of the external insurance agents and the internal team that hosted the service hotline for both agents and end customers.

When we arrived at the scene, we assumed a twin role as technical and process consultants. Most of the requirements had been finalized (while details were often missing), and elements of XP were already used represent the consolidated requirements. We thus were in a typical consulting situation: without being able to replace an installed process, we changed some of its elements to better suit our intention of creating and maintaining a whole design vision:

Existing requirements were divided in XP stories and tasks, with stories representing features and screens rather than individual use cases (as the project concerned the migration from an existing system, many of the stories described legacy features and screens, and often fell short of innovation). Tasks were used to differentiate between sequential development tasks, but also to divide the labor between different teams, e.g. the Mainframe Team and the Web Team.

To create a sense of how the different features and screen would play together, we in some cases used scenarios to represent the work context and illustrate the interplay of different screens, and the interaction with the future design and the existing systems would remain in use after deployment of the new software. These scenarios thus served as a representation of contextual information and as another layer of abstraction on top of the existing stories and tasks – without the restriction of hierarchy, several scenarios could relate to a single story and vice versa (Fig. 2).

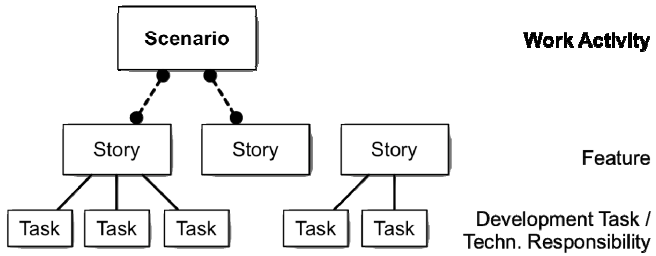

Figure 2: Scenarios connect XP Stories to the use context.

The scenarios were used by the developers to establish a common understanding of the work task with the involved stakeholders. This proved to be more complex as in the University case study as we had to deal not only with the customers (management), but also with internal project management, with the users (service hotline team) and the specialist department that provided deep knowledge about details of the different types of insurances that we put to the display. Connecting the feature requests or change requests resulting from the negotiations with the stakeholders to the work tasks also eased prioritization in some cases. It also became clear that scenarios would not be used where communication

was more straightforward and required no more than two or three people.

As it turned out, the choice of a simple textual format for representing contextual information was a fortunate one: hosting scenarios, stories, and tasks in a Wiki that provided access to most parties was very useful, yet end users often preferred to communicate via Email and the word processor of their choice – here, scenarios introduced no additional challenges.

Lifecycle-wise, we continued to update scenarios, stories and tasks to represent the current state of planning; the ability to access historic versions of the documents in the Wiki was very useful as questions about the decision process had to be answered where different interests induced some disagreement.

Lessons Learned

Applying scenario techniques in agile development

Both case studies demonstrate that UE techniques can only then have a greater impact on agile software processes when they work hand in hand with other techniques; the interface between the contextual investigation and the hierarchical decomposition of functionality is a key element of an integrated process.

In the academic setting, when we could design the process from scratch, scenarios and XP Stories were a natural match; the enriched Stories served to connect even technical Tasks to the use context, and worked together with non-programming tasks (e.g. paper prototyping) to shift the general focus of the process away from programming to design. The source code was no longer the only documentation, the running prototype no longer the focus of feature negotiation.

Yet, we also observed a focus shift in the real-world project, simply because Stories and Tasks were reordered using scenarios. Developers shared a more complete vision with the end users, and were able to make their assumptions explicit and expose them to customers *before implementation*.

An important limitation that we observed in the academic setting was the effort necessary to learn to write good scenarios – most of our students had a difficult time finding the right level of abstraction. It was also difficult to include technical limitations in the scenarios as they were largely unknown and could only be tested with software prototypes. Finally, technical details could temporarily draw the attention of our developers, so we introduced the internal role of the scenario champion who tried to raise the awareness about the context of use.

The scenarios' distance to technology was useful in the real-world project as it allowed to include end users in the requirements negotiation who had never been part of the design team and were uneasy with the terminology that had been developed internally. Scenarios could be used to describe use contexts where several features were used in conjunction, providing an overview of their interaction. And, as they served as another level of abstraction on top of the tasks, they could be used to decouple the users' requirements from the internal project organization that manifested itself in Tasks and Stories and was not always transparent to customers. The universal nature of scenarios was also useful as we were able to reuse development scenarios as a basis for Usability tests and expert reviews in another running project.

However, we also made good experiences with a conservative use of scenario techniques: the cost / benefit ratio is critical in real projects, and we first had to prove the added value of scenarios. Not in all cases, difficulties arose without the use of scenarios – especially where programmers were former users or domain experts, the negotiation process became simpler and did not need moderation.

Several gaps that existed in XP were filled by scenarios as they (a) …served as a *medium for communication* to support the negotiation of required functionality and connected features to context of use; (b) …provided *documentation* and illustrated complex domain knowledge for the development team; (c) …stored the *Design Rationale* for later use; (d) …were finally used to *conserve design ideas* and the *consolidated analysis of the work context* as the project was put on hold for another, more urgent project.

Building Skills to integrate UE and Agile Methods
We feel that for our clients – small, programmer-driven projects and large programming teams in long-existing companies without institutionalized HCI processes – a great need exists to connect to UE techniques in the development process. We found agile methods can be modified fairly easily, and scenario techniques fit well in with their methodology: *agility* – the most important quality of the process – survived. Scenarios contributed to a richer awareness of the use context in the development projects we report on here.

However, we found that the nature of the integrated approach needs more reflection. By using scenarios, and more generally, UE techniques, in an agile development process, the process is augmented to include some of the UE perspective. In consequence, software developers must master new skills and techniques – and Usability professionals must accept a role as facilitators who connect developers with stakeholders, and moderate negotiation processes only when they threaten to fail; they become part of the team on equal level. We think this is only fair: Usability is a key quality of software depending on both technical and non-technical factors; ideally, responsibility needs to be shared.

In the case of XP, software engineers must have or develop an understanding of how to understand, write, and improve scenarios together with end users; they must understand the role of different stakeholders and identify conflicts that need moderation.

While this is not a standard part of computer science curricula, we found that teaching students both software and usability engineering methods, and provoking reflection on the arising conflicts was a successful experiment: Interweaving practice from the two disciplines requires – and encourages – a deeper understanding of the underlying motives [12]. By modifying an existing process, and by changing the internal representations used to document requirements and organize development work, we found it possible to gradually introduce a basis that we could build upon as we tried to teach scenario techniques to individual developers.

Conclusion

Usability Engineering and Software Engineering often work together in order to increase software quality. However, frictions between organizational and technological constraints can lead to difficulties when the interaction of both methodologies is limited. We thus

propose to integrate both approaches, and to preserve the agility of agile development methods, we used scenario techniques within an agile XP process.

In our experience, the use of scenarios can induce the beneficial effect of connecting development tasks to the use context, and thus indirectly even decisions concerning software architecture to the question of how to best support documented use scenarios.

References

[1] Beck, Kent; Andres, Cynthia (2004): "Extreme Programming Explained: Embrace Change". Second Edition. Addison-Wesley.

[2] Beck, Kent (1999): "Extreme Programming Explained: Embrace Change". Addison-Wesley.

[3] Beyer, Hugh; Holtzblatt, Karen (1995): "Apprenticing with the Customer: A Collaborative Approach to Requirements Definition", Communications of the ACM Issue (May 1995).

[4] Bleek, W.-G., Lilienthal, C., Schmolitzky, A. (2005): "Transferring Experience from Software Engineering Training in Industry to Mass University Education – The Big Picture", Conference on Software Engineering Education & Training (CSEE&T) 2005, Ottawa, Canada; IEEE Press, pp. 195-203.

[5] Carroll, John M.; Rosson, Mary B. (1992): "Getting around the task-artifact cycle: How to make claims and design by scenario". ACM Transaction on Information Systems, 10, 181-212.

[6] Conboy, K.; Fitzgerald, B. (2004): "Toward a conceptual framework of agile methods: a study of agility in different disciplines". Newport Beach, CA, 37–44.

[7] Floyd, Christiane; Oberquelle, Horst (2003): Softwaretechnik und –ergonomie. Skript zur Vorlesung. http://swt-www.informatik.uni-hamburg.de/teaching/lvdetails.php?id=2003218121

[8] Greenbaum, Joan M.; Kyng, Morten (1991): "Design at Work: Cooperative Design of Computer Systems". Lawrence Erlbaum Associates, Inc., NJ.

[9] Hertzum, Morten (2003): "Making use of scenarios: a field study of conceptual design". Int. Journal of Human-Computer Studies, 58(2), 215-239.

[10] Holtzblatt, Karen; Wendell, Jessamyn Burns; Wood, Shelley (2004): "Rapid Contextual Design: A How-to Guide to Key Techniques for User-Centered Design", -Morgan Kaufmann.

[11] Kyng, Morten (1995): "Creating Contexts for Design". In Carroll, John (Ed.), Scenario-Based Design, 1-24. John Wiley.

[12] Obendorf, H.; Schmolitzky, A.; Finck, M. (2006): XPnUE – defining and teaching a fusion of eXtreme programming and usability engineering. In: HCI Educators Workshop 2006 inventivity: Teaching theory, design and innovation in HCI, Limerick, Ireland.

[13] Obendorf, Hartmut; Finck, Matthias; Schmolitzky, Axel (2005): "Teaching balance and respect: HCI Group & Software Technology Group at the University of Hamburg", *interactions* 12(5), 36-37.

[14] Obendorf, Hartmut; Finck, Matthias (2007): „Szenariotechniken und Agile Softwareentwicklung". In Mensch & Computer 2007: Interaktion im Plural.

[15] Rosson, Mary Beth; Carroll, John (2002): "Usability Engineering: Scenario-based Development of Human-Computer Interaction". Morgan-Kaufman.

[16] Sommerville, I. (2006): "Software Engineering" (8th Edition). Addison Wesley.

[17] Suchman, Lucy (1995): "Making work visible". Communications of the ACM, 38(9), 56ff., Sept. 1995.

[18] Weidenhaupt, K.; Pohl, K.; Jarke, M.; Haumer, P. (1998): "Scenario Usage in System Development: A Report on Current Practice". In IEEE Software, Mar. 1998, pp. 33-45.