Introducing Measurable Quality Requirements: A Case Study

Stephan Jacobs
Ericsson Eurolab Deutschland
Stephan.Jacobs@eed.ericsson.se

Abstract

In this paper a case study on improving requirements engineering is presented. Improving requirements engineering was initiated in a department at Ericsson Eurolab after an analysis had shown that many of the problems in software development had their root cause in insufficient understanding of the customer and in unclear requirements. A method - in respect to Tom Gilb who supported us in this field - called Gilb Style was introduced. This method focuses on

- *quality (or non-functional) requirements*
- *quantification*
- *strict separation between design and requirements*
- *constraints and assumptions*

After a year of experience and several projects using this method, the findings are presented. The biggest benefit in using this method is a change of culture towards requirements. This change is not limited to requirements specifications for software but includes e.g. requirements on internal service functions. The common understanding of requirements has drastically increased. Several positive side effects include more effective inspections, introduction of weekly reviews, simpler definition of test cases. The biggest problems were communication problems to (internal) customers, who did not participate in the introduction of the method.

1. Introduction

In this paper a case study on improving requirements engineering is given. Two issues shall be communicated.

- Introducing requirements engineering is a change of behavior and culture and not only an introduction of a technology.
- The understanding of quality (non-functional) requirements is essential. This understanding can be reached by making quality requirements measurable.

Although a practical foundation of requirements engineering has been asked for on several occasions (e.g. [4]) there are only a few case and field studies in the field. Some of them are aiming for an overview in the field of requirements engineering [5, 13], others look for specific aspects like traceability [9, 15], or the use of scenarios [16]. The problem of learning and the education of requirements engineering has hardly been addressed. Thus, there is still a big gap between research and practice in this field [1].

The focus in requirements engineering usually lies on functional requirements. Nevertheless, quality or non-functional requirements will be addressed in this paper as well. Research in this field has concentrated on the modeling and representation of quality requirements (e.g. [14]), and on negotiation of conflicts between different requirements (e.g. [3]). The relations between functional and quality requirements has been studied only rarely although this is necessary in understanding requirements, especially in the industry [12]. There is hardly any publication which addresses the measurability of quality requirements.

The paper is structured in the following way. Chapter 2 and 3 describe where and how the case study was performed. Chapter 4 summarizes the findings of the root cause analysis and the conclusions based on these findings. Chapter 5 gives an overview of the method we now use for specifying requirements, chapter 6 outlines our findings.

2. Background

Ericsson is one of the big companies in the telecommunication industry. Ericsson's business lies in developing and selling all kind of telecommunication products from small mobile handsets to complex switches.

The department, in which this case study was performed, is responsible for developing software, starting from requirements engineering until maintenance. Since 1996 the department uses the Capability Maturity Model (CMM) [11] as a guideline for it's improvements. In 1996, effort was put into configuration management. In 1997, project planning and tracking were improved. In both years, the same improvement strategy was used successfully, namely to concentrate the effort to improve on only a few issues. For 1998 we decided to focus our efforts on requirements engineering. The following case study deals with the improvements achieved in this field.

3. The Case Study

The case study is based on five (sub)projects which until now have used the new way of requirements engineering. Three of them started directly after the method was introduced. The other two started half a year later and are based on the experiences we got in the first projects. The persons who contributed to the study were designers, line management, project management, and persons responsible for the product. The study is based on the following input:

- A systematic analysis of weaknesses in our software process using the *seven management tools*. This analysis has been performed twice, at the very beginning of the study, which actually led to the improvement program. The second time one year later, after several projects had made experiences with the new way of requirements engineering.
- Interviews with involved parties. The used questions are shown in figure 1. However, the questions were not intended to be used as a questionnaire but as a guideline. In fact, the interviews took usually up to half an hour.
- Analysis and comparison of several old requirements specifications and five requirements specification using the new style.
- Informal communication and observation.

- Is the Gilb-Style used in your current project? What was the reason to use/ not use the style?
- What are the advantages/disadvantages of the style compared with the "old" way to specify requirements?
- Why has the introduction of the Gilb-Style been successful?
- What are the next issues to address in requirements engineering?

Figure 1: Interview Questions

Our primary intention was not, however, to perform a case study but to improve requirements engineering. Thus, the case study at times has the flavor of a personal report.

Measurements are constantly performed to get insight into quality, project duration and effort, or stability of requirements. The measurements show a clear improvement in all three fields since we introduced the new way of requirements engineering. But as several other factors have changed as well (problem domain, project partners, level of experience, ..), it is not possible to quantify the effect of the changes caused by the new way of requirements engineering.

4. Root Cause Analysis

In the fall 1997 an analysis of the software development process was performed. The analysis was done by experienced software designers, project management and line management. The goal was to define actions to increase the quality in terms of fault density and to decrease the lead time, i.e. the time from the first idea of a product until delivery to the customer.

The root cause analysis identified *"Missing Understanding of Customer Needs"* as the main obstacle for decreasing fault density and lead time. Related problems were *"Too many, too big documents do not give a clear picture of requirements and use cases"*, *"Requirements are often unclear ... especially performance requirements"* and *"Everybody has a different understanding of what should be developed"*. All findings were aggregated under the heading *"No common understanding of 'What to do'"*.

These problems were taken as the basis for the second part of the analysis, which aimed at defining counter measures. Proposals to improve common understanding were defined. Most of the proposals focused on *testing* the quality of requirements rather than on *producing* good requirements. Typical examples were *"Insist on clear requirements"*, *"Don't work on something that is not clearly defined"*, or *"Force people to write clear requirements"*. There was no proposal on *how* to get clear requirements, nor a clear understanding what a clear requirement is.

It became obvious, that we did not just have a technical but a problem of culture and behavior

- Requirements specifications did not contain the information needed for software development
- Requirements specifications contained inconsistencies
- Inspections on requirements specifications were not taken sincerely and hardly detected any faults.
- Tests were not based on the requirements specifications
- Changes in the project scope were not incorporated into the requirements specification.
- Requirements were not taken as commitments [11].
- **Requirements were not taken seriously.**

The whole way in which we were defining and dealing with requirements was made doubtful. Consequently, a new tool, a changed process, or another policy would not have solved the problem. A massive effort was required to effect a change of culture and behavior.

As a first step we invited a consultant, Tom Gilb, to discuss our problem. Each software team in the department had the chance to present a requirements specification and discuss it. In the end, there was a common understanding in the management and in the software design teams, that a course by the consultant would be the right step to go for. There was an agreement, that all people involved in software design had to participate in this course. Requirements engineering should not be a subject for product management only. A date was fixed, all projects were put on ice for a week, and all software developers participated in a one-week-training on requirements engineering.

5. The Method

In this chapter the main concepts of the introduced method - the Gilb-Style - are explained. The chapter first presents an analysis of the weak points in our old requirements specifications. Then the concepts of the new style are presented.

5.1 The old style

When analyzing our own requirements specifications the following shortcomings became obvious:

- Requirements specifications were long and monolithic based on narrative text. The requirements were unstructured. There was no conceptual difference between for example functional and non-functional requirements. There was no hierarchical structure, all requirements were just put into a long list.
- Several of the requirements were no requirements but design proposals, e.g. "there has to be Pull-Down-Menu for file handling", "the database has to be ...", or "use a password mechanism to secure ...".
- Quality requirements were not precisely defined. Typical examples were "increase the usability" or "the system has to become more reliable".
- Costs in terms of elapsed time, man-hours, or money were not stated as a requirement at all.
- Requirements were rarely "sourced", i.e. it was not possible to see if a requirement was based on an explicit customer wish (e.g. stated in an interview), if it was stated by the product manager, by the designer or anybody else or if it was based on a rule or a law.

The most dramatic effect of these shortcomings was, that the requirements specifications were not really used. The requirements on the product were communicated informally rather than through the documentation. As the informal communication was not precise enough, the misunderstandings detected in the analysis (cf. chapter 2) were introduced.

Moreover, the following side effects were noticed:

- Requirements specifications were not maintainable. They included e.g. inconsistencies which were not detected. Introducing new requirements and analyzing their relations to other requirements was not possible.
- Only parts of the requirements were tested. Especially, the non-functional requirements (increase usability) were hardly checked.
- As their were no rules, how to write a requirements specification, inspections could not check these rules. Formal inspections of the documents were "degraded" to walkthroughs or reviews.
- It was not possible to trace requirements, not to their source nor to their target. Thus it was not possible, to contact persons who had introduced a requirement. It was not possible to check which parts in the design could be dropped because of a removed requirement.

5.2 The "Gilb Style"

The goal of the Gilb Style is to overcome the above described shortcomings. The style goes for

- a clear distinction of different type of information,
- a breakdown of requirements into sub-requirements,
- measurable quality requirements (non-functional requirements)
- tags to uniquely identify all requirements
- source information for all requirements

None of these ideas is new in requirements engineering, and nearly all textbooks in the field recommend this in one or the other way. Nevertheless, most of these requirements (on requirements) are not used - not only in our department! Obviously, it is not enough to request these ideas but it is necessary to say, *how* these ideas can be fulfilled.

Structuring Information: To implement these concepts a certain level of structure or formalism is needed. We achieved this by taking a subset of *Planguage*, a language to communicate about "engineering and management work" [8]. Using Planguage, the following three main categories of information are defined:

- Assumption
- Requirement
- Glossary

Assumptions can be characterized as facts which are valid, or assumed to become valid, in some point of time. It is the nature of assumptions, that they can not be proven to be correct. Violating an assumption now or later can risk the requirements. The whole requirements specification is based on the understanding that the assumptions are valid. For example, an assumption could state that a product is based on the GSM standard.

The glossary is used to specify terms used in the specification in more detail. This is, to assure common understanding and to avoid hidden assumptions.

Requirements are divided into

- Functional Requirements
- Quality Requirements
- Constraints
- Cost Requirements

Functional requirements describe the *what* of a system. In the Gilb-Style the functional requirements focus on the mission of a system, that is on the high level requirements. The detailed requirements are usually not required as they specify how certain quality requirements shall be reached.

The focus of the Gilb-Style lies in quality requirements. Quality requirements describe *how well* the defined functionality has to work. Quality requirements specify the soft and analog element of functional requirements. To be precise, quality requirements have to be quantified. Typically, they are divided into several sub-requirements to cover different aspects of global terms. For example, the quality requirement *usability* could be divided into the requirements *easy to use, easy to learn, fast to use*, etc. These requirements can then be divided further. Usually, quality requirements make the biggest part of a requirements specification in Gilb-Style. Moreover, it is most tricky to specify them.

Constraints form a restriction that must be taken into consideration when designing the complete product. Constraints restrict the solution space for the requirements. A constraint could be for example a limitation that a product *has to run on a PC*.

Cost requirements are a special case of quality requirements. They are mentioned explicitly since often costs are not regarded as a part of the product specification. However, a system is never accepted for any price. The term cost covers not only money, but all kind of resources like time, space, people, satisfaction, reputation, etc. As these resources are always limited, they are a crucial part of the requirements specification. Only if the costs are specified, it is possible to balance cost-benefit decisions.

Design ideas are no requirements. For example a statement like "a pull down menu has to be used" is no requirement but a design idea, a proposal for a solution. If these design ideas are not negotiable, they can be expressed as constraints.

Quantification: To be precise, it is necessary to make requirements measurable. Functional requirements are binary, i.e. they are either implemented or not implemented. It is more difficult - and essential - to make quality requirements measurable. From the concepts offered in Planguage we use *Gist, Scale, Meter, Past, Record, Must, Plan,* and *Wish.* These concepts are made visible in our requirements specifications by using keywords in bold letters (cf. example in the annex).

Gist is a rough summary of the requirement; an explanation of what shall be measured afterwards. The Gist is useful to get an overview of all requirements without going into the details. The Gist for the quality requirement *Maintainability* could be "The system has to be maintainable to allow future expansion of the system". The Gist helps to write down the requirement *before* becoming precise.

Scale defines the unit in which the requirements has to be measured. Scale defines a set of measures to express notions of qualities and costs. The scale for the efficiency requirement could be "Number of man-hours required to add a new module to the system" or "Number of modules which have to be changed when adding a new module to the system".

Meter defines the way how the measurement will be performed. It can, for example, specify a test. The meter for the above examples could be "Based on our historical data estimate the number of man-hours which are required to add a module that sends the output of the system to the printer rather than to the monitor". The second scale could be measured in the following way "Take the average number of modules each module calls within the system".

Past and *Record* are benchmarks. Past is a value which is typical for (own) products developed in the past. It is important to know the own level as the customer will know it and compare new products against this already achieved level. Typically, a customer expects that at least this level is reached. On the other hand Record represents the best value which has been achieved for this meter. To target a product on the market it is crucial to know both values.

In contrast to Past and Record which are historical values, *Must, Plan* and *Wish* envisage the future. Must, Plan and Wish characterize the system that is to be built. As the names indicate Must represents the minimal goal level. Not reaching the must level for a single requirement can be a show-stopper for the whole product. Failure to reach the must level indicates a failure level for the system in general. Plan is the envisaged level for a requirement. However, not reaching the plan level indicates a failure only for this requirement, not for the whole system. Finally, the wish level represents the level which is aimed for under optimal circumstances. The Wish value is usually not committed to due to resource limitations.

Sourcing Requirements: To trace the requirements to their source, it is essential to know, where the requirements are coming from. Sources can be a person's name (e.g. Steve Miller) or a role (product management). Sources can point to comparisons in newspapers or other kind of literature (e.g. when defining the record, or the planned level). Graphically, the source is represented by an arrow pointing backwards. For example "**Must:** Time < 1 sec ← Steve Miller" means that the corresponding must-level for the requirement goes back to Steve Miller.

Tagging Requirements: Each requirement including constraints and costs and each assumption has to be uniquely identifiable. To assure this, unique tags have to be assigned to each requirement.

Be precise, what is not precise: If it is not possible to explicitly specify a term, it has to be put into brackets like <>. This incomplete or inprecise specification will typically be clarified later. In this sense <> can be used as To-Be-Done markers.

6. Findings

In the following chapter, the findings we made after introducing the Gilb-Style are summarized.

6.1 Improved Understanding of Requirements

Improving requirements engineering was initiated by the analysis described above (cf. chapter 4). The main findings were missing understanding of the customer, unclear requirements, and differing understanding of requirements.

In this respect, the whole initiative was successful. Unclear requirements have not been an issue in the projects using the Gilb-Style. In an analysis, similar to the one described above (performed in the fall 1998) problems with requirements were not longer mentioned as a problem.

6.2 Change of Behavior

As indicated in the analysis, bad requirements specifications were not just a technical problem but a behavioral one. There was no culture, to be precise, to be committed to requirements. This was true for several type of documents but it became most obvious in the requirements specifications.

There are several indications, that the culture has changed in this respect.

- The new style is introduced into new projects not by management, process engineers, or quality personnel, but by designers. They see *their* benefit in improved communication and understanding – both internal and external. Moreover, it is essential for the designers to have a clear commitment with the customer, which can only be achieved with precise requirements. Designers see, that their position is strengthened if they are precise.
- If new colleagues show up, the designers advertise the new Gilb-Style.
- The usage of the Gilb-Style is not limited to requirements on software but used in other situations as well. For example, the goal for a workshop or demands on service functions have been described in a more precise way using the same style.

6.3 Communication Problems with Customers

The biggest problem we have experienced were communication problems with our (internal) customers and with other related projects. The initiative to improve requirements engineering was only performed in our department. Thus, our communication partners, expecting specifications in the old style, were "disturbed" by the new way. Although nobody argued, that the new style was bad the „project's rhythm" was effected.

This miscommunication made two issues obvious. First, even a good approach can be problematic if not all parties are involved and trained. This is especially true for behavioral changes. Second, requirements specifications are not only the basis for the design but serve several other purposes, for example as the commitment to the customer or as a foundation for marketing information.

6.4 Focus on Quality Requirements

All requirements specifications written in the new style focused on quality requirements. From our experience, it is crucial, to have a common understanding of the quality requirements rather than to go into the details of the functional requirements. To focus on detailed functional requirements has often a flavor of a solution and not of a requirement. Detailed functional requirements often have no source but are explained with statements like "to increase the usability". This, however, points on quality requirements.

6.5 Cost Requirements are not used

So far, we have not succeeded in specifying cost requirements. The reason for this lies in the organization of the projects. The responsibility and authority for products and projects is distributed. That means, costs are often not variable, but fixed from the beginning. Elapsed time is not negotiable, thus it is not a cost requirement but a constraint.

This leads to the fact, that balancing costs and requirements is not done. As a consequence, costs are not specified as requirements stating different kind of must-, plan-, or wish-level. Costs are specified as constraints.

6.6 Inspection Quality and Effectiveness Increased

Based on improved requirements specifications, our inspections have become more effective and more efficient. We find more faults in a shorter time.

Formal inspections, in contrast to reviews and walkthroughs are based on formal rules that the document has to fulfill [6]. Unclear requirements on requirements specifications led to an absence of formal rules for inspections. Consequently, we could hardly find any faults in the inspections on our old specifications.

This has been changed with the new style. The number of faults found per page has drastically increased.

At first sight, the increased number of faults could be interpreted as a disadvantage when using the new style. On the other hand, we regard this as one of the biggest advantages! Preciseness is a prerequisite for really deciding if something is correct or not. An inprecise demand like "high usability" is always correct and thus of no value. Therefore, finding no faults during inspections is no good sign, but an indication that something is going wrong.

Some projects have changed their way of working and perform weekly inspections on the requirements specifications. As the formal rules for the Gilb-Style are quite clear, so are the inspection rules. Assigning rules to designers, i.e. not everybody has to check every rule, leads to more efficient inspections. This allows us, to perform inspections more often (e.g. weekly) and by that to get feedback for the specifications earlier. The big inspection effort is not done on the final document at the end of requirements engineering but continuously.

6.7 Test Cases

Inexact requirements can hardly be tested. Consequently, quality requirements like "increase usability" were not tested. Defining quantified requirements enables the definition of test cases based on the measurements. Moreover, by specifying the meter, a test case is already defined in the requirements engineering phase. To make requirements measurable means to make them testable.

7. Summary

In this paper, a case study has been described on introducing requirements engineering into a department responsible for software development. The report describes the analysis which led to the decision to improve requirements engineering. It describes the Gilb-Style and the way, in which it was introduced and finally it summarizes our findings after several projects used this method.

The following lessons have been learned:

- **Introducing requirements engineering may require a change of behavior.** Based on the maturity of the organization, introducing requirements engineering is a change of culture and behavior and not just a change of process or technology. It is much more difficult to implement a change of culture than a change of technology.
- **Change of behavior requires commitment from all involved parties.** Behavioral changes can not be enforced top-down. An agreement has to be reached by all parties (management, project, designers, ..) on the weak points and on how to tackle these weaknesses.
- **Change of behavior requires massive training.** All involved parties have to participate in an adequate training. Training only a few persons and hoping on the multiplier-effect is likely to fail. This approach might be sufficient for introducing a new technique or process but not for a change of behavior.
- **Importance of quality requirements.** One characteristic of the Gilb-Style is the focus on quality rather than on functional requirements. The experience we have made so far supports this idea. Having a common understanding of the quality requirements is essential for a common understanding of the product. Going into the details of the functional requirements often results in working on a solution rather than on requirements.

What are the next steps for our department? In trying to answer this question, three issues were brought up, namely the specification of cost requirements, the need for making decisions on cost and benefit in the requirements specifications, and the handling of requirements in projects following incremental development.

As mentioned in chapter 6.5 we have not used the cost requirement so far due to the organization of our projects. There is a common understanding that cost requirements will become an issue, if the responsibility for projects and products are changed.

If cost requirements are included, we have to balance cost-benefit (or cost-quality), at least. One tool to manage this could be with the use of simple tables. The idea of using tables to support this kind of decisions is for example suggested by Gilb (Impact-Estimation-Tables), by the method Quality Function Deployment (House of Quality) and is already implemented in some requirement management tools (Doors, RequisitePro, ..). We have to investigate if and how these techniques can be used to balance cost-quality trade-offs in our projects.

We are currently changing the life-cycle of our software development projects. One large waterfall-like development phase is replaced by several increments. These increments always include a phase of re-defining requirements. The impact of this change on the way we specify requirements is not well understood. We have to have a commitment with the customer for the whole project **and** for the next increment. Both commitments have to be based on requirements. It is not clear how to specify and relate these long term requirements with the requirements for each increment. One solution could be to relate different kind of quality levels (similar to the must, plan and wish level) to each increment. In this respect, we see more potential in the Gilb-Style.

8. Literature

[1] D.M. Berry, B. Lawrence: "Requirements Engineering", IEEE Software, vol.15, no.2, pp.26-29, 1998

[2] B. Boehm, J.R. Brown, H. Kaspar, M. Lipow, G.J. MacLeod and M.J.Merrit: "Characteristics of Software Quality", North-Holland Publishing Co., 1978

[3] B.Boehm and H. Ih: "Identifying Quality-Requirement Conflicts", IEEE Software, vol.13, no.2, pp, 25-35, 1996

[4] Janis Bubenko: "Challenges in Requirements Engineering", IEEE International Symposium on Requirements Engineering, RE'95, York 1995

[5] K.E. Emam, N. Madhavji: "A Field Study of Requirements Engineering Practices in Information Systems Development", IEEE International Symposium on Requirements Engineering, York, UK, 1995

[6] M.E. Fagan: "Design and Code Inspections to Reduce Errors in Program Development", IBM Systems Journal, vol.15, no.3, pp.182-211, 1976

[7] T. Gilb: "Principles of Software Engineering Management", Addison-Wesley, 1988

[8] T. Gilb: "Planguage - A Handbook for Advanced practical Management", manuscript, access via http://www.stsc.hill.af.mil/SWTesting/gilb.html, 1996

[9] O.Gotel and A. Finkelstein: "Extended Requirements Traceability: Results of an Industrial Case Study", International Symposium on Requirements Engineering, Annapolis,1997, pp. 169-179

[10] J.R.Hauser, D. Clausing: "The House of Quality", Harvard Business Review, May June, 1988, pp.63-73

[11] W.S. Humphrey: "Managing the Software Process", Addison-Wesley, 1990

[12] Dieter Landes: "Requirements Engineering for Quality Requirements – Industrial Problem Statement", Fourth International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ'98, Pisa, Italy, 1998

[13] M. Lubars, C. Potts, C. Richter: "A Review of the State of the Practices in Requirements Modeling", IEEE International Symposium on Requirements Engineering, pp.2-14, San Diego, CA, 1993

[14] J. Mylopoulos, L. Chung, B.Nixon: "Representing and Using Non-Functional Requirements: A Process Oriented Approach", ACM Transaction on Software Engineering, vol.18, no.6, pp.483-497, 1992

[15] B. Ramesh, T. Powers, C. Stubbs, M. Edwards: "Implementing Requirements Traceability: A Case Study", IEEE International Symposium on Requirements Engineering, pp. 89-97, York, UK, 1995

[16] K. Weidenhaupt, K. Pohl, M. Jarke, P. Haumer: "Scenarios in System Development: Current Practice", IEEE Software, vol.15,no.2, pp.34-45, 1998

Annex

In the following, an example expressed in the Gilb-Style is given. The product called **enote** (electronic note) is a product something similar to today's organizers, having a traditional notebook as metaphor. Of course, the example should not be taken as a complete requirements specification. There is an example for mission, assumption, constraint, functional and quality requirement.

Mission:
Enote will be the modern equivalent of the traditional notebook enhanced by functions which can only be implemented on electronic devices, like alarm-clock functions for calendar, search functions etc.

Assumptions:
The *project* is not responsible for the development of the <input device> and the <output device>.
The product is aimed only for the european/american market, i.e. it has to support only latin letters. ← Marketing

Constraints:
The target hardware will be a X234 offering 2MFlops/sec and 2 MByte ROM/RAM. ← Senior Management

Functional Requirements:
Defintion: Enote has to have a calendar.
Motivation: Each notebook has a calendar! To have no calendar is a show stopper for a product like Enote. ← Marketing
Definition: Enote has to support a To-Be-Done-List
Motivation: This is weak point in classical notebooks. Having a To-Be-Done-List with attributes like category or priority is an enhancement in comparison to paper based notebooks.

Quality Requirements:
Easy to learn:
Gist: Enote has to be easy to learn. The <target group> is not willing to read *short manuals*. The functionality has to be intuitively understandable, based on the notebook metaphor.
Scale: Time which is required until <basic functions> are understood

Meter: Time which a manager from our company needs, until he enters the first address having all <introduction material>.

Must: 1 min ← No manager is willing to spent more than one minute with the manual, Marketing

Plan: 30 Seconds ← Steve Miller, intuition

Wish: 0 Seconds ← The Metaphor is perfectly implemented (cf. Record)

Record: 0 Seconds ← Notebook

Past: 3 Min ← Laptop

Easy to use

Gist: Enote has to be easy to use. The motivation for Enote is to more efficiently handle data. Thus, the tool has to be easy to use.

Scale: Time used for performing functions for a trained person.

Meter: Let a person become familiar with Enote. Measure the time the person takes to a) add an address, b) find and delete an Action Point specified by a category, c) add an appointment for a given date.

Must: 3 Minutes ← intuition, design

Plan: 1,5 Minutes ← time it takes on a notebook (tests performed)

Wish: 1 Minute ← beat the notebook by supporting the search function required in each operation.

Costs:

Elapsed Time:

Scale: Time between approval of requirements specification and first delivery to the market.

Meter: -

Must: 8 Month ← Tool has to be presented at the next office fair

Plan: 6 Month ← Competitor plans to release it's product

Wish: 5 Month ←