Chapter 2

Architecting for Usability

Published as: Architecting for Usability; a Survey, Eelke Folmer, Jan Bosch, Journal of Systems and Software, Issue 70-1, Pages 61-78, January 2004.

Abstract: Over the years the software engineering community has increasingly realized the important role software architecture plays in fulfilling the quality requirements of a system. The quality attributes of a software system are, to a large extent determined by the system's software architecture. In recent years, the software engineering community has developed various tools and techniques that allow for design for quality attributes, such as performance or maintainability, at the software architecture level. We believe this design approach can be applied not only to "traditional" quality attributes such as performance or maintainability but also to usability. This survey explores the feasibility of such a design approach. Current practice is surveyed from the perspective of a software architect. Are there any design methods that allow for design for usability at the architectural level? Are there any evaluation tools that allow assessment of architectures for their support of usability? What is usability? A framework is presented which visualizes these three research questions. Usability should drive design at all stages, but current usability engineering practice fails to fully achieve this goal. Our survey shows that there are no design techniques or assessment tools that allow for design for usability at the architectural level.

2.1 Introduction

In the last decades it has become clear that the most challenging task for a software architect is not just to design for the required functionality, but also focus on designing for specific attributes such as performance, security or maintainability, which contribute to the quality of software (Bosch and Bengtsson, 2002). Evaluating the quality of software is very important, not only from the perspective of a software engineer to determine the level of provided quality but also from a business point of view, such as when having to make a choice between two similar but competing products. To evaluate the quality of a software artifact, the way in which the software operates in its application domain has to be taken into account, rather than evaluate the software out of context. We believe usability is inherent to software quality because it expresses the relationship between the software and its application domain. Software is developed with a particular purpose, to provide specific functionality to allow a stakeholder to support a task in a specific context. Stakeholders such as users and the context in which they operate are an essential part of this application domain. Issues such as whether a product is easy to learn to use, whether it is responsive to the user and whether the user can efficiently complete tasks using it determines to a large extend a product's acceptance and success in the marketplace, apart from other factors such as marketing efforts and reputation. Software that provides much functionality but is awkward to use will not sell, nor will a product that provides little functionality but is usable. In general one can identify a trend towards an increasing focus on usability during software development.

One of the goals of software engineering is to construct computer systems that people find usable and will use (Ovaska, 1991). Usability engineering specifically focuses on this goal. Usability engineering is defined as according to the ACM definition: “Usability engineering, also known as human-computer interaction engineering, is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and the study of major phenomena surrounding them”. Within the software engineering community, usability engineering has become an established field of activity. Usability engineering has several benefits:

- Improve software: Business constraints such as time and money prevent the number of iterations that can be made in the design process. This constraint in iterations often leads to poor usability. User testing is often skipped when approaching deadlines. Proper usability engineering leads to software that is usable; which translates itself into productive and satisfied customers.

- Save customer’s money: usability engineering may not be directly beneficial for the developing organization. However from a customer’s point of view, working with a product which is easily understood, leads to increased productivity and requires less training costs. The effort spend on usability engineering eventually translates itself into a better reputation for the product, hence increasing sales.

- Minimize engineering costs: Studies of software engineering projects (Lederer and Prassad, 1992, Nielsen, 1993) show that most projects do not get their deadlines for delivery. The reasons that projects do not get their deadlines are often concerned with usability engineering: frequent requests for interface changes by users, overlooked tasks and so on. Proper usability engineering can reduce the cost overruns in software projects.

2.1.1 Current software still has low usability

Software development organizations pay an increasing attention to the usability of their software; however most modern day software still has low usability. This statement not only holds for public software such as word processors or email software but also for custom developed software, such as enterprise resource planning (ERP) or content management systems (CMS) software.

An example of bad usability is for instance the undo functionality in Framemaker. The undo function goes back only a few steps, therefore if one is used to working with Word or WordPerfect where the undo function can undo many steps, working with Framemaker can be frustrating if you have become used to that particular functionality. A possible reason for this case of bad usability is that in one of the first versions of Framemaker a choice was made to implement only a limited ability to record user’s steps. Imagine some sort of logging system that only keeps track of several steps, modifying the undo function to record all steps would make it more usable however this modification likely affects many parts of Framemaker source code which makes it expensive to implement. This example is only one of many cases of bad usability where usability is limited because it is too expensive to implement modifications that could improve usability. Several of such cases of bad usability exist; therefore we have reasons to believe that something fundamental goes wrong when designing for usability with current design approaches.

2.1.2 Traditional design approaches fail

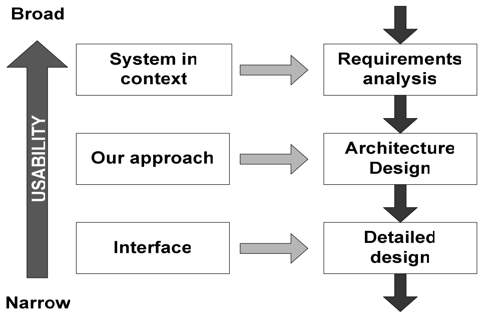


Figure 19: Approaches to Usability Engineering

The research community has developed numerous techniques and design methods such as design guidelines, design heuristics, interface standards and so on, to design software which is usable. Our survey of design techniques for usability as presented in section 2.4 has identified two approaches to usability engineering; distinct by their approach to the definition of usability. These approaches have been depicted in Figure 19. One of the first approaches towards usability engineering considered usability to be primarily a property of the presentation of information; the user interface. If an architecture that separates the user interface from the application is used, such as the model-view-controller architecture, usability can be ensured. If usability needs to be improved, changes to the interface can be easily applied after user testing, which does not affect the functionality of the application. This approach is considered to be naïve nowadays by the community. Most usability issues do not depend on the interface but on functionality, for example the undo function. The community that takes this approach is called the interface engineering community. The interface engineering community deals with usability at a detailed design level when this approach is related to the software design method. It has resulted in various interface standards and guidelines (Microsoft, 1992, Apple Company, 2004). Very detailed usability issues are suggested such as window layout, interface colors and semiotics of buttons and so on.

A different and broader approach towards usability engineering, as suggested by (Bevan, 1995) defines the usability of a software product to be a function of its interface as well as its functionality in a specified context. This approach is considered part of the requirement analysis phase. The focus lies on achieving the right functionality; enabling the user to perform specified goals in a specified context of use. Usability is evaluated by measuring user performance issues; the resources that have to be expended to achieve the intended goals and the extent to which the intended goals of use are achieved such as user performance issues and also the extent to which the user finds the use of the product acceptable, such as user satisfaction issues. The current definitions of usability are based on this broad approach. Usability is often defined according to how it should be measured is one of the conclusions of our survey of usability definitions in section 2.2.

In order to design for usability various sources such as interface guidelines, design heuristics or usability patterns, various design techniques such as prototyping or user/task modeling techniques may be consulted. These sources and techniques in combination with usability evaluation tools allow for design for usability. Design for usability in general can be characterized as an iterative design process. This approach has several shortcomings:

- Most usability issues are only discovered late in the development process, during testing and deployment. This late detection of usability issues is largely due to the fact that in order to do a usability evaluation, it is necessary to have both a working system and a representative set of users present. This evaluation can only be done at the end of the design process. It is therefore expensive to go back and make changes at this stage.

- Requirements change during or after development: it is almost always the case that during the development process, and even after a system is deployed, the requirements have changed. The context in which the user and the software operate is continually changing and evolving, which makes it hard to capture all possible future requirements at the outset. Sometimes users may find new uses for a product, for which the product was not originally intended. Design techniques such as task and user modeling used during requirements gathering can only partly model the future uses of the product.

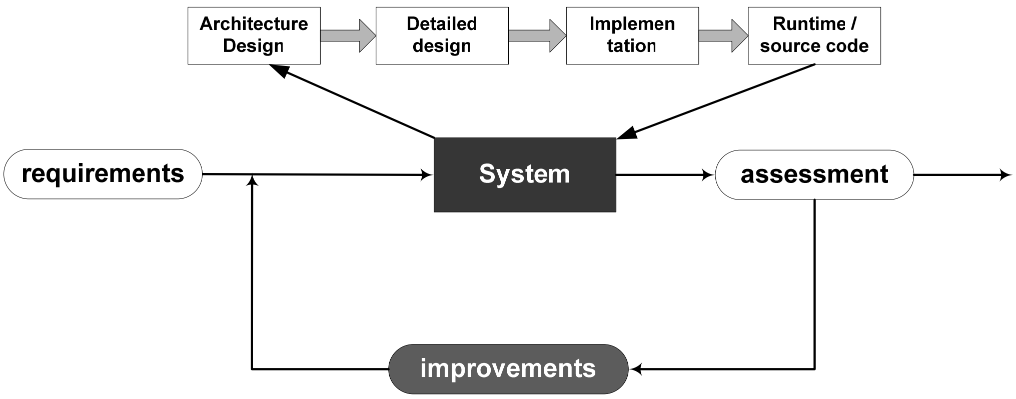


Figure 20: Current Design for Usability

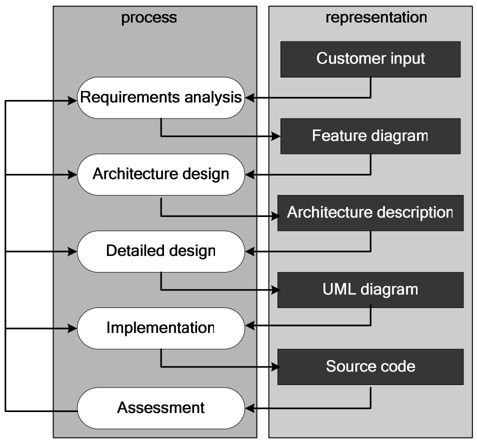


Figure 21: Waterfall Model and its Representations

Too often software systems prove to be inflexible for usability improving modifications. Small modifications which are at the detailed design level can be easily implemented but have little effect on usability. Modifications that have a substantial effect on usability are structural and therefore at the architectural level. However such modifications are too expensive to implement after implementation. Iteratively designing for usability, as depicted in Figure 20, is because of these shortcomings a relatively poor design method. This method only allows for usability improving modifications which are at the detailed design level.

Next to that limitation, iterative development prohibits the preservation of "design knowledge" For instance, some mistakes can be made over and over again without these being recorded an learned from when making a new design. The design knowledge that is captured in interface standards and guidelines provides only suggestions for low-level detailed design issues. The design knowledge captured in design heuristics does not translate itself to solutions that can be applied early on during design. Traditionally usability requirements have been specified such that these can be verified for an implemented system. However, such requirements are largely useless in a forward engineering process. Usability requirements need to take a more concrete form expressed in terms of the solution domain to influence architectural design.

Concluding, current design for usability does not lead to satisfactory usability results. We believe the software engineering community should adopt another approach towards designing for usability which is motivated by the following reasoning.

2.1.3 Software architecture restricts usability

Over the years it is noticed that besides an increasing focus on quality attributes, increasing attention is being paid to the architecture of a software system. Software is still increasing in size and complexity. An explicit defined architecture can be used as a tool to manage this size and complexity. Although there are many definitions of the term software architecture, one commonly accepted definition of software architecture is the following: "The software architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution (IEEE, 1998)".

Within the software engineering community it is commonly accepted that the quality attributes of a system such as modifiability or performance are to a large extent, constrained by its architecture. Our industrial experience leads us to believe that this constraint is also true for usability. In software engineering, an archetypal view of the software development process is called "waterfall model", also known as the systems development life cycle model. The waterfall development has distinct goals and deliverables for each phase of development. The waterfall model depicted in Figure 21 shows the largely linear structure of this process. In practice, the process adopted in software development is often far from linear. Steps of the design process are repeated in an iterative fashion. When it is realized that a part of the design needs to be changed to meet requirements, for example a modification to improve security, steps of the design process are repeated until the desired change has been effected. The goal of the software engineer is to keep the number of iterations to a minimum in order to minimize the engineering cost. The costs of reengineering rise with the level at which the changes are made. The further back in the process the designers have to go to make a change, the more it will cost (Brooks, 1995). For instance, changes at the detailed design level are less expensive to implement than changes at the architectural level. This change in costs has the following causes:

Modifications at the detailed design level can be realized by changes in the already existing source code. Such changes have only a small scale impact, often only at one variation point (Bosch, 2000) which only influences a module or an object and does not affect the software architecture. Thus by changing source code and corresponding design documentation, such as UML diagrams, these changes can be realized. Modifications at the architecture design level however, have structural impact. If some parts of the system have already been implemented at the time that changes are made, modification will likely affect many parts of the existing source code, which is very expensive to modify. Software engineers therefore aim to minimize the frequency of changes with architectural level impact.

In practice it is noticed that such 'architecture sensitive' changes are implemented, but business constraints cause such changes to be implemented in an ad-hoc fashion, rather than structurally. Such modifications erode original architectural design. An architectural design is based upon certain requirements. For those requirements the architectural design and source code that have been developed are optimal. If the architecture and source code are changed because of one structural modification, earlier design decisions may be invalidated and original design is eroded design (Gurp and Bosch, 2002). Next to taking care of how such modifications are implemented it should be realized that apart from the effect architectural changes have on each other, it is often unclear what effect single architectural design decisions have on the system and its quality attributes. Carelessly applying architectural modifications without taking into account earlier design decisions may have a dramatic effect on the quality attributes.
Experience (Häggander et al, 1999) and (Lassing et al, 2002a) shows that that improvement or design for quality attributes often requires the use of certain design patterns or styles. For instance, to improve portability and modifiability it may be beneficial to use a layered architecture style. It is our conjecture that a large number of issues associated to usability may also require architectural support in order to address them.

Because of the reasons discussed above, the software engineering community has realized the crucial role the software architecture plays in fulfilling the quality requirements of a system. Therefore it is of major importance that the quality requirements most central to the success of the software system should drive the design, especially at the software architecture level. In our opinion current design for usability fails to achieve this goal. The main problem is that systems prove to be inflexible. Certain usability improving modifications which are only discovered during deployment or after an initial design, or because usability requirements have changed during development, cannot be implemented. It is too expensive to implement such modifications because of their structural impact. Therefore being able to assess architectures during design for their support of usability could reveal those usability issues for which the system is not flexible.

Being able to iteratively design for and asses for usability at the architectural level, as depicted in Figure 23, improves the usability of systems, not only during design but also after implementation. To achieve this design approach, a different approach to usability engineering is required which is based on a general design approach for quality attributes.

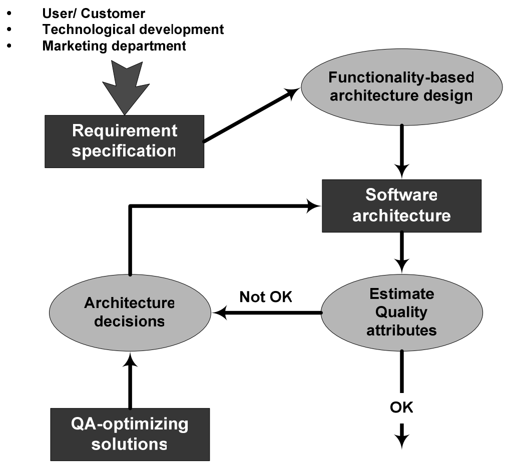

User/ Customer
Technological development
Marketing department
Requirement specification
Functionality-based architecture design
Software architecture
Architecture decisions
Not OK
Estimate Quality attributes
OK
QA-optimizing solutions

Figure 22: Software Architecture Design Method

2.1.4 Architecting for quality

The quality attributes of a software system are to a considerable extent defined by its software architecture. In addition, design decisions in the beginning of the design process are the hardest to revoke. Therefore it is important to have an explicit and objective design process. The software engineering research community has defined various software architecture analysis methods: SAAM (Kazman et al, 1994, Bass et al, 1998) ATAM (Kazman et al, 1998) and QASAR (Bosch, 2000). The latter, the Quality Attribute-oriented Software ARchitecture design method (QASAR), is a method for software architecture design that employs explicit assessment of, and design for the quality requirements of a software system. The architecture design process depicted in Figure 22 can be viewed as a function that transforms a requirement specification to an architectural design. The requirements are collected from the stakeholders; the users, customers, technological developments and the marketing departments. These groups often provide conflicting requirements and have to agree on a specific set of requirements before the design process can start. The design process starts with a design of the software architecture based on the functional requirements specified in the requirements specification. Although software engineers will not purposely design a system that is unreliable or performing badly, most non functional requirements are typically not explicitly defined at this stage.

The design process results in a preliminary version of the software architecture design. This design is evaluated with respect to the quality requirements by using a qualitative or quantitative assessment technique. After that the estimated quality attributes are compared to the values in the specification. If these are satisfactory, then the design process is finished. Otherwise, the architecture transformation or improvement stage is entered. This stage improves the software architecture by selecting appropriate quality attribute optimizing or improving design decisions.

When applying architecture design decisions, generally one or more quality attributes are improved whereas other attributes may be affected negatively. By applying one or more architectural design decisions, a new architectural design is created.

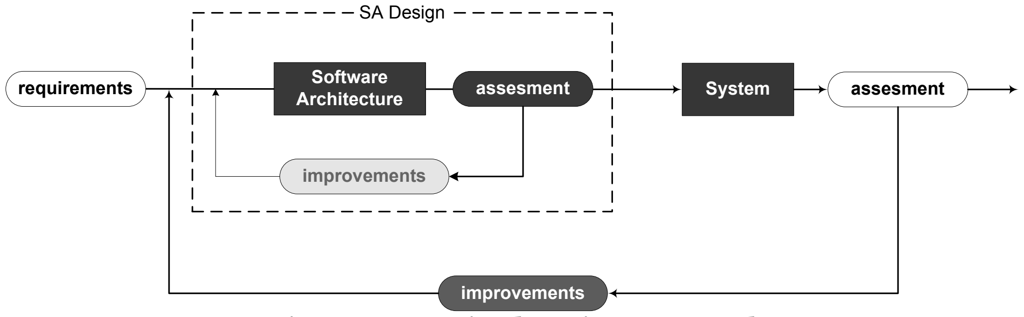


Figure 23: Desired Design Approach

This design is evaluated again and the same process is repeated, if necessary, until all non functional requirements have been satisfied as much as possible. Generally some compromises are necessary with respect to conflicting non functional requirements. This design process depends on two requirements:

- It is required to determine when the software design process is finished. Therefore, assessment techniques are needed to provide quantitative or qualitative data, to determine if our architecture meets the non functional requirements.

- Development or identification of architectural design decisions that improve usability.

Other analysis methods such as SAAM or ATAM take a similar approach with respect to iterative refinement of the design. Our goal is to use this approach to design for usability. This survey examines the feasibility of our design approach. The requirements for this design approach, such as being able to assess usability are surveyed in current practice. The design method presented is used as a reference point for surveying existing practice. Existing practice is thus surveyed from the perspective of a software architect. Our interest is focused on evaluation tools or design methods that allow design for usability at the architectural level.

Three research questions have been formulated that are surveyed in current practice and literature.

- How does current research community design for usability? Are there techniques that can be used for architectural design?

- How can software artifacts be assessed or evaluated for their support of usability? Are there techniques that can be used in our design approach?

- What is usability? Because assessing usability is closely related to how usability is defined, the different definitions of usability are surveyed to find out which definition suits our approach best.

Design for usability relies on being able to assess or evaluate usability. Most assessment techniques surveyed are based on specific definitions of usability. Being able to assess usability requires knowledge on how usability is defined. Therefore these questions are surveyed in current practice in reverse order and they are presented in the next sections.

The remainder of this paper is organized as follows. In section 2.2, the different definitions of usability are surveyed. Section 2.3 covers how usability can be assessed and in section 2.4 it is discussed, how current usability engineering community designs for usability. In section 2.5 a framework is presented that is composed from these three surveys. Finally in section 2.6 the issues that are raised in this survey are discussed and the paper is concluded in section 2.7.

2.2 What is Usability?

Usability has become an established field of activity in software development. Usability has, similar to many other software engineering terms, many definitions. The term usability was originally derived from the term "user friendly". But this term had acquired a host of undesirable vague and subjective connotations (Bevan et al, 1991) therefore the term "usability" was suggested to replace this term. Then again recently "usability" was defined as an attribute of software quality as well as a higher design objective. The term usability was replaced with the term "quality in use" (Bevan, 1995).

Although there is a consensus about the term usability, there are many different approaches to how usability should be measured; hence usability is defined in such a way as to allow these measurements. This definition has resulted in different definitions of usability, because authors have different opinions on how to measure usability. There are many definitions of usability (Shackel, 1991, Hix and Hartson, 1993, Preece et al, 1994, ISO 9241-11, Wixon and Wilson, 1997, Shneiderman, 1998, Constantine and Lockwood, 1999, ISO 9126-1). Although not all authors call the entities, which to them compose usability, usability attributes. Sometimes these entities are defined as dimensions, components, scales or factors of usability. It is our opinion that they mean the same and therefore the term usability attributes is used, which is the term most commonly used.

In our opinion, in usability research, authors spent much effort trying to find the best way to define usability by defining attributes that can be measured and compose usability. In this survey, finding or giving the best or an exact definition of usability is not the goal. Our interest in the concept of usability reaches as far as whether it will be applicable in the context of a design method. Our survey is therefore restricted to discuss in detail only four approaches have been most widely recognized and used in practice. In our opinion other definitions are strongly related to these significant existing ones. The next subsections will discuss four approaches to usability; appearing in chronological order of initial work on the subject published by the author(s).

2.2.1 Shackel

One of the first authors in the field to recognize the importance of usability engineering and the relativity of the concept of usability was (Shackel, 1991). His approach to usability has been much used and modified. Shackel defines a model where product acceptance is the highest concept. The user has to make a trade-off between utility, the match between user needs and functionality, usability, ability to utilize functionality in practice and likeability, affective evaluation versus costs; financial costs as well as social and organizational consequences when buying a product. Usability is defined as: "the usability of a system is the capability in human functional terms to be used easily and effectively by the specified range of users, given specified training and user support, to fulfill the specified range of tasks, within the specified range of scenarios". Shackel considers usability to have two sides:

- Usability is a relative property of the system; being relative in relation to its users, therefore evaluation is context dependent; resulting in a subjective perception of the product.

- The other side of usability relates to objective measures of interaction.

Shackel does not explicitly define how to measure both sides, but proposes to measure usability by its operational criteria, on four dimensions.

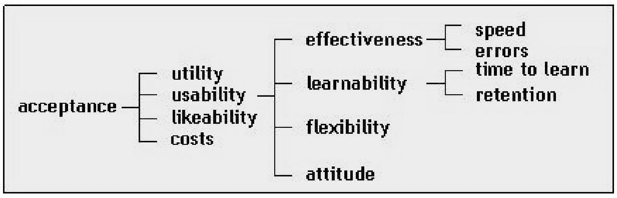


Figure 24: Shackel's Definition of Usability

For a system to be usable it has to achieve defined levels on the following scales:

- Effectiveness: performance in accomplishment of tasks.

- Learnability: degree of learning to accomplish tasks.

- Flexibility: adaptation to variation in tasks.

- Attitude: user satisfaction with the system.

Figure 24 shows the usability concepts defined by Shackel. Shackel provides a descriptive definition of the concept of usability that refers to the complex framework of evaluation and suggests concrete measurable usability criteria.

2.2.2 Nielsen

Another pioneer in the field of usability that recognized the importance of usability engineering was Nielsen. (Nielsen, 1993) just as in the case of Shackel, considers usability to be an aspect that influences product acceptance. Acceptability is differentiated into practical and social acceptability as depicted in Figure 25. Usability and utility; the ability to help the user carry out a set of tasks, together form the usefulness of a system.

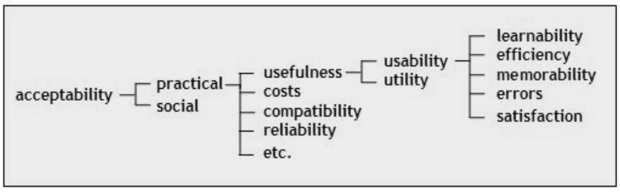


Figure 25: Nielsen's Definition of Usability

Nielsen defines usability to consist of five kinds of attributes:

- Learnability: systems should be easy to learn. Users can rapidly start getting some work done with the system.

- Efficiency: Systems should be efficient to use. When a user has fully learned the system, productivity will be possible on a high level.

- Memorability: Systems should be easy to remember, making it possible for casual users to return to the system after some period of not using the system, without having to learn everything all over again.

- Errors: The system should have a low error rate, which enables users to make few errors during the use of the system. When they do make errors they can easily recover from them. Catastrophic errors should not occur.

- Satisfaction: The system should be pleasant to use; which makes users subjectively satisfied when using it.

Nielsen does not give a precise definition of usability, but presents the operational criteria that clearly define the concept.

2.2.3 ISO 9241-11

The ISO organization has developed various HCI and usability standards over the last 15 years. The function of these ISO standards is to provide and impose consistency. ISO standards for interface components such as icons, cursor control and so on, have not been widely adopted. Industry standards such as IBM, Macintosh or Windows have been more successful in that area. ISO standards (ISO 9241-11) on ergonomic requirements such as VDT workstations, hardware and environment, on the other hand, have been widely adopted by industry. These standards have led to guidelines for software interfaces and interaction based on research done by (Macleod, 1994) and (Bevan, 1995). (ISO 9241-11) provides the definition of usability that is used most often in ergonomic standards. Usability is defined as: "the extent to which a product can be used by specified users to achieve specified goals with 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, in a specified context of use".

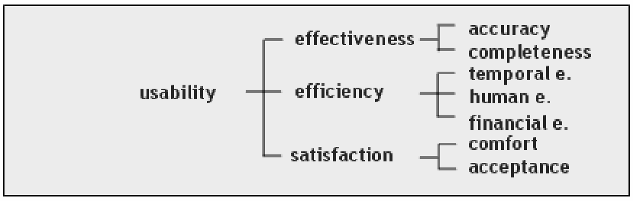


Figure 26: ISO 9241-11 Definition of Usability

According to ISO 9241-11 the attributes of usability are:

- Effectiveness: the accuracy and completeness with which users achieve specified goals.

- Efficiency: the resources expended in relation to the accuracy and completeness with which users achieve goals.

- Satisfaction: the comfort and acceptability of use.

- ISO 9241-11 presents a contextually oriented view of usability. This definition incorporates a user performance view; issues such as effectiveness and efficiency and a user view; issues such as satisfaction. Standard ISO 9241-11 explains how to identify the information which is necessary to take into account when specifying or evaluating usability in terms of measures of user performance and satisfaction.

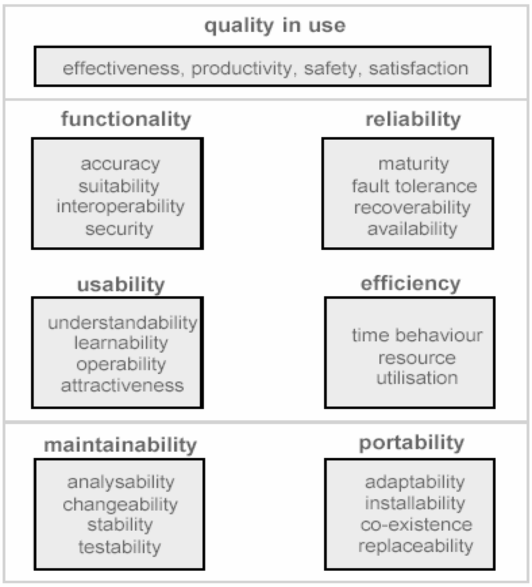


Figure 27: ISO 9126-1 Quality Model

Guidance is given on how to describe the context of use of the product; such as hardware, software or service and the required measures of usability in an explicit way. It includes an explanation of how the usability of a product can be specified and evaluated as part of a quality system, for instance, one that conforms to ISO 9001 standards. It also explains how measures of user performance and satisfaction can be used to measure how any component of a work system affects the quality of the whole work system in use. The standards and the evaluation tools that result from it have been widely adopted by HCI practitioners.

2.2.4 ISO 9126

The software engineering community has always associated usability with interface design. ISO 9126 used to define usability as a relatively independent contribution to software quality associated with the design and evaluation of the user interface and interaction. Usability is defined as: “a set of attributes of software which bear on the effort needed for use and on the individual assessment of such use by a stated or implied set of users”.

This view has changed because new insights led to another approach to usability. (ISO 9241-11) defines a quality model that describes six categories of software quality that are relevant during product development: functionality, reliability, usability, efficiency, maintainability and portability, as depicted in Figure 27
Usability plays two roles in this model:

- Product oriented role: usability is part of a detailed software design activity; it is a component of software quality as defined in ISO 9126.

- Process oriented role: usability provides the final goal; it is a design objective; the software should meet user needs as defined in ISO 9241.

The latter objective is defined by the term “quality in use”. This term is synonymous with the broad definition of usability.

Quality in use is the combined effect of the softwarequality characteristics for the end user (Bevan et al, 1991). Quality in use is defined as: ‘the capability of the software product to enable specified users to achieve specified goals with effectiveness, productivity, safety and satisfaction in a specified context of use”. This definition is similar to how quality of use is defined in ISO 9241-11 except that it adds safety. In ergonomic standards, health and safety is treated separately. It is important to notice that a product has no intrinsic usability of itself only a capability to be used in a particular context. Usability is therefore defined as: “the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions”. The two ISO definitions of usability are complementary (Bevan, 2001). Both standards define usability in measurable design objectives. Software engineers generally use the product-oriented approach to usability in the design of appropriate product attributes, as recommended in ISO 9241 parts 12-17 or specified as usability metrics in ISO 9126 parts 2 and 3. During development of a software product these two approaches to usability need to be combined, the broad goal of quality in use is needed to support user-centered design, while detailed concern with the interface is necessary during development. ISO 9126-1 specifies usability by the following measurable attributes:

- Understandability: The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use.

- Learnability: The capability of the software product to enable the user to learn its application.

- Operability: The capability of the software product to enable the user to operate and control it.

- Attractiveness: The capability of the software product to be attractive to the user. For instance the use of colors or nature of graphical design.

2.2.5 Overview

The different definitions of usability have been discussed to understand where usability evaluation tools and methods, as will be discussed in the next section, are based on. An overview of definitions is provided in Table 6. From this survey the following conclusions can be made. The usability attributes can be divided into:

- Objective operational criteria: user performance attributes such as efficiency and learnability.
- Subjective operational criteria; user view attributes such as satisfaction and attractiveness.

The term usability attribute is quite ambiguous. Authors in the field of usability have quite different perceptions of what they consider to be a useful usability attribute. The approaches discussed are widely adopted by the usability engineering community but seem to coexist without interference. In our opinion the different definitions, not only those that are discussed but also other definitions stated earlier on, largely overlap. Differences include:

- Attribute names, some authors use different names for the same attribute such as memorability and learnability.
- Authors have different opinions on what they consider to be a useful usability attribute. Learnability for instance is only recognized by ISO 9126 standard.
- Authors use different ways of combining attributes which compose usability. For example, in Nielsen's definition errors is part of usability, but in ISO 9126 errors is part of efficiency which composes usability.

From Table 6 it can be concluded that the different definitions of usability overlap. However, care has to be taken that even if there seems to be a considerable amount of overlap, some attributes differ when examined what is actually measured for that attribute on a lower level. On the other hand there are also similarities on a lower level for attributes that seem different. For example, the number of errors made during a task, or the time to learn a task, are measurable indicators for the learnability attribute. On the other hand, the number of errors made during a task is an indication to Nielsen's errors attribute but also to ISO 9216's efficiency attribute. The errors attribute and efficiency attribute are therefore closely related, although authors put them at different places in the hierarchy of usability composition. Further investigation on what exactly is measured for each attribute is therefore required.

The framework as presented in section 2.5 provides the necessary categorization of usability engineering. This framework is a means to categorize and organize the different definitions of usability and visualizes the differences and similarities between the different usability definitions.

Table 6: Overview of Usability Definitions

	Shackel, 1991	Nielsen, 1993	ISO 9241-11	ISO 9126
User Performance	Learnability-time to learn	Learnability		Learnability
	Learnability-retention	Memorability		
(objective)	Effectiveness-errors	Errors	Effectiveness	
	Effectiveness- task time	Efficiency	Efficiency	
				Operability

				Understandability
	Flexibility			
User view (subjective)	Attitude	Satisfaction	Satisfaction	Attractiveness

Relating these different definitions of usability to our design approach for usability, the following conclusions can be made:

- In usability research authors spent a considerable amount of effort trying to find the best way to define usability by defining attributes that can be measured and compose usability. For our design approach the definition of usability is not an issue, the choice of whether to use a particular definition will depend on how well an evaluation tool based upon this definition, will support evaluation of usability at the architectural level as required in our design process.

- ISO 9126 standard is the only approach to usability that recognizes usability to be a quality attribute of a product that is also influenced by other quality attributes, which is inline with our assumptions about usability being also influenced by other quality attributes.

Next section will continue the analysis of our usability engineering approach by discussing usability evaluation tools and techniques that have been developed by usability engineering community.

2.3 Evaluating Usability

Many evaluation tools and techniques which are surveyed in this section are based upon specific definitions of usability. The previous section has surveyed various definitions of usability, which provides the necessary background for discussing evaluation techniques that are based upon such definitions. (Zhang, 2001) has identified three types of usability evaluation methods

- Testing
- Inspection
- Inquiry

The next subsections will present an overview of evaluation tools and techniques for each type of evaluation method.

2.3.1 Usability testing

The usability testing approach requires representative users to work on typical tasks using the system or the prototype. Prototyping models final products and allows testing of the attributes of the final product even if it is not ready yet, simply the model is tested. The evaluators use the results to see how the user interface supports the users to do their tasks. Testing methods include the following:

- Coaching Method (Nielsen, 1993)

- Co-discovery Learning (Dumas and Redish, 1993, Nielsen, 1993, Rubin, 1994)
- Performance Measurement (Soken et al, 1993, Nielsen, 1993)
- Question-asking Protocol (Dumas and Redish, 1993)
- Remote Testing (Hartson et al, 1996)
- Retrospective Testing (Nielsen, 1993)
- Teaching Method (Vora and Helander, 1995)
- Thinking Aloud Protocol (Nielsen, 1993)

2.3.2 Usability inspection

The Usability Inspection approach requires usability specialists or software developers, users and other professionals to examine and judge whether each element of a user interface or prototype follows established usability principles. Commonly used inspection methods are:

- Heuristic Evaluation (Nielsen, 1994)
- Cognitive Walkthrough (Rowley and Rhoades, 1992, Wharton et al, 1994)
- Feature Inspection (Nielsen, 1994)
- Pluralistic Walkthrough (Bias, 1994)
- Perspective-based Inspection (Zhang et al, 1998b, Zhang et al, 1998a)
- Standards inspection/guideline checklists (Wixon et al, 1994)

2.3.3 Usability inquiry

Usability inquiry requires usability evaluators to obtain information about users likes, dislikes, needs and understanding of the system by talking to them, observing them using the system in real work (not for the purpose of usability testing) or letting them answer questions verbally or in written form. Inquiry methods include:

- Field Observation (Nielsen, 1993)
- Interviews / Focus groups (Nielsen, 1993)
- Surveys (Alreck and Settle, 1994)
- Logging Actual Use (Nielsen, 1993)
- Proactive Field Study (Nielsen, 1993)

Another inquiry method that is widely used at usability evaluation are questionnaires. (Zhang) and various other web resources provide an overview of web based interface evaluation questionnaires:

- QUIS: Questionnaire for User Interface Satisfaction (Chin et al, 1988)
- PUEU: Perceived Usefulness and Ease of Use (Davis, 1989)
- NHE: Nielsen's heuristic evaluation (Nielsen, 1993)
- NAU: Nielsen's attributes of usability (Nielsen, 1993)
- PSSUQ: Post Study System Usability Questionnaire (Lewis, 1992)
- CSUQ: Computer System Usability Questionnaire (Lewis, 1995)
- ASQ: After Scenario Questionnaire (Lewis, 1995)
- SUMI: Software Usability Measurement Inventory (HFRG)
- MUMMS: Measurement of Usability of Multi Media Software (HFRG)
- WAMMI: Website Analysis and Measurement Inventory (HFRG)
- EUCSI: End user satisfaction instrument (Doll et al, 1994)

2.3.4 Overview

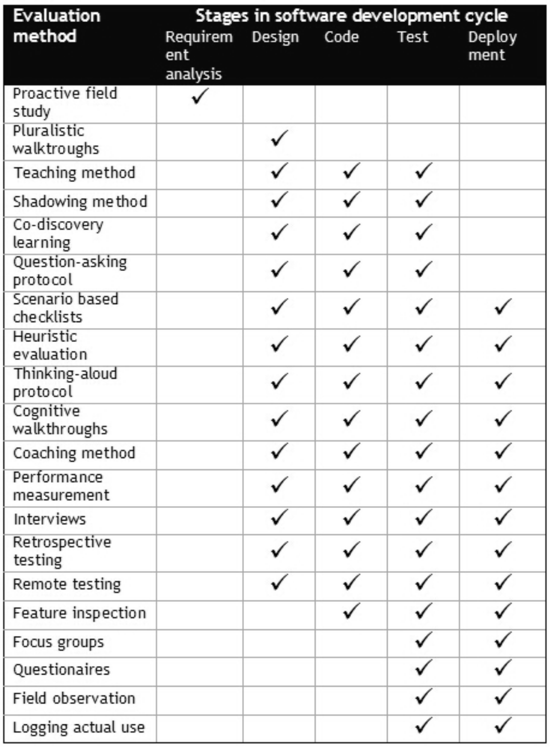
A wide variety of usability evaluation tools is available. Our design approach requires a specific assessment technique to assess architectures for their support of usability during the architectural design phase. Figure 28 gives an overview of different techniques discussed in this section and at which stages in the software development cycle they can be applied. Though several techniques can be used during design, there are no techniques that can be used during architectural design phase. The techniques discussed as usability inspection and usability testing techniques all require a user interface or a prototype of an interface available for evaluation. Usability inquiry focuses on evaluation of usability of real life systems. Most of these techniques evaluate the system for usability requirements/specifications that can actually be measured for complete systems. Such evaluation methods are quite useless when designing a new system from scratch. During architecture design phase a prototype of an interface is not present or it is too expensive to develop one. Furthermore we believe that most usability issues do not depend on the interface but on functionality, for example the undo functionality. Therefore interface or system based evaluation techniques as presented in this section are not useful for our design approach.

The only thing available for evaluation during architectural design is a first version of the software architecture. Assessment techniques should focus on assessing the architecture instead of the interface or the system. Based on our experience is our expectation that development of a checklist or heuristic based approach where one identifies architectural components that support usability, will lead to the desired design approach. Next section will continue the survey by discussing different usability design techniques.

2.4 Design for Usability

There are two approaches to designing for usability as identified by (Keinonen, 1998).

- Process oriented approach; user-centered design.
- Product oriented approach; captured design knowledge.



Evaluation method	Stages in software development cycle				
	Requirement analysis	Design	Code	Test	Deployment
Proactive field study	✓				
Pluralistic walktroughs		✓			
Teaching method		✓	✓	✓	
Shadowing method		✓	✓	✓	
Co-discovery learning		✓	✓	✓	
Question-asking protocol		✓	✓	✓	
Scenario based checklists		✓	✓	✓	✓
Heuristic evaluation		✓	✓	✓	✓
Thinking-aloud protocol		✓	✓	✓	✓
Cognitive walkthroughs		✓	✓	✓	✓
Coaching method		✓	✓	✓	✓
Performance measurement		✓	✓	✓	✓
Interviews		✓	✓	✓	✓
Retrospective testing		✓	✓	✓	✓
Remote testing		✓	✓	✓	✓
Feature inspection			✓	✓	✓
Focus groups				✓	✓
Questionaires				✓	✓
Field observation				✓	✓
Logging actual use				✓	✓

Figure 28: Overview of Evaluation Methods

2.4.1 Process oriented

User-centered design is a process oriented approach towards design for usability; usability is considered to be a design goal. It is a collection of techniques that specifically focuses on providing and collecting that functionality that makes software usable. They are closely related with usability evaluation principles and techniques discussed in section 2.3. The whole process of design for usability, user testing, and redesign is called user-centered design. This view is very important in participatory design. One of its major benefits is that it ties users to the process and lowers their resistance towards change in organizations. Within user-centered design, numerous techniques are used, such as: brainstorming, task analysis, direct observation, questionnaire surveys, interviews, focus groups, user panels, empathic modeling, scenario modeling, task modeling, user modeling, prototyping, contextual enquiry, usability laboratories, user trials, field trials, discount usability engineering, co-operative evaluation, cognitive walkthroughs and so on. Some of these techniques have been surveyed in section 2.3. Specific user-centered design methods offer a collection of these techniques, often including some sort of user modeling technique and an evaluation technique that allow us to design for usability. Some examples of user-centered design suites:

- Discount Usability Engineering (Nielsen, 1995)
- IBM User-centered design process (Vredenburg et al, 2001)
- USER fit (Poulson, 1996)

2.4.2 Product oriented

The product oriented approach considers usability to be a product attribute by naming examples of product or system properties or qualities that influence usability. This approach has collected and described design knowledge over many years of software design. The design knowledge consists of a collection of properties and qualities that have proven to have a positive effect on usability. This approach can be divided into three categories:

- Interface guidelines.
- Design- heuristics and principles.
- Usability patterns.

2.4.3 Interface guidelines

These guidelines provide suggestions and recommendations for low level interface components, for example: directions and guidelines for icons, windows, panels, buttons, fields and so on.

- IBM CUA (IBM, 1991b, IBM, 1991a), Guide to user interface design.
- Windows (Microsoft, 1992) The Windows interface - An application design guide.
- ISO 9241-14 (ISO 9241) Menu dialogues. This part provides recommendations for the ergonomic design of menus used in user-computer dialogues.
- ISO/IEC 11581: Icon symbols and functions. Contains a framework for the development and design of icons, including general requirements and recommendations applicable to all icons.
- KDE user interface guidelines. (KDE, 2001)
- Macintosh human interface guidelines (Apple Company, 2004)

These and various other guidelines provide the raw material for an interface. Usability depends on the extent to which a dialogue implemented in a particular style is successful in supporting the user's task.

2.4.4 Design heuristics and principles

Design heuristics and principles for usability suggest properties and principles that have a positive effect on usability. The following list of design heuristics and principles is created based upon surveys provided in (Baecker et al, 1995, Keinonen, 1998).

- Eight golden rules of dialogue design (Shneiderman, 1986)
- Usability heuristics (Nielsen, 1993)
- Usability principles (Constantine and Lockwood, 1999)
- Evaluation check list for software inspection (Ravden and Johnson, 1989)
- Guidelines on user interaction design (Hix and Hartson, 1993)
- Seven principles that make difficult task easy (Norman, 1988)
- Design for successful guessing (Polson and Lewis, 1990)
- Dialogue principles (ISO 9241)
- Design for successful guessing (Holcomb and Tharp, 1991)
- Design principles (Rubinstein and Hersh, 1984)

The principles stated above almost all address usability issues mentioned below according to (Keinonen, 1998)

- Consistency; users should not have to wonder whether different words, situations, or actions mean the same thing. It is regarded as an essential design principle that consistency should be used within applications. Consistency makes learning easier because things have be learned only once. The next time the same thing is faced in another application, it is familiar. Visual consistency increases perceived stability which increases user confidence in different new environments.

- Task match; designers should provide just the information that the users needs no more no less, and in the order that the users prefers to use this information.

- Appropriate visual presentation; user interface design has focused on this aspect of user control. This issue has recently been extended to include multimedia, for example, voice control applications. For a user to be effectively in control he has to be provided with all necessary information.

- User control; it is a design principle that direct manipulation should be supported, for instance, the user should feel that he is in control of the application. Interaction is more rewarding if the users feel that they directly influence the objects instead of just giving the system instructions to act.

- Memory-load reduction; People do not remember unrelated pieces of information exactly, thus where precise recollection is required; for instance in a task, many errors may be expected. Interaction therefore should rely more on user recognition than on recall. Recall is prone to errors, whereas people are very good at recognizing objects. The allocation of work between humans and computers should be such that computers present alternatives and patterns, while people select and edit.

- Error handling; all usability principles address the issue of error handling or error recovery. Error recovery relieves anxiety, enabling users to discover new alternatives, facilitating learning by doing.
- Guidance and support; In order to help the user understand and use the system, informative, easy to use and relevant guidance and support should be provided in both the application and the user manual.

2.4.5 Usability patterns

Patterns and pattern languages for describing patterns are ways to describe best practices, good designs, and capture experience in a way that it is possible for others to reuse this experience. Although patterns originate from software development, they can be used for any design including user interface design. According to (Alexander et al, 1977) patterns are defined as: "each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution."

A usability pattern is not the same as a design pattern known in software engineering such as discussed by (Gamma et al 1995). Design patterns specifically focus on implementation details and its effect on particular quality attributes whereas usability patterns refrain from specifying implementation details and only state a specific relation with usability. One thing that usability patterns share with design patterns is that their goal is to capture design experience in a form that can be effectively reused by software designers in order to improve the usability of their software, without having to address each problem from scratch. The aim is to take what was previously very much the "art" of designing usable software and turn it into a repeatable engineering process. Another aspect shared with design patterns is that a usability pattern should not be the solution to a specific problem, but should be able to be applied in order to solve a number of related problems in a number of different systems, in accordance with the principle of software reuse. Various usability patterns collection have been described by (Tidwell 1998, Perzel and Kane 1999, Welie and Trætteberg, 2000)(Perzel and Kane, 1999).

Some collections of usability patterns can be found on the internet:

- Common ground: Tidwell's usability pattern collection (Common ground, 1999)
- The Amsterdam Collection of Patterns in User Interface Design (Welie, 2003)
- PoInter (Patterns of INTERaction) collection at Lancaster University (PoInter, 2003)
- The Brighton Usability Pattern Collection (Brighton, 1998)

Most of these usability patterns collections refrain from providing implementation details. (Bass et al, 2001) on the other hand take a software architecture centered approach to usability engineering. Usability is approached from a software engineering perspective. (Bass et al, 2001) give examples of architectural patterns that may aid usability. Several scenarios have been identified that illustrate particular aspects of usability that are architecture sensitive. Several architectural patterns are presented for implementing these aspects of usability.

2.4.6 Overview

Various guidelines and heuristics that have been surveyed can be integrated into traditional iterative development techniques for software development; however, there are no specific design techniques for usability that allow design for usability at the architectural level. There are techniques that allow us to collect those requirements that make software more usable, but there is no explicit process that translates these requirements into specific architectural solutions.

The current approach for designing for usability gives either very detailed design directions; in the interface guidelines, such as suggesting layout of icons and so on, or provides a wide variety of usability principles. These principles are very useful but are typically hard to correlate to the software architecture. Specifically, part of these usability issues such as appropriate visual presentation address the field of interface engineering but large part addresses system engineering such as user control, error handling and so on.
The following questions cannot be easily answered. When designing for usability which architectural choices have to be made? Or which design choices should a software architect consider when designing for usability? There is no apparent relationship between heuristics and architectural solutions yet.

The usability engineering community has collected and developed various design solutions such as usability patterns that can be applied to improve usability. Where these prescribe sequences or styles of interaction between the system and the user, they are likely to have architectural implications.

(Bass et al, 2001) have identified patterns that to them require architectural support. To our opinion usability patterns can be implemented quite differently, influencing architectural sensitiveness, therefore the architectural sensitiveness of several patterns is open to dispute. Next to that some of the scenarios suggested by them are open to dispute whether they are related to usability, in our opinion.

Evaluation of or designing for usability on the architectural level as suggested by (Bass et al, 2001) appears to be just running down a checklist of scenarios that (may) require architectural support. This approach is not an ideal situation because there is not a clearly documented and illustrated relationship between those usability issues addressed by the design principles and the software architecture design decisions required to design for usability. To improve on this situation, it would be beneficial for knowledge pertaining to usability to be captured in a form that can be used to inform architectural design, which allows for engineering for usability early on in the design process.

2.5 Usability Framework

To conclude this survey a framework has been constructed which visualizes the three research questions surveyed in previous sections. Before investigating the feasibility of our design approach the different definitions of usability were surveyed in existing literature. Our survey in section 2.2 discovered that the term usability attribute is quite ambiguous.

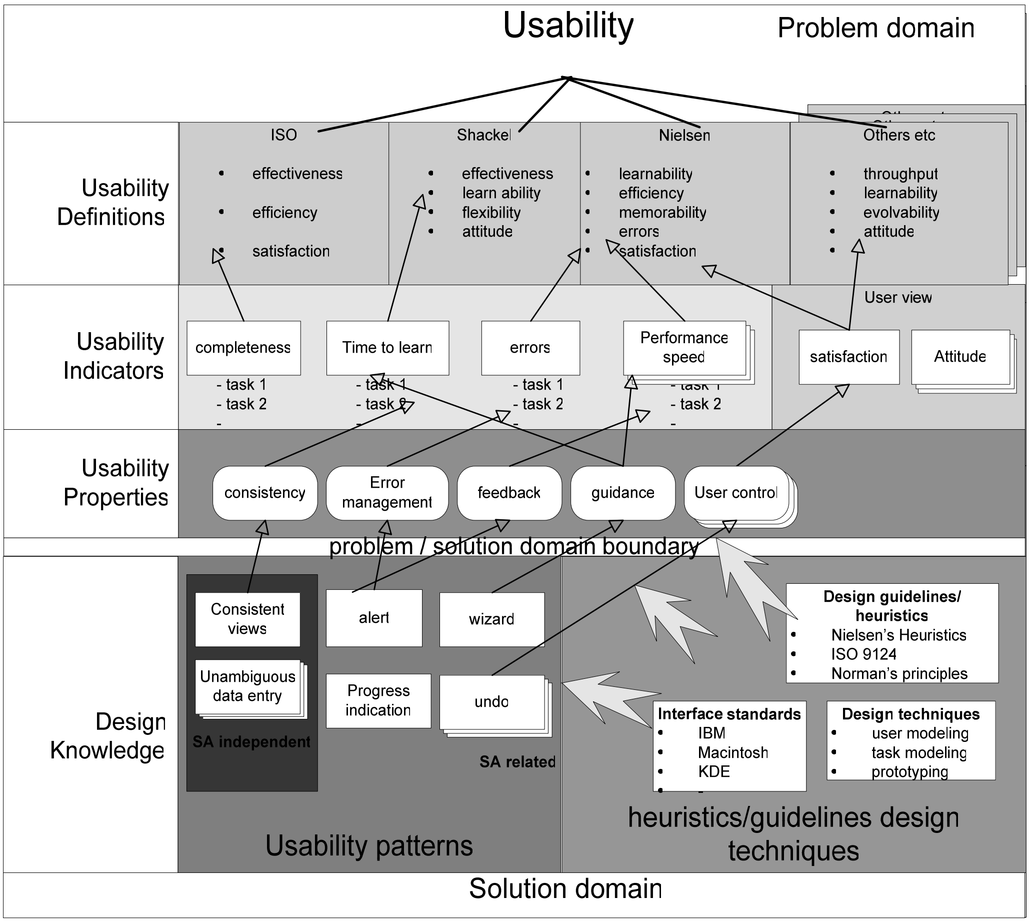


Figure 29: Usability Framework

Next to the need to organize these different interpretations of usability attributes, there was a need to relate usability to software architecture to be able to specifically design for usability at the architectural level. However, it is extremely difficult to draw a direct relation between usability attributes and software architecture. By refinement of the definition of usability attributes, it was attempted to decompose the set of attributes from our survey into more detailed elements, such as "the number of errors made during a task", which is an indication of reliability, or "time to learn a specific task" which is an indication of learnability, but this refinement still did not lead to a convincing connecting relationship with architecture. The only 'obvious' relation identified from our surveys between usability and architecture is that there are some usability patterns that have a positive effect on usability and are architecture sensitive as also identified by (Bass et al, 2001).

These issues led us to define the framework depicted in Figure 10. This framework was derived from the layered view on usability presented in (Welie et al, 1999). The concept of designing for and assessing of usability can be divided into parts relating to the problem domain and to the solution domain:

- The problem domain part deals with the concept of defining usability: what is usability? Which attributes compose usability? How do we assess usability?
- The solution domain provides practice and experience to improve usability by providing various design solutions such as heuristics or usability patterns that exist in literature and practice. How do we design for usability?

The framework created expresses:

- The relationships between the three questions surveyed in existing practice.
- The relation between the problem domain (defining/ assessing of usability) and the solution domain (design for usability).
- The different approaches towards the definition of usability.
- The relation between usability and the software architecture

To relate the problem domain to the solution domain, and hence relate usability to software architecture several intermediate layers have been defined which will be discussed in the next subsections.

2.5.1 Problem domain

Each layer is discussed starting from the problem domain. The framework consists of the following layers:

Usability definitions and attributes

The top layer consists of the concept of usability and its top level decomposition in measurable components. This layer consists of all the different definitions or classifications of usability according to the attributes which define the abstract concept of usability. In section 2.2 several definitions (Shackel, 1991, Nielsen, 1993, ISO 9241, ISO 9126-1) are discussed which are considered to have had the most influence in usability engineering.

Usability indicators

Usability attributes such as learnability or efficiency are abstract concepts. These attributes compose usability but they do not state exactly how these should be measured. Different authors in the field use different indicators to measure usability attributes. Learnability, for example, can be measured by measuring the time it takes to learn a specific task but also by measuring the number of errors made while performing such tasks. Therefore, at the layer below the usability definitions the usability indicators layer is defined. This layer consists of concrete measurable indicators of usability which are related to certain usability attributes. For instance, time to learn is a concrete measurable indicator for learnability. Because these indicators are very interaction dependent, they should be defined for each type of interaction, for instance, time to learn task one, or number of errors made performing task two. This survey does not investigate nor gives a complete list of usability indicators; they are stated here to illustrate the framework presented. Our survey of usability attributes in section 2.2 shows that usability attributes can be categorized into two classifications. These two categories are user performance attributes and user view attributes, which is the reason why the usability indicators have been separated into these two classifications.

2.5.2 Solution domain

The solution domain, for example, the current practice and research in usability engineering, provides us with various guidelines, tools and techniques that allow us to design for usability.

Design knowledge

At the lowest layer the design knowledge layer is defined. This layer expresses all "design knowledge" existing in the current usability engineering community; it consists of design heuristics, interface standards and design techniques such as user modeling, task modeling or prototyping and so on. The design knowledge provides design directives which can be very detailed for example apple interface standards states "concerning positioning windows, new document windows should open horizontally centered". Design knowledge such as design heuristics provide abstract directives such as "applications should provide feedback".
Usability patterns are also part of design knowledge. Usability patterns are proven solutions to a problem in a context; thus not an actual implementation of such a pattern. An example of a well-known and recognized usability pattern is the wizard pattern. Our approach to usability patterns is to relate these patterns to software architecture. A distinction is made between patterns that require architectural support and those that do not.

2.5.3 Relating the problem to solution domain

The problem domain consists of a composition of usability layer and usability indicators layer. The solution domain consists of a design knowledge layer and usability patterns layer. To relate the two domains an intermediate level is defined, called "usability properties" which relates the usability patterns layer to the usability indicators layer; hence relating the problem to the solution domain.

Usability properties

Usability properties are higher level concepts to which patterns and concrete solutions address; they are derived from design heuristics and design principles and ergonomic principles that suggest general "higher level" directions for design; such as providing feedback at any time or providing consistency. They are directly related to software design decisions. Concerning the usability properties it has to be noted that they do not have a strictly one to one relation with the usability indicators. For instance, the wizard pattern uses the usability property of guidance which decreases the time it takes to learn a task but it also increases the time taken to perform a task. Therefore the wizard pattern has a positive relationship to learnability, but a negative relation to performance or efficiency. In the framework only the obvious relations are stated.

The wizard pattern may be architectural sensitive because to implement a wizard, a provision is needed in the architecture for a wizard component. This component can be connected to other relevant components: the one triggering the operation and the one receiving the data gathered by the wizard. The wizard patterns stated here is an example of a pattern that may be architectural sensitive. We state “may” because usability patterns can be implemented quite differently, influencing architectural sensitiveness. Therefore verifying or proving the architectural sensitiveness of usability patterns is quite ambiguous.

Concluding, our framework visualizes the relation between usability and software architecture by defining intermediate terms such as usability properties which relates usability patterns to usability attributes which eventually relates software architecture to usability. Our objective was not to completely fill in the content of all layers but rather to present a framework in such way that it clearly visualizes the usability – software architecture relation.

2.6 Research Issues

Several issues have been identified in these surveys that require more research in order to be able to design for usability at the architectural level. Concerning the design approach taken to specifically design for quality attributes at the architectural level:

The design process depends on two requirements:

- It is required to determine when the software design process is finished. Therefore, assessment techniques are needed to provide quantitative or qualitative data, to determine if our architecture meets the non functional requirements. Our survey has not identified suitable assessment techniques that can be used hence such techniques need to be developed.

- Development or identification of architectural design decisions that improve usability, such as identification of usability patterns.

Concerning specification of usability during requirement analysis:

- Non functional requirements such as usability, performance or maintainability are weakly specified in requirement specifications. A more precise specification of required usability allows identification of architectural issues that are required to provide such a level of usability.

- Traditionally usability requirements have been specified such that these can be verified for an implemented system. However, such requirements are largely useless in a forward engineering process. Usability requirements need to take a more concrete form expressed in terms of the solution domain to influence architectural design.

Concerning the definition of usability:

- For our design approach the definition of usability is not an issue, the choice of whether to use a particular definition will depend on how well an evaluation tool based upon this definition, will support evaluation of usability at the architectural level. However since no suitable evaluation techniques were found in current practice eventually a suitable definition of usability should be used or defined.

Concerning design for usability:

- Due to the distance between the design and the evaluation phase, where feedback is received about the design decisions. Design for usability would benefit from design heuristics which specifically suggest which architectural styles and patterns to use for improving usability. Design knowledge should be captured in a form that can be used to inform architectural design, which allows for engineering for usability early on in the design process.

The framework presented in section 2.5 is a first step in identifying the relationship between usability and software architecture. More research should be spent on the following issues:

- Verifying the architectural sensitiveness of usability patterns.

- The relationships depicted in the framework between usability patterns, usability properties and usability attributes/indicators, indicate potential relationships. Further work is required to substantiate these relationships and to provide models and assessment procedures for the precise way that the relationships operate.

2.7 Conclusions

This survey has identified the weaknesses of current usability engineering practice. Most usability issues are only discovered late in the development process, during testing and deployment. This late detection of usability issues is largely due to the fact that in order to do a usability evaluation, it is necessary to have both a working system and a representative set of users present. This evaluation can only be done at the end of the design process. It is therefore expensive to go back and make changes at this stage. Most usability improving modifications are structural and can hence not be implemented because of its cost.

The work presented in this paper is motivated by the increasing realization in the software engineering community of the importance of software architecture for fulfilling quality requirements. The quality attributes of a software system are to a considerable extent defined by its software architecture. It is our conviction that designing for usability at the architectural level has the greatest influence on the usability of a system.

Usability should drive design at all stages. There are no techniques yet that can evaluate architectures for their support of usability. Iteratively design for usability at the architectural design phase is therefore not possible and hence it can be concluded that the goal that usability should drive design at all stages is not fully achieved. A new approach towards design for usability is proposed that has potential to improve current design for usability. Practice shows that such a design approach is required to successfully design for usability. This survey has explored the feasibility of such a design approach.
The main conclusions of this paper are:

- This survey justifies our design approach.

- There are no design techniques in current practice that allow for design for usability specifically at the architectural level.

- In current practice there are no evaluation techniques that can assess architectures for their support of usability.

A first step in further research could be identification of usability patterns that are architecture sensitive and the relation these usability patterns have with usability. Therefore closer investigation is required to examine the relationship between usability patterns, usability properties and usability attributes. This experience is necessary for development of design heuristics which specifically suggest which architectural styles and patterns to use for improving of or designing for usability. The design method presented is only a preliminary proposal for a design method for usability; future case studies should determine the validity of this method to refine it and make it generally applicable.