ISSUES IN THE DESIGN OF EXPERIMENTS FOR STUDYING USER INTERACTION WITH SOFTWARE DEVELOPMENT TOOLS

Mark A. Toleman
Department of Mathematics & Computing
University of Southern Queensland
Toowoomba 4350 Australia
markt@usq.edu.au

Jim Welsh
Software Verification Research Centre
University of Queensland
St Lucia 4072 Australia
jim@cs.uq.oz.au

Abstract

In the past software engineers have been reluctant to conduct experiments with software and the users of that software. As this reluctance is overcome through, for example, incorporation of human-computer interaction as a topic in software engineering education, we believe that some of the main issues concerning experiments will become task design for an experiment, subject selection, experimental design, measurement, statistical analysis and results interpretation. We are not new in advocating the use of experiments nor in examining the issues associated with such experimentation but we do examine it in the context of one domain where the resistance to such ideas seems to have been strong—the development of software tools used by software engineers.

KEYWORDS: System design and evaluation, experiment design, language-based editors, usability engineering.

1 Introduction

Nielsen [Nie92] proposed a usability engineering life-cycle which included as one of its key elements, empirical testing. The most basic requirement was "simply to do it". Many software engineers fail to evaluate their software with the user population for which the software is developed. They simply develop systems based on their own intuition about what users need and want. In fact there seems to be a proliferation of system designs and not enough quantitative analysis of these designs. Tichy [Tic93], in analysing the content of papers published in prominent Computer Science journals (*ACM Transactions on Programming Languages and Systems, IEEE Transactions on Software Engineering* and *Communications of the ACM*), found only 15% of papers with non-trivial quantitative evaluation and no paper containing hypothesis testing. These criticisms are also valid in our particular research and development area—the development of software tools for use by software engineers. There is a natural tendency for the developers of software tools to see themselves as typical users, and to disregard the considerable variation that actually exists.

Just when to evaluate, what to evaluate and how to evaluate are non-trivial issues. Evaluation can occur at various points in a software project and it has been argued that each stage (specification, design, implementation and maintenance) could benefit from some form of evaluation [Joh92, pages 85–87]. By their very nature software development tools are complex software objects so it is likely that aspects of their design cannot be evaluated until implemented at least at some preliminary level.

What are we evaluating? Clearly this depends on the context but in our context we are interested in comparing and contrasting specific user interface issues for their suitability and usability. At times we may also want to compare different user types and their reaction to a single user interface design.

The actual evaluation procedure is also of importance. It can range from a case-study of a particular design and use by a "typical" user, to a survey of users and their responses to a product, to a classical laboratory-style experiment involving user interface issues relevant to design choices. In each case there are subjects to be selected, experimental tasks to be developed, variables to be measured and analysed, and results to be reviewed and discussed.

We are currently investigating and experimenting with some user interface aspects of sophisticated software development tools—language-based editors. Evaluation or empirical studies on these software objects are rare and, where done, they are usually informal and primarily anecdotal [NS92]. Intuition forms the basis for many evaluations. This is unsatisfactory and more rigorous methods are required if software engineering is to advance as a "true engineering discipline" [RBS93].

We have conducted three studies and two more are planned. This paper presents some of the issues with which we have had to contend and solutions we have adopted or intend to adopt.

2 Experiment 1—A Case-Study

In [TW91] we conducted a case-study of the retrospective application of user interface guidelines by examining the use of guidelines in an evaluation of the user interface of a language-based editor. Guidelines had been shown to be a useful tool for identifying important usability problems in existing software and they can be used by software engineers who are not necessarily user interface design specialists [JMWU91].

This was not a classical laboratory-style experiment. However it was an experiment in the sense that there was a subject—a language-based editor; an evaluation procedure—a comprehensive set of user interface guidelines applied by the authors; and a measurement instrument—whether or not the design of the editor complied with the relevant guidelines. Purists will argue about the use of the term "experiment" to describe such a study but these types of study are commonplace in software engineering and contribute to discipline knowledge.

2.1 Discussion

The availability of a set of guidelines as large as the Smith and Mosier [SM86] set was a considerable advantage. Their coverage included physical and functional level issues in user interface design. The comprehensive nature of the guidelines[1] meant that a wide range of user interface issues could be investigated. In many instances the guidelines simply reinforced the original user interface design decisions taken for the case-study editor. In other cases the guidelines served to confirm subsequent design choices made for the case-study editor's successor. The user interface of the case-study editor is typical of a purely intuitive approach to design whereas the new editor takes due account of current thinking and guidelines on user interface design. In its detailed representation, the user interface of the new editor conforms to the OPEN LOOK™user interface guidelines[2] [Sun89]. These "look and feel" guidelines assisted the designers with aspects of user interface design such as consistency of presentation. Other issues, however, still remained and it seemed clear that only further experimentation could provide answers to the questions that arose.

The size of the guideline set was an advantage but it was also a disadvantage. Molich and Nielsen [MN90] have noted that lengthy guideline sets are often ignored during design and the review of design simply because of the effort involved in using them. In trying to make use of long and complex documents much material goes unread, can be misunderstood or difficult to interpret [dSB90] and, consequently, is frequently not heeded [TS91]. The effort in applying these guidelines, in an "evaluation" sense, was large. Some 60 person-days were required to examine the editor from the perspective of these guidelines. This experience is not wasted, of course, and subsequent use of such guidelines for a similar evaluation task or as a design tool would be expected to take less effort.

3 Experiment 2—An Empirical Study

In a classical laboratory-style experiment we examined the usability of graphical menus versus text-based menus [TWC92]. The menus are representations of the hierarchical structure of a document and in their actual implementation are used for viewing the overall structure and as a means of selecting structural elements from a document for display or edit.

3.1 Experiment Design

For the experiment we simulated the physical interface of an existing language-based editor and used example hierarchies that were not based on program documents. We could have modified the editor itself, however, this would have had significant implications for the rest of the experiment. Firstly, the "look and feel" of the editor would have been an issue for users so we would have needed to restrict the population of subjects to those familiar with the editor and its use. This population is small. Secondly, if we had used a modified version of the editor it would have been difficult to test the issue of interest since the editor is a general editing tool and not specifically designed for conducting experiments such as this. All in all a simulated editor reduced these confounding effects and provided a vehicle for further experimentation. By using non-programming examples of hierarchies, some of which were known to the subjects, within a simulated editor we were able to use a much wider population of subjects. The nature of the experimental situation meant that control over any contaminating variables was good. The results were clear and concise if not astounding.

The type of experiment meant that each subject was only required to participate for a relatively short time, about 20 minutes. In long or complex experiments users are prone to lose interest, they complete the experiment in ways other than that intended for their treatment group [MR92], drop out part way through or simply refuse to participate in the first place. Thus a relatively short experiment time-frame and simple experimental task was an advantage. Computing professionals, as well as novice users, were readily available for the experiment. Both types of user found the experiment interesting and indicated a willingness to participate in further experiments of a similar type.

---

[1] The Smith and Mosier guidelines form the starting point for the ISO Standard 9241 [BM94].

[2] OPEN LOOK is a trademark of AT&T.

3.2 Discussion

One of the first criticisms of this study was that the example hierarchies that were utilised were artificial from the perspective of programs in a programming environment. We countered this with the view that experienced programmers should "understand" the concept of structure whether it is applied to a program or to some other entity. In a program maintenance context the programmer could be very unfamiliar with a particular program and its structure. Two of the hierarchies presented to subjects were unknown to them. Thus these hierarchies could be considered simulations of this program maintenance situation. Two of the hierarchies were familiar so they provided a simulation of a development situation where users were at least aware of the underlying structure. Thus we covered a range of hierarchies. This would have been difficult if we would have insisted on programming examples.

What is the number of subjects required to obtain "statistically significant" results? This is one of the most common questions in experimental design. We choose to use 15 experienced subjects but by using a type of design known as a change-over design [MJ84], in which subjects saw both treatments, this effectively equated to about 30 subjects. One of the main concerns that researchers have with such designs is the residual effect of a previous treatment. However the random allocation of treatments together with the padding provided by a selection of menu/question combinations and the method of statistical analysis eliminates this concern. In fact it provides better estimates of treatment differences since variation among subjects can be eliminated.

At the end of each experiment trial we asked users to indicate their menu style preference and to briefly describe their reasons for that preference. We noted a 2:1 preference for graphical versus text-based representations of the menu. Is user preference not the best way to choose the type of menu to be displayed in the editor? Clearly user preference is important but what we wanted to investigate was an issue of software ergonomics and we found no difference between the representations with respect to timing considerations. That is, it didn't matter which menu style a subject preferred they performed equally well regardless of their preference. From a design perspective, the text-based representation is simpler to implement, although from a user perspective it may be prudent to offer both views eventually. This *post hoc* style of survey is a useful adjunct to the experiment proper since it quickly provided additional information. Several subjects noted that they learned about particular hierarchies as the experiment proceeded but at this stage no analysis of these observations has been conducted. However the design of the experiment and data collection mechanism allows this concept to be investigated.

The style of the graphical menu (and for that matter the text-based menu) may also be of concern. Our view was top-down with drop down sub-menus and sub-sub-menus but could easily have been bottom-up or side-on views. These alternative menus were not examined and there is no intention to do so at this stage.

In this application the menu is more than a simple selection mechanism. One of its main objectives is to provide an overview of the hierarchy of a program or structured document. In that context so-called "walk-in" type menus would not have been suitable since they require explicit action by the user to reveal the structure. Such menu implementations are less efficient because part of the objective of the display of the menu is to provide an aid to memory [WLM94].

We might have also allowed either or both representations to scroll. This complication was not required and indeed deliberately not examined at this stage. It would have introduced extraneous variables to an otherwise uncomplicated design and in any case the issues involved would be better dealt with as another conceptual issue, scrolling. Scrolling would be easy to implement for the text-based style (and is provided in the actual editor in this manner) but given the specific experimental engine the graphical style would have been more difficult.

There have been numerous studies of menu layout and menu use and this could be considered just another. However, the menus considered here, and their context, simulate an important concept for programmers—the modular structure of a program. The results from this "limited" experiment would indicate that for the situation concerned, namely the representation of the overall structure of a program and its use as a menu for module selection, a graphical representation is no more efficient than a more simply implemented text-based representation. However, based on stated user preference, we may have to consider providing both representations in our software development tool at some stage.

4 Experiment 3—A Predictive Model

An alternative to intuition for evaluating user interface designs is the predictive modelling approach [EE89]. Here the idea is to predict the performance of humans interacting with computers in a similar way to predictions based on engineering models. The GOMS/KLM (Goals, Operators, Methods and Selection of Methods/Keystroke Level Model) approach to predictive modelling [CMN83] is an attempt to produce a formal model of the human-computer interaction task. Thus a model can be built which can be used to help evaluate various user interface designs even before prototyping.

With respect to language-based program editors, two basic paradigms for editing have been recognised. In the *tree-building* paradigm the user is only allowed operations which ensure the structural correctness of the program tree at all times. The alternative is the *(text) recognition* paradigm in which the user manipulates the displayed representation in textual terms and the editor parses the changes to validate them and deduce the program tree. Often software engineers ignore both of these and use plain text editors. In this study we applied the techniques of the predictive modelling approach to an evaluation of these editing paradigms [TW94].

4.1 Discussion

There are several problems with the GOMS/KLM approach. Firstly, it assumes that users are skilled in the tasks to be done. This is less of a problem when considering software engineers using a software development tool than it is for other situations where unskilled or casual users are required to use tools and techniques about which they have no expert knowledge.

The main difficulty with the GOMS model is in obtaining its components for any particular task and in applying the approach to complete systems. In some instances the identification of these components is simple. For example a task such as *document correction after mark-up* could involve cut and paste operators to achieve the stated goal which might be implemented using mouse-based menu selections or keyboard-based commands. Another task such as *document comprehension prior to correction*, however, is much more abstract involving the user's knowledge and the user's cognitive skills as well as appropriate perceptual and motor skills for assisting with browsing and searching. It may be difficult or impossible to completely model this comprehension/correction task using GOMS/KLM techniques since it is not just the command structure that is in use—the form of the display also affects the time of tasks and error frequency.

In this study we restricted the tasks to program development and maintenance in Pascal. For program development we considered a range of tasks including input of procedure and function "stubs" and various language construct types. Program maintenance tasks included trivial alterations to code as well as relatively complex structural changes. The actual tasks chosen were significant. In all program development examples text-recognition provided the best time estimate. In more than half the cases text-recognition was more than 20% faster. Using a plain text editor was relatively slow. The difference between tree-building and text-recognition was mainly attributable to the extra memory operators. A plain modeless text editor proved best for program maintenance in almost all instances. The text-recognition paradigm was less efficient than tree-building in situations requiring the user to terminate insertion of text and where the tree-building paradigm allowed simple "modeless" text changes.

The KLM proved to be a useful design tool. It not only assisted in comparing relevant design options but also indicated inadequacies of current editor implementations. However, one of the KLM's main problems seems to be the difficulty of defining where memory operators should be placed. This is non-trivial, needing careful thought and probably experimentation for some design situations. A controlled experiment using software engineers is planned. It will attempt to examine just where these memory operators should be placed for these paradigms and typical unit-tasks as given here.

5 Experiment 4—A Planned Empirical Study

As for the previous experiment we are interested in comparing the two basic paradigms for editing: tree-building and text-recognition. Unlike the previous experiment, however, we shall conduct a laboratory-style experiment.

5.1 Discussion

For this experiment we plan to use an existing text-recognition editor which has been enhanced to allow menu selection. This enhancement effectively allows the editor to simulate template-based input without the need to compromise its basic editing paradigm.

What do we want to measure and how will we measure it? We are interested in the relative efficiency of the two treatments. Quantitative performance levels could best be determined using examples based on programming tasks. Here we can set code development or maintenance tasks for the subjects and measure error frequencies and task completion times. But the actual choice of task is likely to be significant and some tasks will fare better under one paradigm than another. Some such tasks have already been developed in the GOMS/KLM experiment. The preference a user has for one paradigm over the other is also of interest—as in the hierarchical menu experiment.

The type of user we are interested in here is special—a software engineer. This was also the case for the hierarchical menu experiment. However, unlike that experiment we now need users who are familiar with the "look and feel" of the editor itself and its use. As noted earlier this group is small. In practice we shall probably have to be content with computer science students (fourth year honours) who, although highly intelligent, probably have little or no industry experience. What they do have, however, is experience with the experimental platform and programming experience in several programming languages including Pascal. As in the hierarchical menu experiment, already described, such a small sample requires careful thought about the experimental design and a change-over design similar to the one already used is contemplated. This is not an uncommon solution where experimental subjects are expensive or difficult to obtain but it may be necessary to replicate the study at a later stage to confirm the results. The small sample size may necessitate the use of nonparametric rather than parametric statistical analysis with a consequent loss of power and interpretative ability.

User reaction to the different paradigms is also of importance but complex languages, like the specification language Z, may prove better for measuring this than more typical programming languages. The "richness" of such a language almost dictates that the user must have some assistance in the preparation of documents. This is readily provided by menus of selectable constructs and operators.

6 Experiment 5—A Planned Empirical Study

The concept examined in Experiment 2 was part of one described in Welsh and Toleman [WT92] under the heading **block-oriented program display**. Another conceptual issues addressed in that article which is to be the subject of our further experimental study is now discussed.

Detail suppression is a means of avoiding prodigal consumption of display space when showing a program listing by, for example, compressing or eliding some program details. With block-oriented program display and elision of nested block detail, this is not a problem for relatively small program modules, but as the modules get larger scrolling within a window is still a necessary feature. An alternative is *suppression by structural distance* [BW86] which compresses text furthest from the user's focus of attention (the highlighted text) in a fish-eye lens style.

For display of a module in a window which is too small for its preferred presentation, therefore, there are two treatments: scrolling or structural suppression. In both cases the user is interested in viewing and/or editing a relatively large module of program code.

6.1 Discussion

Users are familiar with the concept of scrolling as it is common in graphical user interfaces and indeed in some character-based systems but they are unlikely to be familiar with the concept of structural suppression. This makes the choice of experimental tasks difficult. For some tasks scrolling is naturally favoured while for others structural suppression would win out. Scrolling may be a better mechanism when the user wants to find the number of occurrences of a variable in a construct or when searching for a particular object. Structural suppression will almost certainly be better where there is a major space separation between the items of interest and structural suppression, by its nature, allows them to be viewed in the one window. Tasks could be chosen that deliberately favour one treatment or the other. Is this appropriate experimental design or should "neutral" tasks be chosen?

What do we want to measure and how will we measure it? Again we are interested in the relative efficiency of the two treatments. On a quantitative level we can set tasks (a non-trivial activity itself) and compare completion times. We can also measure comprehension by asking appropriate questions about program text examined by subjects.

In both treatments the "shape" of the displayed text, as it is either scrolled or the user alters their focus of attention, changes but the extent to which this affects the user's ability to read and comprehend program code is unknown and needs to be determined. The preference a user has for one treatment over the other is also of interest—as in the hierarchical menu experiment.

The arguments on user type are the same as for the previous experiment. However, actual experience with the editor and its interface is less critical here than for the editing paradigm experiment since the task is essentially one involving browsing and comprehension. A simulated interface may be sufficient and could require a less "elite" group of participants.

Scrolling basically comes for free as part of the windowing software. Structural suppression certainly has a price but it may be worth it—we won't know until it is implemented and evaluated.

7 Conclusion

We have discussed three experiments that we have conducted and two that we intend to conduct.

The first experiment, a case-study, although time consuming was relatively easy to carry out. It focused our attention on user interface issues and indicated areas needing further investigation. Likewise the predictive modelling exercise was useful in that it confirmed our view on the efficiency of various editing paradigms. It also pointed to gaps in our knowledge and the need for further experiments.

The hierarchical menu experiment embodied a simple concept for experimentation, a good population from which to select, controlled conditions, simple tasks, and relatively easy execution and analysis. In contrast the two experiments that are planned have a number of problems. The selection of appropriate tasks is one issue. To be fair to all treatments, what type of tasks do we choose? The population of potential subjects is more restricted since the users must be familiar with the software to be evaluated before we can evaluate certain aspects of that software. What is most important to measure? Is it time to complete tasks or more subjective measures and if the latter what biases do subjects bring to the experiment based on previous knowledge and experience?

The usability of software is critical and empirical testing of software is essential but it sure isn't easy. We don't have many answers just lots of questions but attempts such as these will surely build an experience base for future research work in this area.

References

[BM94] N. Bevan and M. MacLeod. Usability measurement in context. *Behaviour and Information Technology*, 13(1/2):132–145, 1994.

[BW86] B.M. Broom and J. Welsh. Detail compression techniques for interactive program display. In G.W. Gerrity, editor, *Ninth Australian Computer Science Conference*, pages 83–93, 1986.

[CMN83] S.K. Card, T.P. Moran, and A. Newell. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

[dSB90] F. de Souza and N. Bevan. The use of guidelines in menu interface design: evaluation of a draft standard. In D. Diaper, D. Gilmore, G. Cockton, and B. Shackel, editors, *Human-Computer Interaction—INTERACT'90*, pages 435–440, 1990.

[EE89] R.E. Eberts and C.G. Eberts. Four approaches to human computer interaction. In P.A. Hancock and M.H. Chignell, editors, *Intelligent Interfaces: Theory, Research and Design*, pages 69–127. North Holland, Amsterdam, 1989.

[JMWU91] R. Jeffries, J.R. Miller, C. Wharton, and K.M. Uyeda. User interface evaluation in the real world: A comparison of four techniques. In S.P. Robertson, G.M. Olson, and J.S. Olson, editors, *Human Factors in Computing Systems—CHI'91*, pages 119–124, 1991.

[Joh92] P. Johnson. *Human Computer Interaction: Psychology, Task Analysis and Software Engineering.* McGraw-Hill, London, 1992.

[MJ84] G.A. Milliken and D.E. Johnson. *Analysis of Messy Data*, volume 1: Designed Experiments. Lifetime Learning Publications, Belmont, California, 1984.

[MN90] R. Molich and J. Nielsen. Improving a human-computer dialogue. *Communications of the ACM*, 33(3):338–348, 1990.

[MR92] S. Meyers and S.P. Reiss. An empirical study of multiple-view software development. *ACM SIGSOFT Software Engineering Notes*, 17(5):47–57, 1992. Also in H. Weber (Ed.) ACM SIGSOFT'92: Fifth Symposium on Software Development Environments, Washington D.C., 9–11 Dec.

[Nie92] J. Nielsen. The usability engineering life cycle. *Computer*, 25(3):12–22, 1992.

[NS92] L. Neal and G. Szwillus. Introduction. Structure-based editors and environments. *International Journal of Man-Machine Studies*, 37:395–397, 1992.

[RBS93] H.D. Rombach, V.R. Basili, and R.W. Selby. Experimental software engineering issues: Critical assessment and future directions. In H.D. Rombach, V.R. Basili, and R.W. Selby, editors, *Lecturer Notes in Computer Science 706*, pages v–xiii. Springer-Verlag, Berlin, 1993. Experimental Software Engineering Issues: Critical Assessment and Future Directions, International Workshop, Sep 1992.

[SM86] S.L. Smith and J.N. Mosier. Guidelines for designing user interface software. Technical Report EDS-TR-86-278, 1986. NTIS AD A177 198.

[Sun89] Sun Microsystems, Inc. *OPEN LOOK Graphical User Interface Functional Specification.* Addison-Wesley, Reading, MA, 1989.

[Tic93] W.F. Tichy. On experimental computer science. In H.D. Rombach, V.R. Basili, and R.W. Selby, editors, *Lecturer Notes in Computer Science 706*, pages 30–32. Springer-Verlag, Berlin, 1993. Experimental Software Engineering Issues: Critical Assessment and Future Directions, International Workshop, Sep 1992.

[TS91] L. Tetzlaff and D.R. Schwartz. The use of guidelines in interface design. In S.P. Robertson, G.M. Olson, and J.S. Olson, editors, *Human Factors in Computing Systems—CHI'91*, pages 329–333, 1991.

[TW91] M.A. Toleman and J. Welsh. Retrospective application of user interface guidelines: A case study of a language-based editor. In J.H. Hammond, R.R. Hall, and I. Kaplan, editors, *People Before Technology—OZCHI'91*, pages 33–38, 1991.

[TW94] M.A. Toleman and J. Welsh. A keystroke analysis of language-based editing paradigms. Technical Report 5-94, Software Verification Research Centre, University of Queensland, Brisbane, Queensland, 1994.

[TWC92] M.A. Toleman, J. Welsh, and A.J. Chapman. An empirical investigation of menu design in language-based editors. *ACM SIGSOFT Software Engineering Notes*, 17(5):41–46, 1992. Also in H. Weber (Ed.) ACM SIGSOFT'92: Fifth Symposium on Software Development Environments, Washington D.C., 9–11 Dec.

[WLM94] P. Wright, A. Lickorish, and R. Milroy. Remembering while mousing: The cognitive costs of mouse clicks. *SIGCHI Bulletin*, 26(1):41–45, 1994.

[WT92] J. Welsh and M.A. Toleman. Conceptual issues in language-based editor design. *International Journal of Man-Machine Studies*, 37:419–430, 1992.