Evaluation of Software Systems

Chapter · November 2002

Source: CiteSeer

CITATIONS
35

READS
2,012

3 authors, including:

Günther Gediga
University of Münster
126 PUBLICATIONS **2,301** CITATIONS
SEE PROFILE

Kai-Christoph Hamborg
Universität Osnabrück
57 PUBLICATIONS **532** CITATIONS
SEE PROFILE

Gediga, G., Hamborg, K.-C. & Düntsch, I. (2002). Evaluation of Software Systems. In: A. Kent & J. G. Williams (Eds.) Encyclopedia of Computer Science and Technology (S. 127-153), Volume 45. New York: Marcel Dekker, Inc.

EVALUATION OF SOFTWARE SYSTEMS

INTRODUCTION

Evaluation as a general endeavor can be characterized by the following features (1):

- Evaluation is a task which results in one or more reported outcomes.
- Evaluation is an aid for planning; therefore, the outcome is an evaluation of different possible actions.
- Evaluation is goal oriented. The primary goal is to check the results of actions or interventions, in order to improve the quality of the actions or to choose the best action alternative.
- Evaluation is dependent on the current knowledge of science and the methodological standards.

Evaluation as an aid for software development has been applied since the last decade, when the comprehension of the role of evaluation within human–computer interaction had changed. In one of the most influential models of iterative system design, the Star Life Cycle Model of Hix and Hartson (2), the activities

Task analysis
Requirement specification
Conceptual and formal design
Prototyping
Implementation

are each supplemented by an activity "Evaluation" which helps to decide progression to the next step. Software can be evaluated with respect to different aspects (e.g., functionality, reliability, usability, efficiency, maintainability, portability) (3). In this survey, we concentrate on the aspect of usability from an ergonomic point of view. This aspect has gained particular importance during the last decade with the increasing use of interactive software.

Whereas, in earlier times, evaluation of software took place at the end of the developing phase, using experimental designs and statistical analysis, evaluation is, nowadays, used as a tool for information gathering within iterative design:

> Explicit human-factors evaluations of early interactive systems (when they were done at all) were poorly integrated with development and therefore ineffective. They tended to be done too late for any substantial changes to the system to still be feasible and, in common with other human-factors contributions to development, they were often unfavorably received [. . .] This situation has been improved in recent years in a number of ways. (4)

Within this context, instruments for evaluation are not primarily used for global evaluation of an accomplished product, but these instruments are applied during the development of a product. Indeed, most experts agree nowadays that the development of usable

software can only be done by a systematic consideration of usability aspects within the life-cycle model. One prominent part is the evaluation of prototypes with respect to usability aspects, employing suitable evaluation techniques in order to find usability errors and weaknesses of the software at an early stage (2,5–7).

GOALS AND RESULTS OF EVALUATION

Any evaluation has pragmatically chosen goals. In the domain of software evaluation, the goal can be characterized by one or more of three simple questions:

1. Which one is better? The evaluation aims to compare alternative software systems (e.g., to choose the best fitting software tool for given application) for a decision among several prototypes, or for comparing several versions of a software system. An example of such a strategy can be found in Ref. 8.
2. How good is it? This goal aims at the determination of the degree of desired qualities of a finished system. The evaluation of the system with respect to "Usability Goals" (7,9) is one of the application of this goal. Other examples are the certification of software and the check on conformity with given standards.
3. Why is it bad? The evaluation aims to determine the weaknesses of a software such that the result generates suggestions for further development. A typical instance of this procedure is a system-developing approach using prototypes or a reengineering of an existing system (2).

The first two goals can be subsumed under the concept of *summative evaluation*; the third goal is an instance of the *formative evaluation* approach.

In general, summative evaluation is concerned with the global aspects of software development and does not offer constructive information for changing the design of the system in a direct manner. It is performed when the development of the system is almost or entirely accomplished (2). Summative evaluation is also applied to prototypes in case there is a need for controlling the effect of design changes in comparison to a preceding version of the system.

In contrast, the goals of formative evaluation are the improvement of software and design supporting aspects (10). It is considered the main part of software evaluation and plays an important role in iterative system development. In every development cycle, formative evaluation results in the following:

- Quantitative data for the description of the progress of the realization of usability goals
- Qualitative data which can be used to detect the usability problems of the system

The resulting data can be classified by the following criteria (2):

Objective: Directly observable data; typically user behavior during the use of the interface or the application system

Subjective: Opinions, normally expressed by the user with respect to the usability of the interface or the application system

Quantitative: Numerical data and result (e.g., user performance ratings)

Qualitative: Non-numerical data (e.g., lists of problems, suggestions for modifications to improve the interaction design)

EVALUATION CRITERIA

In the human-factors community, it is a matter of some debate about what constitutes an evaluation criterion. We follow Dzida (11), who advised that "criteria" should mean the measurable part of attributes of design or evaluation. Although this seems clear enough, the literature on software evaluation shows only a few attempts to achieve general principles of design and evaluation criteria.

The concept of *usability*, which is a general quality concept for software systems, is often used for the determination of evaluation criteria (12–14).

The International Standard ISO 9241 (Part 11), which is the methodological foundation of the HCI standard "Ergonomic requirements for office work with visual display terminals" (ISO 9241), states,

> Usability of a product is the extent to which the product can be used by specific users to achieve specific goals with effectiveness, efficiency, and satisfaction in a specific context of use.

Note that the characterization of usability by "effectiveness," "efficiency," and "satisfaction" is not a breakdown to measurable criteria, but only a first step toward reaching such criteria (15, p. 6). The same is true for ISO 9241 (Part 10) (12,16) and for other general guidelines and design principles as well. These standards give no advice on how to achieve trustworthy criteria which will fit into the needs of a specific software evaluation task. For a concrete evaluation project, there needs to be a proper operationalization of the general guidelines and standards, which takes into account the context in which the system will be applied. The context of use is defined in ISO 9241 (Part 11) in terms of the user, the task, the equipment, and the environment. Following this description of the components of the context of use, every evaluation has to take into account the following restrictions:

A. The characteristics of a product's users such as experience, age, gender, or other more specific features
B. The types of activity or tasks that the user will perform
C. The environment of the study itself, ranging from controlled laboratory conditions to largely unstructured field studies
D. The nature of the evaluation object, which can be a paper prototype, a software mock-up, a partially functional prototype, or an accomplished system

Even an expert-based evaluation should consider all of these four contextual restrictions in order to achieve a valid evaluation result.

In principle, there are (at least) four approaches for the derivation of criteria:

1. The successive division of principles into a hierarchy using specialization, until the leaves can be measured by a set of known procedures (top-down I).
2. The successive division of principles into a hierarchy using specialization, until the leaves cannot be subdivided any more. This level will then be operationalized by a tailored technique (top-down II).
3. A classification of known and/or commonly applied methods, and their map-

ping to criteria, or, alternatively, an empirical assignment of items to criteria (bottom up).

4. A direct operationalization of principles, which takes into consideration the measurable demands and the qualities of the object under study.

An advantage of the first strategy is that it is based on known and tried methods. However, known methods will be applicable only to part of the actual problem, namely the final level; therefore, the problem arises that the results may be misinterpreted due to a divergence artifact (17).

The second strategy has the advantage that it intends to measure everything that the evaluation requires. Its disadvantage are the huge amount of labor required for the construction of tailored instruments and that the results usually cannot be compared to the findings of other studies.

The third strategy is expensive as well, including the risk that only some of the basic principles are well covered while others are not.

The fourth strategy not only offers an operationalization but also attempts a more precise formulation of the criteria and the underlying principles as well. Its disadvantage is its cost; furthermore, empirical studies following this procedure offer results which are, in most cases, too general and not sufficiently specific for a concrete application (12).

Tyldesley (18) argues that "objective measurements" such as the time required for learning a system are preferred measurements and should be used as criteria for software evaluation. Although objectivity and reliability of such measures are generally accepted, there are doubts about their validity:

> . . . the so-called objective measurements of performance, for example the number of problems completed within a fixed time span, are widely applied; they are, however, rarely valid as criteria, since the underlying causal relationships are not sufficiently investigated or stated. (translated from Ref. 12)

The derivation of "subjective" criteria such as acceptable levels of human cost in terms of tiredness, discomfort, frustration and personal effort, or even an operational definition of attitude is much harder, and Whitefield et al. (4) come to the pessimistic conclusion that there is, at present, no support for the specification and operationalization of such criteria.

The bottom-up approach offered by Strategy 3 can be a starting point for criteria derivation: At the beginning, one collects "standard" operationalizations with known validity and reliability. From this collection, the designer can choose those instruments which can be subsumed as a set of criteria for the principles under study. At least for the "subjective" criteria, an operationalization via a well-analyzed questionnaire or a survey instrument may be a successful way to cope with the problem.

There seems to be no problem for operationalization on the principles of *heuristic evaluation* proposed by Nielsen (14) and comparable techniques. These are performed by auditors who evaluate the system, given a set of general design principles. In this case, the concretization will be done by the auditor in interaction with the system; therefore, the burden of proper operationalization is put on the auditor. Such methods demand well-educated and well-trained evaluators, and it is no wonder that specialists who are usability *and* task experts perform better with this techniques than usability experts who are not task experts (19).

EVALUATION TECHNIQUES

Evaluation techniques are activities of evaluators which can be precisely defined in behavioral and organizational terms. It is important not to confuse "evaluation techniques" with "evaluation models," which usually constitute a combination of evaluation techniques.

We classify evaluation techniques into two categories: the *descriptive evaluation techniques* and the *predictive evaluation techniques*, both of which should be present in every evaluation:

Descriptive evaluation techniques are used to describe the status and the actual problems of the software in an objective, reliable, and valid way. These techniques are user based and can be subdivided into several approaches:

Behavior-based evaluation techniques record user behavior while working with a system which "produces" some kind of data. These procedures include observational techniques and "thinking-aloud" protocols.

Opinion-based evaluation methods aim to elicit the user's (subjective) opinions. Examples are interviews, surveys, and questionnaires.

Usability testing stems from classical experimental design studies. Nowadays, usability testing (as a technical term) is understood to be a combination of behavior- and opinion-based measures with some amount of experimental control, usually chosen by an expert.

Observe that all descriptive evaluation techniques require some kind of prototype and at least one user. Note, furthermore, that the data gathered by a descriptive technique need some further interpretation by one or more experts in order to result in recommendations for future software development.

Predictive evaluation techniques have as their main aim to make recommendations for future software development and the prevention of usability errors. These techniques are expert-based or, at least, expertise-based, such as walkthrough or inspection techniques. Even though the expert is the driving power in these methods, users may also participate in some instances.

Note that predictive evaluation techniques must rely on "data." In many predictive evaluation techniques, such "data" are produced by experts who simulate "real" users. The criteria *objectivity* and *reliability*, which are at the basis of descriptive techniques, are hard to apply in this setting. Because validity must be the major aim of evaluation procedures, there are attempts to prove the *validity* of predictive evaluation techniques directly (e.g., by comparing "hits" and "false alarm" rates of the problems detected by a predictive technique) (20).

At the end of the section we shall briefly discuss evaluation techniques which can be used either for predictive or descriptive evaluation (e.g., formal usability testing methods) and those which do not fit into the predictive/descriptive classification, such as the "interpretative evaluation techniques" (21).

Behavior-Based Evaluation Techniques

Behavior-based techniques rely on some form of observation in order to detect usability problems. Because the user is confronted with a prototype of the system, these techniques can only be applied in the later stages of system development.

As a first indicator of the expense of the technique, we offer the ratio "analysis time" (time to process the data, by experts after data elicitation) to "time of users' interaction with the system" (time to elicit the data which will be analyzed).

Observational Techniques

User observation as a method of software evaluation takes place directly or indirectly by a trained observer (12), and it can produce quantitative as well as qualitative data (2, p. 306). The techniques try to avoid subjectivity as much as possible by standardizing procedures and documentation, as well as by training the observer. They are applied, for example, if the user's behavior is of interest, especially when the user cannot tell how well (or badly) (s)he behaves using a prototype (14, p. 207).

It turns out that direct observation is not a well-suited technique (22, p. 617), mainly because the observer cannot deal with the amount of information which has to be processed. Therefore, indirect techniques using a video recording of the user's behavior are commonly used. In human-factor laboratories, several parallel video loggings (hands, screen, face, whole body), together with log files, are synchronized and used by the observers; an example is the VANNA system of Harrison (23).

Observational techniques show a ratio of 5 : 1 of analysis time to recording time (22, p. 620). This ratio does not seem to be reduced by using automated procedures. The main reason is that computerized systems offer more information to the observer, which works against the effect of the rationalization.

Thinking-Aloud Protocols

The method of thinking aloud provides the evaluator with information about cognitions and emotions of a user while (s)he performs a task or solves a problem; examples can be found in Refs. 14 and 24–26. The user is instructed to articulate what (s)he is thinks and what (s)he feels while working with the prototype. The utterances are recorded either using paper and pencil (6,27) or with more modern techniques using audio or video recording (25,26). In connection with synchronized log-file recordings, the evaluator has the chance to interpret the user's reaction based on contextual information (25). Nielsen (6, p. 72) expressed the opinion that a large amount of overhead is unnecessary and that simple paper–pencil recordings suffice to elicit the relevant contextual information.

A slightly different approach are the *preevent* and *postevent* thinking-aloud procedures, in which the user thinks aloud before a task has started or after a task has been completed. The postevent technique is useful if the tasks require careful concentration. Their protocols are based on the comments evoked by thinking aloud, while the user observes the video recorded interaction with the system. Postevent protocols were criticized because the user might rationalize her/his own actions. Some researchers, however, point out that the more rational style of postevent protocols provides the evaluator with useful context information which event protocols cannot deliver (28). The empirical comparison of postevent protocols with event protocols shows that the information provided by postevent protocols are of higher quality, although the amount of information is reduced in comparison to event protocols (29,30).

The procedure of analyzing thinking-aloud protocols has not much changed since the seminal paper by Lewis (26) in 1982: The protocols are scanned and those episodes are extracted which describe the users' problems and their complaints. The episodes are listed and coded by a user number and an episode number. Afterward, the episodes are matched with one "feature" of the system. These "features" define the grouping criteria

for the episodes. The design aspect of the "feature" and the frequency information provide further help with the interpretation of the results.

By using thinking-aloud techniques, the evaluator obtains information about the whole user interface. This type of evaluation is oriented toward the investigation of the user's problem behavior and decision behavior while working with the system (12, p. 99). The procedures to transform the lingual information into evaluation criteria is only weakly standardized, which may be problematic.

The expense of the method varies. In our experience, the concurrent-event technique requires a user time factor near 1 : 1 and an evaluator time factor of about 3 : 1 for a trained evaluator. The video-based postevent technique needs more time, with a user time factor of at least 3 : 1 and an evaluator time factor of at least 5 : 1.

Video Confrontation

In the area of human–computer interaction, the video confrontation technique has been in use since the mid-1980s (31,32). It is based on the postevent thinking-aloud technique, but differs in the way the video recordings of the user-system interaction are analyzed. The evaluator picks out interesting recorded episodes and interviews the user about these. The interview is guided by pragmatic or theoretical goals, such as finding usability problems or examining the relationship between user errors and emotional reactions. The questions concern cognition, emotions, or problems which have occurred during the use of the system (32). The main result of the process is the interview protocol. Because the frame for the analysis is set by the evaluator before performing the interview, the questions are concentrated on the salient points, and the protocol is much easier to analyze than a thinking-aloud protocol. Furthermore, the interview questions can be standardized and, thus, they can lead to an operationalization of evaluation criteria, which establish a direct relation to the evaluation goals. It is also possible not only to arrive at a list of problems but also to obtain an indication of the reasons for these problems. Other sources of information such as other recorded sequences and log files may also help the evaluator to interpret the outcome of the procedure (33).

The video confrontation technique is rather costly: A rough estimation of the user time ratio is 5 : 1 and of the evaluator time ratio is 7 : 1.

Opinion-Based Evaluation Methods

Oral or written interview techniques are commonly used to evaluate the user's opinion about software systems. The difference between oral interview techniques and questionnaire-based techniques lies mainly in the effort for setup, evaluating the data, and standardization of the procedure. The development of an interview is more economic than for questionnaires, whereas carrying out and evaluating a questionnaire procedure can be done with less effort and costs. Standardization and accuracy are also better for questionnaires.

The advantage of an interview in comparison with the observational methods of the subsection Observational Techniques is that an interview helps to obtain an insight into the user's opinion of the system which cannot be gathered by observation alone (14, p. 209ff). Furthermore, these techniques are not as expensive as observational techniques.

Oral interviews are held in a flexible manner after the user had come into contact with the system and in a more structured way if the user was faced with unforeseen aspects (14,34).

Another aspect are "critical incidents" (2,14,34) (e.g., examples of worst-case situations created by experts), which can be used to evoke oral responses.

Because there is no standardization of oral interviews as a software evaluation technique, we shall not discuss the various approaches in more detail. As an introduction, we invite the reader to consult the checklist of Nielsen et al. (22,35).

Interviews and questionnaires are primarily used in the specification, design, or reengineering phase, or as a means of system comparison (summative approach), where different techniques have different focal points. In the sequel, we shall discuss three types of questionnaire in more detail: the Questionnaire for User Interface Satisfaction (36, p. 482ff), the Software Usability Measurement Inventory (37), and the IsoMetrics questionnaire (38).

QUIS: Questionnaire for User Interface Satisfaction

The Questionnaire for User Interface Satisfaction (QUIS) aims to provide a measure of overall satisfaction; additionally, it evaluates some aspects of the user interface based on user opinions. The original version [QUIS 5.0 (39)] consists of five scales:

- "Overall User Reactions" with attributes such as the following:

  Terrible/wonderful
  frustrating/satisfying
  dull/stimulating
  inadequate power/adequate power
  rigid/flexible

- "Screen," which evaluates the display appearance
- "Terminology and System Information," which evaluates the distinctiveness of terminology and message display
- "Learning," which evaluates the suitability of the system for learning
- "System Capabilities," which evaluates the efficiency and related aspects of the system in terms of speed and reliability

The current version of QUIS is enhanced by the scales "Technical Manuals and On-line Help," "On-line Tutorials," "Multimedia," "Teleconferencing," and "Software Installation" (40). A short (47 Items) and a long version (126 Items) of QUIS are available. The short version should be used if there are only few time resources or if motivational problems of the user can be anticipated. In the long version, there are more concrete questions explaining and complementing the "leading items" (which constitute the short version) of each scale. At the beginning, there are questions about the properties of the system under study (in terms of the user's opinions) and the characteristics of the user.

The scaling of the items ranges from 1 to 9, and an additional "no answer" option. The end points of the scales are anchored by pairs of adjectives (e.g., difficult/easy). User comments about the system can be expressed at the end of each scale.

Reliability and validity of QUIS(5.0) have been investigated by Chin et al. (39). They could show that its overall reliability is good, but no separate reliability measures for the five subscales were reported. Validity was shown in terms of differentiation of "like" and "dislike" for a software product, and the results of QUIS can be used to differentiate command line systems and menu-driven applications. The predicted dimensionality of QUIS(5.0) is questionable and there should be more analysis for construct validity as well, as " . . . no attempt to establish any construct or predictive validity was

done" (39, p. 218). For the new version of QUIS, psychometric results are not (yet) available.

SUMI: The Software Usability Measurement Inventory

SUMI (37,41) was developed primarily as a summative instrument which measures a user's perception of the usability of a software. SUMI consists of 50 items, which are assigned to five scales. Additionally, a "global" scale was constructed consisting of those 25 items which show a high correlation with an overall usability factor. The "global" scale was constructed to present the perceived quality of the software as one index.

The answer format for the items consists of "agree", "don't agree," and "disagree." In a SUMI evaluation, the recommended number of users is 10–12. The headings of the scales are as follows:

- Global (25 items)
- Efficiency (10 items)
- Affect (10 items)
- Helpfulness (10 items)
- Control (10 items)
- Learnability (10 items)

The efficiency scale evaluates how well the software supports the user while working on the tasks. The affect scale measures the user's general emotional reaction to the software. Helpfulness is intended to measure the degree to which the software is self-explanatory and the suitability of the help system. The control scale is used to measure the degree of the user's feeling that (s)he controls the software. Finally, learnability measures time and effort for learning the handling of the software (or parts of it) from the user's point of view.

For formative evaluation, SUMI was supplemented by the "Item Consensual Analysis" (ICA), which enables the evaluator to locate usability problems more precisely than with the analysis of the scales profiles. ICA needs a "Standardization Database" which consists of expected pattern of responses for any SUMI item. A comparison of expected and observed frequencies for the items (using a χ^2 test) shows which item signals a demand for change.

A second variant of the ICA should be used if a more detailed problem analysis is required. In this case, the first step consists of a sample of at least 12 users who are confronted with the standard SUMI. Based on the ICA analysis of the user data, an interview script is constructed, and interviews are held to obtain explanations for the discrepancies of expected and observed frequencies.

The subscales of SUMI show reliabilities which range from satisfactory to good. Validation studies of SUMI have shown that the questionnaire has the capability to distinguish software of different ergonomic quality (37,41). The usefulness of SUMI in consultancy-based studies as well as in case studies has been exemplified in Refs. 37 and 41.

IsoMetrics

The IsoMetrics usability inventory provides a user-oriented, summative as well as formative approach to software evaluation on the basis of ISO 9241 (Part 10) (38). There are two versions of IsoMetrics, both based on the same items: IsoMetrics^S (short) supports summative evaluation of software systems, whereas IsoMetrics^L (long) is best suited for formative evaluation purposes.

The current version of IsoMetrics comprises 75 items operationalizing the 7 design principles of ISO 9241 (Part 10). The statement of each item is assessed on a five-point rating scale starting from 1 ("predominantly disagree") to 5 ("predominantly agree"). A further category ("no opinion") is offered to reduce arbitrary answers. IsoMetrics^L consists of the same items as IsoMetrics^S and uses the same rating procedure. Additionally, each user is asked to give a second rating, based on the request

> Please rate the importance of the above item in terms of supporting your general impression of the software.

This rating ranges from 1 ("unimportant") to 5 ("important"), and a further "no opinion" category may also be selected. In this way, each item is supplied with a weighting index. To evoke information about malfunctions and weak points of the system under study, the question

> Can you give a concrete example where you can (not) agree with the above statement?

is posed. This gives users the opportunity to report problems with the software, which they attribute to the actual usability item.

The IsoMetrics design provides information that can be used within an iterative software development. In summary, these are as follows:

- Scores of the usability dimension to measure the progress of development
- Concrete information about malfunctions and their user-perceived attributes
- Mean weight of any user-perceived attribute, given a class of system malfunctions

IsoMetrics has proved its practicability in several software development projects. Its reliability was examined in two studies for five software systems and could be justified for each of the seven design principles. In order to validate the IsoMetrics inventory, the scale means of the five different software systems analyzed in the reliability studies were compared. It could be shown that programs with different ergonomic qualities were discriminated in the corresponding scales (38).

Given 10 evaluating users, IsoMetrics^L evokes approximately 100 remarks. [The validity of this procedure has been reported elsewhere (42)]. These are prioritized by their rated importance and their frequency. The content analysis of the remarks results in the identification of weak points of the evaluated software and provides the input to a usability review. In such a review, users, software engineers, and human-factor specialists develop remedies for the system's weak points and discuss its (re)design. This procedure has been highly accepted by developers as well as the users. A version of IsoMetrics which is applicable in group settings is discussed in Ref. 43.

Another summative evaluation method based on ISO 9241 (Part 10) has been adopted as a formative instrument in a system called "Qualitative Software Screening" (44,45).

Usability Testing

Usability Testing is a name for a systematic and—more or less rigid—experimentally based gathering of information about a product or a prototype using user representatives (46,47). A rough classification by Rubin (47, p. 22) can be used to characterize different approaches to Usability Testing:

- "Formal tests conducted as true experiments." This approach is characterized by the use of classical experimental designs for testing hypotheses and for deriving causal dependencies (see also Ref. 48).
- The second class of usability tests, describe to be "a less formal" one, employs an iterative cycle of tests intended to expose usability deficiencies and gradually shape or mold the product in question.

Whereas both approaches differ in their proximity to classical experimental designs in terms of the accuracy of the description of the independent factors, the problem of defining dependent variables—the measurables—is common to both approaches. The dependent variables are chosen pragmatically according to the evaluation goals. Techniques described earlier, including

- Questionnaires and interviews
- Observational methods
- Think-aloud technique
- Video confrontation

are used to measure the impact of differences in system design or different versions of a prototype on the usability of the product. Additionally, so-called measurement criteria (18) are used. These are task-specific measures such as "time to complete a task" or "percentage of tasks completed," which can be easily applied if tasks are accurately described. The thinking-aloud technique is "perhaps the most common technique for qualitative data generation . . . " (2, p. 306); indeed, many other authors emphasize the use of thinking-aloud methods in usability testing (5,7,13,14,25,26,47).

However, Nielsen (14, p. 165) advocates

> User testing with real users is the most fundamental usability method and is in some sense irreplaceable, since it provides direct information about how people use computers and what their exact problems are with the concrete interface being tested.

This approach has been criticized because of its expense and its low cost–benefit relation (49). As a possible remedy, inspection-based methods are being discussed as an alternative evaluation approach (20). These are predominantly used to uncover general usability problems in the interface and consist of a combination of predictive techniques, described more fully below, such as heuristic evaluation, Walkthrough, or group discussions with other quality assurance tools like standard inspection or feature inspection (20,50).

Virzi (50) calls usability inspection a "nonempirical" method and points out that this method has the typical weaknesses of a purely predictive technique, because it relies mainly on the ability of the inspectors to predict problems of users with the software.

Because Usability Testing requires a large amount of expertise for the following:

- To set up the experimental design
- To choose the suitable tasks for comparison
- To select the users and the number of users
- To define the measurables properly

it is perhaps best suited for usability engineers. It is certainly not a suitable technique for untrained evaluators.

Predictive Evaluation Techniques

Because predictive evaluation techniques aim to achieve results which can be used in practice, these methods are related to problem-solving techniques which are used for the same reasons in other contexts, such as requirements analysis.

Whereas observational and opinion-based methods require a (more or less) sophisticated prototype, the predictive evaluations do not need a built system, because empirical methods are replaced by a theory or the contribution of experts. Consequently, user involvement is not as dominant as in the empirical evaluation techniques. Nevertheless, user participation could be more prominent in predictive techniques, because user representatives have the opportunity to influence the development process actively, whereas in descriptive techniques, the user plays a more passive role.

Usability Walkthrough

There are different variants of Walkthrough methods in human–computer interaction (13,47,51–53). The object of a Walkthrough is either the system design, a paper prototype (e.g., Ref. 13, p. 698), a built prototype (e.g., Ref. 47), or a completed system.

The main feature is a set of instructions to check, step by step, the artifact under development. For each step, the participants note problems with the task or screen design–in the case of a paper prototype—or with the user interface of a built prototype. After the data collection, problems are discussed and improvements of the system are proposed. The task sequence of a Walkthrough must be fixed, otherwise the simulation is not meaningful for evaluation purposes.

Walkthrough methods for software evaluation are often performed in a group (13,51,52), mainly because group Walkthrough has proved to be more effective than comparable individual Walkthroughs (22, p. 706). The participants of a group Walkthrough come from different areas (e.g., representatives of the expected user population, product developers, and human-factor specialists). Because of the heterogeneous composition of the group, one also speaks of "Pluralistic Walkthrough" (52).

Walkthroughs are well suited to detect those usability problems caused by a discrepancy of system behavior and a user's habits or expectations (54, p. 107). Furthermore, they are an aid to ascertain whether the user interface is perceived to be adequate. Examples for such Walkthroughs are checking the wordings in the user interface, the distinction of buttons, commands, or menus, or the analysis of small task sequences.

As a general method, Walkthroughs are limited because of their cost. A whole user interface cannot be evaluated by Walkthroughs with a reasonable cost–benefit relation and, therefore, only selected features are usually taken into consideration. A further limitation is the fact that explorative behavior can only be simulated in a limited way; therefore, unexpected errors or misfits cannot be detected by usability Walkthroughs (52). Another problem is the fact that the success of a Walkthrough, if applied as a group technique, depends on the combination and the psychological fit of the group members (13, p. 698ff).

A prominent example is the Cognitive Walkthrough (CW). It is a method for evaluating user interfaces by analyzing the mental processes required of a user (53, p. 717ff). Its scope is the ease of learning—in particular, of learning by exploration—and it can be carried out by individual experts or by a group of peers. Details for a peer group evaluation can be found in Ref. 54 (p. 106).

The CW is guided by four questions, which the evaluator should answer at every task step:

- Will the user try to achieve the right effect?
- Will the user notice whether the correct action is available?
- Will the user associate the correct action with the effect to be achieved?
- If the correct action is performed, will the user see that progress is being made toward a solution of the task?

Input to a CW session consists of a detailed design description of the interface such as a paper mock-up or a working prototype. Additional input may be a task scenario, the precise description of prospective users, the context of system use, and a correct sequence of actions that a user should successfully perform to complete the designated task. Additionally, there must be a mapping of the task steps to corresponding "interface states." The evaluator simulates the task steps according to the limited possibilities of the intended users and judges whether the system offers suitable tools for each step of the task sequence. In particular, it is noted whether all necessary information is presented in a complete and appropriate manner; the evaluator will record other usability problems as well.

For each action within the task sequence, a "success story" is constructed, which contains the reason(s) why a user will or will not perform the action. If a difficulty is identified, a reason is assigned, for example, that a required menu items does not seem related to the user's goal. Problems and their causes are recorded. Finally, the analyst reports alternatives to the proposed design. In this way, the method utilizes factors which influence mental processes such as a user's background knowledge.

The advantage of the CW is that it can be applied in early stages of the development and that not only usability weaknesses of the system are reported but other problems as well.

One of the restrictions of the CW is the fact that the simulated sequence of actions has to be correct. This requirement puts a high demand on the qualification of the analyst, because (s)he has be familiar with the tasks and the environment in which the tasks will be performed. However, whether a user will perform according to the "correct" way cannot be checked beforehand. Therefore, the analyst should have a profound knowledge of human factors and should be aware of the intended user profile; furthermore, the analyst should be familiar with at least the basics of system development.

It has also been criticized that the focus of analysis of the CW sacrifices other important usability information such as overall consistency (50, p. 707).

Inspection by Experts and Heuristic Reviews

Usability inspection by experts can be subdivided into free, structured, or model-based variants. These are used for the generation of problem lists and they form the basis for system optimization (14). The usability inspection is carried out by a human-factor specialist, who is not directly involved in the development of the system. If there are several inspections of the same subject, they will be done independently of each other. The object of the evaluation can be the whole user interface or some of its components (12, p. 100). The technique can be used at an early stage of the life cycle in which paper prototypes will be used.

In case of a free expert inspection, the evaluator checks the software system using only a few general guidelines. Structured expert inspection, like standard inspection or consistency inspection, is based on detailed criteria or standards and uses checklists as a

tool. Model-based inspection commences with a model of the user interface and includes a defined evaluation procedure (12, p. 101).

Heuristic Evaluation (HE) is a special adaptation of a free expert evaluation (19). Unlike the expensive and detailed expert evaluation by standard or consistency inspections, the HE uses the following usability heuristics:

- Provide a simple and natural dialogue
- Speak the user's language
- Minimize user memory load
- Be consistent
- Provide feedback
- Provide clearly marked exits
- Provide shortcuts
- Provide informative error messages
- Prevent errors

The heuristics are general principles which guide the evaluator through the inspection process. A number of evaluators inspect the user interface in individual sessions. Nielsen (19, p. 26) recommended three to five evaluators, because in his studies using HE, a larger number of evaluators will not result in additional relevant information. The degree of standardisation is not very high, and

> In principle, the evaluators decide on their own how they want to proceed with evaluating the interface. (19, p. 29)

Heuristic evaluation is normally carried out by a double inspection of the user interface. In the first round, the evaluator obtains a feeling for the system and its potential. The second inspection allows the evaluator to concentrate on those elements of the interface which are relevant for evaluation purposes. The evaluator steps through the interface design and checks the dialogue elements on the basis of the heuristics. Results of the HE are either fixed in a protocol by the evaluator or the evaluator reports the findings to an observer while checking the system.

An HE session should not last longer than 2 h. If the evaluation of a larger or complicated system requires more time, it is necessary to split the HE into several sessions, which will fit the time restriction.

Inspection methods share a problem with the cognitive Walkthrough: The evaluators must be user *and* task experts. A further disadvantage of the method is that

> . . . it sometimes identifies usability problems without providing direct suggestions for how to solve them. The method is biased by the current mindset of the evaluators and normally does not generate breakthroughs in the evaluated design. (55, p. 255)

Although HE is part of the so-called "discount usability engineering methods" (27,56), there are more pessimistic results as well: The main argument for the "discount"—that there is a need for only a few experts to achieve acceptable results—has been questioned by a comparative study of Gediga and Hamborg (57). The authors show that the number of required experts depends on several variables, including the task, the layout of the evaluation, the quality of the evaluator, and the definition of a "usability problem."

Group Discussion

The object of the evaluation using group discussion is the complete user interface of the system, and different criteria can be used for the evaluation. Perhaps the most interesting aspect of group discussion is the possibility of using it as a creativity technique to generate evaluation criteria as a first step of the evaluation. Group discussion is quite flexible in its orientation: It may be user, task, or organisation oriented (12, p. 101). The application of the technique tends to be more efficient in the early stages of development.

Group discussion among user representatives and/or system designers is not a stand-alone technique, and it must be supported by other means. In the "Group Design Reviews" (50), several users discuss the user interface. The discussion has to be moderated by a human-factor expert; other experts such as designers, training experts, marketing experts, and so forth should complete the group if it seems necessary. A combination with a Walkthrough is straightforward (52) and more effective than a Walkthrough on its own (22, p. 706).

Other Techniques

Although we have listed quite a number of techniques, there are more variants and combinations which have been used in the past two decades.

A very interesting different point of view is taken by the *interpretative evaluation techniques* (21), which have moved away from the evaluator-controlled forms of evaluation to more informal techniques derived from anthropology and sociology. The techniques *contextual inquiry* (58), *cooperative evaluation* (28), and *participative evaluation* (59) advocate a more holistic treatment of evaluation. Up to now, no comparative study includes these techniques, but we are confident that at least the idea of the interpretative evaluation techniques will find its way into the HCI laboratories.

Modeling approaches aim at a formal description or prediction of the human–computer interaction. Their historic root is a model of an abstract human processor which sets the frame for a task description, given a more or less abstract system in a concrete environment (60). The concept has resulted in a number of developments such as the GOMS and NGOMSL approach (61), the task knowledge structure (TKS) and the knowledge analysis of tasks (KAT) (62), the task action grammars (TAG) (63), the external tasks–internal task mapping (ETIT) (64), and the yoked state space (YSS) (65). All these models share some characteristics. They offer convincing results (e.g., the comparison of aspects of the PC-DOS and the Macintosh interface using NGOMSL by Kieras (66), and they are used for comparatively small-scaled applications. Even though their advocates have asserted for the past decade that the models will be more applicable, if computer speed and memory are sufficient to fit the requirements of the models, there are some reservations concerning the validity of such methods for larger applications. It was noted that the human processor model does not take into consideration the huge individual differences among human subjects, as well as results from the large body of results in reaction-time research (67). Finally, it was shown that known results on learning or chunking processes is only considered if they fit smoothly into the proposed model. Because the human processor is therefore, at best, a rough approximation of human behavior, one can conclude that the results may well be acceptable for small-scaled examples but will run into problems if the model is applied to more demanding systems.

Comparison of Software Evaluation Techniques

There is a plethora of studies which deal with the comparison of evaluation techniques in terms of effectivity, efficiency, or utility (57,68–76). It is also investigated whether the information gathered by different methods are the same or not. Frequently, some of the "cheaper" predictive methods such as Heuristic Evaluation or Walkthroughs are benchmarked with a usability test method.

The results reported in the literature lead to the conclusion that an empirical comparison of software evaluation techniques is a minefield. First, researchers are faced with the usual problems encountered in empirical comparisons, such as finding adequate and representative testing subjects, objects, and situations, and a suitable methodology for analyzing the results. Second, as Gray and Salzman (77) pointed out, up to now, no study fulfills the rigorous standards of experimental design. In benchmark studies, confounding variables may intervene (e.g., different testing situations, different time of contact, different number of subjects performing the methods, etc.); therefore, it may not be appropriate to interpret the results in terms of difference of methods alone. Carroll (78, p. 309) commented,

> In their focus on conventional experiments, Gray and Salzman underestimate the importance and the distinctiveness of rare evaluation events.

The main argument in favor of comparison is that if one perceives the confounding variables as specific properties of the methods, comparison is a valuable evaluation of techniques, which may help practitioners to choose the "best" method for evaluating software. However, this pragmatic view puts heavy restrictions on the empirical studies, because a valid benchmarking is only feasible if the methods are applied exactly as they will be used in practice. Gray and Salzman (77) have shown that some studies do not fulfill these restrictions.

The situation is complicated because the definition of some methods is not very precise. If, for example, "the" technique of Usability Testing is compared with other methods, it is often not clear what is really meant. One can argue that there is no Usability Testing method as such, because the technique has shown high variability and evolution over the past two decades. The same holds—to some extent—for other techniques as well. Two classical studies shall demonstrate the problem:

- Jeffries et al. (72) compared Heuristic Evaluation with Software Guidelines, Cognitive Walkthrough, and Usability Testing. They showed that Heuristic Evaluation is more effective and efficient than Usability Testing (Rank 2) and the other two techniques in terms of detection of severe problems and of costs.
- Karat et al. (74) compare a variant of Heuristic Evaluation, Walkthroughs, and Usability Testing. They show, to the contrary, that most of the problems are detected by Usability Testing in a shorter time than by the other methods.

Taken together, the results of both studies (and of other cited studies as well) are inconclusive. Reviewing empirical comparison studies, Karat (79, p. 233) summarized as follows:

> Usability evaluation methods act as different types of filters of user interaction with a computer system.

This statement describes the state of knowledge about evaluation technique benchmarking well. Nevertheless, some lessons can be learned:

- Heuristic Evaluation offers at least satisfactory results. A combination of Heuristic Evaluation with a thinking-aloud technique seems to be a very effective strategy (72).
- Usability Testing procedures—and other descriptive techniques as well—often result in more information than predictive techniques. The findings show that predictive techniques, like Heuristic Evaluation, concentrate on severe problems only, whereas problems discovered by a descriptive technique address more specific—and sometimes "cosmetic"—aspects of the software.
- Usability Testing often needs more effort and equipment than comparable predictive techniques.
- Predictive evaluation techniques can be applied earlier in the life cycle than Usability Testing procedures.
- The goal of evaluation needs to be taken into account:

  If the evaluation goal is summative ("which one is better" or "how good"), the behavioral evaluation procedures applied in Usability Testing are the methods of choice.

  Neither Usability Testing nor pure expert-based techniques support participatory design (80). To achieve this, collaborative Walkthroughs or other group discussion techniques which include user representatives are necessary, because in these techniques, the user has the chance to be an active part of system development.

The decision of which evaluation technique(s) should be used has to be based on the concrete demands of the software development schedule, human-factor considerations, and cost–benefits issues (13, p. 699). The results of the above-mentioned comparison studies can only constitute a limited basis for such a decision. A systematic investigation of the "Return of Investment" (81,82) of usability activities in the software life cycle is still missing:

> The relative cost-effectiveness of usability testing and inspection methods, based on return of investment in implemented changes, is not clearly established at this time. (79).

EVALUATION MODELS

In contrast to software evaluation *techniques*, which were presented in the preceeding section, software evaluation *models* determine the frame of the evaluation, which consists of the following:

- Choosing techniques appropriate for the life cycle
- Setting the focus with respect to the objects under study and the measurement criteria

Evaluation models may provide a standardized treatment of establishing (potentially) successful procedures in the practice of evaluation and are a necessary tool for a comparing different types of software evaluation. Any descriptive evaluation procedure must be combined with some kind of predictive technique to result in an applicable evaluation model; furthermore, some preparatory steps are necessary. For example, the evaluation model, which consists of the IsoMetric^L questionnaire as a basic technique, standardizes

the preparatory steps "choosing the tasks" and "choosing the user group(s)"; it also standardizes the preparation of the report by an expert and the structure of a "usability review", which consists of a standardized result presentation and a Walkthrough technique (43).

There are three classes of evaluation model:

Method-driven models: These models offer a frame for software evaluation based on a collection of techniques. The models are only applicable if the evaluation procedures fits the problems encountered by the user and the system developers perfectly.

Criteria-driven models: More or less abstract criteria are defined and refined; the evaluation in these models aims at a measurement of the criteria.

Usability Engineering: These are evaluation models which are driven by the needs of a specific life-cycle model.

Method-Driven Evaluation Models

The center of method-driven evaluation models consists of the arrangement of evaluation techniques, amended by the regulation of preparatory and subsequent tasks. A method-driven evaluation model can be perceived as a complex evaluation technique as well. An example is EVADIS II (83,84), which is well tested for office automatization software. The EVADIS II model combines interviews, simplified task analysis, and expert judgment in the following steps:

1. Installation and exploration of the software
2. Analysis and relevance weightings of the tasks; construction of test tasks
3. Analysis of the user characteristics; selection of relevant ergonomic test items
4. Evaluation of the software, based on the results of the first three steps
5. Interpretation of the results and composing a test report

The first three preparatory steps can be handled in parallel; they result in testing tasks and a list of ranked ergonomic evaluation criteria, mainly based on the principles of ISO 9241 (Part 10). The ranks of the criteria are deduced from the user profile and they determine the test items which are chosen from an item database. In Step 4, the testing tasks are evaluated by an expert who steps through the tasks and answers the questions formulated in the ergonomic items. The recorded answers form the basis for the test report. Every step of EVADIS II is supported by a large amount of supporting material such as databases and guidelines, which allows a domain expert with only a small amount of knowledge of software evaluation to form a well-founded opinion on the topic.

Criteria-Driven Evaluation Models

Criteria-driven evaluation models start with assumptions about the structure of the design process in which criteria are defined and give advice on how to derive measurables form the criteria. Because these models focus on criteria and measurables, there is no close connection to a design model. Standards such as ISO 9241 or ISO/IEC 9126 can constitute a basis for a criteria-driven evaluation model. As an example, we discuss the "Evaluation Process Model" ISO/IEC 9126 (1991) (3,85), which is the basis for many software quality assurance procedures. The standard ISO/IEC 9126 aims at a product evaluation

from a software quality point of view. It defines a specific process model and some general quality characteristics; for example, the following:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Although "Usability" appears as a criterion, ISO/IEC 9126 does not aim at the ergonomic quality of the software:

> Usability defined in this International Standard as a specific set of attributes of software product differs from the definition from an ergonomic point of view, where other characteristics such as efficiency and effectiveness are also seen as constituents of usability. (3, p. 3)

After the relevant quality characteristics and their weightings have been selected, the evaluation process is performed in three steps.

1. Fixing the requirements: The requirements are derived from the application context of the system. The quality characteristics are formulated concretely in terms of observables, which operationalize the criteria and meet the requirements.
2. Preparation of the evaluation: This step comprises the operationalization of the criteria into metrics and composite measurables.
3. The evaluation step: The evaluation of the product or a part of the product is performed on the basis of the derived requirements and chosen metrics.

Because ISO/IEC 9126 offers a general framework for software evaluation within the context of software quality assurance, we present the tasks accompanying the three steps of evaluation in more detail in Figure 1.

Usability Engineering

Usability Engineering is concerned with the systematic integration of methods and techniques of building usable software in the system development process. It can be characterized as a process which covers the definition and the measure of product usability in general (7, p. 654). Usability Engineering models coordinate the "Usability Engineering activities" and do not replace traditional models of the software engineering process. The evaluation of software is a prominent part of any Usability Engineering model (7,14,86,87).

The models are committed to the approach of the User-Centered Design (88), which is based on the assumption of an iterative system design with user participation (80,86). In this process, the user is regarded not as a producer of usability data, but should be directly and actively involved in the design process:

> I strongly recommend that all team members carry out life-cycle tasks jointly, so that they develop a shared understanding of the requirements and designed issues. In addition, it can be extremely effective to have users participate in many Usability Engineering tasks, not just as objects of study or sources of information, but as active participants. (87, p. 22)

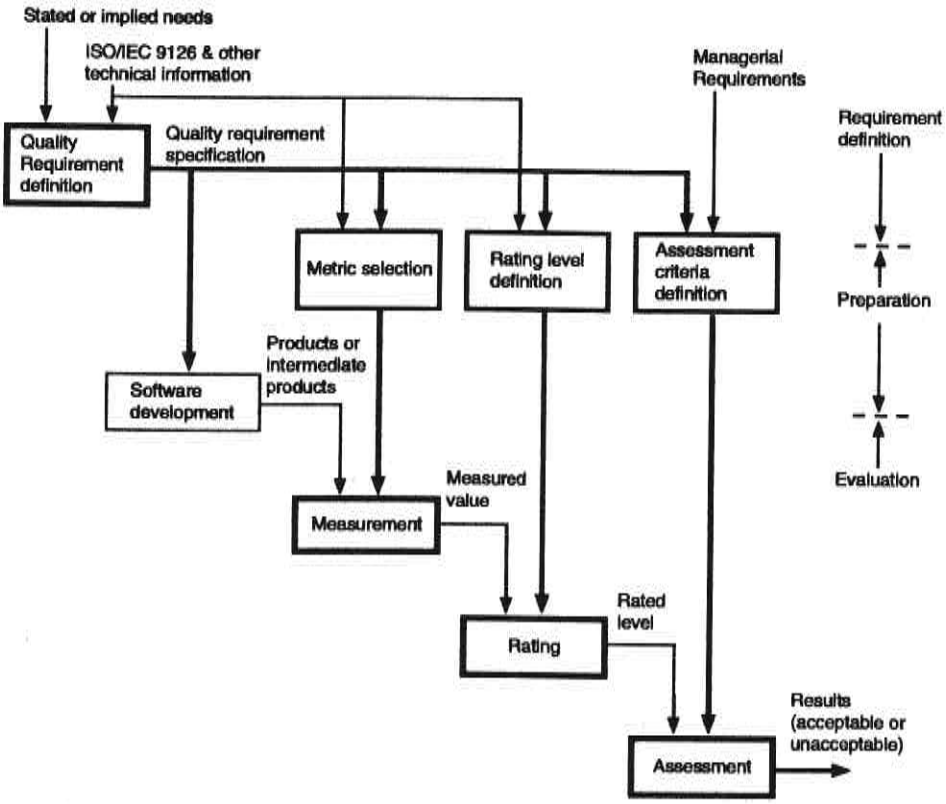

FIGURE 1 The ISO/IEC 9126 evaluation model.

Usability Engineering requires a software engineering model, which allows revisions and feedback loops. These models include the prototyping approaches, iterative system design, and user involvement (5,7,14,87).

Models of Usability Engineering are usually subdivided in three phases (5,14,87,89):

Phase 1: Analysis and specification. The first step—the "Gearing-up Phase"—starts with preparatory activities, such as choosing general principles for the design (e.g., relevant standards, development models, and tools). The next step—the "Initial Design Phase" or "Requirements Analysis"—is concerned with the characterization of users and the setup of user profiles; additionally, task and workflow have to be analyzed. The obtained information is used to plan the activities of the "Work Reengineering" and for the design of the user interface. As a result of this process, usability specifications are developed, which consist of goals for user-oriented design and behavior-oriented measurables of these goals. The outcome of this phase is summarized in a product style guide (87).

Phase 2: Iterative development. The results of the preceding phase are used to design the organizational and workflow part of the system. Conceptual models are developed which can be used to produce paper-and-pencil or prototype mock-ups. Using an iterative design, the prototypes are evaluated and changed continuously. This helps to identify and remove major usability bugs. If the conceptual model shows satisfactory results, the screen design is developed, using screen design standards, and evaluation of the screen design takes place with the aid of a prototype. As in Phase 1, the results are summarized in a product style guide.

Phase 3: System installation. The final phase is concerned with "system installation" and "user training." Furthermore, the acceptance of the system has to be assured, and system support and maintenance must be organized. Software evaluation procedures in the application phase have to be planned in order to obtain feedback about the usability of system. This information can be used for the design of new releases.

The standard ISO/DIS 13407 (90) is closely connected to the Usability Engineering approach. It describes three different usability activities in system design:

1. Analysis and specification of the context of usage and the needs of the users and organizational needs
2. The system design
3. The evaluation of the system design using user-oriented criteria

Similar to the Usability Engineering approach, ISO/DIS 13407 uses the principles

- Prototyping, iterative design, direct user participation, iterative evaluation, quality enhancement of the system

in all phases of the design process.

In case that the evaluation cannot be performed by users, ISO/DIS 13407 recommends evaluation by experts, but it is advocated that a user-based evaluation is done at a later step of system development. The evaluation–(re)design cycle is repeated until the goals of the user-centered design are fulfilled. The product can be evaluated to ascertain whether it fulfills the user-oriented requirements and/or whether it is in accordance with other international standards such as ISO 9241 (91).

A Problem: The Integration of Software Evaluation Models and Software Design Models

It is pointed out in ISO/DIS 13407 that "user-oriented activities" have to be integrated into the system development process, and the standard gives some advice how this can be done. This is an urgent need, because, in classical approaches, the combination of system design with software evaluation models, in general, and the Usability Engineering models, in particular, seems to be more an art than a science. Although some case studies show that an effective combination of both model types can be achieved (92–94). Mayhew (87, p. 4) points out that " . . . Gould and Lewis did not specify exactly how these global usability strategies could be integrated into an overall software development life-cycle . . . " and recommends her own approach as an alternative. She admits, however, that

> . . . the exact overlapping of the two types of tasks is not completely clear. The most appro-

> priate relationship between them is probably not absolute, but is most likely dependent on various project-specific factors and on the particular software development methodology with which the life-cycle is integrated . . . For example, traditional systems analysis and Contextual Task Analysis are highly related tasks, but they are not the same thing and require very different kinds of expertise. Exactly how to integrate them to avoid duplication of effort and yet produce the intended and necessary outputs of both is a topic for future work. (87, p. 17)

To date, there are only loose connections of activities of Usability Engineering and the object-oriented design methodology. These are concentrated in the requirements analysis and in the design/testing/development cycle. The generation of an object-oriented "Requirement model" can be supported by user profiles, which can be obtained by task and workflow analysis. "Work Reengineering Tasks" correspond to the construction of an "Analysis Model" in object-oriented design. The design and programming of the user interface in the object-oriented model (the "Design Model") can be supported by conceptual models (87, p. 30).

FUTURE TASKS

There are many evaluation techniques and few evaluation models which can be applied in practice. Although much has been achieved, there is a dire need to devote more effort to developing applicable models for practical use—in particular, if nonexperts evaluate software as in smaller companies. EVADIS is an instance of such a model, as it combines several evaluation techniques to a coherent evaluation model in the domain of office automation.

At present, the differences in effectiveness, efficiency, and the cost–benefit relation of software evaluation techniques and models are not satisfactory. Due to the huge number of techniques and their variants, the labor of comparing these techniques—and models using them—would be tremendous. An empirical stock-taking of frequently used models should be done to form a basis for future comparison studies. These should not only consider effectiveness, efficiency, and return of investment but other effects on the software as well, such as time efficiency, suitability for learning, or user satisfaction (49).

For a number of techniques, there is a need to investigate where they will fit into a software development cycle. The rule of thumb "early → predictive & late → descriptive" should be replaced by a more differentiated rule system. Because most findings demonstrate that the techniques produce a large amount of overlapping results, the effect of combining evaluation techniques needs to be evaluated as well. Inspecting the evaluation models, we see that there should be a higher integration of models as well. In particular, the goals and techniques of Requirements Analysis overlap with the goals and techniques of software evaluation in such a way that unified approach would save time and money.

Most software evaluation techniques we have presented aim at the operationalization of "Usability" of the software. The investigation of how "Usability" can be described in theoretical terms is still ongoing. "Usability" is a combination of at least two different kinds of construct: the "ease of use" and the "enjoyment to use". Whereas older studies and methods had a closer look at the "ease of use" component, the "enjoyment to use" component has been the focus of newer studies (at least since SUN and SAP proclaimed

that even office software should be enjoyable). There is some evidence that both components vary independently, which implies that there is also a need for an independent measure of both aspects.

Even though much effort and research are still necessary to develop efficient software evaluation procedures, we hope that the article has demonstrated that there is not only a large body of techniques but also that these techniques are applicable and valuable in the development of software systems.

REFERENCES

1. E. A. Suchman, *Evaluation Research: Principles and Practices in Public Service and Social Action Programs*, Russel, New York, 1967.
2. D. Hix and H. R. Hartson, *Developing User Interfaces: Ensuring Usability Through Product and Process*, John Wiley & Sons, New York, 1993.
3. ISO/IEC, *ISO/IEC 9126. Information technology—Software product evaluation—Quality characteristics and guidance for their use*, ISO, Geneva, 1991.
4. A. Whitefield, F. Wilson, and J. Dowell, "A Framework for Human Factors Evaluation," *Beh. Inform. Technol., 10*, 65–79 (1991).
5. J. D. Gould, "How to Design Usable Systems," in *Handbook of Human Computer Interaction*, M. Helander (ed.), Elsevier, Amsterdam, 1988, pp. 757–789.
6. J. Nielsen, "The Usability Life Cycle," *IEEE Computer, 25*, 12–22 (1992).
7. D. Wixon and C. Wilson, "The Usability Engineering Framework for Product Design and Evaluation," in *Handbook of Human–Computer Interaction*, 2nd ed., M. Helander, T. Landauer, and P. Prabhu (eds.), Elsevier, Amsterdam, 1997, pp. 653–688.
8. T. L. Roberts and T. P. Moran, " The Evaluation of Text Editors: Methodology and Empirical Results," *Commun. ACM, 26*, 265–283 (1983).
9. J. M. Carroll and M. B. Rosson, Usability Specifications as a Tool in Iterative Development," in *Advances in Human–Computer Interaction*, H. R. Hartson (ed.), Ablex, Norwood, NJ, 1985, pp. 1–28.
10. M. Scriven, "The Methodology of Evaluation," in *Perspectives of Curriculum Evaluation*, R. Tyler, Gagne, and M. Scriven (eds.), Rand McNally, Chicago, 1967, pp. 39–83.
11. W. Dzida, "Qualitätssicherung durch software-ergonomische Normen," in *Einführung in die Software-Ergonomie*, E. Eberleh, H. Oberquelle, and R. Oppermann (eds.), de Gruyter, Berlin, 1994, pp. 373–406.
12. W. Hampe-Neteler, *Software-ergonomische Bewertung zwischen Arbeitsgestaltung und Software-Entwicklung*, Lang, Frankfurt, 1994.
13. J. Karat, "User-Centered Software Evaluation Methodologies," in *Handbook of Human–Computer Interaction*, 2nd ed., M. Helander, T. K. Landauer, and P. Prabhu (eds.), Elsevier, Amsterdam, 1997, pp. 689–704.
14. J. Nielsen, *Usability Engineering*, AP Professional, Boston, 1993.
15. ISO, *EN ISO 9241-11. Ergonomic requirements for office work with visual display terminals (VDT's)*. Part 11, ISO, Geneva, 1997.
16. ISO, *EN ISO 9241-10. Ergonomic requirements for office work with visual display terminals (VDT's)*. Part 10, ISO, Geneva, 1996.
17. G. Gigerenzer, *Messung und Modellbildung in der Psychologie*, Birkhäuser, Basel, 1981.
18. D. A. Tyldesley, "Employing Usability Engineering in the Development of Office Products," *Computer J., 31*, 431–436 (1988).
19. J. Nielsen, "Heuristic Evaluation," in *Usability Inspection Methods*, J. Nielsen and R. Mack (eds.), John Wiley & Sons, New York, 1994, pp. 25–62.

20. J. Nielsen and R. L. Mack, *Usability Inspection Methods*, John Wiley & Sons, New York, 1994.
21. G. Walsham, *Interpreting Information Systems in Organisations*, John Wiley & Sons, Chichester, 1993.
22. J. Preece, *Human–Computer Interaction*, Addison-Wesley, Harlow, U.K., 1999.
23. B. L. Harrison, "Video Annotation and Multimedia Interfaces: From Theory to Practice," in *Proceedings of the Human Factor Society 35th Annual Meeting*, 1991, pp. 319–322.
24. J. M. Carroll and R. Mack, "Learning to Use a Word Processor: By Doing, by Thinking and by Knowing," in *Human Factors in Computing Systems*, J. Thomas and M. Schneider (eds.), Ablex, Norwood, NJ, 1984, pp. 13–52.
25. A. H. Jørgensen, "Using the Thinking-Aloud Method in System Development," in *Designing and Using Human–Computer Interfaces and Knowledge Based Systems*, G. Salvendy and M. J. Smith (eds.), Elsevier, Amsterdam, 1989, pp. 743–750.
26. C. Lewis, "Using the "Thinking Aloud" Method in Cognitive Interface Design," IBM Research Report RC 9265 (40713), IBM Thomas J. Watson Research Center (1982).
27. J. Nielsen, "Usability Engineering at a Discount," in *Designing and Using Human–Computer Interfaces and Knowledge Based Systems*, G. Salvendy and M. Smith (eds.), Elsevier, Amsterdam, 1989, pp. 394–401.
28. A. Monk, P. Wright, J. Haber, and L. Davenport, *Improving Your Human–Computer Interface: A Practical Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1993.
29. V. A. Bowers and H. L. Snyder, "Concurrent vs Retrospective Verbal Protocol for Comparing Window Usability," in *Proceedings of the Human Factors Society 34th Annual Meeting of the Human Factors Society*, 1990, pp. 1270–1274.
30. K. R. Ohnemus and D. W. Biers, "Retrospective vs Concurrent Thinking out Loud in Usability Testing," in *Proceedings of the Human Factors and Ergonomics Society 37th Annual Meeting*, 1993, pp. 1127–1131.
31. S. Greif, "Organizational Issues and Task Analysis," in *Human Factors for Informatics Usability*, B. Shackel and S. Richardson (eds.), Cambridge University Press, Cambridge, 1991, pp. 247–266.
32. K.-C. Hamborg and S. Greif, "Heterarchische Aufgabenanalyse," in *Handbuch psychologischer Arbeitsanalyseverfahren*, H. Dunckel (ed.), VDF, Zürich, 1999, pp. 147–177.
33. S. Greif, "The Role of German Work Psychology in the Design of Artifacts," in *Designing Interaction. Psychology at the Human–Computer Interface*, J. M. Carroll (ed.), Cambridge University Press, Cambridge, 1991, pp. 203–226.
34. J. Kirakowski and M. Corbett, *Effective Methodology for the Study of HCI*, North-Holland, Amsterdam, 1990.
35. J. Nielsen, R. L. Mack, K. H. Bergendorff, and N. L. Grischkowsky, "Integrated Software Usage in the Professional Work Environment: Evidence from Questionnaires and Interviews," in *Human Factors in Computing Systems, CHI'86 Conference Proceedings*, M. Mantei and P. Obertson (eds.), ACM, New York, 1986, pp. 162–167.
36. B. Shneiderman, *Designing the User Interface. Strategies for Effective Human–Computer Interaction*, 2nd ed., Addison-Wesley, Reading, MA, 1992.
37. J. Kirakowski and M. Corbett, "SUMI: The Software Usability Measurement Inventory," *Br. J. Educ. Technol., 24*, 210–212 (1993).
38. G. Gediga, K.-C. Hamborg, and I. Düntsch, "The IsoMetrics Usability Inventory: An Operationalisation of ISO 9241-10," *Beh. Inform. Technol., 18*, 151–164 (1999).
39. J. P. Chin, V. A. Diehl, and K. L. Norman, "Development of an Instrument Measuring User Satisfaction of the Human–Computer Interface," in *Proceedings of SIGCHI '88*, ACM/SIGCHI, New York, 1988, pp. 213–218.
40. B. Shneiderman, *Designing the User Interface. Strategies for Effective Human–Computer Interaction*, 3rd ed., Addison-Wesley, Reading, MA, 1998.

41. J. Kierakowski, "The Use of Questionnaire Methods for Usability Assessment," 2000; http://www.ucc.ie/hfg/questionnaires/sumi/sumipapp.html
42. H. Willumeit, G. Gediga, and K.-C. Hamborg, "Isometrics^{L}: Ein Verfahren zur formativen Evaluation von Software nach ISO 9241/10," *Ergon. Inform.*, *27*, 5–12 (1996).
43. G. Gediga, K.-C. Hamborg, and H. Willumeit, *The IsoMetrics Manual (Version 1.15)*, Universität Onabrück, Fachbereich Psychologie, Osnabrück, 1997; http://www.eval-institut.de/isometrics
44. M. Burmester, C. Görner, P. H. Vossen, T. M. Zolleis, and V. Zouboulides, "Qualitatives software screening," in *Das SANUS-Hanbuch. Bildschirmarbeit EU-konform*, M. Burmester, C. Görner, W. Hacker, M. Kärcher, P. Kurtz, U. Lieser, W. Risch, R. Wieland-Eckelmann, and H. Wilde (eds.), Dortmund. Schriftenreihe der Bundesanstalt für Arbeitsschutz und Arbeitsmedizin, FB 760, 1997, Teil II, 2.
45. J. Prümper, "Software-Evaluation Based upon ISO 9241 Part 10," in *Human–Computer Interaction. Vienna Conference, VCHCHI '93*, T. Grechening and M. Tescheligi (eds.), Springer-Verlag, Berlin, 1993.
46. P. A. Holleran, "A Methodological Note on Pitfalls in Usability Testing," *Beh. Inform. Technol.*, *10*, 345–357 (1991).
47. J. Rubin, *Handbook of Usability Testing*, John Wiley & Sons, New York, 1994.
48. C. Robson, "Designing and Interpreting Psychological Experiments," in *Human–Computer Interaction*, J. Preece and L. Keller (eds.), Prentice-Hall, Hemel Hempstead, U.K., 1990, pp. 357–367.
49. C.-M. Karat, "Cost–Benefit and Business Case Analysis of Usability Engineering," in *Bridges Between Two Worlds, INTERCHI '93. Tutorial Notes 23*, S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, and T. White (eds.), Addison-Wesley, Reading, MA, 1993.
50. R. A. Virzi, "Usability Inspection Methods," in *Handbook of Human–Computer Interaction*, 2nd ed., M. Helander, T. Landauer, and P. Prabhu (eds.), Elsevier, Amsterdam, 1997, pp. 705–715.
51. R. Bias, "Walkthroughs: Efficient collaborative testing," *IEEE Software*, *8*, 94–95 (1991).
52. R. Bias, "The Pluralistic Walkthrough: Coordinated Empathies," in *Usability Inspection Methods*, J. Nielsen and R. Mack (eds.), John Wiley & Sons, New York, 1994, pp. 63–76.
53. C. Lewis and C. Wharton, "Cognitive Walkthroughs," in *Handbook of Human–Computer Interaction*, 2nd ed., M. Helander, T. Landauer, and P. Prabhu (eds.), Elsevier, Amsterdam, 1997, pp. 717–732.
54. C. Wharton, J. Rieman, C. Lewis, and P. Polson, "The Cognitive Walkthrough Method: A Practitionert's Guide," in *Usability Inspection Methods*, J. Nielsen and R. Mack (eds.), John Wiley & Sons, New York, 1994, pp. 105–140.
55. J. Nielsen and R. Molich, "Heuristic Evaluation of User Interfaces," in *Chi'90 Conference Proceedings, Empowering People*, J. Chew and J. Whiteside (eds.), ACM, New York, 1990, pp. 249–256.
56. K. Potosnak, "Big Paybacks from 'Discount' Usability Engineering," *IEEE Software*, 107–109 (1990).
57. G. Gediga and K.-C. Hamborg, "Heuristische Evaluation und IsoMetrics: Ein Vergleich," in *Software-Ergonomie '97*, R. Liskowsky, B. Velichkovsky, and W. Wünschmann (eds.), Teuber, Stuttgart, 1997, pp. 145–155.
58. J. Whiteside, J. Bennett, and K. Hotzblatt, "Usability Engineering: Our Experience and Evolution," in *Handbook of Human–Computer Interaction*, M. Helander (ed.), Elsevier, Amsterdam, 1988, pp. 791–817.
59. J. Greenbaum and M. Kyng, *Design at Work: Cooperative Design of Computer Systems*, Lawrence Erlbaum, Hillsdale, NJ, 1991.
60. S. Card, T. P. Moran, and A. Newell, *The Psychology of Human–Computer Interaction*, Lawrence Erlbaum, Hillsdale, NJ, 1983.
61. D. Kieras, "A Guide to GOMS Model Usability Evaluation Using NGOMSL," in *Handbook*

of Human–Computer Interaction, 2nd ed., M. Helander, T. Landauer, and P. Prabhu (eds.), Elsevier, Amsterdam, 1997, pp. 733–766.

62. P. Johnson, *Human–Computer Interaction: Psychology, Task Analysis and Software-Engineering*, McGraw-Hill, Maidenhead, U.K., 1992.
63. S. J. Payne and T. R. G. Green, "Task-Action Grammar: The Model and Its Development," in *Task Analysis for Human–Computer Interaction*, D. Diaper (ed.), Ellis Horwood, Chichester, 1989.
64. T. P. Moran, "Getting into System: External Task–Internal Task Mapping Analysis," in *Human Factors in Computing CHI'83 Conference Proceedings*, A. Janda (ed.), ACM, New York, 1983.
65. S. J. Payne, "Complex Problem Spaces: Modelling the Knowledge Needed to Use Interactive Devices," in *Proceedings of the IFIP Conference on Human–Computer Interaction*, H.-J. Bullinger and B. Shackel (eds.), North-Holland, Amsterdam, 1987.
66. D. Kieras, "Bridges Between Worlds," in *INTER CHI'93. Tutorial Notes 5*, S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, and T. White (eds.), Addison-Wesley, Reading, MA, 1993.
67. S. Greif and G. Gediga, "A Critique and Empirical Investigation of the 'One-Best-Way-Models,' in Human–Computer Interaction," in *Psychological Issues of Human Computer Interaction in the Work Place*, M. Frese, E. Ulich, and W. Dzida (eds.), Elsevier, Amsterdam, 1987, pp. 357–377.
68. H. Desurvire, D. Lawrence, and M. E. Atwood, "Empiricism Versus Judgement: Comparing User Interface Evaluation Methods," *ACM SIGCHI Bull., 23*, 58–59 (1991).
69. H. W. Desurvire, J. M. Kondziela, and M. Atwood, "What Is Gained and Lost When Using Evaluation Methods Other Than Empirical Testing," in *Proceedings of HCI '92, People and Computers VII*, Cambridge University Press, Cambridge, 1992, pp. 89–102.
70. K.-C. Hamborg, G. Gediga, M. Döhl, P. Janssen, and F. Ollermann, "Softwareevaluation in Gruppen oder Einzelevaluation: Sehen zwei Augen mehr als vier?" in *Software-Ergonomie '99: Design von Informationswelten*, U. Arend and K. Pitschke (eds.), Teubner, Stuttgart, 1999, pp. 97–109.
71. R. D. Henderson, M. C. Smith, J. Podd, and H. Varela-Alvarez, "A Comparison of the Four Prominent User-Based Methods for Evaluating the Usability of Computer Software," *Ergonomics, 38*, 2030–2044 (1995).
72. R. Jeffries, J. R. Miller, C. Wharton, and K. M. Uyeda, "User Interface Evaluation in the Real World: A Comparison of Four Techniques," in *Proceedings AMC CHI '91 Conference on Human Factors in Computing Systems*, ACM, New York, 1991, pp. 119–124.
73. R. J. Jeffries and H. W. Desurvire, "Usability Testing vs Heuristic Evaluation: Was There a Contest?" *ACM SIGCHI Bull., 4*, 39–41 (1992).
74. C. Karat, R. L. Campbell, and T. Fiegel, "Comparison of Empirical Testing and Walkthrough Methods in User Interface Evaluation," in *Proceedings ACM CHI '92 Conference*, P. Bauersfield, J. Bennet, and G. Lynch (eds.), ACM, New York, 1992, pp. 397–404.
75. E. D. Smilowitz, M. J. Darnell, and A. E. Bensson, "Are We Overlooking Some Usability Testing Methods? A Comparison of Lab, Beta, and Forum Tests," *Beh. Inform. Technol.*, 13, 183–190 (1994).
76. R. A. Virzi, J. F. Sorce, and L. B. Herbert, "A Comparison of Three Usability Evaluation Methods: Heuristic, Think Aloud, and Performance Testing," in *Designing for Diversity: Proceedings of the Human Factors and Ergonomics Society 37th Annual Meeting 1993*, Human Factors and Ergonomics Society, Santa Monica, CA, 1993, pp. 309–313.
77. W. D. Gray and M. C. Salzman, "Damaged Merchandise? A Review of Experiments That Compare Usability Evaluation Methods," *Human–Computer Interact., 13*, 203–261 (1998).
78. J. M. Carroll, "Review Validity, Causal Analysis, and Rare Evaluation Events," *Human–Computer Interact.*, 13, 308–310 (1998).
79. C.-M. Karat, "A Comparison of User Interface Evaluation Methods," in *Usability Inspection*

Methods, J. Nielsen and R. L. Mack (eds.), John Wiley & Sons, New York, 1994, pp. 203–233.

80. M. J. Muller, J. H. Halkswanter, and T. Dayton, "Participatory Practices in the Software Lifecycle," in *Handbook of Human–Computer Interaction*, 2nd ed., M. Helander, T. Landauer, and P. Prabhu (eds.), Elsevier, Amsterdam, 1997, pp. 255–313.
81. C.-M. Karat, "Cost-Justifying Usability Engineering in the Software Life Cycle," in *Handbook of Human–Computer Interaction*, 2nd ed., M. Helander, T. K. Landauer, and P. Prabhu (eds.), Elsevier, Amsterdam, 1997, pp. 767–777.
82. M. M. Mantei and T. J. Teorey, "Cost/Benefit Analysis for Incorporating Human Factors in the Software Lifecycle," *Commun. ACM*, *31*, 428–439 (1988).
83. H. Reiterer, *User Interface Evaluation and Design. Research Results of the Projects Evaluation of Dialogue Systems (EVADIS) and User Interface Design Assistance (IDA)*, Oldenburg, München, 1994.
84. H. Reiterer and R. Oppermann, "Evaluation of User Interfaces. EVADIS II—A Comprehensive Evaluation Approach," *Beh. Inform. Technol.*, *12*, 137–148 (1993).
85. R. C. Williges, B. H. Williges, and J. Elkerton, "Software Interface Design," in *Handbook of Human Factors*, G. Salvendy (ed.), John Wiley & Sons, New York, 1987, pp. 1416–1449.
86. J. M. Carroll, "Human–Computer Interaction: Psychology as a Science of Design," *Int. J. Human–Computer Studies*, *46*, 501–522 (1997).
87. D. Mayhew, *The Usability Engineering Lifecycle. A Practitionert's Handbook for User Interface Design*, Morgan Kaufmann, San Francisco, 1999.
88. J. Karat, "Evolving the Scope of User-Centered Design," *Commun. ACM*, *40*, 33–38 (1997).
89. J. D. Gould, S. J. Boies, and J. Ukelson, "How to Design Usable Systems," in *Handbook of Human–Computer Interaction*, 2nd ed., M. Helander, T. Landauer, and P. Prabhu (eds.), Elsevier, Amsterdam, 1997, pp. 705–715.
90. ISO, *ISO 13407. Human centered design processes for interactive displays*, ISO, Geneva, 1997.
91. A. Çakir and W. Dzida, "International Ergonomic HCI Standards," in *Handbook of Human–Computer Interaction*, 2nd ed., M. Helander, T. Landauer, and P. Prabhu (eds.), Elsevier, Amsterdam, 1997, pp. 407–420.
92. J. D. Gould, S. J. Boies, S. Levy, J. T. Richards, and J. Schoonard, "The 1984 Olympic Message System: A Test of Behavioral Principles of System Design," *Commun. ACM*, *30*, 758–769 (1987).
93. J. D. Gould, S. J. Boies, and C. Lewis, "Making Usable, Useful, Productivity Enhancing Computer Applications," *Commun. ACM*, *34*, 74–85 (1991).
94. J. Nielsen, "Iterative User-Interface Design," *IEEE Computer*, *26*, 32–41 (1993).

GÜNTHER GEDIGA

KAI-CHRISTOPH HAMBORG

IVO DÜNTSCH