Université d'Ottawa • University of Ottawa

A Usability Study of the *"Tksee"* Software Exploration Tool

by

Francisco Herrera

A thesis presented to
the University of Ottawa
in fulfillment of the thesis requirement
for the degree of
Master in Computer Science

School of Information Technology and Engineering
University of Ottawa,
Ottawa, Ontario, CANADA

The Master in Computer Science is a joint program with
Carleton University, administered by the
Ottawa-Carleton Institute for Computer Science

ABSTRACT

A Usability Study of the *"Tksee"* Software Exploration Tool

by

Francisco Herrera

Master in Computer Science

University of Ottawa

Spring 1999

This research project explores effects of not having considered usability when assessing the functionality of a software system. Also, it experiments with the introduction of usability into latter stages of software projects in which developers are new to usability. A particular software project confronting that scenario is used to explore these issues. The selected project had created a program comprehension tool that had been given to a group of users for some time. For that reason, this research also explores challenges posed to studying the usability of program comprehension tools. A usability study is designed in order to perform the research. Three techniques are applied in the study: (a) *heuristic evaluation*, (b) *user and task analysis* and (c) *thinking aloud usability testing*. Heuristic evaluation is used to look for usability deficiencies in the tool design. This is achieved by asking a group of evaluators to judge the tool according to a list of usability guidelines and principles. Thinking aloud usability testing is used both (a) to look for problems that users experience when learning and using the tool, and also (b) to explore how well users who had been given the tool were capable of using its functionality. Both are achieved by observing a group of users performing some tasks typical of the tool's design intent. User and task analysis is used to find a right group of users and a good set of tasks for the thinking aloud usability testing technique. It is also used to obtain information about how the users had employed the tool. The study was very productive in finding information about all the above issues within the particular software project. It revealed that none of the users was capable yet to correctly use the tool functionality, and therefore it was impossible to assess the tool functionality at that moment. Not having considered usability had been one of the causes of this. Also, many issues were observed in relation to (a) having started considering usability at a late stage in the tool lifecycle and (b) the software project's reaction to being exposed to usability for the first time. Finally, certain particularities and difficulties were encountered with planning and studying the usability of the tool.

ACKNOWLEDGEMENTS

"In the immensity of space and eternity of time my happiness lies on sharing an age and a planet with you"

I want to thank: (a) Dr. Timothy Lethbridge for introducing me to this fascinating area as well as for all his support and help during the project, (b) the KBRE group, for their participation and help, (c) Gina (indeed, the best thing that has happened to my life), for her love and help throughout the project, and (d) the people of Mexico who through Conacyt provided the financial support.

With love to Gina, my Father, my Mother, Juan Arturo, Mauricio, Father in law and rest of the family. Also to Omar, Lorena, Armando, Lulú, Hans, Daniel and Pérez family, Quique and the rest of our friends.

TABLE OF CONTENTS

PAGE

CHAPTER

PAGE

PAGE

LIST OF FIGURES

PAGE

LIST OF TABLES

CHAPTER I

Introduction

1.1 Antecedents

One of the reasons for creating software systems is to make it possible for people to achieve their goals more efficiently. Systems seek to do this by providing functionality which supports people in tasks for accomplishing those goals. However, producing systems that can help people *effectively* and which are *fully adopted* by them is a complex process. Part of that process involves designing and implementing the system. Another part involves users learning, employing and finally adopting the system. Many times, a system must pass through these stages several times before it is really efficient and fully adopted. Good system design is key for a system seeking to help users effectively. Critical factors for creating systems with good design are:

a) The system must provide functionality that enables users do what they need. That is, the system must have good *utility*.

b) The system must be done in a way that facilitates users learning and efficiently exploiting its functionality. That is, the system must have good *usability*.

When a new system is created, requirement analysis activities are performed in order to discover possible functionality that can help users in their goals. That is, developers look for *what* might be good for the system to provide to the users. Once having identified certain functionality, developers build the system and give it to the users.

Nonetheless, it is common that developers want to confirm whether that functionality can in fact help users to achieve their goals more efficiently or whether it should be improved. To do that, developers give the system to a group of users for some time so

that they can use it when working on their goals. Then, later, developers explore if the system has helped the users to achieve their goals more efficiently or not. If not, developers asked these users for information about any noticed deficiencies in the functionality as well as suggestions for improving it. In the rest of the document, we will refer to this as *'developers assessing the functionality of the system'*.

Interest in assessing functionality becomes particularly strong in the case of systems seeking to help users in complex problems where it is unknown precisely how a system could help and what functionality it should provide. Typically, these are problems in which not even experts in the domain are able to suggest possible functionality for a system. In these cases, the truly efficiency of this type of functionality remains unknown until users employ the system for working on their goals.

Throughout this document, we will use the term *"experimental functionality"* to refer to functionality for which it is still unknown whether it can help users effectively and which developers want to assess.

1.2 Motivation and Objectives

Nowadays, a common situation in software projects is that a system is developed without considering its usability. Activities are undertaken to discover and implement certain functionality for the system, but nothing is done to ensure that users will be able to easily learn and efficiently employ that functionality. In these conditions, the system is given to a group of users for some time, and later, developers want to assess the functionality. That is, they want to see if the functionality has helped those users to achieve their goals more efficiently, as described in section 1.1. Here, it is important to notice that behind the idea of assessing the functionality, developers assume that the users will correctly use the system when working on their goals. In other words, that users will learn to correctly employ the functionality during the time the system is given to them.

This scenario motivated us to perform this research. We wanted to:

1) Explore effects of not having considered usability when assessing the functionality of a system.

We were interested in exploring effects in relation to (a) the usability of the system and (b) how users learn and employ the functionality during the time the system is given to them.

Another common situation is that people in those projects are not aware of usability. They don't know why it is important and how it can be considered when developing a system. Thus, we also wanted to:

2) Experiment with the introduction of user centered design and usability into the latter stages of software projects in which developers are new to these concepts.

To pursue the research, we selected a particular software project confronting the above scenario. The project we chose had developed a program comprehension tool providing experimental functionality. Based on this, we also wanted to:

3) Explore particular challenges posed to studying the usability of program comprehension tools.

In summary, the objective of the research was to look for evidence relating to these issues within such particular software project developing a program comprehension tool.

1.3 Contributions

The results and conclusions of this research should be of general interest to several sources. They should be of interest to:

- Software projects that want to assess the functionality of a system in which usability has never been considered. Particularly to projects developing systems providing experimental functionality.
- Anybody intending to study the usability of a program comprehension tool.
- Anybody intending to perform usability studies of software systems in general.
- Anybody intending to introduce usability into the latter stages of software projects in which developers are new to usability.
- Finally, to anybody interested in pursuing research opportunities discovered here.

Another secondary but important contribution should be to the usability of the program comprehension tool that we experimented with. The results of this research should help developers to improve the usability of the tool, which in turn should help the software project to assess and improve its experimental functionality. We want to mention that it was not the intent of this research to actually make any improvements to the tool, rather to explore the process that might lead to those improvements.

CHAPTER II

Review of Related Literature

All the issues we wanted to explore were related with usability of software systems. As mentioned in Chapter I, we pursued the research by experimenting with a particular software project. Hence, the literature search focused on two areas: (a) usability of software systems, and (b) details about the particular software project that was selected.

2.1 Usability of Software Systems

As mentioned earlier, the notion of usability is related to the way the functionality of a system is provided to the users. Usability in software systems has been traditionally considered within the scope of the user interface, and most times it is only addressed in detail by literature about user interface development. It has typically not been considered a central issue in software engineering. However, the success of a software system lies among other things in providing what users need in such a way that users can learn it and efficiently use it. In other words, creating a system with good utility and good usability.

Many different approaches for creating software systems in different fields of computer science currently exist, such as: *The Spiral Model for Software Development and Enhancement* [Boehm 1988], *The Unified Software Development Process* [Jacobson, Booch, Rumbaugh 1998], *The Usability Engineering Model* [Nielsen 1993a], *The Usability Engineering Lifecycle* [Mayhew 1999], etc. All discuss and address to a different extent the issues of utility (e.g. functionality) and usability of software systems. Some approaches address in detail the issue of utility but miss at fully addressing usability. The same is true vice-versa. Unfortunately, no complete approach fully addressing both issues currently exists.

Traditional approaches in the *Software Engineering* field describe development of functional and non-functional requirements for creating a system. They focus on capturing aspects of the utility of the system in such requirements and might mention capturing usability aspects of the system in the non-functional requirements. They provide methodologies for implementing those requirements into the system. However, these approaches can lead to creating a system with good utility but that does not necessarily lead to a system with good usability. They miss at providing some fundamental concepts and techniques that are well known in the *Human-Computer Interaction* field for ensuring that a system realizes good usability. For example, *the unified software development process* mentioned above describes a set of contextual task analysis activities to obtain information or *'use cases'* from which the requirements for a system are derived. However, this approach does not explicitly refer to a *'work environment analysis'*. This important part of a contextual task analysis points to aspects of the users' actual work environment that should drive user interface design decisions. Also, this method does not emphasize *'contextual observations of work'*, or even the need to tap into *'current user task organization'* and *'task sequence'* models in the design of the user interface. Consequently, while leading to create a system with good utility, this approach will not necessarily lead to a system with good usability [Mayhew 1999].

On the other hand, traditional approaches in the Human-Computer Interaction field focus on realizing good usability in a system but do not provide an in depth methodology addressing the utility of the system.

This phenomenon of incomplete approaches seems to be related to poor communication among groups of people supporting different approaches as well as different "cultures" or schools of thought. It is important to say that approaches in different "cultures" do not compete but rather potentially complement each other. Although no complete approach fully addressing utility and usability currently exists, some efforts are being undertaken [Rosson and Carroll 1995].

Among all the approaches, probably those that address best the issue of usability are those following the notion of *User Centered Design* [Norman and Draper 1986], which has the concept of usability at its center. Among other things, these approaches address in detail the issue of studying and evaluating the usability of software systems.

In this research, all the issues we wanted to explore involved studying the usability of a software system, therefore, an approach based on user centered design could provide concepts and techniques to achieve our objectives.

2.1.1 User Centered Design

The notion of user centered design involves a whole set of concepts and principles seeking to create products that are highly useful to the users. Much research has been done in the last decade within user centered design with respect to software systems. Gould and Lewis [1985] give a good description of the main principles behind it:

- Establish an early focus on users and the tasks they perform to achieve their goals.
- Perform empirical measurement of the software's usability using simulations and prototypes.
- Perform cycles of iterative design, where a cycle consisting of design, usability evaluation and redesign should be repeated as many times as necessary.

Many authors describe different approaches for creating software systems following the principles of user centered design [Hix, Hartson 1993; Nielsen 1993a; Shneiderman 1998; Mayhew 1999]. All embody the same principles although they differ in certain issues such as steps in the approach, notation, naming, etc. In order to be consistent, we decided to follow *The Usability Engineering* approach proposed by Nielsen [1993a]. In this approach, Nielsen describes a series of activities that should be followed by people developing a system. These activities can be divided into three stages:

1. A *pre-design* stage, which should provide essential concepts to be used through all the lifecycle of the software. This stage comprises the following activities:

    a) User and task analysis.
    b) Performing a competitive analysis.
    c) Setting usability goals.

2. A *design* stage, which guides the design of the user interface. This stage comprises the following activities:

    a) Performing parallel design.
    b) Including users in design.
    c) Coordinating the total interface of the system.
    d) Applying guidelines and heuristic rules.
    e) Prototyping.
    f) Performing empirical testing.
    g) Performing iterative design.

3. A post-*design* stage, in which feedback from the use of the system in the field is collected.

User and task analysis means analyzing the intended user population of the system in order to learn about them as well as the tasks with which the software is intended to help. At a minimum, developers should visit the users at their work place. User differences and variability in tasks are the two factors with the biggest impact on usability. The concept of users should include all people that will be affected by the software system. User experience is an important factor in user differences and that should be considered; Nielsen suggests that it differs in at least three dimensions, as Figure 2.1 shows:

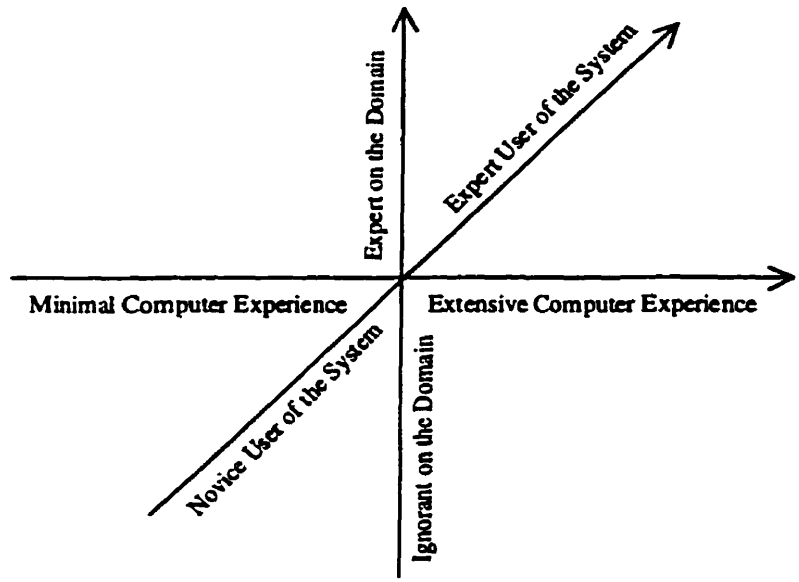


FIGURE 2.1 Dimensions in which users' experience differ

Learning about the tasks should include observing the users' goals, needs, and how they approach tasks. Exceptional circumstances should be considered too. Also, it is necessary to analyze if users' tasks could be improved by the use of the system.

Task analysis centered around users generates concrete statements of representative user tasks that provide good coverage of the functionality that the system should provide and that can be used for usability testing later. This contrasts with other traditional approaches, which attempt to derive an abstract functional specification of the system [Glen 1995]. Also, task analysis centered around users emphasizes human factors more than traditional analysis by attempting to identify where users waste time or are made uncomfortable. Early focus on users and tasks is one of the most important activities in order to reach high usability.

Performing a competitive analysis refers to analyzing similar products according to established usability guidelines in order to notice their strengths and weaknesses. Some user testing with other products can also serve to see how the functionality of the other

software supports the user tasks. This can provide ideas for the new software. Lewis and Rieman [1993] list some points in favor of using other product's ideas:

a) It is not easy for developers to come up with design ideas as good as those already implemented in high quality products.
b) Using ideas from other known products can improve the learning of the intended software because they are more likely to be already known by users.
c) It can save design time.

The intention is not to steal particular copyrighted ideas but rather to take general ideas and try to improve them.

Setting usability goals refers to establishing concrete goals that the system ideally has to comply with before it is released. Many times a trade off exists between different usability attributes, therefore the goals must be based on the results of the user and task analysis. Ideally, the intended system should perform better in these goals than other similar products. Also, Nielsen suggests performing a financial impact analysis to estimate the cost-benefits that will be provided by accomplishing these usability goals with the system.

Performing parallel design means to create different design ideas and then merge them all into one. In parallel design, the goal is to generate as much diversity as possible, therefore developers should work independently until they have completed the first drafts of the ideas, then all ideas are merged together into one design that comprises the best from all. Nielsen claims that parallel design can save development time by exploring many ideas at the same time.

Including users in design refers to have some users (a) criticizing developers' designs, (b) capturing problems with current developers' concepts, and (c) contributing with other ideas. This step is usually good at capturing mismatches between users' actual tasks and developers model of the tasks. Also, users seem to be very good at reacting to designs

they don't like or won't work in practice. However as Nielsen describes, *"users are not designers"* therefore this step shouldn't consist of asking users what they want, it is much better to show users paper mockups or some screen designs presenting the ideas.

Coordinating the total interface of the system means to review the design in order to provide consistency among all its parts. Consistency should apply to all parts of the current software project, including the user interface, documentation, training courses, etc. Developers should share and follow similar principles. Also here, the application of standards can help in achieving consistency.

Applying guidelines and heuristic rules has as its objective to implement well-known principles in the current system design in order to improve its usability. There are many well-known principles and guidelines that can provide improvements in the usability of a system. Standards, for instance, are one type of guideline.

A key step for developing usable systems is the development of an early *prototype* or simulation of the user interface. Prototypes can range from paper sketches to working programs. Any prototype should be quick to develop, as well as quick and easy to modify; therefore a good tool for prototyping plays an important role. Some authors [Hix and Hartson 1993; Lewis and Rieman 1993] suggest that a tool for prototyping should:

a) Be easy to learn and use.
b) Posses good visual capabilities.
c) Allow easy extensions of current prototypes.
d) Give full support for the type of interface being developed.
e) Support modular coding practices.
f) Provide support for turning the prototype into the final program.

Different types of prototypes vary according to the functionality they implement. A *horizontal* prototype is one where the complete user interface is presented but it has no underlying functionality. A *vertical* prototype includes in-depth functionality of only

certain parts of the interface. A *scenario* prototype fully implements selected paths in the user interface. A scenario prototype works well for user testing but it has the problem that users must stay within its paths.

Once a system design has been decided, it is necessary to *empirically test* it in order to capture usability problems. Testing is usually done using prototypes, however systems already implemented should be also tested. Whitefield et al. [1991] provide a classification of the different testing possibilities: (a) whether or not the system has been implemented, and (b) whether or not real users are involved in the testing.

Prototyping and testing are key activities in user centered design and have several purposes:

a) Performing early usability evaluation of the final system.
b) Exploring design ideas to get the best final choice.
c) Making iterative refinements.
d) Making easier the communication about the usability status of the current design to the development team.

One reason behind prototyping and testing is the concept that it is too difficult to design a complex system and get it correct the first time [Hix and Hartson 1993]. Some of the reasons are (a) the current limitations in psychological theory that don't permit people yet to accurately predict users' behavior with the system, and (b) the difficulty for developers at the start of a project to have a complete understanding of the entire context in which a system will be used on the field.

Therefore, any development project should expect and plan for iterations where the system is refined or even redesigned. This leads to the concept of performing iterative design where the system goes through cycles of design and evaluation until it reaches pre-established goals.

It is likely that many design decisions will be made during iterations. In order to maintain control and organization, Nielsen recommends maintaining a *design rationale document.* This document should contain the rationale behind decisions made about the system including the user interface. Also the document (a) captures the process followed throughout design, (b) serves to communicate design to other people, and (c) serves as a guide to future modifications of the system. A suggested format to capture design rationales is the *"QOC"* notation [McKerlie and MacLean 1993]. *QOC* stands for Questions, Options, and Criteria. Design questions are linked to design options, which in turn are linked to the criteria that caused particular options to be accepted or rejected.

Finally, once the system is released it is necessary to keep *collecting information* from its use in the field. This information should be used to plan for future versions of the system or for other products as well.

In cases where resources or time are limited, Nielsen [1994c] suggests a *Discount Usability Engineering* approach. In this approach the techniques to perform are:

a) User and task observation.
b) Cheap prototyping and scenarios.
c) Simplified thinking aloud usability testing where users verbalize their thoughts as they use the prototypes.
d) Heuristic evaluation in which the prototypes are evaluated according to guidelines.

User centered design contrasts with traditional development methodologies that follow the *waterfall* model in that (a) prototyping, (b) usability testing and (c) iterative design are performed. In the waterfall model on the other hand, development tries to go in a series of sequential stages where each stage is completed before the next one starts. Therefore in this model, the system specification is completed before implementation starts. On the other hand, user centered design assumes that usability problems will

appear in the first specification of the system and therefore a prototype is necessary to discover such problems.

Hix and Hartson [1993] describe user centered design as following a *"star life cycle"* (Figure 2.2) in which usability evaluation is at center of the star and the points of the star represent other activities during the design process like prototyping, for instance. The points in the star are not ordered and they can be repeated as necessary.

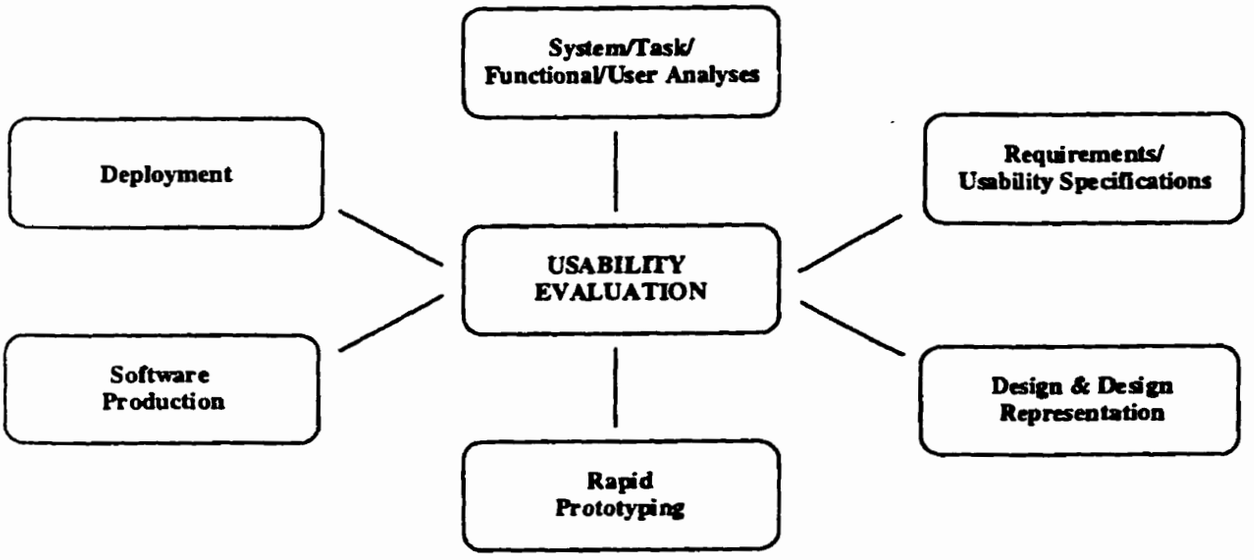


FIGURE 2.2 Start life cycle for software development

In one study [Nielsen 1993b], four case studies on iterative design were used. The interfaces passed through three to five iterations, with usability measured in the same way after each iteration. The study found average usability improvements of about 38% after every iteration.

Another study [Boehm *et al.* 1984] tried to compare the prototyping approach with the traditional software specification approach. Seven teams developed versions of the same

software. Three teams used the prototyping approach and four the specification approach. The study found that better interfaces were developed by the prototyping approach.

2.1.2 Usability Evaluation

As described above, usability testing is a key element in the user centered design approach of system development. Now, there are two important issues to explore: (a) what exactly usability is, and (b) how usability can be evaluated.

Usability is a narrow concern compared with the more general concept of *system acceptability*, which Nielsen describes as *"the question of whether the system is good enough to satisfy all the needs and requirements of the users and other potential stakeholders"*. Figure 2.3 below illustrates the factors of system acceptability and its relationship with usability as defined by Nielsen.

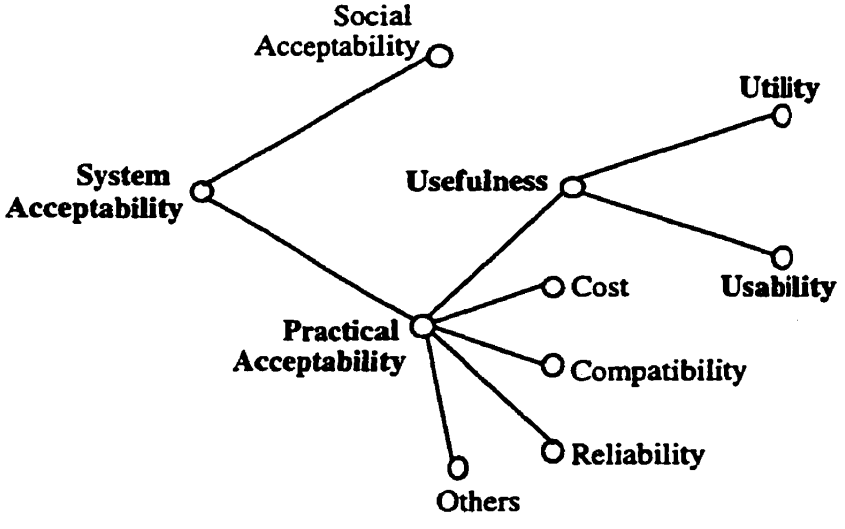


FIGURE 2.3 System acceptability and usability

In this model, *utility* is defined as the question of whether the functionality of the system can do what is needed, and *usability*, as the question of how well users can exploit that functionality. Therefore usability applies to all aspects of the system with which a human

might interact. The model also describes usability as mainly composed by five *usability attributes*:

a) Learnability.
b) Efficiency of Use.
c) Memorability.
d) Error Handling and Prevention.
e) User Satisfaction.

Learnability refers to how easy a system is to learn. In some sense, this is the most fundamental usability attribute since most systems have to be learned, and learning is the first experience that most users have with a system. There are many ways in which learnability can be measured; however, the concept of *learning a system* should always be seen as *users learning the system in order to achieve useful work.*

Efficiency of use refers to how efficient and productive a system is for a user once he or she has learned to use it. Users may never finish learning a system, although their learning curves tend to flatten out after they have acquired some expertise, as Figure 2.4 shows. Therefore, a typical way to measure efficiency of use is to decide on some definition of expertise, getting some users with such expertise and ask them to perform some tasks with the system.

Memorability refers to how easy a system is to remember. Typically, memorability is associated with casual users. A casual user is somebody who has learned the system but has not used it for some period of time. Improvements in learnability usually produce improvements in memorability [Nielsen 1994a].

Error handling and prevention refers to (a) the rate and number of errors that users make while using the system and (b) the ease with which users recover from those errors. Typically, an error is defined as any action that does not accomplish the desired goal. However, not all incorrect actions should be considered as errors, since sometimes users

recover almost immediately from them and there is not a big impact on user's performance. On the other hand, other errors are more catastrophic because they make it more difficult for users to recover from them. Both types of errors should be measured separately when considering this attribute.

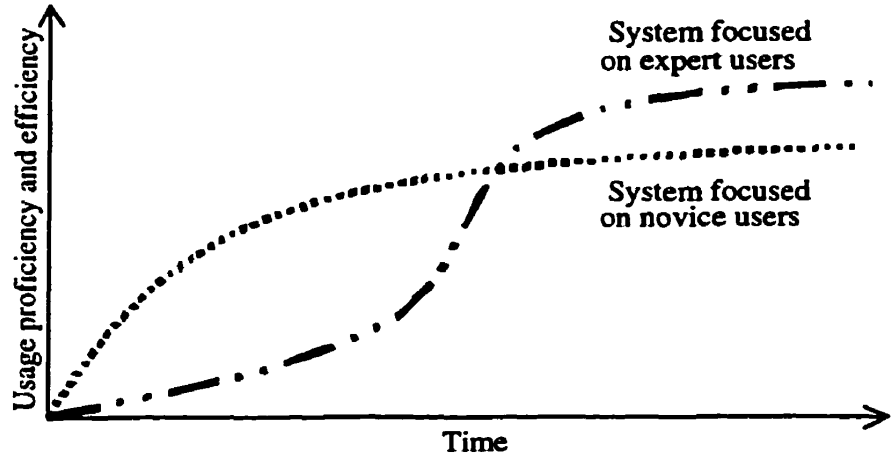


FIGURE 2.4 Learning curves of different systems

User satisfaction refers to a measure of how pleasant to use the system is. A typical way to measure satisfaction is asking users about their opinion of the system, which is generally achieved through the use of questionnaires or interviews. However it is necessary to be cautious with users' opinions because they are often closely related to the *peak* difficulty that users experience with the system, and also, people are often too polite in the responses.

Much research has been done in the last decade about evaluating usability and many different methodologies currently exist. The most common of all has been usability testing involving users [Nielsen and Mack 1994].

Typical goals when measuring usability are to provide quantitative measures of some usability attributes. For example, how long users take on average to complete a particular task or how many errors on average users make while performing certain tasks with the

system. Quantitative measures are normally used to set usability goals for a system, and to evaluate system status to see if it has reached predetermined objectives. However, qualitative measurements are also taken sometimes to measure users' satisfaction, like in the case of asking users to rate a system. The types of measurements to be taken depend on what the desired goal is as well as in what status within the lifecycle the system is.

2.1.2.1 Discount Usability Engineering

There have been some criticisms of classic usability testing methods [Mack and Nielsen 1994a]. Some of those are that classic methods require professional usability experts and special laboratory equipment, also, obtaining large numbers of users so that the results can be statistically significant is very expensive, time consuming and difficult. Furthermore, the costs may out weight the benefits in small projects. As we mentioned before, alternative methodologies have been proposed, like those in the discount usability engineering approach proposed by Nielsen [1994c]. In this approach, the emphasis is to get qualitative information that can be communicated to developers. This information identifies concrete usability problems with the system and helps in generating solutions to them.

Nielsen claims that the methods in this approach are easy to learn, inexpensive, fast to apply, don't require special equipment, and can be performed by software developers. He suggests that discount usability engineering can be used to supplement classical methods or when other more expensive techniques are not feasible. In fact, he says, in many practical situations the goal should be to identify usability problems and not necessarily to get quantitative measures about them.

Many different usability evaluation techniques exist and they can be divided on the basis of whether they use real users or not. Techniques that don't require users are usually referred as *Usability Inspection Methods* [Nielsen and Mack 1994]. In these techniques,

evaluators are used to inspect the system, looking for usability problems that users might have. Evaluators inspect the system based on guidelines and their own judgement.

As we will discuss in Chapter III, two usability evaluation techniques were applied in the research. Both following Nielsen's discount usability engineering approach. A description of such techniques is provided below.

2.1.2.2 Heuristic Evaluation

The first technique used in this research is a usability inspection methodology called *Heuristic Evaluation.* In this technique, evaluators receive a list of guidelines and are asked to explore the system's interface to look for things that violate any of the guidelines or that they consider as problems based on their experience and intuition. Evaluators are allowed to use any guidelines or principles they consider necessary. Each evaluator proceeds independently in his exploration and they are not allowed to communicate with each other until all explorations have finished. An observer may watch the evaluator's exploration to record notes and answer questions they have about the system.

Heuristic evaluation uses a short list of heuristic guidelines and few evaluators. A list of guidelines that has become very popular is the one developed by Nielsen [1992b]. This list contains the following ten usability guidelines that represent what any system with good usability is expected to have:

1. Simple and natural dialogue.
2. Speak users' language.
3. Minimize users' memory load.
4. Consistency.
5. Good feedback.
6. Clearly marked exits.
7. Shortcuts or accelerators.

8. Good error messages.
9. Good error prevention.
10. Good help and documentation.

It has been observed in heuristic evaluations, that single evaluators miss most of the problems in their evaluation, although different evaluators find different problems. Therefore, better results are obtained by combining information from several evaluators. In a study, Nielsen [1993a] averaged results from six different projects that used heuristic evaluation. He found that single evaluators discovered 35% of all the problems detected and five evaluators discovered 75% of them. For many practical purposes, Nielsen recommends the use of three to five evaluators.

Also, it has been seen that evaluators' expertise has some effect in the amount of problems detected. In another study [Nielsen 1992b], the same interface was subjected to heuristic evaluation by three different groups of evaluators: (a) *novices,* who had knowledge about computers but no usability expertise, (b) *single experts,* who were usability specialists but not specialized in the domain of the interface, and (c) *double experts,* who had expertise in both usability and the domain of the interface being evaluated. The study found that novices detected 22% of the problems in the interface, single experts 41% and double experts 60% of them. The study concluded that best results are obtained by using double experts as evaluators. However, the study recommends the use of single experts for many practical purposes and use of double experts just when optimal performance is necessary.

Supporters of heuristic evaluation claim that this technique is easy to learn, fast to apply and inexpensive. Some studies [Desurvire 1994; Jeffries *et al.* 1991] compared heuristic evaluation with other usability evaluation methods. They found that heuristic evaluation detects most of the usability problems discovered by the other techniques and it identifies the most serious ones too. They also confirmed that better results are obtained when evaluators have usability expertise.

2.1.2.3 Thinking Aloud Usability Testing

The second technique used in this research is called *Thinking Aloud Usability Testing.* In this technique, a group of end users of the system or *'participants'* is asked to verbalize their thoughts while performing some tasks with the system. The tasks must represent examples of the uses to which the system will be put in the field, and its solution must cover as much functionality of the system as possible. An observer may watch the participants to record notes, encourage them to keep talking and answer questions they have about the interface. However, the observer must be careful in not helping too much so that he doesn't alter participants' performance. The sessions might be recorded or videotaped for later analysis.

The theory behind speaking aloud while performing the tasks is that verbal comments provide insights into the user's mental process, therefore making it easier to understand how these interpret the system. In this way, it's possible to capture users' misinterpretations and problems with the system. However, care must be taken when listening to users, since data showing what they are doing when a problem is present has more validity than users' own theories about the problem. Thinking aloud provides qualitative information rather than quantitative information as in other classical usability techniques.

Supporters of thinking aloud usability testing, claim that this technique is easy to learn, fast to apply, inexpensive and can be performed by non usability experts. Lewis [1982] describes some advantages of thinking aloud usability testing:

a) Users' comments help to identify a problem as well as its causes.
b) Users discuss problems as they occur when details are fresh in the user's memory.
c) Minor problems that cause annoyance or confusion but don't affect task completion times are more likely to be detected by thinking aloud testing.
d) User's comments help reveal user's subjective attitudes towards the interface.

e) Thinking aloud testing can be used with incomplete prototypes or mockups, since it doesn't attempt to measure task completion times.

Many authors [Nielsen 1993a; Hackos 1998; Mayhew 1999] point out that good ethical principles must be followed in any study or technique involving users. For example, users must be properly informed about the study and their decisions and feelings must be respected. Also, adequate confidentiality must be kept on any information obtained about the users.

2.1.3 Acceptance of User Centered Design in Software Projects

Nowadays, many software projects face the necessity to develop highly useful systems. However, many of those are new to the notions of usability and user centered design. The success of user centered design greatly depends on the acceptance and commitment of the software development project to it. Several authors describe the idea of stages of acceptance and commitment to user centered design in software projects [Bias & Mayhew 1994; Ehrlich & Rohn 1994]. They explain that software projects can be in one of the following stages:

Stage 1. Skepticism

This stage is typical of projects that have never been involved with user centered design. At this stage developers believe that user centered design will only lengthen the development of the system. Normally, this type of project is very concerned just with the functionality and features of the system and not in its usability. If user centered design activities are started, developers don't see what type of benefits can be brought to the system and project.

Stage 2. Curiosity

As projects moved from skepticism, developers become curious about the benefits of user centered design. They recognize the problems with the system and might admit not having the skills for knowing what improvements should be done. They are reluctant to give control over the design of the user interface and utility of the system to someone who is not implementing it.

Stage 3. Acceptance

At this stage, people in the project understand and rely on user centered design. They accept somebody in the project providing guidance in the design and evaluation of the system. They see usability as critical but the system might still be shipped containing usability problems.

Stage 4. Partnership

The whole software project team is committed to user centered design. There is a great deal of communication among everybody developing the system, including developers and usability specialists. Good usability becomes a critical component that the system must have. Improvements in the system functionality are always verified for usability.

The same authors describe that the introduction of user centered design in software projects requires an evolutionary process in which the projects must gradually grow in their acceptance and commitment to user centered design.

2.2 The Software Project and Program Comprehension Tool

As stated earlier, we performed the research by exploring on a particular software project that had created a program comprehension tool. Such tool provided experimental

functionality that developers wanted to assess. In this section, we present details about (a) the software project and the problem it addressed, (b) the program comprehension tool, and (c) the status of the tool and software project at the time of this research.

2.2.1 The Software Project and Problem Addressed

The maintenance of large software systems has become a tremendous problem. Large software systems containing several millions of lines of source code have to be repaired and upgraded. *Software Engineers* (SEs) need to spend long periods of time exploring and trying to understand large bodies of source code in order to make the required changes. Also, it takes a long time for SEs who are novices to the systems to be able to start making the changes. Thus, the creation of effective tools that can help SEs in maintenance activities is a major issue today.

The *Knowledge Based Reverse Engineering software project* (KBRE) is a collaborative project between a large Telecommunications Company and the University of Ottawa, which has been looking for solutions to the mentioned problem.

One major goal of the KBRE project has been the creation of software tools that can effectively help SEs to perform maintenance activities on large software systems. Such tools should help SEs to learn about large software systems more quickly, and should also help SEs that are experts in a system to make changes more easily.

At the beginning of the KBRE project, the tasks and activities that SEs perform when maintaining the systems were not well understood. It was unknown how or what tasks software tools could provide help with and what functionality should be necessary to implement in such tools. Not even SEs expert in maintaining systems were able to accurately suggest possible functionality. This situation led people in the project to perform studies about *SE work practices* [Singer and Lethbridge 1997]. The goal was to generate ideas of possible functionality that could enhance the work practices and that

could be implemented into the tools. The rationale was that creating tools that meshed and enhanced existing behavior should help SEs in maintaining the systems and should be more easily adopted. The lack of tool adoption has been a major problem in the area of tool design for SEs.

Several types of data about SEs work practices were collected during these studies, including:

a) A web questionnaire asking SEs about their work.
b) Observation of the work of one SE at the Company during 14 weeks.
c) Observations of eight SEs at the Company during one hour of their work.
d) Series of interviews with SEs.
e) Analysis of company-wide tool usage statistics.

Many insights and ideas were obtained from these studies. One of these was that *searching* is an important and key component of the real, day-to-day work of SEs. Therefore, improvements in searching should help SEs to do their work better [Singer, Lethbridge *et al.* 1997]. This made people in the KBRE project focus the efforts on designing tools with good search capabilities.

Another observation was that SEs spend a significant amount of time looking at the source code of the systems, which suggested the creation of tools with intelligent source code viewing capabilities. According to these studies, SEs need to explore the software systems with the goal of determining where modifications have to be done. Also, there are two categories of SEs performing this task (a) novice SEs, who are not familiar with the system and must learn about it, and (b) expert SEs, who know the system very well but can not maintain a complete-enough mental model of it. Novices are normally less focused than experts and spend more time studying things that are not relevant to the problem. All SEs repeatedly search for items of interest in the source code and navigate relationships among them. They never try to understand the system in its entirety but they are content to understand just enough to make the changes required. After working on a

particular area they move to other parts of the system and forget details about the previous area, having to re-explore it again if they re-encounter it later. In conclusion, many SEs spend a large proportion of their working time trying to understand the source code prior to making changes to it. People who performed these studies call this approach *Just in Time Comprehension* [Singer and Lethbridge 1997].

Another important result obtained from these studies was a set of patterns that SEs follow when they explore software systems (Figure 2.5).

LOW LEVEL PATTERNS:
1. See if <something> is defined and/or what it is (file/procedure/function/variable/ parameter/data-structure/type/ etc).
2. Find information about the parameters of a certain procedure/function (types, where are these types defined).
3. Place(s) where <something> (procedure/function/data-structure/type) is implemented.
4. Place(s) where <something> is initialized (variable).
5. Place(s) where <something> is called/used (procedure/function/ variable).
6. What and where needs to be changed if <something> is changed (definition/ value/etc).
7. Place(s) where to change <something> specific (constant).
8. Follow some flow of execution.
9. See if <something> is used.
10. See possible returning values.
11. Trace conditions under <something> happens (function-result).
12. Find problems/activities where <something> has been involved (file/procedure/ function).
13. Find an example of <something> (some sorting procedure/ function, for example).
HIGH LEVEL PATTERNS:
14. What does <something> do? (function/procedure/data-structure/ variable).
15. Find where a particular functionality is defined/implemented (call waiting is defined, for example).
16. What are the parameters used by a certain procedure/function? (what do they mean).
17. What is the size of <something>? (variable/data-structure).
18. What is the format of a particular procedure/function call (how do you call this procedure/function, for example).
19. How do you change <something>? (format, for example).
20. Understand what someone changed on a previous problem report.
21. Draw data-flow diagram (for a particular variable).
22. Draw control-flow diagram (for a particular segment of code).
23. What procedure/function do you call to get a particular type of information (string size, for example).

FIGURE 2.5 Patterns that SEs follow when exploring the systems

The results from the studies were used to generate a set of requirements for software tools seeking to help with the Just in Time Comprehension approach. This set of requirements consisted of three functional and seven non-functional requirements, listed below:

"F1. Provide search capabilities such that the user can search for, by exact name or by way of regular expression pattern-matching, any named item or group of named items that are semantically significant in the source code. The term semantically significant refers to exclude arbitrary sequences of characters in the source code text that have no meaning to SEs."

"F2. Provide capabilities to display all relevant attributes of the items retrieved in requirement F1, and all relationships among the items."

"F3. Provide capabilities to keep track of separate searches and problem-solving sessions, and allow the navigation of a persistent history."

"NF1. Be able to automatically process a body of source code of very large size, i.e. consisting of at least several million lines of code. Many systems used by real industrial SEs can be large in size."

"NF2. Respond to most queries without perceptible delay."

"NF3. Process source code in a variety of programming languages." Studied SEs use at least two languages.

"NF4. Wherever possible, be able to interoperate with other software engineering tools." Studied SEs make use of other tools.

"NF5. Permit the independent development of user interfaces (clients)."

"NF6. Be well integrated and incorporate all frequently used facilities and advantages of tools that SEs already commonly use."

"NF7. Present the user with complete information, in a manner that facilitates the just-in-time comprehension task" [Lethbridge and Anquetil 1997].

2.2.2 The Program Comprehension Tool

Based on the above results, the KBRE project created a *Program Comprehension Tool* implementing those requirements in its functionality [Lethbridge and Anquetil 1997]. Since its creation, the following limitations to the tool were accepted:

"L1. The server component of the tool would run on only one particular platform."
"L2. The tool would not handle object oriented source code."
"L3. The tool would not deal with dynamic information, i.e. information about what occurs at run time."

Figure 2.6 below, shows the version of the tool used at the start of this research. The main features that fulfill the requirements of searching and displaying relevant attributes (F1 and F2) are implemented in the bottom two panes. The bottom left pane presents a hierarchy of items that can be incrementally expanded by asking to show attributes of such items and related items, or that can be searched for information about a given item. Selecting an item in the hierarchy presents information about it in the bottom right pane, from which the user can select any item of text and request information about it.

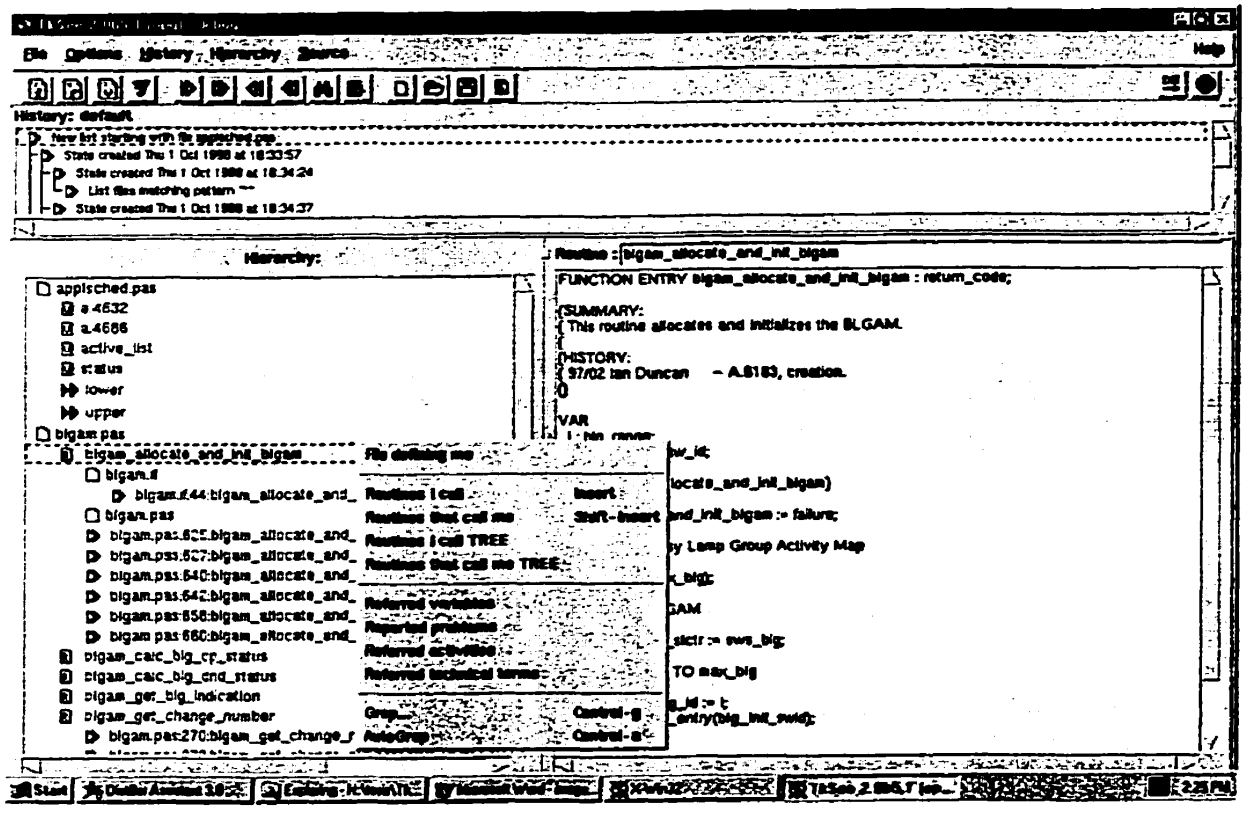


FIGURE 2.6 Main window of the Program Comprehension Tool

The main features that keep track of separate searches and allow a persistent history (F3) are in the top pane. Each element in this pane is a complete state of the bottom two panes. A hierarchy of these states is saved persistently, so that the user can work with the same explorations that he had from previous sessions.

The non-functional requirements are implemented within the architectural design of the system (Figure 2.7). This architecture includes a fast database, an interchange language for language-independent information about software, and a client-server mechanism allowing the incorporation of existing tools.

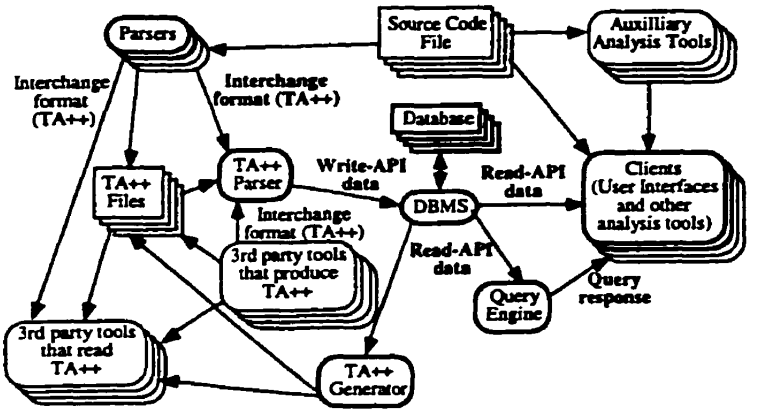


FIGURE 2.7 Architecture of the Program Comprehension Tool

2.2.3 Status of the Program Comprehension Tool and Software Project

As mentioned earlier, the functionality of the tool arose from the study of SEs work practices, and the nature of the problem it seek to help meant it was an *experimental functionality*. Since its creation, the tool had been given to a group of SEs at the Company so that they could use it during maintenance activities. At the time of this research, the tool had been available for 1.5 years to that group of SEs, and developers wanted to assess its functionality.

On the other hand, the tool had been developed without considering its usability. It had not been evaluated and tested for usability. Also, the KBRE project had not consistently monitored many issues such as:

a) Number of SEs that had employed the tool.
b) How those had learned and employ the functionality.
c) Problems and difficulties when learning the functionality.
d) Problems and difficulties for efficiently exploiting the functionality.
e) Level of expertise reached by them.
f) Subjective satisfaction with the tool.

In addition, people in the KBRE project didn't know much about user centered design. Some developers had a basic idea about user interface design, but basically nobody knew about usability and usability evaluation.

In summary, the tool and KBRE project provided an excellent foundation in which we could perform the research. In the next chapters, we will describe the procedure followed to perform the research as well as the results and conclusions obtained from it.

CHAPTER III

Methodology

According to the overall objective of the research (section 1.1), the goal was to look for evidence within the KBRE software project and its program comprehension tool about the issues in which we were interested. Looking for such evidence required performing an appropriate activity. This chapter presents a description of the rationale followed in planning that activity as well as a description of it.

3.1 Ideas for planning the Research

According to the first research objective, we had to look for effects of not having considered the usability of the tool. A first interest was on effects in relation with the usability of the tool at the moment of this research. A second interest was on effects in relation to how SEs had learned and employed the functionality during the past 1.5 years.

As suggested in the literature, we cold look for effects of the first type by studying the usability of the tool with some usability evaluation techniques. In regards to looking for effects of the second type we could proceed as follows:

1. Identify a group of SEs who had used the tool during maintenance activities *(category IV in Figure 3.1)*.

2. Explore how those SEs were capable of using the tool functionality. We could do that by asking them to complete some typical tasks in the best way they knew.

It is important to mention that the group of SEs who had been given the tool consisted of approximately twenty SEs, therefore it would be feasible to perform these investigations.

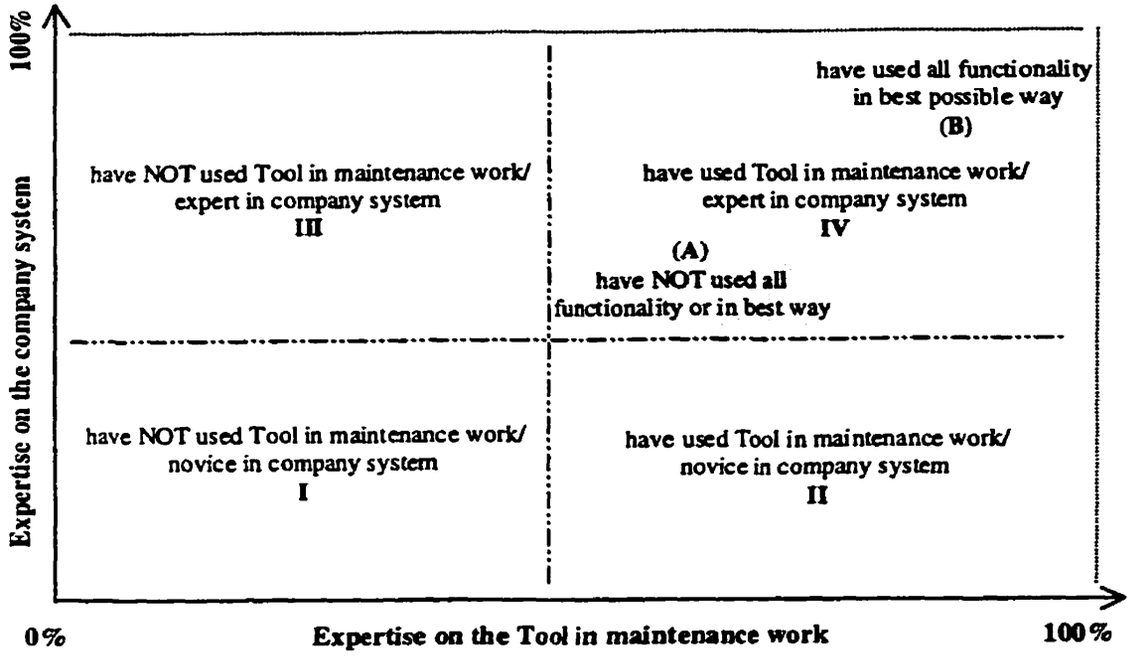


FIGURE 3.1 Classification of users of the tool (SEs)

According to the second research objective, we had to experiment with introducing usability into the KBRE project. Hence, the same study for exploring the usability of the tool suggested in the previous issue could serve for this. Nonetheless, we could broaden the experimentation by doing a whole iteration of evaluation, redesign and re-evaluation of the tool applying different evaluation techniques. We might discover different things when the tool is evaluated, when it is redesigned and when the redesign is evaluated again. Also, the use of different techniques might reveal different things.

According to the third research objective, we had to look for particularities of studying the usability of the program comprehension tool. Again, the same study suggested in the previous issues could serve for this. But once more, we could broaden the search by using

several types of usability evaluation techniques, such as one not involving users *(e.g. usability inspection method)* and another one including users. We might discover different particularities when using different types of techniques.

3.2 Study Plan

Based on the above ideas, we decided to do a usability study of the tool including the following three techniques:

a) *Heuristic Evaluation*
b) *User and Task Analysis*
c) *Thinking Aloud Usability Testing.*

Heuristic evaluation and thinking aloud usability testing would serve to explore the usability of the tool. Also, thinking aloud usability testing would be used to explore how a group of SEs who had used the tool during maintenance activities were capable of using the tool functionality. Because the idea in the research was to find qualitative information about the issues with which we were interested, both techniques would follow a discount usability engineering approach. Also, our interest was in usability issues preventing SEs from correctly using the functionality of the tool, therefore, the study would focus on searching for things about *learning, efficiency of use* and *subjective satisfaction.*

Heuristic evaluation would be the first technique applied. It would detect minor problems in the tool that developers would try to fix as much as possible before the thinking aloud usability testing.

The user and task analysis would be performed before the thinking aloud usability testing. This technique would serve to obtain a right group of SEs *(e.g. participants)* and a good set of tasks for the thinking aloud usability testing technique. Also, it would serve

to gather information about how the tool had been used at the Company. Next, we provide specific details regarding the application of each one of these techniques.

3.2.1 Heuristic Evaluation

This technique had the objective of looking for usability deficiencies in the tool by applying usability guidelines and principles. It would follow the approach described in Chapter II. That is, a group of evaluators would be asked to evaluate the tool according to the list of usability guidelines proposed by Nielsen. Three evaluators with some usability expertise would be used in order to get the best cost-benefit results. Each evaluator would proceed independently and might be accompanied by an observer.

The results would be summarized in a report and communicated to developers. Some sessions would be performed in which developers and the experimenter would review the problems together. Developers would work on solutions to the problems identified while the thinking aloud usability testing was prepared.

3.2.2 User and Task Analysis

This technique had the objectives of (a) identifying a right group of participants, (b) generating a good set tasks for the thinking aloud usability testing, and (c) gathering information about how the tool had been used at the Company. To achieve this, a series of interviews and observations would be performed with (a) people who had conducted the initial studies about SEs work practices, (b) developers of the tool and (c) people at the Company.

Regarding the selection of participants, the following groups of SEs would be identified:

1) SEs who had not used the tool in maintenance activities but knew about the company systems *(category III in Figure 3.1)*.

2) SEs who had used the tool in maintenance activities and knew about the company systems *(category IV in Figure 3.1)*.

From each one of these groups, three to five SEs would be invited to participate in the study. In the rest of the document, we will refer to those SEs simply as the *novice* and *expert* participants. Both groups of participants would be used to explore the usability of the tool. Novices would help at looking for deficiencies in *learning* the functionality and experts at looking for deficiencies about *efficiently using* the functionality. Both groups would be asked for information about their satisfaction with the tool. In addition, experts would serve to explore how well they were capable of using the tool functionality.

In regards to the tasks, these would be real tasks of the type for which the tool was designed. The patterns that SEs follow when exploring software systems would be used to help develop the tasks. These patterns had been observed when the tool functionality was generated and could help at finding real sequences of using the features. Also, some tasks would be designed so that performing them optimally would involve the use of critical features that had not been altered by the solutions to the problems found in the heuristic evaluation (e.g. features that remained in the same place with same label). This was important in order to guarantee that the knowledge that expert participants had about those features were still valid in the test. In other words, to guarantee that if an expert didn't use one of those features during the test, it was because he really didn't know the feature and not because the feature had been altered. In summary, the tasks would comply with the following rules:

a) Involve exploration of a real Company system.
b) Be linked to a real scenario of a maintenance activity.
c) Be possible to perform within a certain time limit.
d) Its solution should cover the patterns as much as possible.

e) Its solution should cover as much functionality of the tool as possible.
f) Its solution should involve critical features not altered after the heuristic evaluation.

3.2.3 Thinking Aloud Usability Testing

This technique had the objective of looking for usability deficiencies experienced by a group of SEs. Also, it would serve to explore how they had learned and used the tool. The participants and tasks obtained from the user and tasks analysis would be used for this technique. The technique would follow the approach described in Chapter II. That is, the participants would be asked to verbalize their thoughts while performing the tasks with the tool. There would be one session for each participant. The tasks would be given one at a time during the sessions. An observer would watch the participants and would take notes during the sessions. The sessions would be videotaped for later analysis. Once the participants had finished the tasks or the time limit had expired, they would be asked to complete a short questionnaire and debriefed for some minutes. The sessions would be performed at the Company facilities using computer equipment similar to that which the participants normally work with.

Like in the heuristic evaluation, the results would be summarized in a report and communicated to developers. Also, some sessions would be performed in which developers and experimenter would review the problems together. Some video clips of participants experiencing the problems would be presented.

Novice participants would receive a 15 to 20 minutes tutorial before the session. The tutorial would be given by a SE who was a member of the group responsible of supporting the tool within the Company.

As recommended in this type of technique, a pilot study would be performed before the sessions with the participants. Two users of the tool similar to the participants would be

asked to perform the tasks. They would follow the same procedure as the participants except that these then would not complete the questionnaire and would not be debriefed. The pilot study would serve to capture possible problems with the tasks, investigate details to be careful of when performing the sessions, and other practical issues about the application of the technique.

Finally, as suggested in the literature, certain ethical considerations would be applied. The videotapes and information obtained about the participants would be kept confidential. Only people performing the study would have access to that information. Also, just small clips showing specific problems would be watched during the sessions with developers. These considerations would be informed in advanced to the participants, who would be asked for their consent.

CHAPTER IV

Results

The following chapter presents the results of our research obtained through the usability study performed to the program comprehension tool. We discuss usability deficiencies found in the tool and how those were related to not having considered usability during its development. We also discuss how usability had affected the way SEs had learned and used the functionality of the tool. We describe particular challenges encountered with studying the usability of the tool and finally, we describe issues detected with introducing usability into the KBRE project.

As mentioned in Chapter III, the study involved applying three different methodologies. Thus, we present the results chronologically, with sections about each one of the techniques. At the end of the chapter there is a section with other results that were independent of the techniques.

4.1 Heuristic Evaluation

4.1.1 Procedure Overview

The first part of the study consisted of applying heuristic evaluation following the procedure described in Chapter III. That is, three evaluators with knowledge about designing user interfaces were used. All of them had some knowledge in usability evaluation although they were not usability experts; they had performed just a few heuristic evaluations before. *Evaluator-1* and *2* had backgrounds in Computer Science and experience in exploring software systems *(the domain of the tool). Evaluator-3* had a background in Psychology and some knowledge about programming languages, however he didn't have much experience in exploring software systems. On the other hand,

evaluator-1 was a developer of the tool and therefore he was fully familiar with it. Evaluator-2 and 3 were unfamiliar with the tool and learned about it one week before the study.

The heuristic evaluation was performed as described in Chapter III. That is, the evaluators received a list of ten usability guidelines and were asked to evaluate the tool according to them. Each evaluator performed the evaluation independently from the others and proceeded as follows:

a) Evaluator-2 (the author) performed the evaluation alone and before the other two evaluators. He would serve as the *observer* of evaluator-3 and as the *experimenter* of the heuristic evaluation. This is, he was in charge of coordinating the technique and summarizing the results obtained from it. Therefore, it was important that he performed the evaluation before the other evaluators so that he wouldn't suffer any bias introduced by the results of the others. Also, evaluator-2 had to be careful when serving as the observer of evaluator-3 to not introduce any bias to this later one.

b) Evaluator-1 performed the evaluation alone and sent his results to the experimenter.

c) Evaluator-3 performed a two-hour evaluation session in which he dictated his results to an observer (evaluator-2). The observer recorded the problems and helped with questions that evaluator-3 had about the tool.

The results from the three evaluators were merged and summarized in a report. We noticed that evaluator-2 (the experimenter) had acquired many insights by observing the results of the other two evaluators. Therefore, thanks to the precaution of performing his evaluation before the others, his results were not biased and it was possible to compare them with the other two evaluators. Finally, some sessions were done so that developers and the experimenter could review the results together.

4.1.2 Usability Insights obtained with the Heuristic Evaluation

The heuristic evaluation supplied a lot of information about the usability of the tool. Most of the results were *'possible'* usability problems that SEs could experience with the tool but there were also other insights like suggestions for improvements given by the evaluators. A total of one hundred and fourteen usability problems were identified and they can be consulted in Appendix 2. Seventy-one problems (62%) were about difficulties in learning the tool, sixty-five (57%) were about efficiency of use, and twenty-two (19%) were about both (notice that a problem can be about learning and efficiency of use at the same time). Three examples of these problems are:

- ***Problem:*** *The tool had two windows that showed a hierarchy of items. Also, the tool had a menu called "Hierarchy" that only referred to the hierarchy of one of the windows. This could be confusing for SEs that don't know yet the relationship (Learning problem).*

- ***Problem:*** *Some searching features worked so that they searched down starting from the point where they had been called. However, they didn't allow searching the rest of the information that had been left above. (Efficiency of use problem).*

- ***Problem:*** *A dialog window presents some radio buttons not according to the standards. This type of radio buttons should not be used for representing exclusive-or (Efficiency and learning of use problem).*

According to the usability guidelines, the problems were distributed as follows:

Usability Guideline	No Problems
1. Simple and natural dialogue	49
2. Speak users' language	20
3. Minimize users' memory load	10
4. Consistency	15
5. Good feedback	11
6. Clearly marked exits	1
7. Shortcuts or accelerators	2
8. Good error messages	0
9. Good error prevention	5
10. Good help and documentation	1

TABLE 4.1 Problems found in the Heuristic Evaluation according to the usability guidelines

As we will describe later, we generated a second categorization of the problems. According to that classification the problems were distributed as follows:

Category	No Problems
1. Non optimal functionality	22
2. Possible confusion	14
3. Possible misinterpretation	11
4. Unnecessary functionality	5
5. Lack of functionality	11
6. Lack of consistency	15
7. Graphical design	9
8. Feedback	11
9. Lack of robustness	6
10. Wrong behavior	2
11. Bad labeling	6
12. Lack of labeling	2

TABLE 4.2 Problems found in the Heuristic Evaluation according to the second classification

According to the usability guidelines (Table 4.1) the biggest category of problems consisted of violations to the *simple and natural dialogue* principle. Its problems accounted for 42% of the problems found and most of them were related with (a) *non-optimal functionality* (30%), (b) possible *users' confusion or misunderstanding*

with the tool (18%), and (c) *bad graphical design* in the user interface (16%) (Table 4.2). Some problems in this category were:

- ***Problem:*** *Certain windows always displayed scroll bars even when they were not necessary. It would be better to implement scroll bars as needed. (Non-optimal functionality).*

- ***Problem:*** *Two icons in the tool bar suggested they perform opposite actions on a selected object. However, this was only true when applied to certain objects (Possible users' confusion).*

- ***Problem:*** *The spacing and alignment of features within some dialog windows made things look cluttered even though there were not space constraints (Graphical design problem).*

The second biggest category of problems consisted of violations to the *speak users' language* guideline, which accounted for 17% of the problems found. These problems were mainly related to possible *user confusions or misunderstandings* with the tool. A problem in this category was:

- ***Problem:*** *The information at the top of one window appears editable but is not. SEs that know similar widgets from other systems could think the information is editable (Speak users' language problem).*

The third biggest category of problems consisted of violations to the *consistency* guideline, which accounted for 13% of the problems found. There were many inconsistencies throughout the interface but major sources of inconsistencies were in (a) the design of the features and its behavior, (b) the treatment of feedback, and (c) the graphical design of the user interface. Three problems in this category were:

- ***Problem:*** *The left and right arrow keys did not always behave the same in the different windows and dialog windows (Inconsistent behavior).*

- ***Problem:*** *Some dialogues and messages were using a different font than the rest of the system (Inconsistent graphical design).*

- ***Problem:*** *Some features gave feedback while processing information but others didn't (Inconsistent feedback).*

Together, the problems about the *simple and natural dialogue* and *consistency* guidelines accounted for 55% of all the problems found. By analyzing the problems, we saw that many of them were a consequence of the lack of systematic usability activities during the development of the tool. For example:

a) Many problems about *simple and natural dialogue* were deficiencies in the organization of the functionality of the tool. Many would have probably not existed if that arrangement had been designed by following tasks within real scenarios that SEs perform. Also, many were of the type of problems normally detected on a first evaluation. An example of these problems is:

   - ***Problem:*** *The feature for loading database files opened a pop-up window that showed all the files in the default directory, including files that were not valid databases. SEs could select wrong files and that could cause unexpected problems (Lack of evaluation problem).*

b) Many consistency and graphical design problems would not have existed if the tool had been implemented by following a standard style definition. An example of these problems is:

   - ***Problem:*** *Some dialog windows presented the scroll bar on the left and others on the right (Lack of standard style problem).*

It is important to say that the problems found were probably not all the existing problems in the tool. As we will mention in the next section, lack of domain knowledge and missing tool information hindered developers from identifying certain types of problems. Many of those problems would have belonged to the categories of (a) *simple and natural dialogue*, (b) *good feedback*, (c) *good error prevention*, and (d) *clearly marked exits*. Likewise, the tool didn't have online nor written help yet, therefore almost no issues regarding usability of the tool in combination with its help were obtained.

4.1.3 Issues during the Heuristic Evaluation

The heuristic evaluation was in general easy to plan and perform although there were certain things particularly more difficult or different than expected. We encountered the following issues when using this technique:

1. Although it had been relatively easy to find three evaluators with some knowledge about usability, we noticed that finding evaluators with good knowledge in usability and experience in heuristic evaluation was not easy in general. Finding a fourth evaluator would have been very difficult, for instance. We saw that people with experience in usability were not very easy to find at that moment.

2. The evaluators ended up relying more on their own intuition and experience with user interfaces than in the abstract usability guidelines when capturing problems. We noticed that little experience evaluating a system according to abstract principles was the main cause of this.

3. Insufficient information about the tool hindered some evaluators from identifying problems. As we mentioned earlier, evaluator-2 had little knowledge about the tool and evaluator-3 didn't know about the tool and its domain either. All this made them seek for

whatever tool information available in order to be able to make certain judgements. For example:

- They looked for a complete list of tool requirements and specifications. They wanted to know how the functionality had been designed to address the SEs tasks. However, this information didn't exist completely and that meant they couldn't always judge whether things in the functionality had or not usability deficiencies.

- They looked for information about the design of the user interface. They wanted to know why the functionality had been organized the way it was. Unfortunately, this information didn't exist either and that meant they couldn't always judge whether things about such organization were good or bad designs. This happened, for instance, with some groupings of features, labels, keyboard shortcuts and graphical design issues.

We belief that, with this information, evaluators would have made better judgements about which were critical problems and would have probably identified a larger number of problems.

4. There were problems found by several evaluators and others just by one evaluator. Also, each evaluator captured a different number of problems. By analyzing the results, we noticed that different evaluators tended to find different categories of problems:

- Evaluator-1 found 28% of the total number of problems captured in the technique. He found many problems regarding *'bugs'* and the utility of the tool, but tended to find fewer problems about user interface design than the other two evaluators. For example, he found many things about (a) the lack of certain tool functionality, (b) the correctness of the results given by the tool, and (c) the correctness and completeness of the feedback given by the tool. He was able to judge whether a feature was indeed necessary or not for exploring software systems and whether it was working correctly. On the other hand, his extensive experience with the tool made him unable

to notice certain types of problems, such as inconsistencies, graphical design, feedback, etc. For example, he didn't find many things about (a) difficulties experienced by SEs learning the tool (like problems for finding certain features, for instance), or (b) inconsistencies and lack of standards in the user interface.

We noticed that being a developer and having extensive knowledge in the tool domain made him capture those types of problems. His insights revealed deficiencies about tool learnability and efficiency of use.

- Evaluator-3 found 72% of the total number of problems captured in the technique. He found, on the other hand, many problems about user interface design but less about the utility and correctness of the tool. For example, he was very good at noticing problems about (a) consistency and standards, (b) graphical design, (c) labels, (d) difficulties for learning the tool, and (e) difficulties understanding error messages.

  We noticed that having little knowledge about the tool and its domain made him capture those types of problems. His insights revealed deficiencies about tool learnability.

- Evaluator-2 found 39% of the total number of problems captured in the technique. He was more balanced than the other two evaluators in finding deficiencies related with both: (a) the utility of the tool and (b) general user interface design. Although, overall, his results were more similar to evaluator-1; that is, he tended to find more problems with the utility and correct behavior of the tool and less about user interface design. We think this probably happened due to his background in Computer Science and experience exploring software systems (the tool domain). His insights revealed deficiencies about tool learnability and efficiency of use.

5. Merging and summarizing the problems required much time and energy. As stated in the plan, the problems would be communicated to developers so that they worked on solutions while the thinking aloud usability testing technique was prepared. But, solving

the problems efficiently required that developers understood the problems very well. As suggested in the literature [Jeffries 1994], the problems had to be presented in the report in a way that facilitated their understanding by developers. However, the problems as given by the evaluators presented many difficulties so that these could be easily included in the report:

- Many didn't provide information about its location in the tool.
- Many were not stated in their most general way. Instead, they only referred to particular instances of the problem within the tool.
- Many didn't have explanations about why they could be in fact problems to the SEs.
- Many didn't have suggestions for improvements.
- Many didn't provide an estimation of its severity.
- Many had not been related to the usability guidelines.

At the end, many problems had to be located in the tool, verified for its existence, rephrased or rewritten before including them in the report. The time and effort for this task was much bigger than expected.

We noticed that little experience with the technique and with communicating usability problems to developers had been the main reasons why the evaluators had not reported the problems better.

6. Having evaluators with different levels of experience in (a) the tool domain and (b) using the tool, provided a broader and larger spectrum of problems. Regarding the first type, we noticed that the tool domain was so specific and complex that only evaluators 1 and 2 who had background in Computer Science and knowledge exploring software systems were able to see deficiencies in the functionality regarding how it helped conveniently at exploring software systems. On the other hand, little knowledge in the tool domain made evaluator-3 capture other types of problems not detected by the other two, such as screen design problems. With regard to the second type, we noticed

that having a developer of the tool as evaluator-1 had been very helpful. He had extensive knowledge using the tool and that made him capture many insights about its correct behavior. On the other hand, evaluators-2 and 3 didn't know how to use the tool and that made them identify many problems with learning the tool.

As mentioned in Chapter II, some authors describe that best results are obtained by using *double-experts* as evaluators (e.g. evaluators with expertise in usability and the domain of the system). However, we obtained a broader spectrum of problems by having different types of evaluators as described above. That is, we had one *single-expert* and two *double-experts*. We also added the category of expertise using the system and we had one expert and two novices on using the tool.

We do not know if this combination can always be better in heuristic evaluations, particularly in evaluations of systems where the problem domain is very complex. But this is a topic for further research where more experimentation could be performed.

7. The number of problems was much bigger than expected and the project was overwhelmed with the results. Many of them represented difficult or costly things to repair and some were not possible to fix. The fact that the tool was already implemented imposed even more difficulties to fixing the problems. All these were causes of not having evaluated the tool before and lack of knowledge of what usability deficiencies meant.

8. Prioritizing and organizing the problems became critical to developers. According to the plan, developers should fix as many problems as possible while the thinking aloud usability testing was prepared. The large number of problems and limited time to fix them made it critical for developers to prioritize and organize the solutions. For that, developers immediately looked at the severity ratings provided in the report. Nevertheless, they ended up relying more on other factors such as time and resources available to prioritize the problems.

To organize the problems, developers tried to use the usability guidelines but these were very abstract for them and consequently not very useful. Based on that, we decided to create a second classification of the problems that they could understand and help them better at organizing the problems. This classification divided the problems in terms of the tool functionality and difficulties that SEs could have with it (Table 4.3).

1. Non optimal functionality
2. Possible confusion
3. Possible misinterpretation
4. Unnecessary functionality
5. Lack of functionality
6. Lack of consistency
7. Graphical design
8. Feedback
9. Lack of robustness
10. Wrong behavior
11. Bad labeling
12. Lack of labeling

TABLE 4.3 Second classification of the problems

9. Developers had difficulties identifying good solutions to the problems. We noticed it was difficult for them to understand the problems in their most general way. They required much time to see solutions that did not lead to other obvious usability problems. Also, they had difficulties comparing the effectiveness of several proposed solutions. We noticed that insufficient usability knowledge and experience solving usability problems was the main cause of this.

On the other side, insufficient tool information also affected developers ability to identify good solutions to the problems. Some developers were not in the KBRE project yet when certain decisions about the design of the tool had been made. Consequently, insufficient tool information meant that these developers didn't know details about the organization of the functionality as well as the design rationale of the user interface. Hence, developers experienced difficulties judging the current design and finding solutions consistent with the rationale used. We noticed this issue made people in the project start thinking about generating some tool documentation.

10. The problems were not always fixed properly and with optimal solutions. Many times, they were fixed for the particular instances mentioned in the report. Also, certain solutions would very likely lead to other usability problems. We noticed that non-optimal understanding of the problems and its consequences, having the tool already implemented, limited time and non-optimal prioritization of the problems were the main causes of this.

11. The sessions between experimenter and developers were extremely helpful. We confirmed that it had been essential to have the report properly written. However, we saw that the report alone would have been far from enough to communicate all the information to developers and facilitate the generation of good solutions. Instead, the sessions help at:

- Reviewing the problems in its most general way.
- Reviewing implications of sub-optimal solutions.
- Evaluating different proposed solutions.
- Prioritizing the problems.
- Deciding about problems that should wait to be fixed. There were some problems for which it was not clear if they seriously affected SEs and therefore their solution should wait until they were confirmed in the thinking aloud usability testing.

We want to mention that the experimenter played a key role in these sessions. He had to actively help developers in all the above issues.

12. Certain problems provided by this technique were not always very convincing to developers. Even though developers accepted most of the problems as deficiencies in the tool, some developers were not very convinced that certain problems should be fixed. We noticed that the lack of belief from developers came mostly from insufficient

understanding of how those problems could produce an effect on the SEs, particularly those that could produce a slow but strong impact on them.

13. After fixing several problems, developers started having some difficulties maintaining control over the source code of the tool. As one developer expressed: *"this code is starting to suffer from the spaghetti syndrome, we are losing order and structure in it"*. We noticed this was a consequence that the tool was already implemented.

14. Knowledge in the tool domain had been essential for the experimenter. He had to help evaluator-3 with several issues about the domain. Also, he had to fully understand the problems, appropriately explain those in the report, communicate them to developers, and provide advice during the generation of solutions. We noticed that the tool domain was so specific and complex that insufficient knowledge in it would have seriously hindered the experimenter in these activities.

15. The heuristic evaluation was a good first step for introducing usability and user centered design into the KBRE project. It made developers become aware of what usability meant and the level of usability of the tool. It also meant that the project matured regarding user centered design. By the end of this technique, the project had passed from the *skepticism* stage to the *curiosity* stage, as described in Chapter II.

4.2 User and Task Analysis

4.2.1 Procedure Overview

The next part of the study consisted of performing a user and task analysis as mentioned in Chapter III. That is, a series of interviews and observations were performed with (a) people who had conducted the initial studies about SEs work practices, (b) developers of the tool and (c) people at the Company.

The first step in the analysis consisted of some interviews with the people that had conducted the initial studies about SEs work practices and developers of the tool. We asked them about the KBRE project, the problem it addressed, the results from those studies, and the tool.

The next step consisted of field observations and interviews with some SEs and other people in the Company involved in the KBRE project. We wanted to gather more information about the project and the tool as well as obtaining a list of possible participants for the thinking aloud usability testing technique.

Once a list of possible participants had been obtained, the next step was to interview each one of them in order to select those who were the best according to the objectives. We had to identify a right group of novice and expert participants. It is important to mention that the experiment was not explained to them at this point.

Other SEs not in the list of possible participants were interviewed next. We asked them to give some examples of typical scenarios and tasks in the exploration of the Company systems during maintenance activities. Such examples would be used in the generation of the tasks. It was important that the possible participants didn't know about the tasks that they would be asked to perform and that's why we asked other SEs for these examples.

After the previous interviews, the manager of some SEs was also interviewed. He had been in involved with the KBRE project and was familiar with the tool. We asked him to verify all the information collected in the previous steps and contribute with missing information. We also asked him for some examples of typical scenarios and tasks in the exploration of the Company systems during maintenance activities.

At this moment, the study and its policies were explained to the possible participants and they were asked to participate in the experiment.

Once having obtained a group of participants, the rest of the analysis focused on generating the tasks. As described in Chapter III, the procedure for obtaining the tasks was performed. However, as it will be discussed below, certain difficulties arose with it as planned and it had to be modified.

4.2.2 Insights obtained with the User and Task Analysis

The user and task analysis provided extremely useful information. It helped to find a right group of participants and a good set of tasks according to the objectives. It also provided many insights about (a) the target population of users of the tool, (b) their needs and tasks, as well as (c) how SEs had used tool during the past 1.5 years and how useful it had been for them. All sources of information were good and provided useful information. Frequently, different types of information were obtained at the same time. Overall, the selection of the participants was completed before the tasks had been fully determined.

One of the first things found was that the target population of users of the tool had been defined as: *'those SEs performing maintenance activities that followed the Just in Time Comprehension approach'*. Also, that the tool had been designed mostly based on information from one particular group of SEs. However, the group of SEs who had been given the tool included (a) the SEs in this previous group as well as (b) other SEs in other groups that also performed maintenance activities and followed the *Just in Time Comprehension* approach.

It was found next that only four SEs had used the tool during maintenance activities so far, and those had been using the tool very differently. Some had employed the tool everyday and others just on specific days. Some had employed the tool for many of their tasks and others only for a few specific tasks. Also, three of them had been more recently hired at the Company. These commented on having used the tool in order to learn the

Company systems. On the other hand, many SEs had not used the tool yet and a few had not even heard about it.

All this made us see that the group of SEs who had used the tool during maintenance activities was much smaller than developers imagined. Developers thought that the tool was more widely known and used by the SEs. It was clear that SEs had not been monitored much after the initial studies.

At this moment, we identified a group of SEs who had used and had not used the tool during maintenance activities. These were good candidates to be the expert and novice participants for the testing respectively.

Next, we noticed that SEs in distinct Company groups had differences that could affect their interaction with the tool. SEs in different groups (a) had different goals in mind when exploring the Company systems and (b) tended to have a very different knowledge about the Company systems. These differences meant that two SEs in different groups could:

a) Approach the same goal with different tasks.
b) Perform the same tasks but in a different order.

These, in turn, had an influence in:

a) When they explored a Company system.
b) How they explored a Company system. That is, what the starting and final points of the exploration were, which paths of exploration they followed, how deep within the Company system the exploration was, etc.

These differences, together with the fact that the tool had been mostly designed based on one particular group, made us conjecture whether the tool was equally useful or not to all SEs within the defined target population. In fact, as it will be discussed later, we

confirmed this hypothesis later during the thinking aloud usability testing. The following is an observed example of all these issues:

SEs in the design group performed tasks where the starting point of exploration was a Pascal procedure. Though, SEs in the support group said they rarely started explorations by looking at a Pascal procedure. On the other hand, SEs in the support group started explorations by looking at descriptions of reported problems. However, the tool provided many features for starting explorations by looking at a Pascal procedure and few for starting explorations by looking at reported problems. Consequently, the tool was probably more useful to SEs in the design group than those in the support group. More information from SEs in the design group had been used to drive the design of the tool than from SEs in the support group.

Likewise, we noticed that SEs within the same Company group had also differences that could affect their interaction with the tool. SEs in the same group could have a completely different knowledge of the same Company system, and that could have the effect that for the same goal certain SE had to explore the Company system but another one not.

At this moment, the candidates were invited to participate in the test. Eight participants were obtained for the thinking aloud usability testing. Initially, as it will be explained below, four were identified as novices and four as expert participants (categories III and IV in Figure 3.1 respectively), but later, we discovered that one of the experts belonged in fact to the novice category (he had very recently started using the tool). Seven were in a same Company group and one of the experts (the one we discovered later was a novice) was from a different group. Some of them had participated in the initial studies about SEs work practices. It is important to mention that the group of experts consisted of the only three SEs who had used the tool during maintenance activities.

Finally, the tasks for the experiment were generated. As stated before, SEs were asked for examples of typical scenarios and tasks in the exploration of the Company systems

during maintenance activities. Here, we noticed that SEs always provided examples that were extremely complex and focused on very high level goals. Many times, they had not been able to accurately describe what sub-tasks or steps would be necessary to perform for solving the examples, neither what explorations of the Company system had to be done. This made us see that SEs not always consciously realized many of these issues during maintenance activities.

The set of tasks obtained contained twenty-eight tasks complying with all the rules established; however, as it will be discussed below, these had to be modified two times later, one after the pilot study and another one after the sessions with the first two participants.

4.2.3 Issues during the User and Task Analysis

The user and task analysis was easy to plan but not very easy to perform. We encountered the following issues during this technique:

1. Efficiently employing theory and principles at the moment of conducting the technique was not very easy to do. Theory and principles about planning and conducting the technique were simple and straightforward to understand. But remembering all and using them efficiently while conducting the technique was not easy. We noticed that the nature and amount of information gathered created a "messiness" atmosphere in which it was difficult to recognize the best principles and the way to use them. This made us see that obtaining best results definitely required experience and maturity with the technique.

2. The tasks were particularly hard to generate. Summarizing, the original procedure planned for determining the tasks was as shown in Table 4.4 below.

1. Ask SEs for some examples of typical scenarios and tasks in the exploration of the Company systems during maintenance activities
2. Take the collection of patterns found in the study of SEs work practices
3. While (there are scenarios)
   3.1 Take next scenario
   3.2 While (there are tasks in that scenario)
   - Take next task
   - See what pattern(s) would be followed when performing that task
   - See what features of the tool would be used to perform that pattern(s)
   - If (features used so far covers functionality that wants to be tested) then go to 4

   3.3 If (features used so far covers functionality that wants to be tested) Then go to 4
4. End

TABLE 4.4 Initial procedure for determining the tasks

Also, the tasks had to comply with the following rules:

- Involve exploration of a real Company system.
- Be linked to a real scenario of a maintenance activity.
- Be possible to perform in less than two hours.
- Its solution should cover the patterns as much as possible.
- Its solution should cover as much functionality of the tool as possible.
- Its solution should involve critical features not altered by the heuristic evaluation.

However, we encountered the following difficulties:

a) The examples of real scenarios and tasks given by the SEs were not useful for determining the tasks. As mentioned above, those examples were extremely

complex and focused on very high level goals. Also, SEs had not been able to say what sub-tasks or steps would be necessary to perform for solving the examples, therefore:

- It was impossible for us to know if the participants would use the tool for solving those examples.
- It was almost impossible to see what patterns participants would follow when solving the examples.
- It was very difficult know what and how the tool features might be used.
- Finally, many of the examples were not possible to complete in less than two hours.

In summary, it was impossible to know when and how participants would use the tool with those examples. In our case, knowing this was essential because we wanted to observe how participants used the functionality and we wanted to have some idea of the features that they would use. Consequently, most of the examples given by SEs were not helpful, as it had been thought. Based on this, we modified the procedure as shown in Table 4.5 below.

Both procedures differed in that the original could be seen as going *top-down* while the modified as *bottom-up*. That is, the original started considering high level scenarios and tasks, and went down trying to identify the required features in the tool for solving the tasks. The modified, instead, started identifying real sequences of features that SEs perform and went up trying to link those sequences with real high level scenarios and tasks.

1. Get an experienced SE in the tool and its domain
2. Take the collection of patterns found in the study of SEs work practices
3. While (there are patterns)
   3.1 Take next pattern
   3.2 With the help of the SE, generate a task about something from the Company system that implements the pattern by using some features of the tool
   3.3 With the help of the SE, see if that task can be part of any existing scenario generated here
   - If (it can be part) Then
     ⇒ Include the task in that scenario
     ⇒ With the help of the SE, establish a possible order among all the tasks in that scenario
   - Else, with the help of the SE, generate a possible scenario where that task could be part of

   3.3 If (features used so far covers functionality that wants to be tested) then go to 4
4. End

TABLE 4.5 Modified procedure for determining the tasks

We found that the modified procedure was easy to follow and produced a set of tasks complying with all the rules. Also, as we saw later, the tasks generated by it worked very well during the tests. Nevertheless, it is important to mention that the patterns were key for using the modified procedure. They represented real *low-level* tasks that SEs perform and therefore served to determine real sequences of using tool features.

b) A second difficulty we encountered was that finding sections of the Company system that the tasks explored was hard. Based on our previous observations of SEs, the participants could be extremely familiar with particular parts of the Company system. So, we had to create tasks that involved exploring sections of the system that were not extremely known by all of them. Otherwise, it was

possible that a certain participant didn't have to use the tool for solving a task simply because he knew the answer already.

Finding those sections of code was very difficult. We selected sections of the code based on suggestions from other SEs, but we could never be sure that such sections were completely unknown to all the participants. In fact, it happened during the sessions later that some participants knew the sections we chose. In order to prevent from possible problems and force the participants to use the tool we used the following tricks:

- For tasks where the participant had to determine if something existed, we phrased them so they asked the participant to *count* the number of things found.

- We phrased all the tasks asking the participant whenever possible to see the answer(s) in the windows of the tool.

c) A final observation when generating the tasks was that some of the missing tool information would have been useful. The specifications, for instance, could have helped to understand better how the tool features supported SEs tasks, and therefore would have helped at identifying sequences of features for solving real SEs tasks.

3. Once generated, the tasks became very helpful for other purposes. Before the study, the KBRE project didn't have any examples of real scenarios and tasks of what the tool was used for during maintenance activities and that is why we had to generate some for the test. But, once the tasks had been generated, we noticed these became very helpful to developers. They started using them for testing solutions to the problems found in the heuristic evaluation as well as testing new design ideas of future tool features.

4.3 Thinking Aloud Usability Testing

4.3.1 Procedure Overview

Once having obtained the participants and tasks, the next part consisted of applying the thinking aloud usability testing technique following the procedure described in Chapter III. That is, the participants were asked to verbalize their thoughts while performing the tasks. The sessions were videotaped for later analysis. The tasks were given to them one at a time. An observer watched the participants and helped them when necessary. The observer took notes about usability insights noticed during the sessions. These notes were used later during the analysis of the videos. Once finishing the tasks, the participants were asked to fill a short questionnaire and debriefed for some minutes. The sessions were performed at the Company facilities using computer equipment similar to that with which the participants normally work. Novices received a 15-20 minutes tutorial before the session given by a SE member of the group supporting the tool within the Company.

A pilot study was performed before the sessions with the participants. Two developers of the tool participated in this test. They were asked to perform the tasks generated for the experiment. They had extensive knowledge of the tool and exploring software systems but no knowledge about the Company systems. Their sessions followed the same methodology described above but they didn't complete the questionnaire, were not debriefed and didn't receive the tutorial. This test revealed the necessity to make some changes to the tasks and a second version of them was created. Some new tasks were included in order to exploit more tool functionality, and some tasks were rephrased due to observed difficulties with their understandability.

Next, the first two participants performed their sessions. These were one novice and one expert participant. These sessions revealed certain things not noticed with the pilot study that made it necessary to do certain changes to the tasks again. Thus, a third version of them was created. This was the final version of the tasks and can be consulted in Appendix 3.

At this moment, the rest of participants performed their sessions. They were asked to do the third version of the tasks. It is important to mention that although the first two sessions were done using the second version of the tasks, they were fruitful in capturing many usability insights about the tool and those were included in the final analysis.

Once finishing the sessions with all the participants, the next step was to merge and summarize the information in a report. The annotations, videos and questionnaires provided useful information that was included in the report.

Finally, like in the heuristic evaluation, some sessions were scheduled so that developers and experimenter reviewed the problems together. For some problems, short video clips about them were presented. The clips showed the participants experiencing the problems.

As stated in the plan and told to participants, the videos and other information about them were all kept confidential. Only people involved in the usability study had access to that information. Also, only clips showing specific problems were watched during the sessions with developers.

4.3.2 Usability Insights obtained with the Thinking Aloud Usability Testing

Like with heuristic evaluation, much information about the usability of the tool was found with thinking aloud usability testing. Most of that information was usability problems that participants experienced with the tool but there was also information about user satisfaction, possible improvements, and insights about the utility of the tool. Also, this technique provided information about how expert participants were capable of using the tool functionality.

A total of seventy-two usability problems were identified and those can be consulted in Appendix 5. Thirty-eight of them (53%) had already been identified in the heuristic evaluation and were confirmed in this technique. Thirty-eight of the problems (53%) were about difficulties in learning the tool, fifty-four (75%) were about efficiency of use, and twenty (28%) about both (notice that a problem can be about learning and efficiency of use at the same time).

Learning problems were mostly about difficulties for (a) perceiving that the tasks could be achieved by using the tool, (b) finding all the necessary features to complete a task, confusions due to (c) misleading information provided by the tool and (d) behavior of the tool different than expected. Two examples of these problems are:

- ***Problem:*** *Much of the power of the tool resides in combining functionality between two of the windows. However, it was very hard for participants to learn how to combine that functionality in order to solve a task (Difficulty in noticing that goals can be achieved).*

- ***Problem:*** *The Delete sub-list worked so that it deleted the objects below in the hierarchy of another object. However, participants tended to see a sub-list as a subset of consecutive objects and not as subset below the hierarchy (Behavior different than expected).*

On the other hand, efficiency of use problems were mostly related with obstacles and difficulties for achieving the tasks in optimal ways. Many times, the participants were able to achieve the tasks but they made many unnecessary or non-optimal steps. One example of these problems is:

- ***Problem:*** *The pop-up window of a searching feature often hid the matches found. Whenever that happened, participants had to move that window in order to see the match (Efficiency of use problem).*

From the seventy-two problems, five (7%) were experienced by the eight participants, ten (14%) were experienced by more than six, and thirty-one (43%) were experienced by less than three. Two examples of these problems are:

- ***Problem:*** *In one of the windows, different objects could look exactly the same. Whenever that happened, participants had to select each one of them in order to be able to see the differences (Problem experienced by all participants).*

- ***Problem:*** *The right numbers (numeric keypad) in the keyboard didn't work in the tool (Problem experienced by only one participant).*

Novice participants experienced forty-five problems (62%) and expert participants experienced forty-eight problems (66%). Also 20% of the problems were only experienced by novices, 7% were only experienced by experts, and the rest (73%) were experienced by both. Most of the problems experienced by novices (75% of them) were about learning the tool. Most of the problems experienced by experts (81% of them) were about efficiency of use. Some examples of these problems are:

- ***Problem:*** *Many important features were within middle mouse menus, but middle mouse menus were extremely hard to discover by participants (Learning problem affecting mainly novice participants).*

- ***Problem:*** *The Auto-grep feature serves to find occurrences of one object within another. It powerfully implements one of the most common SE tasks, which was noticed during the study of SEs work practices. However, the implementation was so non-intuitive and difficult to learn that none of the participants already knew how to use it and it hadn't been used yet (Learning problem affecting both participant groups).*

- ***Problem:*** *Nested procedures/functions in the source code were not clearly identified. Therefore participants had to find them manually by observing the code (Efficiency of use problem affecting mainly expert participants).*

- ***Problem:*** *The tool didn't have undo facilities and participants experienced situations where erroneous actions couldn't be reversed (Efficiency of use problem affecting both participant groups).*

According to the same classifications used in the heuristic evaluation, the problems were distributed as shown in Tables 4.6 and 4.7 below. Like in the heuristic evaluation, the biggest category of problems consisted of violations to the *simple and natural dialogue* guideline (Table 4.6). Its problems accounted for 61% of the problems found and most of them (48%) were about *non-optimal functionality.* The second biggest category of problems (22%) consisted of violations to the *good feedback* guideline. Some examples of these problems are:

- ***Problem:*** *Pop-up menus worked so that they stayed only if the mouse button remained pressed. Normally, participants required time to look and explore the features in those menus. Many times, participants were observing a menu and accidentally released the mouse button, which led them to lose the menu. Also, participants had difficulties with (a) keeping the mouse button pressed for a long time, and (b) keeping it pressed while moving the mouse in order to select something (Non-optimal functionality problem).*

- ***Problem:*** *The tool did not always presented feedback when there were no results from an action. Many participants didn't easily notice when an action with an empty result had finished. Sometimes they scrolled the windows trying to identify any possible changes to the information. Some executed the action again several times to convince themselves that the action had had an empty result. Participants started developing the concept of not trusting the feedback given by the tool (Feedback problem).*

Usability Guideline	No Problems
1. Simple and natural dialogue	44
2. Speak users' language	9
3. Minimize users' memory load	0
4. Consistency	1
5. Good feedback	16
6. Clearly marked exits	2
7. Shortcuts or accelerators	0
8. Good error messages	0
9. Good error prevention	0
10. Good help and documentation	0

TABLE 4.6 Problems found in the Thinking Aloud Usability Testing according to the usability guidelines

Category	No Problems
1. Non optimal functionality	23
2. Users' confusion	6
3. Users' misinterpretation	8
4. Unnecessary functionality	2
5. Lack of functionality	7
6. Lack of consistency	1
7. Graphical design	0
8. Feedback	16
9. Lack of robustness	1
10. Wrong behavior	1
11. Bad labeling	7
12. Lack of labeling	0

TABLE 4.7 Problems found in the Thinking Aloud Usability Testing according to the second classification

An analysis of the problems revealed that many were consequences of the lack of systematic usability activities during the development of the tool. For example:

a) Many problems were related with not efficiently considering the context in which SEs worked in a particular moment. We noticed that depending on the context participants performed different sequences of steps to accomplish the same task. However, the tool provided ways to solve tasks that were not context sensitive,

consequently, the tool sometimes helped participants efficiently and others times not. It is possible that more knowledge about how SEs worked depending on the context could have enabled the creation of a context sensitive tool design.

b) In the tool, some features worked differently depending on the objects selected. Hence, some problems were because the tool did not inform participants efficiently about (a) the objects selected in a particular moment and (b) how certain features would work based on that selection. These problems would have been detected before if the tool had been tested with SEs.

c) Some problems were related with not providing support in all the steps involved in completing full SEs' tasks of exploring the Company systems. It is likely that the use of scenarios of complete SEs goals for designing the tool would have avoided many of these problems. Furthermore, many of them had not been detected because, as we will discuss later, developers had been testing the tool with low-level SEs tasks only.

The questionnaire and debriefing were also productive. They revealed aspects about user satisfaction, utility of the tool and how it had been used so far. The number of participants was not big enough to generate a reliable statistical analysis. But the results were used to complement the other information and obtain a more accurate picture of the status of the tool. The questionnaire and average of the answers were as shown in Table 4.8 below. Among the most important insights were:

- Participants considered that the tool was easy to learn although learning took time (something we noticed SEs didn't have too much of).
- Participants considered that the tool was useful at exploring the software systems, but only once certain practice with it had been acquired.
- Participants considered that the user interface of the tool was easy to use although it was not extremely efficient.

Question	Answer
1. Do you consider that learning the tool is:	
a) **0** = difficult, **4** = easy	**3.0**
b) **0** = confusing , **4** = clear	**2.4**
c) **0** = takes long, **4** = very fast	**2.6**
2. How well do tool menus, options, etc. represent real things or actions that you need for exploring software systems like those you work with at the Company ? **0** = not good, **4** = very good	**2.6**
3. How well does the tool help you on typical sequences of actions for exploring software systems like those you work with at the Company ? **0** = not good, **4** = very good	**2.9**
4. How much are the tool features enough for your purposes of exploring software systems like those you work with at the Company ? **0** = 0%, **4** = 100%	**2.9**
5. Do you consider that the tool interface (menus, windows, buttons) is:	
a) **0** = difficult to use, **4** = easy to use	**3.1**
b) **0** = awkward to use, **4** = convenient to use	**2.8**
c) **0** = slow to use, **4** = quick to use	**2.5**
d) **0** = inefficient to use, **4** = efficient to use	**2.8**
6. How helpful is the tool for you on maintenance tasks of software systems like those you work with at the Company ? **0** = not helpful, **4** = very helpful	**3.1**

TABLE 4.8 Answers given by participants in the questionnaire

In general, the opinions were very homogeneous (standard deviation *"stdev"*, around 0.5). Other interesting insights were:

a) Novice participants considered that learning the tool was *a bit difficult* (1.8) but experts considered it *easy* (3). This was probably because experts had already learned the tool and they were not experiencing many learning difficulties anymore.

b) The opinions of whether the tool was helpful at typical sequences of actions for exploring the system were very consistent in novices (3.3, *stdev* of 0.5) and very inconsistent in experts (*stdev* of 1.3).

c) Opinions of novices and experts also differed about the efficiency of use of the user interface. Experts were consistent in their answers (2.5, *stdev* of 0.5) while novices did not agree on them (*stdev* of 1.1).

An interesting insight arose from one of the expert participants who had learned the tool since it was created. He considered that the version used in the testing was not as helpful as some previous versions. He had learned how to perform his tasks on the first versions of the tool, and differences in the version used in the test (the latest version) had forced him to relearn how to accomplish the tasks on it. Also, not all the tasks possible to do in the previous versions could be done in this version. This made us see that changes in the features of the tool had not been properly analyzed and had affected the SEs. Also, this was another indication that SEs had not been monitored.

All the tasks were completed in the ten sessions (including the pilot study). However, certain tasks were completed in very inefficient ways. That is, not in the best way possible to complete them with the available tool functionality. This indicated that novices were not able to discover the best resources of the functionality and experts did not know many of them yet. For example, some tasks required looking for all the occurrences of a particular string in many files. The tool provided a feature for solving such task in one step. However, none of the participants used that feature and, instead, completed the tasks by using another feature that found occurrences of a string one by one within one file only. So, not knowing about the more powerful feature made that participants completed the tasks using the less powerful feature many times.

These observations along with information obtained from the debriefs with the expert participants, made us see that none of the experts knew all the tool functionality yet and no one had used many of the most advanced features in it. Therefore, because these participants were all the SEs who had used the tool during maintenance activities, that meant it was impossible for the KBRE project to assess the functionality of the tool at that moment. In other words, nobody in the group of SEs who were given the tool had used all the power of the functionality during maintenance activities yet.

The most common reasons why SEs had not used all the functionality were:

a) They had not discovered many features.
b) They had not understood the meaning of certain features by looking at its label.
c) They had not understood the way certain features had to be used. Certain features required to be used in a peculiar way or order. So, each time that SEs had tried those features they had been unlucky and had employed them in an incorrect way.

An interesting observation was that many things intended for experienced users of the tool, such as keyboard shortcuts, were rarely used (that's why there were not many problems with them). This was another indication that expert participants had not used all the functionality yet.

All these made us see how not having considered usability had caused poor learnability in the tool design, which in turn had caused that SEs never learned and used all the functionality during the past 1.5 years in which the tool had been available. Thus, making impossible for people in the KBRE project to see whether the experimental functionality of the tool could help SEs to achieve their goals more efficiently or not. In other words, making it impossible to assess the functionality.

On the other hand, participants expressed a positive attitude towards the tool. Novice participants commented on advantages they noticed between having the tool and manually exploring the Company systems. Expert participants commented on having received some benefits from the tool, particularly those who had used the tool for learning the Company systems. However, based on our observations about SEs needs we believe that if expert participants had known all the benefits possible to get from the functionality, they would have probably had a much better opinion about the utility of the tool. None of them had been aware of all the possible benefits so far.

We want to mention that at the end of each session, we instructed participants on how they could have achieved the tasks very efficiently. We saw that they easily learned the features involved in solving the tasks and made comments such as: *"if at least somebody had told me about that feature before, I would have been using the power of this tool since long time ago"*. This made us realize that besides good learnability, other means for helping SEs at learning the system would have been very helpful for the KBRE project.

In general, the application of this technique greatly helped participants at learning the tool. Solving the problems posed by the tasks and instructing them on efficiently using the functionality for achieving those tasks made that they learned the core functionality of the tool and realized great benefits possible to get.

During the study, we saw that participants encouraged other SEs to start using the tool. This situation was very well received by the KBRE project, which had always been seeking to have SEs using the tool. This made us see that good usability could have generated positive attitudes in SEs learning the tool and those would have encouraged other SEs to learn it and use it, which in turn would have helped in the assessment of the functionality.

Finally, because nobody knew well all the tool functionality, it is likely that many usability deficiencies were not captured, particularly about efficiency of use. Finding more of this type of deficiencies required testing with SEs who knew the functionality better. This made us see that having the same participants from this test in other future tests could reveal more insights about efficiency of use.

4.3.3 Issues during the Thinking Aloud Usability Testing

The thinking aloud usability testing was in general easy to plan and perform but like in the other techniques, there were certain things particularly more difficult or different than expected. We encountered the following issues during this technique:

1. Planning and applying this technique took much longer and was more difficult than the heuristic evaluation. There were many more practical issues and details to deal with than in the heuristic evaluation. Many were in regards to using the video and audio equipment, performing and controlling the sessions, and analyzing the information. We noticed that insufficient experience with the technique was the main cause of this.

2. The pilot study was essential and very productive. It revealed some problems with the tasks and many other practical issues about the application of the technique, such as the use of the video equipment, difficulties with testing at the normal Company facilities, etc.

3. There were some interesting insights during the sessions with the first two participants.

a) We found important differences between these participants and the developers that participated in the pilot study. We had thought that developers of the tool were similar enough to the participants and therefore appropriate for the pilot study. Developers were SEs like the participants and also had knowledge in exploring software systems (the tool domain), therefore they should behave similar to the participants. However:

- Participants had much more knowledge about the Company systems than developers and that made them see several different ways for solving the tasks, including some in which the tool was not necessary. Developers, on the other hand, found the tool as the only way to solve the tasks. This made us confirm our observations that in order to force participants to use the tool, the tasks had to be cautiously selected and phrased, otherwise, they could have solved the tasks without using the tool. We found that the tricks used for writing the tasks had worked well, but as we will discuss next, certain adjustments to them were done.

- Participants were harder to control and it was more difficult to make them focus on the tasks than developers. They asked many more questions during the sessions than developers. When participants found the answer of a task, they tended to turn and look at the observer, express the answer to him and then wait for a gesture from him. Consequently, we realized that the tasks had to be phrased so that the participants could talk while performing them but not requiring (as much as possible) help from the observer. In other words, phrasing them in a way that the communication between observer and participants was kept at a minimum. Based on this we adjusted the tricks for writing the tasks as follows:

  - For tasks where the participant had to determine if something existed, we phrased them asking the participant to *write down* the number of things found and sometimes its names (depending on how long the names were). This included writing *"none"* in case of not finding anything.

  - For tasks where the participant had to reach certain conclusion, we phrased them asking the participant to write down his conclusion (whatever that was).

  - We phrased all the tasks asking the participant whenever possible to see the answer(s) in the windows of the tool.

  These new ideas were used to modify the tasks and worked remarkably well in the rest of the experiment.

These differences between participants and developers made us realize that testing with SEs from the group who had been given the tool was essential. We found that knowledge in the particular Company system being explored with the tool had an important effect on participants' behavior. Also, knowledge in exploring the Company systems for maintenance purposes was very important. Just having knowledge in exploring software systems in general (the tool domain) was not

enough to consider a certain SE as similar enough to the SEs who had been given the tool.

b) We found important differences between the participants in these first two sessions. The first participant had been identified as belonging to the expert category after the user and task analysis. He was a SEs who had been given the tool but he did not belong to the same Company group of the other participants. By looking at how he used the tool and the comments he made, we noticed that the tool was not very useful for him and probably neither to others in his group. This made us confirm the observations that differences among SEs in different Company groups made the tool not to be equally useful for all the groups.

On the other hand, we noticed that he was not really an expert in the tool as he had expressed during the interviews. Therefore, we had to consider his results as from a novice participant. Hence, the study ended having five novice and three experts participants instead of four in each category. This made us see that the real level of proficiency that he had with the tool had not been truly discovered in the user and task analysis but until we watched him using the tool.

These two previous insights (a and b) made us see that the real target population of users of the tool was not *'those SEs performing maintenance activities that followed the Just in Time Comprehension approach'* as it had been originally defined. But rather, those SEs that performed exactly the type of maintenance activities of the particular group of SEs from which the tool had been mostly based.

4. Two hours was about the maximum time for good sessions. The average times of the sessions were 1:32 hrs for novices (not including the tutorial) and 1:13 hrs for experts including the questionnaire and debriefing. We noticed that longer sessions would have been uncomfortable for the participants.

On the other hand, the tasks obtained with the modified procedure worked very well. All the tasks could be completed in less that two hours and, as we had seen, many tasks typical of what the tool was designed for could have taken much longer than that.

5. All sources of information used in the technique provided useful and different types of data. The annotations, questionnaire and debriefing served to capture usability problems with the tool but were particularly good for capturing comments from the participants about how much the tool had been used so far and how useful it had been for them. It was possible to relate usability problems with participants' opinions. These comments revealed aspects about:

a) Participants' expertise with the tool.
b) Participants' attitudes towards the tool.
c) Participants' problems beyond the scope of the tool that had affected its use.
d) Strengths and weaknesses of the tool according to real goals of participants.
e) Utility of the tool according to participants' Company group.

The videotapes, on the other side, were extremely helpful at confirming aspects captured on the annotations and revealing many usability problems not observed during the sessions.

6. We noticed several issues regarding videotaping:

a) Having the videotapes was extremely helpful. We noticed that usability sessions can be very difficult to conduct by only one person and especially if that person is not yet a usability expert. In our case, videotaping released some pressure in capturing usability problems and gave us more resources for controlling the sessions.

b) On the other hand, the analysis of the videos was tiring and very time-consuming. We had to spend about 3 to 5 times the duration of a video in order to analyze it. Videotapes of novices lasted an average of 1:14 hrs and videotapes of experts an

average of 0:56 hrs, however, the total time spent analyzing the tapes for all the eight sessions was about forty hours.

7. Like in the heuristic evaluation, the number of problems was big again and that increased the concern of people in the KBRE project about the usability of the tool and its consequences.

8. Like in the heuristic evaluation, the sessions between experimenter and developers were extremely helpful. Again, we confirmed that having the report properly written had been essential. However, the report alone would have not been enough to communicate all the information to developers. Instead, the sessions served to:

- Watch video clips of the participants struggling with the tool. We found this was extremely effective. Developers easily understood the problems and developed commitment to solve them.
- Understand the problems and prioritize its solutions.
- Experiment with different procedures for presenting the problems and video clips to developers. The one we found most effective was:

    1) *Ask developers to read a problem.*
    2) *Establish a short discussion about it.*
    3) *Show the video clip(s) presenting the problem.*

Again, the experimenter played a key role in these sessions. He had to actively help developers in many issues.

9. Following the policies of the study about confidentially created a feeling of professionalism and commitment in everybody around it, including developers, participants and other people in the Company. We noticed that explaining the procedures and policies to everybody before the study, and then following them carefully had been essential.

10. The thinking aloud usability testing was an excellent step after the heuristic evaluation to continue introducing usability and user centered design into the KBRE project. It increased developers' awareness about the usability status of the tool as well as their knowledge in usability. Like in the heuristic evaluation, we saw that the KBRE project matured more in its attitude towards user centered design. By the end of this technique, it had passed from the *curiosity* stage to the *acceptance* stage, as described in Chapter II.

4.4 Other Insights and Issues independent of the Techniques

Up to now we have presented the particular results obtained with and during the different techniques applied in the study. Next, we present other results independent of the techniques.

1. In total, the tool presented a large number of deficiencies. One hundred and forty-eight different usability problems were found in the tool. According to the usability guidelines, the problems in both techniques were distributed as shown in Figure 4.1 below.

Most problems were related to (a) *simple and natural dialogue*, (b) *speak users' language*, (c) *consistency*, and (d) *good feedback*. Many of these were effects of having designed the tool with insufficient information about complete SEs' tasks and working environment as well as user interface guidelines and standards. Many problems were the type of problems that normally appear on a first usability evaluation. We found that many could have been easily avoided with the application of certain user centered design activities, such as having used complete scenarios and tasks to drive design, having evaluated tool with prototypes and having followed user interface standards and guidelines more systematically.

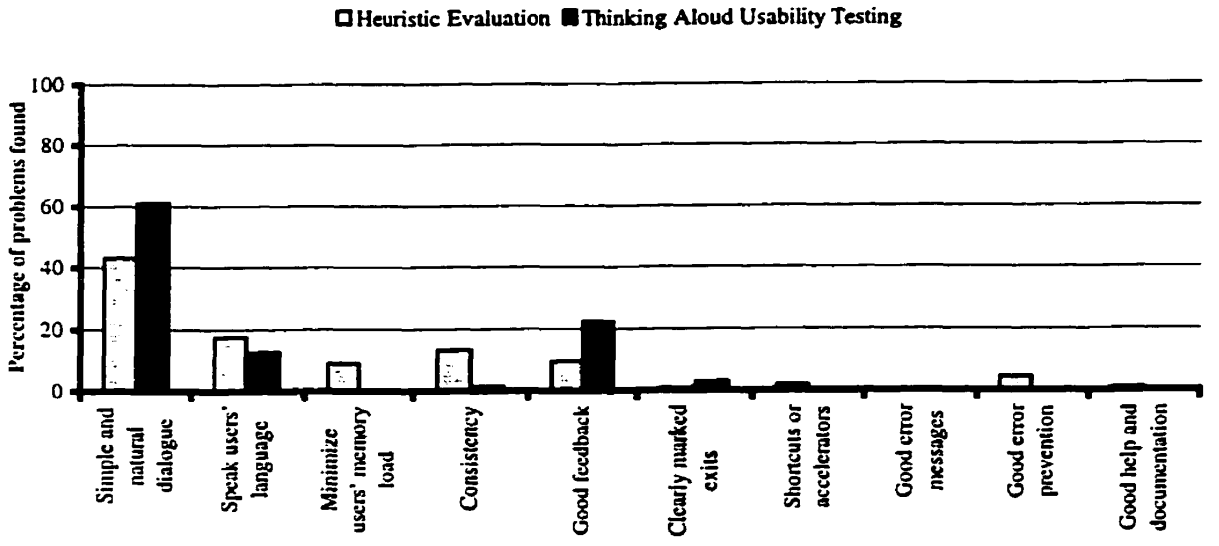


FIGURE 4.1 Problems according to the usability guidelines

Ninety-three problems (62%) were about deficiencies in learning the tool, eighty-seven (59%) about efficiency of use, and thirty-two (21%) about both (Figure 4.2).

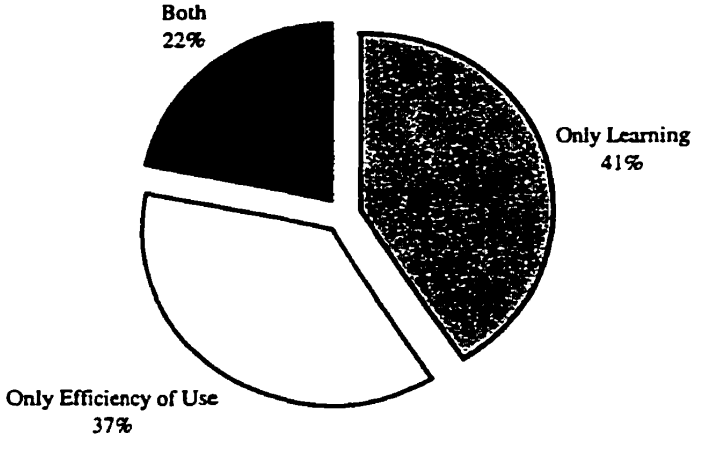


FIGURE 4.2 Types of problems found using both techniques

These problems indicated that the tool design did not facilitate learning and using efficiently its functionality. In fact, as stated earlier, those problems had already affected SEs. None of them was capable yet to efficiently exploit all the functionality.

2. Not all the problems meant that something in the tool had to be fixed. The results indicated that good tool learnability and other means for helping SEs to learn the tool were key elements for assessing the functionality. Many problems existed simply because participants didn't know certain information, but once they knew it, the problems didn't exist anymore. We noticed that developers realized that some learning problems did not necessarily mean that something in the tool had to be fixed, instead, including some information in a tutorial or training course, or in a manual or on-line help could possibly solve the problem. Hence, people in the KBRE project started thinking on other possible means for helping SEs to learn the tool besides improving its learnability, such as training the SEs with some courses or practical examples.

3. Testing the tool without the intention of solving high level SEs goals had led to not detecting certain deficiencies in the tool. We noticed that developers tested the tool with typical tasks that SEs do for exploring the Company systems, however, when they performed such tasks they never had the intention of solving some high level goal that SEs normally had. This phenomenon was also observed with the SE who gave the tutorials to novices. In the sessions, we had observed that with different goals in mind, participants (e.g. SEs) solved the same task using different sequences of features. Consequently, not having high level goals in mind had made developers always complete a task using the same sequence of features, thus, not exploring other possible paths of using the features that SEs could perform, and ultimately, not detecting problems with them. In fact, we noticed that the sequences of features used by developers to solve the tasks tended to encounter many fewer *'bugs'* than others used by participants to solve the same tasks. This certainly showed that sequences used by developers had been already cleaned of errors.

On the other hand, not having a high level goal in mind when solving the tasks during the demos had only shown a partial power of the functionality. That is, demos had only shown how the tool could be useful under particular circumstances.

4. Insufficient communication between the KBRE project and SEs had been a cause seriously affecting assessing the functionality. The tool had been given to a group of SEs so that they used it during maintenance activities. On one side, poor usability in the tool had affected some of those SEs from exploiting the functionality efficiently. But, on the other side, a large number of SEs had not used the tool at all. These last SEs had obviously not been affected by poor tool usability but had not used the tool for other reasons, such as ignorance that it existed or was available, low motivation or limited time for trying it, etc. On the other hand, the project had not monitored the SEs much, and therefore, they never had known the status exactly.

We noticed that a permanent channel of communication between the project and SEs was key for having a group of SEs using the tool. The project had not established such channel for several reasons, such as, ignorance about its importance, lack of people involved in it, insufficient time and resources, etc. We found that the usability study worked extremely well for providing that channel.

5. Combining heuristic evaluation and thinking aloud usability testing worked excellent for obtaining information about the usability of the tool. One hundred and fourteen problems were found with the heuristic evaluation and seventy-two with the thinking aloud usability testing. Thirty-eight of the problems found with the heuristic evaluation (33%) were confirmed with the thinking aloud usability testing (Figure 4.3).

Both techniques complemented each other very well. Each one provided different types of problems. Also, by using both, we could confirm certain problems and obtain a more reliable severity of them.

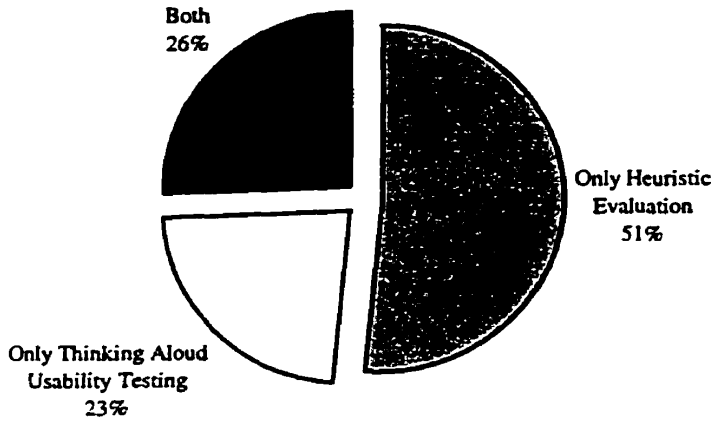


FIGURE 4.3 Percentage of problems found by the techniques

6. The study, as performed, worked very well for introducing usability and user centered design into the KBRE project. We noticed that applying two evaluation techniques took longer but produced a much better effect in the project. Both usability evaluation techniques worked well but in particular thinking aloud usability testing.

Although late in the tool lifecycle, having introduced usability at that time was worthwhile. The study revealed problems and issues unfeasible to solve, but it helped the project seeing how to proceed for assessing the functionality. Before the study, the project was facing the problem that SEs had not used the tool much and it was not known exactly why and what to do.

By the end of the study the project had passed from the *skepticism* stage to the *acceptance* stage, as described in Chapter II. We noticed that people in the KBRE project became aware of the following issues:

- Assessing the functionality at that time was not possible yet.
- It would be necessary to have a group of SEs efficiently using the functionality in order to assess it. Therefore, certain activities should be planned for helping SEs

to learn the functionality. They realized that usability studies could be one such activity. Also, it would be important to monitor SEs more closely.

- Solving the problems in the tool would indeed improve its usability but that would be only a first step towards achieving good usability. They realized that (a) problems captured this time were probably not all the existing ones and (b) solutions to the problems found should be tested with SEs. They saw it would be necessary to have more iteration of evaluation and redesign.
- Any new functionality should be prototyped and evaluated before being implemented.
- The scenarios and tasks generated for the study should be kept and used for testing solutions to usability problems, new design ideas and in any other future usability studies.
- It would be necessary to have people in the project helping with usability. They realized they didn't have the skills nor the time for that.

CHAPTER V
Conclusions

The present chapter presents the main conclusions and lessons learned from our research done through the usability study of the tool. These conclusions are organized according to the issues we were interested in the research. We provide as well a series of recommendations based on such conclusions and other insights from the research.

5.1 Detailed Conclusions

Following are the conclusions according to the three issues we were interested in the research (section 1.2).

5.1.1 Objective One

In regards to possible effects of not having considered usability when assessing the functionality of a system, we found the following things:

1. Not considering usability can affect for the assessment of the functionality. The usability study revealed that there was not a group of SEs that could use all the power of the functionality of the tool at that moment, and therefore, it was impossible for the KBRE project to assess such functionality. In other words, it was impossible to determine if the functionality was very good already for helping SEs at maintaining the Company systems more efficiently or if it should be improved. This group of SEs didn't exist for several reasons. On one side, problems such as, ignorance that the tool existed, low motivation, limited time, etc., had caused that many SEs did not use the tool at all. But, on the other hand, not having considered usability had meant that the tool presented a

large number of usability deficiencies, which in turn, had made that those SEs who had used the tool were not capable of using all the power of the functionality. In other words, bad usability in the tool meant it was not possible to assess the functionality at that moment.

2. Helping users learn the system is essential for assessing its functionality and good usability plays a key role, particularly good *learnability*. As we mentioned above, there was not a group of SEs that could use all the power of the functionality at that moment. It was clear that just having left the tool available to SEs during 1.5 years had not been enough so that such a group of SEs emerged. Also, not having monitored how SEs used the tool had meant that the KBRE project never realized causes affecting the use of the tool. In summary, it was clear that in order to have a group of SEs capable of using all the power of the functionality it would have been necessary to help the SEs at learning the functionality. For that, good tool *learnability* would have been key.

3. Usability testing can be an excellent activity for helping users learn the system. We saw that the usability study served as an excellent communication channel between developers and SEs. On one side, it meant that more SEs learned and used the tool better. Participants learned to exploit the power of the functionality by doing the tasks of the test, and they encouraged other SEs to learn and use the tool as well. On the other side, the study helped the KBRE project at discovering causes affecting the use of the tool as well as how to proceed in ensuring that a group of SEs could be capable of using all the power of the functionality.

4. Ideally, usability should be considered from the beginning of a project, but considering it late is worthwhile and much better than never considering it. Despite all the problems and effects that not having considered usability had caused, having started usability activities at the time of this research produced immediate benefits. The KBRE project was facing the problem that not many SEs had used the tool and it was unknown exactly why and what to do. As we mentioned before, the usability study exposed many of the causes of that problem and helped at seeing how to proceed next. After the study,

developers had a much clearer idea of what kind of things they should pay attention to in order to improve the tool. It is important to mention that the advanced status of the tool within its lifecycle meant that not all the problems and things revealed by the study could be solved or improved.

5.1.2 Objective Two

With regard to experimenting with the introduction of user centered design and usability into the latter stages of software projects in which developers are new to these concepts, we made the following observations:

5. Usability must be gradually introduced by following a process. Throughout the study, we confirmed that working towards good tool usability definitely required commitment and participation of many people in the KBRE project, including developers, SEs and managers of both. Also, it required certain level of developers' maturity with usability, its effects and how to address it. But we saw as well that people in the project required time and being exposed to different types of activities in order to acquire that maturity and commitment to usability. In summary, we confirmed that it was necessary to use a gradual process including different types of activities to introduce usability into the KBRE project.

6. Initial activities must focus at creating usability awareness in the software project (maybe sacrificing obtaining the best possible usability results). As we mentioned above, working towards good tool usability required developers' maturity and commitment to usability. After the study, we saw that performing the best techniques to obtain the best usability insights was not as important before the study as developing such commitment and maturity. We found that making people in the project become aware of (a) what usability was, (b) the usability status of the tool, and (c) how it had affected the SEs, generated the most powerful effect for developing maturity and commitment to the usability of the tool. The best activities had been evaluating the tool, fixing tool usability

problems and testing the tool with SEs. However as well, insufficient maturity of the project at that time meant that these activities not always produced the best effects with regard to the usability of the tool. At that moment, developers were only becoming aware of what usability meant and how difficult achieving good tool usability was.

7. Creation of scenarios and tasks of system usage is very helpful. We confirmed that examples of scenarios and tasks reflecting the tool's design intent produced many benefits. They served to perform the testing but also greatly supported developers in testing the solutions of the problems and testing designs of future features.

8. System requirements and specifications as well as user interface information are very important. We saw that knowledge on (a) the tool domain, (b) how it addressed the problem, (c) the rationale of its design, and (d) how it had to be used, was essential for developers when trying to improve the tool. Not having means to completely acquire that knowledge seriously affected new developers that had recently come into the KBRE project. We saw that system specifications and user interface information would have been particularly helpful.

9. Experience introducing usability is an important requirement for the usability specialist, particularly when developers are new to usability. Insufficient knowledge about usability meant that developers of the tool had difficulties:

- Understanding usability problems in its most general way.
- Seeing implications of not efficient solutions.
- Evaluating effectiveness of different solutions.
- Prioritizing the problems.
- Deciding how to proceed on fixing the problems.

Therefore, the person performing the study had to support developers in all these issues. We found that this type of support was not always easy to provide; producing the best results definitely required experience dealing with the above issues.

10. Developers have difficulties generating good solutions the first time they solve usability problems. The above difficulties meant that developers not always devised good solutions to the problems. Sometimes, the solutions would lead to other obvious usability problems. This situation along with the state of development of the tool, limited time and insufficient resources meant that developers did not always implement good solutions.

11. Once exposed to usability, software projects can quickly mature about accepting usability. During the study, the KBRE project passed from the *skeptical* stage to the *acceptance* stage, as described in Chapter II. In all the study, we never noticed anything that could have caused the project to reject or disagree with usability. Basically the only reason why usability had not been considered was insufficient knowledge. Every time people in the project learned something new, they were more convinced and committed to usability.

5.1.3 Objective Three

In regards to exploring particular challenges of studying the usability of program comprehension tools, we found the following things:

12. Domain knowledge is an important requirement for the usability specialist. The tool domain was so particular and complex that knowledge in it was important for all the usability specialists involved in the study. On one hand, evaluators in the heuristic evaluation required this knowledge to be able to make accurate judgments about the tool. In fact, insufficient domain knowledge posed difficulties for one evaluator. On the other hand, the experimenter in the study had to provide support to this previous evaluator on several issues about the domain. Also, he had to fully understand the problems, appropriately explain those in the report, communicate them to developers, and provide advice during the generation of solutions. All these required domain knowledge on the part of the experimenter.

13. Testing with real users is essential. It is well known that that the more similar the participants in a usability study are to the users of the system the more accurate the results of the study will be. As we mentioned above, the tool domain was so particular and complex that the population of users of the tool (SEs) was very specific. In the study, we saw that people with very similar characteristics to the SEs behaved very differently to those when approaching the problems that the tool was designed for, and therefore they used the tool very differently. We noticed that testing with real users of the tool (SEs) was even important for the pilot study.

14. Task determination becomes particularly difficult and complex. Like in the previous issue, the complexity of the tool domain meant that SEs could approach the problems very differently and in very unpredictable ways. This implied that finding suitable tasks for the study that made them use the tool functionality in a more or less predictable way was very difficult.

5.1.4 Other Conclusions

We also found the following things:

15. In regards to heuristic evaluation:

a) Using evaluators with usability knowledge is important. This knowledge was important so that they knew what to do and look for in the evaluation. However, we noticed that people with this knowledge were not easy to find.

b) Better results can be obtained by combining evaluators with different levels of *domain knowledge* and *system expertise*. We saw that each type of evaluator captured different types of problems and this provided a broader spectrum of

problems. We don't know if this combination is always better in general. But this is a topic for further research where more experimentation could be performed.

16. In regards to thinking aloud usability testing:

a) This technique is more difficult than heuristic evaluation. Applying this technique required dealing with many more issues during its planning and application.

b) It is an excellent way to introduce usability into a software project. It worked excellently in creating awareness and commitment to usability in the KBRE project.

17. Combining both techniques is an excellent approach. Applying both techniques took us longer but produced a much better effect on introducing the KBRE project to usability and user centered design. Using the two techniques exposed the project more to usability, which made it mature and become more committed to it. Also, both techniques complemented each other very well. Each one provided different types of problems. By using both, we could confirm certain problems and obtain a more reliable severity of them.

5.2 Recommendations

This research led to some recommendations. The following set of recommendations is for software projects developing a system in which its functionality will be assessed. They were derived from the conclusions discussed in section 5.1.1 above.

1. Pay attention to both the utility and the usability of the system. Plan for resources and activities to look over both. Not paying attention to one could affect the assessment of the functionality and consequently reduce chances for improving the system.

2. Create a plan for ensuring that the group of users who is given the system correctly learn and use the system. Establish a permanent channel of communication between them and developers. Plan for activities to help them at learning and using the system. Monitor how they employ the system (see conclusions in section 5.1.1 above).

The following set of recommendations is offered for software projects that have started without usability activities but that want to initiate activities to look at and improve the usability of a system. They were derived from the conclusions discussed in section 5.1.2 above.

3. Establish a plan of action. If time and resources are limited, use a discount usability engineering approach. Important activities to include in the plan are:

   a) A user and task analysis. Clearly identify the target population of users of the system. Identify characteristics of those users that can affect how they use the system. Identify the tasks that the system is trying to help them with. Identify possible groups of users within the target population and their characteristics. Document all identified information.

   b) Identify and maintain information about the system such as system requirements, specifications and design rationale of the user interface. Emphasis should be on information that can help people at learning the system and its domain.

   c) Create a set of typical scenarios and tasks that reflect the system's design intent. Use identified information as help.

   d) Establish usability goals for the system. Consider pre-established objectives of the system and use identified information as help.

   e) Plan for iterations of evaluation and redesign. Iterate until reaching the usability goals.

4. As much as possible maintain a usability *"champion"* in the project.

5. Instruct developers on usability before starting the project. Also, plan for activities to continually improve the usability knowledge of people in the project.

6. If using heuristic evaluation:

    a) Try to get evaluators with as much knowledge in the domain of the system as possible.

    b) Provide support to evaluators with information about the system such as, requirements, specifications and design rationale of the user interface.

    c) Plan for sufficient time and resources to summarize problems found by evaluators.

    d) Plan for some sessions with developers to review the problems.

    e) Integrate evaluators into a pool of usability specialists for future evaluations of this or other systems.

7. If using thinking aloud usability testing:

    a) Use system information such as requirements, specifications and design rationale of the user interface as help for determining the tasks.

    b) Phrase the tasks in a way that the communication between observer and participants is kept at a minimum. Participants must able to talk while performing the tasks but should not require (as much as possible) help from the observer. This could be accomplished by asking participants to count

something, write down the answers, or observe something on the screens of the system.

c) Plan for at least three times the duration of videos for their analysis.

d) Plan for some sessions with developers in order to review the problems.

e) Integrate participants into a pool of participants for future tests of this or other systems.

5.3 Summary

The research project presented here investigated several issues. First, it explored effects of not having considered usability when assessing the functionality of a particular program comprehension tool. This tool had been created to provide certain experimental functionality that its developers wanted to assess. For that, the tool had been given to a group of users for some time. Second, the research experimented with introducing usability into the software project that was creating the tool at a late stage of the lifecycle. Finally, the research explored challenges posed to studying the usability of the program comprehension tool.

A usability study was designed and performed to pursue the research. Three techniques were applied in the study: (a) *heuristic evaluation*, (b) *user and task analysis* and (c) *thinking aloud usability testing*. Heuristic evaluation served to look for usability deficiencies in the tool design. For this, a group of evaluators judged the tool according to a list of usability guidelines and principles. Thinking aloud usability testing served both (a) to look for problems that users experienced when learning and using the tool, and also (b) to explore how users who had been given the tool were capable of using its functionality. Both were achieved by observing a group of users performing some tasks typical of the tool's design intent. User and task analysis served to obtain a group of users

and a set of tasks for the thinking aloud usability testing technique. It also served to gather information about how the users had employed the tool. The study followed a discount usability engineering approach.

The study was in general easy to plan and perform although certain things were more difficult than expected. Most difficulties were related to practical problems in planning and applying the techniques. It was difficult to find within the usability literature advice or examples of certain practical issues.

The study was very productive in finding information about all the objectives in the research within the particular software project. In summary, it revealed that none of the users who had been given the tool was capable yet of correctly using its functionality, and therefore, it was impossible to assess the tool functionality at that moment. Not having considered usability had been one of the causes of this. Also, the study exposed many issues in relation to (a) having started considered usability at a late stage in the tool lifecycle and (b) the software project's reaction to being exposed to usability for the first time. Finally, the study exhibited certain particularities and difficulties with planning and studying the usability of the program comprehension tool. In summary, the research project was satisfactory and accomplished all its objectives.

References

Bachman, D. (1989). ***A methodology for comparing the software interfaces of competitive products.*** Proceedings of the Human Factors Society 33rd annual meeting 1989, pages 1214-1217.

Bias, R., Mayhew, J. (1994). ***Cost-Justifying Usability.*** Academic Press.

Boehm, B., Gray, T., Seewaldt, T. (1984). ***Prototyping versus Specifying: A Multiproject Experiment.*** IEEE Transactions on Software Engineering 10, 3, pages 290-303.

Boehm, B. (1988). ***A Spiral Model for Software Development and Enhancement.*** Computer, May 1988, pages 61-72.

Ericsson, K., Simon, H. (1984). ***Protocol analysis: Verbal reports as data.*** MIT Press.

Denning, S., Hoiem, D., Simpsom, M., Sullivan, K. (1990). ***The value of thinking-aloud protocols in industry: A case study at Microsoft Corporation.*** Proc. Human Factors Society 34th annual meeting, pages 1285-1289.

Desurvire, H. (1994). ***Faster, Cheaper!! Are Usability Inspection Methods as Effective as Empirical Testing?.*** In Usability Inspection Methods. J. Nielsen and R. L. Mack Eds. John Wiley & Sons, New York, pages 173-202.

Ehrlich, K., Rohn, A. (1994). ***Cost Justification of Usability Engineering: A Vendor's Perspective.*** In Cost-Justifying Usability, D. J. Mayhew and R. G. Bias Eds. Academic Press, New York, pages 73-110.

Gaylin, B. (1986). ***How are windows used ? Some notes on creating an empirically-based windowing benchmark task.*** Proc. ACM CHI '86 Conf., pages 96-100.

Glen, David (1995). ***A case study on moving from a character-based full screen user interface to a graphical user interface.*** Master's thesis, California State University, Chico.

Gould, J. D., Lewis, C. (1985). ***Designing for Usability: Key Principles and What Designers Think.*** Communications of the ACM 28, 3, pages 300-311.

Hackos, J., Redish, J. (1998). ***User and task analysis for interface design.*** John Wiley & Sons.

Hix, D., Hartson, H. (1993). ***Developing User Interfaces: Ensuring Usability Through Product & Process.*** John Wiley & Sons, New York.

Jacobson, I., Booch, G., Rumbaugh, J. (1998). ***The Unified Software Development Process.*** Addison Wesley.

Jeffries, R., Miller, J., Wharton, C., and Uyeda, K. (1991). ***User Interface Evaluation in the Real World: A Comparison of Four Techniques.*** In Proceedings of the CHI '91 Conference on Human Factors in Computing Systems. ACM Press, New York, pages 145-151.

Jeffries, R. (1994). ***Usability Problem Reports: Helping Evaluators Communicate Effectively with Developers.*** In Usability Inspection Methods. J. Nielsen and R. L. Mack Eds. John Wiley & Sons, New York, pages 273-294.

Lewis, C., Rieman, J. (1993). ***Task-Centered User Interface Design: A Practical Introduction.*** University of Colorado, Boulder, Colorado.

Lethbridge, T. (1997). ***User Interface Design and Implementation SEG 3120 -course notes-***; University of Ottawa.

Lethbridge, T., Anquetil, N. (1997). ***Architecture of a source code exploration tools: A software engineering case study***. University of Ottawa. Computer Science Technical Report TR-97-07.

Lethbridge, T., Singer J., (1997). ***Understanding Software Maintenance Tools: Some Empirical Research.*** Workshop on Empirical Studies of Software Maintenance (WESS 97). Bari, Italy, October, 1997.

Lewis, C. (1982). ***Using the 'Thinking-Aloud' Method in Cognitive Interface Design.*** Research Report RC9265. IBM T. J. Watson Research Center, Yorktown Heights, New York.

Mayhew, J. (1991). ***Principles and Guidelines in Software User Interface Design.*** Prentice Hall.

Mayhew, J. (1999). ***The Usability Engineering Lifecycle.*** Morgan Kaufmann Publishers.

McKerlie, D., MacLean, A. (1993). ***Experience with QOC Design Rationale.*** In Adjunct Proceedings of the INTERCHI '93 Conference on Human Factors in Computing Systems, pages 213-214.

Nielsen, J. (1992a). ***Evaluating the thinking-aloud technique for use by Computer Scientists***. In Hartson, H.R., and Hix, D. (eds.), Advances in Human-Computer interaction, Vol. 3, Ablex, Norwood, NJ., pages 69-82.

Nielsen, J. (1992b). ***Finding usability problems through Heuristic Evaluation***. Proc. ACM CHI '92 Conf., pages 373-380.

Nielsen, J. (1993a). ***Usability engineering.*** AP Professional.

Nielsen, J. (1993b). ***Iterative User Interface Design.*** IEEE Computer 26, 11, pages 32-41.

Nielsen, J. (1994a). ***Enhancing the explanatory power of Usability Heuristics***. Proc. ACM CHI'94 Conf., pages 152-158.

Nielsen, J. (1994b). ***Usability Laboratories: A 1994 Survey***. Behaviour & Information Technology 13, 1&2, pages 3-8.

Nielsen, J. (1994c). ***Guerrilla HCI: Using Discount Usability Engineering to Penetrate the Intimidation Barrier.*** In Cost-Justifying Usability, D. J. Mayhew and R. G. Bias Eds. Academic Press, New York, pages 245-272.

Nielsen, J. (1994d). ***Heuristic Evaluation.*** In Usability Inspection Methods. J. Nielsen and R. L. Mack Eds. John Wiley & Sons, New York, pages 25-62.

Nielsen, J., Mack, R. (1994). ***Usability Inspection Methods***. John Wiley & Sons.

Norman, D., Draper, S. (1986). ***User Centered System Design: New Perspectives on Human-Computer Interaction.*** Hillsdale, NJ: Erlbaum Associates.

Norman, D. (1990). ***The design of everyday things.*** Currency Doubleday.

Preece, J., Rogers, Y., Sharp, H., Benyon, D. (1994). ***Human-Computer Interaction***. Addison Wesley.

Rosson, M., Carroll, J. (1995). ***Integrating Task and Software Development for Object-Oriented Applications.*** CHI' 95 Proceedings, pages 377-384.

Shneiderman, B. (1998). ***Designing the user interface, Strategies for Effective Human-Computer Interaction.*** Addison Wesley. 3rd edition.

Singer, J., Lethbridge, T. (1997). ***Work practices as an alternative method to assist tool design in software engineering***. University of Ottawa. Computer Science Technical Report TR-97-08.

Singer, J., Lethbridge, T., Vinson, N. and Anquetil, N. (1997). ***An Examination of Software Engineering Work Practices.*** CASCON '97, Toronto, October, pages 209-223.

Sommerville, I. (1997). ***Software engineering.*** Addison Wesley. 5th edition.

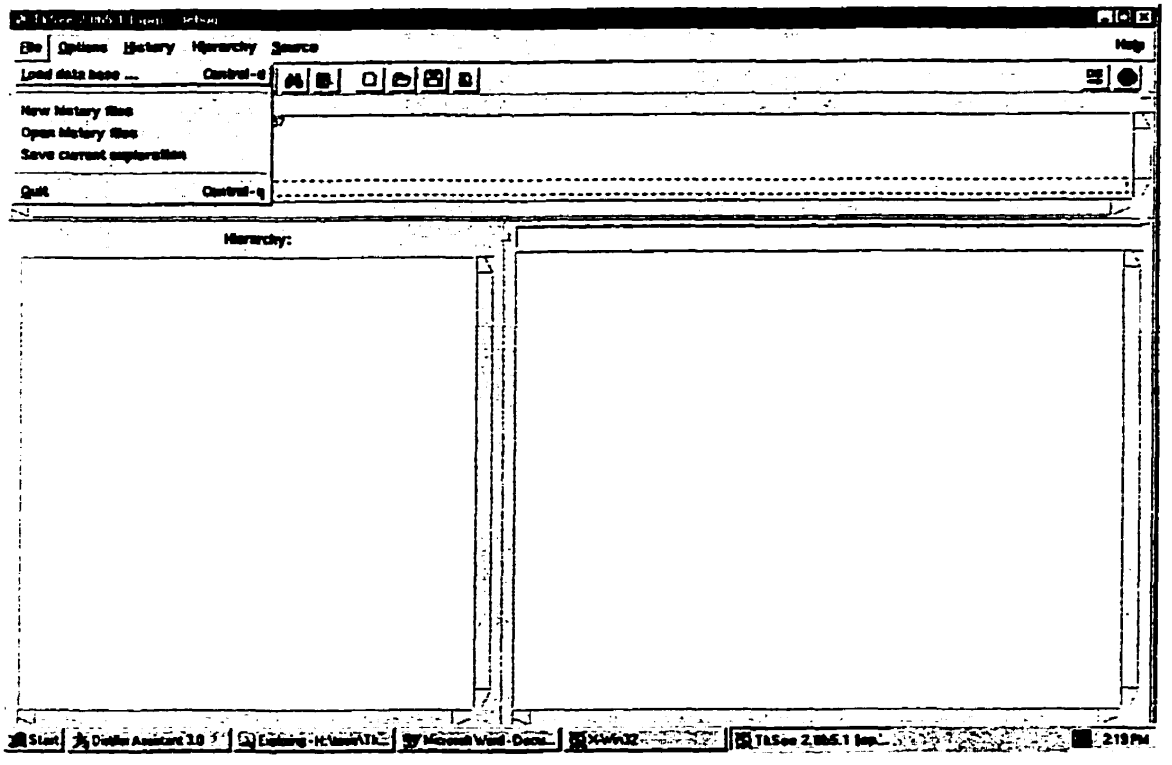
Storey, M.-A.D., Wong, K., Fong, P., Hooper, D., Hopkins, K., Müller, H.A. (1996). ***On Designing an Experiment to Evaluate a Reverse Engineering Tool.*** Proceedings of the 3rd Working Conference on Reverse Engineering, (WCRE'96), 1996, pages 31-40.

Storey, M.-A.D., Wong, K., Müller, H.A. (1997). ***How Do Program Understanding Tools Affect How Programmers Understand Programs.*** Proceedings of WCRE'97, 1997, pages 12-21.

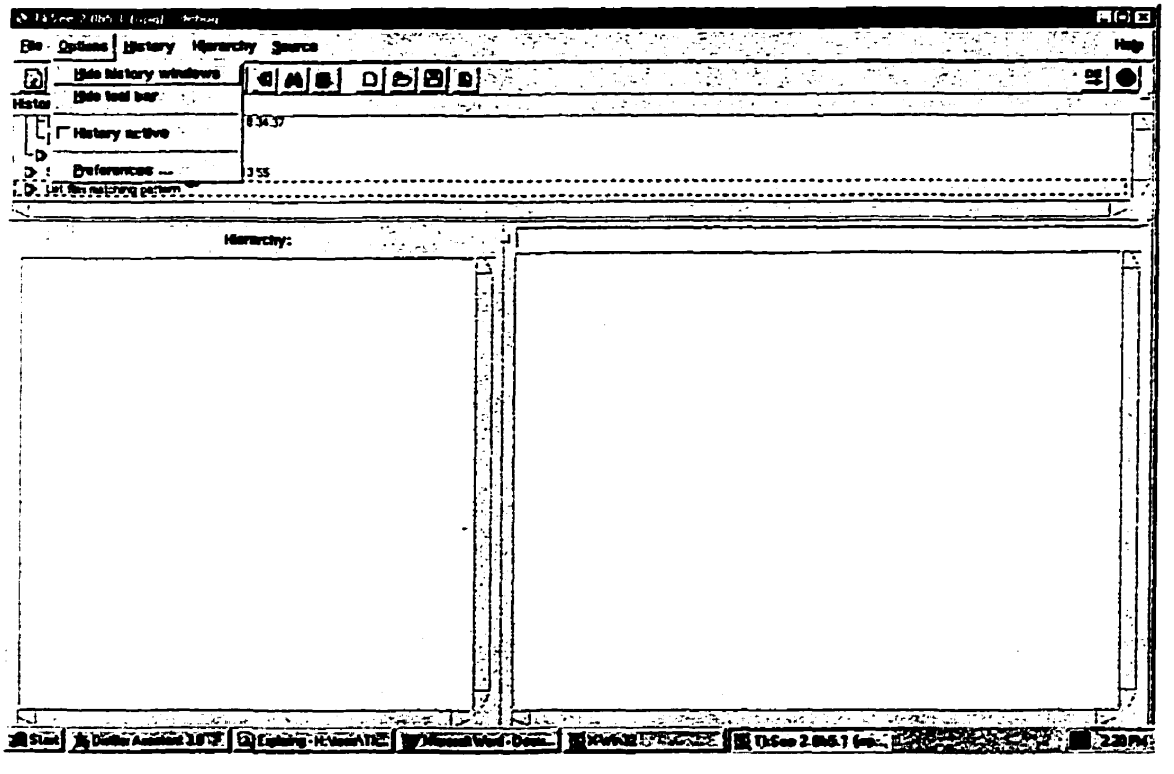
Whitefield, A., Wilson, F., and Dowell, J. (1991). ***A framework for human factors evaluation.*** Behaviour & Information Technology 10, 1 (January-February), pages 65-79.

APPENDIX 1
The Tool before the Project

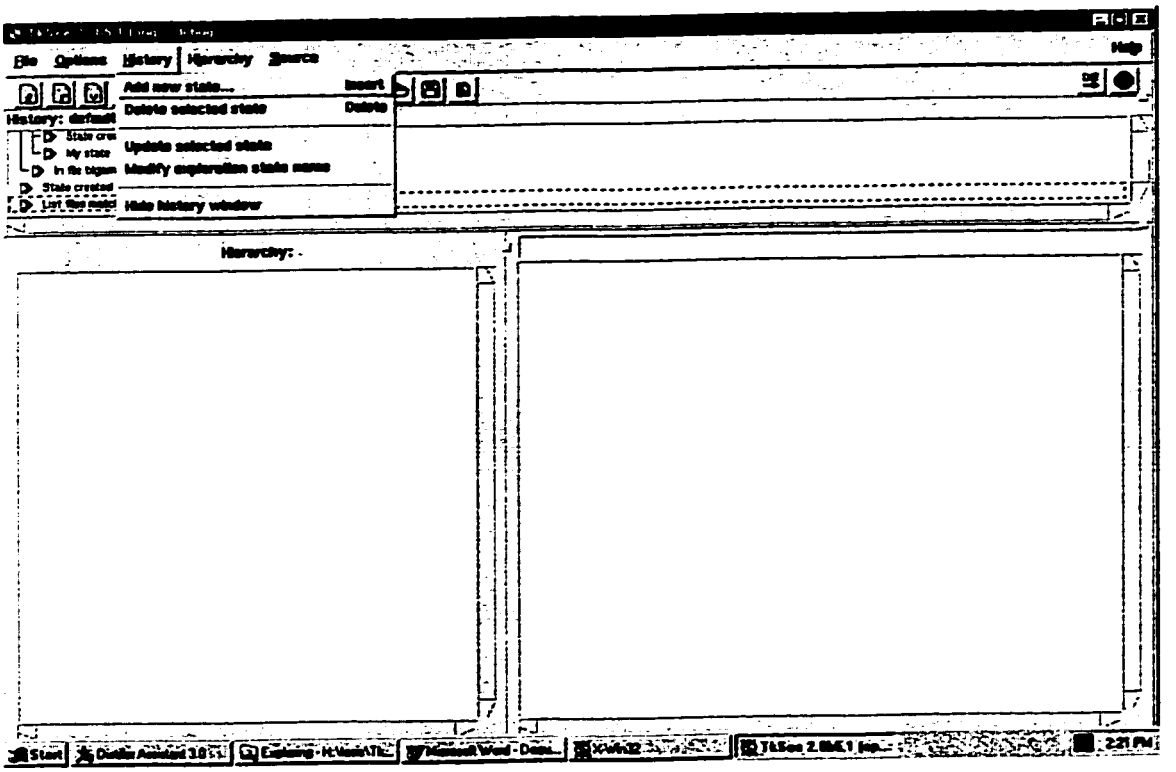
1)



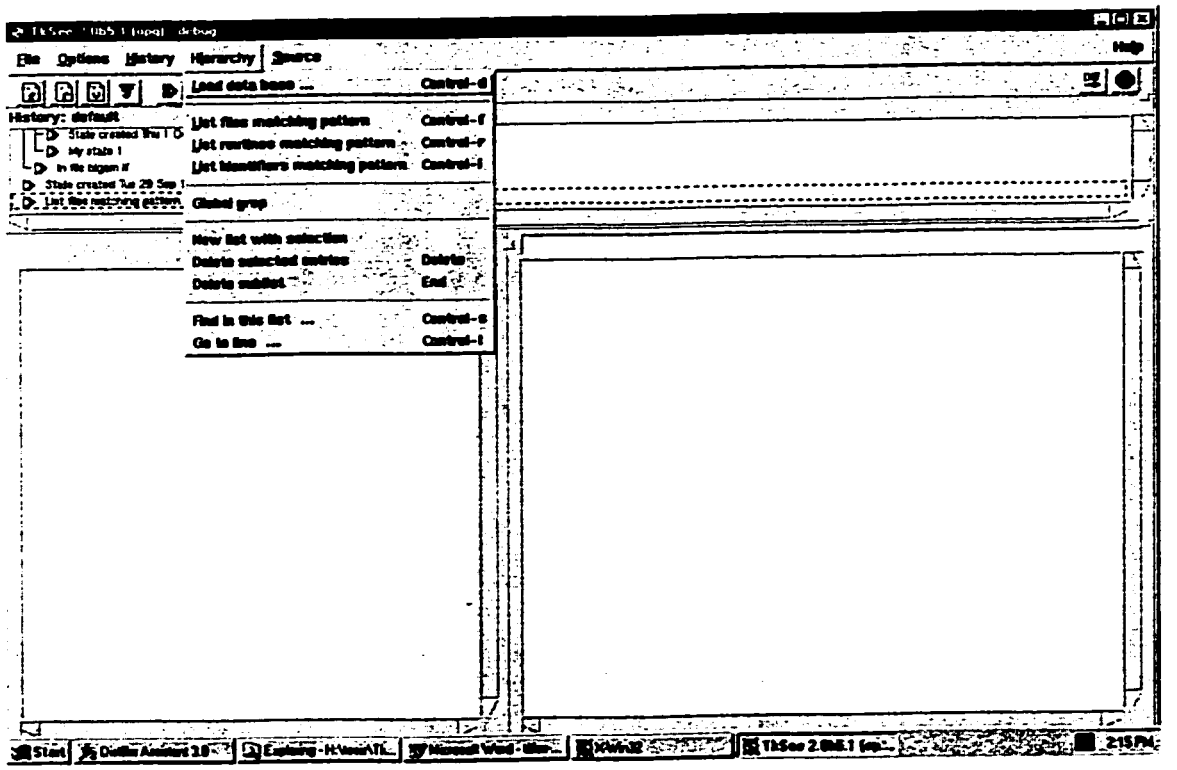
2)



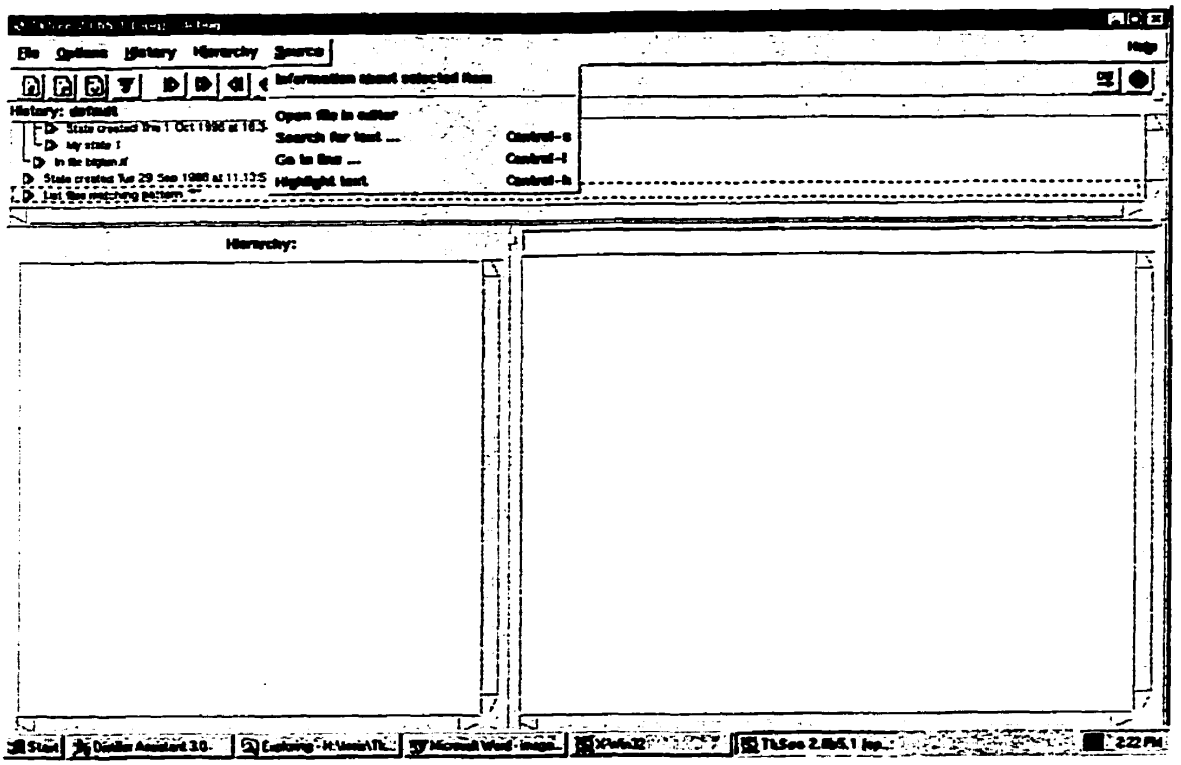
3)


File Options History Hierarchy Source
Help
Add new state... Insert
Delete selected state Delete
Update selected state
Modify exploration state name
Hide history window
Hierarchy:

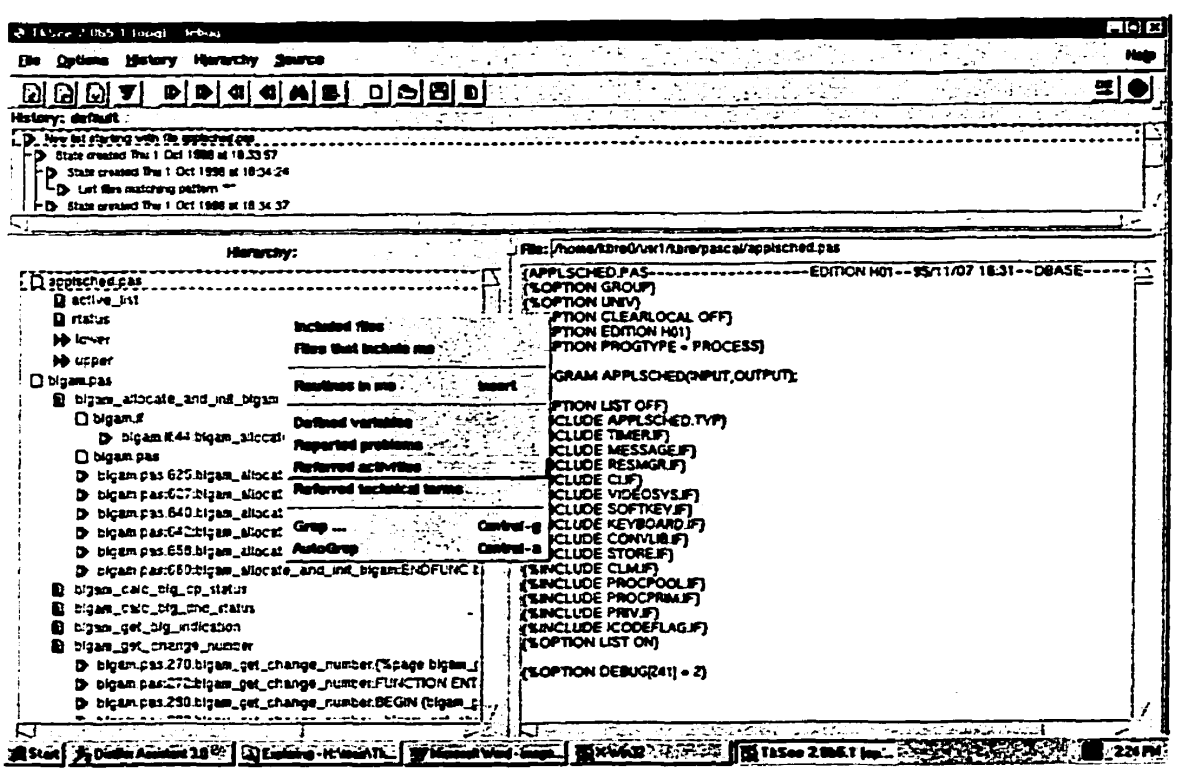
4)


File Options History Hierarchy Source
Help
Load data base ... Control-d
List files matching pattern Control-f
List routines matching pattern Control-r
List identifiers matching pattern Control-i
Global grep
New list with selection
Delete selected entries Delete
Delete subtree End
Find in this list ... Control-s
Go to line ... Control-l

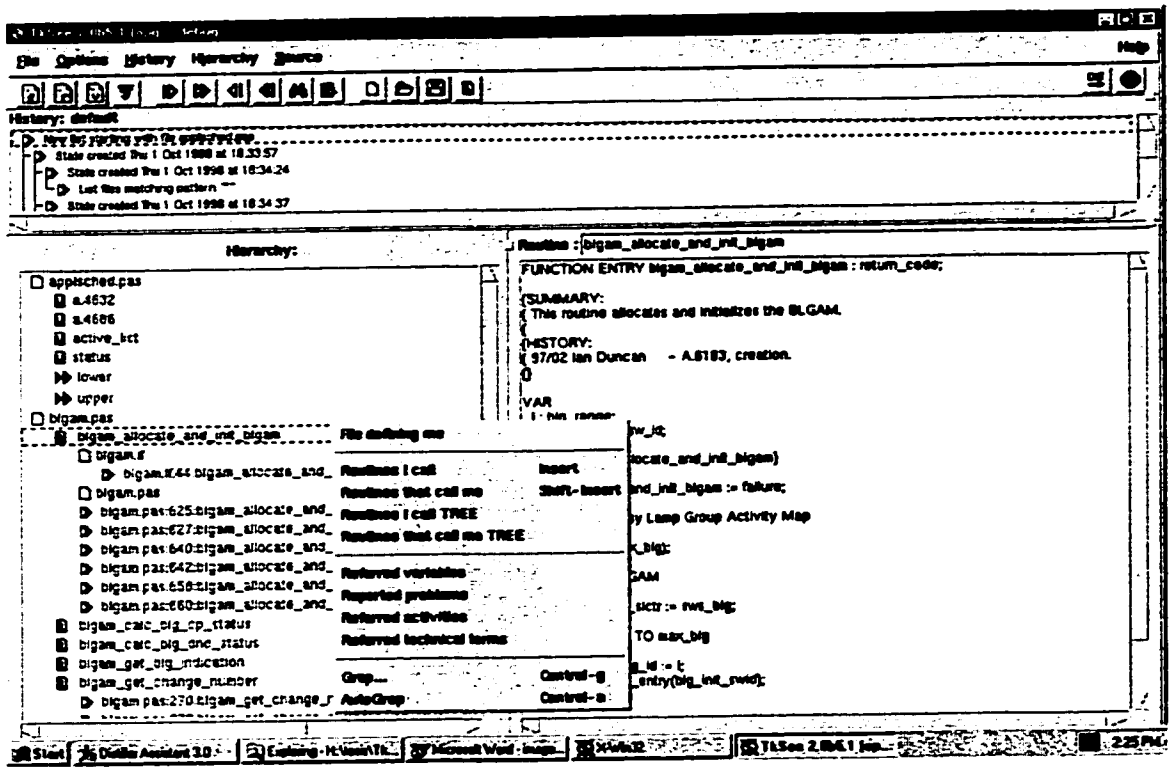
5)


File Options History Hierarchy Source
Help
Hierarchy:

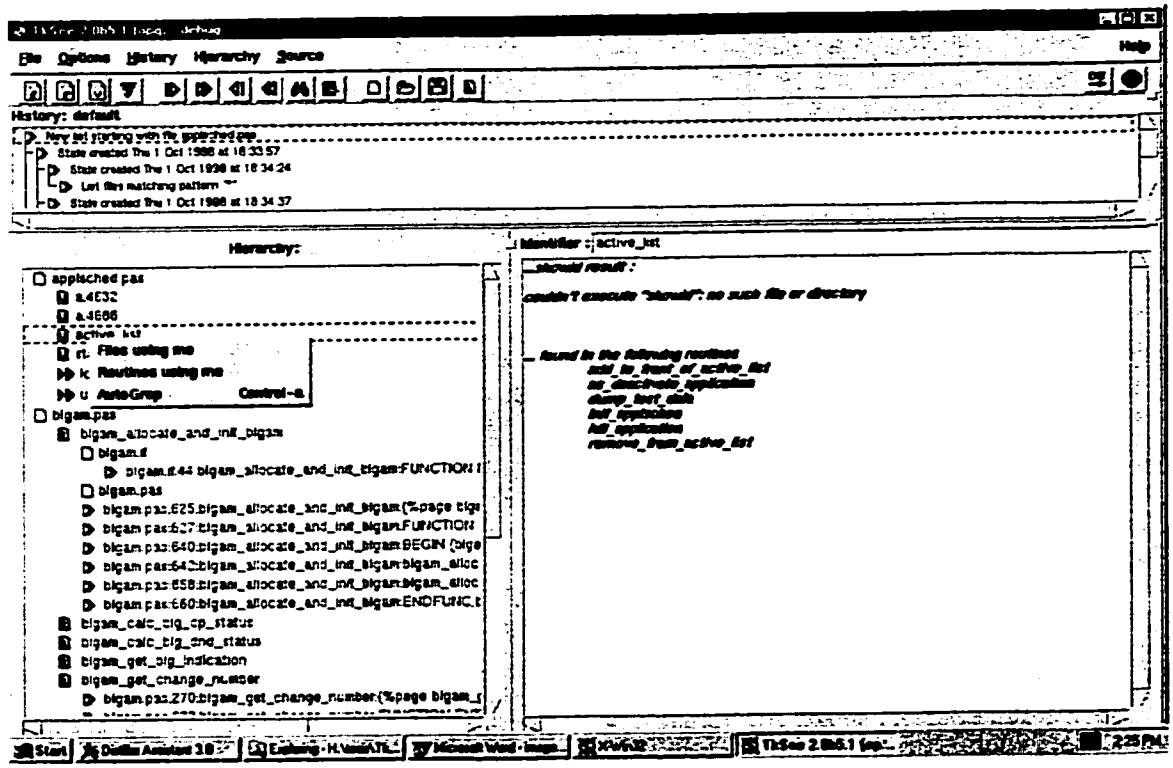
6)


File Options History Hierarchy Source
Help
History: default
Hierarchy:
Included files
Files that include me
Routines in me
Defined variables
Reported problems
Referred activities
Referred technical terms
Grep ...
AutoGrep
(APPLSCHED.PAS----------EDITION H01--95/11/07 18:31--DBASE-----
{%OPTION GROUP}
{%OPTION UNIV}
{%OPTION LIST OFF}
{%INCLUDE CLM.IF}
{%INCLUDE PROCPOOL.IF}
{%INCLUDE PROCPRIM.IF}
{%INCLUDE PRIV.IF}
{%INCLUDE ICODEFLAG.IF}
{%OPTION LIST ON}
{%OPTION DEBUG[241] = 2}

7)


File Options History Hierarchy Source Help
Hierarchy:
appsched.pas
a.4632
a.4686
active_lst
status
lower
upper
bigam.pas
bigam_allocate_and_init_bigam
bigam.if
bigam.pas
bigam_calc_big_cp_status
bigam_calc_big_cnd_status
bigam_get_big_indication
bigam_get_change_number
Routine : bigam_allocate_and_init_bigam
FUNCTION ENTRY bigam_allocate_and_init_bigam : return_code;
{SUMMARY:
{ This routine allocates and initializes the BLGAM.
{HISTORY:
{ 97/02 Ian Duncan - A.6183, creation.
VAR
Routines I call Insert
Routines that call me Shift-Insert
Routines I call TREE
Routines that call me TREE
Referred variables
Reported problems
Referred activities
Referred technical terms
Grep... Control-g
AutoGrep Control-a

8)


File Options History Hierarchy Source Help
Hierarchy:
appsched.pas
a.4632
a.4686
active_lst
rt. Files using me
k Routines using me
u AutoGrep Control-a
bigam.pas
bigam_allocate_and_init_bigam
bigam.if
bigam.pas
bigam_calc_big_cp_status
bigam_calc_big_cnd_status
bigam_get_big_indication
bigam_get_change_number
Identifier : active_lst
found in the following routines
add_to_front_of_active_list
de_deactivate_application
dump_bad_data
bad_application
bad_application
remove_from_active_list

APPENDIX 2

Usability Problems found in the Heuristic Evaluation

The following notation is used in the first row of the tables:

1. The first field indicates the number of the problem and its location in the user interface according to the codes below. It also contains a letter "v" in case the problem was Verified during the thinking aloud usability testing.

   OW: Overall in the Interface
   DW: about Dialog Windows opened by interface features
   HW: related to the History Window
   HiW: related to the Hierarchy Window
   SW: related to the Source Window
   TS: related with the Tool bar and Shortcuts
   PW: related with the Preferences dialog Window
   GG: related with the Global Grep feature and its window

2. The second field indicates the usability guideline violated by the problem.

3. The third field indicates whether the problem was about *Learning* (letter "**L**"), *Efficiency of use* (letter "**E**"), or both. It also classifies the problems according to the following categories:

   i. Non optimal functionality
   ii. Possible confusion
   iii. Possible misinterpretation
   iv. Unnecessary functionality
   v. Lack of functionality
   vi. Lack of consistency
   vii. Graphical design
   viii. Feedback
   ix. Lack of robustness
   x. Wrong behavior
   xi. Bad labeling
   xii. Lack of labeling

4. The fourth field indicates an approximate Usability Severity (**US**) of the problem according to the codes below. The severity values were established when reviewing the problems in the sessions with developers.

   1 = Cosmetic problem, fixing it should be given low priority.
   2 = Minor problem, important to fix but can be given low priority.
   3 = Major problem, important to fix and should be given high priority.
   w = Wait to fix it until it is verified in the thinking aloud usability testing.

PROBLEMS:

I. OVERALL IN THE INTERFACE

1. OI	**Violated principle:** Speak users' language -confusing labeling-	**Problem classification:** L, Possible confusion	**US:** 2-w

Problem: There are two options labeled *'History'* and *'Hierarchy'* in the main menu. Likewise, two windows are labeled *'History'* and *'Hierarchy'*. This could suggest that features within the options in the main menu correspond to things or actions in those windows. However, only some features are intended like that. Novice users might get confused by having features affecting globally in the tool and others affecting locally within those windows.

Features with the problem:

History->Delete selected state
History->Hide history window
Hierarchy->Load database ...
Hierarchy->List files matching pattern
Hierarchy->List routines matching pattern
Hierarchy->List identifiers matching pattern
Hierarchy->Global grep
Hierarchy->New list with selection

2. OI, v	**Violated principle:** Minimize users' memory load	**Problem classification:** L, Possible misinterpretation	**US:** 2-w

Problem: As mentioned in the previous problem, certain features within the options *'History'* and *'Hierarchy'* affect things in those windows only, however some of them have labels that could be interpreted affecting not only that window but other things in the tool as well. Therefore, users will have to remember the way they work.

Features with the problem:

History->Add new state ...
History->Delete selected state
History->Update selected state
History->Modify exploration state
Hierarchy->New list with selection
Hierarchy->Delete selected entries
Hierarchy->Delete sublist
Hierarchy->Find in this list ...
Hierarchy->Go to line ...

3. OI	**Violated principle:** Simple and natural dialogue	**Problem classification:** L, Lack of labeling	**US:** 3

Problem: Features in the *'Source'* option in the main menu affect the *'Source'* window. However, such window doesn't present a label many times therefore making harder that users discover the relationship.

4.	**Violated principle:**	**Problem classification:**	**US:**
OI	Minimize users' memory load	**L**, Possible misinterpretation	2-w

Problem: As mentioned in the previous problem, the features within the *Source* menu affect the *'Source'* window only, however some of them have labels that could be interpreted affecting not only that window but other things in the tool as well. Therefore, users will have to remember the way they work.

Features with the problem:

Source->Information about selected item
Source->Open in a file editor
Source->Search for text ...
Source->Go to line ...
Source->Highlitght text

5.	**Violated principle:**	**Problem classification:**	**US:**
OI	Simple and natural dialogue -graphic design-, Consistency	**L**, Graphical design	2

Problem: In the *History* window the label is left aligned and in the *Hierarchy* window is center aligned. This is a lack of consistency in the graphical design of the user interface. Good graphical design is known to help users at learning and using the interface faster and more efficiently. Also, consistency helps users to transfer knowledge from one part to another (e.g. the user can know how to use something new but similar).

6.	**Violated principle:**	**Problem classification:**	**US:**
OI	Simple and natural dialogue -graphical design-	**L**, Graphical design	2

Problem: The design of the *History*, *Hierarchy* and *Source* windows doesn't suggest a clear division among windows. For example, the bar at the top looks different in height. This is part of the graphical design of the user interface.

7.	**Violated principle:**	**Problem classification:**	**US:**
OI	Simple and natural dialogue	**E**, Non optimal functionality	1

Problem: The *History*, *Hierarchy* and *Source* windows present scroll bars even when these are not necessary. Also, some scroll bars might be rarely used (e.g. the horizontal scroll bar in the *History* sub—window). It would be better to present scroll bars only when it is necessary.

8.	**Violated principle:**	**Problem classification:**	**US:**
OI	Simple and natural dialogue	**L**, Possible misinterpretation	3

Problem: The history states in the *History* window have the same icon that some objects in the *Hierarchy* window. However, they are not related at all. This could suggest a relation between both and confuse users.

Location observed:

History window, history event
Hierarchy window, grep result

9.	**Violated principle:**	**Problem classification:**	**US:**
OI, v	Speak users' language	**L**, Possible confusion	2

Problem: The message at the bottom-right of the interface seems to inform about the *Source* window only, however, it presents information global in the tool. Also, this message doesn't have any label indicating its meaning. Users will have to deduce it themselves.

10. OI	**Violated principle:** Simple and natural dialogue, Consistency	**Problem classification:** L, Possible confusion	**US:** 2
Problem: The message at the bottom-left of the interface seems to belong to the *Hierarchy* window. However, sometimes presents information about that window and others about the whole system. **Situations observed:** When loading a database			

11. OI	**Violated principle:** Speak users' language	**Problem classification:** L, Lack of labeling	**US:** 2
Problem: When loading a database, the message at the bottom-left of the interface displays a file with its path. There isn't a label indicating its meaning. Also, the same file name is displayed at the bottom-right at the same moment. Users will have to deduce the meaning themselves. **Observed:** When loading a database			

12. OI	**Violated principle:** Simple and natural dialogue	**Problem classification:** L, E, Possible confusion	**US:** 3
Problem: Using *Options->Hide history window* changes the main menu options. Changing options in a menu can degrade user learning and performance. Some users memorize system features by their location in the interface. It is usually better to gray out options not available in a certain moment than disappearing them from a menu. **Location observed:** Options->Hide history window History->Hide history window History window pop-up menu->Hide history window			

13. OI, v	**Violated principle:** Clearly marked exits	**Problem classification:** E, Lack of functionality	**US:** 3
Problem: There are not *'undo'* facilities for any actions in the tool. Users will want to go back from erroneous actions. This can degrade user learning and performance. On the other hand, there are certain actions in the system that can be reversed by doing other actions or *tricks*, however, users might not notice that. **Some locations observed:** - Undo the execution of a query - Undo any kind of deletion			

14. OI, v	**Violated principle:** Feedback –response time-, Clearly marked exits	**Problem classification:** E, Feedback	**US:** 3
Problem: The time for stopping queries is sometimes very long. Users will feel trapped by the tool. They might click on the *stop* button several times trying to stop the query. Waiting long will affect their concentration and performance.			

15. OI, v	**Violated principle:** Feedback –response time-, Clearly marked exits	**Problem classification:** E, Feedback	**US:** 3
Problem: The response time of the *stop* button is sometimes very long (the time it takes to indicate that the button is working). Like in the previous problem, users will feel trapped by the tool. They might click on the button several times trying to make it work. Waiting long will affect their concentration and performance.			

16. OI, v	**Violated principle:** Feedback	**Problem classification:** L, E, Feedback	**US:** 3

Problem: The feedback in the tool is not clear or properly supplied. The following are different situations where this was observed:

i) Sometimes an action was started but there was no indication that it had been started. This was particularly notorious on actions that took long or when it was desired to know if the system got the proper input from the user. An observed situation with this problem was:

- When queries were stopped, the stop button didn't react immediately.

ii) Sometimes the tool was processing something but there was no indication of that. This was particularly notorious on actions that took long. An observed situation with this problem was:

- When the evaluator deleted long lists of items in the *Hierarchy* window.

iii) Sometimes the tool was processing something but there was way of knowing how long it would take to finish. This was particularly notorious on actions that took long. An observed situation with this problem was:

- When processing queries that took long. In this case, the system displayed a mouse clock. However, that only indicated that the system was doing the action but didn't indicate how longer it would take.

iv) Sometimes the tool did not indicate that an action had finished. Sometimes there was no indication at all, and others the feedback was not explicit enough to be noticed. Some observed situations with this problem were:

a) When a query or action produced an empty result.
b) When actions finished very fast and left the interface exactly as it was before. This happened when using:
   - Hierarchy window, File pop-up menu->Routines in me
   - Hierarchy window, Routine pop-up menu->Routines I call
c) When the evaluator deleted items in the *Hierarchy* window.
d) Executing *History->Update selected state*
e) Executing *Hierarchy->Find in this list ...*
f) Executing *Source->Information on selected item*
g) Executing *Source->Highlitght text*

v) Sometimes the information presented in the interface did not correspond to the reality *(misleading feedback).* This can make users assume things that are not true. Some observed situations with this problem were:

a) When some dialog windows were open, if the mouse was moved out of the window it changed to a mouse clock, however, there was no operation being performed at that moment. This happened when using:
   - Hierarchy->List files matching pattern
   - Hierarchy->List routines matching pattern
   - Hierarchy->List identifiers matching pattern
b) After doing certain actions in the *Hierarchy* window, the message at the bottom-left was not corrected.
c) Deleting objects in the *Hierarchy* window left the label of the corresponding history state unaffected.
d) Selecting objects in the Hierarchy window didn't refresh the *Source* window properly.
e) Using *Routines I call* and *Routines I call TREE* displayed repeated information in the *Hierarchy* window.
f) Routines in the Hierarchy window were displayed alphabetically.

17. OI	**Violated principle:** Simple and natural dialogue, Consistency	**Problem classification:** L, E, Possible confusion	**US:** 3

Problem: The feature *Options->History active* doesn't work properly. It blocks the *History* window so that selecting history states doesn't refresh the other windows. However, some operations in the *Hierarchy* window that affect the history still work. Users will have problems learning and understanding this behavior. Also, this option might be better located within preferences rather than in the *Options* menu.
Location observed:
Options->History active

18. OI	**Violated principle:** Consistency -standards-	**Problem classification:** L, Lack of consistency	**US:** 1

Problem: In the main menu, when the menu of an option is open, moving the mouse to another option highlights the other option but the same menu remains displayed and it is not changed to that of the new option. For example, open the *History* menu and then move the mouse to *Hierarchy* without clicking. The *Hierarchy* option is highlighted but the displayed menu is still the *History* menu.
Location observed:
Main menu

19. OI	**Violated principle:** Simple and natural dialogue	**Problem classification:** E, Lack of robustness	**US:** 1

Problem: If the main window of the interface is shrunk, the interior windows (*History*, *Hierarchy* and *Source*) are not adjusted proportionally. Sometimes, if such window is enlarged back again, some of the interior windows remain shrunk, not proportional and not functional. Users will have to adjust the interior windows manually.
Situations observed:
Specially observable with extreme shrinking

20. OI	**Violated principle:** Simple and natural dialogue	**Problem classification:** E, Lack of robustness	**US:** 2

Problem: There are no limits when resizing the interior windows (*History*, *Hierarchy* and *Source*). An interior window can be resized until it makes another window disappear, thus leaving the other one not functional. Users will have to adjust the interior windows manually.
Location observed:
Specially observable with extreme resizing

21. OI	**Violated principle:** Simple and natural dialogue, Bug	**Problem classification:** E, Wrong behavior	**US:** 2

Problem: The tool accepts saving a history file with a name containing special UNIX characters like * or ?. After saving files like that, it is even possible to look at them through the windows for creating and opening history files.
Location observed:
File->New history files
File->Open history files
File->Save current exploration

22. OI, v	**Violated principle:** Simple and natural dialogue, Consistency –standards-	**Problem classification:** E, Non optimal functionality	**US:** 3

Problem: A name of a history file is always asked when creating a new exploration. This is not a problem itself, however it's is not a common procedure in other systems and combined with the problems below then it becomes an inefficient behavior of the system.
Location observed:
File->New history files

23. OI, v	**Violated principle:** Simple and natural dialogue, Consistency -standards-	**Problem classification:** **E**, Unnecessary functionality	**US:** 2
Problem: When saving the current exploration, the name of a file is always asked. There is no feature for saving the exploration without asking a name. Most other systems implement a feature for that. Also, as indicated in the previous problem, the name of a history file has already been asked when creating the exploration. This problem is also present when the user wants to leave the system, create or open another history file, the tool always prompts for a name. The user will waste time attending this action each time the exploration is saved. In addition, the label for saving the exploration is *File->Save current exploration*, which doesn't suggest that a file name will be prompted. It would be better *File->Save current exploration as ...* **Location observed:** File->Save current exploration			

24. OI, v	**Violated principle:** Simple and natural dialogue, Consistency –standards-	**Problem classification:** **E**, Unnecessary functionality	**US:** 2
Problem: When leaving the tool, creating or opening another history file, the system always prompts to save the current exploration. The user has always to attend this action even if the exploration hasn't changed and there is nothing to save. It would be better if the tool kept track of any changes made to the history and just prompt in that case. **Location observed:** File->New history files File->Open history files File->Save current exploration File->Quit			

25. OI, v	**Violated principle:** Simple and natural dialogue, Consistency -standards-	**Problem classification:** **E**, Unnecessary functionality	**US:** 2
Problem: The tool always prompts for confirmation when the user leaves the system. This is a waste of time for users that are already prepared to exit. It is better if the tool prompted the user just if there is anything that has to be saved. **Location observed:** File->Quit			

26. OI	**Violated principle:** Simple and natural dialogue -less is more-	**Problem classification:** **L**, Possible confusion	**US:** 2
Problem: Some features are repeated twice in different menus. This can confuse users because they won't know at first if those features at different locations are indeed the same feature. Also, it makes bigger the menus and therefore the user has more information to learn and remember. Finally, this is not very common in other systems. **Location observed:** File->Load database ... Hierarchy->Load database ... Options->Hide history windows History->Hide history window			

27. OI	**Violated principle:** Minimize users memory load	**Problem classification:** L, Possible confusion	**US:** 3
Problem: There are different features in different menus with exactly the same label. Users will have to notice that they are different features that work differently. **Location observed:** Hierarchy->Go to line ... Source->Go to line ...			

28. OI, v	**Violated principle:** Simple and natural dialogue, Minimize users' memory load	**Problem classification:** L, E, Non optimal functionality, Feedback	**US:** 3
Problem: All features in the interface look active all the time. There is no gray out of features that do not work at some moment. If a feature is active but doesn't work, it can affect on users' performance. Users will have to learn by experience, what features work and what don't in different situations.			

29. OI	**Violated principle:** Consistency	**Problem classification:** L, Lack of consistency	**US:** 1
Problem: The labels and names of menus, buttons and windows, sometimes have the words starting capitalized and others don't. This is a lack of consistency in the graphical design of the user interface. Good graphical design is known to help users at learning and using the interface faster and more efficiently. Also, consistency helps users to transfer knowledge from one part to another (e.g. the user can know how to use something new but similar). **Some locations observed:** *File->Load database ...* , and the dialog box that is opened *Hierarchy->Global grep* , and dialog box that is opened			

30. OI	**Violated principle:** Consistency	**Problem classification:** L, Lack of consistency	**US:** 2
Problem: The labels of some features are different from the titles of the dialog windows they open. Some users, especially when learning a system, compare window titles with labels of the features to see if the windows opened come from those features. **Some locations observed:** File->New history files File->Open history files File->Save current exploration File->Quit Options->Preferences History->Modify exploration state name Hierarchy->List files matching pattern Hierarchy->List routines matching pattern Hierarchy->List identifiers matching pattern Hierarchy->Find in this list ... Hierarchy->Go to line ... -it says "Goto"- Source->Search for text ... Source->Go to line ... -it says "Goto"- Help->About Tksee			

31. OI	**Violated principle:** Consistency	**Problem classification:** L, Lack of consistency	**US:** 2

Problem: The labels of the help flags of some buttons from the tool bar do not correspond to the labels of the same features in the menus. Some users, especially when learning the system, compare those labels to establish relationships among buttons and features.

Location observed:

Tool bar->Grep into the entire system
Hierarchy->Global grep
Tool bar->Routines I call (TREE)
Routine middle-mouse button pop-up menu-> Routines I call TREE
Tool bar->Routines that call me (TREE)
Routine middle-mouse button pop-up menu-> Routines that call me TREE

32. OI	**Violated principle:** Speak users' language	**Problem classification:** L, Bad labeling	**US:** 1

Problem: Some dialog windows have messages within them. However, those messages alone or combined with the window titles do not always suggest users the purpose of the window. It's important to help users not just on what to do within a dialog window but also with what the window is for. For example, in *Hierarchy->Find in this list ...* the window title is *Find window* and the message inside is *Enter regular expression.* A better message might be something like *Enter text to find in the exploration hierarchy.*

Location observed:

Hierarchy->Find in this list ...

33. OI	**Violated principle:** Speak users' language, Consistency	**Problem classification:** L, Bad labeling	**US:** 1

Problem: The following features have labels in plural, however they work on one object or thing only. The first two features below open dialog boxes with window titles in singular.

Location observed:

File->New history files
File->Open history files
Options->Hide history windows

34. OI, v	**Violated principle:** Simple and natural dialogue, Consistency -standards-	**Problem classification:** E, Non optimal functionality	**US:** 2-w

Problem: It's necessary to hold down the mouse button so that pop-up menus stay on the screen. This is or not a problem depending on other reasons, however it always requires more effort to the user. Also, this is not the standard practice in other systems.

Location observed:

Pop-up menus in Hierarchy, History and Source windows

35. OI	**Violated principle:** Consistency	**Problem classification:** L, E, Lack of consistency	**US:** 2

Problem: The default colors in the interface are used inconsistently. The same color is used in different places for different meanings.

Some locations observed:

- Blue is used in History window and sometimes in Source window as well

36. OI, v	**Violated principle:** Simple and natural dialogue, Consistency -standards-	**Problem classification:** E, Lack of functionality	**US:** 2

Problem: Some searching features work so that they search just from a certain point-down, they don't allow searching the rest of the information left above. Also, they don't allow searching bottom-up. In order to search a whole document, users have to go at the start of the document and then perform the search. Also, the dialog windows that these features open have a button labeled *Find*, but it might be better to label it as *Find next* as in other systems..

Some locations observed:

Hierarchy->Find in this list ...
Source->Search for text ...

37. OI, v	**Violated principle:** Simple and natural dialogue, Shortcuts -improper use-	**Problem classification:** E, Non optimal functionality	**US:** 1

Problem: The buffer for searching actions is the same for several features and is not cleaned from previous searches. For example, looking for a file and then opening the dialog window again to look for variable will present the input previously given when looking for the file. It might be better if just similar types of searches shared the same previous buffer. Also, previous inputs are not presented highlighted when dialog windows are opened, therefore, the users has to clean them manually character by character.

Some locations observed:

Hierarchy->List files matching pattern
Hierarchy->List routines matching pattern
Hierarchy->List identifiers matching pattern
Hierarchy->Find in this list ...
Source->Search for text ...
Hierarchy->Go to line ...
Source->Go to line ...

38. OI	**Violated principle:** Error prevention	**Problem classification:** E, Non optimal functionality	**US:** 1

Problem: The following dialog windows keep input from previous occasions and it would become a problem if the user accidentally pressed the default action with such input present (e.g. this would start closing the current history file).

Location observed:

File->Load database ...
File->New history files
File->Open history files
Hierarchy->Load database ...

39. OI, v	**Violated principle:** Simple and natural dialogue	**Problem classification:** L, Possible confusion	**US:** 3-w

Problem: It is possible to have several objects selected in the interface at the same time. Some features have labels referring to *selected objects*, however such labels do not clearly suggest what selected object(s) the feature refers to. Thus, it won't be easy for users to know what objects will be affected.

Location observed:

History->Delete selected state
History->Update selected state
Hierarchy->New list with selection
Hierarchy->Delete selected entries
Source->Information about selected item

40. OI	**Violated principle:** Simple and natural dialogue, Error prevention	**Problem classification:** L, E, Lack of robustness	**US:** 3
Problem: There is a potential unnatural scenario with the databases and history files. It's not clear what will happen if having a database loaded, the user decides to open a history file that was not created using the same database. The tool should manage these possible erroneous situations and not expose the user to them.			

41. OI	**Violated principle:** Simple and natural dialogue	**Problem classification:** L, Non optimal functionality	**US:** 2-w
Problem: Loading a new database creates a new history file, however this is not clearly informed. A user might not realize that and he will have to learn by experience.			

42. OI	**Violated principle:** Consistency	**Problem classification:** E, Lack of consistency	**US:** 1
Problem: The left and right arrow keys do not behave the same in the different windows and dialog windows. **Location observed:** History, Hierarchy and Source windows Dialog windows			

43. OI	**Violated principle:** Speak users' language	**Problem classification:** L, Possible misinterpretation	**US:** 3-w
Problem: There are two windows in the interface that show a hierarchy of items (Hierarchy and History). However, the features in the *Hierarchy* menu only affect the hierarchy in the Hierarchy window. This could confuse users because they won't easily notice what hierarchy the features refer to. Changing the labels could help, for example, the Hierarchy menu and window could say *Hierarchy of software objects*, and the History window *Hierarchical history of explorations.*			

44. OI, v	**Violated principle:** Minimize users' memory load	**Problem classification:** E, Non optimal functionality	**US:** 3
Problem: The Hierarchy and History windows can contain very big and long hierarchies. Users will have to scroll a lot in order to navigate them. It is known that users are not good for managing big hierarchies presented as long lists. It would be better to have expansion and contraction facilities for those hierarchies. **Location observed:** History and Hierarchy windows			

45. OI, v	**Violated principle:** Simple and natural dialogue, Minimize users memory load	**Problem classification:** E, Lack of functionality	**US:** 3
Problem: In the Hierarchy and Source windows, there is a feature for jumping to a certain line specifying its number. However, the tool doesn't provide a way to know the number of a particular line, therefore making difficult for a user to know later what number he must enter in order to jump to that line. Also, even if the user enters a number to jump to certain line, he has to trust that the resulting line with the focus is the line corresponding to the number he entered. **Location observed:** Hierarchy and Source menu			

46. OI, v	**Violated principle:** Speak users' language	**Problem classification:** L, E, Possible confusion	**US:** 3-w
Problem: Using "*" in some searching features don't give any result, however the tool does not inform about that. This behavior is due to the definition of regular expressions and how the system is built. Other systems don't work in this way. Users with other background in other systems will probably experience problems with that behavior. It's important to consider if the users of the tool are aware of regular expressions as well as if they use them in their work. Otherwise, users not familiar with them will probably have problems. **Location observed:** Hierarchy->List files matching pattern Hierarchy->List routines matching pattern Hierarchy->List identifiers matching pattern			

47. OI, v	**Violated principle:** Minimize users' memory load	**Problem classification:** E, Lack of functionality	**US:** 2
Problem: Users tend to forget how to construct regular expressions, so it would be possible that the tool gave a couple of examples in a pop-up dialog box to refresh peoples' memory (especially with previously used regular expressions). **Location observed:** Searching features throughout the tool			

48. OI, v	**Violated principle:** Simple and natural dialogue, Consistency -standards-	**Problem classification:** E, Lack of functionality	**US:** 1-w
Problem: There are no edition facilities in the tool such as for copying and pasting. Thus, users have to rely on the operating system facilities for this. However, such operating system facilities might not be always known or easy to use. Many systems provide editing features so users might probably look for them in the tool.			

49. OI, v	**Violated principle:** Simple and natural dialogue	**Problem classification:** E, Lack of functionality	**US:** 2-w
Problem: Users might be interested in other aspects of the objects displayed by the tool than the just the ones currently supported, for example types.			

50. OI, v	**Violated principle:** Simple and natural dialogue	**Problem classification:** L, E, Non optimal functionality	**US:** 3
Problem: Currently it is possible to make explorations of things that do not exist. However, the tool can know about some of these non-existing things and therefore it could provide clues so that users didn't waste time. For example, the tool could slightly change the color of the icons. Thus, a routine that had no callers could look different from one that had.			

51. OI	**Violated principle:** Consistency	**Problem classification:** L, Lack of consistency	**US:** 1

Problem: Some dialogues and messages use a different font than the rest of the system. This is not a serious problem but contributes to the good overall graphical design of the interface.

Location observed:

File->Quit
History->Modify exploration state name
Hierarchy->List files matching pattern
Hierarchy->List routines matching pattern
Hierarchy->List identifiers matching pattern
Hierarchy->Find in this list ...
Hierarchy->Go to line ...
Source->Search for text ...
Source->Go to line ...

52. OI	**Violated principle:** Help and documentation	**Problem classification:** L, Lack of functionality	**US:** 2

Problem: Currently the only online help available is a small tutorial of the tool. However, that tutorial is does not describe the version of the tool that users have.

II. ABOUT DIALOG WINDOWS OPENED BY INTERFACE FEATURES

53. DW	**Violated principle:** Consistency -standards-	**Problem classification:** L, Lack of consistency	**US:** 2

Problem: Some features open dialog windows but their labels don't have "...". This is the standard way in other systems to indicate that another dialog window will be open.

Location observed:

File->New history files
File->Open history files
File->Save current exploration
History->Modify exploration state name
Hierarchy->List files matching pattern
Hierarchy->List routines matching pattern
Hierarchy->List identifiers matching pattern
Hierarchy->Global grep

54. DW	**Violated principle:** Consistency -standards-	**Problem classification:** L, Lack of consistency	**US:** 2

Problem: Some features have "..." at the end but don't open a dialog window.

Location observed:

History->Add new state ...

55. DW	**Violated principle:** Simple and natural dialogue -graphic design-, Consistency	**Problem classification:** L, Graphical design	**US:** 1

Problem: The spacing and alignment of features within some dialog windows is not good and consistent. Many things look cluttered even though there are not space constraints. Good graphical design is known to help users in learning and using the interface more efficiently.

Location observed:

File->Load database ...
File->New history files
File->Open history files
File->Save current exploration
File->Quit
Options->Preferences ...
Hierarchy->Global grep

56. DW	**Violated principle:** Simple and natural dialogue, Consistency -standards-	**Problem classification:** E, Lack of robustness	**US:** 1

Problem: Some dialog windows can be resized without any limits. This is not the standard way in other systems and there is no meaning on allowing this.

Location observed:

File->Load database ...
File->New history files
File->Open history files
File->Save current exploration
File->Quit
Options->Preferences
History->Modify exploration state name
Hierarchy->Load database ...
Hierarchy->List files matching pattern
Hierarchy->List routines matching pattern
Hierarchy->List identifiers matching pattern
Hierarchy->Global grep
Hierarchy->Find in this list ...
Hierarchy->Go to line ...
Source->Search for text ...
Source->Go to line ...
Help->About Tksee

57. DW	**Violated principle:** Simple and natural dialogue, Consistency -standards-	**Problem classification:** E, Lack of robustness	**US:** 2

Problem: Some dialog windows can be minimized. This is not the standard way in other systems and there is no meaning on allowing this.

Location observed:

File->Load database ...
File->New history files
File->Open history files
File->Save current exploration
File->Quit
Options->Preferences
Hierarchy->Load database ...
Help->About Tksee

58. DW	**Violated principle:** Consistency -standards-	**Problem classification:** L, E, Lack of consistency	**US:** 1

Problem: Some windows have the scroll bar on the left side. This is not the standard way in other systems.
Location observed:
File->Load database ...
File->New history files
File->Open history files
File->Save current exploration

59. DW	**Violated principle:** Simple and natural dialogue	**Problem classification:** E, Non optimal functionality	**US:** 2

Problem: The default size in some edit boxes is too small for the information that needs to be entered. That might force users to resize the window when entering the information.
Location observed:
File->Load database ...
File->New history files
File->Open history files
File->Save current exploration

60. DW, v	**Violated principle:** Simple and natural dialogue, Consistency -standards-	**Problem classification:** L, E, Non optimal functionality	**US:** 3

Problem: Some dialog windows are not well implemented. Some block the main window of the tool but can disappear behind it. Others don't disappear behind but don't allow working correctly on the main window while open.
Location observed:
File->Load database ...
File->New history files
File->Open history files
File->Save current exploration
Options->Preferences
History->Modify exploration state name
Hierarchy->Load database ...
Hierarchy->List files matching pattern
Hierarchy->List routines matching pattern
Hierarchy->List identifiers matching pattern
Hierarchy->Global grep
Hierarchy->Find in this list ...
Hierarchy->Go to line ...
Source->Search for text ...
Source->Go to line ...
Help->About Tksee

61. DW, v	**Violated principle:** Speak users' language, Consistency -standards-	**Problem classification:** L, Possible confusion	**US:** 3

Problem: Some dialog windows present a path-name edit box with a dot as default. This is not the standard way to express the current path in other systems. Users might have problems understanding its meaning.
Location observed:
File->New history files
File->Open history files
File->Save current exploration

62. DW, v	**Violated principle:** Minimize users memory load, Consistency -standards-	**Problem classification:** E, Lack of functionality	**US:** 3

Problem: Directory paths and file names have to be entered manually in some dialog windows. There are not directory browsing facilities like in other systems. Also, there is no indication of the current path. Users will have to enter the whole path and names of files manually. They will also have to remember the current path location.

Location observed:

File->Load database
File->New history files
File->Open history files
File->Save current exploration
Hierarchy->Load database

63. DW	**Violated principle:** Consistency -standards-, Bug	**Problem classification:** E, Lack of consistency	**US:** 2

Problem: The Tab key does not change the focus consistently throughout the tool options. The *Tab* key works but *Shift-tab* doesn't work. This is the standard way in other systems. Also, *Shift-tab* sometimes deletes the selected entry.

Location observed:

File->Load database ...
File->New history files
File->Open history files
File->Save current exploration
File->Quit
Options->Preferences
History->Modify exploration state name
Hierarchy->Load database ...
Hierarchy->List files matching pattern
Hierarchy->List routines matching pattern
Hierarchy->List identifiers matching pattern
Hierarchy->Global grep
Hierarchy->Find in this list ...
Hierarchy->Go to line ...
Source->Search for text ...
Source->Go to line ...
Help->About Tksee -Tab inserts "\t"-

64. DW	**Violated principle:** Speak users' language, Bug	**Problem classification:** E, Wrong behavior	**US:** 3

Problem: Using the *Tab* key within the dialog windows moves the black rectangle representing the current focus through different options. However, pressing *enter* sometimes executes the default action of the window even though it does not have the focus.

Location observed:

File->Quit

65. DW	**Violated principle:** Consistency	**Problem classification:** L, Lack of consistency	**US:** 2

Problem: Some dialog windows have the default button surrounded by a square and other windows don't. This convention is not used consistently.

66. DW, v	**Violated principle:** Error prevention	**Problem classification:** E, Non optimal functionality	**US:** 2
Problem: Some dialog windows present a list of history or database files, and double clicking on them executes the default action of the window. However, this can become dangerous in certain situations such as when saving a history file. **Location observed:** File->Save current exploration			

67. DW	**Violated principle:** Error prevention, Simple and natural dialogue	**Problem classification:** E, Non optimal functionality	**US:** 3
Problem: The *File->Load database* ... dialog window shows all the files in the default directory, however not all files are valid databases that the tool can work with. The same is true for history files in the *File->Open history files* dialog window. Users might select wrong files that can produce unexpected tool behavior. **Location observed:** File->Load database ... File->Open history files			

68. DW	**Violated principle:** Consistency	**Problem classification:** L, Lack of consistency	**US:** 1
Problem: Most dialog windows have a *Cancel* button. However, some have a *Close* button instead but there is not a strong reason why using a different label is better. **Location observed:** Hierarchy->Find in this list ... Source->Search for text ...			

III. RELATED WITH THE HISTORY WINDOW

69. HW	**Violated principle:** Speak users' language	**Problem classification:** L, Possible misinterpretation	**US:** 3-w
Problem: The concept of hierarchy in the History window could be interpreted wrong. For example, a child state could be interpreted as if it performed operations considering only the objects from the parent state. That is, operating over certain subset of objects from the whole database. However, that is not the way it works. Operations in a child state consider objects from the whole database. A child state simply means that it was created after (in time) its parent state but it is a complete state considering everything in the database. Better labeling and training would help users to understand the correct tool functioning.			

70. HW	**Violated principle:** Speak users' language	**Problem classification:** L, E, Possible misinterpretation	**US:** 3-w
Problem: A manually added state in the History window does not work as a real history state. It does not present objects from the database and any operations done in it will add new states in the history window. In other words, such state works just as a label to group history states. However, it presents the same icon as real history states, which doesn't suggest it is different. Users might not understand this behavior. Better labeling and different icons could help. **Location observed:** History->Add new state ...			

71. HW	**Violated principle:** Simple and natural dialogue	**Problem classification:** L, E, Unnecessary functionality	**US:** 3
Problem: The feature *History->Update selected state* doesn't do anything. **Location observed:** History->Update selected state			

IV. RELATED WITH THE HIERARCHY WINDOW

72. HiW, v	**Violated principle:** Feedback, Bug	**Problem classification:** E, Feedback	**US:** 2
Problem: When using *Hierarchy->Find in this list ...*, the searched item is highlighted but the window is not always scrolled to display the selection. Thus, users will need to scroll and look for item in the Hierarchy window. This usually happens when just one item was found. **Location observed:** Hierarchy->Find in this list ...			

73. HiW, v	**Violated principle:** Feedback	**Problem classification:** E, Feedback	**US:** 2
Problem: When using *Hierarchy->Go to line ...*,the matching line is not highlighted. Users will need to discover that the first line in the window at the top is the line matched. **Location observed:** Hierarchy->Go to line ...			

74. HiW, v	**Violated principle:** Feedback -misleading feedback-, Simple and natural dialogue	**Problem classification:** L, E, Feedback	**US:** 3
Problem: In the main window, the message at the bottom-left most times refers to the Hierarchy window. In that case, it does not indicate the status of the Hierarchy window but rather results from previous actions. It might not be clear to the users what those messages refer to. Also, information about previous actions might be useless to the users. When performing new actions users might not remember what the previous actions were.			

75. HiW	**Violated principle:** Error prevention, Feedback -response time-	**Problem classification:** E, Non optimal functionality	**US:** 2-w
Problem: Clicking on an item in the Hierarchy window starts refreshing the Source window. However, source files can be very long and take long to be displayed. Also, clicking on something else while the refreshment is being done does not necessarily stop it. It might be common that users click on something they didn't want and therefore will have to wait until the refreshment finishes or stop it with the stop button.			

76. HiW, v	**Violated principle:** Feedback	**Problem classification:** E, Feedback	**US:** 3
Problem: Sometimes selecting a group of variables and pressing *delete* takes long time to finish, however, the tool does not provide feedback indicating that the operation is being achieved. Thus, users might think that the operation is not being achieved and might try to do it again. **Location observed:** Hierarchy->Delete selected entries			

77. HiW, v	**Violated principle:** Minimize users memory load, Simple and natural dialogue, Feedback -response time-	**Problem classification:** E, Lack of functionality	**US:** 2
Problem: The number of matches in a query can be very large and the query can take long to finish. It might be good to allow the user adjust the maximum number of results to display. **Location observed:** Hierarchy window			

78. HiW	**Violated Principle:** Speak users' language, Consistency	**Problem classification:** L, Possible misinterpretation	**US:** 2-w
Problem: Certain queries in the Hierarchy menu can be interpreted as if they searched just within the current list of objects in the hierarchy window. However, some of them work in this way but others search in the whole database and create a new state in the History window. Users might get confused by this and won't easily learn the way it works. **Location observed:** Hierarchy->List files matching pattern Hierarchy->List routines matching pattern Hierarchy->List identifiers matching pattern Hierarchy->Global grep Hierarchy->New list with selection Hierarchy->Delete selected entries Hierarchy->Delete sublist Hierarchy->Find in this list ... Hierarchy->Go to line ...			

79. HiW	**Violated principle:** Minimize users' memory load	**Problem classification:** L, Bad labeling	**US:** 2
Problem: The labels of some deletion features could be interpreted as if they physically deleted the objects in the database. This is not true but users will have to learn this by experience. **Location observed:** Hierarchy->Delete selected entries Hierarchy->Delete sublist			

80. HiW, v	**Violated principle:** Speak users' language	**Problem classification:** L, Bad labeling	**US:** 1
Problem: The feature *Hierarchy->Delete sublist* can be misinterpreted. For example, it could be interpreted as if the currently selected item will be deleted, which in fact it won't. It might be better to label it *Delete sub-hierarchy below selected.* **Location observed:** Hierarchy->Delete sublist			

81. HiW, v	**Violated principle:** Feedback –misleading feedback-	**Problem classification:** L, E, Feedback	**US:** 3
Problem: Some deletion features in the Hierarchy window remove objects in the hierarchy but leave the corresponding state in the History window unaffected. Therefore, the label of such history state does not correspond to the objects displayed in the Hierarchy window after having deleted. **Location observed:** Hierarchy->Delete selected entries Hierarchy->Delete sublist			

82. HiW, v	**Violated principle:** Simple and natural dialogue	**Problem classification:** L, E, Non optimal functionality	**US:** 2
Problem: Many important features are located in middle mouse button menus. It is known that such menus are hard to discover. Although they are sometimes used in other systems it is necessary to consider the trade-off involved. Also, the middle mouse button in a mouse with two buttons is implemented by clicking both buttons at the same time, which is hard to discover as well. Users won't easily discover these menus and therefore the features provided in them.			

83. HiW, v	**Violated principle:** Minimize users' memory load	**Problem classification:** L, E, Non optimal functionality	**US:** 3
Problem: The tool provides several different menus depending on (a) where the mouse is located, (b) what mouse button is clicked and (c) what objects are highlighted. It'll be hard for the users to remember which buttons have to be clicked in order to bring a desired menu. **Location observed:** Hierarchy menu Middle mouse button menus			

84. HiW	**Violated principle:** Shortcuts	**Problem classification:** E, Lack of functionality	**US:** 1
Problem: Some features of the tool can only be accessed through middle-mouse button menus. Users that not discover these menus will never discover the features. It would be better to have other ways to access them. **Location observed:** File middle-mouse button pop-up menu->Included files File middle-mouse button pop-up menu->Files that include me File middle-mouse button pop-up menu->Defined variables File middle-mouse button pop-up menu->Reported problems File middle-mouse button pop-up menu->Referred activities File middle-mouse button pop-up menu->Referred technical terms Routine middle-mouse button pop-up menu->File defining me Routine middle-mouse button pop-up menu->Referred variables Routine middle-mouse button pop-up menu->Reported problems Routine middle-mouse button pop-up menu->Referred activities Routine middle-mouse button pop-up menu->Referred technical terms Variable middle-mouse button pop-up menu->Files using me Variable middle-mouse button pop-up menu->Routines using me			

85. HiW, v	**Violated principle:** Feedback -misleading feedback-	**Problem classification:** E, Feedback	**US:** 3
Problem: Sometimes selecting an object in the Hierarchy window doesn't refresh the Source window. Different behaviors were noticed when using different databases.			

86. HiW	**Violated principle:** Feedback –misleading feedback-	**Problem classification:** L, E, Feedback	**US:** 3
Problem: In the Hierarchy window, selecting a routine and using *Routines I call* and its equivalent TREE feature leads to having the child routines displayed twice. The same is true for *Routines that call me* and its equivalent TREE feature. Users might get confused with repeated information in the Hierarchy window. They will have to discover that it is only repeated information.			

87. HiW	**Violated principle:** Feedback –misleading feedback- Error prevention	**Problem classification:** E, Feedback	**US:** 3
Problem: The feature to see the routines contained in a file displays the routines alphabetically. However that's not the order they are within the file. Users could wrongly assume that is the real order. **Location observed:** File middle-mouse button, pop-up menu->Routines in me			

88. HiW	**Violated principle:** Speak users' language	**Problem classification:** L, Possible misinterpretation	**US:** 3
Problem: In the middle mouse pop-up menus, the Grep feature could be misinterpreted. For example, it's not clear what objects it will involve. It might be better to change the label for *Grep in selection* instead of *Grep....* **Location observed:** File middle-mouse button pop-up menu->Grep ... Routine middle-mouse button pop-up menu->Grep ...			

89. HiW	**Violated principle:** Simple and natural dialogue –less is more-	**Problem classification:** E, Non optimal functionality	**US:** 2
Problem: The Grep feature gives the possibility to change the grep command that is executed. It is not likely that users will frequently change that. A better alternative would be to move the option into the preferences dialog window. **Location observed:** File middle-mouse button pop-up menu->Grep Routine middle-mouse button pop-up menu->Grep			

90. HiW, v	**Violated principle:** Speak users' language	**Problem classification:** L, Possible confusion	**US:** 3
Problem: The name of the *auto-grep* feature is not a known term and its meaning is not very clear. Also, the way it has to be operated is not very easy to understand. Changing the labeling might help, for example *Grep for x in y* instead of *Autogrep* (where x and y are dynamically specified). **Location observed:** File middle-mouse button pop-up menu->Autogrep Routine middle-mouse button pop-up menu->Autogrep Variable middle-mouse button pop-up menu->Autogrep			

V. RELATED WITH THE SOURCE WINDOW

91. SW	**Violated principle:** Speak users' language	**Problem classification:** L, Possible misinterpretation	**US:** 1
Problem: The label *Open current file in a text editor* does not clearly suggest what the feature does and it can be misinterpreted. For example, a novice user might think it would open the history file. **Location observed:** Source->Open file in editor			

92. SW	**Violated principle:** Simple and natural dialogue	**Problem classification:** L, Unnecessary functionality	**US:** 2
Problem: The feature *Source->Highlitght text* doesn't work anymore but it is still there. Users will try to use any features available in order to solve their tasks. Every extra feature in the system is something more that users will have to learn and remember. If a feature is there but doesn't work, it will make harder learning the tool. **Location observed:** Source->Highlitght text			

93. SW	**Violated principle:** Speak users' language, Consistency -standards-	**Problem classification:** L, Possible misinterpretation	**US:** 1
Problem: In the Source window, the information at the top of the window looks like editable but is not. The reason for this box is that the information might be copied and pasted, however it's always better to use widgets for its intended purposes. Users who know how that widget works in other systems will try to use it in the way it works in other systems. **Location observed:** Source window			

94. SW	**Violated principle:** Simple and natural dialogue	**Problem classification:** L, Possible confusion	**US:** 1
Problem: The label *Source* in that window suggests that source code will be displayed in it. However, no source code is displayed in the case of selecting a variable in the Hierarchy window, therefore *Source* is not a very good label for the window.			

VI. RELATED WITH THE TOOL BAR AND SHORTCUTS

95. TS	**Violated principle:** Shortcuts, Simple and natural dialogue	**Problem classification:** E, Lack of functionality	**US:** 1
Problem: The tool is mouse dependent. It is not possible to navigate through the different interface elements with the keyboard. Some examples are: a) Certain important operations don't have a keyboard shortcut. For example, *Global grep*, opening a Source Code file, etc. b) The Tab key doesn't work consistently changing the focus in the interface. c) It is not possible to access the menus through the keyboard.			

96. TS	**Violated principle:** Minimize users' memory load, Error prevention	**Problem classification:** L, E, Possible confusion	**US:** 3
Problem: Some different features in different menus have the same keyboard shortcut. The same shortcut works different depending on the context established by the current focus of the mouse. Users will have to learn that and always remember the current context in order to use the shortcuts. This presents extra effort to users and is also prone to mistakes. **Location observed:** Hierarchy->Find in this list ... Source->Search for text ... Hierarchy->Go to line ... -related with the previous problem- Source->Go to line ... -related with the previous problem-			

97. TS	**Violated principle:** Simple and natural dialogue -less is more-, Shortcuts	**Problem classification:** E, Non optimal functionality	**US:** 2
Problem: It is not clear why the selection of the buttons presented in the Tool bar is adequate and enough. Certain important features don't have a button and some actions that are not very important or frequent have a button (e.g. creating a new history file might not be very frequent but it has a button).			

98. TS	**Violated principle:** Simple and natural dialogue	**Problem classification:** L, E, Possible misinterpretation	**US:** 3
Problem: In the middle group of buttons, the icons of the first and third buttons (left-to-right) suggest they perform opposite actions. However, this is not always true. In the case of working with routines they perform opposite operations but not in the case of files. In this last case, the third button doesn't even work. Also, the help flag in the first button always informs that it works on files. **Location observed:** Tool bar, middle group			

99. TS	**Violated principle:** Simple and natural dialogue –less is more-	**Problem classification:** L, Graphical design	**US:** 1
Problem: There is a little arrow in the upper-left corner of the help flags. It is most times hidden back and mixed with the mouse. It is not useful at all and might distract users a bit.			

100. TS	**Violated principle:** Simple and natural dialogue -graphic design-	**Problem classification:** L, Graphical design	**US:** 3
Problem: The icons for *List files matching pattern*, *List routines matching pattern*, *List identifiers matching pattern* are not very descriptive of the operations they perform. Users might not easily see how they work.			

101. TS	**Violated principle:** Error prevention	**Problem classification:** E, Non optimal functionality	**US:** 1
Problem: The *exit* button is very close to the stop query button. Users could accidentally press it when trying to stop the processing of a query and then start exiting from the tool.			

102. TS	**Violated principle:** Simple and natural dialogue -graphic design-	**Problem classification:** E, Non optimal functionality	**US:** 1
Problem: The icon for stopping queries is very far from the place where the queries are started. Users will have to make long mouse movements whenever they use it. This is not a serious problem but will render users' performance.			

103. TS	**Violated principle:** Simple and natural dialogue -less is more-	**Problem classification:** E, Lack of robustness	**US:** 1
Problem: Contracting the History window affects the shape of the tool bar. Also, it is possible to resize the history window from the top or from the bottom, however one way would be enough.			

104. TS	**Violated principle:** Consistency -standards-	**Problem classification:** L, Lack of consistency	**US:** 1
Problem: The icons for new, open and save history are on the right, however the corresponding operations in the main menu are on the left. This is not the way other systems implement it. Users with experience in other systems will probably look for these buttons at the left under its menu option.			

105. TS	**Violated principle:** Speak users' language, Simple and natural dialogue -graphic design-	**Problem classification:** L, Graphical design	**US:** 2
Problem: The icon for opening a Source Code file in an editor is very similar to the one for creating a new history file. However, both operations are completely different. It won't be easy for users to learn the difference.			

106. TS	**Violated principle:** Speak users' language	**Problem classification:** L, Bad labeling	**US:** 1
Problem: The help flag of the icon for opening a Source Code file doesn't suggest which type of file will be opened (e.g. whether source code or history file). **Location observed:** Tool bar->Open current file in an editor, Help flag			

VII. RELATED WITH THE PREFERENCES DIALOG WINDOW

107. PW	**Violated principle:** Simple and natural dialogue -graphic design-	**Problem classification:** L, Graphical design, Bad labeling	**US:** 3
Problem: This dialog window presents several graphic design problems. Some of them are: a) Many labels are redundant, badly phrased or can be interpreted in different ways. b) The spacing, alignment and symmetry of features are not consistent and aesthetically good. c) There is unnecessary space between the options and the buttons at the bottom. Problem a) is more critical because users can have problems understanding the features of the tool. Problems b) and c) are not very serious but contribute to the good overall graphical design of the interface. Good graphical design is known to help users at learning and using the interface more efficiently.			

108. PW	**Violated principle:** Simple and natural dialogue -graphic design-, Speak users' language	**Problem classification:** L, Graphical design	**US:** 1
Problem: Many features are not grouped properly. For example, (a) some group boxes contain just one option, (b) it is no clear that certain groupings are the best possible, (c) certain related features are not grouped together, etc. These are not serious problems but contribute to the good overall graphical design of the interface.			

109. PW	**Violated principle:** Speak users' language	**Problem classification:** L, Bad labeling	**US:** 2
Problem: Under the *General Preferences* option, the label *External Development tools* doesn't clearly suggest what it is for, what tools it refers to, etc.			

110. PW	**Violated principle:** Simple and natural dialogue -graphic design-	**Problem classification:** L, Non optimal functionality	**US:** 1
Problem: The type of drop-down list currently used for selecting options (e.g. fonts, sizes, etc.) is not the best widget that could be used. It might be better to use another like the one used in Microsoft word, for example. This is not a serious problem but contributes to the good overall graphical design of the interface.			

VIII. RELATED WITH THE GLOBAL GREP WINDOW AND FEATURE

111. GG	**Violated principle:** Consistency -standards-	**Problem classification:** L, E, Lack of consistency	**US:** 1
Problem: The radio buttons in the *Global-grep* feature are not used according to the standards. This type of buttons is normally not used for representing exclusive or. This is not a serious problem but contributes to the good overall graphical design of the interface.			

112. GG	**Violated principle:** Simple and natural dialogue, Bug	**Problem classification:** L, Non optimal functionality	**US:** 1
Problem: The dialog window displayed when executing this feature is opened very slowly and grows incrementally. This is not a serious problem but will distract the users.			

113. GG	**Violated principle:** Simple and natural dialogue -graphic design-	**Problem classification:** L, E, Graphical design	**US:** 1
Problem: This window doesn't use the group boxes properly. The first group box has too many group boxes nested together. The group box at the bottom contains just one item. This is not a serious problem but contributes to the good overall graphical design of the interface.			

114. GG, v	**Violated principle:** Simple and natural dialogue	**Problem classification:** L, E, Possible confusion	**US:** 3
Problem: The way that the *Global grep* feature works is not very easy to understand from the design of its dialog window. For example: a) The pattern that will be looked for is the most important aspect in the search, however its edit box is presented at the bottom of the window, which doesn't suggest is the most important. b) The meaning of the options is not very to understand from reading the labels. c) The three sentences in the group box in the middle do not clearly suggest a partition of the different possibilities. d) The *Glob-style* term is not a well-known term for all the users. e) The labeling in the group box at the bottom is redundant.			

APPENDIX 3

Tasks used in the Thinking Aloud Usability Testing
(final version)

I. Suppose you are investigating about the signal tones that are given to peripheral devices. Right now you want to know more about the message that is sent to the Network Controller when the ***apply_tone_signal*** procedure is called, therefore:

1. Start Tksee.
2. Load the database corresponding to the SX-2000 system (***sx.main.q10.opq.mf***).
3. You want to keep the exploration that you are about to perform, therefore create a new history file and use your name as its name.
4. Find the definition of the ***apply_tone_signal*** procedure.
5. For each parameter, find its name and type. Also find out how that type is defined (e.g. byte, integer, record, pointer, etc.). Write down all that information.
6. Looking at the implementation of ***apply_tone_signal***, look for any information about the purpose of the ***tone_signal*** parameter. Write down whether you found anything or not, and give a clue where it is in case you found something.

II. Now, in the SUMMARY information of that procedure it says that the ***tone_signal*** parameter specifies the tone signal that'll be given to the peripheral device. However imagine you want to find out if a tone signal given as argument to the procedure is the tone signal actually sent to the Network controller or if another tone signal might be sent instead. In order to investigate that:

7. Search within the ***apply_tone_signal*** procedure to see if there is any place(s) where the ***tone_signal*** parameter is being reassigned. Write down how many places you found.
8. Also, find all the functions/procedures called within the ***apply_tone_signal*** procedure. Write down how many routines you found.
9. See if, in these calls, the value of the ***tone_signal*** parameter could be modified. Write whether or not the parameter could be modified. Give the name of each procedure/function that you found could modified it.
10. Save your recent exploration (using the history file you created).

III. From the previous steps, you found that within the ***apply_tone_signal*** procedure, the ***tone_signal*** parameter could only be modified in the call to ***music_to_be_applied***. Now suppose you want to investigate more about how this could happen:

11. Remove any information displayed in the Hierarchy window that you consider unnecessary.

12. Locate the implementation of ***music_to_be_applied***.

13. Look if there is any place(s) where the ***tone_signal*** parameter is being reassigned and write down all the possible reassigned values.

14. Find the value(s) reassigned to the ***tone_signal*** parameter whenever the function ***music_to_be_applied*** returns with TRUE status. Write down the value(s).

15. For that value, locate its definition and write down to what number it is equivalent.

16. Go to the file where ***music_to_be_applied*** is implemented and look for line 1915.

17. What would be the effect in the SX-2000 system if you deleted that line ? (Hint: look at some of the previous if statements). Write down your answer.

18. Looking at the implementation of ***music_to_be_applied***, find and read its SUMMARY information.

IV. From what you read in the SUMMARY you know that the function ***music_to_be_applied*** works when the user is put on hold. Now, imagine that you want to know if any problems have arisen when the music is set on hold. Therefore:

19. Locate all possible problems associated with ***music_to_be_applied*** and which are related to ***"music on hold"***. Write how many problems you found as well as the ***sms-number*** of the problems.

20. Locate all possible problems associated with ***apply_tone_signal*** and which are related to ***"music on hold"***. Write how many problems you found as well as the ***sms-number*** of the problems.

21. Save your recent exploration (using the history file you created) and exit Tksee.

V. Suppose you want to convert the ***apply_tone_signal*** procedure into a function. This would enable you to use its return status. However before doing any change you want to know some of the impacts. Therefore:

22. Start Tksee and open the history file of your previous exploration.

23. Create a new list in the Hierarchy window containing the ***apply_tone_signal*** procedure. Here you will continue with your exploration.

24. Locate the file(s) that would be modified if you converted the ***apply_tone_signal*** procedure into a function. Write down the names of the file(s).

25. Locate the file containing the implementation of ***apply_tone_signal*** and open it in the editor.

26. Close the editor (don't make any changes to the file).

VI. You want to know more about some of the places where you could use a value returned by ***apply_tone_signal***, therefore:

27. Find all the functions/procedures that call ***apply_tone_signal***. Write down how many they are.

VII. You are just interested in the functions/procedures starting with "a", therefore:

28. Remove any information displayed in the Hierarchy window that you consider unnecessary.

29. Now, considering only the functions/procedures starting with "a", find all the places where you could make use of a value returned by apply_tone_signal. Write down how many places you found.

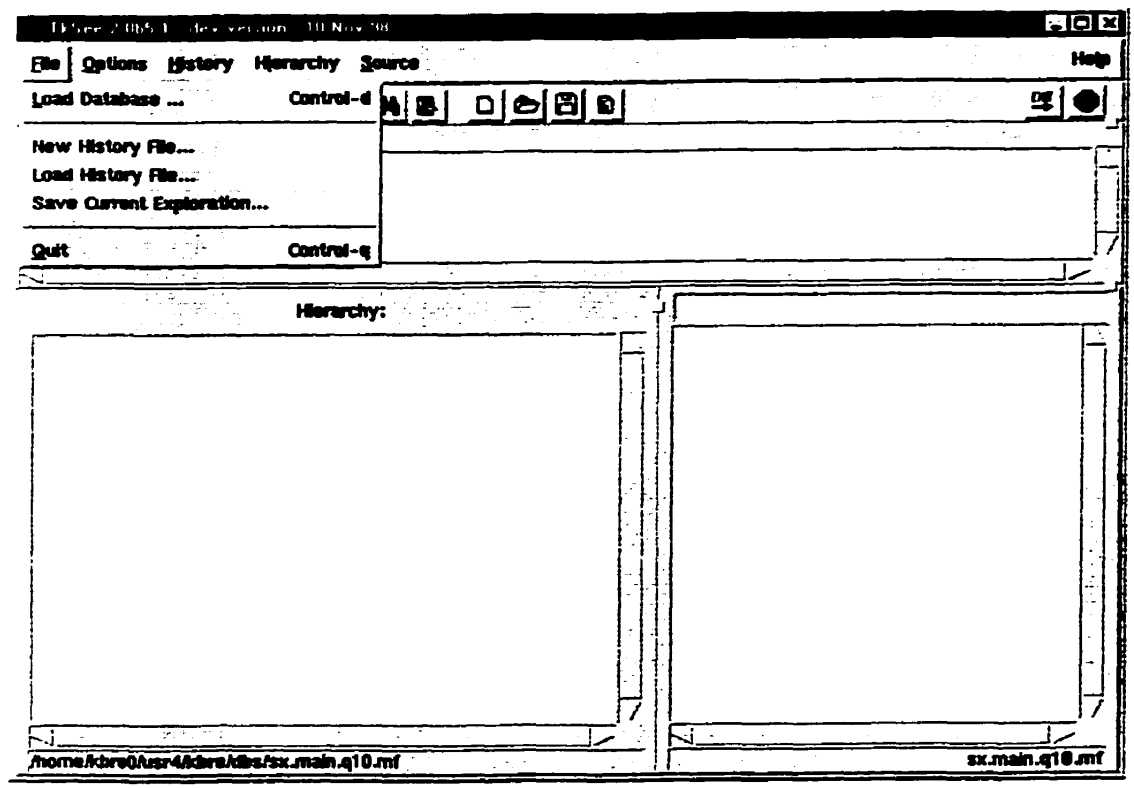
VIII. Suppose that in all places found in the previous step, you decided to use a variable to capture a value returned by ***apply_tone_signal***, therefore:

30. Locate the files that would be modified by implementing this change. Write down the names of those files.

31. Save your current exploration and exit Tksee.

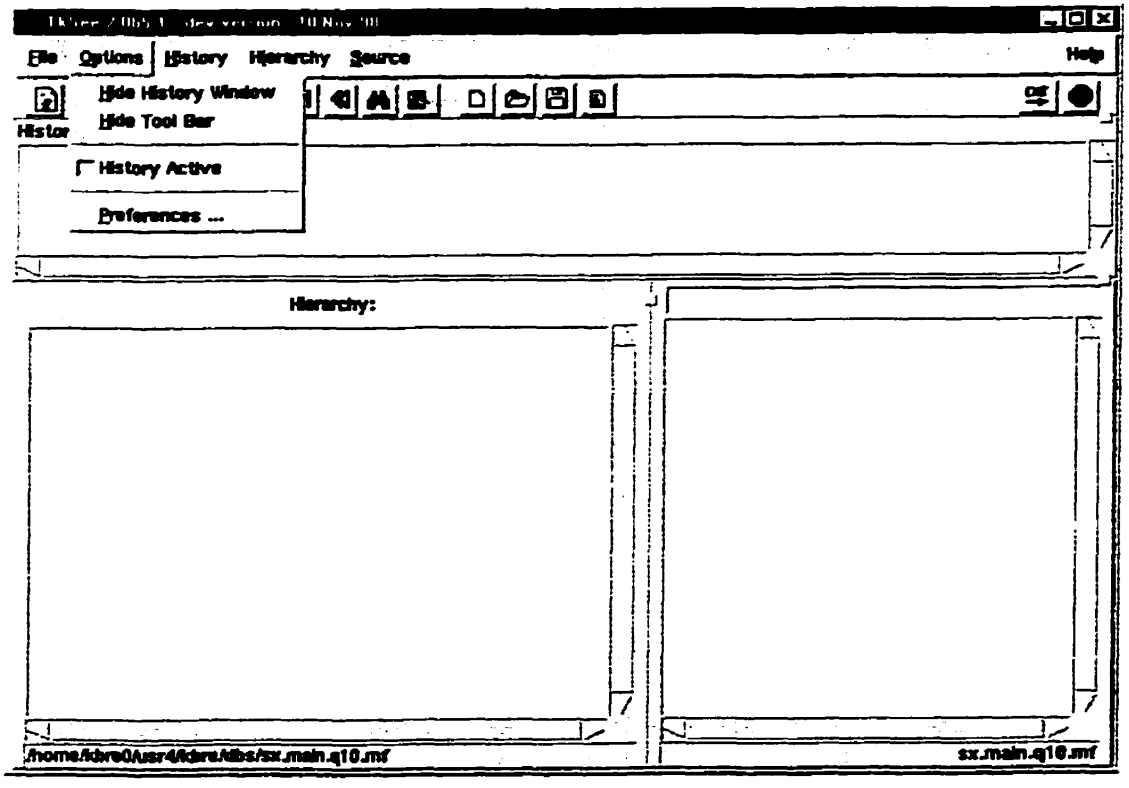
APPENDIX 4
The Tool before the Thinking Aloud Usability Testing
(including changes after the Heuristic Evaluation)

1)

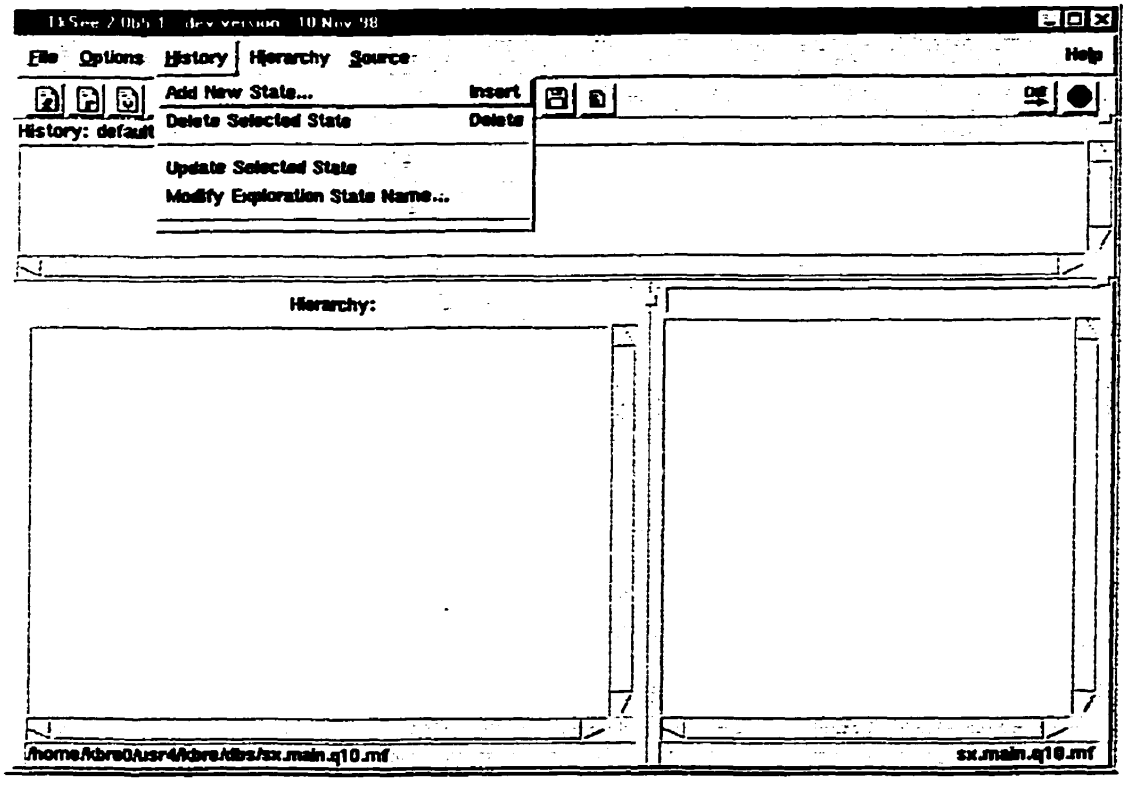


Change	Description	Before	After
1	Change of label (solution to problem 33, Appendix 2)	*File->New history files*	*File->New History File...*
2	Change of label (solution to problem 33, Appendix 2)	*File->Load history files*	*File->Load History File...*

2)

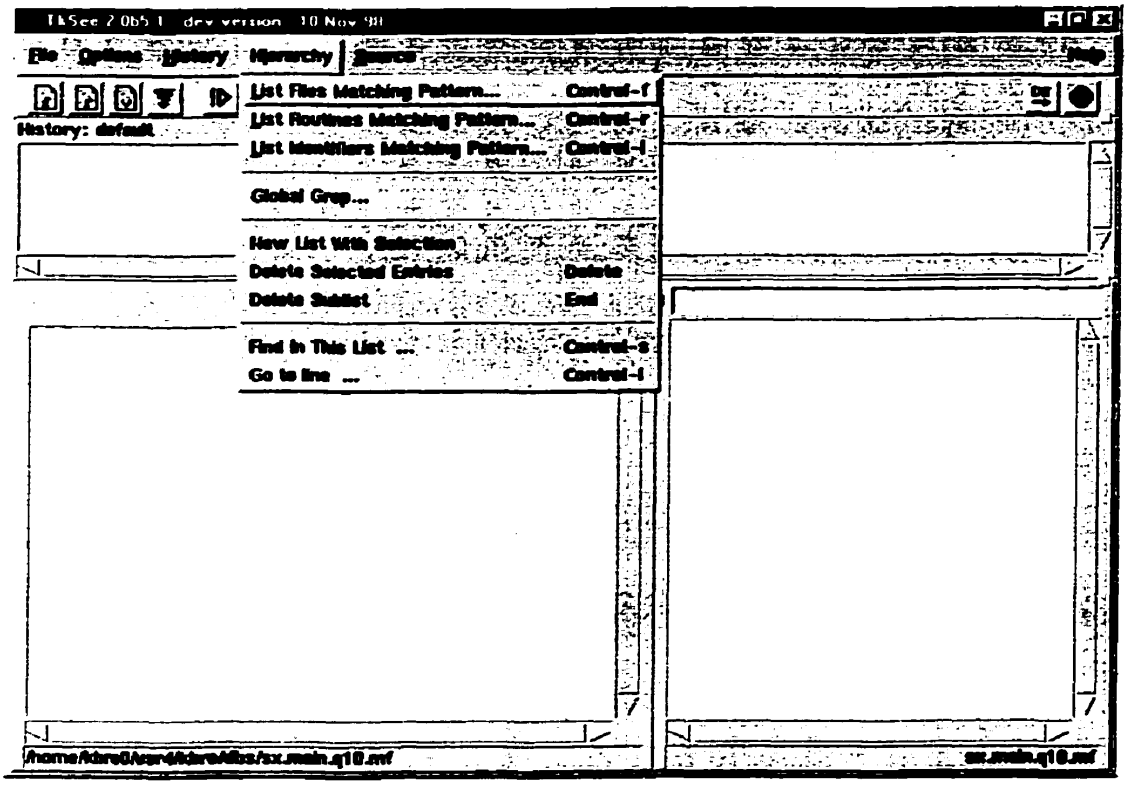

File Options History Hierarchy Source
Help
Hide History Window
Hide Tool Bar
History Active
Preferences ...
Hierarchy:

3)


File Options History Hierarchy Source
Help
Add New State... Insert
Delete Selected State Delete
Update Selected State
Modify Exploration State Name...
History: default
Hierarchy:

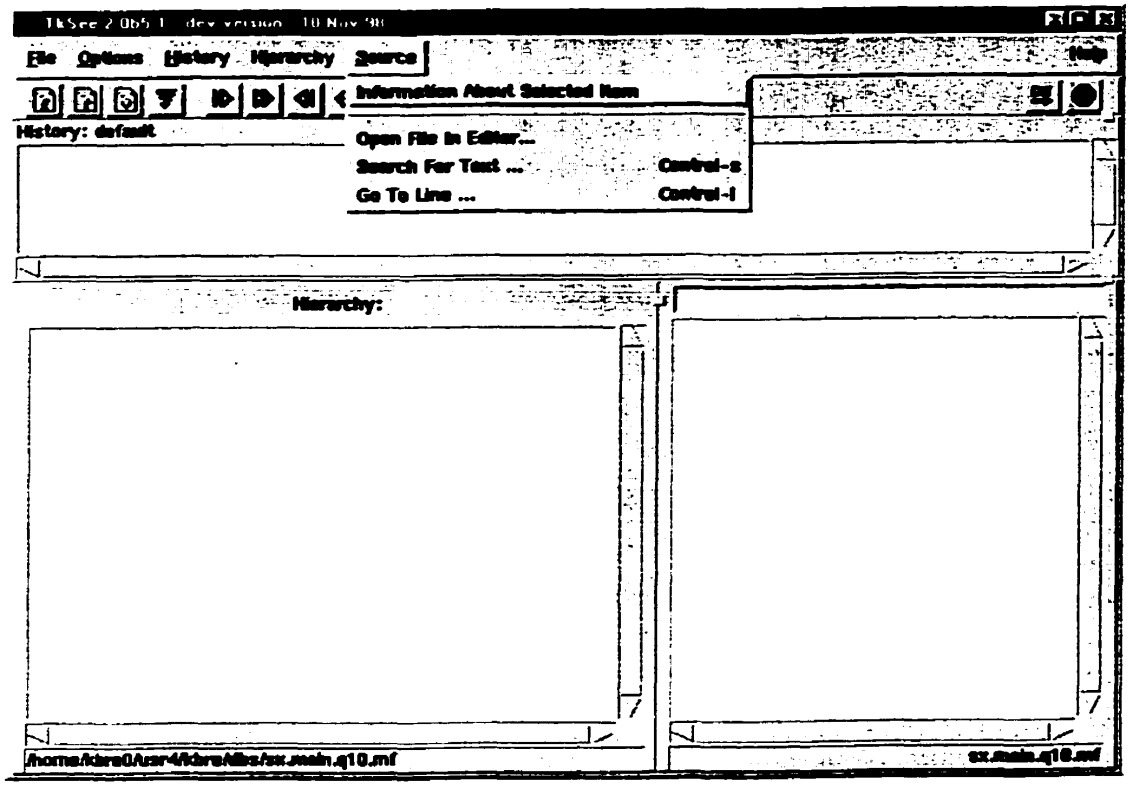
Change	Description	Before	After
3	Feature removed (solution to problem 12, Appendix 2)	*History ->* *Hide history window*	None

4)



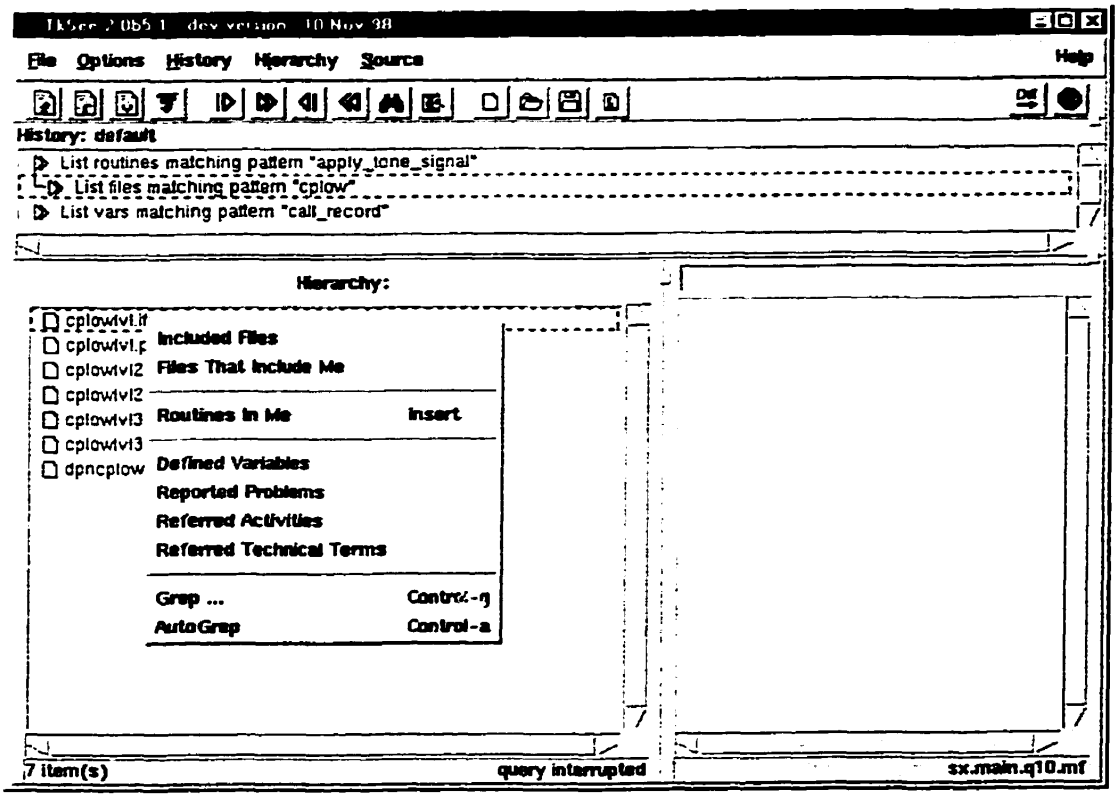
Change	Description	Before	After
4	Feature removed (solution to problem 26. Appendix 2)	*Hierarchy ->* *Load data base ...*	None

5)

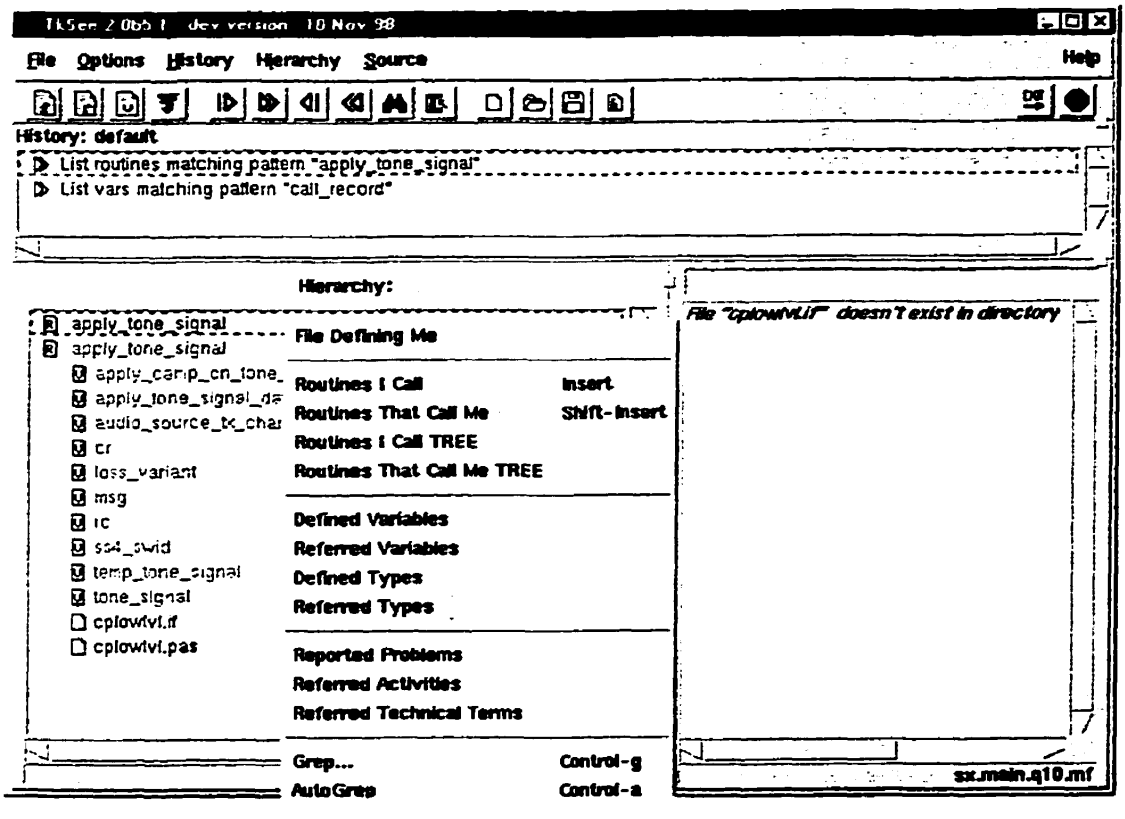


Change	Description	Before	After
5	Feature removed (solution to problem 92. Appendix 2)	*Source* -> *Highlight text*	None

6)

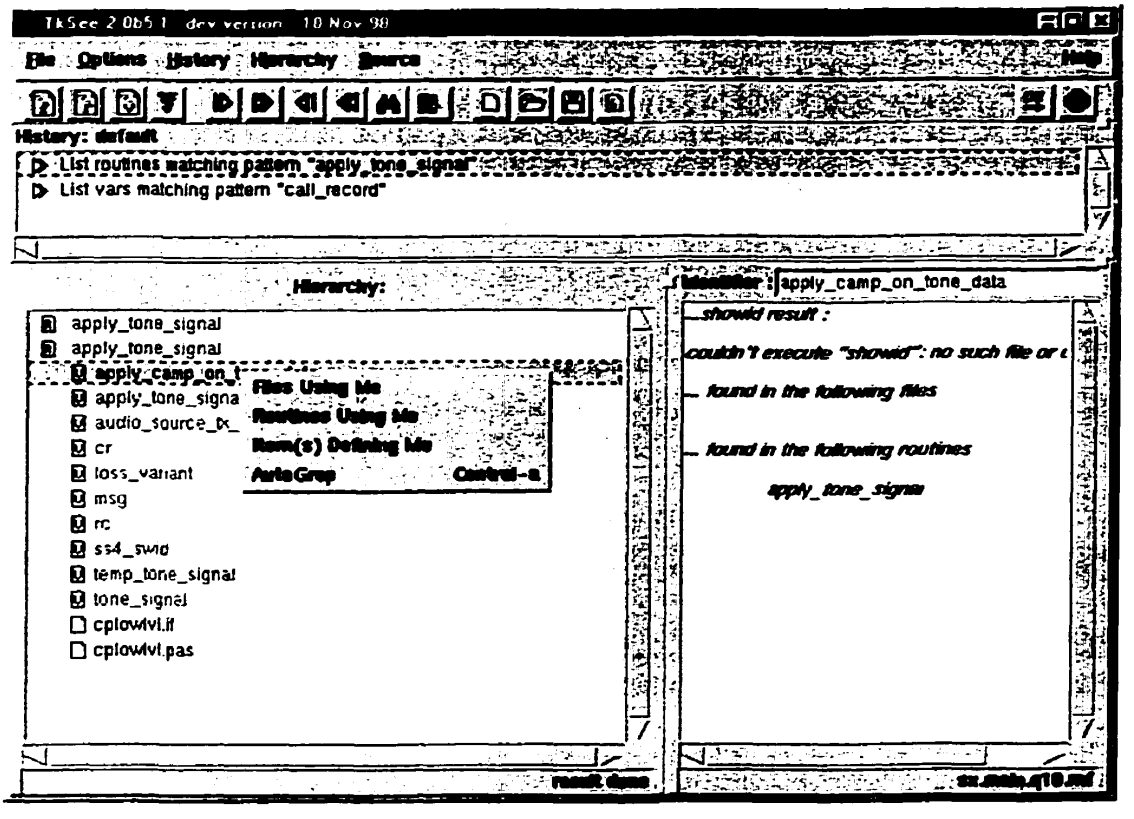

File Options History Hierarchy Source Help
History: default
List routines matching pattern "apply_tone_signal"
List files matching pattern "cplow"
List vars matching pattern "call_record"
Hierarchy:
cplowivi.if
cplowivi.p
cplowivi2
cplowivi2
cplowivi3
cplowivi3
dpncplow
Included Files
Files That Include Me
Routines In Me
Insert
Defined Variables
Reported Problems
Referred Activities
Referred Technical Terms
Grep ...
AutoGrep
Control-a
7 item(s)
query interrupted
sx.main.q10.mf

7)


File Options History Hierarchy Source Help
History: default
List routines matching pattern "apply_tone_signal"
List vars matching pattern "call_record"
Hierarchy:
apply_tone_signal
apply_tone_signal
apply_camp_on_tone_
apply_tone_signal_da
audio_source_tx_char
cr
loss_variant
msg
rc
ss4_swid
temp_tone_signal
tone_signal
cplowivi.if
cplowivi.pas
File Defining Me
Routines I Call
Insert
Routines That Call Me
Shift-Insert
Routines I Call TREE
Routines That Call Me TREE
Defined Variables
Referred Variables
Defined Types
Referred Types
Reported Problems
Referred Activities
Referred Technical Terms
Grep...
Control-g
AutoGrep
Control-a
File "cplowivi.if" doesn't exist in directory
sx.main.q10.mf

Change	Description	Before	After
6	Feature added (solution to problem 49, Appendix 2)	None	*Hierarchy window, Routine object, Middle mouse button -> Defined Variables*
7	Feature added (solution to problem 49, Appendix 2)	None	*Hierarchy window, Routine object, Middle mouse button -> Defined Types*
8	Feature added (solution to problem 49, Appendix 2)	None	*Hierarchy window, Routine object, Middle mouse button -> Referred Types*

8)



Change	Description	Before	After
9	Feature added (solution to problem 49, Appendix 2)	None	*Hierarchy window, Variable object, Middle mouse button -> Item(s) Defining Me*

APPENDIX 5

Usability Problems found in the Thinking Aloud Usability Testing

The following table shows the times during the sessions with the participants.

Session No.	Tasks time	Session time	Participant Type
1	1:33 hr	1:55	Novice
2	0:57 hr	1:12 hr + 17 min tutorial	Novice
3	1:22 hr	1:48 hr + 17 min tutorial	Novice
4	0:44 hr	0:58 hr + 17 min tutorial	Novice
5	1:32 hr	1:48 hr + 17 min tutorial	Novice
6	0:51 hr	1:05	Expert
7	1:02 hr	1:15	Expert
8	0:56 hr	1:20	Expert
	Novices only	28	problems
	Experts only	9	problems
	Both	35	problems
	TOTAL	72	problems

The following notation is used in the first row of the tables:

1. The first field indicates the problem number and its location within the interface according to the codes below. It sometimes presents two capital letters that indicate the problem is related with other problems marked with the same letters (**AA**, for example).

   OW: Overall in the Interface
   DW: about Dialog Windows opened by interface features
   HW: related to the History Window
   HiW: related to the Hierarchy Window
   SW: related to the Source Window
   TS: related with the Tool bar and Shortcuts
   PW: related with the Preferences dialog Window
   GG: related with the Global Grep feature and its window

2. The second field indicates the type of participants that experienced the problem.

3. The third field indicates whether the problem was about *Learning* (letter "**L**"), *Efficiency of use* (letter "**E**") or both. It also contains a short description of the problem.

PROBLEMS:

I. OVERALL IN THE INTERFACE

1.	**Problem observed in:**	**Short description:**
OI	2 experts	E, Problems with nested routines

Problem: The tool didn't work very well with nested functions/procedures.

- They were not displayed as icons in the Hierarchy window. Therefore participants looking for one had to find the parent routine and scan into its code.
- In the Source window, within the shadow area of a routine, there was no way to differentiate the code of a nested routine from the code of the parent routine. Therefore participants looking at the code of the parent routine were mislead by the code of the nested routine. They required some time to notice the nested routine.
- One participant commented: *"The tool doesn't show information about nested routines, however there are many of them in the Company system"*

2.	**Problem observed in:**	**Short description:**
OI	1 expert	L, The term ***opq*** is not known by everybody

Problem: Not all ***experts*** knew about ***opq***, therefore not all understood well the names of the different databases that could be loaded.

3.	**Problem observed in:**	**Short description:**
OI	2 novices, 3 experts	E, Problems with *Go to line* ... features

Problem: There were several observations with the *Go to line* ...features. First of all, the tool didn't indicate line numbers, therefore, it was very hard to find the number of a specific line. Normally, when a user wants to go to a certain line number is because he knows that in such line there is something he is looking for. During the testing participants didn't use line numbers as a way to locate information. Experts hadn't even noticed the existence of *Go to line* ...features. In one task they were asked to go to a certain line number in the Source window. Most were able to do it because they found the feature during the session. However, it seemed that in general they had not used the feature:

- Because the difficulty of seeing the number of a specific line and use it later to locate such line.
- Because they are very used to use line numbers as a way to locate information.

4.	**Problem observed in:**	**Short description:**
OI	several users,	L, Tool labels not according to users concepts
EE	1 novice commented	

Problem: In their work users know about ***files, procedures, functions, variables*** and ***types***, however:

a) The tool uses the label *routine* to refer to *functions* and *procedures*.

b) The tool uses the label ***identifiers*** to refer to ***variables*** and ***types***.

Because of this some participants didn't know exactly what those features were going to do. In general, participants tended to briefly *scan* the labels in the interface and match them with their concepts. Just after several unsuccessful tries, participants more carefully read and tried to understand the labels.

5. OI	**Problem observed in:** 1 novice	**Short description:** E, Users are unnecessarily asked to name a history file when it is created

Problem: Whenever participants created a history file, the tool asked to name it. However, this was unnecessary work:

- The name of the file would be asked when saving anyway.
- When prompted to give a name, participants tended to use the name suggested by the tool. It was clear that they didn't want to name the file at that moment.

6. OI	**Problem observed in:** 1 novice	**Short description:** E, Name of the current *history file* is not always seen or understood

Problem: Participants were asked to create a history file and name it using their names. Also, they were asked to save the exploration on such history file several times during the session. One participant incorrectly entered the name of the file in the *Path name* field and saved the exploration. He never realized that the exploration had being saved with the *default* name and not with his name. Later, he was asked to start the tool again and load his history file. The *default* history file was loaded automatically and he saw his exploration on the windows. Again, he never realized that the exploration was named with the *default* name and not with his name. All the time a label indicating the name of the history file containing that exploration (e.g. *default*) had been displayed at the top of the History window. Therefore, he either didn't see the label or never understood it.

7. OI	**Problem observed in:** 1 novice noticed that	**Short description:** E, What's the meaning of loading a database and continue with the same history file ?

Problem: When users load a database they have the option of *continuing with the same history file*. However, one participant following that option didn't understand if all previous information in the exploration would be kept or overwritten. In a similar way, if a user starts the tool for the first time, what would be the meaning of the ***"same"*** history file ?

8. OI	**Problem observed in:** 2 novices	**Short description:** L, The label *history* is displayed in several places in the tool and confuses users

Problem: The options for the ***History file*** reside within the ***File*** menu. However, there is another menu called ***History*** that misled participants. Participants were asked to create a new *history file* and some decided to open the *History* menu and look in it for an option to create a *history file*. Specially when learning, participants noticed different places with the label ***History*** and they didn't understand the differences very easily.

9. OI	**Problem observed in:** 1 expert	**Short description:** E, Strange behavior of the tool when saving exploration

Problem: One participant when saving his exploration entered a name in the dialog window and pressed the button for saving. Then, all the information in the tool was blanked for one second and came back again. Because it had been fast, it didn't confuse the participant a lot but he noticed this strange behavior.

10. OI FF	**Problem observed in:** 2 novices	**Short description:** **E**, Problems copying and pasting

Problem: There were several problems observed when participants tried to copy and paste. The facilities to do that were implemented by the operating system and not by the tool, therefore:

a) Not all participants knew how to use them.

b) Certain dialog windows didn't allow participants to copy from the tool and paste in their fields. For example:

- A participant clicked on *List Identifiers Matching Pattern...* and couldn't copy from the Source window to the input field of the dialog window.
- A participant highlighted something in Source window, clicked on *List Identifiers Matching Pattern...* and tried to use the *copy-paste* keys of the keyboard. This didn't work.

11. OI	**Problem observed in:** 3 novices, 3 experts	**Short description:** **E**, Some dialog windows inefficiently block the tool

Problem: Some dialog windows partially and inefficiently blocked the tool while they were opened. While the dialog window were opened participants could still select certain things in the main window, however no refreshment of information was done in the Source window as it is normally done when there was no dialog window. Many participants tried to continue working while a dialog window was open, but they were confused when certain things worked and others didn't. It wasn't obvious for them that closing the dialog window would unblock the tool. In summary, some dialog windows that were supposed to block didn't block everything, and others supposed to allow working while open didn't fully allow it.

Situations observed:

- The participant left the *Search For Text ...* window open and clicked on a different routine in the Hierarchy window. The code of the selected routine was not refreshed in the Source window. Then, he closed the *Search For Text ...* window and the refreshment was automatically done.
- The participant left the *Search For Text ...* window open and clicked on a different routine in the Hierarchy window. The routine was selected but its code was not refreshed in the Source window. The participant didn't understand why. Then, with the focus in the routine that he selected he went to a different history state. Later, he came back to the previous history state and this time the Source window was refreshed with the code of the routine that had the focus when he left. He didn't notice the refreshment and he used *Search For Text ...* to find matches in that code.

12. OI	**Problem observed in:** 1 novice commented that	**Short description:** **E**, Does loading a history file loads the correct database ?

Problem: This was not an problem but something a participant asked, when loading a history file is the corresponding database loaded as well ?

13. OI CC	**Problem observed in:** 2 novices	**Short description:** **L**, How to combine the functionality of the Hierarchy window and Source window is not always obvious

Problem: Much of the power of the tool resides in combining functionality between the Source and Hierarchy windows. However, for some novices it wasn't obvious how to combine that functionality in order to solve a task. For example, it wasn't obvious for them that if something was highlighted in the Source window and *Information About Selected Item* was used, then the information would appear in the Hierarchy window. Also, it was not obvious that they would have to click on the new object in the Hierarchy window so that the information was displayed in the Source window. This seemed to be obvious once they knew it but they required time to understand it.

14.	Problem observed in:	Short description:
OI	1 novice	L, Relationship between main menus and windows not noticed

Problem: Participants not always noticed a relationship between main menus and windows.
Situation observed:
- The participant selected a file in the Hierarchy window, he was trying to open it in the editor. He looked in the pop-up menu of the Hierarchy window and didn't find anything for that, then he went to the main menu and in the Source menu read *Open File in Editor...*, then he clicked on it and worked. He thought the file had been open because of having selected it in the Hierarchy window and not because its source code was in the Source window.

15.	Problem observed in:	Short description:
OI	2 novices	L, Some features have the label *selected* and mislead users

Problem: Some features had labels referring to *selected* things. Participants were misled by that several times, such as:
a) When different things were selected in different windows at the same time.
b) When several things were selected at the same time in the same window.
Situation observed:
- In the Source window, two colors were used to show ***"selected"*** places or things:
  1) The gray color for the shadow area
  2) The blue color for selections done with the mouse

  Therefore some participants tried to use *Information About Selected Item*, but they were not sure whether the feature would display information about the shadow area or the selections with the mouse.

16.	Problem observed in:	Short description:
OI	5 novices, 3 experts (all users)	L, Long in dialog windows not always read in detail and lead to inefficient performance

Problem: Many dialog windows had titles and messages below the tittle bar. These messages and titles were scanned and rarely read in detail. For example, when reading *"Enter regular expression"*, participants noticed they had to enter something but they rarely realized that the system expected a regular expression. At first, participants just scanned the messages and entered something. Just after several unsuccessful tries, participants more carefully read and tried to understand what the labels said.

17.	Problem observed in:	Short description:
OI HH	1 novice, 1 expert	L, E, Problems with regular expressions

Problem: Participants experienced different types of problems in features requiring regular expressions:
- As mentioned above, participants not always noticed when features were expecting regular expressions. Sometimes they used other types of patterns that produced different results as the ones they were expecting. Most times they didn't realize why the results were not as expected. Therefore, they simply made another search by looking for a different pattern or decided to use another feature instead.
- Few participants were fully familiar with regular expressions. Even when they noticed that the feature was expecting a regular expression, they not always knew how to build a regular expression to look for what they wanted. Some made some tries with different regular expressions, others decided to look for another pattern and others to use another feature instead.

In conclusion, participants couldn't always fully exploit those features and they wasted energy and time.

18. OI	**Problem observed in:** several users	**Short description:** L, E, Poor feedback while processing

Problem: In some situations when the processing took long, participants didn't know if the system was still processing or something had happened. The tool didn't provide feedback or it had not been noticed by participants. In both cases, participants looked at the clock mouse and sometimes the messages at the bottom. They sometimes got desperate and started clicking in other places.

Situations observed:

- When a variable was selected in the Hierarchy window the information from ***showid*** sometimes took long to appear in the Source window.
- When using the TREE features for routines that call or are called.

19. OI II	**Problem observed in:** several users	**Short description:** L, E, Bad feedback after actions finish

Problem: The feedback given by certain features when actions had finished was not easy to notice. Participants clicked on those features waited for something to happen. The features finished and displayed gave some feedback, however participants didn't notice it and kept waiting for some time.

Situation observed:

- After a database was loaded

20. HiW II	**Problem observed in:** 3 novices, 2 experts	**Short description:** L, E, Poor feedback when there are no results

Problem: Sometimes participants didn't notice when actions had finished with empty results. When participants didn't notice any change in the interface they scrolled the windows to identify any changes in the information. Some executed the action again several times to convince themselves that there had not been any results. Participants developed the concept of not trusting the feedback given by the tool.

21. OI	**Problem observed in:** several users, 1 expert commented that	**Short description:** E, The tool doesn't run in the background by default

Problem: The tool didn't automatically run on the background. Many participants started the tool from an operating system terminal and most typed an ***"&"*** in order to free the terminal. One participant commented that it was probably better to implement that the tool automatically run in the background.

22. OI AA	**Problem observed in:** 1 novice, 3 experts	**Short description:** E, Messages in the terminal distract and mislead users

Problem: Many participants had an operating system terminal opened when they were using the tool. They noticed all debugging messages sent to the terminal. Many times the messages said things like ***not found, etc.*** Participants tended to think that these messages were created by the actions they were doing and therefore there were problems. However, basically none of these messages was concerned with the tool but affected participants' attention.

23. OI AA	**Problem observed in:** 3 novices, 1 expert	**Short description:** **E**, Tcl/Tk error messages confuse the users
Problem: Sometimes Tcl/Tk generated error messages that were displayed on the screen. These messages affected participants' attention. Participants not always understood what the messages meant and confused some participants. It would be better to instruct Tcl/Tk to send this messages to a ***Log*** file and don't present them to the users.		

24. OI	**Problem observed in:** 3 novices, 2 experts	**Short description:** **E**, Tksee doesn't have *undo* facilities
Problem: Participants tried to *undo* actions they did, especially after erroneous actions. However, the tool did not provide such facilities. Common situations were: - Trying to undo delete actions on the Hierarchy window - Trying to undo certain queries		

25. OI	**Problem observed in:** 2 novices	**Short description:** **E**, Many windows are displayed when loading a database
Problem: When loading a database, participants had to deal with many windows. There was a window for selecting the database, another one warning about creating or not a new history, and finally another for giving the name of the history file. This problem was due to the tool design requiring to specify the name of a history file at the moment it was created.		

26. OI	**Problem observed in:** several users	**Short description:** **E**, Users unnecessarily have to specify filename when saving an exploration
Problem: When saving an exploration, the tool required participants to specify a name for the history file. However, such file already had a name. It had been required when the file was created.		

27. OI	**Problem observed in:** 3 novices, 1 expert	**Short description:** **E**, Problems at exit
Problem: There were two different situations when participants exit from the tool: - When using the *Exit* feature participants were always prompted to save the exploration. However, there were times when there was nothing to save. That meant participants wasted time and didn't like having to deal with the dialog window for saving. - Other participants exit by using the *Quit* feature from the X-windows manager. In this case they were not asked to save before exiting even when there were new things that the participant probably wanted to save.		

28. OI BB	**Problem observed in:** 3 novices, 1 expert	**Short description:** **L**, Discovering all the possible pop-up menus is very hard and not always done
Problem: Discovering all the possible pop-up menus in the tool interface was very hard. Participants learning the tool didn't discover certain menus in some windows, particularly those provided by the middle mouse button. Novices required long time before discovering these menus and experts had problems remembering them. Not discovering those menus meant that participants didn't use the features in them and therefore many tasks were completed in very inefficient ways. It's important to mention that in all the sessions none of the participants discovered and used the most powerful features, even experts who had used the tool before achieved some tasks in very efficient ways (nobody efficiently used the *Autogrep* and *Grep* features, for example).		

29. OI	**Problem observed in:** 3 novices, 1 expert	**Short description:** E, Mouse buttons need to remain pressed so that pop-up menus stay
Problem: Participants required time for looking and exploring the different features in the pop-up menus. Many times they accidentally released the mouse button while observing a menu, thus disappearing the menu. Keeping the mouse button pressed for a long time was hard for participants. Also, keeping it pressed and moving the mouse in order to select something was very hard as well.		

30. OI	**Problem observed in:** 1 novice	**Short description:** E, Right numbers in the keyboard don't work
Problem: The numbers at the right in the keyboard didn't work. Some participants tried to use them when looking for a line number.		

31. OI	**Problem observed in:** 3 novices, 1 expert	**Short description:** E, Mismatch between problems reported by feature and the ones written in the source code
Problem: The features that displayed *sms-problems* showed only the problems properly written in the source files. Many participants easily noticed that the problems displayed by the tool were not all the problems in the source code and concluded that the feature provided incomplete results. They also commented that: - Not all the problems were always written in the source files. People sometimes didn't write some. - Not all the problems written followed the standards. For example, some people didn't put *"."* between *"p"* and the problem number. Some people used *"p"* (lowercase) and others *"P"* (uppercase). Therefore, whenever participants wanted to see the problems they ended up scanning the code manually in the Source window and not using the feature. However, they were never absolutely sure if they had found all the problems. The same happened with *activities*.		

II. ABOUT DIALOG WINDOWS OPENED BY INTERFACE FEATURES

32. DW	**Problem observed in:** 1 novice	**Short description:** E, Dialog windows can disappear behind the tool
Problem: Some dialog windows disappeared behind the main window of the tool and led to problems. Some participants clicked on the tool out of the area of an open dialog window and that made the dialog window disappeared behind the main window. Thus, participants had to bring it up again. During the sessions it was never a big problem because part of the dialog window remained visible. However, in general the dialog windows could have completely disappeared causing other problems to the participants. **Location observed:** - This happened with the dialog window from the *Search For Text ...* feature. In particular this window was very frequently moved because it many times hid the matches found. Fortunately, users didn't tend to move the window completely out of the area of the main window of the tool and thus didn't disappear completely.		

33. DW II	**Problem observed in:** 2 novices, 2 experts	**Short description:** E, Sometimes dialog windows take too long to disappear

Problem: Sometimes dialog windows took very long to disappear once the participant had clicked on the button for the action. Some problems with that were:

- Participants noticed that the window didn't disappear and they thought they hadn't clicked on the button, so they decided to click again. Sometimes this generated a Tcl/Tk error message that confused the participants. They weren't sure if the action had been taken or not.

Situation observed:

- The participant was trying to load a history file, in the dialog window of the *Load History File* ... he double-clicked in one of the history files in the list. Because the window stays there, he clicked on the loading button again and got a message saying: *"do you want to save your current exploration"*, he didn't understand and was very confused. What happened was that when he double-clicked on the filename, the tool started opening the file, and when he clicked the button again, the tool interpreted that he wanted to open another history file, therefore it prompted the user for saving the previous one. The user never understood what happened. Once he saw his exploration on the screen he decided to continue like that.

34. DW	**Problem observed in:** 2 novices	**Short description:** L, E, *Path name* field in dialog windows is useless

Problem: Some dialog windows had an input field for entering a *pathname* location. There were several problems with it:

- Such field was the first thing presented from top-bottom in the window. Participants rapidly scanned the fields in that order and paid better attention to the first things they saw. Therefore when some wanted to enter the name of a file, they used the fist field the saw, thus using the *Path name* field instead of the *File name* field. Just a few times they realized that they had entered the name in an incorrect field. Participants never tried to change to another directory location.
- Participants were not used to have an extra field for entering just a path.
- Participants always selected the files by clicking on them not by entering their names.

A much better design would be if users could navigate through the directories by double-clicking on a *".."* displayed with the list of files.

35. DW	**Problem observed in:** 2 novices, 2 experts	**Short description:** L, E, Clock mouse appears in unnecessary situations and misleads users

Problem: When certain dialog windows were open, participants moved the mouse out of the area of the window and it changed to a clock shape even though the tool was not processing anything. Some participants thought that the tool was doing something although in fact the tool was expecting input from them.

Locations observed:

- Global grep dialog window

36. DW FF	**Problem observed in:** 3 novices, 3 experts	**Short description:** E, Input field is not presented highlighted

Problem: Many dialog windows in the tool presented the most recent input in certain fields. However such input was not highlighted, therefore:

- Any time participants wanted to enter something new they had to clean the field manually.
- Some participants didn't notice that the previous input was not highlighted and they simply pasted something into the field. That pasted the string together with the old input. Participants got upset because they had to manually clean and enter the correct input.

37. DW FF	**Problem observed in:** several users	**Short description:** E, Input fields of searching features present just the last string searched
Problem: Participants frequently looked for previous patterns. However, the tool only presented the last pattern searched. Thus, participants had to always enter the input if it was not the most recent one. Sometimes they wanted to use the most recent pattern, but it would definitely have been very helpful if the tool remembered others previous patterns as well.		

III. RELATED WITH THE HISTORY WINDOW

38. HW	**Problem observed in:** 1 expert	**Short description:** L, E, *Update Selected State* feature is useless
Problem: In the History window, the feature *Update Selected State* was not working anymore but it was still there and caused problems. For example: - One participant accidentally deleted everything in the Hierarchy window. He clicked on the corresponding history state to see if he could get back what he had. Because it didn't work, he clicked on *Update Selected State* to see if it solved the problem but nothing happened. This confused the participant who didn't know that the feature was not working anymore.		

39. HW	**Problem observed in:** 1 expert	**Short description:** L, E, Sometimes the *Delete Selected State* feature doesn't work
Problem: In the History window the feature *Delete Selected State* didn't always work. One participant tried to delete a history state and clicked on the feature without success.		

40. HW	**Problem observed in:** 1 novice, 1 expert	**Short description:** L, E, Label in *history states* mislead users
Problem: Sometimes participants went back to a previous history state after a long period of time (long time in terms of remembering in detail that particular state). They read the label of that state and used it to decide if that history state could have the information they wanted. Once they clicked on it, users scanned the information in the displayed Hierarchy window and saw if that history state was in fact helpful. However, the tool had never updated the label of the history state according to deletions or changes done in the Hierarchy window. Therefore, in certain situations participants were misled assuming that the information in that history state really contained what the label said. **Situation observed:** - An interesting case was when a participant deleted everything in the Hierarchy window. Everything he did after that created a new history state, however the previous state remained empty and with its original label. He never understood what happened.		

41. HW	**Problem observed in:** 1 novice, 1 expert	**Short description:** L, E, Things in the History window that look like history states but are not mislead users
Problem: In the History window, some labels look like history states but they don't work like them. None of the participants knew or discovered that they were not real history states. This confused some participants. For example: - In most participants' explorations the first line presented was just a label, it was created when the exploration was started. One participant deleted that line and noticed that there were no consequences at all. Therefore he thought that he had deleted a history state and never understood why nothing happened.		

IV. RELATED WITH THE HIERARCHY WINDOW

42. HiW	**Problem observed in:** 3 experts	**Short description:** E, Pascal built in ***primitives*** are not properly treated by the tool

Problem: The *pascal* language in which the Company systems is written has certain built in ***primitives***. The tool displayed some of these primitives in the Hierarchy window as if they were ordinary routines, however clicking on them displayed error messages in the Source window.

- Participants that didn't remember a ***primitive*** thought it was a real routine and didn't understand the error.
- Participants that remembered the ***primitive*** noticed that the tool was not handling it properly and thus developed the concept that the tool was not very reliable.

Primitive observed:

- Pascal primitive ***BIND***

43. HiW GG	**Problem observed in:** 5 novices, 3 experts (all users)	**Short description:** L, The *Autogrep* feature is extremely hard to discover and learn

Problem: The Aurogrep feature was extremely hard to discover and learned. There were several reasons:

1. First, the feature was inside a middle mouse button menu. Thus making it very difficult to discover.
2. Third, its name was a term unknown by the participants.
3. Second, its operation was very complex.
   a) It had to be used involving two objects.
   b) Those objects had to be on a parent-child relationship.
   c) It had to be started having the child object selected and not the parent.

Some interesting insights were:

- Only one expert participant had seen the feature but he had never tried it.
- One novice participant tried to use it. However, he managed to succeed in points 1, 2, 3a and 3b (above), however he started the feature having the parent object selected instead of the child. Thus, the feature didn't work and he concluded that the feature was not for finding occurrences of one object within another.

Due to this, participants could never exploit the power of that feature, which was one of the most powerful ones. This meant that they performed many steps for solving tasks about finding occurrences of one object within another. They either manually scanned to look for those occurrences or used the *Search for text* feature.

44. HiW GG	**Problem observed in:** 5 novices, 2 experts	**Short description:** L, Grep feature is very hard to discover and learn

Problem: The Grep feature was very hard to discover and learned. The most important reasons were because it was inside of a middle mouse button menu and because participants thought it would work in a different way. Due to this, participants never exploited this feature and instead completed some tasks in very inefficient ways using other features.

45. HiW **HH**	**Problem observed in:** 1 novice	**Short description:** **L,** Grep does not work as users expect
Problem: In the Hierarchy window, some participants tried to use the *Grep* feature in the same way it worked in other systems. For example, some tried to use a pipeline to filter results. However, the feature not always accepted what participants entered. **Situations observed:** - A participant trying to find all assignments in a routine entered: ***<string>*** **| :=**		

46. HiW	**Problem observed in:** 2 novices	**Short description:** **L,** *Find In This List* ... doesn't scroll and therefore participants not always see the results
Problem: In the Hierarchy window, sometimes the *Find In This List* ... feature didn't scroll to the point where the results had been left. Participants didn't see any changes on the interface and were uncertain if there had been any results. Some scrolled in order to look for them.		

47. HiW	**Problem observed in:** 1 novices	**Short description:** **E,** *Find In This List* ... selects results but leaves unchanged the Source window
Problem: In the Hierarchy window, the *Find In This List* ... feature highlighted the results but it didn't refresh the information in the Source window. This is the normal tool behavior for other features in the tool. This caused some problems to participants, for example: - The feature found one match and highlighted it in the Hierarchy window. However, the participant never noticed that information in the Source window remained the same than before executing the search. He thought that information was about the object highlighted in the Hierarchy window. Although in this case there were not serious problems with that this misled the participant.		

48. HiW	**Problem observed in:** 4 novices, 1 expert	**Short description:** **L, E,** Results appended at the end of the sub-list and window is not scrolled
Problem: In the Hierarchy window, the results from certain actions were appended at the end of the sub–list where the action had been started, however the window was not scrolled to the location of the results. Many times there were many objects in the window and the results were not immediately visible to the participants, therefore they had to scroll and look for the results. However, they not always discovered the results, especially when there were many objects in the window. They got upset because of the extra work and time to look for the results.		

49. HiW	**Problem observed in:** 4 novices, 3 experts	**Short description:** **L, E,** Deleting many items takes long and the feedback is not very good
Problem: In the Hierarchy window, deleting many items normally took very long and the feedback was not very fast. Therefore, users required much time and effort to notice that the deletion had indeed worked but it had been slow.		

50. HiW	**Problem observed in:** 1 novice, 1 expert	**Short description:** **E,** Being highlighted is not the same as having the focus and that confuses users
Problem: The difference between being highlighted and having the focus was not always noticed. In the Hierarchy window, one participant pressed the delete key on one item but it was not removed. The item was highlighted but didn't have the focus. Thus, participant pressed the delete key several times and didn't understand why it was not being deleted.		

51. HiW JJ	**Problem observed in:** 3 novices, 1 expert	**Short description:** **L,** *Delete Sublist* feature is misinterpreted
Problem: In the Hierarchy window, the concept of sub-list was very hard to understand. Participants didn't see a sub-list as the group of objects below in a hierarchy level of another object. Instead they saw a sub-list simply as a subset of consecutive objects. Thus, sometimes they used the *Delete sublist* feature when there were no objects below in the hierarchy level. They didn't get any results but never understood why.		

52. HiW JJ	**Problem observed in:** 1 novice	**Short description:** **L,** *Find In This List ...* feature was misinterpreted
Problem: One participant was trying to search for something inside the code of some routines. In the Hierarchy window, he selected a group of routines and used the feature *Find In This List ...* He thought that such feature was going to look inside of the selected objects. However, that functionality was implemented by the Grep feature and not by the feature he used. Like in the previous problem, participants tended to see a list as anything selected consecutively and as not the whole set of objects in the Hierarchy window.		

53. HiW	**Problem observed in:** 2 novices	**Short description:** **L,** The meaning of the relationship between child and parent in the Hierarchy window is difficult to understand
Problem: In the Hierarchy window, participants tended to see a relationship "contains-contained" when an object was child of another. This led them to draw incorrect conclusions. For example: - Some thought that if a child appeared once below the parent was because it was contained just once within the parent. - Some thought that selecting a child would display the parent in the Source window with the child highlighted in it. This was an interesting result that gave some clues about how users interpreted the relationship child-parent in the Hierarchy window. In summary, the relation was many times seen as "contains-contained" instead of "before-after" in time. This problem mainly affected to participants learning the tool. **Some particular situations observed:** - This happened with a routine and the routines that it called. If a *routine that was called* appeared once below the one that *called* it, participants thought it was because it was called only once. - This happened when starting with a routine definition, participants clicked on *File defining me.* Some participants thought that if they selected the file defining the routine, the tool would display the file in the Source window at the point where the routine was defined in it.		

54. HiW	**Problem observed in:** 3 novices, 3 experts	**Short description:** L, E, Message about items at the bottom-left is not seen, remembered or well understood

Problem: Participants were never sure about the meaning of the message about items at the bottom-left:

- Some guessed the correct meaning (the number of items resulting from a query).
- Others thought it was the total number of items in the Hierarchy window.

55. HiW	**Problem observed in:** several users, 1 novice commented that	**Short description:** E, Label *done* displayed before results appear

Problem: In the Hierarchy window, just before certain actions finished a message *"done"* appeared at the bottom. The message was presented before the results of the action. Sometimes the results appeared much after. Some participants noticed the message *done* and thought that the result had been empty, suddenly the results appeared.

Situations observed:

- When the TREE features were going to finish the message *"done"* appeared, then suddenly a bunch of information appeared.

56. HiW	**Problem observed in:** several users, 1 novice commented that	**Short description:** L, E, Messages at the bottom and mouse shaped is not updated at the same time and that misleads users

Problem: In the Hierarchy window, there were several problems when certain queries were finishing:

- The messages at the bottom (e.g. number of items & *done)* were updated but the mouse still looked like a clock for some seconds. Participants were confused a bit when they looked the messages and noticed the mouse still looked like a clock.

57. HiW KK	**Problem observed in:** 3 novices, 2 experts	**Short description:** L, Clicking on something in Hierarchy window doesn't always present what users expect in the Source window

Problem: In the Hierarchy window, the following was observed with certain objects:

- Some participants selected an object **X** and the tool showed object **X** within a shadow area in the Source window. He went to the Source window and scrolled beyond the shadow area. Then, he went back to the Hierarchy window and selected another object **Y**. Later, he selected object **X** again in the Hierarchy window, the Source window was presented in the point where he had scrolled and left it. However, participants tended to think that selecting object **X** in the Hierarchy window would always bring object **X** within the shadow area in the Source window.

Some problems here were:

- Participants got confused because they didn't know why object **X** was not presented
- When they selected object **X** again, many did not remember that they had scrolled

In both cases, participants had to find object **X** in the Source window manually. Some scrolled and others used the *Search for text ...* feature (falling into all the problems associated with both methods).

58. HiW	**Problem observed in:** 3 novices, 1 expert	**Short description:** E, Sometimes the Source window is not refreshed

Problem: In the Hierarchy window, when participants clicked on something there were two strange behaviors:

- Participants clicked on an object but the Source window remained blank. It happened many times after deleting objects in the Hierarchy window. Selecting another history state and going back to the first one made the Source window worked again.
- Participants clicked on an object and nothing happened. Not even the mouse was changed to a clock shape

59. HiW DD	**Problem observed in:** 5 novices, 3 experts (all users)	**Short description:** L, E, Objects can look the same in Hierarchy window and that presents problems to the users

Problem: In the Hierarchy Window:

a) Many times the tool displayed different objects looking the same, therefore:
   - Novices tended to think that it was the same object but repeated several times. Just after selecting each one of them several times, some participants realized they were in fact different.
   - Experts knew they were different but had to spend time finding the one were looking for.

   Some examples of this were with:
   - The *definition & implementation* of a routine. One expert participant had the idea that the *implementation* was always below the definition, which was not always true.
   - Different routines with the same name (for example, the *message* routine).
   - *Variables* and *types.*

   In all cases participants had to select the objects and look at the information in the Source window. This problem suggested that:
   - The messages at the bottom-left in the Source window were not usually seen or understood.
   - The shadow area was not always enough to suggest differences between routine *definition* and *implementation.*

b) Other times, the tool also displayed the same object repeated in the same sub-list at the same hierarchy level several times. This happened when the object was the result of different queries. The tool did not keep track of the existing objects in a particular moment in the hierarchy and displayed repeated information.

60. HiW BB	**Problem observed in:** 3 novices, 1 expert	**Short description:** L, E, Problems with the middle mouse button pop-up menus

Problem: In the Hierarchy Window, participants had many problems with the middle mouse button pop-up menus, for example:

a) Some participants were used to a mouse with two buttons, therefore:
   - They didn't know that the middle button was equivalent to pressing both buttons simultaneously.
   - They were not very good at pressing both buttons simultaneously and manipulate the mouse.

b) Participants had difficulties remembering the features in these menus:
   - What button had to be pressed in other to bring the correct menu
   - What menu contained the desired feature

Participants brought incorrect menus many times. They found features by trial and error, thus resulting in very inefficient performance.

V. RELATED WITH THE SOURCE WINDOW

61. SW	**Problem observed in:** most sessions	**Short description:** E, Launching editor takes long and there is no feedback meanwhile

Problem: Sometimes launching the editor took a long time and there was no feedback meantime. Participants lost the focus on the dialogue By the time it appeared some were almost starting it again.

62. SW	**Problem observed in:** most sessions, 1 novice & 1 expert commented that	**Short description:** **E,** Corrupted information from *sms* is displayed corrupted in the tool

Problem: The tool displays information from *sms* exactly as it is received. If it comes corrupted then it is displayed corrupted as well. Participants didn't know if such information came corrupted from *sms* or it had been corrupted in the tool. For example, information about a *sms* problem was displayed as an ASCII text without ends of lines.

63. SW KK	**Problem observed in:** 5 novices, 3 experts (all users)	**Short description:** **E,** Problems scrolling in the Source window

Problem: In the Source window, scrolling required precise movements of the mouse and that was difficult for participants, especially when the displayed information was very long. The most common problems were:

- Positioning the mouse in the arrows or in the scrolling-cursor. Many times participants accidentally positioned the mouse out of these items and clicked, thus scrolling the window not in the way they wanted.
- Clicking directly on the scroll bar moved the file in a range more than desired.
- Clicking on the arrows scrolled too slowly and it was not very efficient.
- Sometimes the computer's reaction to scrolling was very fast and participants noticed just a big jump rather than several continuous and smooth jumps.

Some of these problems led users to waste time and effort to position the window in the desired location. This problem happened many times when participants tried to locate shadow areas containing routines.

64. SW KK	**Problem observed in:** 5 novices, 2 experts	**Short description:** **E,** Finding the shadow area in the Source window can be difficult

Problem: In the Source window, participants had problems to locate the shadow area or routines, for example:

a) Sometimes they lost the shadow area of a routine and the tool didn't provide any clue about where within the Source window it was (top, center, bottom). Consequently, participants had to scroll and manually look for the routine. Some decided to scan the code and others to use the *Search for text* feature. Some participants suggested that the tool could:
   - Have some indication in the scroll bar so that it was possible to know where in the Source window a shadow area is.
   - Implement a feature for repositioning the Source window at the start of a shadow area.

b) Some participants had difficulties distinguishing the gray tone of the shadow areas from the rest of the window.

65. SW CC	**Problem observed in:** several users	**Short description:** **L, E,** Results from *Information About Selected Item* feature are not obvious to notice

Problem: In the Source window, the feature *Information About Selected Item* displayed its results in the Hierarchy window. Participants not always noticed that and therefore didn't find the results. This particularly happened when the results didn't appear because they had been appended at the bottom of some objects in the Hierarchy window.

66. SW DD	**Problem observed in:** 2 novices, 2 experts	**Short description:** **L, E,** *Information About Selected Item* displays all objects with the same name as the one selected

Problem: In the Source window, highlighting an object and clicking on *Information About Selected Item* displayed all objects with the same name that existed in the database. Participants had to click in all those objects in order to see which was the one they had highlighted in the Source window. In other words, the feature was not context sensitive and meant that participants had to work more when there were several objects with the same name. For example, one participant highlighted a parameter type and clicked on the feature, he got five matches with the same name. One of those was the parameter type and the other four were routines that by coincidence had the same name. In this case, there was a difference in color among the objects (green and red) and that helped him to find which was the parameter type. This problem was more critical with other types of objects like routines, files, etc. where they were not displayed in different colors.

67. SW	**Problem observed in:** 5 novices, 3 experts (all users)	**Short description:** **E,** Lack of robustness of the *Search for text ...* feature

Problem: In the Source window, the *Search for text ...* was one of the most used features. Participants heavily relied on this feature, however it presented many problems:

a) It didn't allow searching backwards.

b) It just worked downwards from the point where the feature was started.

c) Sometimes under the same conditions, it didn't find anything on the first try but found matches on a second try. For example:
   - Novices were very confused because the feature sometimes found matches and others did not.
   - Some expert participants knew that the feature sometimes found objects and others did not. But they got upset for having to scroll to the top of the file or shadow area in order to find all possible matches. For example, some participants went to a certain point in the Source window and selected something so that the search began from there.

d) Sometimes participants were seeing the string they were going to search on the screen but when they used the feature, it said that there were no matches. This made the feature very unreliable and difficult to learn. Participants commented that in real tasks, they needed to find all existing occurrences of something and it was necessary to be very sure about that.

e) Some participants looked for a keyboard shortcut in order to keep searching for the next occurrences. However, the tool didn't provide a shortcut for looking the next occurrence of the same string.

f) The button for finding remained pressed during all the time of some searches and that confused some participants.

g) The feature scanned for regular expressions and therefore it considered as a match anything containing the searched string. This was not always what participants wanted.

h) When the last match was found and the dialog window was closed, the Source window was left scrolled in the place of the last match. However, sometimes that last match was not of interest to the participant, therefore he had to scroll up the window again. In the case of displaying routines, the window was left many times outside of the shadowed area.

i) The following situation observed with several participants:
   - The participant was positioned at beginning of a shadow area in the Source window. He used *Search for text ...* and that sent him to the first match, which happened to be out of the shadow area, however the second match fell inside of the shadow area.

   From these situation:
   - Participants thought: *How is that if I was just right at the beginning of the shadow area, the first match is out of it and the second falls within ?*
   - When looking at the routine, participants were not always conscious that when a match was out of the shadow area, it didn't belong to the routine they were exploring. Some thought it was and made incorrect conclusions.

68.	**Problem observed in:**	**Short description:**
SW	4 novices, 3 experts	**E,** The dialog window of the *Search for Text ...* feature hides matches found

Problem: In the Source window, the dialog window of *Search for Text ...* many times hid the matches found. This was one of the most frequent problems. Almost any time that the feature was used participants had to move the window to some other place.

VI. RELATED WITH THE TOOL BAR AND SHORTCUTS

69.	**Problem observed in:**	**Short description:**
TS	several users	**L,** The features of some buttons in the tool bar are hidden in the middle mouse button menus

Problem: Certain buttons in the Tool bar had their corresponding features in the middle mouse button menus. Some participants looked for those features in a menu (probably to confirm before trying them) but they rarely found them. It took a long time before some novices discovered where they were.

70.	**Problem observed in:**	**Short description:**
TS	4 novices	**L, E,** Problems related with stopping the processing

Problem: Participants experienced different problems related with stopping the processing of queries.

a) Sometimes participants wanted to stop a query but didn't find how to do it. Nobody noticed the Stop button. Many novices ran into problems and wanted to stop a query. Experts tended to do things where the processing was fast, they were never observed wanting to stop a query. An observed situation was:
   - A participant was using the up-down arrow keys to move the focus to the next object. However every time the focus passed over one object, the tool started processing its information. Therefore, the participant had to wait for the refreshment of information and then click the arrow again. He got upset because the tool was all the time *"thinking and thinking and thinking"*. Sometimes he wanted to stop the processing but he didn't find how.

b) Other times, the tool was processing and participants started clicking on some other things. However, clicking on certain objects (in the Hierarchy or History window) interrupted the query. Participants never understood exactly how or why the query stopped.

71.	**Problem observed in:**	**Short description:**
TS EE	several users, 1 novice & 1 expert commented that	**L,** Some icons are hard to learn

Problem: In the Tool bar, there were some problems with the first three buttons. As mentioned before, users know about *files, procedures, functions, variables* and *types*. However:

a) The first button was labeled with an *"f"* which some participants confused as *functions* while it referred to files.
b) The second button referred to *routines*, however participants didn't use the concept of routine in their work and therefore not always noticed it as referring to *procedures* and *functions*.
c) The third icon at the right had a *"v"* that suggested *variable*, however the help flag mentioned *identifiers*. Users didn't use the concept of *identifier* in their work therefore participants didn't know exactly what the feature was going to do.

Only after several unsuccessful tries participants started reading the help flags more carefully.

VII. RELATED WITH THE GLOBAL GREP WINDOW AND FEATURE

72. GG	**Problem observed in:** 3 novices, 1 expert	**Short description:** L, E, Problems with the *Global grep* feature

Problem: Running a *Global grep* query generated a new state in the History window and cleaned the Hierarchy window. Participants noticed that the Hierarchy window was cleaned but they didn't understand why. Also:

- The message *"query started in the background"* was not always seen.
- The message at the bottom-left was not cleared from the previous history state and it misled some participants.
- Participants didn't know how long a *Global grep* was going to take and there was no continuous feedback about its progress. Some participants waited and waited and finally got tired. Some decided to go to some other places and forgot about the results of the query.
- Sometimes participants decided to go to other History states and continue working. After a while they went back to the History state of the *Global grep*. However, when nothing was there yet they didn't know its progress and how longer it would take.
- Some participants didn't notice when there had been no results and query had ended.
- The only feedback indicating that *Global grep* had finished were the results. Therefore, participants working in other history states never knew when it finished.
- In one occasion the results were left within the History state where the query was started and not in the state generated by the *Global grep*. At that moment, the participant was looking at that state and suddenly saw a bunch of information appearing. He never understood what that new information was as well as why it suddenly appeared.
- One participant forgot once to enter something in the first field of the dialog window of the *Global grep*. He started the query and the feature didn't complain neither said what would happen.