PAIR-COLLOBORATED USABILITY STUDY OF RABBIT IN REQUIREMENTS MODELING AND GENERATION

HALIL ERHAN AND FATIHA DJEBBAR
SIAT Simon Fraser University Surrey, Vancouver BC
hierhan@sfu.ca
CIT U.A.E. University, Al-Ain, P.O. Box: 17555, U.A.E.
fdjebbar@uaeu.ac.ae

Abstract. The traditional manual methods of programming in architectural design have definite disadvantages that could, in part at least, be overcome by computational support. We have developed RaBBiT, a support tool for programming that can be easily adapted to various programming styles end terminologies. In the present paper, we report on the evaluation of its usability in requirements modeling by expert designers. The evaluation is performed through the following steps: (a) measuring usefulness of RaBBiT and its effectiveness in requirements modeling and generation and (b) assessing the usability of RaBBiT by applying 'pair-collaborated usability evaluation using heuristics'.

Introduction

RaBBiT (Requirements Building for Building Types) is a computer-based tool aiming to assist programming and requirements management phase of architectural design. We have discussed RaBBiT in earlier papers (Erhan, 2003; Erhan and Flemming, 2004; Erhan and Flemming, 2005). In the present paper, we focus on the usability evaluation and assessment of the system. Since RaBBiT suggests a different approach than current information management tools used in architectural programming, which are usually built on some general purpose office productivity tools, we would like to determine the response from architectural programmers towards its usability, effectiveness, and acceptability. Our initial observations with RaBBiT motivated us to pay special attention to the challenges posed by its generality, (computational) programmability and interactivity.

The system is meant to be used by domain experts with little or no experience in computer programming. RaBBiT, while aiming to address fundamental requirements management issues, focuses on usability and user interface design. Despite our present emphasis on RaBBiT's usability, we have to briefly introduce the functionality because the user interaction with RaBBiT has to be understood in its terms.

Users and Functions

The primary users of RaBBiT are practitioners with a particular interest in architectural programming for specific building types. The secondary users are other computer-aided design reasoning systems and clients who want to utilize a model or a generated program in their own domain of interest—

such as layout generators or budget planners. Specifically, a primary user may play one of two roles (Figure 1). The architectural programming knowledge modeler (APM) models interactively with RaBBiT the process to be used when generating a program for a recurring building type independently of project specifics. The APM is able to edit interactively the resulting programming knowledge model (PKM) and to save it persistently. When generating a PKM, an APM can use any preferred terminology and model any programming process consistent with G-MEA (Erhan, 2003). The architectural program composer (APC) is able to retrieve a PKM to compose—with RaBBiT's assistance—a program for a particular project calling for the respective building type. After a program has been generated, the APC can modify and save it.

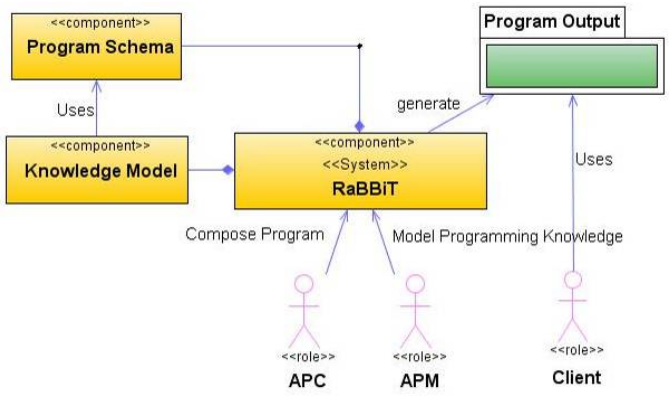

Figure 1. RaBBiT: Knowledge modeling and program output

Select User Interface Paradigm and Metaphor

RaBBiT's GUI adapts the direct manipulation paradigm as introduced by Shneiderman (1982; Shneiderman and Plaisant, 2005; Hutchins, Hollan, and Norman, 1986). The paradigm lets users manipulate directly (domain) objects visible on the screen and observe the consequences of an action (immediate feedback). This interaction style appears appropriate for RaBBiT not only because of its inherent advantages: Designers, in particular, are trained to create and organize visual information and may therefore take naturally to the direct manipulation of graphical objects.

In RaBBiT's GUI, the model-world metaphor guided us specifically in our handling of the possibly complex associations among requirements in a PKM, which we display as a graph. Its nodes represent requirements and global parameters; dependency and relational associations are shown as lines connecting nodes. Users are able to visualize and manipulate a PKM through the nodes and lines of the graph. Parametric associations, on the other hand, are too complex to be shown in the same graph. We therefore handle parametric associations through separate tables embedded in the nodes that can be modified in a spread-sheet-like fashion, which can be viewed as a sub-metaphor. Figure 2 shows the display of a PKM in RaBBiT. Component nodes appear as small, floating tables with a yellow (light) header and rows showing the associated constructs in the form of parameters or attributes. Global constructs are also represented as floating tables, but with a blue

(dark) header and exactly one parameter. A dependency is shown as an arrow and, if needed, an attached box containing a dependency condition. Other associations are shown with a light gray line connecting the two components involved and an attached label indicating association.

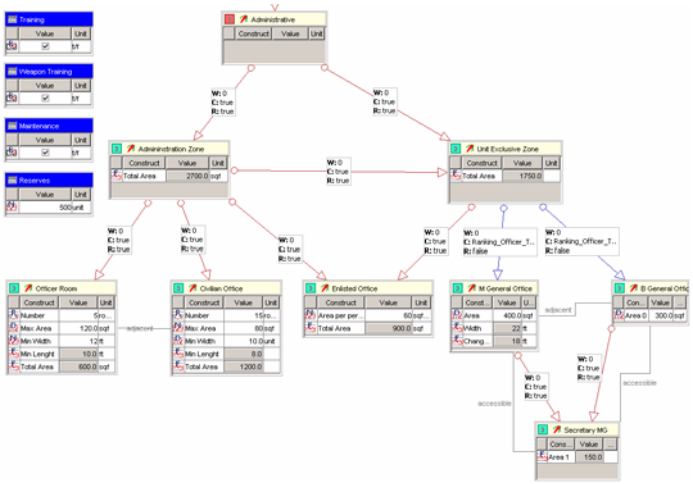

Figure 2. UI Representation of a configurable and parametric graph

Use Cases

The APM and APC accomplish their respective goals through a series of actions or operations performed in interaction with RaBBiT that involves a host of interface objects or widgets, from commands accessible through menus or tool boxes to the various dialog boxes and system messages needed to accomplish individual tasks. These are of concern of use case model of a system. Use cases that describe how the application under consideration makes the required features available to the actors. A use case describes a "sequence of actions an actor (user) performs using a system to achieve a particular goal" (Rosenberg, 1999). We may group the use cases defining the functionality of RaBBiT under four categories (Table 1).

TABLE 1. Crucial use cases of RaBBiT

Session Control	Knowledge Modeling	Program Generation
Start new session Save a PKM/program Close a PKM/program Exit session	Define category levels Create (insert) a component Insert a construct into a component Create a global construct Insert an association between two components Modify a component/ construct/association Remove a component /construct/ association	Input project-specific information Modify global parameters Generate a program Display sample program

Usability Testing

The three main objectives of this research are achieved by an experiment using heuristic evaluation methods (Nielsen, 1995). The experiment addresses usefulness and effectiveness of the system in achieving a given requirements specification tasks that focus on the user interface evaluation. We think that this approach is appropriate since a strict quantitative evaluation as prescribed in low-level usability evaluations of the user interfaces centers around the details, such as the number of key-strokes, while our concern is to find out whether the usability of the system is achieved from the user's perspective. For the usability and usefulness tests, we designed a formal three-phase experiment. Three phases are (a) design of the experiment; (b) conducting experiment and data collection; and (c) evaluation of findings.

DESIGN OF USABILITY EVALUATION

The usability is measured by five main factors by its traditional definition (Dix et al., 1998; Lauesen, 2004; Davis, 1989): ease of learning, task efficiency, easy of remembering, understandability, and subjective satisfaction. The objective of this experiment, therefore, to measure to what degree RaBBiT satisfies these factors. The process designed and followed during the evaluation had three groups of activities (Figure 3). The first activity involved the introduction of RaBBiT through two distinct tutorials: Generic and Domain Specific. The generic tutorial presents the underlying structure of the RaBBiT. The domain specific tutorial is intended to make the participants aware of how this structure maps on design requirements specifications through parametric and component-based configurations. In addition, the generic tutorial is meant to be domain-neutral to test if the designers can form a mental model of RaBBiT when isolated from domain knowledge.

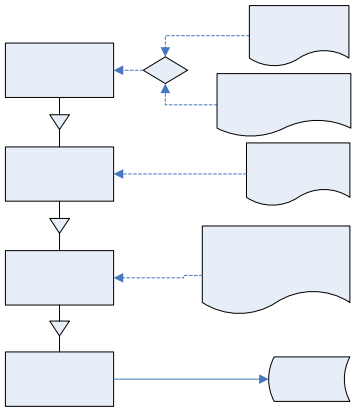

Figure 3. Process of the usability test conducted

The tutorial session was followed by the second group of activity. The participants are given a design requirements logic for a small healthcare clinic (facility) and were asked to use RaBBiT to model and generate architectural design requirements. After the participants completed the first two group of activities, they were asked to assess the usability of RaBBiT by filling four types of short questionnaires, which are used to measure UI

satisfaction, usefulness and ease of use; system usability; after scenario; and practical heuristics. The different questions in each questionnaire specify and measure the four usability factors.

EXPERIMENT ENVIRONMENT AND SETTING

The evaluation took place in an office with a desk large enough to accommodate two participants and one observer around (Figure 4). A special computer supporting two cloned-screens is used: the first screen is used by the participants and the other showing the same view as the first screen is used by the observer for recording. This setting provides a comfortable distance between the participants and the observer; and though to be less intrusive than having only one screen. The activities that the participants performed during testing were also recorded using a digital camcorder with a wide-angle lens to gather data for further analysis.

Figure 4. Usability experiment setting

TUTORIALS AND TASK PERFORMANCE

In the first phase the generic tutorial is given to the participants. The generic tutorial showed the main components of RaBBiT and its functionality. In order not to cause any sudden and unexpected reaction from the participants we intentionally did not emphasize that the tool can assist programming phase in design. Instead, we expected the participants to discover RaBBiT's functionality through a generic tutorial and use these features to implement the given clinic program. In addition, the analogy for how we learn word processors seems appropriate: we don't learn how to write a letter or a report, but we learn how to use the tool so that we can write any document we want using the word processor's features. The domain-specific example is followed to clarify abstract aspects of RaBBiT.

In the second phase, the participants were asked to build a knowledge model for a simple clinic and generate programs for different clinics using the model. The clinic problem is formulated as follows: The requirements for a clinic are derived from number of doctors and their specialty that will work in that clinic. For every two doctors, there must be a nurse available as part of the staff. Using this staffing pattern, we arrive at lower level spatial requirements, such as the number of exam rooms must be equal to the number of doctors. In addition, the clinic has a nurse room with 2 square meter area per nurse. Other spatial requirements added in a similar fashion in the sample problem (see Erhan 2003). The participants are asked to model

these requirements and generate programs for clinics with different number of doctors. Please note that this is a simplified clinic definition and does not reflect a realistic case.

PAIR-COLLABORATED USABILITY EVALUATION

We had 8 participants to test our application. The participants had average 10 years of experience in architectural design and programming. Two participants were teaching courses on architectural programming and design requirements specifications. However not all participants used a specialized computer-supported architectural programming tool before. The three participants commented that they are not aware of such a tool yet.

Inspired from pair-programming principles, we designed a pair-collaborated evaluation and limited the design requirements portions of the application domain (Beck, 2000; Wildman, 1995). Among these principles, pair-collaborated evaluation as in pair-programming proved specifically helpful in our tutorials (Figure 5). Pair-programming is accomplished by two people looking at one machine, with one keyboard and one mouse. The person with the keyboard and the mouse focuses on the best way to implement the portion of the system at the same time the other partner keeps track of the overall design and the more strategic aspects of the tasks.

Figure 5. Pair collaborating in evaluation of RaBBiT (from pilot testing)

In the experiment, a pair typically worked on a specific sequence of use cases given in the tutorials. A pair provided with general program requirements is asked to enter them as specified in the tutorial. If a pair thinks they accomplished the task, they progress further in the tutorial. However, note that the pair's evaluation of the system goes in parallel with this process. The pairs typically are given some clues that allowed them to get started, and while they were at work, assisted by the observer taking the feedback from pairs into account and when requested. The pairs also worked in a highly concentrated fashion to complete the sample (health clinic) design requirements using RaBBiT. At certain times, the participants in one pair changed roles.

The tutorial and problem sessions were followed by individual evaluation in which each participant completed the online questionnaires tailored to measure specific usability features of RaBBiT, after which the individual responses were compiled through online submissions. The analysis phase was proceeded by two weeks of data-formatting and -compiling. The participants made comments on the tool during the tutorial and problem phase. Some of these comments were recorded as a result of the 'pair's

discussions' and some reported directly to the observer. We believe this approach worked more effectively than 'think-aloud' style experiment; since the dialogs have been natural and raising new questions and invoking new ideas by the participants.

SURVEY QUESTIONNAIRES

We chose to adopt some standard usability tests rather than formulating a new test from start. We did not have enough time to validate a new questionnaire, there already existing questionnaires in the literature specifically designed for usability testing that their validity and reliability established. We found the tests shown in Table 2 appropriate for our purpose and the usability aspects of RaBBiT that we intend to measure. The survey questions selected are based on the usability guidelines and principles described in the literature (Shneiderman and Ben, 2005; Nielsen, 1995; Dix et al., 1998).

TABLE 2. List of questionnaires used and their grouping

Questionnaire	Purpose	Usability factors addressed
Perceived Usefulness and Ease of Use	Perceived usefulness and easy to use are measured.	Learnability Understandability Subjective satisfaction
After Scenario Questionnaire	Questions users reaction to a system after performing a given set of tasks	Task efficiency and effectiveness
Practical Heuristics for Usability Evaluation	Evaluates the usability based on Nielsen's heuristics.	Learnability Understandability Easy to remember

The selection of the surveys focuses on two measure: how successfully the heuristics are satisfied, and how the participants think of the quality of the generated design requirements. The heuristics, such as learning time, using user's language, preventing errors etc. are used for usability testing while quality of design requirements target the usefulness of the system. Please note that the questions in these questionnaires are not all relevant to our testing purposes. In total there are 75 questions which only about 50 are found relevant for the evaluation. The left out questions were about online documentation, which is not available in the current version of RaBBiT.

Findings

The results combined from the tests revealed different aspects of usability of RaBBiT. The survey questions are adopted seven-point Likert scale where '1' reports strong disagreement and '7' means strong agreement to each 'positive' question. The percentage of the participants agreeing with the statements in the questionnaire is calculated in one standard deviation range. Due to the nature of this evaluation, a broader statistical analysis is not required: The reported percentages show the general tendency by average users rather than extreme opinions. The statistical data from the heuristic evaluation is documented in Table 3. The participants found RaBBiT an interesting tool for programming. About 90% of them noted that its design is

intuitive to be used in programming and its user interface is consistent. About three out of every four participants strongly agreed that RaBBiT's error-recovery and task progress strategy provides an easy and efficient use. However, about 62% of the participants marked that learning RaBBiT takes time due to its complexity caused by the terms used and memory load. Only 25% of the participants thought RaBBiT's terminology may cause challenges in architectural programming process and found that RaBBiT's error prevention mechanism needs more improvement. Although this is not an ignorable group, the terminology comes from the framework used and is a promising alternative to domain-specific terms. Almost 65% of the participants agreed that RaBBiT provides simple and natural dialogs as well as error messages.

TABLE 3. Results of Practical Heuristics Evaluation

	Survey Questions (using 7 point Likert scale)	Mean	Standard Deviation	% in one standard deviation
1	Provide easy-to-use task-oriented help	5.63	0.74	88
2	Adopt the user's viewpoint; speak the user's language.	6.25	1.39	75
3	Simple and natural dialogue avoids extraneous information, steps, and actions.	5.63	1.06	63
4	For advanced users it provides shortcuts.	5.75	1.04	63
5	Gives the user a way return-to previous contexts	5.25	1.16	75
6	Show the user what is (not) possible provide affordances to indicate what can be done	5.38	1.19	75
7	Intuitive mappings design good response compatibility between controls/actions	5.50	1.07	63
8	Minimize memory load remove the need to remember across dialogues. Provide multiple views for easy comparisons	6.13	0.83	38
9	Consistency in the system and to standards make sure the same term/action has one meaning.	6.00	1.20	88
10	Provides timely feedback about system status	5.38	0.92	75
11	Prevent errors make it difficult to make errors	5.63	1.30	38
12	Error messages diagnose the source and cause of a problem and suggest a solution	5.88	0.99	63
13	Clearly marked exits and error recovery make sure the user can get out of an undesirable state easily.	6.00	1.07	88

The majority of the participants showed agreement with most of the statements in usefulness and easy-of-use survey (Table 4). The participants stated that using RaBBiT can increase the efficiency of programming tasks and have potential to increase the programmer's effectiveness. The average agreement on system usefulness is recorded as 6.13 but only 75% of the group in one-step standard derivation marked the systems potentially useful. All participants agreed that the system is flexible to interact with, clear and understandable, and easy-to-use within its own terms.

TABLE 4. Results of Perceived Usefulness and Ease of Use test

	Survey Questions (using 7 point Likert scale)	Mean	Standard Deviation	% in one standard deviation
1	Using RaBBiT in my job would enable me to accomplish tasks more quickly	6.25	0.71	88
2	Using RaBBiT would improve my job performance	5.88	0.99	88
3	Using RaBBiT in my job would increase my productivity	6.13	0.64	88
4	Using RaBBiT would enhance my effectiveness on the job	5.63	0.74	100
5	RaBBiT would make it easier to do my job	6.13	0.64	88
6	I find RaBBiT useful in requirements management	6.13	0.83	75
7	Learning to operate RaBBiT would be easy for me	6.00	0.76	75
8	I would find it easy to get RaBBiT to do what I want it to do	5.75	0.71	63
9	My interaction with RaBBiT is clear and understandable	5.25	0.71	100
10	I would find RaBBiT to be flexible to interact with	5.63	0.92	100
11	It would be easy for me to become skillful at using RaBBiT	5.50	0.76	100
12	I find RaBBiT easy to use	6.13	0.64	88

The after-scenario survey is used to measures the participants' reaction to the system after modeling the health clinic program as described (Table 5). The participants' agreement on RaBBiT's usefulness is consistent with the findings of the previous tests. About all of the participants agree or strongly agreed that RaBBiT can provide efficiency in performing programming tasks. Three out of four participants showed agreement on easy of use. However, most of them noted their reservation on this idea when it comes to entering complex dependencies and parametric relationships. RaBBiT was not as successful in providing support-information during task performance as in other features. The participants complained several times on lack of online help particularly related to 'formula' creation and editing.

TABLE 5. Results of perceived usefulness and ease of use test

	Survey Questions (using 7 point Likert scale)	Mean	Standard Deviation	% in one standard deviation
1	Overall, I am satisfied with the ease of completing the tasks in this scenario	5.88	0.64	75
2	Overall, I am satisfied with the amount of time it took to complete the tasks in this scenario	6.13	0.35	100
3	Overall, I am satisfied with the support information when completing the tasks	4.50	0.93	100

The outcomes of the other tests were consistent with the findings of these three tests. Therefore, we summarized our findings into our conclusions. Overall findings from the usability test showed that RaBBiT's prototype is powerful enough to receive appreciation from the expert users. However, there are still areas to be improved for better effectiveness and usefulness in architectural programming domain. In order to show the overall reaction of the users to RaBBiT, the combined results of the three tests reported below

(Figure 6). The questions on each test are designed in a seven-point Likert scale. Therefore, they pose a reliable reporting on the participants' opinion. The wording of each question also goes in parallel with each other; i.e. positive answers, such as strong agreement, are always asked for the positive expected features.

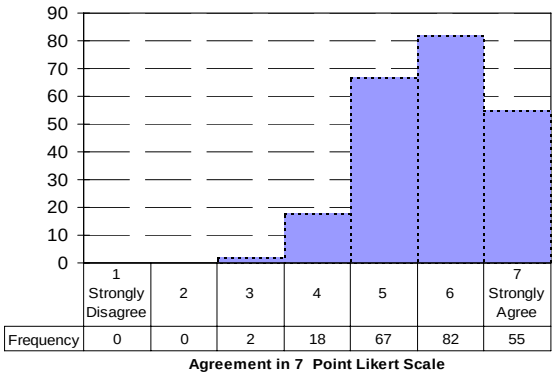

Figure 6. Cumulative agreement chart of all questions in the three tests

The responses to all questions indicate an agreement or strong agreement on RaBBiT's positive features supporting usability, usefulness, and its effectiveness. The participants agreed with most of the positive questions—i.e. questions that are asking if RaBBiT's features contribute towards usability. In addition to the data collected during the questionnaires, we observed several positive comments on the tool. The initial reaction of the participants were cautious and some-what conservative.

Conclusions

The participants several times reported that the way they perform architectural programming tasks manually different than they do it in RaBBiT. Therefore, there were some comments about easy of learning RaBBiT and understandability due to a mismatch in mental models. We categorize this reaction in the acceptability of the tool rather than learnability and understandability. However, mimicking the manual methods would defeat the purpose of having computational tools. Please note that a novel tool requires novel approaches that may bring additional concepts and techniques to be learned, which is normal and unavoidable. We face similar problems with learning other new tools, such as new versions of word processors. In RaBBiT's case, we note these comments and are dedicated to make it a better tool.

Although our participants noted about the difficulty in learning the tool, after 45-minutes tutorial they were able to model the sample architectural program for health clinics without any help from the observers. Like new users of any tool, the participants have developed skills in performing the given tasks: modeling architectural program. At the end of the evaluation, the participants found a strong potential in RaBBiT to increase programming task efficiency. The comments from the participants showed that our participants enjoyed using RaBBiT. The interactive and dynamic nature of the tool goes beyond simple data entry; the parameters and associations are visible to the users in a graphical format (as graph) that can be directly

manipulated. We observed that some of the participants played with the components (tables) to see how the dependencies are updating as they are dragged—as one of the participants noted—like a 'rubber band'.

We believe that the outcome of the usability testing is worth to investigate further. Particularly, how it can be adapted in other 'design-based' domains. For the future work, we are planning to continue same experiment with participants with expertise in other domains. Theoretically, this is possible. We can use the framework to configure a system that is composed of multiple parts with different variations; the parts can be selected based on certain conditions. The concepts in the framework, components, constructs, dependencies, relations, and parametric associations are generic enough to model such a system, for example, to configure computer (hardware). As a future research direction, I like to test with experts from other domains to what extend the framework (and RaBBiT) can be used their domains.

Acknowledgement

This work was financially supported by the Research Affairs at the UAE University under a contract no. 05-07-9-11/05.

References

Beck, K.: 2000, *Extreme Programming Explained*, Addison Wesley, Boston.

Davis, F.D.: 1989, Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly*, 13:3, pp. 319-340.

Dix, A., Finlay, J., Abowd G., Beale, R., and Finley, J.: 1998, *Human-Computer Interaction* (2nd Ed.), Prentice Hall, New York.

Erhan, H. I. and Flemming, U.: 2003, Interactive Support for Modeling and Generating Building Design Requirements, *Proceedings of Generative-CAD 2004 Symposium*, Carnegie Mellon University, Pittsburgh, PA.

Erhan, H. I. and Flemming, U.: 2005, User-System Interaction Design for Requirements Modeling. *Proceedings of the 10th Annual Conference of CAADRIA*, New Delhi, India.

Erhan, H.I.: 2003, *Interactive Support for Modeling and Generating Building Design Requirements*, PhD Thesis, Carnegie Mellon University, Pittsburgh, PA..

Hutchins, E.L., Hollan, J.D., and Norman, D.A.: 1986, Direct Manipulation Interfaces, *in* Norman, D.A. and Draper, S.W. (*eds.*), *User Centered System Design: New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum, Hillsdale, NJ, pp. 87-124.

Lauesen, S.: 2004, *User Interface Design: A Software Engineering Perspective*. Addison-Wesley Inc., London, England.

Lewis, J.R.: 1995, IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use in *International Journal of Human-Computer Interaction*, 7:1, pp. 57-78.

Nielsen, J.: 1995, *Usability Engineering*, Academic Press, Boston, MA.

Rosenberg, D.: 1999, *Use Case Driven Object Modeling with UML: A Practical Approach*, Addison-Wesley, Reading, MA.

Perlman, G.: 1997, Practical Usability Evaluation. Based in part on Nielsen's 1993 Heuristics and Norman's 1990 Principles

Sheiderman, B.: 1982, The future of Interactive Systems and The Emergence of Direct Manipulation, *Behavior and Information Technology*, 1(3), pp. 237-256.

Shneiderman, Ben and Plaisant, C.: 2005, *Designing the User Interface, Strategies for Effective Human-Computer Interaction*, Addison-Wesley, Reading, MA.

Wildman, D.: 1995, Getting the Most from Paired-User Testing. *Interactions*, 2 (3) pp. 21-27