Published in: Proceedings of OZCHI'98, IEEE Computer Society, 1998

Usability Requirements in a Tender Process

Søren Lauesen
Copenhagen Business School
Howitzvej 60, DK-2000 Frederiksberg
slauesen@cbs.dk

Abstract

How should the customer specify usability requirements in a tender situation? This is particularly difficult if the product is a standard system with enhancements, since the customer cannot prescribe a specific user interface. Furthermore it must be possible to verify the usability requirements with a reasonable effort, and the requirements must not discourage serious proposers. This paper discusses six different styles of usability specification and shows an example of how to combine them in a complex real-life case to meet these goals. The final requirements are presented.

1. Introduction

A common type of development project is business applications for small and medium sized companies. Such applications are often based on standard systems, modified and enhanced for the customer. The standard system is selected during a tender process where various suppliers propose to deliver their own standard system and develop modifications.

The central part of the tender document is the requirements specification. It specifies what the system must do (the *functional requirements*) and also how well it should do it (the *non-functional requirements*). Usability is an example of non-functional requirements.

How do we handle usability in a tender situation? In some tender processes, the customer has specified the exact user interface as a set of screen pictures and menus. But when the system is based on a standard system, this approach would exclude suppliers which have a standard system with different screens.

The traditional design approach with usability test and iterative development is also impossible since most of the system has been developed already. We might apply it to the system enhancements, but is the supplier willing to offer iterative development at a fixed price? And how are we going to specify the usability of the standard part of the system? In practice, we have to choose between a small set of suppliers, and if we make unrealistic usability requirements, we might end up without any suppliers. How do we balance the various requirements against each other and against what is possible?

In this paper, I suggest a general approach to usability requirements and show how to use it in a complex, real-life tender case.

2. The Shipyard Case

The tender case took place in a medium sized Danish shipyard which specialises in complex ship maintenance, rebuilding, and repair.

The shipyard has 150 employees and up to 350 temporary workers from subcontractors. Orders are known only one to three months in advance. Fast delivery is crucial for this kind of order, and much of the work is not agreed upon until the ship is docked and inspected by shipyard staff and a representative for the shipping company. The situation is thus quite dynamic, and efficient data registration and quotation calculation are essential.

The shipyard decided to replace most of their business applications including accounting, payroll, production planning, order handling, inventory, and sales support. The requirements specification mentioned these goals for the replacement:

System Goals

(a) Part of the system platform had to be replaced. The old system consisted of two different hardware platforms with loosely coupled software systems. One system was proprietary and could not be maintained anymore, so it had to be replaced. The other system was based on Unix/Oracle and parts of it had to survive and become integrated into the new system.
(b) Text-based documents and data base information should be fully integrated.
(c) Data should be up-to-date in all applications.
(d) Systematic marketing should be supported.
(e) Experience data from earlier orders should be available for calculation of new quotations.
(f) Invoicing and post-calculation should be speeded up in order to finish the administrative procedures while the shipping representative performs final inspection of the ship just before launch.

During the tender process, four suppliers submitted a proposal, and one was selected. The original requirements specification with a few corrections was part of the contract and specified what to deliver.

The system was based on the supplier's standard accounting system, which used Oracle and could share data with the surviving system parts. A developer from the supplier worked one year full time on system extensions and data conversion in close co-operation with shipyard staff. The system was delivered on schedule and within budget.

3. System Problems

Together with a graduate student (Susan Willumsen), the author conducted an audit of the requirements specification and the actual system. The aim of the audit was to analyse the relation between the actual system and the original requirements in order to identify good approaches and residual problems.

During the audit, we identified several problems in the final system. We could trace most of them to an insufficient translation of the system goals (a to f above) into requirements [7]. Some problems related to invoicing, which turned out to be a critical task. Why is an ordinary task like invoicing critical in the shipyard? You could not see it from the requirements specification, but invoices are not what you would expect from traditional domains. Some shipyard invoices are more than 100 pages with more than 2000 items.

Each item on the invoice may be annotated with a long explanation. Furthermore, various offices enter the items into the computer system in more or less random order, but on the invoice they must be grouped according to the list of repairs originally agreed with the customer. Each repair comprises several items and they have to be grouped under the original customer heading.

The customer (the ship-owner's travelling inspector) scrutinises the invoice, looking for strange expenses, things where he is charged twice, etc. Proper wording of the invoice is thus important to avoid haggling.

Furthermore, the entire invoice has to be produced and accepted by the customer while the ship is still in dock and the travelling inspector is in town. The invoice is thus made under time pressure.

We noted two important usability problems with invoicing:

(1) The full invoice text was not visible on the screen. The system could show the text for only a single item at a time (in an auxiliary window), which made editing and reviewing difficult. In practice, users had to print out the full text several times for editing.

(2) There were performance problems when editing long invoices. When an invoice was more than 20 pages, the time to scroll from one end of the invoice to the other was unacceptable to users, and editing became cumbersome.

It is interesting that management had heard about these problems, but did not consider them system problems. Management believed that the problems were due to user resistance against IT in general, and they were surprised when we pointed out that the cause actually was system deficiencies.

What did the requirements specification say about usability? Like most requirements, very little. We could find four requirements mentioning usability. The first three were in the section dealing with IT support for sales and marketing. It said:

1. It shall be easy to learn the user interface.
2. The user interfaces in the systems used by marketing shall be consistent.
3. The query functions shall be fast and perceived as more efficient than the present filing systems which are based on manual letter files.

The fourth was under "quality properties" (non-functional requirements). In the final contract it literally read like this (with visible changes from the tender version):

4. ~~No response times shall be so long that employees find them 'stressful'~~. There shall be no waiting time at all in connection with data entry and master file maintenance due to inappropriate programming.

The first sentence was part of the tender document, but in the final contract it was cancelled since the supplier could not take responsibility for employee stress. The underlined part was added in the final contract, since the supplier could only commit to things if they were fairly easy to implement.

It should be obvious that these "usability requirements" are difficult to verify during and after development. They are also insufficient to guard against problems like those we observed with invoicing.

On the other hand, the shipyard case is very much state of the art, and there is no obvious way of improving the requirements. Practitioners definitely need a practical guideline for how to specify usability.

Surprisingly, the literature has very little to say about usability requirements and rarely provides real-life examples. Nielsen [10], Preece [12, chapter 19], and Macaulay [8] give much advise on usability requirements, but in rather abstract settings without real-life examples. Re-

quirements specialists find all non-functional requirements difficult to handle, including usability requirements [3].

4. Usability Factors

Most developers have only a vague understanding of what usability is. Usability has nothing to do with program bugs or system crashes. We assume that the system works as intended by the designer. Usability is about how the user perceives and uses the system.

According to traditional definitions, usability consists of five *usability factors*:

1. Learnability. The system should be easy to learn for both novices and users with experience from similar systems.
2. Efficiency. The system should be efficient in daily use.
3. Recallability. The system should be easy to remember for the casual user.
4. Understandability: The user should understand what the system does.
5. Subjective satisfaction. The user should feel satisfied with the system.

The combination of all the factors is the essence of usability.

Developers often say that it is impossible to make a system that scores high on all factors. This may be true, and one purpose of the usability requirements is to specify the necessary level for each factor.

5. Styles for Usability Requirements

Before proposing usability requirements in the shipyard case, I will show six general styles for usability requirements. These styles are based on my observations from practice, combined with research knowledge from the HCI field.

No style is ideal. Domain-oriented requirements that catch the essence of usability are hard to verify during design. The developer runs a risk when committing to them. More system-oriented requirements are easy to verify during design, but do not guarantee the usability the customer expects. The customer runs a risk.

The styles also specify and measure the usability factors more or less directly.

The best choice in practice is often a combination of the styles, so that some usability requirements use one style, and others use another style. I will show examples in the shipyard case. Here is a summary of the requirement styles. The styles are illustrated with outline requirements marked R1, R2, etc. In a real specification, more precision is usually needed (see larger examples in [5]).

Performance style

R1: Novice users shall be able to perform tasks Q and R in 15 minutes. Experienced users shall be able to perform tasks Q, R, and S in 2 minutes.

In the performance style we specify how fast users can learn various tasks, how fast they can perform after training, etc. We can verify these requirements through usability tests. By means of prototypes, we can make usability tests early during development, thereby tracing the requirements forward into design.

The style catches quite well the essence of usability. However, some of the usability factors, e.g. efficiency in daily use, are difficult to estimate during development. The main problem with the style is that effort and experience is needed to choose the right tasks and iteratively correct the design to meet the specification. A good example of how performance specifications can drive development is given by Gould, Boies, and Lewis [2]. However, the idea cannot readily be used for selecting standard systems.

Defect style

R2: On average, a novice user shall encounter less than 0.2 serious usability defects when performing tasks Q and R. [*A serious usability problem is typically a task failure, i.e. that users cannot complete the task on their own. Thus the requirement roughly says that at least 80% of users shall be able to complete the tasks on their own*]

The defect style resembles the performance style, but instead of measuring task times, it identifies the usability defects in the system and specifies how frequently they may occur. A usability defect is something which causes the user to make mistakes or feel annoyed. The user is asked to think aloud during usability tests, and an observer records the defects. The technique has been extensively described. See Dumas & Redish [1], or Jørgensen [4] for a low-cost approach with high effect on development.

The main advantage of the style is that the list of defects gives excellent feedback to developers, allowing them to correct the design more easily. The disadvantage is that we are less sure to catch the essence of usability. For example, low efficiency in daily use will only be reported as a usability defect if the user complains about it.

Process style

R3: During design, a sequence of 3 prototypes shall be made. Each prototype shall be usability tested and the defects most important to usability shall be corrected.

The process style specifies the development procedure to be used for ensuring usability. The style does not say anything about the result of the development, but we hope that the process will generate a good result. We could specify various processes such as heuristic evaluation, structured dialogue design, etc. The example specifies iterative prototype-based development since it is recognised as an effective process.

We could specify the termination criteria for the design iterations, e.g. continue until no serious usability defects are left, but then we would actually have a defect style, rather than a process style. However, you could specify that more iterations shall be negotiated between customer and supplier after the three iterations. This would still be in process style.

The main advantage of the process style is that it avoids the need for finding target values such as task performance times. The disadvantage is that much is left to developers. Developers often select the wrong tasks and users for usability testing, or they only make minor changes to the prototypes [6]. The style is useful in many cases where developers can commit to a specific process, but not to performance or defect styles.

Subjective style

R4: 80% of users shall find the system easy to learn and efficient for daily use.

With the subjective style, we ask users about their opinion, typically with questionnaires using a Likert scale. Some specialists claim that this catches the essence of usability. Unfortunately, users often express satisfaction with their system in spite of evidence that the system is inconvenient and wastes a lot of user time. (If managers knew about this, they would not be as satisfied as the users.) Nielsen & Levy [11] summarise investigations of this factor.

Satisfaction with the system is heavily influenced by organisational factors outside the reach of system development. Another problem with the subjective style is that it is hard to verify the requirement during development. Many usability experts ask users about their subjective opinion after prototype-based usability tests, but the answers do not correlate well with opinions after system deployment.

Design style

R5: The system shall use the screen pictures shown in App. xx.

The design style prescribes the details of the user interface, essentially turning the usability requirements into functional requirements. They are easy to verify in the end product and easy to trace during development.

Through the design, the requirements engineer has taken full responsibility for the usability. The system designer and programmer can do little to change the usability. If the requirements engineer has done a careful job with task analysis, prototyping, and usability tests, the resulting usability is adequate.

Unfortunately, the prototype style is often used without any kind of usability testing, and the result is as if usability had not been specified at all. Untested prototypes can be used as examples of what the user has in mind, but not as usability requirements.

Guideline style

R6: The system shall follow the MS-Windows style guide. Menus shall have at most three levels.

The guideline style prescribes the general appearance and response on the user interface. You may think of it as a set of broad functional requirements that apply to every window, etc. Guidelines may be official or de facto style guides, or they may be company guides or experience-based rules. It is possible, but cumbersome, to verify and trace these requirements.

Although guidelines usually improve usability, they have little relation to the essence of usability. In other words, you can have a system that users find very hard to use although it follows the guidelines. (Such systems are actually quite common, as demonstrated by the many programs that follow the MS-Windows guidelines, yet are very difficult to use.) As a supplement to other styles, the guideline style is quite useful, particularly to help users switch between applications.

6. Eliciting the requirements

In this section I will show how usability requirements could have been elicited and formulated in a systematic fashion in the shipyard case. Table 4 shows the final usability requirements for the tender.

In general it is a good idea to identify the issues or concerns first, and later translate them into verifiable requirements. Below I have used this method:

1. Identify the key usability issues by looking at critical tasks, user profiles, system goals, previous usability problems, etc.

2. Choose requirements styles to cover the issues.
3. Choose metrics and target values.

The method is not a formal step-by-step procedure. Creativity, experience, and judgement is needed to carry it out.

6.1. Identify key usability issues

Critical Tasks

In a complex system, the number of user tasks is very large, and it is unrealistic to fully cover usability for all of them. So we have to identify the critical tasks.

We identify critical tasks and critical usability factors by analysing system goals, time-consuming tasks, tasks made under stress, and difficult tasks. In the shipyard case, we have a statement of system goals which can give some clues (points a to f above). We need further domain knowledge to identify difficult tasks, etc. Table 1 shows the resulting critical tasks and issues in the shipyard.

Note that invoicing comes up several times since it is made under stress, it is difficult, and it is critical for one of the system goals.

The critical usability factors for invoicing are efficiency and understandability. (Understanding what the system does is particularly important under stress). We have added a non-standard usability factor, *overview*, to denote the need for overview and navigation in long texts. Learnability is not critical for this task, since all invoice staff will receive special training.

Learnability is critical for some other tasks, such as using experience data, since these tasks might still be performed in the old manual way. Using the system will give better results, however, and it is important that users find it easy to do so.

Similar discussions lie behind the other critical tasks and factors.

	Critical tasks:	**Issues:**
System goals:		
Use experience data for quotation	Recording experience data	Efficiency
	Using experience data	Learning
Shorten administration of ship departure	Invoicing	Efficiency
Other goals	(No critical tasks)	
Tasks taking much of the working day:		
	Accounting	Efficiency
Tasks made under stress:		
	Invoicing	Understanding, efficiency
Difficult tasks:		
	Invoicing	Efficiency, overview
	Detail planning	Learning, overview

Table 1. Critical tasks and usability factors

User Profiles

Setting up user profiles will often highlight some usability issues. Table 2 shows user profiles and related usability issues for two roles: marketing and accounting. We can make similar profiles for other user groups. Some issues turn up again, other issues are new, e.g. the cut-over issue and the switching issue.

Other Issues

Some system goals give rise to critical tasks, other system goals give rise to different usability issues. In the shipyard, one of the system goals was "to encourage employees to use computers, e.g. by making the interfaces uniform". This gives rise to this issue:

Issue: Uniform interfaces

The issue is closely related to the issue of easy switching between different systems.

Previous experience from text processing suggests that editing of long texts may take an unacceptable time because the system has a long response time for scrolling

User role: **Marketing**	No. of users: 4
Domain experience	Experts
IT experience	Text processing. Job costing with old system.
Domain attitude	Proud
IT attitude	Reluctant
Learning new system	Must use many systems in the future. Difficult to take time off for courses. Prefer learning gradually on their own.
Issues	**Easy to learn on your own.** **Easy to switch between systems.**

User role: **Accounting**	No. of users: 6
Domain experience	Experts
IT experience	Much, different systems
Domain attitude	Other staff delay things and don't provide correct data
IT attitude	Integrated part of work. Willing to learn
Learning new system	Cut over to new system critical. At most two days
Issues	**Cut over: Short course to learn all basic daily routines**

Table 2. User profiles and associated issues

and searching. It gives rise to this issue:

> **Issue:** Reasonable response time for scrolling and searching invoice text

We have chosen to handle this issue as a usability requirement. Since it is more of a technical requirement, it could also have been handled as a performance requirement.

If we compile all the usability issues into one list, omitting redundancies and overlaps, we end up with the nine issues shown in table 3. The next step is to transform the issues into requirements using an appropriate style.

6.2. Choose Requirement Styles

We do not have to use the same style for all the usability requirements. Some issues are better dealt with in one style, others in another style. Table 3 gives an overview of the possibilities. An X in the table shows that an issue can use a specific style.

Since we assume that the suppliers will suggest solutions based on their standard business application with enhancements, some styles are not useful at all. A prototype (design style), for instance, cannot be used as a requirement since the prototype may not be implementable at a reasonable cost under that standard application.

The table shows that the process style might be used for several issues. This means that the supplier would have to make a number of prototypes, usability test them, and improve them. Such a process only makes sense if the supplier lacks the feature and needs to enhance the system. As an example, the supplier might not have a standard solution for the use of experience data. But in case he has a standard solution, we have to specify the usability requirements in some other way. The table shows that instead of the process style, we could in all cases use the performance or the defect style.

Why not use the performance or defect style in all cases? This might exclude potential suppliers that would not commit to a risky performance specification. The solution is to leave it to the supplier to choose between alternative requirements. We will show details below.

The table shows that guidelines may be useful for easy switching (issue 9). This is no surprise, since guidelines are particularly useful for that. However, we have also shown that guidelines are useful for invoicing (issue 3). Why is that?

The reason is that it is quite difficult to specify usability requirements for invoicing, particularly to ensure a good "overview" of the entire invoice. If we use the performance style, we have to specify tasks that reveal whether the user has a good overview, but such tasks are difficult to specify. On the other hand, experience shows that a good overview of 100 pages is barely possible with a good text processor, but we should accept the text processor approach as a possible solution since we do not know better solutions. As a result, a guideline saying that "it shall be possible to edit an invoice in the same way as a full text" might be acceptable.

Some suppliers may have a better solution than the text processing approach. We could allow for that by leaving it to the supplier to choose between performance, process, or guideline styles.

6.3. Choose Metrics and Target Values

The final step is to write the actual requirements. We have to specify something that can be verified (the metrics) and the target values we require. Table 4 shows the final usability requirements corresponding to the first four issues. These are the more complex ones.

In table 4 we briefly explain *why* each requirement is necessary. This explanation gives a link to the issues we have identified. It also helps the supplier understand the purpose of the requirement. The requirements themselves are numbered in the typical manner used in practice.

The first three requirements are in the performance/defect style, and many suppliers may hesitate to accept them, particularly if it is an added feature. For this

	Style					
Issue	Performance	Defect	Process	Subjective	Design	Guideline
1. Recording experience data, efficiency	X		X			
2. Marketing, learn on your own, particularly using experience data		X	X			
3. Invoicing, efficiency, understanding, overview	X		X			X
4. Invoicing, scroll and search time	(X)					
5. Accounting, cut-over course	X					
6. Accounting, efficiency	X					
7. Detail planning, learning	X		X			
8. Detail planning, efficiency, overview	X		X			
9. Easy to switch beween systems						X

Table 3. The requirement styles suitable for each usability issue are shown with X. In the final requirements, some issues are covered by alternative styles, allowing the supplier to choose.

reason the supplier can choose a process oriented requirement instead, R10.6, which specifies that iterative design is to be used. For R10.3 (ease of invoicing), the supplier may even choose a guideline style, R10.9, which essentially says that if invoicing looks like text processing, the usability is adequate.

As usual, it is difficult to set target values. In some cases we have defined a value, for instance 30 seconds to record experience data. We based this figure on observations of what people do in similar cases when they are not in a hurry. We also believe that it is quite easy to satisfy the demand.

In general, it is risky to insist on such targets in a tender process with standard systems. If the target is too restrictive, suppliers may decide not to make a proposal. In reality, the customer might be satisfied with a system that does not fully meet the target, if the system has other qualities. On the other hand, why set a too pessimistic target if you could get something better.

The solution is to let suppliers specify the target values. For instance we ask them to specify the necessary course time for performing certain jobs. Experience from actual tender processes shows that suppliers vary a lot in the course times they recommend for their product. A recent paper by Maiden & Ncube [9] explains how to collect information from suppliers when buying package software (COTS).

In one case (R10.10), we have given the suppliers a clue to what we expect, but leave the actual specification to them.

When the customer later compares the various proposals, he will compare prices as well as performance figures and other issues. The decision of which supplier to choose is always a complex affair, where apples are compared against oranges. These multi-criteria decisions are not the topic for this paper.

When the customer has selected a supplier, they set up a contract based on the tender requirements. In the contract, the requirements show the supplier's choices and target values.

In case the supplier chooses an iterative design, there is a risk that he cannot provide a satisfactory design in three iterations. In this instance, the customer might want to cancel the contract, but that is difficult since the supplier has not committed to any specific usability level. A way out is needed, and we suggest that the customer pay a fee for the cancellation, while the supplier specifies the fee up front (R10.8). Customer and supplier may also agree to make more iterations, of course.

Table 4. Final usability requirements for the shipyard tender. [Comments are shown in brackets.] A mixture of several styles is needed, and the supplier may choose between alternative requirements.

Section 10. Usability Requirements

Some of the usability requirements below cover the same issue, but in different ways. The efficiency of invoicing, for instance, is covered by R10.3, R10.6, and R10.9. The vendor may choose between the alternatives as shown below.

It must be easy to record experience data. Otherwise it will not be done. This will most conveniently be done while entering or editing job data:

R10.1 When a job has been selected for data entry, it shall be possible for an experienced user to attach experience data within 30 seconds, including lookup of experience keywords. (See task description in App. xx.1.) The vendor may choose R10.6 through R10.8 instead of R10.1. Chosen requirement: _____.

Marketing has little time for courses and prefer to learn on their own. The vendor should specify the minimum course time that will allow marketing staff to use the system through their own experiments:

R10.2 After a ___ hour course, marketing staff shall be able to perform 90% of the tasks in App. xx.2 on their own. [This essentially limits the number of serious usability defects. We don't care about task time. Users are allowed to take the time they think necessary. App. xx.2 has about 20 tasks, two of them dealing with the use of experience data.] The vendor may choose R10.6 through R10.8 instead of R10.2. Chosen requirement: _____

Invoicing is critical. Invoice staff need an efficient solution, easy to understand and with a good overview of the entire invoice:

R10.3 After the cut-over course, it shall be possible for an invoice user to edit the invoice printed in App. xx.3 (as shown by the edit markings) within __ minutes. This includes time to verify the corrections without printing the invoice. [*App. xx.3 shows an invoice about 50 pages long with 20 corrections.*] The vendor may choose R10.6 through R10.8 or R10.9 instead of R10.2. Chosen requirement: _____

R10.4 After the cut-over course, the invoice user shall be able to explain the effect of editing the invoice text, the cost fields, and the discount fields, for instance what changes it causes in the data base and on the accounts. The user shall also be able to explain what effect a system break down has on a partially completed invoice.

The cut-over to the new system must be accomplished in a few days. This means that accounting and invoice staff must be able to learn the new system at a short course and soon after use the new system:

R10.5 After a cut-over course of ___ days, accounting and invoice staff shall be able to perform the daily tasks listed in App. xx.4. [*App. xx.4 contains about 5 tasks for each of the functional areas mentioned in the original requirements specification.*]

The vendor may choose an iterative design approach instead of some of the above requirements:

R10.6 During design of non-standard features, a sequence of 3 prototypes shall be made. Each prototype shall be usability tested and the defects most important to usability shall be corrected. Usability testing shall include the tasks mentioned in the appropriate requirement above.

R10.7 After the last usability test, the customer and the vendor negotiate whether to make additional prototypes at an additional fee, whether to implement the last prototype, or whether to cancel the contract due to insufficient usability.

R10.8 In case the contract is cancelled according to R10.7, the customer shall pay $_____ as compensation.

Instead of fulfilling R10.3, the vendor may provide an invoice system resembling a text processor:

R10.9 During invoicing, the user shall be able to see and edit the entire invoice as in word processing (WYSIWYG style), including cut and paste, undo, scrolling, and searching.

Scrolling and searching in long invoices are frequent operations. The customer expects a response time of less than 5 seconds.

R10.10 Scrolling one page up or down in a 200 page invoice with 4000 items shall take at most _____ seconds. Searching for a specific word or item number shall take at most _____ seconds.

7. Conclusion

This case study investigated a tender situation where a standard system was the major component. The study showed that usability specifications in this case could be handled by a mixture of four requirements styles. Two other styles were useful in other situations, but not here.

In a complex case like the one studied, it seemed unrealistic to specify usability for all user tasks. The effort of verifying all such specifications would be excessive. The solution is to select only the more critical tasks. There is also a risk of specifying too strict requirements, for instance a very short time to learn the system. The result could be that no supplier offers a proposal. The solution is to ask the suppliers to specify the learning times and include the values as criteria in the decision process.

In practice, it seems necessary to give the suppliers alternative requirements. If a supplier has a standard feature that covers a certain functional requirement, he may accept one style of usability requirement, but if he covers the functionality through an added feature, he may accept another style.

Acknowledgements

The author thanks Jens-Peder Vium for his willingness to reveal and discuss the original requirements, and Houman Younessi for many discussions while we struggled with the concept of requirement styles.

References

1. Dumas, J.S. & Redish, J.C.: A practical guide to usability testing. Ablex 1993.
2. Gould, J.D., Boies, S.J., & Lewis, C.: Making usable, useful, productivity-enhancing computer applications. Comm. ACM, Jan. 1991, Vol. 34, No. 1, pp. 75-85.
3. Hochmüller, E.: Requirements classification as a first step to grasp quality requirements. In: Dubois & al.: Proceedings of the Third International Workshop on Requirements Engineering, REFSQ'97, 1997, Barcelona.
4. Jørgensen, A.H.: Thinking-aloud in user interface design: a method promoting cognitive ergonomics. Ergonomics, 1990, Vol. 33, No.4, pp. 501-507.
5. Lauesen, S. & Younessi, H.: Six styles for usability requirements. In: Dubois et al. (eds): Proceedings of REFSQ'98, Presses Universitaires de Namur, 1988.
6. Lauesen, S.: Usability engineering in industrial practice. In Howard et al. (eds.): Human-Computer Interaction, Interact'97, Chapman & Hall, 1997, pp. 15-22.
7. Lauesen, S. & Vium, J-P.: Lessons learned from assessing a success. Fifth European Conference on Software Quality, Dublin, September 1996, pp. 335-344.
8. Macaulay, L.: Requirements engineering. Springer, 1996.
9. Maiden, N.A. & Ncube, C.: Acquiring COTS software selection requirements. IEEE Software, March/April 1998, pp. 46-56. [COTS means Commercial, Off The Shelf software, i.e. standard applications.]
10. Nielsen, J.: Usability engineering. Academic Press, 1993.
11. Nielsen, J. & Levy, J.: Measuring usability, Preference vs. performance. Communications of the ACM, 37(4), 1994, pp.66-75.
12. Preece, J.: Human-computer interaction, Addison Wesley, 1994.