Measuring the Usability of GUI Applications

Avi Harel

ErgoLight Usability Software Ltd.,

6 Giv'on St. Haifa 34335, Israel,

Tel: +972 4 826 3012, Fax: +972 4 825 8199

Email: avi@ergolight-sw.com

Keywords

usability, testing, validation, human factors, friendliness, user productivity, user error, capture/record, backtrack.

This article is concerned with the human factors of software usability, namely: to what extent does the application help users to achieve their operational goals. The article examines methodological considerations and methods employed in usability testing and evaluation. Usability main measures are the rate and time for task completion. Criteria for usability acceptance should be chosen according to the product positioning, the main criteria being user productivity, user satisfaction and avoiding critical errors.

Usability measurements are presented in two types of "user profiles". "User operation profiles" allow software developers to identify UI components of exceptional operation time. "User failure profiles" allow the developers to identify particular user failure modes. This information is useful for preventing system errors resulting from user errors. It is also useful for identifying instances of "almost accidents", in order to prevent real accidents.

More than half of the interaction is wasted to recover from user's errors. Usability testing tools should identify situations of user confusion and should provide statistics of situations that have similar patterns. In order to obtain user failure profiles, we need to on-line ask the users regarding their intentions. Software tools designed for usability testing should provide incentives aimed at getting the user's cooperation.

Usability cannot be tested by software engineers. It can only be tested by end users, preferably, with the supervision of usability professionals. "Silent mode" operation, which does not intervene with the user's actions, is required in order to obtain valid user operation profiles. Such tools can be used either in-house, in usability labs, or for remote testing, at the user's own site.

Definitions of usability

People interpret the term usability in various ways. Among the most common interpretations are:

1. The extent to which the product is operational and functions as specified
2. The Total Costs of Ownership (TCO)
3. The extent to which the product works according to the user's needs.

Although there is some overlap between these definitions, they do not coincide. For example, many software applications are overloaded with functions, which make them too complex for the end user. Such applications may fit the first definition quite well. The TCO of these applications will be either high or low, depending on the existence of alternative solutions.

This article is concerned with the human factors of software usability, namely: to what extent does the application help users to achieve their operational goals.

The costs of user failure

In the USA, the industry spends $500 billion annually on computers, networks and IT, with a resulting decrease in productivity (Landauer, 1995). Lederer and Prassad (1992) found that 63% of all software projects overran their estimates. The development costs of 46% of all new products go to failure (Business Week). More than 30% of

software development projects are cancelled before completion. The result is a loss of approximately $80 billion annually to the economy (Standish Group, 1995).

There are few resources that indicate that a main reason for project failure is usability. Recent findings indicate that the number of usability errors in a software product is much higher than expected (Molich et al., 1998). MacIntyre et al. (1990) found that the percentage of software code that is devoted to the interface has been rising over the years, with an average of 47%-60% of the code devoted to the interface. Lederer and Prassad (1992) found that the top four reasons for projects to overrun their estimates were all related to usability. The Standish Group report (1995) indicate that the primary reason for canceling project before completion are because of inadequate user design input (Standish Group, 1995). Landauer (1995) indicates that the average software program has 40 design flaws that impair employees' ability to use it. The cost in lost productivity is up to 720%. Of software lifecycle costs, 80% occur during the maintenance phase. Of the maintenance expenses, only 20% is due to bugs or reliability problems, the other 80% is due to unmet or unforeseen user requirements (Pressman, 1992).

In addition to those usability issues whose costs are apparent, there are many costly usability issues that are hidden. Few examples follow:

Ricoh found that 95% of the respondents to a survey never used three key features deliberately added to the product to make it more appealing. Customers either didn't know these features existed, didn't know how to use them, or didn't understand them (Nussbaum and Neff, 1991).

Bulkeky (1992) found that the cost of emplyees helping each other figure out how to do particular tasks on PCs is an estimated $6,000-$8,000 per PC per year.

The percentage of accidents due to human errors is in the range of 15%-90% of the cases of system failures (e.g. Hannaman, 1984).

Usability: from art to engineering

Traditionally, software engineers prefer to design an application properly right from the beginning, rather than to change the implementation at test time. To design the application properly right from the beginning, designers typically look for design rules, style guides and standards. Often, after choosing a design rule, they may realize that other people in the organization think that they chose the wrong one. As a matter of fact, many design-rules expressly conflict with each other. For example, a rule that dialog boxes must have a consistence appearance undoubtedly contradicts a rule that each dialog box will be distinctive from all other dialog boxes. For this reason, software engineers typically consider usability issues as a matter of personal taste, which should rely on the designer's intuition.

In order to guarantee a consensus at the organization, we need to agree on usability criteria first. The problem is that we have too many of these. Examples of usability criteria include:

- It should be friendly
- It should be easy to use
- It should be easy to learn
- It should maximize the user's satisfaction
- It should maximize the user's performance
- It should maximize the user's productivity
- It should forgive the user's errors.

The multitude of available criteria, which often conflict with each other, results in the typical suspicious attitude towards usability design, namely, that usability is a matter of personal taste.

Choosing criteria for usability testing

To choose criteria for usability testing, we need to consider:

1. The type of software application; for example, the criteria used to evaluate production systems are different from those used to evaluate mission critical or safety critical systems

2. The users' expectations and needs; for example, the criteria for evaluating applications to be used by people of technical background are different from those targeting computer novices
3. The user's organizational roles and the organization needs; for example, the criteria used for judging tools aimed for use by decision makers are different from those used for judging tools aimed for use by computer operators
4. The testing goals; for example, the criteria used for software acceptance are different from those used for providing feedback to the product designers.

To be able to obtain criteria that are widely acceptable for a particular application, we need to decide which of the considerations above are most relevant. When so many considerations and parameters are involved, it is very difficult to formalize rules. No wonder, software developers typically prefer to rely on their own intuition and experience, rather than on design-rules and standards.

Setting criteria by product positioning

Fortunately, there is one consideration that is the most relevant for most software developers, namely, the product positioning. Considering three main categories of product positioning, one can choose acceptance criteria as follows:

1. If the product is targeting user productivity, then one should compute the time required to complete each task, and then weigh each task according to importance and expected frequency of operation
2. If the product is aimed for obtaining a competitive edge, then one should evaluate the user's satisfaction
3. If the product is a front end of a mission critical or safety critical system, then one needs to mark critical controls - those controls which are liable to result in accidents - and then count the instances of the user's inadvertent activation of the critical controls.

Measuring the user's operation

The advantage of objective usability measures was demonstrated in several comparative studies (e.g., Molich et al., 1998). Recent studies show that subjective evaluation performed by usability specialists is not reliable (Nielsen et al., 1998).

Usability measurements are presented in two types of "user profiles". "User operation profiles" allow software developers to identify UI components of exceptional operation time. "User failure profiles" allow the developers to identify particular user failure modes.

User operation profiles

Few software-testing tools offer measures of the user's operation; known as "user profile" (not to be confused with the "user profile" as used by usability specialists, consisting of properties of the "user model"). These tools capture the user's actions and provide statistics of usage of User Interface Components (UIC) (controls, menu items etc.) such as the number of control activations and the average operation time.

Typically, testing engineers use "user profiles" in Use Case Engineering to describe the user's needs "in a way that supports both design and development" (Jacobson, 1998) and then to derive "test cases" from "use cases" (Major, 1998). To avoid confusion, in this article we refer to a statistics of UIC activation by "a user operation profile".

User operation profiles allow software developers to identify UICs with exceptional properties, such as:

- UICs that are used often
- UICs that are rarely used
- UICs that are easily operated (short operation time)
- UICs that are difficult to operate (long operation time)

Average operation time is easy to measure but difficult to interpret. For example, suppose that the user activated a control A and then a control B, that the elapsed time between activating controls A and B is 20 seconds and that a usability specialists considers this value too high. In order to reduce this value, we need to identify what part of the elapsed time users spend in evaluating the software response to activating control A, and what part of the elapsed time they spend in searching for control B. To allow decision making, the user operation profile should

break the operation time interval into useful parts, such as evaluation time, time spent doing non-UI related tasks and search time.

The role of tools in usability engineering

Tools may support usability engineering at the design phase, at the testing phase and at the deployment phase.

Theoretically, design tools may incorporate human factors, formulated as "design rules". Practically, however, most design rules that apply to certain situations, are not applicable to other situations. A simple example is the one given above of the contradiction between the rule of "consistent appearance" of application dialog boxes and the rule of "distinctiveness". Until we know how to formalize the applicability of design rules, User Interface (UI) designers should need to rely on usability specialists.

User behavior is often unpredictable, even for usability specialists. Many usability specialists prefer to ask the users for their preferences and to test the users' performance, rather than to rely even on their own designs. Still, empirical studies show that during usability inspections, inspectors often detect only a small portion of the existing usability problems (Desurvire, 1994; Jefferies et al., 1991; Nielsen, 1992). Tools may be used at the testing phase, to collect data from the users, in order to help usability specialists understand the user's behavior. The objectives of such tools may be to:

- Capture user's failure modes that are difficult to capture manually
- Collect data from remote users
- Provide objective measures for comparing design alternatives
- Provide measures of costs of design flaws.

The first objective in the list above is most important for enhancing system reliability, which is most appreciated by developers of mission critical and safety critical systems.

Understanding the user's confusion

Just by looking at the record of user's actions, it is often impossible to decide whether a particular situation indicates an instance of user confusion. Examples:

- The user works in the wrong mode
- The user activated a particular UIC unintentionally or inadvertently
- The user hesitates because s/he cannot recall an operational procedure.

"User failure profiles" allow the developers to identify particular user failure modes. Such information is useful for preventing system errors resulting from user errors. It is also useful for identifying instances of "almost accidents", in order to prevent real accidents.

Measuring the user's confusion

Experienced users may spend about half of their operation time doing real work, the other half being wasted for recovery from their own errors. Novices typically spend only small percentage of the operation time doing real work, wasting most of their operation time trying to figure out what the software functions are and how to get their work done.

Unfortunately, the time that users waste for error recovery and for learning the software operation cannot be derived from the user's operation profile. For this reason, usability specialists are concerned not only with the record of the user's actions, but also with those actions that the user failed to perform. Usability testing tools should be able to identify situations of user confusion and to provide statistics of situations that have similar patterns, such as the inadvertent activation of a particular UIC.

An example of objective measure of the user's confusion is the number of occurrences of each failure mode. Still better, the testing tool should provide estimations for the costs of each failure mode, in terms of the total time that the users have wasted trying to resolve all instances of that failure mode.

Usability problems that users do not identify

Typically, users identify only a small subset of the problems they encounter. Many usability problems are not identified even if the users are carefully selected to represent the real users. Often, users do not identify deficiencies related to actions that are incompatible to their intention. Some examples follow:

Users do not identify a usability deficiency if they cannot repeat the sequence of actions that ended up in the confusing situation. For example, when the user unintentionally activates a menu item that changes data, and no indication of that action is displayed on screen.

Users do not identify usability deficiencies resulting from mode discrepancy. A mode error occurs when a user tries to execute a function that is not relevant to the current application mode. For example, when the application is in "Read Only" mode and the user tries to modify data.

Usability problems that users would not report

In many other situations, users do identify usability deficiencies but avoid reporting on them for various reasons:

First, users avoid reporting on confusing situations that they have identified, when they are not sure about the reasons or when they cannot easily repeat them.

Second, many severe usability problems are the result of poor design. When the user fails to follow the designer's logic, both the designer and the user tend to consider it the user's fault, rather than a design deficiency. Most users will not report on an operational difficulty, unless they are convinced that the failure was not their own fault. Examples:

Users hesitate to report problems in understanding the application concepts and terminology as well as problems in knowing the procedures, which they need to follow in order to perform a task. Unless the information is not found, users would not report on difficulties they experience in accessing information in the user documentation, the training program or the on-line help. This type of usability problem is typical of complex systems, supplemented by a large body of user documentation that the user has no chance to read thoroughly, much less to remember.

Users would not report on instances of unexpected system response to inadvertent control activation. Inadvertent control activation occurs after a control has been used frequently. Hence, this type of confusion is typical of experienced users. Inadvertent control activation often occurs in cases of mode discrepancy, when the user wrongly "feels" that s/he is in the proper mode for operating that control. Even when the result of inadvertent operation is disastrous, users usually consider it to be their own fault and avoid reporting it as a design problem.

Third, users would not report on a deficiency that they consider as minor. For example, users typically would not report on problems that they have with the Insert key, which are seemingly negligible. No one has yet measured the overall costs of user wasted time due to data deletion while unintentionally editing in Overwrite mode. It is likely that overall, these costs sum up to millions of dollars.

Fourth, whenever the user feels that s/he has a task to complete, s/he postpones making the report until after the task is completed. Later, the user often cannot remember what the situation or the sequence of operation was. Goal oriented users are cooperative in reporting only on those problems which prevent them from completing their tasks. These users are not likely to report on a problem if they find a way to work around it.

Capturing the user's confusion

In order to decide whether the user is experiencing an operational difficulty, we need to find out the user's intention. The procedure involves three steps: first we need to identify the instances of users' confusion, next we need to interact with the user to get the user's intention, and then we need to provide to the users, so that they will be willing to cooperate.

Identifying instances of user confusion

A straightforward way to identify users' confusion is by asking them to report on difficulties that they encounter. Recent findings indicate that users can identify and report their own critical incidents (Castillo et al., 1998). However, this method provides very low yield, since users typically hesitate before reporting on operational difficulty they encounter.

The user's intention should be elicited in "real time". Inquiring about the user's intention after the session is over is often useless, since users typically fail to recall the sequences of their own operation. Therefore, usability specialists often prefer "on-line" observation that allows "real time" intervention, just to ask the user to specify his/her intention.

The main problem with the method of on-line intention elicitation is that the intervention distracts the user from his/her task. To reduce the number of interventions, usability testers use a variety of techniques, including asking the users to "think aloud" and video recording of the user interaction, which allows off-line analysis of the user's behavior.

Unfortunately, video recording is difficult to interpret off-line. An observer can rarely decide whether the user's hesitation expresses a difficulty in understanding the software response to recent actions or a difficulty in preparing for the next sequence of operation. To work around this problem, usability specialists often ask the user to "think aloud" (e.g., Nielsen, 1993).

Available software testing tools do not intervene in the user's actions. In order to understand the user's confusion, the usability-testing tool should prompt the user to specify his/her intention at the time when s/he experiences an operational difficulty.

Forms of intention specification

Users may specify their intentions either in a free format, using their own words, or in a structured format, from a description of the user's tasks. The task structure may vary; useful examples of task structures are task hierarchy, representing the user's task breakdown, and object-action representations. Task hierarchies may be accessed either by a sequence of list selections, or from a "tree view".

When stated in a structural way, the user's intention is easier to interpret than when stated in free format text; it allows identification of mode errors, when modes are expressed as constraints on the user tasks; it provides a consistent and comprehensive way of presenting the user's failure modes; and, it enables the measurement of the costs of failure modes.

Software engineers use the same terms that usability specialists use, with totally different meaning. For example, a Behavioral Model in the developers jargon is that of the Application Under Test (AUT) (e.g., Clarke, 1998) while usability specialists typically use this term to describe the user. Usability engineering tools may serve as protocols that help to bridge these gaps.

Software engineers should not specify the user's tasks, unless they are qualified as usability specialists. To assist the knowledge transfer from usability specialists, the testing tool may include means to copy parts of the task breakdown from external sources, such as user guides, Help/Contents and CASE tools.

Getting the user's cooperation

Asking the users to specify their intention distracts them from performing their main operational tasks. Users may cooperate with their inspectors in usability labs but not with a software tool that distracts them from doing their work. Software tools designed for usability testing should provide incentives aimed at getting the user's cooperation. Action related Help might be provided to assist the user in resolving situations of difficulties caused by unintentional, inadvertent actions or by working in the wrong modes.

Reporting

At the evaluation phase, information is required that will help the evaluator in deciding:

- Which controls did the user activate more often than their alternatives
- Which controls and software modes are error prone
- Which tasks did the user fail to accomplish
- Which deliverables (software, user documentation, and training) should be modified?

In order to analyze the reasons for the user's confusion, the evaluator needs to identify:

- When did the user experience operational difficulties
- What was the user's intention in those particular instances
- What were the user's actions that resulted in these instances?

The reports should be classified by the deliverables, allowing each member of the development team to get the feedback required for fixing the software. Means, such as time stamps, should be provided to synchronize the instances of operational difficulties with video or tape recording, commonly used in usability labs.

Experimental design

Sampling

Many software houses use the practice of asking members of the testing team to try the user interface and to report on usability flaws they encounter. Typically, if the software developed is not for internal use, the results of such testing are not valid for real end users. Software engineers, who are not used to thinking of the user's needs, operate the application in ways that are different from the way the end user will operate it.

Better approaches are either to get representatives of the end users to try the user interface in usability labs or to send the software over to the end users, for beta testing in their real environment, performing real tasks.

Remote testing

Remote usability data collection "is extremely useful to learn about usage patterns, attitudes and customers satisfaction of products and services around the world" (Tondre, 1998). Recent findings indicate that the inspector's focus is a major determinant of the capability to detect usability problems (Zhang et al., 1998). Recent studies show that users tend to distort responses in remote interaction (Moon, 1998).

Task structure

Testing can be either structured or in free form.

Typically, at the prototyping phase, usability professionals develop test scenaria. A scenario is a list of user tasks, chosen in a way that emphasizes the main functions of the application.

At beta testing, either in-house or remotely, users are typically asked to use the application doing their real work, performing their real tasks.

Silent mode operation

Interaction with the user to elicit the user's intention biases the measurement of the wasted time in two ways:

- ⊐ It adds the intention elicitation time to the wasted time
- ⊏ It shortens the wasted time by resolving the user's confusion.

In order to obtain valid user operation profiles, at least part of the measurement should involve "silent mode" operation, which does not intervene with the user's actions.

Customizing the dialogs

Users may vary in their expertise and experience levels. For example, experienced users would prefer more control and shorter prompts than inexperienced users.

To accommodate the variety of user's preferences, the testing tool should support user categorization. The software developers should be able to customize the user categories. Typical user classifications are by the users' level of expertise and by their organizational roles.

References:

Bulkekey, W.M. (1992). Study finds hidden costs of computing. *The Wall Street Journal*, 2 November, B4.

Castillo, J.C., Hartson, H.R. and Hix, D., 1998, Remote usability evaluation: can users report their own critical incidents?; CHI 98 Summary, pp 253-254, ACM Press.

Clarke, J.M., 1998, Automated test generation from a behavioral mode. Quality Week Proceedings.

Desurvir, H.W., 1994, Faster, cheaper!! are usability inspection methods as effective as empirical testing? In Jakob Nielsen and Robert L. Mack (eds) Usability Inspection Methods, ch 7, pp 1730202, John Wiley & Sons, Inc.

Hannaman, G.W. (1984). SHARP: Systematic Human Action Reliability Procedure, *EPRI NP-3583*, Palo Alto.

Jacobsen, N.E., 1998, The evaluator effect in usability tests. CHI 98 Summary, pp 255-256, ACM Press.

Jacobson, I., 1998, User cases from requirements to test. STAR 98 Proceedings.

Jeffries, R., Miller, J.R., Wharton, C. and Uyeda, K.M., 1991, User interface evaluation in the real world; a comparison of four methods. CHI'91 Conference Proceedings, pp 261-266, ACM, 1991.

Landauer, T. (1995), *The trouble with computers*. MIT Press.

Ledere, A.L. and Prasad, J. (1992). Nine management guidelines of rbetter cost estimating. *Communcations of the ACM 35(2)* (February), 51-59.

Major, M., 1998, Prioritizing OO tests build with use cases, STAR 98 Proceedings.

McIntyre, F. Estep, K.W. and Sieburth, J.M. (1990). Cost of user-friendly programming. *Journal of Forth Applicationand Research* 6(2), 103-115.

Molich, R., Bevan, N., Curson, I., Butler, S., Kndlund, E., Miller, D. and Kirakowski, J., 1998, Comparative evaluation of usability tests. UPA 98 Proceedings, pp 189-200.

Moon, Y., 1998, The effect of distance in local versus remote human-computer interaction; CHI 98 Proceedings, pp 103-108, ACM Press.

Nielsen, J., 1993, Usability Engineering. Academic Press, Boston.

Nielsen, J., Finding usability problems through heuristic evaluatioin. CHI'92 Conference Proceedings, pp 373-380, ACM, 1992.

Pressman, R.S. (1992). *Software Engineering: A Practioner's Approach*. McGraw-Hill, New York.

Standish Group (1995), The COMPASS report, *Forbes*.

Tondre, A., 1998, Remote usability testing at Sun Microsystems. UPA 98 Proceedings, pp 279-280.

Zhang, Z., Basili, V. and Shneiderman, B., 1998, Perspective based usability inspection. UPA 98 Conference Proceeding.