Towards an Integrated Model for Specifying and Measuring Quality in Use

Mohammad K. Donyaee

A Thesis

In

The Department

Of

Computer Science

Presented in partial fulfillment of the requirements
For the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

March 2001

ABSTRACT

Towards an Integrated Model for Specifying and Measuring Quality in Use

Mohammad K. Donyaee

Although usability has received widespread attention within the software engineering community, there are few agreed software quality models that capture our current meaning of usability. The Human Computer Interaction community has also developed different models for specifying or measuring usability. However, HCI models are not well integrated within the software engineering models. Another problem with those models is the lack of tools that makes them hard to use and apply. These limitations are the main motivations for our Quality in Use Integrated Model (QUIM).

In this thesis, the similarities and differences of various standards and models for defining usability, specifying, measuring or predicting quality in use are reviewed, and a comparison between the software engineering approaches and the HCI standards is committed. Then, we describe the rationale and foundations of QUIM, an integrated framework that aims to integrate these diverse models while supporting the main activities related to quality in use during the software lifecycle, including specification, testing, measurement and improvement.

Table of contents

List of Figures

List of tables

Chapter 1:

From System Quality to Usability and Quality in Use

Every day we use the word "Quality" as an attribute for a human made product just to compare it with other products in the same category, we say it has a high quality or that one has a low quality. We use it, but never pay attention to the fact that this concept is likely vague. Kitchenham notes that "quality is hard to define, impossible to measure, easy to recognize"[16], Gillies states that quality is "transparent when presented, but easily recognized in its absence"[17]. By today we have numerous solutions and methods in hand, to control a production process and acquiring the required quality on the final product. But there are lots of problems with every solution. No quality assurance method is perfect. Because in every method, we are dependent on its definition of quality. Using every method needs its required experiences and has its own expenses. We are confronted with the same problem in software quality and more specifically in system usability quality.

In this chapter, we will take a look at different definitions of quality. Also we will review software quality, usability and quality in use. Finally we will present briefly how people in software industry, today evaluate the usability of a product.

1.1 The concept of quality

1- ISO 9000 says the degree of quality of a product is how it satisfies its requirements. So even if a product from scratch has no appropriate requirements, still it could be a high quality product. Garvin called it "Manufacturing quality"[15].

2- ISO 8402 says quality is the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs. It means the presence of specified features. This is a product orientation view, i.e. an inherent characteristic of the product determined by the presence or absence of measurable product attributes. Garvin calls it "Product quality"[15].

It is obvious that if we want to be satisfied while using a product, it is supposed to present both types of quality. So every quality expert believes that we need to develop rules and evaluation methods and to apply them to both, Product and Production processes.

Now let's take a look at different definitions of software quality:

1- IEEE 1061-1998 says software quality is the degree to which software possesses a desired combination of quality attributes. And in turn a quality attribute is a characteristic of the software.

2- ISO/IEC 9126 says software quality is the totality of features and characteristics of a software product that bear on its ability to satisfy stated or implied needs.

3- Schulmeyer and McManus define software quality as "The fitness for use of the total software product"[18].

There are different views to the quality. Garvin has defined five views:

- **The transcendental view**: This view sees quality as something that can be recognized, but not defined, e.g. what an ugly interface.
- **The user view**: The user sees quality as fitness for his/her purpose.

- **The manufacturer view**: The user observes quality as conformance to specification.
- **The product view**: This view sees quality as tied to inherent characteristics of the product.
- **The value-based view**: This view sees quality as dependent on what a customer is willing to pay for it, i.e. a software product could exist without any fault and with the highest degree of quality in any sense, but for what price?

ISO/IEC 9126 has defined three perspectives of software quality:

(1) **User view**: The definition of quality in ISO 8402 reflects the user view. The user is mainly interested in using the software, its performance and the effects of using it. Then the user is not interested in internal aspects of the product. The user just sees observable external attributes of the software. Or one may say user is interested in final product quality.

(2) **Developer view**: The developer is interested in internal attributes of software. Of course there are some common attributes for user view and developer view, but they see the common attribute in different ways. As an example user and developer both are interested in performance of the software, but user could see this attribute as response time to the event entered by him, and developer thinks of it as data structure depth or path length.

(3) **Manager view**: A manager likely is interested in the overall quality rather than in a specific characteristic or attribute. The managers' view of software quality is

pragmatic and relatively simple. From this perspective a high quality software is the one that "works well enough" to serve its intended functions and is "available when needed" to perform those functions. Managers concern about risk assessment in software quality. Sometimes managers balance the quality improvement with management criteria such as schedule delay or cost overrun.

1.2 Usability as a Quality Factor

Two approaches for usability assurance are recognized. One is a "top-down" approach that is concerned with usability as a broad quality objective: the ability to use a product for its intended purpose in a specified context. The other is a product-oriented "bottom-up" view that is concerned with attributes of the user interface that make an interactive system easier to use.

1.2.1 Usability as a high-level quality objective

Disciples of this approach relate to usability as a high-level quality objective, and usability is defined in this way in the ISO 9241-11 standard as:

The extent to which, a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

ISO 9241-11 supports the following activities:

- Specification of usability goal and metrics as well as evaluation against these requirements (ISO 9241-11 and ISO/IEC 14598-1)
- Definition of the activities necessary in the development lifecycle for achieving quality in use (ISO/IEC 13407).

1.2.2 Usability as an independent contribution to software quality

In the product-oriented view, usability is seen as an independent factor to software quality. It deals with attributes of a software product, mainly its user interface that makes it easy to use. Usability is defined in this way in ISO/IEC 9126 as:

A set of attributes of software that bear on the effort needed for use and on the individual assessment of such use by a stated or implied set of users

The standard in its revision ISO/IEC 9126-1 defines usability as [62]:

The capability of the software to be understood, learned, used and liked by the user when used under specified conditions.

This definition can be used in the following ways:

- To specify details of the look and feel as well as the behavior of the user interface
- To provide detailed guidance on the design of easy to use user interface
- To provide metrics/goals for the evaluation of an interactive product

Even if it appears that, the first definition of ISO/IEC 9126 and ISO 9241-11 are opposite, they are in reality complementary. The interactive system has no intrinsic usability, only a capability to be used in a particular context of use. The ISO 9241-11 standard on usability guidance can be used to help understand the context in which particular attributes specified in ISO 9126 may be required. The definition of usability in ISO 9126-1 is closer to ISO 9241-11.

1.2.3 Usability versus Quality in Use

ISO 9241-11 explains how usability can be measured in terms of the degree of excellence in use: effectiveness (the extent to which the intended goals of use are achieved), efficiency (the resources that have to be expended to achieve the intended goals), and satisfaction (the extent to which the user finds the use of the product acceptable). ISO 9241-11 also emphasizes that usability is dependent on the context of use and that the level of usability achieved will depend on the specific circumstances in which a product is used. The context of use consists of the users, tasks, equipment (hardware, software and materials), and the physical and social environments which may influence the usability of a product in a work system. Measures of user performance and satisfaction thus assess the overall work system, and, when a product is the focus of concern, these measures provide information about the usability of that product in the particular context of use provided by the rest of the work system.

It is important to note that while this definition provides a practical way to measure usability, it is also measuring the consequences of other software quality characteristics such as the functionality, reliability and the efficiency of the computer system. Changes in these characteristics, or other components of the work system, such as the amount of user training, or improvement of the lighting, can also have an impact on user performance and satisfaction. For this reason, early drafts of ISO 9241-11 also defined a broader concept:

Quality of use: *the extent to which specified goals can be achieved with effectiveness, efficiency and satisfaction in a specified work system.*

However, this was removed from later drafts, as an unnecessary complication. The concept of the quality of a product in use did, however, provide the link between the ISO 9241-11 and ISO/IEC 9126 views of usability. Quality in use was incorporated as a high level quality objective into the revision to ISO/IEC 9126-1, and the related ISO/IEC 14598-1 standard (Software product evaluation - General guide):

> ***Quality in use:*** *the extent to which a product used by specified users meets their needs to achieve specified goals with effectiveness, productivity and satisfaction in a specified context of use.*

The revised ISO/IEC CD 9126-1 now distinguishes three broad approaches to improving the quality of a product (Figure 1) [62]:

- Set criteria for process quality: attributes of the software development processes, e.g. by application of ISO 9001, or ISO 15504 .
- Set criteria for product quality: attributes of the software (internal measures) or the behaviour of the software when tested (external quality).
- Set criteria for quality in use: the extent to which the code meets user needs for effectiveness, productivity and satisfaction in use.

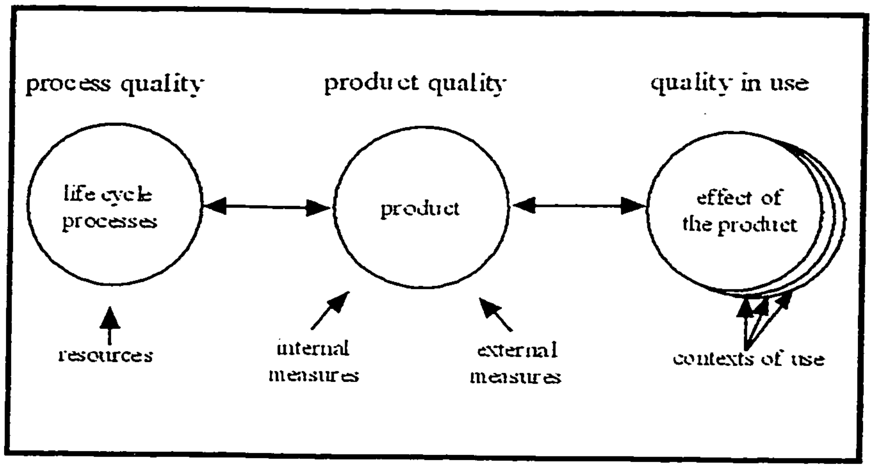


Figure 1: Approaches to software quality

1.3 Cost justifying usability

Not very long ago, most users of computers were programmers. Today, most computers are seen as tools that magnify a person's ability to perform all kinds of tasks that were formerly done without computers. So people are not trying to use computers, rather than they are trying to get their jobs done. Today, interactive facilities play an ever-increasing part in the application areas of computers. Software production is a business and like any business field has its own risks. Usability is one of the high risk items in software production.

1.3.1 Software Risk Management and Usability

Like many fields in their early stages, the software field has had its share of project disasters. The frequency of these software project disasters is a serious concern, a survey of 600 firms indicated that 35 percent of them had at least one runaway software project [47]. Most postmortems of these software projects have indicated that their problems would have been avoided or strongly reduced if there had been an explicit early concern with identifying and resolving their high-risk elements.

Webster's dictionary defines "risk" as the "possibility of loss". This definition can be translated into the fundamental concept of risk management: risk exposure, sometimes also called "risk impact" or "risk factor". Risk exposure is defined by the relationship:

$$RE = P(UO) \times L(UO)$$

Where RE is the risk exposure, P(UO) is the probability of an unsatisfactory outcome and L(UO) is the loss to the parties affected if the outcome is unsatisfactory. To relate this definition to software projects, we need a definition of "unsatisfactory outcome". Given that projects involve several classes of participants (customer, developer, user and maintainer), each with different but highly important satisfaction criteria, it is clear that unsatisfactory outcome is multidimensional:

- For customers and developers, budget overruns and schedule slips are unsatisfactory.
- For users, products with wrong functionality, user interface shortfalls, performance shortfalls, or reliability shortfalls are unsatisfactory.
- For maintainers, poor quality software is unsatisfactory.

Here we are not going into the details about risk management, and how a manager could determine the probabilities of losses, we would just like to mention two of software risk management steps:

- Risk identification: produces lists of the project-specific risk items likely to compromise a project's success. Typical risk identification techniques include checklists and decomposition. One of the top ten software risk items is developing the wrong user interface [47].
- Risk analysis: assesses the loss probability and loss magnitude for each identified risk item, and it assesses compound risks in risk-item iteration. Typical techniques here include quality factor analysis such as reliability and usability.

The above discussion just came to see how usability can affect the risks of a software. According to Boehm [47], risk management techniques to reduce the risk on user interface could be prototyping, scenarios, task analysis and user participation. Then if we can have a technique giving us objective decision criteria, it is obvious that, how it could reduce the risk of software on this item.

1.3.2 Lack of usability engineering

Tom Gilb [25] gives the definition of engineering as:

"Engineering is the use of principles to find designs that will meet multiple competing objectives, within limited resources and other constraints, under conditions of uncertainty."

The Institute of Electrical and Electronics Engineers (IEEE) defines software engineering , in IEEE Std 610.12, as:

"(1)The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, that is, the application of engineering to software. (2) The study of approaches as in (1)."

Engineering can be viewed as a closed feedback loop as shown in Figure 2. An engineering process consists of related activities performed in response to a statement of needs and consuming resources to produce a product. In order to manage or improve the process, one must exert control. Control is a decision-making mechanism that considers goals and constraints in the formulation of action that is intended to direct or modify the process. The decision to take action is based on measurements and quantitative evidence regarding the state of the process. Measurements can be made of conditions inside the process, products of the process, and the satisfaction of users of the products.

Building usability into a product requires an explicit engineering process. That engineering process is not logically different from any other engineering process. It involves empirical definition, specification of levels to be achieved, appropriate methods, early delivery of a functional system, and the willingness to change that system. Together these principles convert usability from a "last minute add on" to an integral part of product development.

We would expect usability standards, guidelines and maps to contribute to such a model of engineering, with respect to the development, operation and maintenance of interfaces.

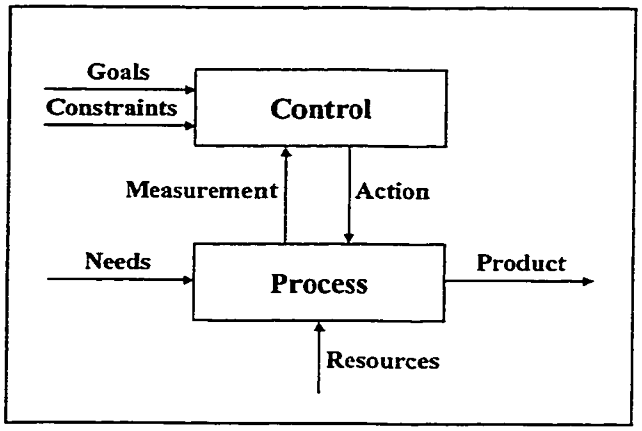


Figure 2: A model of engineering

1.4 Current usability assessment methods

Usability assessment techniques vary in their degree of formality, rigour and user involvement depending on the environment in which the evaluation is conducted. The choice is determined by financial and time constraints, the stage of the development lifecycle and the nature of the system under development. The INUSE project [64] has set up a network of European Usability Support Centres for the information engineering industry. These centres provide a portfolio of state-of-the-art usability services to support the development of systems that meet user's needs and have quality in use. They provided a framework for assessing the usability of a software product, consisting of five broad categories of evaluation methods. Those broad methods are usability context analysis, heuristic evaluation, user-based evaluation, design guidance and model-based assessment. Here we have a review of these methods.

Usability Context Analysis (UCA) [64]: is the prerequisite method for any assessment. It is a paper based questionnaire and a guide to ensure that the circumstances in which a prototype or product is assessed match the intended circumstances of eventual use.

The UCA Guide provides firstly a simple method for describing key features of:

- **Users**: for whom a system is designed ;
- **Tasks:** for which a system is designed to help users achieve ;
- **Environments:** in which a system is designed to operate - these are technical, physical and organizational.

Using a questionnaire format gives the evaluator a structured method for describing the users, tasks and environment in which the system is intended to be used. It then helps the evaluator to document which aspects of the context will be useful in the assessment, and enable a reviewer to judge, how accurately this matches the intended context of use.

Heuristic: (sometimes known as 'rule-based' evaluation) is usually carried out by human factors specialists, possibly supported by task experts. It can be fast and economical and is good for identifying major problems but might not be sufficient to guarantee a successful interactive system. Experts are not considered to be able to predict all the problems that end users will experience. The following list is a typical list of heuristics that the product can be assessed against:

- use simple and natural dialogue,
- speak the users language,
- minimize user memory load,

- be consistent,
- provide feedback,
- provide clearly marked exits,
- provide shortcuts,
- provide good error messages,
- prevent errors.

User-based evaluation: will generally include evaluation of the users' performance and attitudes. The assessment of usability should normally include analysis of both the users' performance and attitudes as these provide complementary information. User-based evaluation can be used to provide feedback at any stage of design. In the early stages, users may be involved in the evaluation of scenarios, simple paper mock-ups or partial/rapid prototypes. As design solutions become more developed, evaluations involving users will be based on progressively more complete and concrete versions of the system. Required data in this method are collected through the questionnaires and videos, which are recorded when the user is using the system or prototype.

Design guidance: evaluation against design criteria is an established technique which can contribute to usability. These criteria are contained in design guides, collections of ergonomic guidelines and standards. In the INUSE project, a guide is devoted specifically to Design the "Design Guidelines", focusing on what the product shall be used for, thus it is a higher level approach than many collections of style and interface guides. Guidelines specify attributes of a product which have been shown to improve usability. Some guidelines are at a surface level, e.g. screen layout of a menu, and others state higher level objectives, e.g. consistency. In many cases the usability of a product will be improved by redesigning the interface to be consistent with guidelines. But a much bigger improvement to the usability can often be made by considering whether the task can be carried out more effectively by a more fundamental redesign e.g. avoiding the use of menus to search for information by supplying a unique key which gives direct access.

Model-based assessment: takes place against a theoretical model of human abilities, and the specification of how the human-computer interface is going to operate. The nature of the models used, is usually quite generic and does not guarantee that the end users will react in the same way to the product. Tools for this kind of assessment are still in an early stage of development. They depend on a central description of the system, the tasks to be carried out with, and characteristics of users, which are together known as the model. A run-time system then executes the interface and accepts user input from the model database. Developers fine-tune the system by improving the interface features and looking at the resulting predicted user performances.

1.5 Summary

Today almost every developer uses design guidelines to create a user interface for the system and then they evaluate the results, find some problems, redesign and again the same story. This costs money. What if we have a tool that let us to assure some aspects of final product's quality from early stages of production. Obviously the existing models and solutions could not act properly in usability area. We will see the reasons in this research. Then we need a solution that gives us the confidence of having the quality in use in the final system, but not at a very high price. Also we need to fill the gap between software engineers and human –computer interaction experts. If we have a usability tool that we can integrate it into the software production process, the goal of HCI people will get clearer to software engineers and software developers. Through this research we will see how QUIM (Quality in Use Integrated Model) can help us to achieve those goals.

Chapter2:

Survey on Metrics, Quality Models, Standards, and Tools

In this chapter we are going to talk about software metrics and metrics plan. We will discuss the scope of software metrics and how to validate a metric. A metric could be a generic one or highly dependent on the domain knowledge. How to collect data to calculate a metric? What are the problems with metrics? What are the metrics impacts on usability? How may we measure objectively or subjectively?

Also we will have a look at different quality models that have been so far presented and have been used in the software industry world. What is the role of usability in these models? Finally, we will see the lessons learned from measurement programs and quality models.

2.1 Software metrics and measurement

"What is not measurable make measurable." Galileo Galilei (1564-1642)

A clear definition of measurement makes the use of metrics easier. Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules. Formally we define measurement as a mapping from the empirical world to the formal, relational world. Consequently a measure or metric is the number or symbol assigned to an entity by this mapping in order to characterize an attribute [40]. Then the real world is the domain of the mapping and the mathematical world is the range. When we map the attribute to a mathematical system, we have many choices for the mapping and the range. We may use real numbers, integers, or even a set of non-numeric symbols, such as what we ask software user in a subjective questionnaire.

The IEEE metrics standard defines a software quality metric as follows: " A software quality metric is a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality "[42].

Using metrics to have a good judgment on software quality is not a very recent topic. To justify the usage of metrics we would like to bring you a discussion from Cavano and McCall (1978)[30]: "*The determination of quality is a key factor in everyday events, wine tasting contests, sporting events, talent contests, etc. In these situations, quality is judged in the most fundamental and direct manner. Side by side comparison of objects under identical conditions and with predetermined concepts. The wine may be judged according to clarity, color, bouquet, taste, etc. However this type of judgment is very subjective. To have any value at all, it must be made by an expert. Subjectivity and specialization also apply to determining software quality. To help solve this problem, a more precise definition of software quality is needed as well as a way to drive quantitative measurements of software quality for objective analysis. Since there is no such thing as absolute knowledge, one should not expect to measure software quality exactly, every measurement is partially imperfect.*"

Metrics are used to provide visibility to engineers and managers, to assess adherence to standards, guidelines and principles, to estimate and predict, and to accept the product [27]. Metrics can provide visibility and insight to decision-makers. In large projects, the total amount of inherent information is often much more than a decision-maker can digest and understand. Therefore, to make an informed decision, a decision-maker needs summary information. Quantitative metrics can be used to summarize information needed to make distinctions, to isolate software characteristics, or to detect trends. Metrics can be used to assess adherence to a well-defined and documented standard, guideline or principle. Adherence metrics can be employed to measure deviations which in turn can be used in a management feedback loop to recover to an acceptable level. Adherence metrics provide a mechanism for assessing adherence to guidelines that are difficult or impossible to express and establish as a standard.

Metrics can be used as input into a model designed to estimate or predict software parameters, timing, effort or utilization of a resource. Finally metrics could be used as a standard for acceptance of a software product produced under a contractual agreement. Adapting a metric as an acceptance criterion implies that producer have a high degree of confidence based on past experiences in the relationship between the metric and desired properties of the system. Very few metrics are well understood. However, it is possible to integrate weaker metrics into a meaningful metrics policy.

There are several important characteristics that are associated with useful software metrics. Software metrics must be [46]:

- Simple to understand and precisely defined in order to facilitate consistency both in the calculation and the analysis of metric values.
- Objective as much as possible in order to decrease the influence of personal judgment to the calculation and analysis of metric values.

- Cost effective in order to have a positive return on investment (the value of the information obtained must exceed the cost of collecting the data, calculating the metric, and analyzing its value).
- Informative in order to ensure that changes to metric values have meaningful interpretation.

2.1.1 Metrics categories

Software metrics can be classified under different categories, although it is not unusual that the same metric belong to more than one category. A classification of metrics based on their intended use is as follows [46]:

- Process metrics: are those that can be used for improving the software development and maintenance process, e.g. efficiency of inspection.
- Product metrics: are those that can be used for improving the software product.
- Project metrics: or resources metrics are those that can be used for tracking and improving the project, e.g. programmer productivity.

Another classification of software metrics is based on the product itself [46]. In this way we divide them into two categories, i.e. external and internal.

- External software metric: is a quantitative scale and measurement method, which can be used for measuring an attribute or characteristic of the software product derived from the behavior of the system of which it is a part. External metrics are applicable to an executable software product during testing or operating in later stages of development and after entering to operation process.
- Internal software metric: is a quantitative scale or measurement method, which can be used for measuring an attribute or characteristic of a software product,

derived from the product itself, either direct or indirect (it is not derived from measuring a behavior of the system). Internal metrics are applicable to a non executable software product during designing and coding in early stages of development process.

There are different potential audiences of software metrics and their primary interests in using metrics are also different [46]:

- Software users are interested in the quality and value of the software products.
- Senior managers are interested in overall control and improvement across the projects in the business unit.
- Software engineers are interested in control and improvement of the specific software project activities and work products in which they are involved.
- Software process engineers and software quality assurance are interested in a cross section of what the three previous audiences are interested, depending on whether they are working at the business unit, or the project level.

A software metrics initiative must address the needs of all those potential metric audiences and users by [46]:

- Defining metrics and obtaining consensus/acceptance by the user community.
- Training metrics users and providing consulting support for implementation.
- Automating the data collection, analysis, and feedback process.

Such an initiative must also account for the different levels of measurement, examples of such levels include, but are not limited, to the following [46]:

- The company or business unit level, at which data across several projects may be lumped together to provide a view of attributes such as productivity, and quality.

- The project level, at which data within the project is tracked and analyzed both in-process and post mortem in order to plan and control the project, as well as to improve similar projects across the business unit.
- The component level, at which data within a component, e.g. subsystem, of a project is tracked and analyzed for managing the development of that component, as well as improving its quality over time.

2.1.2 Metrics plan

It is important to remember that the primary responsibility of any developer is to produce a product, not to collect measurement data. Thus, it is essential that the job of metrics data collection and analysis be made as simple and unobtrusive as possible by using tools, models and standards. Models help us understand how the metrics relate to one another. Standards provide counting and collection rules to ensure consistency and completeness in our data. And by automating as much as possible, tools help to speed up and standardize both collection and analysis. In particular, we can embed standards in our tools, allowing our counts to be as objective and consistent as possible.

A metrics plan or according to some references a metrics program, is much like a newspaper article. It must describe the who, what, where, when and why of metrics. With answers to all of these questions, whoever reads the plan knows exactly why metrics are being collected, how they are used, and how metrics fit into the large picture of software development. The plan usually begins with the why. The plan lays out the goals and objectives of the project, describing what questions need to be answered by project members. For example, if usability is a major concern to the developers, then the plan discusses how usability will be defined and what reporting requirements are imposed on the project, later sections of the plan can then discuss how usability will be measured and tracked.

Next, the plan addresses what will be measured. In many cases, the measures are grouped or related in some way. For example, productivity may be measured in terms of two component pieces, size and effort. So the plan will explain how size and effort are defined and how they are combined to compute productivity.

At the same time, the plan must lay out where and when during the process the measurements will be made. Some measurements are taken once, while others are made repeatedly and tracked over time. The time and frequency of collection are related to the goals and needs of the project, and those relationship should be made explicitly in the plan.

How and who address the identification of tools, techniques, and staff available for metrics collection and analysis. It is important that the plan discusses not only what measures are needed but also what tools will be used for data capture and storage, and who is responsible for those activities.

The plan must state clearly what types of analysis will be carried out with the data, who will do the analysis, how the results will be conveyed to the decision makers, and how they will support decisions. Thus the metrics plan or metrics program paints a comprehensive picture of the measurement process, from initial definition of need to analysis and application of the results.

Software metrics plan is a term that embraces many activities, all of which involve some degree of software measurement [40]:

- Cost and effort estimation
- Productivity measures and models
- Data collection
- Quality models and measures
- Reliability models
- Performance evaluation and models
- Structural and complexity metrics
- Capability-maturity assessment
- Management by metrics
- Evaluation of methods and tools

The author of the above list and some other experts such as Littlewood [40] believe that even though most quality models include reliability as a component factor, but the need to predict and measure reliability itself express the need of a separate specialization in reliability modeling and prediction. Reliability is a high level, external product attribute that appears in all quality models. The accepted view of reliability is the likelihood of successful operation during a given period of time. Many users view reliability as the single most important quality attribute to consider. So Littlewood and others provide a rigorous and successful example of how a focus on an important product quality attribute has led to increased understanding and control of our products. Now one may ask this question: "is the importance of the usability of a software product less than its reliability on the user opinion?"

2.1.3 Metrics Validation

Validation is the process of building objective evidence that a metric is effective in meeting its stated purpose. For example validation could involve determining that a visibility metric is useful to decision makers or that an adherence metric accurately reflect adherence to the stated standard, guideline or principle [27].

Validating a software measure is the process of ensuring that the measure is a proper numerical characterization of the claimed attribute by showing that the representation condition is satisfied [40].

IEEE Std 1061 defines the validation of a software quality metric, like this: *"The act or process of ensuring that a metric reliably predicts or assesses a quality factor"*. So based on this standard a validated metric is one whose values have been statistically associated with corresponding quality factor values. It is necessary to remind that a quality factor may be affected by multiple variables. Then a single metric may not sufficiently represent any one quality factor if it ignores these other variables. The standard considers six validity criteria to assess whether a metric is valid. Those criterions are as follows:

1. Correlation: The variation in the quality factor values explained by the variation in the metric values. This criterion assesses whether there is a sufficiently strong linear association between a quality factor and a metric to warrant using the metrics as a substitute for the quality factor, when it is infeasible to use the latter.
2. Tracking: If a metric M is directly related to a quality factor F, for a given product or process, then a change in a quality factor value from FT1 to FT2, at times T1 and T2, shall be accompanied by a change in metric value from MT1 to

MT2. This change shall be in the same direction. This criterion assesses whether a metric is capable of tracking changes in product or process quality over the life cycle.

3. Consistency: If quality factor values F1, F2 corresponding to products or processes 1, 2, have the relationship F1>F2, then the corresponding metric values shall have the relationship M1>M2. This criterion assesses whether there is consistency between the ranks of the quality factor values of a set of software components and the ranks of the metrics values for the same set of software components. This criterion shall be used to determine whether a metric could accurately rank, by quality, a set of products or processes.
4. Predictability: This criterion assesses whether a metric is capable of predicting a quality factor value with the required accuracy.
5. Discriminative power: A metric shall be able to discriminate between high-quality software components and low-quality software components. The set of metric values associated with the former should be significantly higher (or lower) than those associated with the latter. This criterion assesses whether a metric is capable of separating a set of high-quality software components from a set of low-quality components. This capability identifies critical values for metrics that shall be used to identify software components that have unacceptable quality.
6. Reliability: A metric shall demonstrate the correlation, tracking, consistency, predictability, and discriminative power properties for at least P% of the application of the metric. This criterion is used to ensure that a metric has passed

a validity test over a sufficient number or percentage of applications so that there shall be confidence that the metric can perform its intended function consistently. For information on the above methodology of validating, calculation methods and statistical techniques used, one may refer to Schneidewind[31], Conover[32], Gibbons[33], Kleinbaum and Kupper[34].

2.1.4 Domain Dependency of Metrics

To show how usability of a software and quality in use metrics could be related tightly to the domain, we adopt the following example.

The idea for hypertext (where documents are linked to related documents) is credited to Vanner Bush's famous MEMEX idea from 1945 [35]. Ted Nelson coined the term "hypertext" in 1965[37]. Mosaic, the first popular hypertext browser for the World Wide Web was developed at the University of Illinois' National Center for Supercomputer Applications (NCSA). Having the foundation from the above concepts, today the structure of the web is rapidly evolving from a loose collection of web sites into organized marketplaces. We are talking about the term "e-business", good business practice dictates the use of effectiveness measurements to guide the design of all web site features. An interesting question is what metrics are the best for evaluating the effectiveness of web site features? A good example of measuring effectiveness comes from the online ad banner industry. E-marketers rely on ad banners to direct visitors to their web sites, and the ad banner companies set their prices based on click-through and look-to-buy metrics, both of which measure effectiveness. Click-trough metric, measures the ratio of clicks to impressions, where an impression is simply the display of an ad banner on a web page. A high click-through rate means visitors who see the ad click on it frequently, therefore the ad is bringing many visitors to the site. Look-to-buy data compares ad banner impressions with sales transactions and revenue directly attributable to the ad banner. Look-to-buy rates are shown on a graph such as scattering graph. The X-axis of the diagram represents product impressions, and the Y-axis represents purchases. Every product on the chart is shown by a colored rectangle, its width represents the profit margin of the product and its height represents the product's retail price. Products in the lower-right corner of the graph are over-promoted, since there are many lookers and relatively few buyers. Products in the upper-left corner may be under-promoted, since the few shoppers who see these products tend to buy them. The required data to create such a graph are gathered from the web, when the visitors just click [38].

Using this kind of metrics to glean the overall effectiveness of the site can be viewed from the perspective of the web site owner and the perspective of the site visitor. From the business perspective, metrics may suggest where improvement may be made with regard to the design, layout, and navigation issues. Metrics can also be used to create visualizations that demonstrate visitor's behaviors such as user profile. For example, the number of times a product appears on the site compared to the number of times people actually bought a product.

Finally, in a word, using the mentioned metrics to improve the features of web site, results in better usability of the site interface, and in turn it results in higher profit for the e-commerce site.

2.1.5 Objective and Subjective Measures

When measuring attributes of entities, we strive to keep our measurements, objective. By doing so, we make sure that different people produce the same measures, regardless of whether they are measuring a product, process or resource. This consistency of measurements is very important, because it gives us the ability of comparison. Although no measurements is truly objective (because there is always some degree of subjectivity about the entities and attributes) some measures are clearly more subjective than others. Subjective measures depend on the environment in which they are made. The measures can vary with the person measuring, and they reflect the judgment of the measurer. What one judge considers bad, another may consider good, and it may be difficult to reach consensus on attributes such as process, product or resource quality [40].

Nevertheless, it is important to recognize that subjective measurements could be useful, as long as we understand their imprecision. For example, suppose we want to measure the quality of the interface requirements, before we turn the specification over to the test team, who will then define test plans from them. We may ask the test team to read and scale each requirement on a scale from 1 to 5, where 1 means " I understand this requirements completely and can write a complete test script to determine if this requirements is met" and 5 means " I do not understand this requirements and can not write a test script for it." Suppose the result of the assessment is like the table 1:

Requirement Rating	1	2	3	4	5
Number of Interface Requirements	3	4	6	7	1

Table 1: Example of subjective measurements

In the example we see 6 interface requirements have been assigned rating 3 by the test team. And only 3 of the whole interface requirements are clear. Even though the measurement is subjective, the measures show us that we have problems with our interface requirements, perhaps the interface requirements should be reviewed and rewritten before proceeding to test plan generation or even to design.

In a process, a product or a resource, we may distinguish between two categories of measurable attributes, internal and external. The definitions of Internal and External attributes have been presented in the previous chapter.

Examples for internal attributes of a software could be the number of modules, size of a module, and these are the attributes which usually developers are interested in. But a project manager is interested in cost of development which is an external attribute. The end user is not interested in knowing how many modules exist in the software, but rather he/she is interested in the number of nested levels of a menu or in the other words depth of menu. The user doesn't pay attention to the complexity of the code but the complexity of the user interface is very important for him or her.

It is sometimes difficult to define the attributes in measurable ways with which every one agrees. For example, we all want to build and purchase systems of high quality. But we do not always agree on what we mean by quality, and it is often difficult to measure quality in a comprehensive way. Thus we tend to define these high level attributes in terms of other, more concrete attributes that are well defined and measurable. The McCall model and Boehm Model are good examples of this phenomenon (the models will come in the next sections), where software quality is defined as a composite of a large number of narrower, more easily measurable terms. And this is exactly what we need to do for usability attributes. That is one reason why we need to have such a model, so the model, enables us to predict the usability, to assess it and to improve it on a software product.

One thing more that we have to consider here is that there is a clear need for internal attribute measurements to support measurement and decision making about external attributes. One of the goals of software measurement research is to identify the relationships among internal and external attributes, as well as to find new and useful methods for measuring directly the attributes of interest [40].

2.2 Usability Metrics

Usually, to evaluate the usability of an user interface, we expect the user to interact with the system over time based on an operational profile. In this way we may assess the usability by the external attributes of the system. But this requires extensive and careful data collection. We seek a simpler approach. One technique is to look for internal characteristics that we think lead to good usability [40], such as the complexity of code which can affect the task execution time. Unfortunately it is not easy and convenient to define the explicit relationship between internal attributes and the external notion of usability. Task execution time is affected by the arrangement of items on the display, too. In the literature, we find that proposed measures of usability encompass both measures of internal attributes (believed to affect particular views of usability) and measures of particular views of (the external attributes) usability.

2.2.1 External View of Usability Measurement

Here we may see some examples or previous works on decomposing usability into more fundamental attributes so that we can measure them. Following Gilb's approach , we can identify some measurable attributes of usability, each of which must be assessed by the user on a particular type of product [40]:

- Entry level: This attribute could be measured in terms of experience with similar classes of applications (especially pertinent for popular applications such as word-processing systems or spreadsheets), or simply age (in the case of, for example, a junior school teaching program).
- Learnability: This attribute could be measured in terms of speed of learning, one measure might be hours of training required before independent use is possible.
- Handling ability: This attribute could be measured in terms of speed of working when trained, or errors made when working at normal speed.

Another approach is COQUAMO (COnstructive QUAlity MOdel) which provides automated support for setting targets for usability and measuring usability. This method is invented by Kitchenham and Pickard in 1987. It provides a set of templates to be filled in. The approach is concerned with the effort required for learning and operating the system. Although this approach does not produce definitive, objective measures, clear measurable objectives can be set, and we can decide whether the objectives have been met.

The MUSiC (Metrics for Usability Standards in Computing) project was specifically concerned with defining measures of software usability [41]. The project proposed a number of user performance measures, such as Task Effectiveness, Temporal Efficiency and Productive Period. But the performance view of usability can not capture any notion of user satisfaction, convenience or ease of use. The most direct way to measure user satisfaction is to survey actual users. The MUSiC project developed a standard 50 items questionnaire to assess user satisfaction. The completion of the questioner takes 10 minutes. It is called Software Usability Measurement Inventory (SUMI) and it provides an overall assessment and usability profile that breaks the overall assessment down into five attributes: Effectiveness, Efficiency, Helpfulness, Control and Learnability.

2.2.2 Web usability metrics

Unless a web site meets the needs of the intended users, it will not meet the needs of the organization providing the web site. Web site development should be user-centered, evaluating the evolving design against user requirements. The simplest usability metric is the success rate when performing a representative task: record the percentage of test users capable of accomplishing what they were asked to do. Many users will give up after a certain amount of struggle, though sometimes it is necessary to specify a maximum duration of the study and count users as having failed if they didn't achieve the goal within that time.

Success rates provide a best-case estimate of the true success rate for customers who are not part of a usability study. The very fact that users are part of a test means that they will continue longer and try harder than normally. Test users often don't give up until they have tried everything they can think of. When people are not part of a test, they will abandon a site once it becomes clear to them that using the site is too difficult.

One benefit of the success rate measure is that it can be collected while running a traditional thinking-aloud study where you ask the users to verbalize their thoughts and tell you what they are thinking at each step of the way. With this method, you can continue to collect qualitative insights while you collect a formal usability metric.

Unfortunately, most other usability metrics only work if you stop asking the users questions and let them concentrate on using the site without interruptions. This is particularly true for any time measures: you want to know how fast people can perform tasks when they simply do them and not when they spend half their time chatting with you in order to explain and talk about every small thing they are doing.

Because of this variation, it is often necessary to run two sets of usability studies, one to collect the numbers and another to get insight into the users' behavior and thinking. The first study gives you the hard-hitting numbers; the second study gives you the information you need to redesign.

In addition to success rates, the two main usability measures are [61]:

- **Task performance**: usually measured by the amount of time it takes an average user to perform an average task with the site.
- **Subjective satisfaction**: usually measured by the user's answers to a questionnaire with statements like "I felt completely in control of everything that happened while I was using this Web site."

It is also possible to measure the quality of the outcome of the user's task. For example, if the task is to book the cheapest airline ticket to London, you can measure:

- whether the user ever ends up with a ticket to London (success rate)
- how long it takes the user to book the ticket (task performance)
- how much the user enjoyed the experience
- whether, in fact, the ticket was the cheapest or if not, how it compared to other ticket prices.

Most of the usability measures can be collected for both novice users and experienced users. You almost always find much better scores for experienced users. For example, a travel agent would typically be much faster at booking the ticket to London than somebody who was using a travel site for the first time.

On the Web, it is important to get good usability measures for novice users since people don't hang out at any individual Web site long enough to become very experienced in its use. By contrast, Intranet applications and certain extranet sites may have a large amount of repetitive use and thus require special attention to measuring expert user performance [61].

Jakob Nielson believes that to improve a design, insight is better than numbers[21]. He says usability metrics let you track progress between releases and applying metrics can cost four times as much as conducting qualitative studies. But we believe this is not the only case, before measuring the usability of a web site involving real user which is an expensive method, we need something that can help us in predicting its usability during design process. Everybody agrees that the design of a web site should take into account the established guidelines for web writing style, navigation, site structure and page design. The problem with guidelines is that, they provide subjective information to the developer, for example they say "minimize the need for scrolling" [63]. An experienced web site designer can imagine it, but if we provide objectives, i.e. measurable attributes, so every developer can use it. Let's say the length of a page and the number of alphabetical characters on a page both can have target values in specification of user interface, to achieve that goal. We have identified some metrics which are useful to acquire the quality in use in the context of hyper media. The list of those metrics is presented in the appendix A.2.

2.2.3 Potential problems with usability metrics

Before talking about problems with measurement and metrics, first let us clarify the different categories for expressing a value. To express values and animate them, we use different scales. For a complete classification and properties of measurement scales, one may refer to reference number [40].

- A Ratio scale has the following characteristics:

1. It is a measurement mapping that preserves ordering, the size of intervals between entities, and ratios between entities.
2. There is a zero element, representing total lack of the attributes.
3. The measurement mapping must start at zero and increase at equal intervals, known as units.
4. All arithmetic operations can be meaningfully applied to the classes in the range of the mapping.

Examples of this scale could be: Time interval, Length.

- A Nominal scale has two major characteristics:

1. The empirical relation system consists only of different classes, there is no notion of ordering among classes.
2. Any distinct numbering or symbolic representation of the classes is an acceptable measure, but there is no notion of magnitude associated with the number of symbols.

Example: we may classify software fault types into, data faults, controls faults , ...

- An Ordinal scale is defined using the following characteristics:

1. The empirical relation system consists of classes that are ordered with respect to the attribute.
2. Any mapping that preserve the ordering (That is, any monotonic function) is acceptable.
3. The numbers represent ranking only, so addition, subtraction, and other arithmetic operations have no meaning.

Example: ordering software failure by severity, i.e. negligible, marginal, critical, catastrophic.

Measuring usability is suffering from a case of "physics envy", because usability is so difficult to measure. A major problem is that a measurement result of, say, 12% decrease in task time might mean 12% better productivity, but could just as well mean 12% more time wasted on other activities or 12% more stress for the users. So we do not necessarily have a true ratio scale for usability measurements.

We also have difficulties with the simpler types of measurement scale. It may actually be hard to achieve a nominal scale for usability problems, even though nominal scales are the weakest form of measurement scale and only require us to be capable of deciding when two usability problems are the same. Unfortunately, it is hard to know to what extent two observations are really cases of the same underlying usability problem, and we do not even have a good enumeration of usability problems with which to match observations.

Using the somewhat more ambitious ordinal scales to measure usability problems, we face the counting problem. We would like to know whether one interface is better than another (the ordering of the ordinal scale) based on counting the frequencies of various usability problems in the two systems. Assume, for example, that we have measured the frequencies of usability problems A, B, and C as it comes in table 2:

	Problem A	Problem B	Problem C
System 1	1	1	8
System 2	2	1	1

Table 2: Example of metrics problems

It may initially seem as if System 2 is much better than System 1, but that is only true if Problem C is actually a key problem. It could be the case that Problem A is a usability catastrophe that keeps users from ever getting to use the system, whereas Problem C is a triviality that slows down users a second or so. Then it would obviously be preferable to use System 1[19s].

2.3 Data

Having the right measures is only part of a measurement program. We can not make good decisions with wrong data. We said measures or metrics should be well defined and valid, i.e. they should reflect the attributes we claim they do. But even when we have a well defined metric that maps a real world attribute to a formal relational system in appropriate ways, still we need to ask several questions about the data [40].

1. Correctness: the data were collected according to the exact rules of definition of the metric. A measure of task duration is correct if time is measured from a specified activity and ends at the completion of another specified activity.
2. Accuracy: refers to the difference between the data and the actual value. Time measured using an analog clock may be less accurate than time measured using a digital clock.
3. Precision: deals with the number of decimal places needed to express the data. Task duration need not be reported in tenths of second.
4. Replication: data are often collected to support surveys, case studies and experiments. These investigations are frequently repeated under different circumstances, and the results are compared. Then, project histories and study results are stored in a database, so the baseline measures can be established and organizational goals can be set.

5. Consistency: data should be consistent from one measuring device or person to another. Thus two evaluators should calculate the same value for the same measure at the same circumstances. Finally it means that data should be captured in the same way, so for different circumstances the results of measures are comparable.

Thus it is very important to assess the quality of data and data collection before data collection begins. The measurement program must specify not only what metrics to use, but also what precision is required, and what rules govern the data collection. For example, whether a particular tool or questionnaire will be used to capture the data.

There are really two kinds of data with which we are concerned :

- Raw data (countable): results from the initial measurement of process, product or resources.
- Refined data (calculable): extracted essential data elements from raw data.

Deciding what to measure is the first step. We must specify what direct measures are needed, and also what indirect measures may be derived from the direct ones. Sometimes we begin with the indirect measures. From a GQM (Goal, Question, Metric) analysis (the explanation will come in the next sections), we understand which indirect measures we want to know, and from those, we must determine what direct measures are required to calculate them. Figure 3 shows the role of data collection in the software measurement program.

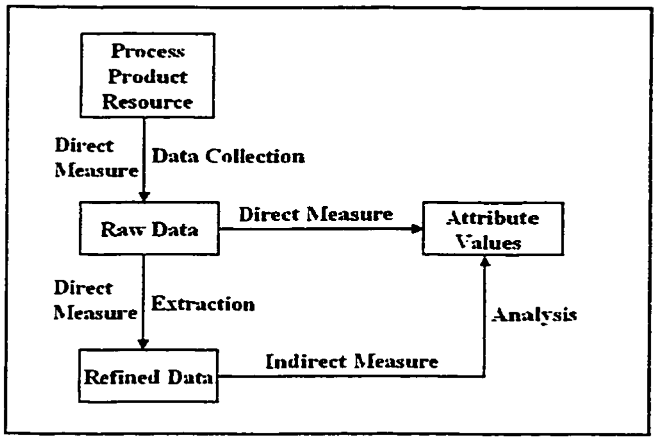


Figure 3: The role of data collection in software measurement

Most organizations are different, not only in terms of their business goals but also in terms of their corporate culture, development preferences, and staff skills. The differences also are happening between different software products. So a GQM analysis of apparently similar projects and products may result in different metrics at different companies, and also it could lead to different metrics at the same company for different products. That is exactly the reason that, GQM is preferable to a "one-size-fits-all" standard measurement set [40]. But still most organization and most software products share some problems.

Since the production of software is an intellectual activity, the collection of data requires human observation and reporting. Managers, system analysts, programmers, testers and users must record raw data on forms. This manual recording is subject to error, omission and delay, deliberately or unconsciously. Therefore automatic data capture is desirable, and sometimes essential, such as recording the number of nodes to be passed to do a specific task in a web site, or the completion time of a specific task. Unfortunately, in many instances, there is no alternative to manual data collection. To ensure that the data are accurate and complete, we must plan our collection effort before we begin to measure and capture data. Ideally we should:

- Keep procedures simple
- Avoid unnecessary recording
- Train the responsible staff to record data and procedures
- Validate all data collected

The last point is especially important, otherwise we will go to the wrong direction. Basili and Weiss, examined data from NASA's Goddard Space Flight Center in Maryland [44]. They found half of the data collected were corrupted in some way and therefore not useful for analysis.

2.4 Usability in Software quality assurance models and Standards

In this section, we look at software quality models that have gained acceptance within the software engineering community, and some of them address the usability explicitly.

1. **McCall model**: This model uses a decomposing approach to describe software quality. Figure 4 shows the structure of McCall model.

   This model includes 41 metrics to measure the 25 quality criteria generated from the quality factors. We review an example to see how the model could be used. Measuring any factor requires us first to consider a checklist of conditions that may apply to the requirements (R), the design (D), and the implementation (I). The condition is designated "yes" or "no", depending on whether or not it is met. To see how the metrics and checklists are used, consider measuring the criterion "Completeness" for the factor "Correctness".

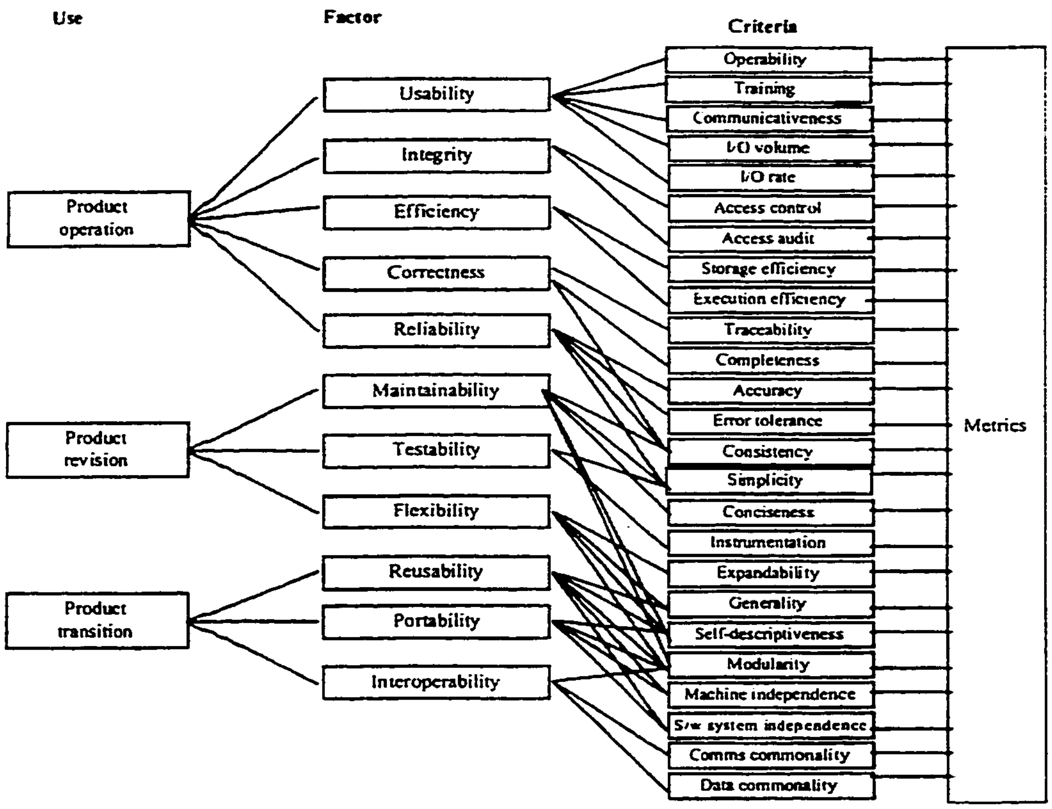


Figure 4: McCall software quality model

The checklist for completeness is:

- Unambiguous references (input, function, output) for R, D, and I
- All data references defined, computed or obtained from external source for R, D, and I
- All defined functions used for R, D, and I
- All referenced functions defined for R, D, and I
- All conditions and processing defined for each decision point for R, D, and I
- All defined and referenced calling sequence parameters agree for D, and I
- All problems reports resolved for R, D, and I

- Design agrees with requirements for D
- Code agrees with design for I

Notice that there are six conditions that apply to requirements, eight to design and eight to implementation. We can assign a 1 to a "yes" answer and 0 to a "no" answer, and we can compute the completeness metric in the following way to yield a measure that is a number between 0 and 1:

$$(1/3)\times((\text{Number of yes for } R/6)+(\text{Number of yes for } D/8)+(\text{Number of yes for } I/8))$$

since the model tells us the "Correctness" depends on "Completeness", "Traceability" and "Consistency", we can calculate analogous measures for the latter two. Then, the measure of "Correctness" is the mean of their measures:

$$Correctness = (x + y + z)/3$$

Where x, y and z are the metrics for "Completeness", "Traceability" and "Consistency", respectively. In this example, all of the factors are weighted the same. However it is possible to use different weightings, so that the weights reflect importance, cost or some other considerations which are important to the evaluator.

In this model Usability has come as a factor and it is decomposed into criteria Operability, Training, Communicativeness, I/O volume and I/O rate. It defines usability as the effort required to learn, operate, prepare input, and interpret the output of a program.

Unfortunately many of the metrics defined by McCall, can only be measured subjectively [52]. The metrics may be in the form of a checklist that is used to grade specific attributes of the software. The grading scheme proposed by McCall is a 0 (low) to 10 (high) scale. The definition of some criteria for factor usability which are used in the model is as follows:

Operability: the ease of operation of a program.

Training: the degree to which the software assists in enabling new users to apply the system.

2. **Boehm model:** This is one of the first quality models for software defined by B.W.Boehm in 1978. He proposed a multilevel hierarchy or a tree of software criteria. He used a different terminology for levels of tree, i.e. Primary uses, Intermediate constructs, Primitive constructs and finally Metrics (Figure 5). He suggested that a software has General Utility if it is Portable, Maintainable and, and in turn at the next level he decomposed those attributes into the others:

   Maintainability: Testability, Understandability, Modifiability.

   As is Utility: Reliability, Efficiency, Human Engineering.

   Boehm has not anything about usability in his model, just he talks about Human Engineering as an intermediate constructs and decomposes it into Communicativeness and Accessibility.

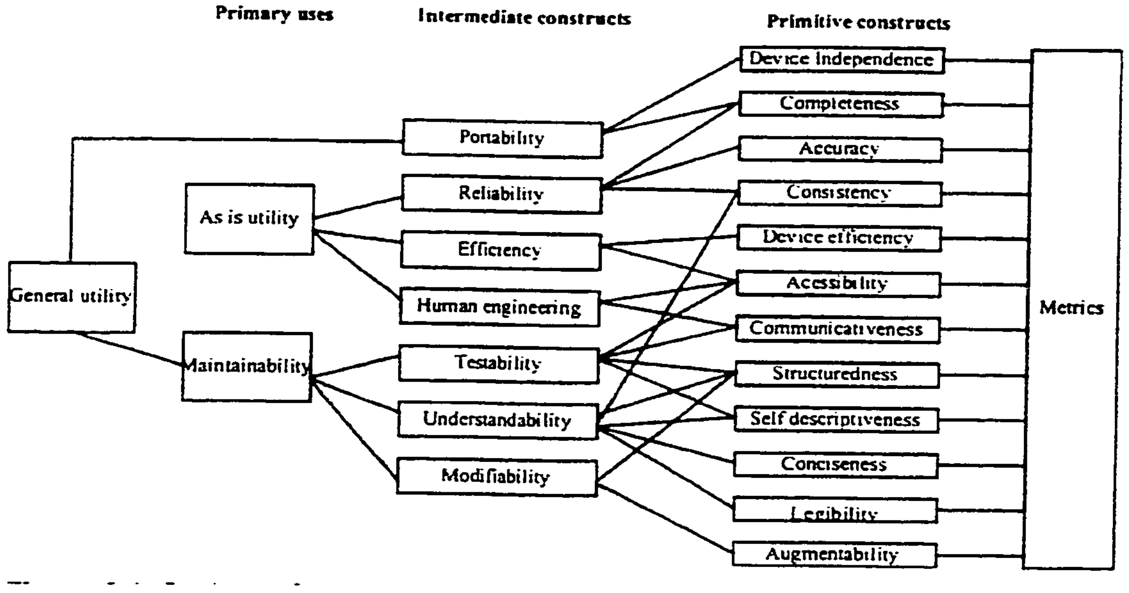


Figure 5: Boehm software quality model

3. **GQM model:** It stands for Goal-Question-Metric. This is a goal oriented approach. The fundamental idea is a simple one; managers proceed according to the following three stages:

   1) Set goals specific to needs in terms of purpose, perspective and environment.

   2) Refine the goals into quantifiable questions that are tractable.

   3) Deduce the metrics and data to be collected (and the means for collecting them) to answer the questions.

GQM is more a philosophy than a model. It is a top-down approach. The goal/question/metric paradigm is intended as a mechanism for formalizing the characterization, planning, construction, analysis, learning and feedback tasks. It represents a systematic approach for setting project goals, tailored to the specific needs of an organization, and defining them in an operational and tractable way.

Goals are refined into a set of quantifiable questions that specify metrics. This paradigm also supports the analysis and integration of metrics in the context of the questions and the original goal. Feedback and learning are then performed in the context of the GQM paradigm.

The process of setting goals and refining them into quantifiable questions is complex and requires experience. In order to support this process, a set of templates for setting goals, and a set of guidelines for deriving questions and metrics has been developed. These templates and guidelines reflect the experiences of Basili and his colleagues [48] from having applied the GQM in a variety of environments such as NASA, IBM and AT&T. Of course they do not claim that these templates and guidelines are complete. Different sets of guidelines exist for defining product-related and process-related questions. Product-related questions are formulated for the purpose of defining the product such as physical attributes, context and defining the quality perspective of interest such as reliability and user friendliness.

- Guidelines for product-related questions: for each product under study there are three major sub goals that need to be addressed: 1) definition of the product, 2) definition of the quality perspectives of interest, and 3) feedback related to the quality perspectives of interest. Definition of the product includes questions related to:

    1. Physical attributes, i.e. a quantitative characterization of the product in terms of physical attributes such as size, complexity, etc.
    2. Cost, i.e. a quantitative characterization of the resources related to this product in terms of effort, time, etc.

3. Changes, i.e. a quantitative characterization of the adaptations, enhancements related to this product.

4. Context, i.e. a quantitative characterization of the customer community using this product and their operational profiles.

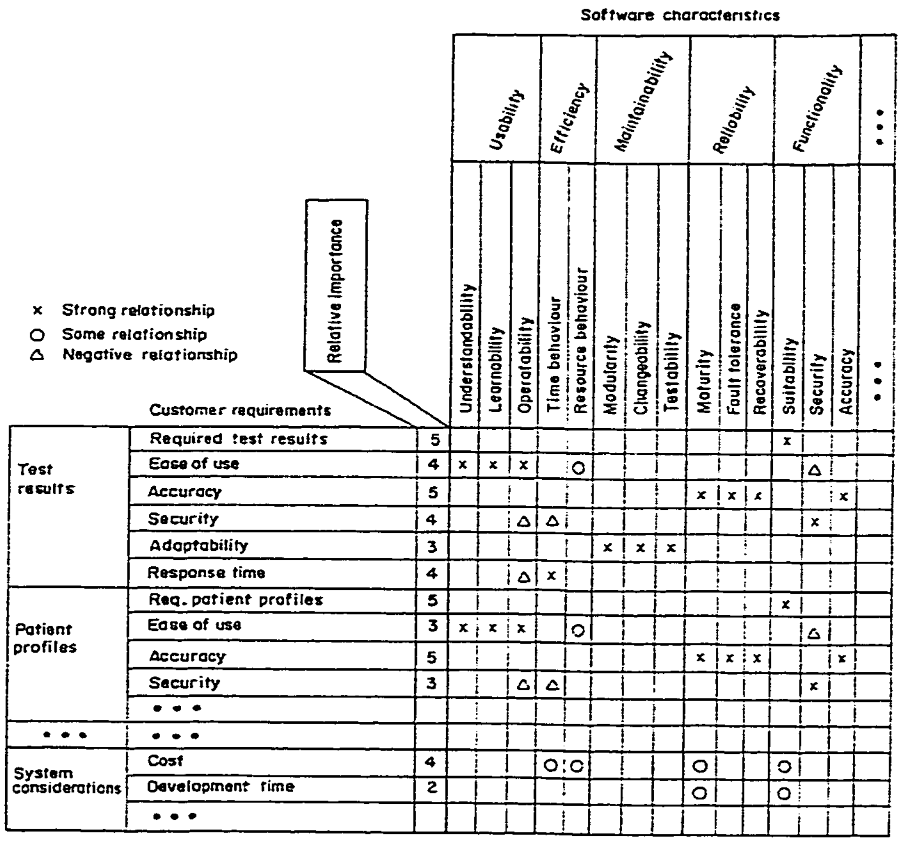
- Guidelines for metrics, data collection, and interpretation: the choice of metrics is determined by quantifiable questions. The guidelines for questions acknowledge the need for generally more than one metric, for objective and subjective metrics, and for associating interpretations with metrics. The actual GQM models generated from these templates and guidelines will differ from project to project and organization to organization. This reflects their being tailored for the different needs in different projects and organizations. Depending on the type of each metric, we choose the appropriate mechanism for data collection and validation.

Many metrics programs begin by measuring what is convenient or easy to measure, rather than by measuring what is needed. Such programs often fails because the resulting data are not useful to the developer, designer or maintainer of the software. There are lots of metrics that are easy to measure in usability field, such as the number of individual items on the screen. Reduction in the number of items on the screen reduces memory load of the user, but where to put what the user needs, or in the other word, how to make things accessible to user? Then for our goal, i.e. reducing user memory load, developer may use metrics such as Task visibility and Layout uniformity, that are of course more difficult to determine.

4. **Quality Function Deployment:** QFD is a technique that evolved from total quality management principles, it aims at deriving indicators from the user's point of view. QFD consists of activities supported by its defined matrices and tables: House of Quality, Part Deployment, Process Planning and Production Planning. The approach helps to make the product developer sure, about which of the quality characteristics or technical parts of the product should be increased, starting with analyzing customer requirements. The method was introduced for industrial and consumer products in Japan in 1972, this approach has been successful in different areas such as telecommunication, and has been adopted to give service in software industry. Here the basic idea is to translate customer requirements into the appropriate technical requirements for each stage of product development and production. The customer's requirements are expressed in their own terms and the technical counterparts are expressed in measurable terms. These technical counterparts to customer quality requirements represent final product control characteristics. Consequently they should directly affect customer perceptions. Because they are expressed in measurable terms the relevant units of measurement must be determined as well as target values. The measures are used for assuring that the required quality is achieved. QFD can be applied for planning, production and control, that is, throughout the whole process. The hierarchical structure of the matrices helps in the stepwise process to reach detailed and precise information starting from rather vague and abstract user requirements.

Figure 6 shows the first matrix concluded form analyzing the user requirements for an information management system in medical organization [67]. The matrix represents mapping customer requirements to the software characteristics. The type of relationship at each cross point is determined by an expert, resulting from his experiences, previous empirical jobs and meeting with the customer. At the next step, software characteristics on figure 6, are mapped to the terms that are commonly used by software engineers. These terms are called production features. Again the new matrix determines the relevancy of every cross point. Finally, at the third step, those product features are mapped to the metrics. Then through three steps, by creating three matrices, we have followed system requirements to the measurable metrics. As an example, on figure 6, response time is mapped to time behavior with strong relationship, in the second matrix time behavior will be mapped to size of source code with a strong relationship. And finally, in a third matrix, size of source code is determined by LOC (Lines Of Code).

There is a well known problem with this method. It requires a lot of experimental results to establish reliable rules, and also it needs an experienced developer.



Software characteristics

× Strong relationship
○ Some relationship
△ Negative relationship

	Customer requirements	Relative importance	Usability			Efficiency		Maintainability			Reliability			Functionality			• • •
			Understandability	Learnability	Operatability	Time behaviour	Resource behaviour	Modularity	Changeability	Testability	Maturity	Fault tolerance	Recoverability	Suitability	Security	Accuracy	• • •
Test results	Required test results	5												x			
	Ease of use	4	x	x	x		○								△		
	Accuracy	5									x	x	x			x	
	Security	4			△	△									x		
	Adaptability	3						x	x	x							
	Response time	4			△	x											
Patient profiles	Req. patient profiles	5												x			
	Ease of use	3	x	x	x		○								△		
	Accuracy	5									x	x	x			x	
	Security	3			△	△									x		
	• • •																
• • •	• • •																
System considerations	Cost	4				○	○				○			○			
	Development time	2									○			○			
	• • •																

Figure 6: The example of QFD matrix

5. **IEEE Software Quality Metrics Methodology [42]:** This standard provides a methodology for establishing quality requirements, and identifying, implementing, analyzing and validating process and product software quality metrics. This methodology applies to all software at all phases of any software life cycle. This standard does not prescribe any specific metric. The model suggests a hierarchy including different levels for quality factors, quality sub factors and metrics as well.

   The standard expresses that the use of software metrics reduces subjectivity in the assessment and control of software quality by providing a quantitative basis for making decisions about software quality. However, the use of software metrics does not eliminate the need for human judgment in software assessment. The use of software metrics within an organization or project is expected to have a beneficial effect by making software quality more visible. More specifically, the use of this standard's methodology for measuring quality enables an organization to:

- Assess achievement of quality goals
- Establish quality requirements for a system as its outset
- Establish acceptance criteria and standards
- Evaluate the level of quality achieved against the established requirements
- Detect anomalies or points to potential problems in the system
- Predict the level of quality that will be achieved in the future
- Monitor changes in quality when software is modified
- Assess the ease of change to the system during product evolution
- Validate a metric sets .

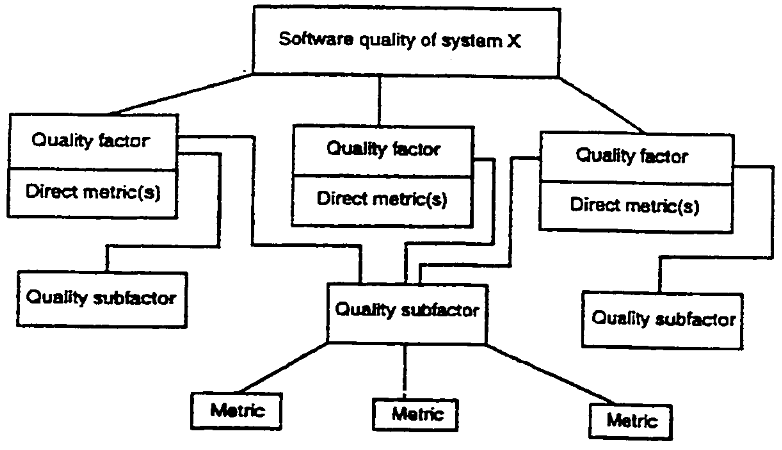


Figure 7: IEEE software quality metrics model

6. **Define-Your-Own-Model:** The define-your-own-model approach has been pioneered by Gilb and also by Kitchenham and Walker. Gilb's method can be thought of as "design by measurable objectives", it complements his philosophy of evolutionary development. The software engineer delivers the product incrementally to the user, based on the importance of the different kinds of functionality being provided. To assign priorities to the functions, the user identifies key software attributes in the specification. These attributes are described in measurable terms, so the user can determine whether measurable objectives, in addition to the functional objectives, have been met. This simple but powerful technique can be used on projects of all sizes. It is the antithesis of the more traditional "big-bang" development approach, where an entire system is delivered in one go, and there are no priorities assigned to the several functions [40].

   The COQUMO (COnstructive QUality MOdel) approach of Kitchenham and Walker extended Gilb's ideas and supported them with automated tools as part of an ESPRIT (European Community) project [45].

2.4.1 ISO/IEC 9126

The standard originally was introduced in 1991 and called Software Product Evaluation – Quality Characteristics and Guidelines for their use. ISO/IEC 9126 breaks software quality down into six broad categories of characteristics of the software. Later revision of this standard is considering quality in use as part of its model.

ISO/IEC FDIS 9126-1 (2000) Software Engineering - Product quality - Part 1: Quality model specifies a two-part model for software product quality (Figure 8) [69]:

a) internal quality divided into six characteristics: functionality, reliability, efficiency, usability, maintainability and portability. External quality is manifested when the software is used as a part of a computer system, and is the result of internal software attributes.

b) quality in use characteristics: effectiveness, productivity, safety and satisfaction. Quality in use is the combined effect for the user of the six software product quality characteristics.

The other parts of this standard are:

ISO/IEC PDTR 9126-2 (2000) Software Engineering - Product quality - Part 2: External metrics. These metrics are used when the software is operated.

ISO/IEC PDTR 9126-3 (2000) Software Engineering - Product quality - Part 3: Internal metrics. These metrics can be used when inspecting a specification.

ISO/IEC PDTR 9126-4 (2000) Software Engineering - Product quality - Part 4: Quality in use metrics. This report contains metrics for effectiveness, productivity, safety and satisfaction.

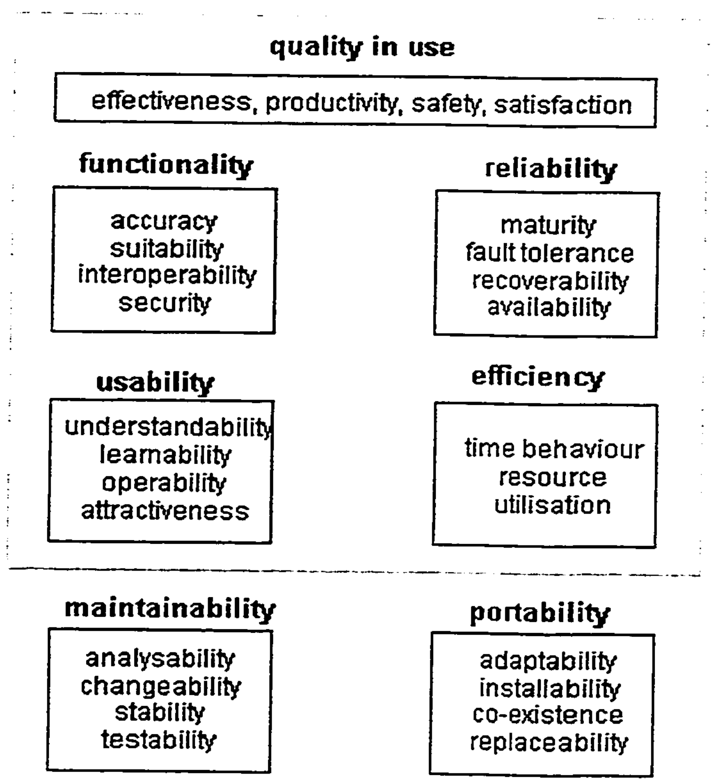


Figure 8: ISO/IEC 9126-1 quality model

As we see in the structure of model, quality in use is decomposed into Effectiveness, Productivity, Safety, and Satisfaction. Example of metrics from the standard is the following measure [70]:

Task completion = Number of tasks completed / Total number of tasks attempted

The metric is defining what proportion of the tasks are completed. It has a value between 1 and 0, and obviously the closer to 1 the better. This metric affects the effectiveness.

Another example is:

Productive proportion = (Task time – Help time – error time – search time) / Task time

This metric affects productivity and shows what proportion of the time is the user performing production actions. It takes a value between 0 and 1, the closer to 1 the better. Productive proportion requires detailed analysis of a video tape of the interaction between user and system. The problem with this model is the lack of prediction power.

2.4.2 ISO/IEC 14598-1

This standard is called Information Technology – Evaluation of Software Products – General Guide. The ISO/IEC 14598-1 suggests a model for studying and measuring quality in use from the internal software attributes in a particular context. Software quality attributes are the cause, quality in use the effect. Quality in use is (or at least should be) the objective; software product quality is the means of achieving it.

The user's needs are expressed as a set of requirements for the behavior of the product in use (for a software product, the behavior of the software when it is executed.) These requirements will depend on the characteristics of each part of the overall system including hardware, software and users.

The requirements should be expressed as metrics that can be measured when the system is used in its intended context, for instance by measures of effectiveness, efficiency and satisfaction.

External quality can only be assessed for a complete hardware/software system of which the software product is a part. External metrics are applied when executing the software. The values of external measures necessarily depend on more than the software, so the software has to be evaluated as part of a working system. Software that performs satisfactorily in one environment may show quality defects in another environment. External evaluation of quality characteristics should therefore take place under conditions that emulate as closely as possible the expected conditions of use. External measurements of characteristics are made when the code is complete, though as it may not be possible to emulate the exact conditions of use (e.g. network environment and user characteristics), external measures are often only indicators of the actual quality in use.

The required values of these external metrics provide goals for design. To achieve these goals the internal attributes of the system can be specified as internal requirements. These attributes of the software can be evaluated to produce internal metrics verifying how closely the internal requirements have been met. Although these attributes contribute to achieving quality in use, users and tasks vary so much that conformance to requirements for internal metrics is rarely sufficient to ensure quality in use.

If the external quality requirements are not achieved, the results of the evaluation can be used as feedback to modify the internal software attributes in order to improve the external quality, thus supporting a continual improvement process. And again this standard is a guideline and implementing it is another story, which is not covered by the document.

2.5 Usability Assessment Tools

In the following section we provide a review of user interface measurement and development tools.

2.5.1 SANe

SANe (Skill Acquisition Network method) is a method for analytic evaluation of the quality of use of interactive devices [64]. The SANe user interaction model describes user tasks, the dynamics of the device, and user procedures for the execution of user tasks.

In the measurement procedure a task model and a device model are developed and linked together. Then user procedures are simulated automatically for the linked model. 60 measures are calculated with a subset of 24 complexity measures related to reference values grouped to quality measures. Five quality measures are produced.

- Efficiency

  Efficiency is determined by the length of user procedures and the cost of executing user procedures

- Learning

  Learning cost is determined by the amount of states and state transitions.

- Adaptedness

  Adaptedness describes the adequateness of the functionality of the device for a given application domain.

- Cognitive workload

  This measure is determined by controllability, decision complexity and memory load. Controllability is the amount of states and the state the user must know.

Decision complexity is the alternatives from which the user can choose. Memory load is the information about the state variables the user must remember while using the device.

- Effort for error correction

This describes the robustness of a device and the cost for error recovery.

To conduct SANe evaluation expert HCI evaluators are needed. SANe can be used in the process of developing new software products, in procurement projects and for test of conformance with standards and guidelines.

2.5.2 DRUM

The Diagnostic Recorder for Usability Measurement (DRUM) helps evaluators to organize and analyze user-based evaluations, and to deliver measures and diagnostic data. DRUM consists of four modules [65]:

- management of data through the various stages of observational evaluation
- task analysis, and representation of classes of events and usability problems
- video control and creation of interaction logs of evaluation sessions
- analysis of logged data and calculation of metrics (log processor)

The Log Processor performs the calculation, from any log in the DRUM database, of performance measures and performance-based usability metrics, including:

- task time - total performance time for each task being studied (with a facility for subtracting times when the task is suspended)
- snag, help and search times - measures of the time users spend having problems, seeking help or unproductively hunting through a system

- effectiveness - derived from measures of the quantity and quality of task output, this is a measure of how fully, and how well, users succeed in achieving their task goals when working with a system
- efficiency - this relates effectiveness to the task time: it is a measure of the rate of producing the task output
- relative efficiency - a measure of how efficiently a task is performed by a specific user or group of users, compared with experts (or with the same task on another system)
- productive period - the percentage of task time not spent in snag, help and search. This indicates how much users of a system spend their time working productively towards their task goals.

Results are displayed in numerical and graphical form, and can be saved in the database for grouping of data from different subjects, for further statistical analysis.

2.5.3 AIDE

AIDE stands for semi-Automated Interface Designer and Evaluator [28]. This tool demonstrates the potential of incorporating metrics into the interface development process. It is capable of generating initial interface layouts and evaluating some aspects of a design. With this tool the burden of making the final decisions will still lie with the designer. AIDE is intended to demonstrate, how metrics may be applied to user interface design process. The tool focuses on providing advice to designers, not replacing them. In this way, AIDE facilitates rapid prototyping and may reduce costs by decreasing decision times. This tool integrates task-sensitive metrics as well as task-independent metrics. Task-sensitive metrics incorporate task information into the development process ensuring that the user's tasks guide the semantic design of the interface. Task-independent metrics tend to be based on principles of graphic design and help to ensure that the interface is aesthetic. The example for task-sensitive metric is Layout Appropriateness [20]. AIDE focuses on a single aspect of the design process which is organizing the items (e.g. buttons, menus, text boxes, ...) on the screen, once they have been chosen, and is intended to be one of many tools that a designer may use while developing an interface. Designer specifies weights for the metrics and AIDE displays a layout that optimizes the weighted metrics. Then the designer can interactively evaluate and alter the design. The tool uses five metrics, i.e. efficiency (this is the same as Layout Appropriateness), alignment, horizontal balance, vertical balance and constraints. Efficiency evaluates how far the user must move a cursor to accomplish the task. Alignment evaluates how well aligned objects are both vertically and horizontally. The balance metrics evaluate how well balanced the screen is both vertically and horizontally. Constraints provide a quick overview of the status of any designer specified constraints. The example for constraints could be, the designer wants to put an element onto a specific location on the screen, so that location is a constraint represented by its x and y.

2.5.4 GLEAN

Before talking about GLEAN, we would like to recall some words on GOMS and engineering models of human performance [54]. Engineering models for usability are analogous to the models used in other engineering disciplines in that they produce quantitative predictions of how well humans will be able to perform tasks with a proposed design. Such predictions can be used as a surrogate for actual empirical user data, making it possible to iterate through design revisions and evaluations much more rapidly. The overall scheme for using engineering models in the user interface design process is as follows: following an initial task analysis and proposed first interface design, the interface designer would then use an engineering model to find the applicable usability problems in the interface.

The major extant form of engineering model for interface design is the GOMS model. GOMS stands for Goals, Operators, Methods, and Selection rules [54]. A GOMS model is a description of the knowledge that a user must have in order to carry out tasks on a device or system. Briefly, a GOMS model consists of descriptions of the methods needed to accomplish specified goals. The methods are series of steps consisting of operators that the user performs. A method may call for sub-goals to be accomplished, so the methods have a hierarchical structure. If there is more than one method to accomplish a goal, then selection rules choose the appropriate method depending on the context. Describing the goals, operators, methods, and selection rules for a set of tasks in a formal way is called a GOMS analysis, or constructing a GOMS model. GOMS models and techniques could be used for predicting key aspects of usability of an interface. In particular, execution time can be predicted by simulating the execution of the methods required to perform a task. The time to learn how to operate the interface can be predicted from the length of the methods and transfer of training from the number of methods or method steps previously learned. One important feature of these GOMS models is that the "how to do it" knowledge is described in a form that can actually be executed either by analyst or by a computer tool.

GLEAN stands for GOMS Language Evaluation and Analysis. In essence, it is a facility for simulating the interaction of a simulated user who interacts with a simulated device (the computer interface) to execute a set of benchmark tasks. To set up the simulation, the designer supplies three representations contained in simple text files and expresses in defined notation. The first one gives the description of the tasks, the second one gives the user's procedural knowledge, i.e. GOMS model, and third one expresses the behavior of the simulated interface, e.g. location of icons and their behaviors in response to user input. Then running GLEAN, the Evaluator/Interpreter generates measures of usability from the GOMS model such as the predicted procedure learning time.

2.6 Lessons learned from Software Measurement Programs

As software become more pervasive and software quality more critical, measurement programs will become more necessary. In 1990, Rubin reported that 300 US information technology companies (companies with at lest 100 IT staff) had implemented measurement programs [40]. Among them he determined that sixty were successful, where success means:

- The measurement program results were actively used in decision making
- The results were communicated and accepted outside of the IT department
- The program lasted more than two years

Rubin's colleague, Verdugo, suggests several reasons for failure in the remaining IT companies:

- Management did not clearly define the purpose of the program and later saw the measures as irrelevant
- Systems professionals resisted the program, perceiving it as a negative commentary on their performance
- Already burdened project staff were taxed by extensive data collection requirements and cumbersome procedures
- Program reports failed to generate management action
- Management withdrew support for the program, perceiving it to be mired in problems and no-win situations

Grady and Caswell list ten steps [40] to success for a measurement program, based on their experience at Hewlett-Packard (1987):

1. Define the company and project objectives
2. Assign responsibility to each activity
3. Do research
4. Define the initial metrics to collect
5. Sell the initial collection of these metrics
6. Get tools for automatic data collection and analysis
7. Establish a training class in software measurement
8. Publicize success stories and encourage exchange of ideas
9. Create a metrics database
10. Establish a mechanism for changing the standard in an orderly way

To have a successful usability measurement, we adopt some of the above lessons as follows:

1. Define the usability objectives of the project
2. Define the metrics regarding to the objectives
3. Keep the metrics simple and easy to understand
4. Automate data collection and analysis as much as possible
5. Create a metrics database

The following is the list of lessons learnt from experiences on measuring and evaluating software engineering processes and products in a variety of project environments, by Victor Basili and Dieter Rombach [48]. They declared those lessons as measurement principles. The first four measurement principles address the purpose of the measurement process, i.e. why should we measure, what should we measure, for whom should we measure. The remaining ten measurement principles address metrics and the overall measurement process, including characteristics of metrics, i.e., what kind of metrics, how many are needed, and characteristics of the measurement process, i.e., what should the measurement process look like, how do we support characterization, planning, construction, learning and feedback.

1. Measurement is an ideal mechanism for characterizing, evaluating, predicting and providing motivation for the various aspects of software construction processes and products. It is a common mechanism for relating these multiple aspects.
2. Measurements must be taken on both the software processes and the various software products. Improving a product requires understanding both the product and its construction processes.

3. There are a variety of uses for measurement. The purpose of measurement should be clearly stated. We can use measurement to examine cost, effectiveness, maintainability, efficiency, user friendliness, etc.
4. Measurement needs to be viewed from the appropriate perspective. The corporation, the manager, the developer, the customer's organization and the user each view the product and the process from different perspectives. Thus they may want to know different things about the project and to different levels of details.
5. Subjective as well as objective metrics are required. Many process, product and environment aspects, can be characterized by objective metrics (e.g., product complexity, number of defects or effort related to processes). Other aspects can not be characterized objectively yet (e.g., experience of personnel, type of application, understandability of processes and products), but they can at least be categorized on a quantitative (nominal) scale to a reasonable degree of accuracy.
6. Most aspects of software processes and products are too complicated to be captured by a single metric. For both definition and interpretation purposes, a set of metrics (a metric vector) that frame the purpose of measurement needs to be defined.
7. The development and maintenance environments must be prepared for measurement and analysis. Planning is required and needs to be carefully integrated into the overall software engineering process model. This planning process must take into account the experimental design appropriate for the situation.

8. We can not just use models and metrics from other environments as defined. Because of the differences among execution models (software engineering models) the models and metrics must be tailored for the environment in which they will be applied and checked for validity in that environment.
9. The measurement process must be top-down rather than bottom-up in order to define a set of operational goals, specify the appropriate metrics, permit valid contextual interpretation and analysis, and provide feedback for tailorability and tractability.
10. For each environment there exists a characteristic set of metrics that provides the needed information for definition and interpretation purposes.
11. Multiple mechanisms are needed for data collection and validation. The nature of data to be collected determines the appropriate mechanism, e.g., manually via forms or interviews, or automatically via analyzers.
12. In order to evaluate and compare the projects and to develop models we need a historical experience base. This experience base should characterize the local environment.
13. Metrics must be associated with interpretations, but these interpretations must be given in context.
14. The experience base should evolve from a database into a knowledge base, supported by an expert system, to formalize the reuse of experience.

2.7 Summary

Today, it is proved that measuring and using metrics is a convenient type of control method that we can apply in an engineering process. This type of process control has been applied for several years in software production. Experts have provided their experiences, and we will use those experiences for developing QUIM, our model to assess the quality in use of software products. Also there are weaknesses with every model or guideline which is used today by software developers to evaluate the quality in use. We will try to remove those problems in QUIM.

Chapter 3:

QUIM - Quality in Use Integrated Model

In this chapter we will examine the rational for QUIM and its structure. We will see also, how we may use the model.

3.1 Requirements and Justification

The following are the main reasons for developing QUIM:

1. The importance of quality in use in a software product, so far the proper attention has not been paid to the usability in software quality models.
2. Assuring the quality in use objectively rather than subjectively, because numbers are stronger than words.
3. Reducing the risks of software product regarding the user interface.
4. Reducing testing time and testing expenses of a software product.
5. Incorporating quality in use assurance into the first stages of software development.
6. Making the acquiring of quality in use easy for all developers with different levels of experience.
7. Making the acquiring of quality in use more consistent between developers.
8. Helping the developers to create more consistent software products.
9. Providing a dynamic quality model.
10. Creating a collection of usability metrics that are developed so far by different developers and organizations.

11. Facilitating the integration of usability into the software engineering models.
12. Filling the gap between Human-Computer Interaction experts and software engineers.
13. Integrating all usability attributes of a software that have been recognized into one model.

Here we would like to discuss some aspects of the above reasons.

For the assessment of the quality in use of a software interface, we may distinguish two broad classes of methods. One class requires direct participation of the end user and the other applies just to the attributes of the graphical interface. We may use first class methods to evaluate the product, but for prediction, we are forced to use the second class. The second category includes expert evaluation, analytical methods, usability inspection methods and so on. Usually in these methods, usability experts are concerned with evaluating the conformance of interface's attributes to guidelines, principles, heuristics and standards. As an example, Smith and Mosier's 1986 guideline [7], which is a good one, contains 944 instructions for designing a user interface, they are stated in the form of single statements [55]. Some guidelines use illustrated examples such as Macintosh human interface guidelines [56]. Conducting surveys on use of the guidelines [57], shows that the designers had difficulties in locating the relevant instruction within the guideline assembly, choosing which instruction to use and translating general guidelines into specific design rules [58].

A comparison of four user interface evaluation techniques [60] in 1991, gave the following results as their disadvantages.

- Heuristic evaluation needs skilled heuristic evaluator
- Software guidelines missed a large number of the most severe problems with the user interface
- Cognitive walkthrough needs a task definition methodology, it is tedious to implement, and it missed general problems
- Usability testing did not find all serious problems (evaluator gather data as problems arise), also it missed consistency problems and is of a high cost

But the major drawbacks of those methods are the requirement of usability expert and participation of the end-user.

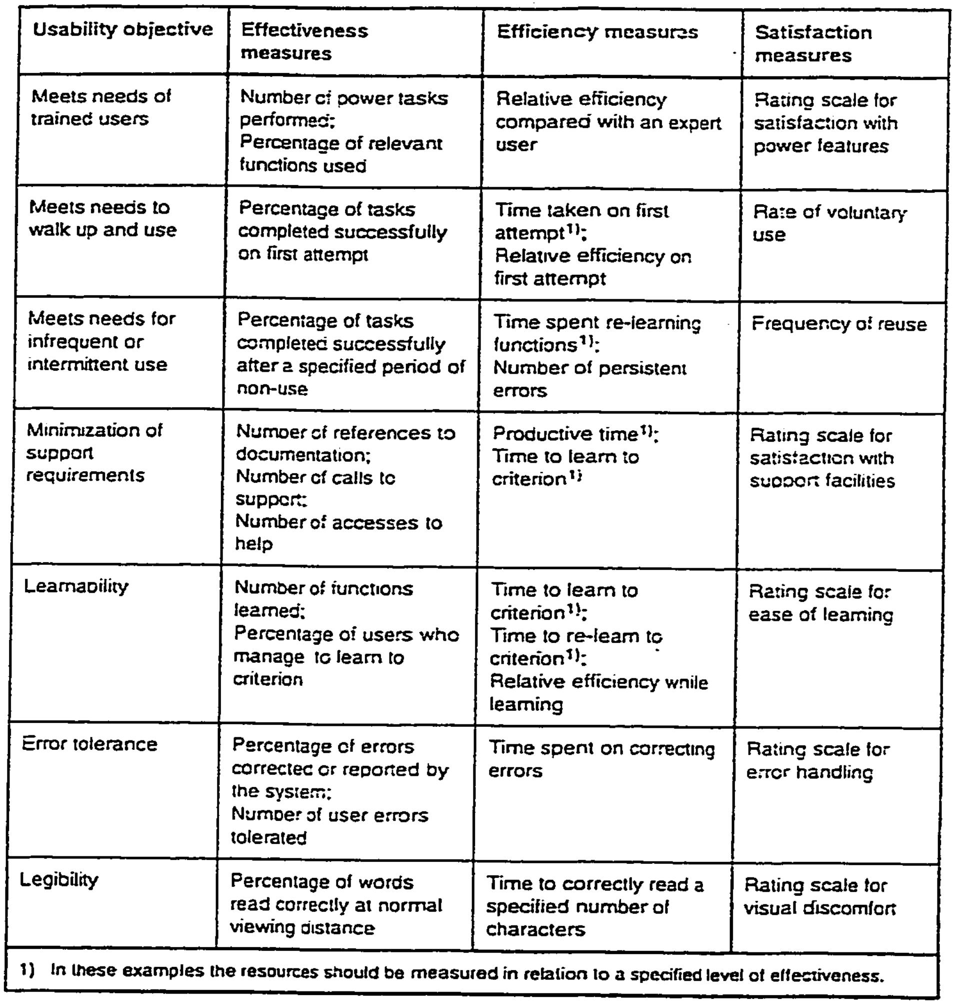
Then to achieve the goal, i.e. quality in use, we need something more concrete and reliable. We need something that provides coherent results over time, and consistency among two or more designer or evaluator, and even non-expert in usability can benefit from it. Also it could be applicable to all phases and artifacts of development. That is a Metrics based model using quantitative entities.

Numbers always speak louder than words. If you can say that there are twice as many customers who are capable of finding and ordering products after a redesign of an e-commerce web site, then your message carries much more weight than if you give a speech about the improved web navigation system and streamlined check out process. It is useful to track the evolution in the usability metrics over time. Doing so will tell you how much you are improving, or whether your latest great redesign ended up hurting users.

The main advantage of the metrics-based model is that it provides a definition of usability against which the system can be tested. Unfortunately, skilled software engineers in usability are required and the measurements are made in unnatural laboratory environments hence the context of the workplace is lost. Like metrics in software engineering, they are difficult to interpret or translate into the design solution.

Then if we make a model, i.e. arrange every thing formally, give the definition of a metric, how to calculate it, how to interpret it, and how it affects the product's quality, it could be easier and more convenient to use those metrics.

There are a couple of software quality models, but not a usability model. In every quality model usability is considered just as a small part of it, and nobody has paid enough attention to this important aspect of a software product. We would like to recall that usability is one of ten high-risk items in software production and it is one of the three views to software quality. There is just one standard, i.e. ISO 9241-11, called guidance on usability. It gives some guidance in considering context of use in usability and usability objectives, also in its appendices it gives some examples for usability metrics. Here we would like to mention a sentence from the standard: "ISO 9241-11also provides a basis from which measures of usability can be generated. Product developers can develop appropriate measures of efficiency, effectiveness, and/or satisfaction". Figure 9 shows a list from appendix B of the standard. The problem is here, not every software developer has experience in developing metrics. Developing a metric is not a very simple process. A metric needs validation. This is time consuming. On the other hand we have certain categories for software products and entities in each category may benefit from the same set of measures. Then it is obvious that every software developer prefers to use, ready to use metrics, to increase the quality of his job. Another problem with suggested metrics in the standard is that, they are all useful for evaluating the interface, it means they are used at the later stages of development. Finally, ISO 9241-11 is a guidance for software metrics and quality model developers, not for the software developers.



Usability objective	Effectiveness measures	Efficiency measures	Satisfaction measures
Meets needs of trained users	Number of power tasks performed; Percentage of relevant functions used	Relative efficiency compared with an expert user	Rating scale for satisfaction with power features
Meets needs to walk up and use	Percentage of tasks completed successfully on first attempt	Time taken on first attempt[1]; Relative efficiency on first attempt	Rate of voluntary use
Meets needs for infrequent or intermittent use	Percentage of tasks completed successfully after a specified period of non-use	Time spent re-learning functions[1]; Number of persistent errors	Frequency of reuse
Minimization of support requirements	Number of references to documentation; Number of calls to support; Number of accesses to help	Productive time[1]; Time to learn to criterion[1]	Rating scale for satisfaction with support facilities
Learnability	Number of functions learned; Percentage of users who manage to learn to criterion	Time to learn to criterion[1]; Time to re-learn to criterion[1]; Relative efficiency while learning	Rating scale for ease of learning
Error tolerance	Percentage of errors corrected or reported by the system; Number of user errors tolerated	Time spent on correcting errors	Rating scale for error handling
Legibility	Percentage of words read correctly at normal viewing distance	Time to correctly read a specified number of characters	Rating scale for visual discomfort

1) In these examples the resources should be measured in relation to a specified level of effectiveness.

Figure 9: The examples of metrics from ISO 9241-11

Now we analyze the existing software quality models. The previous chapter introduced them in details. Here we would like to analyze their drawbacks and strong points.

The problem with QFD (Quality Function Deployment) is that it needs lots of experimental results and an experienced judge to assign software attributes to the requirements. In fact every time matrices should be constructed from scratch. Another problem with the model is providing the hierarchy in separate matrices, then it is difficult to follow up the relationships and it is easy to miss some attributes and metrics.

GQM (Goal, Question, Metric) is more a philosophy than a quality model. The process of setting goals and refining them into quantifiable questions is complex and requires experience. Of course in order to support this process, a set of templates for setting goals, and a set of guidelines for deriving questions and metrics is developed, but not on usability. Then it is necessary to define those templates for usability.

ISO 9126, IEEE 1061, McCall and Boehm models are all of a hierarchical shape, consisting of several layers. Table 3 shows the terminology used for different layers in those models. We believe the tree shape is the best form to show the relationships between several related entities. Then we adopt this form to create QUIM. Of course those models are not useful by themselves in this context. There is not any suggested metric with IEEE 1061. Metrics provided in ISO/IEC DTR 9126-4 have no prediction power. There is nothing about usability in Boehm model. In McCall model and ISO 9126, usability has come just as a quality factor. Measuring many of the quality factors described in formal models, including McCall's and Boehm's is dependent on subjective rating.

One difference between QUIM and those models, is the number of layers. QUIM defines required data for calculating a metric in a separate layer. In this way QUIM makes using a metric clearer and easier.

Layer	Boehm	McCall	ISO 9126	IEEE 1061
1	Primary uses	Use	Software Quality	Software Quality
2	Intermediate constructs	Factor	Characteristics	Factor
3	Primitive constructs	Criteria	Sub characteristics	Sub Factor
4	Metrics	Metrics	Metrics	Metrics

Table 3:The correspondence of terminology between quality models

One problem with existing models and their metrics is the clarity. Fenton and Pfleeger [40], define Portability as:

$$Portability = 1 - ET/ER$$

Where ET is a measure of the resources needed to move the system to the target environment, and ER is a measure of resources needed to create the system for the resident environment. Gilb recommends that the decision for setting measurable targets for these attributes should be left with the user. The problem here is clarity, is this user the developer, meaning the user of evaluation formula or he/she is the end user of system. What are the possible entities for resources? We know that different stack holders (Managers, Developers, Users) have different resources in interest.

Finally QUIM is an integrated model because;

- It supports quality activities during the whole development life cycle of software, including specification, testing, quality improvement, and so on.
- It brings together both software engineering and Human Computer Interaction quality models. Software engineering models view the quality problem from the engineering view such as McCall or IEEE 1061, but HCI models mostly are seeing the quality , subjectively and heuristically, such as Heuristic evaluation method introduced in chapter 1. For example, the model says "Minimize user memory load", but using this model needs an experienced expert and sometimes knowledge of human factors and psychology. In QUIM we provide the way to achieve the goal, using an engineering approach, Interface shallowness or Visual coherence can affect the user memory load. In addition we are able to determine those metrics during the design phase, and if the result is not satisfactory, developer can redesign his model.
- QUIM is an integration or a large repository of all factors, criteria and metrics that are recognized and defined so far, for quality in use of a software product.
- QUIM is not independent of the other software quality views, i.e. Manager's view and developer's view. It could be easily integrated into any larger software quality model providing all perspectives of software quality.

3.2 Design principles of QUIM

The development of the model has been guided by the following goals:

- Decomposing attributes: Software consumers often express their needs in general qualitative terms such as reliability and efficiency. So it is necessary to decompose consumer-oriented attributes into technically-oriented attributes that are more meaningful to software producers.

- Functionality: The model should cover the entire software development process.
- Usability: The model should help the software developer or interface developer to assure, as early as possible, that the resulting system will be easy to learn and easy to use.
- Maintainability: Since the software quality in use is directly dependent on the context of use, then the model should be modifiable to reflect developer organization needs.
- Automated support: The model should be supported by automated tool that improves the process of software development and the resulting system.

3.3 QUIM Structure and Description

QUIM is a hierarchical model like most of software engineering models. The difference is that it distinguishes five levels called factors, criteria, metrics, data and artifacts (Figure 10). The relationship between these layers is an N-M relationship.

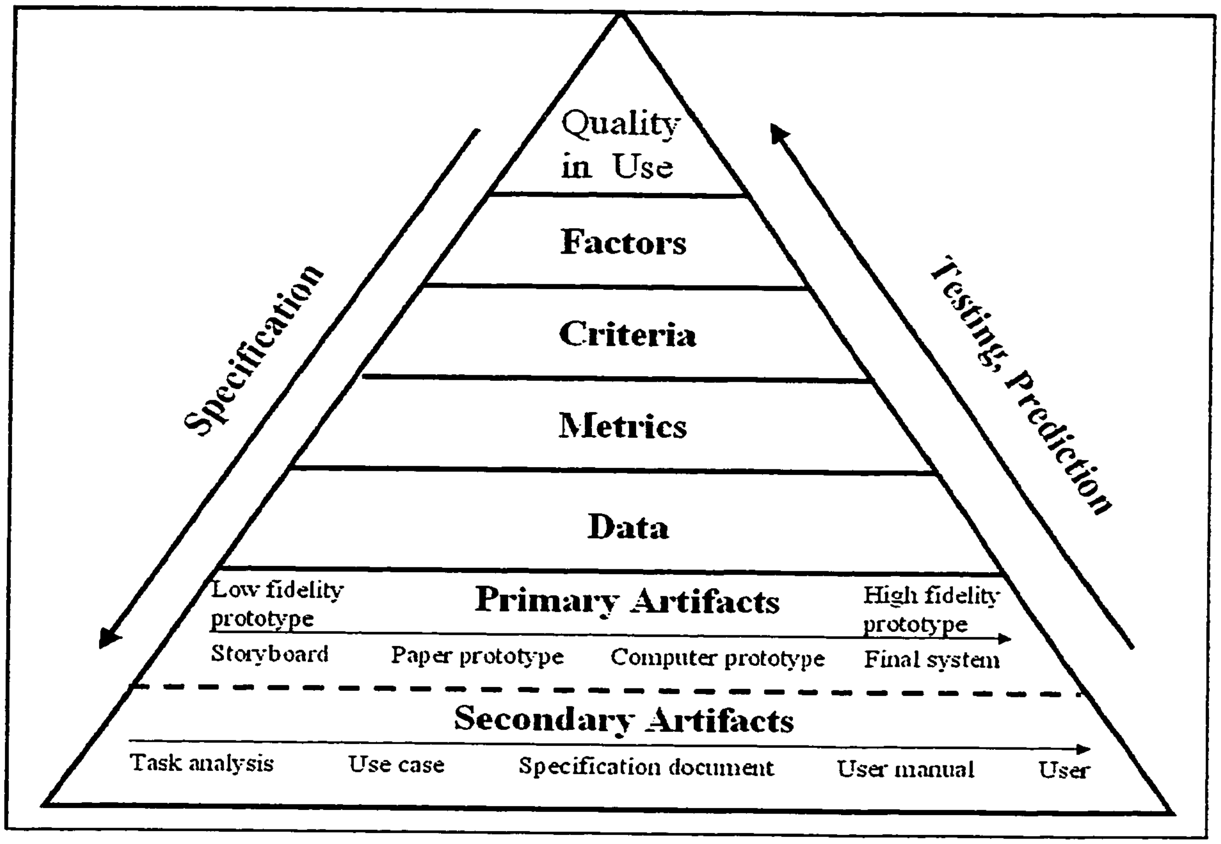


Figure 10: The hierarchy of QUIM

3.3.1 Quality in Use

In QUIM, we define quality in use as the end user perspective of software quality. That is one of the three perspectives of software quality. The definition of quality in ISO 8402 reflects the user view. User is mainly interested in using the software, its performance and the effects of using it. Then the user has no interest in internal aspects of the product. The user just sees observable external attributes of the software. Or one may say that the user is interested in final product quality. We may consider the definition of quality in use from the revision to ISO/IEC 9126-1 and ISO/IEC 14598-1 as well. They say "Quality in use is the extent to which a product used by specified users meets their needs to achieve specified goals with effectiveness, productivity and satisfaction in a specified context of use". Then we define some characteristics that are applicable as generic characteristics to all software products and leave the model open for context specific characteristics.

3.3.2 Factors

In the context of this research, a quality in use factor is a user-oriented attribute or characteristic of the user interface. This characteristic is easy to understand by the user, and defines the quality of user interface in the user language. But it is not easy to measure and specify. A factor could be refined into the sub-factors or criteria. Currently, the list of Factors of QUIM consists of the following:

1- **Effectiveness:** The degree of accuracy and completeness with which user achieves a specified goal. [4]

2- **Efficiency:** The amount of resources expended in relation to the accuracy and completeness with which user achieves a goal. [4]

3- **Satisfaction:** Freedom from discomfort and positive attitude towards the use of the software product. [4]

3.3.3 Criteria

A criterion is a sub factor or sub-characteristic of the user interface, and it is a more technical attribute. Criteria are more difficult to understand by the user, and define the quality in use in the language of user interface developer. A criterion could be determined by more than a metric. The list of Criteria of QUIM consists of the following:

1- **Understandability:** The extent bears on measuring the difficulty of user on understanding software functions, operations and concepts while user has no previous knowledge about software.

2- **Operability:** Measures the user's effort for operation and operation control [3]

3- **Attractiveness:** measures the extent of which user likes the software during the operation

4- **Compliance:** Attributes of interface that make it adhere to related standards or conventions or regulations in laws and similar prescription [3]

5- **Consistency:** Attributes that bear on the visual uniformity of user interface [14]

6- **Flexibility:** Indicates the degree of possible modification to user interface by the user, i.e. The user can adopt his/her preferences

7- **Minimal Action:** The extent to which user needs to take minimal effort to achieve a specific task [14]

8- **Minimal Memory load:** The extent to which user needs to keep minimal amount of information in mind to achieve a specified task [14]

9- **User Guidance:** Indicates How the interface helps the user to use the application [14]

10- **Accuracy:** Indicators that bear on the provision of right or agreed results or effects [3]

11- **Completeness:** The extent to which the user can complete a specified task

12- **Resources:** Attributes that bear on the amount of resources used and the duration of such use in performing its function [3]

3.3.4 Metrics

The IEEE metrics standard [42] defines a software quality metric as "a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality ". In the context of this research, the output of a metric function is a numeric value that summarizes the status of specific user interface attribute. The advantage of using metrics is that they are faster, cheaper and less ambiguous than other usability evaluation entities, actually in most cases using these metrics does not require test users and additional usability experts. About context specific metrics, care should be taken in generalizing the results of any measurement of quality in use to another context which may have significantly different types of users, tasks or environments.

We have identified about 40 usability metrics, some of them are functions and some are just simple countable data. We may categorize the second group of metrics in the data level of QUIM. As examples of metrics, we are going to introduce and use one that is defined previously and has been validated, and also is general enough, so it could be applied to most of software and context of use. To have detailed explanation and examples of calculation one may refer to reference [22].

Layout Uniformity measures selected aspects of the spatial arrangement of interface components without taking into account what those components are and how they are used. The metric is not task sensitive, i.e. it could not been affected if the number of operations or their order to complete a task is changed. For developers who do not have a graphic designer's eye for layout, Layout Uniformity is a structural metric that gives a quick handle on one important aspect of visual design. It assesses the uniformity or regularity of the user interface layout. The rational is that, usability is hindered by highly disordered or visually chaotic arrangements. Of course complete regularity is not the goal. Too much uniformity not only can look unappealing but also can make it harder for users to distinguish different features and different parts of the interface. Layout Uniformity is defined as:

$$LU = 100 \times (1 - (((Nh + Nw + Nt + Nl + Nb + Nr) - M) / 6 \times Nc - M))$$

Where Nc is the total number of visual components on the screen, a dialogue box or other interface composites. Nh, Nw, Nt, Nl, Nb and Nr are, respectively, the number of different heights, widths, top-edge alignments, left-edge alignments, bottom-edge alignments and right-edge alignments of visual components. M is an adjustment for the minimum number of possible alignments and sizes needed to make the value of LU range from 0 to 100 and is calculated by:

$$M = 2 + 2 \times \lceil \sqrt{Nc} \rceil$$

Layout Uniformity goes up when visual components are lined up with one another and when there are not too many different sizes of components. The role of Layout Uniformity can best be appreciated by the examples. Figure 11 shows one view of the software called Blender and figure 12 represents a view of another software called 3DStudioMax. Both software are applications for creating computer graphics and computer animation. 3DStudioMax is a commercial product and Blender is a Freeware. We have to consider that such software by nature are not easy to use, then the job of quality in use model is more important in creating these kind of user interfaces. I calculated Layout Uniformity on both views in figures 11 and 12, just considering push down buttons. There are other visual components such as text boxes and tabs. Data for Blender are Nc = 84, Nh = 5, Nw = 10, Nt = 16, Nb = 16, Nl = 35, Nr = 33 and LU for this view is 81%. On 3DStudioMax data are Nc = 68, Nh = 5, Nw = 6, Nt = 13, Nb = 13, Nl = 53, Nr = 54 and LU is 63%.

As we see Blender has more buttons than 3DStudioMax, but it gives higher value for LU. The difference here is the distribution of buttons on the screen. In Blender all buttons are placed on one partition but in 3DStudioMax, buttons are distributed along with three edges of the screen.

The complete list of identified usability metrics has come in the appendices A.1 and A.2, including a brief description of every software metric, what is its impact and how to calculate it.

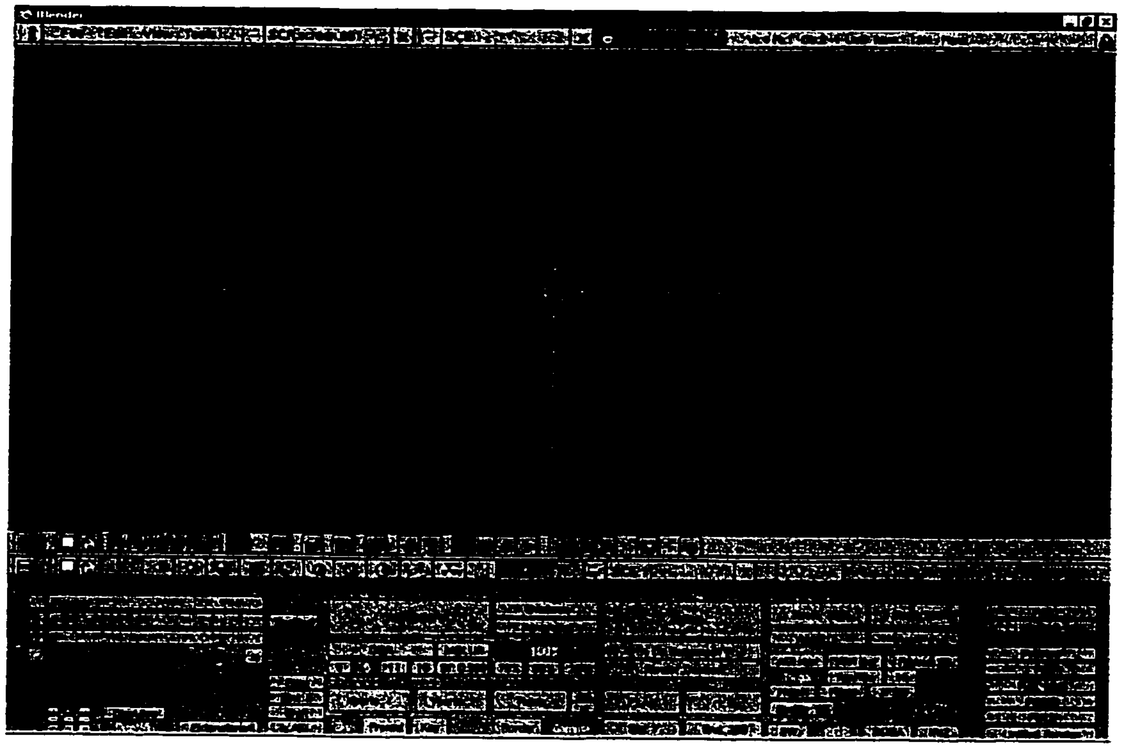


Figure 11: One view of the user interface of Blender

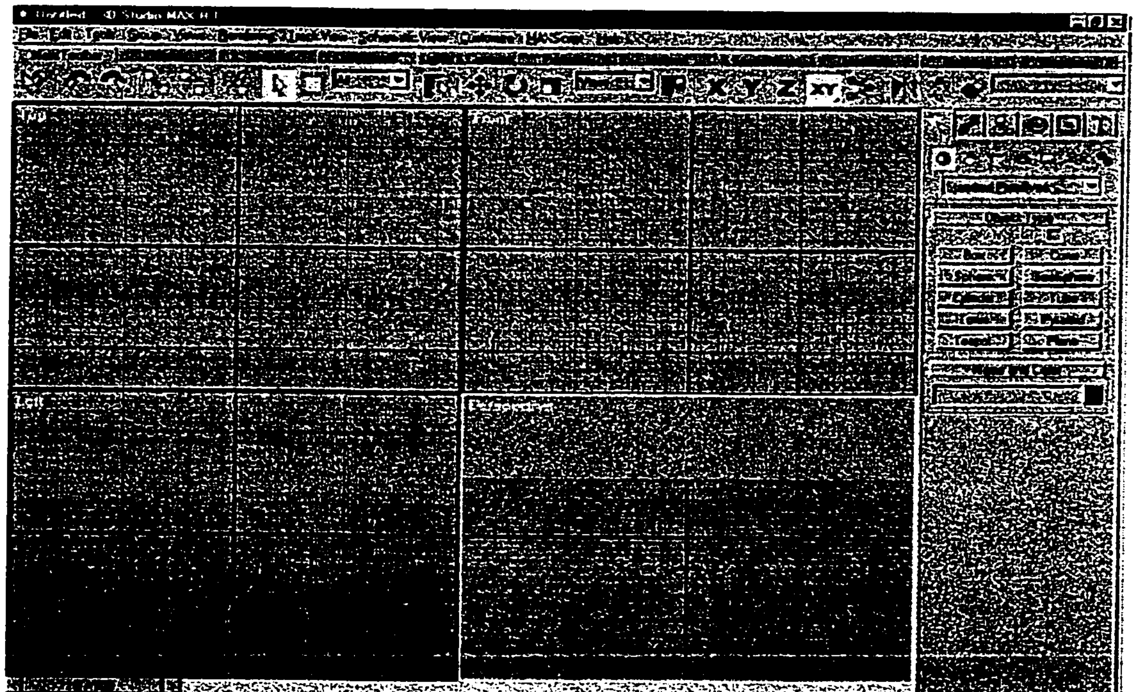


Figure 12: One view of 3DStudioMax

3.3.5 Data

The lowest layer of QUIM is the list of data that are used to calculate metrics. There are different methods to acquire the data, or in the other words we can have different types of data:

- Countable: Sometimes it is a countable entity, and in that case it could be considered as a metric and directly related to a criterion. The example is, the Number of individual items on the screen, or Time to complete a specific task. The data gathered from questionairs are falling also in this category, and they feed input of statistical function metrics.
- Calculable: Some data are determined by calculation. For example, percent of task completed. This data is going to be used in metric Task Effectiveness:

$$TE = Quantity \times Quality / 100$$

In the formula quantity is the percent of task completed (analogous to completeness) and quality is the percent of goal achieved (analogous to correctness), e.g., when evaluating an information retrieval system the user might retrieve specific items of information. This output is then analyzed to measure how many of the specified items the user retrieved, and how well all the retrieved items met the original requirements.

A metric generally needs different data that come out from different sources. The following are some of the sources that are used in our model to gather data:

1- User

2- Final system

3- User documentation

4- Use case

5- Task analysis

6- Prototypes including storyboard, paper prototype, computer prototype

7- Software process documents (specification, Design, Implementation)

3.4 QUIM Applications

QUIM as we mentioned before is not exactly a tree, because, for example, a specific metric could affect more than one criteria and then is connected to more than one criteria node. This is true for every level in QUIM. Figure 13 is an example that shows those relationships. As we see in the example, the data "Number of visual components" is an input to two different metrics "Visual Coherence" and "Layout Uniformity". Those metrics affect criterion "Minimal Memory Load" and the latter in turn affects factors "Efficiency" and "Satisfaction". Whatever is coming in the example map is just to show that the relationship between components of QUIM can be very complex.

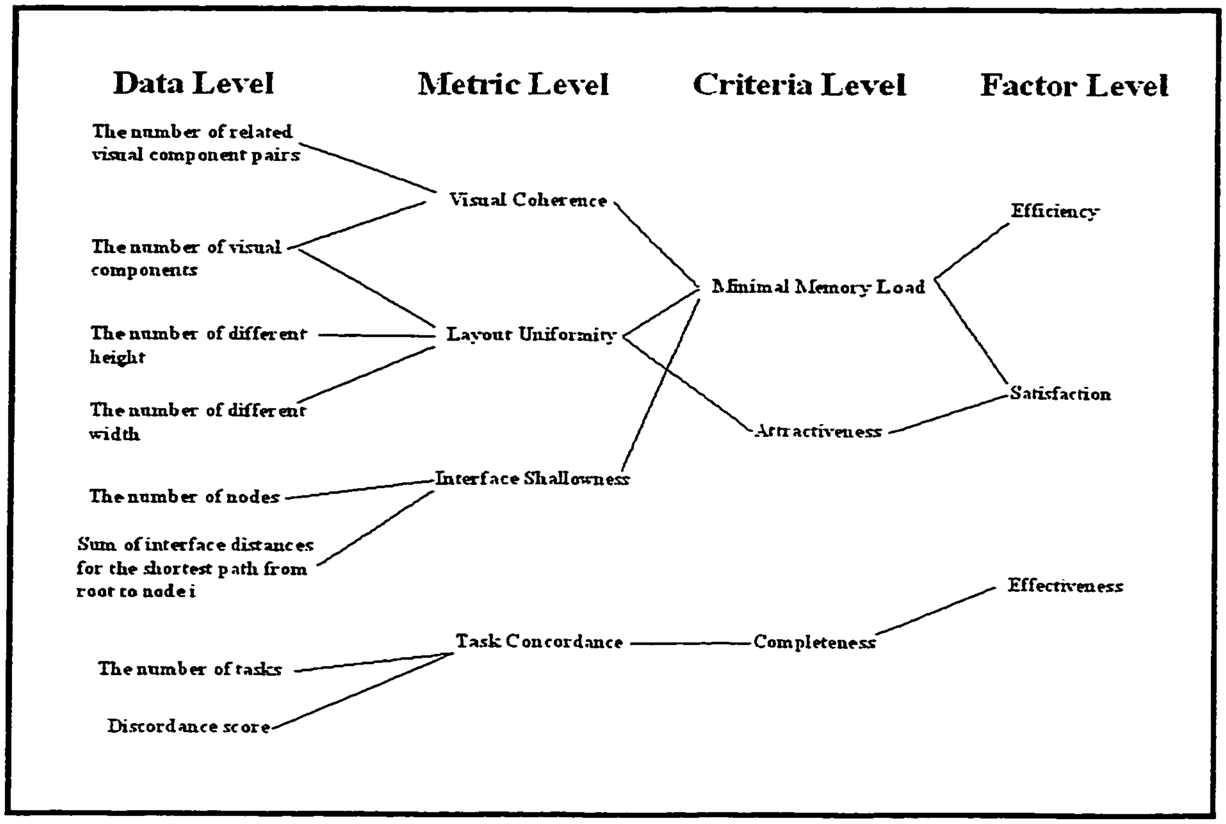


Figure 13: An example tree of relationship between QUIM components

3.4.1 Usability Specification

If you start from top and you say I want to meet the user satisfaction, then the model gives you all the criteria that should be met. In order for every criterion you may find related measurable or metrics, and finally at the lowest level of hierarchy, i.e. Data level, it gives you the list of required data and the ways of collecting them to determine the value of a metric. Then the developer can put goal values on the metrics and prepare his/her specification document or a test plan.

3.4.2 Usability testing and Prediction

The other direction of moving on the model is bottom-up. One may use it for prediction of usability. Suppose you have the specification and you would like to see how different goal values for a metric, or different combination of metrics can affect your system's quality in use. It is more clear if we go with an example (Figure 13). Consider metric interface shallowness who affects criterion minimal memory load and in turn the latter affects factor satisfaction. If we reduce interface shallowness and so the users' memory load, then we have increased the user satisfaction. Again consider another path, that is metric Layout Uniformity who affects criteria Attractiveness and Minimal Memory load, and those criteria affect factor Satisfaction. It means if we increase Layout Uniformity, we have increased Attractiveness and decreased the user memory load, and the final result of this operation is increasing the user satisfaction. We see to acquire Satisfaction, Layout Uniformity is more effective because they are related through two paths. Then we have to put more effort on Layout Uniformity rather than Interface Shallowness, during the design phase.

3.4.3 Other Potential Applications

QUIM is designed to support all activities related to quality during the whole life cycle of software. The following list presents some of its potential applications:

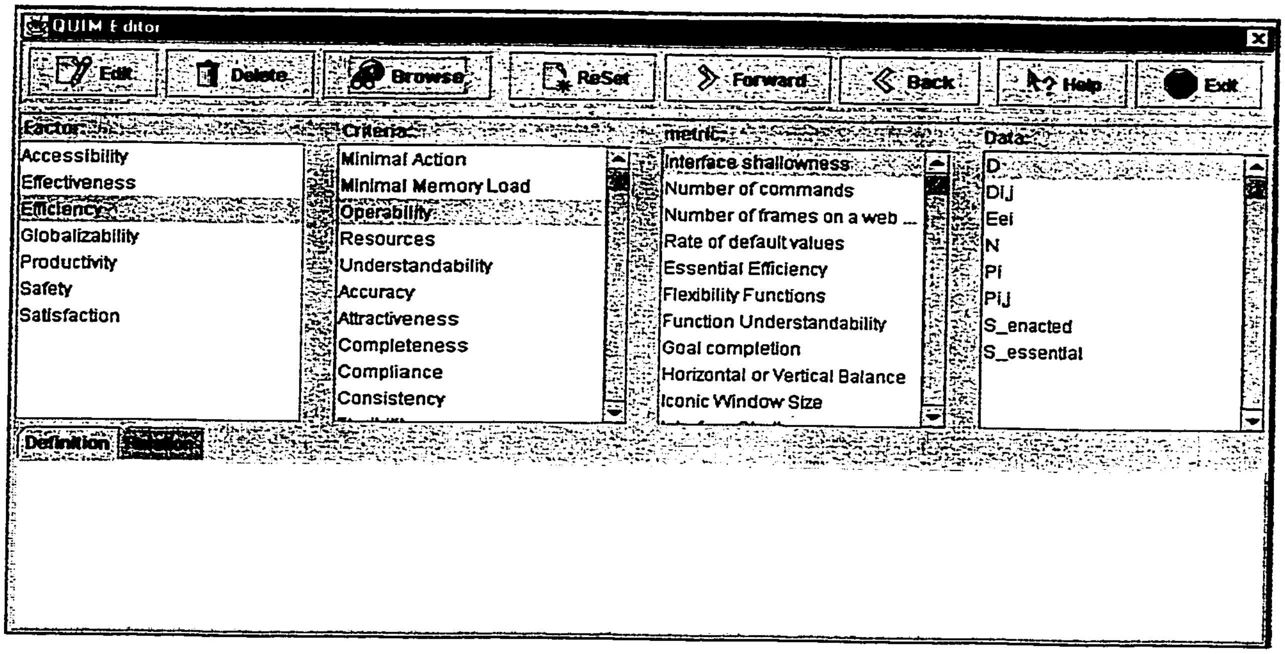
- Controlling and improving the production process
- Deciding in the acceptance of a product
- Selecting a product among alternative products
- Creating quality in use model for specific products

- Conducting empirical studies in usability

3.5 QUIM editor

To help the user interface developer in using QUIM, we developed a tool. The tool is called QUIM editor. It consists of two modules, i.e. User interface (Figure 14) and a database. The user interface is developed in Java, and the CASE tool in development process is Java Swing environment. The database module is developed in Microsoft Access. The two modules are communicating together through the ODBC (Open Data Base Connectivity, i.e. the technology from Micro Soft company that provides the required connection between applications and databases) interface. QUIM editor is running on Windows platform. This tool provides browsing functionality within the model in two directions, one is top-down, i.e. starting from a Factor and going towards the metrics and data, this direction in the tool is indicated by forward button. This could be used for creating software requirements specification. The other direction is bottom-up, which is used for testing and prediction purposes. The tool provides the functionality for editing the database. The organization can add any factor, criterion or metric to the database or remove from database. Using this functionality, QUIM behaves as a dynamic model. The modular design of QUIM editor provides the possibility of having different models in different databases.

Figure 19 in Appendix A.3 shows the E-R diagram of QUIM database. The section above dashed line is the currently implemented part of database and the lower section is the future extension to QUIM editor tool. This extension helps the organization to deposit the project specific data about their metrics and values. Those data repositories could be used in empirical studies of usability.



Factor:
Accessibility
Effectiveness
Efficiency
Globalizability
Productivity
Safety
Satisfaction

Criteria:
Minimal Action
Minimal Memory Load
Operability
Resources
Understandability
Accuracy
Attractiveness
Completeness
Compliance
Consistency

metric:
Interface shallowness
Number of commands
Number of frames on a web ...
Rate of default values
Essential Efficiency
Flexibility Functions
Function Understandability
Goal completion
Horizontal or Vertical Balance
Iconic Window Size

Data:
D
Di,j
Eei
N
Pi
Pi,j
S_enacted
S_essential

Figure 14: One view of QUIM editor

3.5 Summary

QUIM has a hierarchical structure and it distinguishes more layers than the existing models. It distinguishes required data for metrics and also the artifacts that can be used to acquire the data in separate levels. This model provides the information explicitly, then makes the job of developer easier. In this sense QUIM is a fixed model, and at the same time it provides the ability of adding any desired quality attribute to the model. Of course adding any additional attributes needs validation and empirical studies by R&D or research department in the organization.

Chapter 4:

Conclusion and Further investigations

In this chapter, we present the advantages of QUIM over the existing models. Also we provide a method to help the developer for analyzing the values obtained from QUIM. Finally we will see a list of topics that could be taken under investigation to complete this research.

4.1 Advantages of QUIM

Here we would like to express the advantages of QUIM over the other models.

1. QUIM is a dynamic model, it means the organization can add its own Factors, Criteria and Metrics to the model. This is not true about models such as McCall or ISO/IEC 9126. They are fixed models.
2. QUIM is as easy to use as a fixed model is. It helps any developer other than usability experts to create a usable interface. This is not true for models such as Quality Function Deployment and Goal-Question-Metric. Working with these models needs experiences in usability.
3. QUIM offers the organization, the power of prediction. Because it is using predictive metrics. Other models mostly just provide the power of evaluation.
4. QUIM makes the interfaces of different systems more consistent, because of the nature of its metrics. This characteristic makes the job of both developer and end-user easier.

4.1.1 Viewing several metrics at once

The model prescribes a set of metrics to reflect the product characteristics that are important in controlling or evaluating all or part of user interface development. Viewed in isolation, each metric describes a particular characteristic that can be compared with a goal or norm. Examined individually over time, these measurements can be helpful in noting trends and predicting future characteristics or outcomes. However, viewed in combination, measurements can provide a picture of the balance between factors that is often difficult to determine in other ways.

It has been known for a long time that when a set of characteristics is important, optimizing on a single one usually leads to unacceptable results in the others [49]. Thus, it is important to be able to view several metrics in context and to be able to balance the goals of one against the others.

A technique for depicting combination of metrics is called "multiple metrics graph" [50]. It is a variation of a Kiviat diagram or graph [51]. The Kiviat diagram, originally used to depict characteristics reported in simulations, displays characteristics on slices of a large, circular pie, as shown in figure 15.

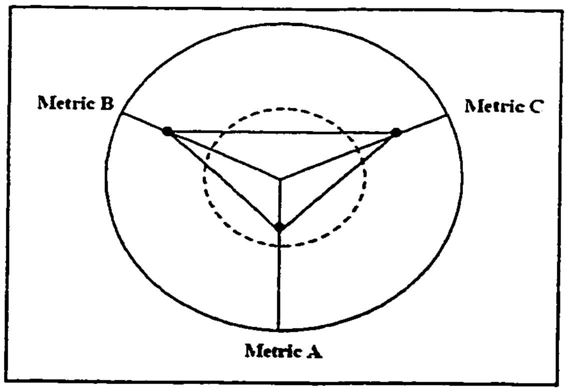


Figure 15: Kiviat diagram with three metrics

The pie is divided into equal slices, one for each measurement or characteristic to be presented. The inner circle represents a goal value or minimum required value, and the outer circle represents a maximum possible value. A point is placed on each line represents the measured value of metric, indicating its position with respect to the goal. Finally the points are connected, so that the resulting polygon is in some sense an indication of overall performance.

In a multiple metric graph, unlike the Kiviat graph, the pie is divided into unequal slices, so that the size of the slice represents the importance of the metric. The size of slice is indicated by the number of degrees in the arc. Thus, the larger the slice, the more important the metric. The small concentric circle represents the goal for each metric. Within each slice, a point is placed in the center, having equal distances from the adjacent radii, to represent the degree to which the goal is met. The center of the pie represents the best case and outer edge of the pie shows the worst case. The goal and radius are normalized among slices so that the goal arcs from a circle. Finally, a line is drawn from the representative point to the intersection of the goal line and radii forming each side of the pie slice.

How to analyze the graph? It is obvious that by connecting points we have a polygon. This polygon has its own area that could be compared with the goal circle area. polygon area shows the view of multiple metrics. Drawing graphs using data over time gives a good track of our job. Using different values lets us to compare generated graphs and to reach a compromise. Obviously, having points within the goal circle, means we met the goal. And the smaller polygon means closer to the target. By using the multiple metrics graph, we can see how we have improved and where more work is needed. A single index number for the quality factor or a metric can be generated, and it is directly related to the area of the overall polygon. We would like to recall that a quality factor based on the model QUIM is depending on several metrics, also a metric could be evaluated overall the whole software product. Examples can make the concepts clear.

Suppose we have four different interface modules or views, e.g. A, B, C and D, in our software product. We would like to reach a relatively good layout uniformity in the whole software. The value of layout uniformity is in a range from 0 to 100. We would like to recall from previous chapter that layout uniformity is based on the rationale that usability is hindered by highly disordered or visually chaotic arrangements of visual components. However complete regularity is not the goal. Too much uniformity not only can look unappealing but also can make it harder for users to distinguish different features and different parts of the interface. We can expect that moderately uniform and orderly layouts are likely to be the easiest to understand and to use. According to the experiences [22], the acceptable value lays between 50 and 80. And suppose the layout uniformity of module B is the most important and for the other three modules is of the same degree of importance. This decision could be taken by customer, end user, developer or even project manager. Then we give a weight of 40 over 100 to module B, and 20 to each of the other modules. We can draw the multiple metrics graph with slices of 144 degrees (0.40x360), and 72 degrees (0.20x360). Our goal value is 60, then if the radius of circle is 1, the radius of goal circle is 0.6. Figure 16 shows the multiple metrics graph for the values determined from the first design effort. From this, we can realize that we have to improve layout uniformity in modules B and D.

Module C already has met the goal and module A is close to the goal. Then if we improve modules B and D, i.e. trying to reduce the area of polygon and make it closer to goal circle area, we can get the desirable layout uniformity for the product. Figure 17 shows the desirable results after next iterations of design effort.

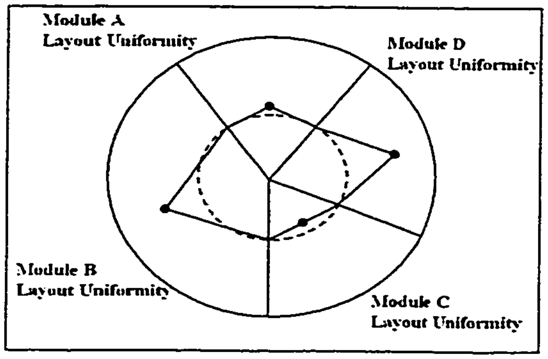


Figure 16: Example of multiple metrics graph

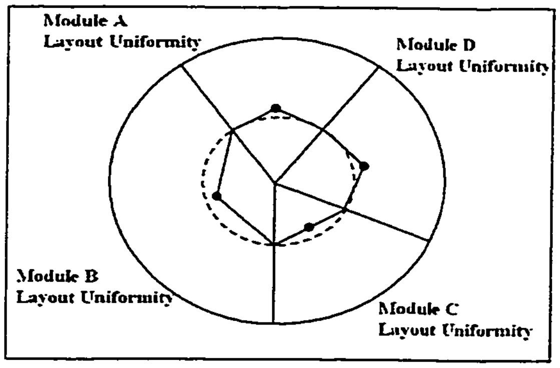


Figure 17: Refinement on example of multiple metrics graph

4.1.2 Coupling QUIM with other quality in use models

Even QUIM could be integrated into the other usability evaluation methods. Performance Measurement Method is a method developed at the National Physical Laboratory, UK, and is a part of MUSiC project. MUSiC which stands for Metrics for Usability Standards in Computing, is a project by Esprit, the European information technologies program. The method gives the quantitative results for usability testing of a user interface, and it follows the definition of usability in ISO 9241-11. It means the method consider the context of use in its input and output. Figure 18 shows the steps of process and its required tools. At step 3 it deploys guide and handbook to specify the usability targets. But, as we have seen before, using guidelines is tedious, vague and requires experienced experts. So if simply we substitute those tools with QUIM, it would be much easier to do the job, and every developer can handle it. Specially if we enhance our tool to a knowledge base system, instead of just a simple database.

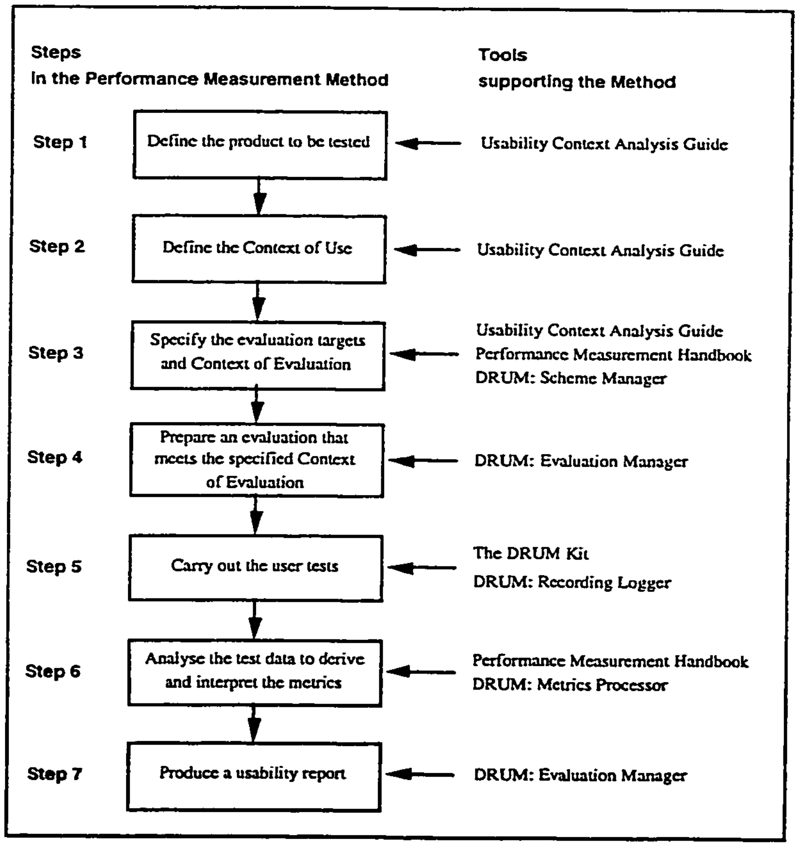

Steps
In the Performance Measurement Method
Tools
supporting the Method
Step 1
Define the product to be tested
Usability Context Analysis Guide
Step 2
Define the Context of Use
Usability Context Analysis Guide
Step 3
Specify the evaluation targets and Context of Evaluation
Usability Context Analysis Guide
Performance Measurement Handbook
DRUM: Scheme Manager
Step 4
Prepare an evaluation that meets the specified Context of Evaluation
DRUM: Evaluation Manager
Step 5
Carry out the user tests
The DRUM Kit
DRUM: Recording Logger
Step 6
Analyse the test data to derive and interpret the metrics
Performance Measurement Handbook
DRUM: Metrics Processor
Step 7
Produce a usability report
DRUM: Evaluation Manager

Figure 18: Steps and Tools used in Performance Measurement Method

4.2 Future works

4.2.1 Formalizing the validity of the relationships between Factors, Criteria and Metrics

When we say Layout Uniformity contributes to the criterion Minimal Memory Load, does this relationship shows consistency across all products. If LU of interface one is greater than LU of interface two, then does interface one creates less memory load for user than interface two or not. We need to validate the metrics and their relationships to the criteria. Informally, we say a metric is valid if it accurately characterizes the criterion it claims to measure. It means, we have to compare the model performance with known data, and this involves experimentation and hypothesis testing. It is necessary to conduct empirical studies to validate potential relationships between QUIM components. There are some methodologies presented by researchers for validating software metrics. Those methodologies could be deployed to validate quality in use metrics. I gave a brief description of methodology provided by IEEE organization, in chapter two, and a detailed process has been presented by Norman F. Schneidewind [68].

4.2.2 Extending the list of Factors, Criteria and Metrics

As we discussed in previous sections, QUIM is a dynamic model. Then one job that is necessary to be done on it is the extension of model. Here are some suggestions for new factors, but before adding them to the model they need some investigation.

Accessibility: The degree to which software can be used comfortably by a wide variety of people, including those who require assistance technologies like screen magnifiers or voice recognition. For example, an accessible JFC application employs the Java Accessibility API, enables its users to select an appropriate look and feel, and provides keyboard operations for all actions that can be carried out by use of the mouse.

Globalization: The degree to which software can be used for the global marketplace, taking into account variations in regions, population stereotypes, languages, and cultures.

Productivity: In our mind usually is equivalent to money, Oxford dictionary 2nd edition defines productivity in economics as:

"The rate of output per unit of input, used specially in measuring capital growth, and in assessing the effective use of labor, materials and equipment."

The idea is comparing input and output and in software engineering community it is used to describe the effectiveness of a developer's job. It is measured by, size over effort or more precisely lines-of-code over person-months. Although, so far nobody has talked about this concept as an attribute of software usability, it seems a quite explanatory measure for a user interface, since the user prefers to get more output while spending less input. Then it needs to work on the attribute "Productivity", to reach a proper definition and related metrics in usability engineering. It has to be done as a general concept and context oriented concept.

In MUSiC project [66], the same concept has been considered as User Efficiency. From a user's viewpoint, the amount of effort input may be quantified in terms of the time spent carrying out the task, or the mental/physical effort required to complete the task. The efficiency with which users make use of an IT product is defined in the MUSiC Performance Measurement Method as their Effectiveness in carrying out their task divided by the time it takes them to complete the task.

UE = Task Effectiveness / Task Time

Where Effectiveness is measured as described in appendix A.1. Task Time is the time spent by a user completing a task, being measured by the technique described in the Performance Measurement Handbook, or using GOMS models. GOMS model is described in previous chapters.

This metric provides a measure of User Efficiency in a particular context. It is task-specific. By itself, its value has little meaning unless there is another measure of efficiency against which to compare it.

For example, it can be used to compare the efficiency of two or more:

- similar products, or versions of a product, used in the same context, i.e. by the same user groups for the same tasks in the same environments
- types of users using the same product for the same tasks in the same environment
- ways of achieving given task goals when carried out by the same users on the same product in the same environment.

4.2.3 Developing a wizard for creating e-commerce web sites

One of the QUIM applications that we propose, is using the model for creating a wizard. This wizard could be designed for every knowledge domain, the purpose of the wizard is to make it easy to create an application in any specific domain for developers and software engineers who are not expert in Human-Computer Interaction. Even we may make a wizard usable by ordinary people. The example is a tool to help people to create their own web site. Today e-commerce or electronic commerce is getting more popular and the number of electronic shops and virtual malls is growing unbelievably. Unfortunately every body thinks that creating a web page is an easy job. Usually, people think if they know HTML flags, how to load an image file, and how to create a link to another page, it is enough to make a web site. As we have seen, e-commerce web sites disappear as fast as they appear. The problem is that, they can not attract users. One of the important reasons, is the usability of the site. Usually, those sites are developed by people who are not proficient in usability and Human-Computer Interaction. For sure, the user doesn't like to wait for loading huge images, or he/she doesn't like to be lost in an ocean of links. Then if we develop a wizard using attributes and metrics from QUIM, any individual who is not familiar with the rules of thumb in web development, can create his or her own web site. The wizard can provide goal values for metrics and put restrictions on developer's job. For example, it can restrict the number of links on a page, the number of images and the size of them on a page. It can control the length of text, and the dimensions of a page which is going to be appeared on the web browser. Even the wizard can help developer to create graphical icons in proper size with less number of colors. Using QUIM in creating wizards could be extended to the other domains.

References

Papers and Journals

[1]Bevan, N., Usability is quality of use. In: Anzai and Ogawa (eds) Proceedings of the 6th International Conference on Human Computer Interaction, Yokohama, July 1995 Elsevier.

[2]Boehm, B.W., Software Engineering Economics, Englewood Cliffs, New Jersey: Prentice Hall, 1981.

[3]ISO 9126: Software product evaluation - Quality characteristics and guidelines for their use, 1991.

[4]ISO/DIS 9241-11: Guidance on usability. Ergonomic requirements for office work with visual display terminals (VDT), 1996.

[5]ISO/DIS 13407 Standard on Human Centered Design Processes for Interactive Systems, 1998.

[6]Karat, C., Cost-justifying usability engineering in the software lifecycle. In Helander, M., Landauer, T., and Prabhu, P. (Eds), Handbook of Human-Computer Interaction. Elsevier Science, Amsterdam. 1997.

[7]Smith, S.L., Mosier, J.N., Guidelines for Designing User Interface Software, Rep. No. ESD-TR-86-278, Mitre Corporation, Bedford, MA, 1986.

[8]Landauer, T.K., The Trouble with Computers: Usefulness, Usability and Productivity. MIT Press, 1995.

[9]Martin, J. and McClure, C., Software Maintenance: The Problem and Its Solutions. Englewood Cliffs, New Jersey: Prentice Hall, 1983.

[10]Mayhew, D.J., The Usability Engineering Lifecycle: A Practitioner's Handbook

for User Interface Design, Morgan Kaufmanns Publishers, 1999.

[11]Norman, D.A., The Invisible Computer: Why Good Products Can Fail, the Personal Computer is so Complex, and Information Appliances are the Solution. MIT Press, 1998.

[12]Nielsen, J., Heuristic evaluation. In Nielsen, J., and Mack, R.L. (Eds.), Usability Inspection Methods. John Wiley & Sons, New York, NY., 1994.

[13]Seffah, A., Hayne, C., Workshop Conclusions on Integrating Human Factors in Use Case and OO Methods, 12th European Conference on Object-Oriented Programming. Lisbon, Portugal, June 14-20, 1999. LNCS 1743 SpringVerlag

[14]Lin, H.X., Choong, Y.Y., Salvendy, G., A proposed index of usability:A method for comparing the relative usability of different software systems usability evaluation methods, Behavior and Information Technology 1997 v.16 n.4/5 p.267-278.

[15]Garvin, D., "What does quality Really mean?, Sloan Management Review, Fall 1984, p. 25-45.

[16]Kitchenham, B., and Pfleeger, S., Software Quality:The Exclusive Target, IEEE Software, January 1996, p. 12-21

[17]Gillies, Alan C., Software Quality, Theory and Management, Chapman &Hall, 1992, p. 19-40

[18]Schulmeyer, G. and McManus, J., Handbook of Software Quality Assurance, Second Edition. NewYrok: Van Nostrand Reinhold.,1992.

[19]Nielsen, J., Usabilities Metrics and Methodologies, http://www.useit.com/papers/tripreports/bcs_metrics.html, 1990.

[20]Sears, Layout Appropriateness: A Metric for Evaluating User Interface Widget Layout, IEEE Transaction on Software Engineering, 1993.

[21]Nielsen, J., Usability metrics, 2001, http://www.useit.com/alertbox/20010121.html

[22]Constantine, Lockwood, Software for Use, Addison-Wesley, 1999.

[23]Yamada, S., Hong, J. K., Sugita, S., Development and Evaluation of Hypermedia for Museum Education: Validation of Metrics, ACM Transactions of Computer Human Interface, Vol2, No 4, 1995.

[24]Szejko, S., Applying HoQ to Software Development, Dept. of Applied Informatics, Gdansk Technical University, Poland, stasz@pg.gda.pl

[25]Gilb, T., "Level 6, Why we can't get there from here", IEEE Software, Vol 13, No. 1, pp.97-98, Jan. 1996.

[26]Ephraim P. Glinert, Nontextual Programming Environments, Principles of Visual Programming Systems, Shi-Kuo Chang, Editor, Prentice Hall Inc, pp. 195, 1990.

[27]Steven E. Keller, Laurence G. Kahn, Roger B. Panara, Specifying Software Quality Requirements with Metrics, System and Software Requirements Engineering, IEEE Computer Society Press Tutorial, pp 145-163, 1990.

[28]Sears, A., AIDE: A step toward metric-based interface development tools, UIST 1995 Pittsburgh PA USA, ACM 1995 0-89791-709-x/95/11.

[29]Tullis, T. Predicting the usability of alphanumeric displays, Ph.D. Dissertation, Dept. of Psychology, Houston, TX: Rice University, 46-61, 1984.

[30] Cavano, J. P., and J. A. McCall, "A Framework for the measurement of Software Quality," Proc. ACM Software Quality Assurance Workshop, pp. 133-139, Nov. 1978.

[31]Schneidewind, Norman F., "Methodology for Validating Software Metrics," IEEE Transactions on Software Engineering, vol 18, no 5, pp. 410-422, May 1992.

[32]Conover, W.J., Practical Nonparametric Statistics. New York: John Wiley & Sons, 1971.

[33]Gibbons, J.D., Nonparametric Statistical Inference, New York: McGraw-Hill, 1971.

[34]Kleinbaum, D.G., and Kupper, L.L., Applied Regression Analysis and Other Multivariable Methods. Boston: Duxbury Press, 1978.

[35]Bush, V. "As We May Think.", 1945. Reprinted in Interactions, pp. 35-67, March 1996.

[36] Brad A. Myers, A Brief history of Human Computer Interaction Technology, http://www.cs.cmu.edu/~amulet/papers/uihistory.tr.html

[37]Nelson, T., "A File Structure for the Complex, the Changing and the Intermediate," in Procceding ACM National Conference, pp. 84-100, 1965.

[38]E. Schonberg, T. Cofino, R. Hoch, M. Podlaseck, S. Spraragen, Measuring Success, Communication of the ACM, vol 43 No. 8, pp. 53-57, August 2000.

[39]Dix, Finlay, Abowd, Beale, Human Computer Interaction, 1993, http://www.dcs.napier.ac.uk/marble/Usability/UsabilityMetrics.html

[40] Norman, E., Fenton, S.L., Pfleeger, Software Metrics A Regorous and Practical Approach, International Thomson Publishing Company, 1997.

[41]Bevan, N., Measuring Usability as quality of use, Software Quality Journal, 4(2), pp. 115-30, 1995.

[42]IEEE Std. 1061, Software Quality Metrics Methodology, 1998

[43]Moor, J.W., Software Engineering Standards: A User's Road Map, IEEE Computer Society, 1998.

[44]Basili, V.R. and Weiss, D., "A Methodology for Collecting Valid Software Engineering Data", IEEE Transactions on Software Engineering, SE-10(6), pp. 728-738, 1984.

[45]Kitchemham, B.A. and Walker, J.G., "A quantitative approach to monitoring software development", Software Engineering Journal, 4(1), pp. 2-13, 1989.

[46]Daskalantonakis, M.K., A practical view of software measurement and implementaion experiences within Motorola, IEEE Transaction on Software Engineering, Vol. 18, No. 11, pp. 998-1010, Nov. 1992.

[47]Boehm, B.W., Software risk management: principles and practices, IEEE Software, Vol. 8, No. 1, pp. 32-41, Jan. 1991.

[48]Basili, V.R., Rombach, H.D., The TAME project: Towards improvement-oriented software environments, IEEE Transactions on Software Engineering, Vol. 14, No. 6, pp. 758-773, June 1988.

[49]Weinberg, G. M., Schulman, E. L., Goals and performance in computer programming, Human Factors, Vol. 16, No. 1, pp 53-65, 1974.

[50]Pfleeger, S.L., Fitzgerald, J.C., Rippy, D.A., Using multiple metrics for analysis of improvement, Software Quality Journal, Vol. 1, pp. 27-36, 1992.

[51]Morris, M.F. and Roth, P.F., Computer performance evaluation, Van nostrand, 1982.v

[52]Roger S. Pressman, Software Engineering A Practitionere's Approach, McGraw Hill, 1987.

[53]Brad A. Myers, A breif history of human computer interaction technology, http://www.cs.cmu.edu/~amulet/papers/uihistory.tr.html

[54]David E. Kieras, Scott D. Wood, Kasem Abotel, Anthony Hornof, GLEAN: A computer based tool for rapid GOMS model usability evaluation of user interface designs, ACM UIST, November 1995, pp 91-100, ACM 0-89791-709-x/95/11.

[55]Smith, S.L., Mosier, J.N., Guidelines for designing user interface software, Rep. No. ESD-TR-86-278, Mitre Corporation, Bedford, MA, 1986.

[56]Apple Computer Inc., Macintosh Human Interface Guidelines, Addison Wesley, Reading MA, 1992.

[57]Smith, S.L., Mosier, J.N., Application of guidelines for designing user interface, Software Behavior and Information Technology , No. 5, pp. 39-46, 1986.

[58]Bastien, J.M.C., Scapin, D.L., Leulier, C., The ergonomic criteria and the ISO/DIS 9241-10 dialogue principles: A pilot comparison in an evaluation task, Interacting with Computers, No. 11, pp. 299-322. 1999 .

[59]Bevan, N., Quality in use: Incorporating human factors into the software engineering lifecycle, National Physical Laboratory, UK, nigel@hci.npl.co.uk

[60]Jeffries R., Miller J.R., Wharton C., Uyeda K.M., User interfqce evaluation in the real world: A comparison of four techniques, Proceeding of ACM CHI 91, New Orleans, LA., 27 April 1991-2 May 1991, pp 119-124, http://www.csc.vill.edu/~beck/csc8570/jim1.htm

[61] Nielsen J., Usability Metrics -- How Good Are You? How to Measure Usability http://www.zdnet.com/devhead/stories/articles/0.4413.2321381.00.html

[62]Bevan, N., Quality and Usability: A new framework, National Physical Laboratory, UK, 1997, ftp://ftp.npl.co.uk/pub/hci/papers/qualusab.rtf

[63]Bevan, N., Usability issues in web site design, Proceeding of UPA'98, Washington DC, 1998, ftp://ftp.npl.co.uk/pub/hci/papers/Web-Usab.rtf

[64]Kirakowski, J., Barry, N., Usability assessment, version 1.2, November 1996, INUSE project, http://info.lboro.ac.uk/research/husat/inuse/assess.doc

[65]Macleod, M., Rengger, R., The development of DRUM: A software tool for video-assisted usability evaluation, ftp://ftp.npl.co.uk/pub/hci/papers/DRUM.rtf

[66]Macleod, M., Bowden, R., Bevan, N., The MUSiC performance Measurement Method, ftp://ftp.npl.co.uk/pub/hci/papers/PMM.rtf

[67]Erikkson I., McFadden F., Quality Function Deployment: A Tool to Improve Software Quality, Information and Software Technology, Vol. 35, No. 9, pp. 491-498, Sept. 1993.

[68]Schneidewind, N.F., Methodology for Validating Software Metrics, IEEE Trans. Software Eng., Vol. 18, No. 5, pp. 410-421, May 1992.

[69]Bevan, N., ISO Usability Standards, Human Factors 2000 Symposium, http://www.lboro.ac.uk/research/husat/news/prog/nbpresentation.html

[70] Common Industry Format for usability, Test Reports, Version 1.1, Oct. 1999, Industry USability Reporting project, http://www.nist.gov/iusr

Related Web Sites

[W1] http://www.useit.com/papers/tripreports/bcs_metrics.html

[W2] http://www.npl.co.uk/inuse

[W3] http://www.useit.com/alertbox/20010121.html

[W4] http://www/blender.nl

Appendices

A.1 General Metrics

Table 4: List of General Metrics

Metric	Definition
Essential Efficiency (EE) [22] This is a measure of how closely a given user interface design approximates the ideal expressed in the essential use case model.	EE=100 × S_essential / S_enacted S_essential = The number of user steps in the essential use case narrative S_enacted = The number of steps needed to perform the use case with the user interface design(rules for counting the number of enacted steps has come in the reference)

Weighted Essential Efficiency[22]
 	
Shows the overall efficiency of a design for an entire mix of tasks.
	EE_weighted = $\sum$ (Pi × EEi) $\forall$ i
	Pi=Probability (or weighted importance) of task I
	EEi=Essential efficiency for task i

Layout Appropriateness (LA) [20] This metric favors arrangements where visual components that are most frequently used in succession are closer together, reducing the expected time (cost) of completing a mix of tasks. Higher Layout Appropriateness means better usability.	LA = 100 × C_optimal / C_designed $C = \sum P_{i,j} \times D_{i,j} \ \forall i \neq j$ $P_{i,j}$ = Frequency of transition between visual components i and j $D_{i,j}$ = Distance between visual components i and j

Task Concordance (TC) [22]
 	
This is an index of how well the expected frequencies of tasks match their difficulty, good design will generally make the more frequent tasks easier (less steps or less efforts). To calculate this first we have to list all tasks ranked in order of descending expected frequency, along with their number of user steps in use case.
	TC = 100 × D / P
	P = N (N - 1) / 2
	N=The number of tasks being ranked
	D=Discordance score, i.e. The number of pairs of tasks whose difficulties are in right order minus those pairs whose difficulties are not in right order(One could find example for this calculation in the refernce)

Layout uniformity (LU)[22] Shows how well visual components of interface are arranged, This is a simplified version of Layout Complexity[19].	LU = 100 × (1-(Nh+Nw+Nt+Nl+Nb+Nr-M) / ((6 × Nc) - M)) $M = 2 + 2 \times \lceil \sqrt{Nc} \rceil$ Nc=The number of components Nh=the number of different heights Nw=The numbert of different widths Nt=The number of different top-edge alignments

	Nb= The number of different bottom-edge alignments Nl= The number of different left-edge alignments Nr= The number of different right-edge alignments
Task Visibility (TV)[22] Shows how many percent of necessary features (objects or elements) to complete a task or use case are visible to the user. It is reduced when we get unnecessary features on the user interface.	$TV = 100 \times (1 / S\_total \times \sum Vi) \quad \forall i$ S_total=Total number of enacted steps to complete the use case Vi=Feature visibility (0 or 1) of enacted step i (How to count enacted steps and allocate a visibility value to them is defined by some rules in the reference)
Visual Coherence (VC)[22] Shows how well a user interface keeps related components together and unrelated components apart.	$VC = 100 \times G / (N \times (N - 1) / 2)$ G=The number of related pairs in the group N=The number of visual components in the group Total Visual Coherence of a design for an interaction context could be computed by summing recursively over all the groups at each level of grouping: $VC = 100 \times (\sum Gk) / (\sum(Nk \times (Nk - 1) / 2)) \quad \forall k$ K is a group of component
Interface Shallowness[23] This metric is defined for hypermedia such as web sites, and is applicable to other graphical interface components such as menus as well. The indicator doesn't just show the depth of a node from root, rather than it indicates the degree of heaviness of the cognitive load on the user.	$ISh=(n \times (n - 1) \times (n - 1)) / (n \times (n - 1) - \sum IDpi)$ n=The number of nodes (it is greater than 1) IDpi=Sum of the values of IDs for the shortest path from root to node I IDsi,j=Interface distance (it is 0 if i and j are at the same level, it is 1 if i and j are at different levels)
Iconic Window Size (Wiconic) [26] Wiconic provides a measure of the order of the maximum number of icons that can be displayed on a screen at one time in a non overlapping manner. The goodness of this measure of course depend upon the degree of uniformity of icon size in the environment, and will be best when all icons are of equal size.	$Wiconic = (MD / MI)^{1+MA}$ MD=The diagonal measure, in centimeter or pixels, of the display screen used by the environment at hand MI=The diagonal measure of the icons used in the environment at hand, calculated in the same units as MD MA=The aspect ratio of the screen used in the environment at hand
Horizontal or Vertical Balance[28] The balance metrics evaluate how well balanced the screen is both vertically and horizontally. An optimal score for this metric is 100, and it means perfectly balanced.	$Balance = 200 \times W1 / (W1+W2)$ W1 = Weight of side one W2 = Weight of side two Weight of a side = Number of pixels used $\times$ side's distance from the center Center = Halfway between the left edge of the left-most visual element and the right edge of

	the right-most element
Task Effectiveness (TE) This metric comes from MUSiC project (Bevan, 1995). For example, suppose the desired goal is to transcribe a two-page document into a specified format. We could measure quantity as the proportion of transcribed words to original words, and quality as the proportion of non-deviations from the specified format. Another example, when testing a drawing package, the user might reproduce a picture composed of different shapes. The output is then analyzed to measure how many of the shapes the user reproduced, and how well those shapes match those in the original picture.	TE = Quantity × Quality / 100 Quantity = Percent of task completed Quality = Percent of goals achieved
Task Time [39]	Time to complete a specific task
Rate of default values (RDV)[Author]	RDV = N1 / N2 N1 = number of inputs with default value N2 = total number of inputs
Function Understandability (FU)[Author]	FU = N1 / N2 N1 = number of interface functions whose purposes are correctly described by user N2 = total number of functions available from interface
Rate of message font (RMF) [Author]	RMF = N1 / N2 N1 = number of fonts used in message boxes N2 = total number of message boxes
Rate of message box size (RMBS) [Author]	RMBS = N1 / N2 N1 = number of different sizes used for message boxes N2 = total number of message boxes
Flexibility Functions (FF) [Author]	The number of functions offered for changing the interface environment
Rate of wizards (RW) [Author]	RW = N1 / N2 N1 = number of provided wizards for complex operations N2 = total number of complex operations
Rate of cancel (RC) [Author]	RC = N1 / N2 N1 = number of provided cancel operation N2 = total number of operations
Task completion [39]	Percent of a task completed
Goal completion [Author]	Percent of goal achieved
Task help frequency [39]	Frequency of using help and documentation for a task

Task help time [39]	Time spent using help and documentation for a task
Number of acronyms [Author]	Number of acronyms used on the interface and are not found in a general dictionary
Number of icons [Author]	Number of different icons on the interface
Overall density [29]	The percentage of the display used to present information
Local density [29]	The percentage of the space used within each individual group of items
Number of items [29]	The number of individual items on the screen
Number of groups [29]	The number of groups of items on the screen
Number of commands [39]	The number of commands required to achieve a specific goal

A.2 Metrics for web

Table 5: List of metrics for web applications

Metrics	**Definition**
Number of colors [Author] This metric affects the load time	The number of colors used in every image
Number of characters [Author] This metric affects reading time and searching time over a page	The number of alphabetical characters appear on a page
Number of frames [Author] This metric affects the controlability of a page, because at the same time browser can show several pages in different frames	The number of frames on a view of the web site
Images size [Author] This metric affects loading time	The total size of all images on a page, measures in Kbytes
Longest depth [Author]	The number of links to the final page in a web site, in the other word the length of longest path in the web site
Page length [Author] This metric affects the amount of effort for scrolling and searching over a page	The length of a page in the web site

A.3 Potential relationships between Factors, Criteria and Metrics

Table 6: Potential relationships between Factors and Criteria

Factor	Criteria
Satisfaction	• User Guidance • Attractiveness • Flexibility • Understandability • Operability • Minimal Action • Minimal Memory Load
Efficiency	• Understandability • Operability • Resources • Minimal Action • Minimal Memory Load
Effectiveness	• Consistency • Completeness • Accuracy • Compliance

Table 7: Potential relationships between Criteria and Metrics

Criteria	Metrics
Understandability	• Frequency of using help and documentation (evaluation) • Time spent using help and documentation (prediction, evaluation) • Number of acronyms which are not found in a general purpose dictionary (prediction, evaluation) • Function understandability (evaluation)
Operability	• Number of commands (prediction,

	evaluation) • Interface shallowness (prediction, evaluation) • Number of frames on a web page (prediction, evaluation) • Rate of default values (prediction, evaluation)
Attractiveness	• Layout uniformity (prediction, evaluation) • Horizontal balance (prediction, evaluation) • Vertical balance (prediction, evaluation) • User subjective rating (evaluation)
Compliance	
Consistency	• Rate of message fonts (prediction, evaluation) • Rate of message box size (prediction, evaluation)
Flexibility	• Flexibility functions (prediction, evaluation)
Minimal action	• Task time (prediction, evaluation) • Number of commands (prediction, evaluation) • Task concordance (prediction, evaluation)
User guidance	• Rate of wizards (prediction, evaluation) • Rate of cancel (prediction, evaluation) • Visual coherence (prediction, evaluation)
Accuracy	• Task effectiveness (evaluation) • Goal compilation (evaluation)
Completeness	• Task completion (evaluation) • Task concordance (prediction, evaluation)
Resources	• Essential efficiency (prediction, evaluation) • Weighted essential efficiency (prediction, evaluation)
Minimal memory load	• Visual coherence (prediction, evaluation) • Layout uniformity (prediction, evaluation) • Interface shallowness (prediction,

	evaluation) • Number of icons (prediction, evaluation)

A.4 Specification of QUIM database and tool

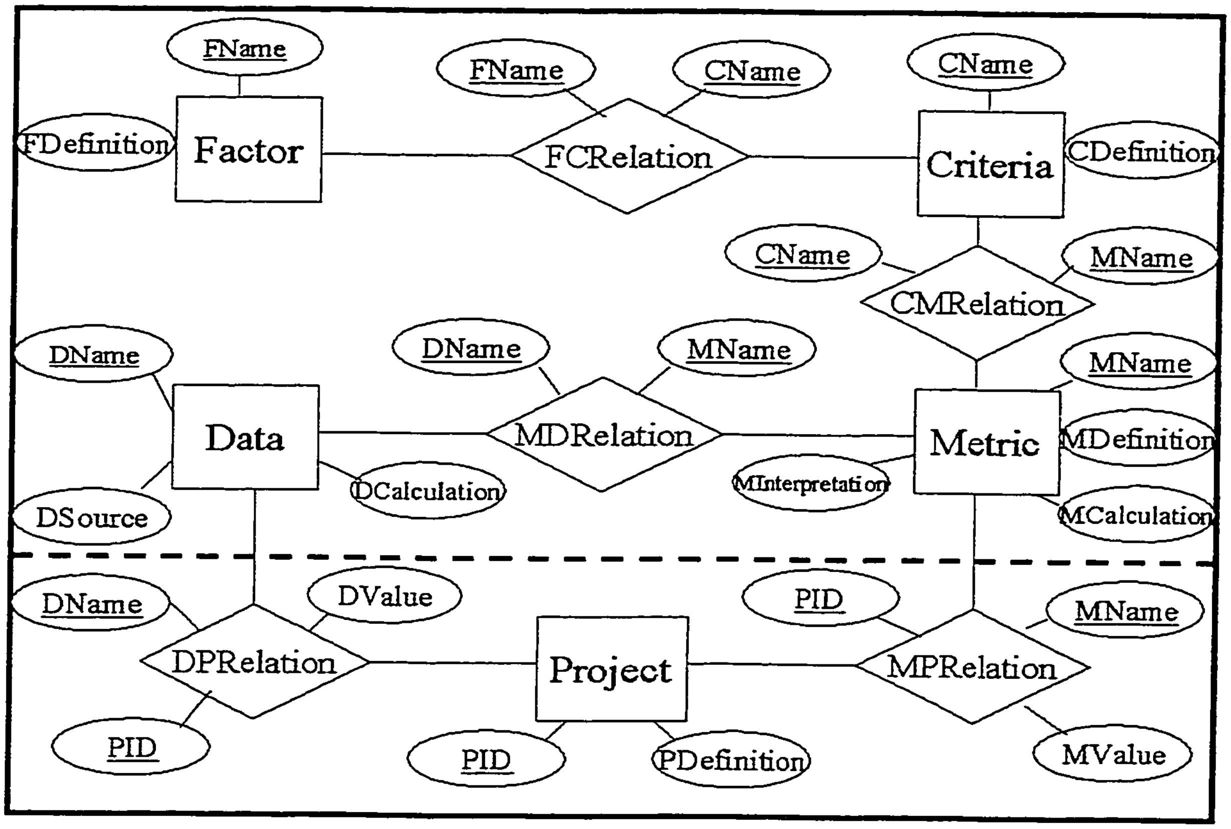


Figure 19: Entity-Relationship diagram for QUIM database