Early user-testing before programming improves software quality

John Sören Pettersson
Department of Information Systems
Karlstad University
Karlstad, Sweden
+4654 700 2553
John_Soren.Pettersson@kau.se

Jenny Nilsson
Department of Information Systems
Karlstad University
Karlstad, Sweden
+4654 700 1135
Jenny.Nilsson@kau.se

ABSTRACT

This position statement does not focus on usability although it presents data from a software up-date cycle where several usability- and user-centred methods were used. The important lesson learnt is that a better (more complete) specification before programming results in fewer errors in the code and that such a specification can be reached by user tests of interactive mockups.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications – *Elicitation methods.*

D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Evolutionary prototyping, User interfaces.*

General Terms

Design, Experimentation, Human Factors.

Keywords

Software quality, Early user-testing, Wizard-of-Oz prototyping.

1. CASE STUDY

Frequent testing of developing software can certainly increase the usability in the program. However, as we found in a case study, the method seems to continuously introduce changed or new requirements which in turn results in more complex code and thereby more errors. This case study consisted of a large update cycle of a software package in the area of decision support system for civil protection. The update involved a complete re-programming of the four largest modules. Several smaller updates had been made prior to the large update cycle, and requirements for the update had (as always) been collected from the large user groups. The organisation had routines for collecting requirements from users, client organisations, and other stakeholders.

There was thus much resemblance of their approach to principles found in user-centric approaches such as the MUST method [2]. The organisation had however recognised that usability was an issue even if the type of functions provided by the system was requested by client organisations and their employees. They had also included a continuous process of debugging using experienced users and content experts in their update cycles. One can say that the developers were not aware of the methodological critique expressed in one paper as "Close Co-operation with the Customer Does Not Equal Good Usability" [4] (cf. also [1]). Through an HCI student's exam work for the organisation, its developers became aware of the Wizard-of-Oz method by which one can test mocked up designs as if they were already implemented [3]. A more experienced Wizard (second author) was hired as a usability expert and design-aide and stayed through the 3-year update project of the software package.

Due to the size of the project, the Wizard could not pre-test every module: one of the four largest modules was not mocked up in advanced. Figure 1 shows the two user-centred processes employed in this large update project (the debugging commenced half a year after programming had started).

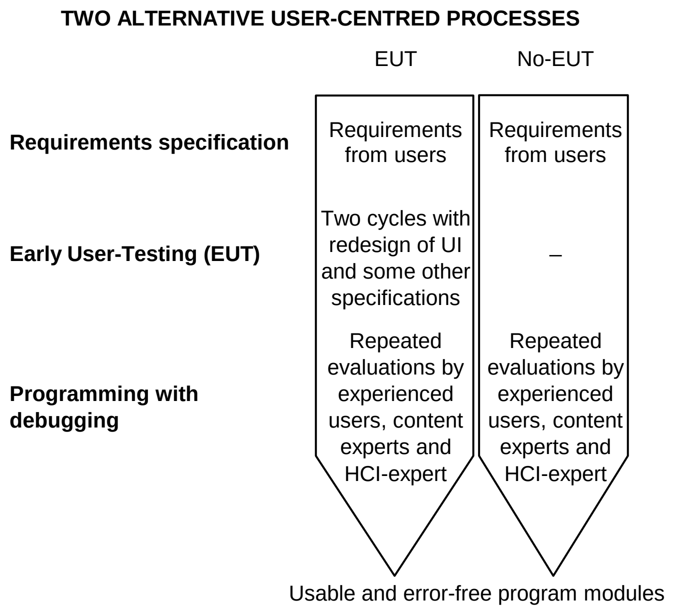
TWO ALTERNATIVE USER-CENTRED PROCESSES
EUT
No-EUT
Requirements specification
Requirements from users
Requirements from users
Early User-Testing (EUT)
Two cycles with redesign of UI and some other specifications
–
Programming with debugging
Repeated evaluations by experienced users, content experts and HCI-expert
Repeated evaluations by experienced users, content experts and HCI-expert
Usable and error-free program modules

Figure 1. Flow of work with and without Early User-Testing

2. ERROR RATES

The debugging process showed an interesting difference in the number of errors found in the module lacking pre-testing and a

Table 1. Error rates relative to program size (MB and # of files)

Program \ Error type	Prio 1		Prio 2 + 3		Priority 1,2,3	
	# / MB	**# / files**	**# / MB**	**# / files**	**# / MB**	**# / files**
Early User-tested module of 1.5 MB and 145 files	4.67	0.05	68.00	0.70	72.67	0.75
Not-EUT module of 2.0 MB and 230 files	32.50	0.28	101.00	0.88	133.50	1.16
Error rates proportionally (EUT / Not-EUT)	*0.14*	*0.17*	*0.67*	*0.80*	*0.54*	*0.65*

Note: Priority 3 was in the error reports noted as "Next version", often new requirements, while Priority 1 was "critical errors".

comparable pre-tested module. Table 1 indicates both the size (in MB) and the number of files of the two modules. Error rates are given both in relation to size and number of files. The EUT-developed module has about one-fifth of the error rate of the not-EUT module for the "Prio 1" errors (called "critical errors" in the debugging reports). In total, the error rate for the first module is only half of what was found in the second module.

It is not meaningful to compare program modules without considering the relative complexity of each module. The two other EUT-modules were only half the size of the one we select for this error comparison but contained, relative to their size, many more errors than the modules in Table 1. However, these other modules contained specific, database-related complexities and can only be used for certain comparisons (2.2).

2.1 The debugging process

The debugging process commenced nearly a year before the final launch of the new version. The debugging was conducted by three groups which were very familiar with the functional requirements: a group of very experienced users, the HCI expert, and the content managers for the different modules' databases.

The bug-finding by experienced users sometimes resulted in new requirements coming up. Interestingly, this was also the case for the debugging made by the content experts (who had not been involved in the pre-tests before programming; they had only seen and accepted the requirements specifications).

2.2 New requirements

For the first module in Table 1 there was only 4 new requirements coming up in the extensive debugging process while for the second module there was 13. This we hold to be the source of many of the other errors. When new functions are introduced into the developing process, it is harder for the programmers to maintain a clean and easily predictable code.

That early user-testing can capture many requirements was shown by a third module, smaller in size than the two modules in Table 1 (0.7 MB and consisting of only 55 files). This third module mainly consisted of a library and the content expert of this module found many faults during the debugging process: among these were in effect 24 new requirements. In the HCI expert's (i.e. Wizard's) opinion, most of the new requirements would have been possible to spot if the content expert had been included in the pre-testing, which could have been done *without the wizard setting up special test scenarios for content experts*. This is important when the Wizard-of-Oz method is used as the method incurs some extra costs when mockups have to be prepared before tests.

3. EARLY USER-TESTING

The much criticized Waterfall model for systems development, where all specifications should be settled before the laborious tasks of modelling and programming take place, admittedly has some advantages, but *only if all requirements really can be settled in advanced*. By early prototyping designers can approach this goal. In the case study, the Wizard-of-Oz method was used with user interfaces often based on previous versions of the system. What was needed was elaboration of the interaction design and uncovering interdependences between various function requirements. This was met by the WOz prototyping, which was conducted in two rounds: a first one on a rough design with 8 participants; a second one six months later on a detailed design with 5 participants. Although the interaction is 'real' in WOz experiments, the graphics can be crude in early design phases.

Setting up a WOz environment for testing is laborious as the Wizard must have control over what the user sees on the monitor (and hears from the loudspeakers), but in our research group we have developed a 'general-purpose' WOz system which we call Ozlab ,which facilitates the setting up of tests enormously (cf. e.g. [5]). A WOz set-up also allows designers to probe their own designs and find interaction bugs even before testing.

Still to evaluate is how much more costs the error-correction took in comparison with the cost for the Wizard work, but from our experiences of this project (and noting the difference in salaries between usability people and programmers…) it seems a safe bet that the EUT injected as in Figure 1 pays of very well to say nothing of how much frustration is saves.

4. REFERENCES

[1] Ambler, S.W. 2004. Tailoring Usability into Agile Software Development Projects. Maturing Usability, eds. Law, Hvannberg & Cockton. Pp 75-95. Springer-Verlag

[2] Bødker, K., Kensing, F. and Simonsen, J. 2004. Participatory IT Design. Designing for Business and Workplace Realities. MIT Press.

[3] Gould, J. D. and Lewis, C. 1985. Designing for usability: key principles and what designers think. Com. ACM 28:300-311.

[4] Jokela, T. and Abrahamsson, P. 2004. Usability Assessment of an Extreme Programming Project: Close Co-operation with the Customer Does Not Equal Good Usability. PROFES 2004 Proceedings, pp 393-407. Springer-Verlag.

[5] Molin, L. 2004. Wizard-of-Oz Prototyping for Cooperative Interaction Design of Graphical User Interfaces. Proceedings of the Third Nordic Conference on Human-Computer Interaction, 23-27 October, Tampere, Finland, pp. 425-428.