

The Role of Usability Testing in Software Quality Control: Finding Mismatches, not Defects

Paul Hubert Vossen, {voice & visual interaction design}, Kanalstraße 25,
D-70771 Leinfelden-Echterdingen, Germany, voiceandvisual@gmx.info

Abstract. Suppose, you have thoroughly tested some application system. You have found some software bugs, but all critical errors and even most cosmetic errors have been repaired, as confirmed by regression tests. Furthermore, due to the use of advanced testing specification techniques, the expected coverage of potential failures was very high. Why should you still want to do Usability Testing?

Some answers are given below. Most importantly, the notion of a mismatch between human beings and information appliances is introduced, which differs on several points from the classical notion of a software defect. This has far-reaching implications for testing management and practice, so at the end of this paper we will introduce a general reference model combining Software Testing with Usability Testing.

1 Introduction

Usability testing is necessary, when the software or product developed will actually be used by human beings, i.e. when we are talking about interactive systems as opposed to automated or autonomous systems. In interactive systems, the computer is dependent upon input from a human being for its ongoing operation and vice versa: a human being who is in charge of the computer needs specific information from the computer in order to check the progress and state of the ongoing job and to decide what to do next.

In this context, it is irrelevant, whether we are talking about experts or novices, about regular or casual use, about work-related or leisure-related activity. What counts, is that human beings will interact and communicate with a responsive device and thus become part of an overall human-computer system, sometimes even without an (easy) way out.

The crucial issue with interactive systems is to draw the right boundary between system and environment, or context. This means, taking the human being on board of the system, and to regard him or her in the role of the user of this system as an essential part of the system, albeit one that cannot as easily be modeled or changed as the rest of the man-made system. This view is not at all new, being the essence of a genuine systems approach to information engineering, including disciplines such as human-computer-interaction (HCI), industrial or engineering psychology and information or cognitive ergonomics.

Trivial as it may seem, this is all too often forgotten, neglected or downplayed in the turmoil of software and hardware development and testing, which has severe and deep implications for the so-called User Interface, because there is a strictly asymmetric relationship here: you have to adapt the computer system to the human capabilities and limitations in order for the entire system to work all right. Or, in the words of Raskin [1, p. xix]:

“If a system’s one-to-one interaction with its human user is not pleasant and facile, the resulting deficiency will poison the performance of the entire system, however fine that system might be in its other aspects.”

Failure to do this will result in software that may work fine from a purely technical point of view, i.e. according to design specifications based on technical (mainly functional) requirements, but that does not match actual human, organizational or social requirements – and thus will be useless from a pragmatic point of view – a fate shared by so many software products of the past.. This paper will address these issues in more detail through the following topics.

2 Mismatches in the user interface and interaction

Mismatches between human beings and information systems can occur at all levels of Human-Computer-Interaction (HCI): the lexical, syntactical, semantic and pragmatic level of user dialogs (cf. the famous Seeheim model, defined in analogy to the OSI model). Also, mismatches may pertain to human perception of system output (e.g., reading a screen of text, viewing a picture, hearing a voice or signal) or to system input of human behavior (e.g., keyboard typing, moving or clicking the mouse, selecting an area on a touch-sensitive screen). Plentiful examples may be found e.g. in the books and columns of computer-risk-guru Peter Neumann about cases where assumed safe technology has (deeply) failed by so-called human error [2], or in books and papers of HCI-converted computer specialists like Alan Cooper [3].

Even a technically perfect system, i.e. all hardware and software bugs have been detected and removed¹, may contain all kinds of user interaction mismatches. The basic reason and explanation is, that mismatches can only be observed and identified, if and when the interactive system is put to test with actual users, not just by software testers alone. Of course, software testers may act as stubs for the users, but they will only be able to detect those mismatches which happen to correlate to deviations from usability specifications, i.e. a type of specification rarely seen in practice, although there have been several initiatives since the mid 80's to integrate them in the software design and development cycle.

Detecting, analyzing and resolving those human-computer mismatches is indeed a real challenge, but a deeply rewarding activity too, as it tells us a lot about our own

¹ The assumption, that this is at all possible, is indeed not supported by mainstream testing experts for several reasons, one of which is that it is still very uncommon to find fully up-to-date and worked out hardware and software specification documents, so that looking for and finding bugs is very often left to the tacit knowledge and vivid imagination of testers, who should at the same time be knowledgeable in whatever application domain is being tested.

“make-up” [1]. Also users – even expert users – aren’t always aware of the mismatches they are exposed to, e.g. because they have learned to take shortcomings in software and other interactive products for granted, as evident by the market success of a range of inferior software products. But then: market success is not the same as product quality. Human-computer interaction experts have also learned to deal with that and to offer advice on how to get the right balance between (minimal) ergonomic requirements² and (maximal) economic profit of applications under development.

3 Tracking down usability problems

Great software: few bugs left, delivered just in time within budget, customer completely satisfied and willing to pay. But wait a moment, there’s a tiny problem left: users can’t work with it, they even refuse to do so, because the application is difficult to learn, to use, to adapt to their working habits.

Unfortunately, this happens more often than not, and it is documented often enough, e.g. [4]. Why does it happen? We suppose, that it is not simply that usability requirements engineering is not well mastered at this moment and not well integrated into design and development activities, despite some very useful texts aimed at software engineers [5] and despite noteworthy initiatives to get standards in place (e.g., ISO 9241 and ISO 13407).

In fact, the source of most (severe) usability problems in software is more hidden and complex: many violations of usability requirements have little to do with or can’t be easily tracked down to a single piece of code – and even if they could, their rectification often involves and affects neighboring or dependent elements, modules and other components in the user interface/interaction and beyond.

That makes usability engineering (in particular, testing) so hard to understand and to integrate with standard software design and development (testing) approaches. Presumably, it calls for a kind of expertise, i.e. knowledge and skills, that is not yet part of the mainstream professional training of software architects and engineers, as will be explained in the next topic, and less still to be observed in current practices of software design and development.

4 The profession of usability testing

Usability testing is a well-developed, well-established profession. As such, it is a sub-discipline of usability engineering³, which itself is the “hard” core of the interdisciplinary field of Human-Computer-Interaction (HCI) – with deep roots in biology, er-

² See e.g. ISO 9241 (the 17 part international standard on hardware-software ergonomics) and many related international as well as national ergonomic standards and initiatives for testing, accreditation and certification procedures.

³ See e.g. Nielsen [6] for an easy introduction to the field. More up-to-date views and topics can be found in e.g. Chen [7], especially the first three chapters.

gonomics, psychology, sociology, and some other human (behavioral as well as cognitive) sciences.

As a testing discipline, usability testing has much in common with conventional software testing approaches, but it also clearly differs on a couple of methodical and technical points. However, it is by no means new, or underdeveloped. On the contrary: all you might ever want to know about usability in general and usability testing in particular can be found on the web⁴ or by more conventional means, including professional⁵ and commercial organizations like the UPA.

So why this recent hype about usability? Let's face the current situation: a lot of young IT managers and professionals are new to the field of Human-Computer-Interaction or don't even know about its existence, still they are forced – in particular by the internet market – to take usability testing into account, hoping to find “some-where” ready-made answers to all of those “awkward” (while subjective) usability questions (it can't be so difficult, after all, it's just usability ...), e.g., by adopting self-made checklists or questionnaires, and are surprised to find out, that it doesn't work out as expected.

Indeed, all caveats and myths about software testing have their counterparts in usability testing: the challenge of test case coverage, abundance of testing methods and techniques, lack of money and/or time for thorough testing, bad integration within the design and development cycle, and so on. Even more to the point: you will need special training in models, methods, techniques and tools which you even didn't know exist when you are new to the HCI field [8].

One of the challenges of usability testing in particular is that for obvious reasons the user is not and will never be fully specified beforehand, not even during use. This makes it hard for software architects or designers to take user requirements into account and eventually makes them reluctant to adopt a user-oriented perspective. The way out of this dilemma is evident – but not always simple to implement: (1) always take representatives of users on-board of your development team, (2) hire a professional usability tester.

5 Usability testing and software quality control

Software testing without coordinated usability testing is like manufacturing a hammer and checking its suitability without ever giving it in the hands of a real craftsman and observing under different conditions, if he can use it right away, with minimal instruction and whether he still likes to use it after having used it for different purposes.

If you have designed some software merely based on usability requirements and user-centric design specifications, you are completely at the mercy of those requirement specifications as far as overall system and software quality is concerned. And how often will you find out that the specs are missing, incomplete, ambiguous, outdated? And even if you have got a nice requirements document, this does not neces-

⁴ A good starting point might be "<http://www.hcibib.org>", a professional bibliographic database for the entire field of human-computer-interaction with an excellent reputation.

⁵ Particularly noteworthy to mention in this context is the Usability Professionals Association with world-wide subchapters and its own series of yearly conferences.

sarily guarantee that the intended users will ultimately understand, learn, use and enjoy working with this “virtual” machine. A lot of human factors, capabilities as well as limitations, regarding the use of information appliances are difficult to codify in requirements or – worse – have been left open (“for the moment ...”) in the design specifications.

Thus usability testing has to be integrated and coordinated – within a generalized approach to Software Quality Assurance and Control – with software testing, following slightly different routes as in the following overall reference model:

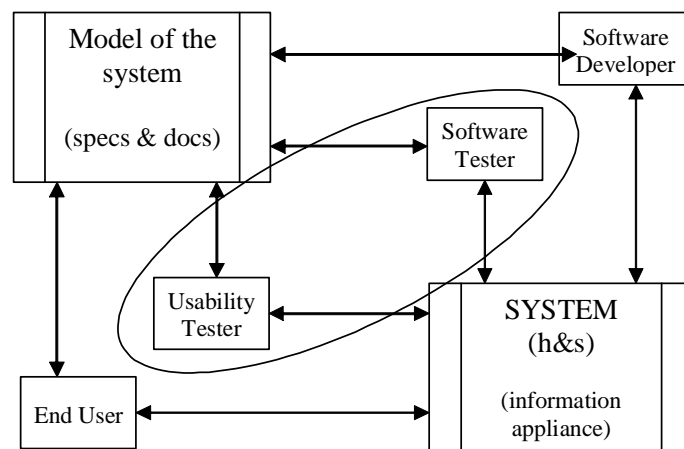


Fig. 1: Reference model for the relationship between usability testing and software testing in an overall model of software quality control

Clearly, such an approach has practical consequences for management and control of ICT development projects, in particular its testing subprojects, e.g. there must be a place for and there must be time to involve the end user in the testing activities!

The good message however is, that most if not all ingredients for such a symbiosis already exist: what are needed are more awareness and more commitment on part of software quality managers to put the necessary structures and processes in place. And the reward will be evident: end users will thank you for more usable [9] and enjoyable [10] interactive systems!

References

1. Raskin J., *The Humane Interface: New Directions for Designing Interactive Systems*, ACM Press, 2000
2. Neumann, P.G., *Computer related risks*, ACM Press, 1995
3. Cooper, A., *The inmates are running the asylum: why high-tech products drive us crazy and how to restore the sanity*, SAMS P.C., 1995
4. Cooper, A., *About Face: The Essentials of User Interface Design*, IDG Books, 1995

5. Constantine, L.L., Lockwood, L.A.D., Software for Use: A Practical Guide to the Models and Methods of Usage Centered Design, Addison-Wesley Professional, 1999
6. Nielsen, J., Usability Engineering, Academic Press, 1993
7. Chen Q., Human Computer Interaction: Issues and Challenges, Idea Group Publishing, 2001
8. Helander, M.G., Landauer, T.K., Prabhu, P.V., Handbook of Human-Computer Interaction, North-Holland, 1997
9. Norman, D.A., Things that make us smart: defending human attributes in the age of the machine, Addison-Wesley, 1993
10. Norman, D.A., Emotional Design: why we love (or hate) everyday things, Basic Books, 2004