Erschienen in: Universal Acess in Human Computer Interaction : Coping with Diversity : Proceedings, Part 1 / Stephanidis, Constantine (Hrsg.). - Berlin : Springer, 2007. - (Lecture Notes in Computer Science ; 4554). - S. 453.-462 - ISBN 978-3-540-73278-5
https://dx.doi.org/10.1007/978-3-540-73279-2_51

Agile Methods and Visual Specification in Software Development: A Chance to Ensure Universal Access

Thomas Memmel[1], Harald Reiterer[1], and Andreas Holzinger[2]

[1] Human-Computer Interaction Lab,
University of Konstanz, D-78457 Konstanz, Germany
{memmel,reiterer}@inf.uni-konstanz.de
[2] Institute for Medical Informatics, Statistics & Documentation (IMI),
Medical University of Graz, Auenbruggerplatz 2, A-8036 Graz, Austria
andreas.holzinger@meduni-graz.at

Abstract. Within the eEurope2010 initiative "An Information Society for All", development methods which enable the inclusion of the end-user become essential in order to ensure the paradigm of Universal Access. It is important to understand the end-users, their behavior, their knowledge of technology and their abilities and the context in which the applications will be used. In this paper, we combine our experiences in both Agile Methods and Usability Engineering and show that the resulting agile usability methods – however these maybe designated – are ideally suited to design and develop applications which follow the idea of Universal Access and where the end-user is having great influence on systems design.

Keywords: Human-Computer Interaction, Usability Engineering, Extreme Programming, Agile Methods, Universal Access.

1 Introduction

To achieve maximum benefits by making both useful and usable applications, it is strongly recommended to apply an usability engineering (UE) approach [1]. Some key principles of UE methods include understanding the users and analyzing their tasks, setting measurable goals and involving the end-users from the very beginning.

Based on experiences in the recent MoCoMed project and on previous work [2, 3, 4, 5], we consistently assess the role of UE in the realization of both usable and useful applications, especially in the difficult environment of an outpatient clinic.

Facing melting budgets and shorter time to markets, engineering processes in general have to come up with design approaches that can still guarantee quality in terms of functionality, reliability, user performance and user experience etc. Ensuring user interface (UI) usability with ordinary methods is then a demanding, if not an impossible undertaking. This leads to challenges for project managers but moreover to an eminent change in software engineering (SE).

With this article we want to emphasize that simple, cheap and easy-to-use development models can be a step closer to the information society for all, where people are assisted by Information Technology [6].

2 Agile Approaches to Software Development

The challenges and conflicts with development times and changing requirements are partly addressed by agile approaches to SE. Pressure of time is accommodated with less documentation, pair programming or coding from the very beginning etc., while uncertain requirements are addressed by incremental and iterative development.

However, agile software lifecycles, e.g. most popular eXtreme Programming (XP) [7] and Agile Modelling (AM) [8], lack end-user involvement and do not explicitly take care of User Interface Design (UID) issues [9]. Reasons for this exclusion are the belief that good UI quality is an effortless by-product of stakeholder feedback and the bad reputation of UE as a heavy-weight, time-consuming and expensive activity. Nevertheless, many professionals know by experience that typical agile properties, such as incremental design or refactoring, contradict UID due to problems with learnability, UI consistency etc. [9, 10]. When UE becomes part of agile SE, this helps to reduce the risk of running into wrong design decisions by asking real end-users about their needs and activities. Ultimately and contrary to its reputation, UE can decrease project costs and help to increase the acceptance of software products.

Consequently, interdisciplinary scientists came up with ideas about integrating (agile) SE and UE [3, 9, 10]. They all agree that the design of usable software demands (agile) UE methods embedded throughout the lifecycle, e.g. in terms of prototyping and evaluation. Likewise, all approaches share a limiting shortcoming of usual UE practice: none starts visual design and coding before the Requirements Engineering (RE) phase is finished. However, agile methods cannot afford waiting longer to start coding (AM: Software Is Your Primary Goal). Approaches which employ role and task models [10] during RE, can make typical UID less trial-and-error driven and more task-oriented. But models still need to be transferred into code and their abstract representation fails to show UID vision and UI behaviour, which is essential for good usability. It therefore needs to be defined and assessed, together with stakeholders, as soon as possible (UE: Participatory Design).

Also, typical set phrases of usability goals, e.g. “easy to learn” or “easy to use”, do not state anything about UI behaviour. In UE, the assessment of user performance and user experience goals, as well as the analysis of impacts of UID on the system architecture, is usually postponed to later stages of the lifecycle. When initial designs do not match expectations of stakeholders and require iteration and enhancement, these iterations slow down the progress of the overall development, which contradicts time-critical agile processes.

3 How to Build Better Software with Agile Usability Methods

Our development method is, on the one hand, based on research about prototyping [11, 12] as a bridging technique for UE and SE. On the other hand, we tie up with our previous findings about the extension of XP by principles and practice of AM and UE [9]. We also integrate our experience within the MoCoMed project (see Chapter 5).

For merging UE with XP (see Figure 1), we encourage an earlier externalization of design vision in order to make the development of usable software more effective: Interactive prototypes of specific fidelity, enable a better understanding of end-users

(AM: Active Stakeholder Participation) and their tasks, lead to a better collaboration (AM: Model With Others) and make it possible to produce better software faster. All stakeholders should be able to collaboratively discuss the look and feel of the UI from the very beginning (AM: Model To Communicate, Model To Understand) and cross-check the outcome with their requirements (AM: Prove It With Code). In order to make this course of action effective and efficient, all stakeholders need to be able to visually express and share their ideas and talk the same language [11, 13, 14, 15].

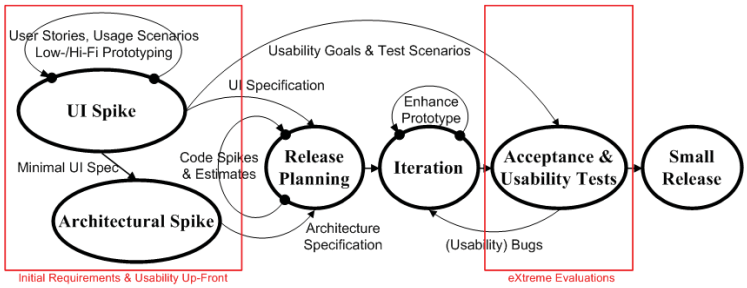

Fig. 1. The XP lifecycle extended by UE methods during up-front and test phase

Interaction and functional issues can be addressed sooner and the usability requirements realized during early stages of design by using expressive prototypes [5]. Prototypes act as discussion pieces and all stakeholders are invited to change them. Although this is very much in alignment with UE's practice of Participatory Design (PD), on the contrary we do not build throw away prototypes just in order to discover the requirements and document them. We employ the prototypes themselves as living requirements repositories. In usual UE practice, style guides are developed in order to have a reference document for designers, to share knowledge, to ensure consistency with UID standards and to save experience for future projects. A running simulation can also imply and express much of this knowledge and is less ambiguous. When expressive prototypes transport design reference along the development process, they can decrease the necessity for using abstract and extensive documents. Writing a style guide for a complex product can take up to hundreds of hours of effort. Instead of produce documents that require permanent updating (XP: Software Is Your Primary Goal), one should rather change the prototype (AM: Model With Purpose), and enhance it to a visual specification that guides development.

As illustrated in figure 1, XP's up-front is extended by adding UI prototyping. This increases time and effort, but also adds important value to the engineering process. As many UI elements as possible are gathered up-front in order to agree upon a minimalist UI specification [10] to be forwarded to system developers. It decreases the probability of later changes of core parts of the UI, which could harm UI consistency and have a delicate impact on the system architecture [9, 10]. It reduces leaping between (re-)design and evaluation. Later on, back-end system development can take place in parallel (AM: Create Several Models In Parallel) to the further enhancement of the UID prototype towards a visual specification.

4 Participatory Prototyping for Visual Specification

Prototypes for visual specification are required to have characteristics of both experimental and exploratory prototypes (see Table 1), as e.g. incorporated by functional prototypes (see Table 2). We neither build throw away prototypes, nor do we need pilot systems as the outcome of the requirements up-front.

Table 1. Approaches of prototyping, based on [2, 4, 13, 16]

Goal	Description
Evolutionary prototyping	Continually adapt a system to a rapidly changing environment; ongoing effort to improve an application
Experimental prototyping	Used to test hypotheses; try out solutions to meet requirements; communicate on technical and usability issues; gather experience
Exploratory prototyping	Used when problem at hand is unclear; express how something should work and look; design exploration understand requirements; elicit ideas and promote cooperation

Table 2. Classification of UI prototypes, based on [4, 13]

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

The low fidelity versus high fidelity debate (see Table 3) has a long history. For early stage prototyping during RE, the degree to which the prototype accurately represents the UID and – even more important – the interaction behavior, is the determining factor guiding the development process.

Abstract or low fidelity prototypes are generally limited in function but only need limited prototyping effort. They usually do not require programming skills and coding. They are constructed to facilitate discussion of UI concepts and design alternatives, rather than to model the user interaction with a system.

Therefore, low fidelity prototypes mainly demonstrate the look and rarely demonstrate the feel of an UI. They will show design direction but will not provide details about how navigation is going to work or what interaction behaviour is like [15].

Table 3. Main (dis-)advantages of low- and high-fidelity prototyping, based on [15]

Type	Advantages	Disadvantages
Low-Fidelity	less time & lower cost evaluate multiple UID concepts address screen layout issues proof-of-concept	poor detailed specification to code navigational and flow limitations facilitator-driven limited error checking
High-Fidelity	complete functionality fully interactive defines navigational scheme look & feel of final product serves as a "living" specification	time-consuming to create more expensive to develop blinds users to major representational flaws management may think it is real

Among widely known low-fidelity prototyping methods (see Table 4), paper prototyping is one of the cheapest and fastest visual techniques one can employ in a design process. It is also popular as a method for rapid prototyping [16].

Table 4. Overview on popular low-fidelity prototyping methods

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

Realistic prototypes help resolve detailed design decisions in layout, visual presentation, and component selection, as well as finding points in interaction design and interface behaviour [17]. If a developer has to present his design visions to less experienced users, executives, or a more technical audience, "a more robust and aesthetically invested prototype might be appropriate" [2].

High fidelity prototypes range from detailed drawings to fully interactive simulations (see Table 5), which show real system behaviour rather than just presenting static screens. They address issues such as navigation and work flow, as well as the matching of design with user models [15].

High fidelity prototypes should not be used for exploring design alternatives. More simple designs (AM: Use The Simplest Tools, Depict Models Simply) can externalize initial design problems better and cheaper. They provide the starting point for

discussion and requirements engineering. But after they helped to narrow the design space to the most promising solution(s), they are then too sketchy and vague to give guidance for developers (AM: Iterate To Another Artifact).

Table 5. Overview on high-fidelity prototyping methods, partly based on [13]

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

Especially when elderly end-users are involved from the very beginning, high-fidelity prototyping (see Chapter 5) adds important value to the design process. Because they are fully functional, they can provide a better basis for thorough evaluation with end-users. Although the application of low fidelity and high fidelity prototyping is comparatively effective at detecting usability issues [12, 18, 19], users are likely to prefer working with more detailed prototypes. They get a better feeling for how the product will behave and can therefore make more valuable recommendations about functionality and usability (AM: Apply The Right Artifacts).

As we want to utilize prototypes collaboratively, together with various kinds of stakeholders, the necessity of coding should be avoided, for which GUI-based UI builders are required (see Table 5). With mockups, simulation of UI behaviour (reaching from simple *mouse over* effects to animations or *zoom operations*), is impossible, for this more sophisticated tools need to be used (see Table 6).

Table 6. High-fidelity prototyping tools for visual specification

Tool	Description
Adobe Flash	Flash can be used for abstract sequences of static screens or fore full functional applications with complex interaction behavior. By adding Action Script code, any prototype can be enhanced to a full system.
iRise Studio	iRise Studio allows the creation of screens and their stringing together to storyboards. Besides standard UI components, the designer can use templates and master components to build the UI.

With appropriate tools, changes can be done quickly and stakeholders can see the impact of their suggestions immediately (AM: Rapid Feedback). If there is no lag between decision and testing, stakeholders become fully integrated and contributing members of the design task force (AM: Model With Others).

5 Sample Application

We were able to gain experience with our development model during research for the automotive industry and in several projects at Graz University Hospital. One of these projects was the MoCoMed-Graz project, first described in [20]. As a part project of the Melanoma Pre-care/Prevention Documentation, which is an important step toward fighting skin cancer, the project MoCoMed-Graz dealt with the design, development and implementation of a fully functional mobile solution to assist patient data surveys. The problem was that the paper based questionnaires had several disadvantages; including the necessity of retyping them manually into the database, most of all, they were awkward to fill out by elderly and partially sighted patients, or for example by patients with tremors. The idea of using mobile computers was to ensure that the data acquisition within the clinical department runs smoothly and also that the cancer researcher is allowed to collect data away from the clinic, for example during a survey study in an outdoor swimming pool.

The workflow: The patient reports to the central administration desk of the outpatient clinic of the dermatology department. There, they are registered via the MEdical DOCumentation System (MEDOCS) administration program into the pigmented lesion outpatient clinic. At the clinical workplace, an overview of the waiting patients, who have been already registered in the system, but not yet released by a medical doctor, can be seen. In the corresponding column on the clinical workplace, there is an indication of whether or not they have already filled out a questionnaire. Now the medical doctor or the nursing staff of the clinic can decide whether this patient is to fill out a questionnaire and/or which questionnaire to provide to the patient. After the decision to ask the patient to fill out a questionnaire, an empty questionnaire is created in MEDOCS, by pushing a button. The questionnaire in MEDOCS is registered with a definite user and a unique identification code, so that it is clearly evident that it corresponds to a version from the patient and not the medical doctor. At the terminal, the patient is equipped with a touch based Tablet PC and a code, with which he/she can login to MoCoMed and complete the questionnaire following the instructions from the touch based application. After the questions have been answered and the questionnaire is completed, MoCoMed transfers the data into MEDOCS. Further technical background of MoCoMed can be found in [20]. Further issues on touch based interface desing can be found in [21].

6 Requirements Engineering: Flexibility Is Essential

The first step was to determine both requirements and clinical context. It is necessary to differentiate between the primary end-users, the secondary end-users and the stakeholders. However, the stakeholders influence, or are influenced by, the system but are not the actual users. A precise specification of end-users is necessary (unlike to XP), which includes the typical end-user characteristics, e.g. age range, computer literacy or physical limitations (disabilities). Within clinical development it is necessary to adapt the usability of the system to the abilities of the anticipated patients, in order to enable universal access. One objective included to capture as much as possible information about the workplace and physical conditions. Actually,

the work atmosphere within an outpatient clinic is difficult, hectic and chaotic. For example, the noise level made several ideas of providing audio feedback inappropriate. Also, both low and high levels of lighting have an impact on end-users (office versus outdoor swimming pool, where sunlight is always a problem and causes glare on the screen). However, we also considered room and furniture because the characteristics of the place of installation must be studied in order to operate the system safely and comfortably. It is also important to consider user posture; in our case it is possible to use the mobile device within a total mobile setting or on an adjustable wheel table (e.g. sitting versus standing and looking down at a display). The social and organizational context is most often neglected, however this is essential for the success and is also a crucial factor to enable universal access.

6.1 Level 1: Lo-Fi Prototyping

Following our previous experience [22], we employed paper prototypes for exploring the design space. With standard office supplies, each interface element (see figure 3) has been sketched. This led to an easy creation of design alternatives, since it encouraged more suggestions due to the ease of alteration.

The intensive study of end-users by the application of paper mockups resulted in a great advantage and clear benefit. Some advantages were that the first sketches allowed immediate usability feedback (AM: Rapid Feedback). At the beginning of our project, we were able to concentrate on abstract interface concepts rather than on technological details (AM: Create Simple Content).

Fig. 2. One of our elderly patients is operating the paper mock-up

Fig. 1. Various input possibilities have been tested on paper

During the interaction, we were confronted by many problems, particularly with the navigational model and the sequence of the screens. However, as expected, it was relatively difficult to simulate real interface behavior, for example, how the interface components react upon touch or how the system converts screen states.

6.2 Level 2: Hi-Fi Prototyping

The hi-fi prototyping had the advantage that end-users both participated and were studied in a realistic setting (users could work with it directly). We found out why end-users preferred certain styles of interaction and could specify our design rationale accordingly. With more sophisticated UI representations, we were able to assess problems with screen content, i.e. form structure or their understanding of the

questions. This procedure helped to trace the source of and anticipate many problems in the early stage of RE and before the programming started. Consequently, we avoided later misconceptions as well as more iterations in systems design.

Concerning the design and the content of the questionnaire, we found that there were iterative improvements possible until the final experiments, including words, phrases and familiar - in the sense of intuition - concepts. However, we followed an aesthetic and minimalist design: none of the dialogues contained irrelevant information (AM: Create Simple Content).

7 Lessons Learned: Designing for Universal Access

By using prototypes for the visual specification of interactive systems, we entered a very interesting field of research. High-fidelity prototyping can be a partial substitute for any textual UI specification. When the described UI properties are available both visually and by code, such a prototype becomes a living design guideline and programming basis. "Whenever the programmer needs UID guidance, the prototype is fired up and the function in question is executed to determine its design" [15]. This allows an efficient and agile reuse of such prototypes [23] for further development (AM: Model With Purpose, Software Is Your Primary Goal).

Our future work will concentrate on the question to which extent functional and non-functional requirements and design knowledge can be transported in executable simulations. We will try to find out whether additional properties are required to completely replace textual documents with prototypes.

During MoCoMed, we also realized that testing methods such as Thinking Aloud were experienced as strenuous and, apart from its application with more typical end-users, took more time and preparation. Consequently, our further research will also focus on "eXtreme evaluation" [24] methods (see Figure 1) for agile development.

Altogether, we encourage developers to use a cost-efficient design of usable software systems for Universal Access based on XP, AM plus UE, in order to enable access for people who are non expert end-users.

References

1. Norman, D.A., Draper, S.: User Centered System Design. Hillsdale (NY): Erlbaum (1986)
2. Berkun, S.: The art of UI prototyping. (2000), Online: http://www.scottberkun.com
3. Campos, J.C.: The modelling gap between software engineering and human-computer interaction. In: Kazman, R., Bass, L., John, B. (eds.) (ICSE 2004), Workshop: Bridging the Gaps vol. II pp. 54–61 (2004)
4. Schneider, K.: Prototypes as Assets, not Toys -Why and How to Extract Knowledge from Prototypes. In: Proceedings International Conference on Software Engineering, ICSE-18, pp. 522–531 (1996)
5. Folmer, Eelke, Bosch, J.: Cost Effective Development of Usable Systems; Gaps between HCI and Software Architecture Design. In: Preceedings of ISD'2005, Karslstad, Sweden (2005)
6. Stephanidis, C., Savidis, A.: Universal Access in the Information Society: Methods, Tools and Interaction Technologies. Universal Access in the Information Society 1(1), 40–55 (2001)
7. Beck, Kent.: Extreme Programming Explained. Addison-Wesley, London (1999)

8. Ambler, W.S.: Agile Modeling. John Wiley & Sons, New York (2002)
9. Gundelsweiler, F., Memmel, T., Reiterer, H.: Agile Usability Engineering. In: Keil-Slawik, R., Selke, H., Szwillus, G. (Hrsg.): (Mensch & Computer 2004),: Allgegenwärtige Interaktion, München, pp. 33–42 Oldenbourg Verlag, (Mensch & Computer 2004) (2004)
10. Constantine, L.L.: Process Agility and Software Usability: Toward Lightweight Usage-Centered Design. Information Age 8(2), (2002)
11. Gunaratne, J., Hwong, B., Nelson, C., Rudorfer, A.: Using Evolutionary Prototypes to Formalize Product Requirements. In: Proceedings of ICSE 2004 Bridging the Gaps Between Software Engineering and HCI, Edinburgh, Scotland (2004)
12. Walker, M., Takayama, L., Landay, J.A.: High-fidelity or low-fidelity, paper or computer? Choosing attributes when testing web prototypes. In: Proceedings of the Human Factors and Ergonomics Society 46th Annual Meeting, pp. 661–665 (2002)
13. Bäumer, D., Bischofberger, W.R., Lichter, H., Zllighoven, H.: User Interface Prototyping - Concepts, Tools, and Experience. In: Proceedings of the 18th International Conference on Software Engineering (ICSE), Berlin, Germany, pp. 532–541 (1996)
14. Hoyos, C.G, Gstalter, H., Strube, V., Zang, B.: Software-design with the rapid prototyping approach: A survey and some empirical results. In: Salvendy, G. (ed.) Cognitive Engineering in the Design of Human-Computer Interaction and Expert Systems, pp. 329–340. Elsevier, Amsterdam (1987)
15. Rudd, J., Stern, K., Isensee, S.: Low Vs. High-Fidelity Prototyping Debate. Interactions, pp. 76–85 (1996)
16. Gutierrez, O.: Prototyping techniques for different problem contexts. In: Proceedings of the SIGCHI conference on Human factors in computing systems: Wings for the mind, pp. 259–264. ACM Press, New York (1989)
17. Constantine, L.L.: Canonical Abstract Prototypes for Abstract Visual and Interaction Design. In: Jorge, J.A., Jardim Nunes, N., Falcão e Cunha, J. (eds.) DSV-IS 2003. LNCS, vol. 2844, Springer, Heidelberg (2003)
18. Sefelin, R., Tscheligi, M., Giller, V.: Paper prototyping – what is it good for? A comparison of paper-and computer-based prototyping. In: Proceedings of CHI (2003), pp. 778–779 (2003)
19. Virzi, R.A., Sokolov, J.L., Karis, D.: Usability problem identification using both low- and high-fidelity prototypes. In: CHI Conference on Human Factors in Computing Systems, pp. 236–243 (1996)
20. Holzinger, A., Sammer, P., Hofmann-Wellenhof, R.: Mobile Computing in Medicine: Designing Mobile Questionnaires for Elderly and Partially Sighted People. In: Miesenberger, K., Klaus, J., Zagler, W., Karshmer, A.I. (eds.) ICCHP 2006. LNCS, vol. 4061, pp. 732–739. Springer, Heidelberg (2006)
21. Holzinger, A.: User-Centered Interface Design for disabled and elderly people: First experiences with designing a patient communication system (PACOSY). In: Miesenberger, K., Klaus, J., Zagler, W. (eds.) ICCHP 2002. LNCS, vol. 2398, pp. 34–41. Springer, Heidelberg (2002)
22. Holzinger, A.: Application of Rapid Prototyping to the User Interface Development for a Virtual Medical Campus. IEEE Software 21(1), 92–99 (2004)
23. Heinecke, H., Noack, C., Schweizer, D.: Software Reuse in Agilen Projekten. In: Net.ObjectDays (2003)
24. Gellner, Michael, Forbrig, Peter.: 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)