Introducing the Agile Requirements Abstraction Model - Requirements Engineering in a Scrum Environment

Christian Hedin
<ch03@student.lth.se>

Sigurdur Órn Birgisson
<sb03@student.lth.se>

Supervisors:
Richard Berntsson Svensson, LTH
Martin Källström, Tactel

Examiner:
Björn Regnell, LTH

October 2, 2008
Preface

This Master Thesis was performed at the Mobical department of the company Tactel AB in Malmö in collaboration with the Department of Computer Science at the Faculty of Engineering, LTH, at Lund University. The thesis is the final examination of our Master of Science in Engineering degree in Information- and Communications technology. It corresponds to 30 ECTS, which equals one semester of studies.

It has been a long, but very interesting, journey that luckily did not include too many bumps. It has provided us with great knowledge to be used in our professional future. We are proud of the report presented and hope that this contribution can be an asset, both within industry and academia.

For these insights within the software business we would like to extend our deepest gratitude to:

- Björn Regnell for raising our interest of Requirements Engineering and setting aside valuable time to examine our master thesis.
- Richard Berntsson-Svensson for being a superb, dedicated and helpful supervisor at LTH.
- Martin Källström for helping out and spending time reading our work as the supervisor at Tactel.
- The external evaluators providing deeper knowledges within our work:
  - Lars Bendix, LTH
  - Tony Gorschek, BTH
  - Björn Granvik, Jayway
  - Christian Pendleton, Softhouse
- The librarians at LTH, who have patiently and skillfully assisted in retrieving literature for us.
- The Mobical team for always giving a helping hand when needed, participating in interviews and providing us with a constant stream of great music.
- Tactel AB for allowing us as thesis workers and supplying us with a fresh supply of high tech coffee.

The time at Tactel has been a very nice experience with helpful and fun colleagues in a good working environment. This has been an important factor for the success of our work. Thank you all!

Malmö, July 2008

Christian Hedin
Sigurdur Örn Birgisson
Abstract

Agile practices have been growing strong within the software development industry in recent years. In particular, Scrum has become a well known and widely accepted development framework, since the industry has realized that its approaches can yield advantages such as software projects launched on time and with higher quality. It is, however, still often necessary, or advantageous, to follow traditional practices such as requirements engineering when reaching for higher maturity levels in an organization. Agile methods and traditional software engineering are in some ways each others opposites. People and interactions over processes and tools and working software over comprehensive documentation are concepts from the Agile manifesto that illustrate how these two worlds collide. There is still much research needed to investigate in particular organizations trying to grow in maturity, and how this is affected by Agile environments.

Tactel is a leading developer of mobile applications and provides solutions and consulting services to many of the worlds major network operators and mobile handset vendors. The department of Mobical specializes in providing synchronization and backup services for mobile phone users. In order to improve product quality and reduce rework between projects, Tactel requested for a Master Thesis performing a review of the current working process at Mobical with the focus on requirements and quality assurance through testing, and that would suggest improvements that benefits the organization through higher product quality. Mobical was using Scrum as a project development framework while at the same time having traditional fixed-price contracts and project plans. The bridging of these approaches was essential in our work.

To achieve the goal of the investigation we chose to use an empirical investigation through a case study, gathering qualitative data in close cooperation with the team through observations, interviews and archive studies. This was complemented with theoretical studies and a tool evaluation to get more knowledge in the area.

The results were compiled into a process model, adapted from the Requirements Abstraction Model (RAM) according to the Agile needs of the organization, called Agile RAM (ARAM). It is a process model that visualizes and guides the work with requirements all the way from project inception to an adequate product backlog. The goal is to create well-defined and validated requirements in a backlog structure of multiple abstraction levels for bridging traditional requirements engineering and Agile methodology. To support an implementation of the model, we have provided support material in the form of summarized and detailed model descriptions, guidelines for implementations, process guides and some templates. In addition to ARAM, our contribution in this thesis consists of general process and Scrum improvement suggestions as a remedy for problems that were identified in the case study but not handled by the new requirements process model.

The model has been evaluated through internal and external reviews as well as through a workshop, and the results show that the initial goal of creating a bridge between traditional and Agile workflows in this particular case has been fulfilled. One of the key ingredients is that requirements engineering early takes the Agile aspects into account and tries to keep initial requirements at a high level of abstraction. Close customer collaboration will result in mutual trust that enables variable-scope contracts which is most suitable for Agile development. Our new model enables a solid requirements engineering process that is compatible with Scrum since well-structured requirements arrive in the product backlog at high abstraction levels and of high quality. The requirements are validated and contain acceptance criteria, which will improve accuracy and reliability of testing.
Contents

1 Introduction 10
  1.1 Background ................................................. 10
  1.2 Purpose and goals .......................................... 10
  1.3 Focus and limitations ...................................... 11
  1.4 Report composition ....................................... 11

2 Method 12
  2.1 Scientific viewpoint ....................................... 12
  2.2 Investigational approach .................................... 13
    2.2.1 Literature studies .................................... 15
    2.2.2 Interviews ........................................... 16
    2.2.3 Observations .......................................... 17
    2.2.4 Archive studies ....................................... 17
    2.2.5 Tool evaluation ....................................... 18
  2.3 Evaluation ................................................ 19
    2.3.1 Internal evaluation .................................... 19
    2.3.2 External evaluation ................................... 19
  2.4 Validity evaluation ........................................ 20
    2.4.1 Internal validity ...................................... 21
    2.4.2 External validity ...................................... 21
  2.5 Practical approach ......................................... 22

3 Theory 24
  3.1 Requirements Engineering ................................... 24
    3.1.1 Elicitation and analysis ............................... 27
    3.1.2 Documentation ......................................... 29
    3.1.3 Validation .............................................. 30
    3.1.4 Prioritization .......................................... 32
    3.1.5 Planning and management .............................. 32
  3.2 Market-driven Requirements Engineering .................. 36
3.2.1 Data management .................................................. 37
3.2.2 Market analysis .................................................... 38
3.2.3 Elicitation .......................................................... 40
3.2.4 Release planning ................................................... 40
3.3 Requirements Abstraction Model ....................................... 42
  3.3.1 Abstraction levels .................................................. 42
  3.3.2 Action steps ......................................................... 42
  3.3.3 Attributes .......................................................... 43
3.4 Software testing ....................................................... 45
  3.4.1 Basic testing principles ............................................ 45
  3.4.2 Testing approaches ................................................ 46
3.5 Software development processes ........................................ 48
  3.5.1 Defined processes ................................................ 49
  3.5.2 Empirical processes .............................................. 51
3.6 Software process improvement ......................................... 53
  3.6.1 Process and product quality ...................................... 54
  3.6.2 Process measurement ............................................. 55
  3.6.3 Defined process improvement models ............................ 56
  3.6.4 Empirical process improvement models ......................... 56
  3.6.5 Requirements Engineering process improvement ............... 57
3.7 Agile development .................................................... 59
  3.7.1 Principles and practices .......................................... 59
  3.7.2 Requirements Engineering for Agile Methods ..................... 61
  3.7.3 Synergy between Requirements Engineering and Agile Methods .......... 63
  3.7.4 Agile contracts ................................................... 64
3.8 Scrum ................................................................. 68
  3.8.1 Roles .............................................................. 68
  3.8.2 Scrum process ..................................................... 69
  3.8.3 Tools ............................................................... 72
4 Case study ............................................................... 74
5.3.6 Scrum ................................................................. 91
5.4 Agile Requirements Abstraction Model (ARAM details) .................... 92
  5.4.1 Inception .......................................................... 92
  5.4.2 Elicitation ......................................................... 93
  5.4.3 Placement ......................................................... 95
  5.4.4 Work up .......................................................... 96
  5.4.5 Documentation .................................................. 99
  5.4.6 Scrum .............................................................. 100
  5.4.7 Process artifacts ................................................ 101
  5.4.8 Special cases ..................................................... 102
5.5 Tools analysis ........................................................ 103
  5.5.1 Hansoft ............................................................ 103
  5.5.2 Mingle .............................................................. 103
  5.5.3 ExtremePlanner ................................................ 103
  5.5.4 ProjectCards ..................................................... 104
  5.5.5 Xplanner .......................................................... 104
  5.5.6 VersionOne ....................................................... 104
  5.5.7 Physical tools and spreadsheet ................................... 104
  5.5.8 Conclusions ....................................................... 104
5.6 ARAM implementation guidelines ............................................. 106
  5.6.1 Documentation ................................................... 106
  5.6.2 Team education ............................................... 106
  5.6.3 Tools .............................................................. 107
5.7 Scrum and project management issues ........................................ 108
6 Evaluation ................................................................. 110
  6.1 Internal evaluation in the Mobical team .................................. 110
  6.2 External evaluation ................................................. 113
  6.3 Evaluation conclusions ............................................... 116
    6.3.1 Problems solved ............................................ 116
    6.3.2 Advantages ..................................................... 118
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3.3 Disadvantages</td>
<td>118</td>
</tr>
<tr>
<td>7 Conclusions</td>
<td>119</td>
</tr>
<tr>
<td>7.1 Results</td>
<td>119</td>
</tr>
<tr>
<td>7.2 Fulfillment of purpose</td>
<td>119</td>
</tr>
<tr>
<td>7.2.1 Goals</td>
<td>119</td>
</tr>
<tr>
<td>7.2.2 Research questions</td>
<td>120</td>
</tr>
<tr>
<td>7.3 Generalizability</td>
<td>121</td>
</tr>
<tr>
<td>7.4 Discussion</td>
<td>122</td>
</tr>
<tr>
<td>7.5 Future work</td>
<td>123</td>
</tr>
<tr>
<td>Appendices</td>
<td>128</td>
</tr>
<tr>
<td>A Interview questions</td>
<td>128</td>
</tr>
<tr>
<td>B Tools evaluation plan</td>
<td>131</td>
</tr>
<tr>
<td>C Quality criteria for tools</td>
<td>138</td>
</tr>
<tr>
<td>D Tools evaluation results</td>
<td>139</td>
</tr>
<tr>
<td>E Sample Software Requirements Specification</td>
<td>140</td>
</tr>
<tr>
<td>F Sample Request For Quote</td>
<td>141</td>
</tr>
<tr>
<td>G Requirements Abstraction Model guides</td>
<td>142</td>
</tr>
<tr>
<td>H Story card template</td>
<td>144</td>
</tr>
<tr>
<td>I Internal evaluation questionnaire</td>
<td>145</td>
</tr>
<tr>
<td>J External evaluation questionnaire</td>
<td>146</td>
</tr>
<tr>
<td>K ARAM process guide</td>
<td>147</td>
</tr>
<tr>
<td>L ARAM requirements guide</td>
<td>148</td>
</tr>
</tbody>
</table>
List of Figures

1. The method used in our thesis work ............................................. 15
2. Strategy for performing tool evaluation ........................................... 18
3. Requirements in product life cycle (adapted from [48]) ....................... 26
4. Elicitation techniques [48] .............................................................. 28
5. Major requirements management activities [83] ............................... 33
6. The types of traceability to and from requirements [83] ....................... 34
7. Requirements state model or salmon ladder [65] ............................... 38
8. Five forces affecting product development companies [63] .................... 39
9. The requirements engineering process from a requirements selection point of view [45] 40
10. The four abstraction levels in RAM [32] ......................................... 42
11. RAM action steps [32] ................................................................... 43
12. V-model [28] ................................................................................. 47
13. The two basic testing strategies [15] ................................................ 48
14. Definition of a process model [49] .................................................. 49
15. Traditional waterfall model, adapted from [77] ................................. 50
17. Process improvement learning curve, adapted from [83] ...................... 54
18. Principal software product quality [77] ............................................ 55
19. Process improvement cycle [83] ....................................................... 57
20. Empirical management model [72] .................................................. 58
22. The cost of change in Agile (eXtreme Programming) software development [10] . 62
25. An example of a burndown chart ..................................................... 72
27. Tactel Group organization ................................................................ 74
28. Tactel Sweden AB organization ........................................................ 75
29. Main customers for Tactel [81] ........................................................ 76
List of Tables

1. Main causes of project failure [34] ................................................. 24
2. Requirements classification, adapted from [71] ................................. 25
4. Suggested requirements statuses [83] ............................................. 35
5. Examples of different kinds of software products, [65] ...................... 36
8. Process characteristics [77] ......................................................... 55
9. Maturity levels in CMM [16] ....................................................... 56
10. Example question from requirements process survey, extracted from [83] 58
13. Proposed requirement attributes .................................................. 95
14. Proposed requirement abstraction levels ....................................... 96
15. The most important criteria requested by Tactel .............................. 138
16. Additional criteria needed according to our preferences .................. 138
Document revision

- 080707 - Fixes from supervisor, ready for presentation.
- 080630 - Introduction, evaluation and conclusions added, report almost completed.
- 080613 - Full report rework, missing introduction, evaluation and conclusion sections.
- 080527 - Analysis section, except validation, complete and submitted to supervisors.
- 080508 - Method and case study sections complete and submitted to supervisor.
- 080407 - Theory section complete and first submission to supervisor.
- 080207 - Initial draft.
1 Introduction

This initial section outlines the area of research and why we have chosen our master thesis. The overall purposes and goals are presented, as well as focus and limitations, concluding with a report outline.

1.1 Background

Tactel is a leading developer of mobile applications and provides solutions and consulting services to many of the world’s major network operators and mobile handset vendors. The main focus is creating genuinely original applications that work on a wide variety of networks and handsets. The department of Mobical specializes in providing synchronization and backup services for mobile phone users. In order to improve product quality and reduce rework, Tactel has requested us to perform a review of the current requirements engineering process and come up with suggestions for improvement.

The Mobical team is using Agile methodologies, in particular Scrum, for developing software and this offers a challenging environment for requirements engineering, which traditionally is seen as strict, rigid and formalized. The clash between the fast-moving, change-embracing Agile world and the old way of working with requirements up front is very interesting and one of the main reasons why this master thesis was chosen.

This confrontation leads to our three research questions:

- RQ1: How is it possible to bridge the traditional aspects of requirements engineering with the modern and Agile methods of Scrum?
- RQ2: Is it possible to be Agile throughout contract and planning, while still maintaining good requirements engineering practices?
- RQ3: Can product quality in Agile software development be increased by introducing a requirements engineering process?

We were able to answer these questions after constructing the new requirements engineering process, and how this was done is described in the next sessions.

1.2 Purpose and goals

Tactel is in need of better requirements management, and specifically need it to fit with existing test and project management processes. The company is both looking for a new requirements engineering and project management tool, which makes it valuable to have an evaluation and comparison of existing tools that can be used for requirements engineering and works with the Scrum methodology.

The overall purpose of this master thesis was to create a suggestion for a requirements engineering process that would fit the Mobical department at Tactel. In order to fulfill this we identified certain goals:

- G1: Compilation of current academic work in the area.
- G2: Investigation of current work processes at Mobical.
• G3: Comparison of existing and reasonable priced requirements and project management tools compatible with Scrum.
• G4: Construction and formalizing of a process that aids the work with requirements without interfering the overall project management method.
• G5: Evaluation of the implemented process at the company.

These goals also provided means to answer the research questions mentioned in section 1.1.

1.3 Focus and limitations

The focus was to answer the research questions, and hence investigate how to bridge requirements engineering and Scrum. We were, in particular, interested in the actual working artifacts, usually called stories, and how they relate to more abstract requirements. Tactel requested that the new process especially considers testing and the improvement of the testing process. The new process should be tailored specifically for this organization's needs, but might turn up to be generalizable to other companies as well. This was investigated in the conclusions section (7.3).

The acquisition of background information relies heavily on gaining access to key individuals in the Mobical project group and we were limited to working with this team, so the rest of Tactel falls outside of our project scope. The new process, and any suggested tools, must also have a reasonable price tag that does not exceed what Tactel otherwise spends on project management support.

1.4 Report composition

This report is divided into several main sections:

1. Introduction - This section, which contains background, goals and document structure.
2. Method - Our scientific viewpoints and means of performing the project.
3. Theory - The bulk results of our document and archive studies.
5. Analysis - Our contribution, in form of a new process model, is presented.
6. Evaluation - The new process model is investigated by internal and external validation.
7. Conclusions - A summary of overall results, including a discussion about whether the new process can be generalized.
2 Method

This section shows our understanding of some basic ideas and problems regarding scientific work. It also explains and motivates why our methodology and way of work were chosen.

The method is a description of how to do to achieve the pre-stated purpose with the highest degree of success [14]. A clear purpose and problem formulation simplifies this process of choosing methodology. There are often several different options to choose between, and the goal is to select the option which is most likely to yield most truthful representation of reality. In real world there are also other factors that must be taken into account, such as cost. The method, however, does not dictate exactly what to do or how to perform the work, but rather act as guidelines for gaining more knowledge after setting up the goals [37].

2.1 Scientific viewpoint

All human beings have a different view of the world and how knowledge is actual truth or not. This view will highly influence the choice of methodology and ultimately the results of the academic work. Natural science has a long history of wanting to represent objectivity and truth, in contrast to i.e. social science [14]. The study of how a person views the world is called ontology with the two extremes called normalism and realism. The normalist only sees social reality as a name, concept and label to form a structure, and reality is subjective by the perceiver. The other ontological extreme, realism, sees social reality as existing regardless of who observes and interprets it.

Björklund and Paulsson [14] point out how a researchers view of the world also has impact on how the concept of knowledge is seen. The positivist considers knowledge as a cumulative process where new knowledge is added to old. The objective is true and honest knowledge ("truth"), and the means to collect it is verification or falsification of hypotheses and theories. Positivistic gathering of knowledge is about observing without influencing what is being studied. The goal is to find out how something works, not why it works. In contrast, a researcher opposed to positivism would gather knowledge through some kind of understanding. Then, not only what is being studied is important, but also its surroundings and the methods for collecting data are not as strict as for a positivist. Positivism has its roots in 19th century philosophy, but is still today embraced by many scientists as the "standard view" of science [66].

The positivistic approach is used in this report as overall scientific view and concept of knowledge, but is a general and abstract idea. To make it a bit more concrete we use Robson's [56] description of "scientific attitude", which says research should be carried out systematically, sceptically and ethically:

- **Systematically** - which means giving serious thought about what is being done and why. This is why we include this method section in our master thesis.
- **Sceptically** - which means subjecting ideas to possible disconfirmation and let observations and conclusions be thoroughly reviewed by ourselves and others. This is why we evaluate our results through internal and external evaluation.
- **Ethically** - which means that we safeguard interests and concerns for those participating in the study. This is why we keep interview attendees anonymous and only include findings that participants agree to publish.
By following these practices we aim to use an overall scientific mindset to discover some kind of "truth", but we need further guidelines still for how to write our report. Höst et al [37] have listed a number of characteristics of scientific work that we conform to:

- **Allows for independent review**: It must be possible for a third party to judge about the credibility if the knowledge is to be accepted as truth. It is of significance to simplify these kind of reviews.
- **Based on, and relates to, existing knowledge**: It is of good scientific methodology to first investigate what others have concluded within the same field.
- **Claims and results are well founded**: All claims or results should be credible and based on thorough reasoning and facts.
- **Extensive description of methodology**: To allow for independent reviews it is important to describe methodology in detail. Motives behind choosing specific investigational approaches should be explained.
- **Open accounting of sources**: To be able to continue with further studies it is important that all sources are clearly referenced.
- **Points out own contribution**: It should be clearly accounted which parts contain new knowledge.
- **Unprejudice and objectivity**: Scientific work should be performed objectively and without prejudicial influence. Positive and negative results should be equally treated, even though it is uncomfortable for someone.
- **Clear accounting of limits and threats to validity**: All new knowledge is gained in some form of context and it is important to clarify if it is limited to that context, or can be applied in a wider and more generalized field. It is also good scientific methodology to clearly and openly describe any possible flaws or threats to validity.

### 2.2 Investigational approach

There are several overall methods to choose from for academic work. Höst et al [37] have listed the following:

- **Descriptive** studies aim to find out and describe how something works or is performed.
- **Exploratory** studies are in-depth research aimed towards finding out how something works or is performed.
- **Explanatory** studies are trying to find cause and effect explanations to how something works or is performed.
- **Problem solving** studies, which is the most common method at technical universities, to try to find a solution to a problem that has been identified.

This report uses a mixture of the methods. First we need to gather knowledge (literature studies in Figure 1), in the sense of explorative collection of theory in reports, articles and books. Initially the spectrum was broad and was being narrowed down later, as knowledge was descriptively summarized in the completed theory section.
Wohlin et al. [84] describe the two kinds of research paradigms that have different approaches to empirical studies. In qualitative research objects are studied in their natural setting and explained by people. This means that different people can have different explanations and that there are multiple alternatives for interpretation. The data collected is subjective opinions of the persons participating in the research. In contrast, quantitative research includes getting larger amounts of comparable data that can be a basis for statistical conclusions.

Figure 1 shows how our work has proceeded throughout our project. Since the thesis has as a sub-goal to thoroughly investigate current working procedures we have selected to do a case study. This method is well-suited for investigating how work is performed in an organization [66]. A case study selects a specific case that is chosen depending on the purpose of the study. We chose to use the methods of interviews, observations and archive studies as described more in detail below. This suited us well since we studied the current situation in one particular part of an organization and identified problem areas for which we later constructed our solutions. This is an evidence of our work being also of problem solving nature using mostly qualitative data. Another benefit of the case study approach is that it is flexible and adaptable depending on early results and observations. There are drawbacks with this method as well. Case studies have lower degree of control than for example experiments and a small or simplified case study is not likely to scale well, with risk of not focusing on the real issues [84]. It is not either certain that the results can be generalized for other cases. Since our objective was to construct a requirements engineering process for a specific development team, the problems of scaling and generalizing does not influence us much.

The techniques we used for gathering knowledge are explained below. The analysis with problem areas and solutions are described in section 5 while the evaluation is described in sections 2.3 and 6.
2.2.1 Literature studies

Literature is all forms of written material and the information gained from literature studies is called secondary data [14]. This means that the information was originally collected for some other purpose than ours. This information gathering technique has the advantage of being able to collect vast amounts of data with quite little resources. We used the literature study to build a theoretical frame of reference for our other work.

We chose to delve deeper into the following areas, based on our initial observations and knowledges about Mobical:

- **Requirements engineering**: The basic concepts and ideas behind the requirements engineering field. This is intended to be a wide overview of current work methodology.
• **Market-driven requirements engineering**: The specific context of requirements engineering when products are developed for an open market and not one single customer.

• **Requirements Abstraction Model**: An example model for breaking up requirements into several abstraction levels.

• **Software testing**: A brief overview of the area software testing, covering basic concepts, testing approaches and principles.

• **Software development processes**: A description of common processes used to develop software.

• **Software process improvement**: Theory behind improving software development processes.

• **Agile development**: Description and examples of Agile software development, a new trend with many special characteristics.

• **Scrum**: The roles, process and tools are explained in detail for this Agile software development framework.

### 2.2.2 Interviews

We chose to perform interviews with all team members at Mobical, including all project management roles, developers and testers. The reason for this is the advantage of getting **qualitative** information that is relevant to the purpose of the thesis. The interviews were carried out during a three week period, and each session was time-boxed to last for one hour. Since the team explicitly has different roles in the projects, we used these roles when presenting the results, which gives a better overview of what was said. The different roles were "developer", "tester" and "project manager". To keep attendee responses anonymous we have used "project manager" for naming the Product owner, Scrum master/Project leader and Customer project leaders. Using interviews we could adopt the answers and delve deeper depending on attendee responses, while at the same time avoiding pointing out what is said on an individual basis. Interviews reveal even more from body language and subtle signals. The main drawbacks is that interviews are time consuming, but we believe this is well worth in expected results. Interview attendees were:

- 2 Customer project leaders
- 1 Product owner
- 1 Scrum master/Project leader
- 4 developers
- 3 testers

The interviews were **semi-structured** [66], which means that we prepared a number of questions to go through, but we were still free to discuss some areas more than others and switch order depending on replies. In some cases attendees actually answered questions before they were stated, which let us skip those questions since they were already answered. In order to keep full focus during the interview sessions, all audio was recorded on a laptop computer for later processing. This way we the interviews became less time consuming, more effective and more personal - and hence more likely to yield valuable information. On the other hand this might give a source of insecurity with the interviewee being uncomfortable with recording the conversation. The audio recordings were later transcribed into text, which yielded a good overview of the results. It also allowed us to use parts of
the interview responses in the report as support for our arguments. The transcribed interviews did on average contain 2992 words.

Privacy is one important concern about interviews, and we informed all participants that they were treated anonymously as far as it was possible. Before including quotes from individuals in the report, we asked their permission first. Each participant were also given the transcript of their interview, so they could make sure that we interpreted everything they said correctly.

2.2.3 Observations

Further knowledge was gained from observing how work was currently being carried out. Observation was performed without participation, we just noted what was happening. This contributed to our neutral standpoint. We attended as many meetings as possible throughout our time at Tactel.

The meetings we attended were:

- **Scrum standup meetings** were held daily and lasted about 15-20 minutes.
- **Scrum sprint planning meetings** were held at the beginning of every sprint, once every 4 weeks, and lasted typically 4 hours.
- **Scrum retrospectives and demos** were held at the end of every sprint, once every 4 weeks, and lasted typically 4 hours.

The Scrum sprint planning and retrospective/demo meetings were in most cases held on the same day.

We also kept our eyes open and participated wherever there was insightful knowledge to be gained, and where we were allowed. In the beginning of our work, we participated in at least three daily standups a week to get the most insight in the projects. This frequency got lower as our work progressed past the information collecting phase.

There are both advantages and disadvantages with observations as a research method. Robson describes the directness of the approach (being able to ask about feelings and attitudes in real situations) as a main advantage. The object observed however, always has a reaction to the presence of the observer. Our physical location was in the room next to the Mobical team, so there were many opportunities to listen in on interesting and relevant discussions.

2.2.4 Archive studies

To get even more understanding of the current work, we studied some different documents that according to the team and Product owner best reflected the situation. These included sales and marketing documentation, contracts, projects plans, RFQs, functional and system specifications and the current Scrum tool. We carried out these studies all the way throughout the information collecting phase of the thesis work. Robson describes this kind of studies to be used mostly from a curiosity point of view, but that the information collected from the content can give fast and overall knowledge about the past. A problem with this approach is that the documents studied are often created with another purpose and goal, which means that the observer has to take this into account when analyzing the content.
2.2.5 Tool evaluation

The tool evaluation had the purpose of investigating if a tool could assist the Mobical team in performing their tasks. Since we did not know how our proposed requirements process would look, it was important that the tools could be customizable to fit our needs. Since the team followed the Scrum development method it was important to have good support for that too. With these short criteria at hand, we found a set of tools using Internet searches and briefly looking at the available documentation and introductions. The Mobical team had themselves conducted a small evaluation of some tools that we had in mind as well.

The tool evaluation strategy is not a part of the big picture with the thesis (figure 1) but can be seen in figure 2. As shown there were different sources of background information that provided the basis for the evaluation. The theoretical studies investigated how Scrum is used and also what tools that are recommended by its practitioners. Observations, as described above, provided insight how Scrum is tailored specifically in Mobical. The other branch of gaining knowledge was rooted in the interviews with the entire Mobical team, where questions and discussions about how a tool could assist in daily work were included. The interviews were then summarized and tool related questions were compiled to a list. This list was then presented to the Scrum master who is in charge of the Scrum process. We then asked for feedback and prioritization to identify what issues were the most important. These were then investigated and a number of quality criteria for tools were constructed (see appendix C). The quality criteria and our observations of Mobical’s Scrum process were used to construct scenarios (appendix B) that contain task descriptions of how daily work with the tools could be performed. These scenarios were used when evaluating the actual tools. Behaviour and any problems experienced were noted. When each tool was evaluated we rated each of them compared to each quality criteria on a scale 0-5, where 0 means not-compliant and 5 means perfect compliance. The tool evaluation was concluded with selecting the most feasible tool, in terms of fulfilling quality criteria and the economic aspect. It is important to note that the quality criteria were not weighted, so one criteria might be more important than others. We then discussed conclusions with management at Tactel to make sure the right tool was chosen.
2.3 Evaluation

We have conducted two different types of evaluation, which are described below.

2.3.1 Internal evaluation

The internal evaluation includes evaluation activities that were performed in collaboration with the Mobical team. Because of our work situation it was easy to have spontaneous conversations with the team, creating a good atmosphere for cooperation. Since our internal supervisor was a part of the team as well, we got continuous feedback and evaluation on our work.

When we had structured our initial thoughts of the new process we had a discussion meeting with the testers (2008-04-15) to evaluate the early derivation from their perspective, since the test process would be affected. Another meeting with Scrum master and Product owner (2008-04-28) was held discussing same issues as on the test meeting and generated some good suggestions for the process. We used physical paper cards representing process artifacts and spread them out on the table to illustrate the flow.

After that, a meeting with the Scrum master (2008-05-14) took place for early suggestions and thoughts from his perspective. It also enabled us to know that we were right on track, and in particular that the issues and problems located in the case study were actually areas of concern.

After the process had been derived in its whole from the results of the above stated evaluations, another discussion meeting took place, involving all persons from the project management (2008-06-02) at Mobical. During this meeting the new process was presented and participants had the opportunity to ask questions. We wanted to save what was said during this meeting and used an audio recording here. Satisfaction from this meeting made the team want to try the process in a new real project. An implementation workshop (2008-06-04) took place, where the process was implemented in whole using a real world project, evaluating the fitness and ease-of-use of the requirements process and if the current tool could be used. Discussions after the workshop and a short questionnaire to the project management (see appendix I) was the final internal evaluation activity from which the results are specified in section 6.1.

2.3.2 External evaluation

For general evaluation we decided to complement the internal evaluation with an external one, involving four other professional experts. The specialists were chosen according to their area of interests and experience, including two Scrum experts from the industry, a RAM expert from the academia and a configuration management expert from the academia.

Lars Bendix is an associate professor at Lund Institute of Technology where he for several years has been holding courses in configuration management and extreme programming among others. He is also initiator of the Network of Excellence in Software Configuration Management. Lars Bendix provided us with academic viewpoints from both traditional and Agile aspects of software engineering.

Tony Gorschek is an associate professor at Chalmers and Blekinge Institutes of Technology, CTO of Spidexa Technologies and has been a software consultant for 10 years. He is the creator of the Requirements Abstraction Model and has numerous published journals in the field of requirements engineering. Tony Gorschek has been kind enough to provide feedback on our implementation of RAM.
Björn Granvik is CTO at Jayway, a Java software development consultant firm. He is an experienced programmer, lecturer and Scrum expert. He has almost 20 years professional experience from the industry and has interviewed experts from the Agile community, such as Jeff Sutherland and Martin Fowler. Björn Granvik evaluated our process from an industrial point of view, and in particular from an Agile perspective.

Christian Pendleton is senior consultant at Softhouse, a company that among other things provides Scrum educations and certifications. He has 14 years professional experience of working as software engineer in the fields of configuration management and process improvement. His current position at Softhouse involves Agile development models with configuration management as special area of interest. This is rather similar to what we have been investigating, and as such his input is very valuable to us.

These experts were given the finished process proposal including the introductory sections (sections 5.1 - 5.4) and a questionnaire (see appendix J). To be able to take the answers from all experts into consideration regardless of expertise, they got the same questions. We, however, realized that the purpose of this external evaluation was a little different and had another scope than the internal one, and decided not to have the same questions as for the internal evaluation, even if some of them are the same. The results from this evaluation is described in section 6.2.

2.4 Validity evaluation

It is highly relevant to investigate wether our conclusions actually are valid and adress the purpose of the thesis [84]. Good validity refers to that the results should be valid for the population of interest. Good validity does not in itself provide general validity, and we investigated generalizability further in section 7.3. Two threats to validity listed by Wohlin et al [84], internal and external validity, are shown below:

- **Internal validity.** If a relationship is observed between the treatment and the outcome, we must make sure that it is a causal relationship, and that it is not a result of a factor of which we have no control or have not measured. This means that the input should cause the output.

- **External validity.** This validity is concerned with generalization. If there is a causal relationship between the construct of the cause and the effect, can the result of the study be generalized outside the scope of the study? Is there a relation between the treatment and the outcome?

Validity is only one aspect of showing credibility in a study. Rosengren and Arvidson [67] use three categories, of which validity is one, to show how conclusions are well founded and if results are general in nature. A fourth category is presented by Björkström and Paulsson [14]. All these categories are described together:

- **Reliability** is the credibility of data collection and analysis considering random variations.

- **Validity** is how measurements are performed on what is intended to measure and keep focus on systematic problems.

- **Representability** means that the conclusions can be applied generally.

- **Objectivity** is how the authors preconceptions and opinions influence the study.
Data collection and analysis was thoroughly described and the reader can get an opinion about reliability through the methodology section. As discussed in 2.2.2, the interview transcripts were sent to the participants - which increases reliability and makes sure that their view is interpreted correctly. Sources for the literature study, discussed in 2.2.1, had to be carefully selected and we consider articles that have been peer reviewed and/or published to be more trustworthy than other information. We have always tried to use recent literature, since there is a lot of change in the scientific fields we have investigated. There are no quantitative methods used in this thesis, so we do not need to take any statistic selection into account.

Validity is about the connection between the object that is intended to be measured, and what is actually measured. To further find weight behind our literature study sources, we have often checked in multiple sources for the same information in a process called data triangulation [14]. It is perfectly possible to have high reliability without good validity. This can be illustrated by a dart board with all darts hit closely together (good reliability), but far from the bulls eye (bad validity). We will discuss validity further in the next two sections.

To find validity in our process derivation from the problems identified, both internal and external evaluations were performed, as described in sections 2.3.1 and 2.3.2.

Representability for a result depends on how data is selected. Our empirical section is the case study and conclusions drawn cannot be generalized, except in environments that are very similar to what we have studied. Representability can be increased by describing the context in detail [37].

Objectivity in this report is increased by clearly stating and motivating our choices made when selecting literature, interview participants and other information. The reader will then be able to judge our results and hence increase objectivity.

2.4.1 Internal validity

During the interview sessions there were a number of threats to validity. Participants could fear having their identity exposed and be hesitant with disclosing information that might be sensitive or harm them in some way. This risk is amplified by having all interviews recorded on audio. However, we never noticed any signs of anxiety or participants being uncomfortable with talking to us. There is also a risk of participants wanting to “please the interviewer” by providing the answers they think we wanted to hear. During meetings, such as the daily Scrum meetings, there are several risks of us getting the wrong information. The team members could be reluctant to “act like usual” when we are present, in the sense that they try to behave more structured than normally or skip certain procedures for our sake. We judge this risk to be quite low since we have been present in close proximity for several months and the team is used to having us around. That also implies the risk of team members allowing their personal feelings towards us to influence interview answers and observed behaviour, but we deem this risk rather low since we have been working very independently and only passively observed the team.

2.4.2 External validity

As described above, external validity deals with how generalizable the results are. We cover this issue in section 7.3.
2.5 Practical approach

After initial planning and meeting with the supervisor from the Department of Computer Science at Lund University we officially started working on the master thesis in the beginning of February 2008. The first week was dedicated mostly to planning, finding out our goals and writing a project plan. We also started straight away with reading some articles that our supervisor knew were relevant information. The first weeks we had informal talks with some members of the Mobical project team, to get a picture of what they worked with and explain about our plans. We also started consuming large amounts of articles and books. Some were recommended by our supervisor but the vast majority we found by searching through the university article database (ELIN). Simultaneously preliminary interview questions were formulated and the theory section was coming to life. We also rather early started attending daily Scrum standup meetings, to get a good information of what was going on and further insight how work was being performed. By week 3 the interview questions were finished and sent to our supervisor for feedback.

All through out the project we have had meetings with our supervisor about every 2 weeks. This has been valuable opportunities to check that we were on track with time planning and could ask if we had understood complex theory correctly. To be able to keep better track of our status, a wiki with for example “TODO-items” and log for past work were used throughout the project.

Week 5 we started with the first interview sessions. We rather quickly learned that timeboxing is important and that we have to keep a good pace for the interviews not to take too long. The interviews were transcribed from audio recordings, which proved to be valuable but a bit time consuming.

Week 6 was fully scheduled with interview sessions but we also managed to set aside some time for visiting a Scrum-seminar hosted by Softhouse in Lund. This gave us further insight into how Scrum works, and also provided us with some easy-to-read guides and pictures that we used to explain points about Scrum to the Mobical team members the coming week.

We took a break from the thesis during Easter week.

Week 9 we finished the theory section and reviewed it quite thoroughly before submitting it for formal review by our supervisor. The theory section proved to be quite well written from the start, but we still got good feedback but postponed doing the fixes until later since we still had to change the reference-system in use by LaTeX. The week after we also started preparing for tool evaluations by formulating goal of evaluation, key concerns and did an informal interview specifically about tools with the Mobical Scrum master. The next week tool evaluation was performed.

Week 12 we wrote about methodology and compiled results from the tool evaluation into the report and appendices. At the same time the Mobical team changed their Scrum process (partly because of our feedback) since they felt an urgent need to get more control. This meant that some parts of our interviews were not accurate anymore, which was taken into account when writing the case study section which started the week after. The current process description was extracted from interview transcripts by finding key concerns that were present in many interviews or constituted obvious problems. These key issues were then sorted and organized into a list that was presented to the Product owner and the Scrum master of Mobical. We discussed the issues and made sure they were accurate. They were then used as basis for writing about current process in the case study section. During this week we also started converting references into bib format. Week 13 was a short week due to holidays, but we still made good progress and wrote additional information about testing in the theory section, which we felt was relevant in the context of Mobical.

Week 14 we continued with writing company and background information in the case study, as well as further work in the methodology section. We also added some information about Agile contracts. The method and case study sections were completed and submitted to supervisor. Week 15 started
off with a meeting, making an outline of the rest of our progress and presenting our initial process thoughts. The rest of the week was spent on discussions about the process also involving the Scrum master and some fixes in the report. The analysis breakthrough was made week 16 when the analysis of our own process was specified and got the real structure. The whole analysis section got a shape up and was sent to the supervisors for inspection in week 17. The rest of that week and the next was spent on rework for the whole report along with starting the evaluation of our process in cooperation with Mobical. A meeting with Mobical management followed by a workshop two days later was held in week 18 that gave us great input on the process. We also sent the analysis on external evaluation this week along with workshop follow-up questions to the Mobical team.

Week 19 was spent on bibliography rework for the whole report and evaluation thoughts in the analysis. After deriving "TODO-items" from the document, and fixes thereafter, there was a great satisfaction about our work so far. This week also contained oppositions on two other thesis works. Gathering the evaluation thoughts from our internal and external evaluations, week 20 was spent on finalizing the analysis section with the evaluations.

The last two weeks were spent on doing a lot of rework and finalizing the conclusions section. We also provided the team with two "Cheat sheets", short guides, on how to work with our process. When the report was finished we prepared the presentation material.
3 Theory

This section summarizes current research in areas relevant to the master thesis. First the overall definitions and concepts of requirements engineering in general are explained, after which the focus shifts to market-driven requirements engineering. The Requirements Abstraction Model is studied as a case study followed by some basic principles and concepts in the area of software testing. The next section focuses on different software engineering processes and models for how to improve them. Finally in the last sections, the theory and ideas behind Agile development and Scrum are investigated thoroughly, including how to combine best practices with requirements engineering.

3.1 Requirements Engineering

The lack of common definitions is a problem in the software industry [83] and the definition of requirement is no exception. The word has different meaning depending on who you ask and at what level of detail it is observed. The IEEE Standard Glossary of Software Engineering Terminology [41] defines a requirement as:

1. A condition or capability needed by a user to solve a problem or to achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
3. A documented representation of a condition or capability as in 1 or 2.

Sommerville and Sawyer [78] use the following definition: Requirements are... a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system.

Why are requirements an important issue for software development companies? According to the CHAOS report made by the Standish Group [34] five of the eight most significant factors for project failures are related to requirements, as presented in table 1. Incomplete requirements is the most outstanding issue, but also customers low level of involvement is causing problems to many projects. The study also concluded that 52.7% of sample projects cost 189% of their original budget estimates, but still only contained 42% of the planned features.

<table>
<thead>
<tr>
<th>Problem</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incomplete requirements</td>
<td>13.1</td>
</tr>
<tr>
<td>Low customer involvement</td>
<td>12.4</td>
</tr>
<tr>
<td>Lack of resources</td>
<td>10.6</td>
</tr>
<tr>
<td>Unrealistic expectations</td>
<td>9.9</td>
</tr>
<tr>
<td>Lack of management support</td>
<td>9.3</td>
</tr>
<tr>
<td>Changes in requirements</td>
<td>8.7</td>
</tr>
<tr>
<td>Lack of planning</td>
<td>8.1</td>
</tr>
<tr>
<td>Useless requirements</td>
<td>7.5</td>
</tr>
</tbody>
</table>

Table 1: Main causes of project failure [34]

As seen in table 2 there are many different classes of requirements. Functional and non-functional requirements are classes of requirements. The Goal/Domain/Product/Design-level requirements
describe at what perspective they should be observed, from abstract to concrete. Business requirements describe commercial opportunities and technical requirements how something should function. Product requirements can be contrasted versus process requirements where the flow of interaction is described. Role based requirements are seen from the perspective of a particular actor. In real life environments it can be challenging to correctly classify requirements in a meaningful manner [7].

| **Functional requirements** - what the system will be capable of doing. |                  |
| **Non-functional requirements** - constraints on the types of solutions that will meet the functional requirements e.g. accuracy, performance, security and modifiability. |                  |
| **Goal level requirements** - related to business goals. |                  |
| **Domain level requirements** - related to problem area. |                  |
| **Product level requirements** - related to the product. |                  |
| **Design level requirements** - what to build. |                  |
| **Primary requirements** - elicited from stakeholders. |                  |
| **Derived requirements** - derived from primary requirements. |                  |
| **Business requirements versus technical requirements.** |                  |
| **Product requirements versus process requirements**, like business needs versus how people interact with the system. |                  |
| **Role based requirements**, e.g. customer requirements, user requirements, IT requirements and security requirements. |                  |

Table 2: Requirements classification, adapted from [7].
According to Lauesen [48] Requirements Engineering (RE) is the process of identifying, analysing, documenting, validating and managing [software] requirements. A sample requirements engineering process for a product's life cycle is illustrated in figure 3. The project starts with inception and an idea. A project team is formed and scope and business goals are outlined. Elicitation means to gather and collect requirements, which are then specified and validated for correctness. When the customer has compared different competing proposals, the two parties can sign the requirements and the contract. The requirements are then turned into source code and working software. When the product is ready for deployment it is tested for acceptance and delivered. This is a very traditional product development process. We will discuss these kind of processes in more depth in section 3.5.1.

The following steps described by Karlsson [42] are followed in a standard requirements engineering process, and quite similar to the process outlined by Lauesen.

- Elicitation and analysis
- Documentation
- Validation
- Prioritization
- Planning and management

Figure 3: Requirements in product life cycle (adapted from [48])
The next sections will describe each of these steps in further detail.

3.1.1 Elicitation and analysis

The process of finding and formulating requirements is called elicitation [48]. Overall goals of the system are preferably elicited first and then further investigation goes into more detail. It is not necessarily done in a fixed order since depth studies may show that the overall goals were wrong, and need to be changed.

There are many different techniques used for elicitation, but as the ultimate goal of RE is finding out the needs of the stakeholders, an intuitive approach would simply be to ask them. Unfortunately, there are a number of barriers that makes this approach more difficult than initially conceived [48]:

- It is common that stakeholders cannot express exactly what they need. Sometimes they see problems, and even exaggerate them. Sometimes no problems are seen at all. Even in the cases where they see the problem, it can be difficult to express this as a requirement or need.
- Often users cannot explain their task descriptions or why they actually perform them.
- There is a risk that stakeholders state a solution instead of specifying a demand.
- Sometimes stakeholders cannot imagine new ways of doing things, or the impact of doing familiar tasks in a new way.
- There are different kinds of stakeholders that can have conflicting interests. For example, marketing personnel could have another view of the goals than developers.
- Many people are generally resistant to change, whatever the benefits. One reason for this can be the above stated point that stakeholders can have difficulty imagining new ways of doing things.
- Many of the proposed requirements are not essential and could simply be nice to have features. It can prove problematic to have stakeholders agree on what is relevant or not.
- Demands, external factors and priorities change over time. This can also lead to a number of new requirements that were not present at the start of project.

There are many different kinds of elicitation techniques that are useful in different phases of development. Figure 4 is a listing of elicitation techniques by Lauesen [48]. Each technique is noted by a gray shade, which signifies how useful the elicitation technique is in different areas of investigation. Lighter shade means less useful and dark shade means very useful. For example, the figure shows how stakeholder analysis is helpful for investigating goals and key issues. Some of the more common techniques are further described in detail.

Interviews

Interviewing stakeholders is a good method to gain knowledge about current work in the domain and what problems are experienced. Many consider this the main elicitation technique and depending on the questions asked it can be quite flexible [48]. The right people should be invited to interviews and the representatives should know what is going on in the business. The right questions should be asked, with focus on critical issues. It can also be interesting to find out why tasks are carried out in a certain way. Interviews can be individual or in group. Being in group might inspire to more discussions but can also make some people feel unsafe about talking freely.
Figure 4: Elicitation techniques [48]
Focus group meetings

Focus group meetings are a kind of structured brainstorming sessions. Sometimes the term future workshop is used as well. The group meeting starts by having participants describing the current way of doing things and what problems can arise. They then try to imagine the ideal way of performing the tasks and explain why the ideas are good. Different kind of stakeholders should participate and this makes it excellent as a method to provide understanding between the different groups of stakeholders [48].

Use cases

The use case approach to requirements elicitation is taken from object oriented modelling and a use case describes a sequence of interactions between a system and an external actor that results in the actor accomplishing a task that provides benefit to someone [83]. The actor could be a person, a piece of hardware, another software product or something similar. Use cases represent functional requirements and can be used early in the development process.

Observation

This elicitation technique aims to observe users as they are performing tasks. Observation can be direct with the investigator present, or indirect where the task is recorded on video or by other means of remote observation [58]. Users are not always aware of how they do things, or might have problems expressing it and that is where observation can help. The technique is best suited for investigating current work and associated problems. The main problem is that it is unlikely to catch the critical situations that only occur sometimes [48].

When requirements have been elicited they are analyzed and often first result in the construction of abstract descriptions and models. This can be i.e. creating Entity-Relationship (ER) diagrams in order to visualize how a database should be understood, manipulated and managed. Dynamic behaviour can be modelled with flow charts. The primary benefit of modelling requirements is the opportunity for analysis they provide [55]. When developers are not certain about the requirements Wiegers [83] suggests to create a user interface prototype. This is a partial possible implementation that shows concepts and allows users to evaluate it to make sure project participants understand the problem domain.

3.1.2 Documentation

When requirements are elicited they must be satisfactory documented. Usually this is done in a Software Requirements Specification document (SRS) or a requirements database or some other software requirements tool. One way of storage does usually not exclude the other. There are at least three factors why it is important that the documentation of requirements is of high quality [42]:

- It will function as a basis for all coming development activities, since it describes all requirements that are to be fulfilled.
- It can function as a formal contract between supplier and customer and it is hence important that all stakeholders agree on the content and interpret it in the same way.
- It is often used in the testing phase to verify that the system really fulfills requirements.

The IEEE-standard suggests a structure for SRS documents as seen in table [3]. This provides a good basis for writing requirements documents, but often it is necessary to tailor it specifically for the particular needs of the organisation. Especially the Specific requirements section is very general and will differ a lot depending on needs.
1. Introduction
1.1 Purpose of the requirements document
1.2 Scope of the product
1.3 Definitions, acronyms and abbreviations
1.4 References
1.5 Overview of the remainder of the document

2. General description
2.1 Product perspective
2.2 Product functions
2.3 User characteristics
2.4 General constraints
2.5 Assumptions and dependencies

3. Specific requirements
This covers functional, non-functional and interface requirements.

4. Appendices

5. Index

There are many different styles to write requirements. The more sophisticated range from UML diagrams and flow charts to ER diagrams and scenarios, but most companies use Natural Language (NL) which is the language of customer. Different stakeholders have different vocabulary though, which can make NL problematic if it is not clear how to interpret the specification. We will further describe how to write good specifications in the next sessions.

3.1.3 Validation

The validation activity is a process constantly in progress, with the main objective of making sure documented requirements are describing the wanted properties and describing them in the right way.

Validation tries to answer the following questions:

- Do all parts match and is everything included?
- Are the stakeholders happy?
- Where are the major risks?
- Is the specification consistent with product goals?

A good requirements specification should fulfil a number of quality criteria:

- Each requirement should be correct and reflect a customer need or expectation.
- The specification should be complete and cover all necessary requirements. This is not realistic to achieve, but at least cover business goals and critical issues.
• Requirements should be **unambiguous**, which means that all parties agree on what they mean. There is a big risk that for example the developers think they know what a requirement means, when in fact it means something else.

• **Consistency** means that no parts in the specification conflict. Two requirements should not contradict each other.

• That the specification is **ranked for importance and stability** means that all requirements have a prioritization and a notion of how likely it is expected to change. Prioritization is an important aspect of requirements engineering, and we discuss this topic more later in this report. Stability ranking helps supplier identify what functions should be easy to modify.

• If a specification is **modifiable**, then it is easy to change without affecting consistency. A numbering and referring scheme will help to achieve this.

• It should be possible to **verify** that the product matches the requirements. This is an essential part of software testing as described in section 3.4.

• A specification is, according to IEEE [1], **traceable** if it is possible to see where requirements come from and where they are used in design and code. Traceability will be described further in following sections.

There are several techniques that can aid in validating requirements. One common alternative is the manual process of **requirements review**. Multiple readers from contractor and client staff check the requirements for anomalies and omissions [7]. There can be formal and informal reviews, where the latter are loose discussions between stakeholders. During formal reviews, the development team walks the client through the requirements document. Each requirement is checked against the quality criteria above.

One important aspect is that all stakeholders agree on the requirements. Sometimes this can be difficult due to different interpretations of textual requirements that do not provide dynamic representations of the system. A technique for overcoming this obstacle is called **prototyping**. Three different kinds of prototypes are described below [48]:

• **Rapid prototyping** aims at constructing quick prototypes for both validation and elicitation purposes. They are unstable and destroyed after usage.

• **Evolutionary prototypes** are based on a foundation of stable and correct requirements. When new requirements are identified, the initial prototype have those added to finally result in a complete system.

• **Operational prototypes** is a combination of the two above prototypes. A stable base is created and then an unstable layer is added for elicitation and validation. The unstable layer is collapsed when stable requirements are identified. The prototype evolves in this way, until it is considered a complete system.

**Test case generation** can be performed as part of validating requirements since good requirements should be testable [7]. Through early construction of test cases, the test engineers gain a better understanding of the requirements and can contribute with identifying weaknesses. They also bring a new perspective on requirements, for example if it is hard to create test cases it could point on weaknesses in requirements [24]. Making test cases before program code has been written is sometimes called **test first** in Agile methods.
3.1.4 Prioritization

An important activity in requirements engineering is prioritization. There are always more requirements than there are resources for fulfilling them all, and prioritization is also one of the most difficult activities \[42\]. Research has shown that few IT-companies have efficient and systematic methods for prioritizing requirements \[50\]. Requirements can be said to follow the Pareto principle, that 80% of the utility come from 20% of the causes; in this context, requirements \[42\].

The objective of the prioritization activity is to identify good requirements, those that bring high value compared to the cost of implementing them \[45\]. Sub goals with prioritization include planning which requirements should be incorporated in which release, handling contradicting requirements and assigning the right resources to the right requirements.

There are several different techniques to choose from when prioritizing requirements. According to IEEE \[1\] some can be considered absolute, such as marking requirements as essential, conditional or optional. Other techniques are relative where all requirements are weighted against each other and given different priorities. Absolute priorities allow many requirements to have the same priority, whereas relative ones have different values.

A common technique for prioritizing requirements is simply to ask stakeholders how important the different requirements are. This is usually done with an absolute scale, such as rating requirements from 1 to 5. The problems with this method is that stakeholders tend to consider all requirements to be important and that stakeholders think of prioritization as a part of the negotiations. Absolute ratings are very difficult to use reliably since there is no common point of reference \[42\].

In some Agile software development environments the Planning game \[10\] is often used for prioritization. Written requirements (so called story cards) are elicited from the customer, who then sorts the cards into three piles: 1) Those without which the system will not function, 2) those that are less essential but still provide significant business value and 3) those that would be nice to have. Developers then estimate how long time it will take to implement the stories by putting them into three piles again: 1) those that they can estimate precisely, 2) those that they can estimate reasonably well and 3) those they cannot estimate at all. Based on these decisions customers prioritize which requirements are most important and should be planned for the next release. The result of the Planning game is an ordered list of requirements which in fact is similar to results from a numeral assignment algorithm \[44\].

An approach suggested by Karlsson and Ryan \[44\] is to use pairwise comparison with the Analytical Hierarchy Process (AHP). This lets stakeholders weight requirements against each other and after some calculations conclude which are most important. If there are \( n \) requirements to prioritize, the total number of comparisons to perform is \( n(n - 1)/2 \). For each pair the relation is estimated on a scale \{9, 7, 5, 3, 1\} where 1 represent equal importance and 9 that one requirement is much more important than the other. AHP also includes a consistency check where judgemental errors can be identified.

3.1.5 Planning and management

Requirements management includes all activities that maintain the integrity and accuracy of the requirements agreement (baselined requirements) as the project progresses. Requirements management emphasizes \[53\]:

- Controlling changes to the requirements baseline.
• Keeping project plans up to date with the requirements.
• Controlling versions of both individual requirements and requirements documents.
• Managing the relationships between requirements, and links or dependencies between individual requirements and other project deliverables.
• Tracking the status of the requirements in the baseline.

Requirements management is also closely linked to configuration management and change management, as well as impact analysis of changes \[42\]. The goal of configuration management is to maximize productivity by minimizing mistakes \[8\]. Wiegers \[83\] suggests the requirement management activities as shown in figure \[5\]. We here further investigate these activities in detail.

**Change control** is the task of managing changes in the requirements. Since source code and tests depend on the requirements it is important to have a structured procedure to follow when a change request is filed. Daniels \[20\] provides a model for the change process in configuration management, but it can also be applied to change management in requirements engineering. The first thing to do is analyse and document the change proposal. It is then reviewed and distributed to other parties. The impact of the change is investigated and a Change Control Board (CCB) decides if the change is accepted or not. Reasons for denying the change could be that it is considered faulty, unnecessary or too expensive. The decision is documented and then the change is implemented and verified. The CCB plays an important role in this process and acts as the decision authority. The standard CCB structure consists of a chairperson (often program manager of equivalent), secretary (often configuration manager), members (representatives from major components, i.e. testers, developers) and specialists (called to meeting on a case-to-case basis when needed) \[20\]. All this administration and bureaucracy might seem unnecessary, but it allows for traceability and auditable.

**Version control** is the aspect of requirements management that handles different versions of the requirements specifications. On a larger scale it includes all kinds of project artifacts to be version controlled by the configuration manager. Since design and testing is based upon requirements, it is very important with proper version management so that the product reflects the latest requirements. Versions of requirements specifications can be called revisions, where a new version is intended to supersede the old one \[8\]. The most powerful approach to version control is to store the requirements in a commercial requirements management tool \[83\].

**Requirements traceability** refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction \[33\]. A software requirements specification (SRS) is traceable if (i) the origin of each of its requirements is clear and if (ii) it facilitates the referencing of each requirement in future development or enhancement documentation \[11\]. Figure \[6\] shows the different types of traces to
and from requirements. Customer needs are traced forward to requirements, enabling visualization of which requirements change if the customer wishes a change. Requirements can be traced backwards to show their origins. The bottom part of the figure shows how requirements can be traced forward by defining links between individual requirements and specific product elements, such as software source code. The final type of link traces specific product elements backwards to requirements. Most applications include code that does not directly relate to user-specified requirements, but the programmer should at least know why every line of code is written [83].

![Figure 6: The types of traceability to and from requirements](image)

To enable traceability, each requirement must be uniquely labeled to be able to refer to it unambiguously [21]. Traceability links help to keep track of parentage, interconnections and dependencies among individual requirements [83]. There are many benefits with having good traceability in software projects: Traceability links can be used for certification, to demonstrate that all requirements actually are implemented. If requirements are to be added, deleted or changed, tracing will assist in change impact analysis. Reliable traceability can improve productivity during maintenance since it otherwise might be necessary to build one piece at a time as code is refactored or enhanced. Tracing will provide accurate record of implementation status and greatly assist in project tracking. Missing links can indicate that a feature has not been implemented yet. Another benefit of having good traceability is the connection between requirements, code and testing, that allows developers to quickly locate parts of the code to look for a bug when a test fails [83].

**Status tracking** is an important aspect of requirements management [16]. Overall project status can be examined by checking the percentage of requirements that has a certain status. When setting up a requirements process it should be decided which statuses to use. Wiegers [83] suggests the use of statuses seen in table [4].
<table>
<thead>
<tr>
<th>Status</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed</td>
<td>The requirement has been requested by an authorized source.</td>
</tr>
<tr>
<td>Approved</td>
<td>The requirement has been analyzed, had its impact analyzed and allocated to a baseline or release.</td>
</tr>
<tr>
<td>Implemented</td>
<td>The code that implements the requirement is designed, written and unit tested.</td>
</tr>
<tr>
<td>Verified</td>
<td>The requirement has been verified through the projects verification procedure. It is not considered to be complete.</td>
</tr>
<tr>
<td>Deleted</td>
<td>The requirement has been deleted and an explanation why and by who is added.</td>
</tr>
</tbody>
</table>

Table 4: Suggested requirements statuses [83]  

It can be helpful to visualize statuses and plot a graph with time on the x-axis and percent of requirements on the y-axis. Project staff will then (hopefully) see how statuses change from proposed and approved to implemented and verified over time. It does not show the number of requirements, but illustrates how the goal of complete verification of all approved requirements is approached [83].
3.2 Market-driven Requirements Engineering

The traditional requirements engineering process focuses on development effort for one single customer and is said to be customer specific or contract-driven. The customer often represents a large organisation such as a military, governmental or financial institution [45]. The requirements specification acts as a contract between supplier and customer and all details, such as release date, cost and scope is negotiated.

In contrast, some products are offered to an open market where it is not always known in advance who the customers are, but it could be organisations or end-consumers. Requirements engineering in this case is labeled as market-driven and the products are often called commercial-off-the-shelf (COTS) [45]. Typical products in this category are mobile phones, computer software and development tools. Market-driven requirements engineering (MDRE) covers traditional RE activities described in the previous section, such as elicitation, specification and validation but they are adapted to a market-driven situation with many customers. The distinction between market-driven and customer-specific development is not strict and it is not uncommon that a development organisation both sells a generic product to an open market and tailored customer-specific versions [65].

Software products can be characterized based on the two dimensions: degree of customization and hardware/software content. Table 5 shows an example of different kinds of software products. For example, a portrait painting is highly specific to the customer (the one being portrayed), and does not contain any software at all. The other extreme is a firewall, which is fully made up of software and is a generic product.

<table>
<thead>
<tr>
<th></th>
<th>Pure hardware</th>
<th>Embedded systems</th>
<th>Pure software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic</td>
<td>Note sticks</td>
<td>Mobile phone</td>
<td>Firewall</td>
</tr>
<tr>
<td>Customized</td>
<td>Office furniture</td>
<td>Customized car</td>
<td>Enterprise resource planning systems</td>
</tr>
<tr>
<td>Customer-specific</td>
<td>Portrait painting</td>
<td>Military vehicle</td>
<td>Web site</td>
</tr>
</tbody>
</table>

Table 5: Examples of different kinds of software products, [65]

MDRE is different from traditional RE in a number of ways because of the market aspect. Success is measured by sales, market share and product reviews, while in the traditional contract-driven case it is determined by customer satisfaction and contract fulfillment [65]. Additional differences between market-driven and contract-driven product development can be seen in table 6. Market-driven development has a large span of customers, potentially the entire market. That also means there are many competitors. Content and pricing is not fixed in a contract and project success is determined by sales, market share and profit. Typically many versions of the software is released iteratively and the requirements specification is often less formal. The market-driven process deals with managing streams of requirements, prioritizing and cost estimating them as well as planning for future releases.

According to Regnell [65], typical MDRE is characterized by the developing organization taking all decisions, but also all risks. Requirements appear continuously throughout the product lifetime and the sheer volume of requirements is very large. Most requirements are only informally described. The product is evolving through multiple releases and release planning focuses on time-to-market and return-on-investment (ROI).

A number of challenges with MDRE were identified in a survey [43] based upon interviews at five different companies of varying size and maturity.


<table>
<thead>
<tr>
<th>Market-driven</th>
<th>Contract-driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>Many customers</td>
<td>One customer</td>
</tr>
<tr>
<td>Many competitors</td>
<td>No competitors</td>
</tr>
<tr>
<td>Content is very influenceable</td>
<td>Content is specified in contract</td>
</tr>
<tr>
<td>Pricing is very influenceable</td>
<td>Pricing is specified in contract</td>
</tr>
<tr>
<td>Success criteria: Sales, market share, profitability</td>
<td>Success criteria: acceptance, satisfaction</td>
</tr>
<tr>
<td>Many iterative releases</td>
<td>One release, then maintenance</td>
</tr>
<tr>
<td>Often less formal specification</td>
<td>Contract between supplier and customer</td>
</tr>
<tr>
<td>Process: Manage stream of requirements, prioritization, cost estimation, release planning</td>
<td>Process: Requirements elicitation, modelling, validation, conflict management</td>
</tr>
</tbody>
</table>

Table 6: Some differences between market-driven and contract-driven product development

- **Balancing market pull and technology push.** What do customers need and what is possible to do with technology? Innovative technology can give a competitive edge, but it is also important to satisfy users’ needs.

- **Chasm between marketing and development.** Many companies report internal problems and different viewpoint between the development and marketing departments.

- **Organisational instability and market turbulence.** When key personell leave for another company or department, they can leave large gaps in knowledge that are hard to cover for. When the organization changes a lot it is hard to implement stable and repeatable processes.

- **Simple tools for basic needs.** Process support tools are often complex and difficult to use.

- **Requirements dependencies.** Release planning is complicated by dependencies among the requirements.

- **Cost-value-estimation and release planning.** Underestimation of cost can result in exceeded deadlines while overestimation may exclude valuable requirements.

- **Overloaded requirements management.** MDRE elicitation is continuous and gathers vast amounts of requirements, which may make it difficult to get an overview and filter out bad requirements.

The following sections will go deeper into specific important areas of market driven requirements engineering and describe them in more detail.

### 3.2.1 Data management

Because of the vast amount of requirements in MDRE it is extra important to provide good management of data. Two key concepts are the requirements state model and the requirements repository. The simplified view of these models as used by Regnell is presented below.

**Requirements state model**

This model is also called salmon ladder, because of the uncertainty of a salmon to get back upstream to the breeding currents. Requirements can be received at any time, but releases are done at fixed dates. In continuous mode requirements are received and registered by the product manager from all kinds of sources. In release mode the content of next release, the scope, is frozen and changes only added through a formal change procedure. It is during release mode that development and verification is performed. Figure 7 illustrates the requirements state model.
Requirements repository
In order to store many requirements, many development teams use a repository. For small amounts of requirements this can be a simple spreadsheet, but for large-scale projects it is better to use a real requirements management tool [65]. This is especially important in MDRE because of the amount of requirements.

3.2.2 Market analysis
To elicitate requirements, the market must first be analyzed. Different customers have different needs and the better we understand our customers, the higher probability of successful products [43]. One important aspect of analyzing the market is to segment it. Market segmenting is done in four steps: 1) Strategy, 2) Segmenting, 3) Focus and 4) Positioning.

Strategy
The first step is to formulate a strategy to fit the product being developed. The classical marketing strategy study by Ansoff suggests the different strategies in table 7.

<table>
<thead>
<tr>
<th>Current products</th>
<th>New markets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market penetration</td>
<td>Market development</td>
</tr>
<tr>
<td>Product development</td>
<td>Diversification</td>
</tr>
</tbody>
</table>

Table 7: The classification of market strategies [6]

*Market penetration* means to further go along the same road to use current products in the current market and gain more customers. This strategy is also called "do-nothing". *Market development* means moving your product into new markets where it has not been before and find new customers. *Product development* is about enhancements or additions to current products. If it is considered easier to sell new products to existing customers than current products to new customers, this might be a good
strategy. The last strategy, diversification, is to offer a new product to new customers. This can be seen as expanding the overall goals of the company [43].

Segmenting
The next step is based on the first one. Market segmenting aims to divide the market so product development can focus on fulfilling the needs of customers that are likely to purchase the product. A segment is an identifiable group of customers with similar needs, buying patterns, geographic location or other similarities [43]. Common variables for segmenting are geographical (such as region and city), demographic (such as age, education, income) and lifestyle (such as personality and attitude).

Focus
When market segments have been identified the next step is to focus on the right market. It is often not viable to choose all segments so the challenge lies in trying to find the most attractive segment. Key factors to take into account are market size, market growth, current position, profit levels and competitor situation [43]. Porter describes the five forces that affects all product developing companies seen in figure 8 and described below.

![Five forces affecting product development companies](image)

*Figure 8: Five forces affecting product development companies [63]*

*Threat of new entrants* means threat from new competitors. More companies competing in the same market segment will often bring lower prices to the product and lower profits [43]. *Bargaining power of customers* is how strong the buyers are in terms of being able to put pressure on the companies to lower prices or their ability to make good judgment in terms of quality versus cost. *Bargaining power of suppliers* reflects the position of suppliers on the market. In some cases suppliers can be very powerful, like if they have a monopoly in a particular market. *Competitive rivalry within an industry* is competition between established companies within the segment. Often decisions by one company (i.e. lowering prices) will affect the other competing companies [43]. The entire industry can also be affected negatively, for example if one company has low quality of products. *Threat of substitute products* is the competition from products not in the same industry, but still attractive alternatives to potential customers. These alternative products acts as a kind of roof for industry earnings, since their price level influences how much money can be earned.

Positioning
How companies position their products depends on if they are market leaders or not. Market leaders tend to not mention competitors at all, while trying to maintain the number one position. They also
often create internal products to compete with, in a product developing strategy. New products and companies trying to gain market share try to point out that they are better alternatives than the market leaders. Good positioning makes it easy for customers to realize the product benefits and in some cases it is clear that the product brand name equals the general name of the product as can be seen with products such as Xerox, Inlines and Frisbee [43].

3.2.3 Elicitation

The traditional requirements engineering activity of elicitation is in MDRE not governed by any external request or customer order, but is continuous by nature, where development efforts are initiated as needed according to a release plan [30]. MDRE uses the same type of elicitation methods as in traditional RE, but some methods are more common than others. Requirements elicitation in MDRE is often devoted to innovation combined with market analysis [65] as described in the section above. To identify customer needs is one of the hardest, but also most important, aspect of market driven product development [43]. Customers are obviously the greatest source of requirements, but there are many other good methods for elicitation in a market driven environment. Requirements can also be found by looking at competitors, from within the company itself and by looking at support centres. Research based on interviews with industry practitioners [46] has shown that market surveys rarely are performed in market driven companies. Instead other techniques such as customer visits, meetings with sales personnel and working in customer projects. Many requirements also originate from internal sources, such as ideas from developers. Some companies prefer to let customers send feedback and failure reports (that can be seen as requirements) directly from within the product.

3.2.4 Release planning

The activity of selecting an optimal subset of requirements for realisation in a certain release is called release planning. In MDRE a vast quantity of requirements are collected, but it would not be cost-efficient to try to implement all of them. Release planning can thus be said to be an attempt to optimize the relation between the requirements’ value and cost in order to achieve stakeholder satisfaction [45]. One major activity to achieve this is, as in traditional RE, requirements prioritization. The selection of requirements for a release is often made in several steps of the RE process, since release plans are revised and changed throughout development [45]. Figure 9 shows the process where requirements are screened.

![Figure 9: The requirements engineering process from a requirements selection point of view](image)

**Elicitation** phase includes collecting requirements, as described in the previous section. **Screening** is performed to quickly remove the requirements that are rather obvious to omit. It is a major challenge to prevent the requirements repository from being flooded with requirements [65] so screening is important. **Evaluation** activity includes prioritization and identification of independent requirements. During **development** further knowledge might be gained, which may postpone or cancel requirements chosen during the previous phases.

Previous research [46] noticed that companies in practice use quite different methods for release planning. Some use intuitive prioritization like top 10 lists to select the most important requirements.
and some use the approach of the Planning game as described in section 3.1.4. Another method to prioritize requirements is pair-wise comparison as described in section 3.1.4. Experiments by Karlsson [45] showed that Planning game and pair-wise comparison both can be accurate, but pair-wise comparison needs to be supported by a tool to be efficient.

If the success of requirements selection is measurable, it allows for continuous process improvement [30]. The PARSEQ-method proposed by Karlsson et al [45] focuses on post-release analysis of requirements selection quality. The method is based on an in-depth analysis of requirements selection decision outcomes after the release has been launched. Non-functional requirements are of major importance in MDRE and for this reason Jacobs et al [40] created a model called QUPER (QUality PERformance) to add quality as a dimension in addition to cost and value traditionally used in prioritization of functional requirements.
3.3 Requirements Abstraction Model

Requirements Abstraction Model (RAM) is an empirically developed model that helps structuring and specifying elicited requirements in market driven requirements engineering [32]. Traditional elicitation techniques involve stakeholders with very different background and domain knowledge. It can be everything from end users and marketing teams to product managers and developers. This results in elicited requirements being very non-homogenous in terms of span and structure.

3.3.1 Abstraction levels

To get structure from this set of requirements RAM specifies a number of abstraction levels. In the study by Gorschek the number of abstraction levels are four, but they state that this is tailored for their specific case study and can be variable. When implementing for a specific organisation, more or fewer levels can be used. The four level model can be seen in figure 10. The Product level represents the most abstract requirements. These actually do not fulfill the normal definition of requirements like being testable and unambiguous, but are of a more goal-like composition representing the product interface towards overall product strategies, organizational strategies and future vision of goals. Feature level requirements explain what features should be included in the system, without going into more details of how the feature are to work in practice. The Function level describes what the user is able to do with the system and the different features. This is a very detailed level where requirements should be testable and unambiguous, and should be good enough to be handed to a system designer for sketching and implementation. Component level requirements are often as detailed as a beginning of a solution to the problems on function level. These kind of requirements usually come from internal sources or as breakdowns of complex functional requirements.

Organizational Strategies
Product Strategies

RAM - Abstraction Levels

Product Level (goal)
Feature Level (features)
Function Level (functions/actions)
Component Level (details - consists of)

Figure 10: The four abstraction levels in RAM [32]

3.3.2 Action steps

RAM has three specified action steps (figure 11) to achieve the right requirements specification based on the abstraction levels. These steps are iterated for all requirements in the system. First of all, the requirement should be specified (elicited). This is the first sketch of an idea about what should be included in the system. Here the four most general attributes to the requirement - including [32] description, reason/benefit/rationale, restrictions/risks and title - are specified. After this specification the requirement is placed (evaluated) according to the heart of RAM model, the abstraction levels. Placing means to determine abstraction level and this can be done with assistance of the guides provided by Gorschek (see appendix C). It should be noted that the initial placement does not have
to be the final one. The next step is abstraction (work-up). All specified requirements are now abstracted and broken down to fit into the other abstraction levels. This means that new requirements are created with connections to the original ones, but with another abstraction. It could also mean connections to other existing requirements. The goal is to be able to compare all requirements with product strategies to be sure that the product developed is on the right track and meets customer goals. For comparison to be possible, they all have to have the same abstraction level. Gorschek specifies this as a first work-up rule:

\[ R1: \text{No requirement may exist without having a connection to the product level.} \]

This rule can be met if the original requirement itself was on that abstraction level, or if a new one was created in the work-up step. At the same time he also specifies the second work-up rule:

\[ R2: \text{All requirements have to be broken down to function level.} \]

This is important since it has to be possible to verify all requirements, and on this level they are unambiguous and testable. As a result of this work-up, we now have chains of requirements where all satisfy the traceability between business and product goals down to verification and testing.

![Figure 11: RAM action steps](image)

3.3.3 Attributes

In the process of specifying, placing and working-up the requirements, additional attributes are added. These include the two groups traceability and process attributes for fitting into RAM. The traceability attributes suggested by RAM are Requirement source, Requirement owner, Requirements manager and Relation/dependency. The process attributes include State, Reject reason (for rejected requirements only), Due date, Version, Date of creation and Last changed. Many of the attributes are the same or almost the same as in traditional RE with some anomalies. Then there are two RAM specific attributes for Original (is it an original or worked-up requirement) and Level (what abstraction level). RAM has been implemented at two companies with good result [31], but tailoring this overall method
takes some effort in deciding abstraction levels, requirements attributes and role management to fit into the specific company. At the two companies this was done in workshops with different roles within the companies. These decided the structure of the tailoring based on real in-project requirements and decided what attributes that were the most important and what levels to use. The hard thing about the attributes was to scale down the number of wanted attributes to ease up the use of the model. As few attributes as possible were set as mandatory and extra ones set to optional.
3.4 Software testing

This section is intended to be a very brief overview of the principles and basic concepts of software testing in theory.

In the "Living Glossary of Testing Terms" [75], the definition of testing is The process of exercising software to verify that it satisfies specified requirements and to detect errors. The first part of this sentence describes the value of checking the system against the specification, while the last part explicitly expresses the error detection as a big part of the testing.

Testing is often mentioned as an important part of Quality Assurance (QA) of software. According to IEEE [41], there are two definitions of software quality:

• Quality relates to the degree to which a system, system component, or process meets specified requirements.

• Quality relates to the degree to which a system, system component, or process meets customer or user needs, or expectations.

With these two different approaches, testing is seen as a short description for Verification (the first) and Validation (the second) of the software quality. We will describe it in further detail below.

The process of testing is often thought of as the last phase of a defined software development process [15]. There are many views of testing that differ from this, like the QA approach which means that the product as a whole should be finished by all means before testing. On the other hand, testing activity can be that of a continuous process of automated unit-testing during continuous integration processes in for example nightly builds of the product. This wide spectrum can also be stretched to include validation of the requirements (section 3.1.3) and/or code reviews.

3.4.1 Basic testing principles

For the basic concept of system testing, regardless of activity Burnstein [15] describes 11 principles that summarize the process of testing in a good way:

• Testing is the process of exercising a software component using a selected set of test cases, with the intent of (i) revealing defects, and (ii) evaluating quality. The fact that software contains defects is the key to all testing; it is just a matter of revealing them in good time before launch to be able to decide if the quality of the software is good enough.

• When the test objective is to detect defects, then a good test case is one that has a high probability of revealing a yet-undetected defect(s). Since the foundation of testing is the presence of defects, the testing approach should aim at possible weaknesses and probable defect areas of the system.

• Test results should be inspected meticulously. The fact that test cases also contain defects makes it important to interpret all results, even if passed, to avoid extensive rework at a later stage. If failure occurs, both software and test cases should be examined.

• A test case must contain the expected output or result. While the input is obvious in a test case, it is of no value if the output cannot be evaluated according to an explicitly stated criteria. This is the essence of the pass/fail status of the test and should be included at all times.
• **Test cases should be developed for both valid and invalid input conditions.** While valid input conditions are essential for proving working software, the robustness can be shown with invalid inputs. Invalid inputs should be expected from both human users as well as other system integrations, because the risk of getting typos or unexpected failure outputs respectively is possible at all times.

• **The probability of the existence of additional defects in a software component is proportional to the number of defects already detected in that component.** The existence of defects are often clustered, with more defects in high complexity components than ones with lower complexity. If a component has too many defects, a decision of re-factoring could be better than bug-fixing, since there could be many more bugs left.

• **Testing should be carried out by a group that is independent of the development group.** This means that the particular developer should not test his own code, since he has pride in his work and might be biased about creating invalid inputs as defined above. The testing should also test if developers have any misconceptions about the specification.

• **Tests should be repeatable and reusable.** Defects found that are not reproducible are not defects. This is why tests preferably are step by step instructions and provide recordable results, that can be followed by anyone wanting to see the defect. If the developer cannot reproduce the defect, he will have a harder time finding the reason for it.

• **Testing should be planned.** Without planned testing activities, it can be hard to push them into an already planned development cycle. With planning and management of testing, it can be monitored and measured, creating better visualization of the quality.

• **Testing activities should be integrated into the software life cycle.** It is more and more important to involve testing activities as early in the development as possible, preferably in the requirements analysis phase of the project. Test planning and usability testing are activities that can be used before parts of the system is implemented.

• **Testing is a creative and challenging task.** There are many difficulties included in a testers situation of work, such as different ways of thinking and being creative and at the same time have a good eye for details.

### 3.4.2 Testing approaches

With the above principles in mind, there is a great deal of approaches to testing that focuses more on some of the principles and less on others. What approaches to choose is up to the test planning to decide for the specific product. We will describe some concepts that are essential for testing in general [15].

#### Levels of testing

Independent of how big, complex or critical the system is, it needs testing in some form. Usually this is done in different levels, testing different things in the system. A good approach to this is described as the V-model in figure [12]. **Unit testing** is performed on single components in the system. Exactly what these components are, classes and methods, procedures and functions or procedure-sized reusable components, is decided by the test team. The important thing is that the unit is the smallest testable component in the system. **Integration test** is preferably a continuous process of testing the interfaces between components. This is done when components are done and integrated with each other, but before the whole system is completed. **System test** is performed on all parts of the complete system and is a very resource demanding task. Planning of this test phase is essential to assure cost-efficient and complete testing, where external hardware and software interface defects can be found.
An *Acceptance test* is usually a light version of the system test where the end users, or representatives for the end users, evaluate the system according to the original ideas and requirements on the system functionality and business values. A *Regression test* is performed when any changes have been made in the code as a result of bug fixing or change requests. This type of testing is not scheduled into the V-model, but is of essential importance for developing a higher quality system.

![V-model Diagram](image-url)

*Figure 12: V-model [28]*
Practical testing

Depending on the nature of the system and the level of testing, test cases can be developed in different ways. Burnstein [15] mentions two main strategies of testing, black box or functional/specification testing and white box or clear/glassbox testing. Black box testing uses the system as a box with predefined output values for every input value. Because of this nature, this strategy usually mostly reveals requirements or specification defects which are the main source of information for these tests. The white box strategy focuses on the inner structure of the system, looking at the high level design and flow graphs to derive test cases. The extensive focus on the code makes analysing and running white box tests a resource demanding activity, which makes it usually carried out in small pieces of the system, running unit tests and/or integration tests. Both strategies have several different methods and approaches for developing test cases as seen in figure 13, but this is out of the scope of this thesis. Although the reader should understand that these methods generate a vast amount of test cases and it is of great importance to evaluate the cost of extensive testing to the value it generates. One way of dealing with this problem is to automate the test cases if possible to enable faster test execution and coverage. Another very important issue in software testing is to decide when testing is supposed to stop. It is usually decided by recording the executed tests and validated towards found defects with statistics [15].

<table>
<thead>
<tr>
<th>Test Strategy</th>
<th>Tester’s View</th>
<th>Knowledge Sources</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black box</td>
<td>Inputs</td>
<td>Requirements</td>
<td>Equivalence class</td>
</tr>
<tr>
<td></td>
<td></td>
<td>document</td>
<td>partitioning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specifications</td>
<td>Boundary value analysis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Domain knowledge</td>
<td>State transition testing</td>
</tr>
<tr>
<td>White box</td>
<td></td>
<td>Defect analysis</td>
<td>Cause and effect graphing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>data</td>
<td>Error guessing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>High level design</td>
<td>Statement testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Detailed design</td>
<td>Branch testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Control flow graphs</td>
<td>Path testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cyclomatic</td>
<td>Data flow testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>complexity</td>
<td>Mutation testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Loop testing</td>
</tr>
</tbody>
</table>

Figure 13: The two basic testing strategies [15]

3.5 Software development processes

To be able to understand the software development process, we first have to define our view of what a process is in general. Ljungberg et al [49] describe a process as a repetitive network of ordered activities that use information and resources to transform ‘object in’ to ‘object out’, from identification to satisfaction of customer needs. This definition is displayed in figure 14. The input object enters the process where it is processed under the influence of internal resources and external information. The activities in the process produces the output object.

This is a very general definition and can be applied to any kind of development. The specification of in and out as customer needs and satisfaction is the core of this statement. Without these customer values, the existence of the whole process can be questioned. The identification of the needs trigger a set of activities which in a specific order will create satisfaction. Since the project itself is the execution of these activities, a process description in this perspective equals to the instructions followed by the organisation to achieve satisfaction. This repeatability is the characteristic of a process. While the
projekt is a time-boxed activity, it dies or gets finished, the process description is eternal and can be reused in another project. This is why it is of great importance to analyse processes and improve them over time.

Since the process itself does not contain any resources, but will obtain these from the project using the process, it is of great importance that the process matches project structures and needs [49]. If it does not fit into the organization, there could be reluctance within the personnel using it, resulting in a chaos of a non-complete process in the project. Depending on the characteristics of the project and/or team, this could be devastating to the product being developed concerning quality and deadlines.

The business area of software development is comparably new in the sense of producing products to a market. Customer satisfaction is created differently compared to other manufacturing industries. When the software industry has grown bigger, there has been a need for process practices within organisations [26]. Especially the US department of defense has been a pioneer in demanding defined process information from its contractors. This is supposed to be a measurement of product development quality [26] [39].

These early process model descriptions are called defined processes. This means that every step of the process is described in detail in the process model. While this is a realistic way of describing an industrial process, it does not really fit software development because of its exploratory nature. Schwaber [72] describes software development as a complex process which cannot be described in such detail that it, when executed two times in a row, creates the same result both times. This is why many software vendors have started to implement empirical process models in their organizations. Below we describe and give examples of the two opposites in detail.

3.5.1 Defined processes

Schwaber [72] describes the defined process models as manuscripts of how to develop software. The most common traditional process model is the waterfall model [68]. Another famous method is the Cleanroom [52] model that was developed at IBM in the 1980s. These models are described below.

Waterfall model
This model was first published 1970 by Royce [68]. It was derived from other existing engineering processes to fit to the software development. The name waterfall describes the overall characteristics
of the model where the activities are executed in a sequential order one after the other. The main activities specified in the model are as shown in figure 15 and described below:

Requirements analysis and definition is where the requirements are elicited and specified according to traditional requirements engineering practices in consultation with stakeholders of the product.

System and software design based on the requirements. High level models and descriptions of the system architecture are created and visualized.

Implementation is the stage where high level design documents are turned into functional source code. The source is compiled into runnable machine code.

Integration and system testing makes up verification where the small modules are integrated and tested as a whole system. Testing is made against the specification. When testing is finished, the system is shipped to the customer.

Operation and maintenance is usually the longest phase in a time perspective. The customer installs the system and starts to use it. Maintenance covers correcting bugs not found during testing and improvement coding.

![Figure 15: Traditional waterfall model, adapted from [77]](image)

Cleanroom
This model was developed by Harlan Mills and his colleagues at IBM in the late 1980s. The name comes from the electronics industry, where a clean environment is used to prevent defects in hardware. This is the core of the process model as well, preventing defects instead of removing them. To support this, three principles are defined on SEI webpage [74]:

Software development based on formal methods usually means that mathematical principles are used to control that the product developed meets the specification. The box structure method is used in the specification and design and can therefore be used when verifying the functional parts of the system against the requirements. No code execution is made before verification in a team review.

Incremental implementation under statistical quality control means by doing incremental development, the process model provides statistical quality control to the product. Every increment is evaluated according to certain standards to see if the process itself is under control. If the standards are not met, development will return from testing to design state.

Statistically sound testing where the testing is made on a sample of the whole system. Analysing the tested part, statistical analysis is made to evaluate the whole system. In other words, the whole system is not tested like in other process models.
3.5.2 Empirical processes

Schwaber \[72\] explains the difficulties with the defined process control models, and while developing Scrum, he presents his ideas within the chemical industry where he gets input on the empirical process control models used within that field. The processes here are of the same nature as software development and needs frequent adaptation to the results.

This is emphasized with Schwaber \[72\] describing the project complexity in figure 16 that shows the noise level in any systems development project. The axes describe the certainty of technology used and requirements in the project. For example, with high certainty on both attributes gives a simple project. With only any one of the attributes uncertain the project gets complex, while having both axes uncertain gives almost an impossible project. Schwaber also states that the factor of people is left out, which would give more complexity if inserted.

The main difference between the defined and the empirical process models is the lack of isolated activities like specification, development and validation \[77\]. Instead of having them separated, empirical process models have the activities run incremental in parallel with rapid feedback between increments. This approach to software development creates a knowledge within the organization about the complex situation described by Schwaber, and promotes expecting the unexpected in the development. By using frequent feedback and adapting to the results, the empirical development models handle unpredictable results in predictable ways.

![Figure 16: Project complexity](image)

Sommerville \[77\] describes these empirical development models as evolutionary development and gives two examples of this type of approach:
• *Exploratory development* where a close dialogue with the customer is essential. By starting with what is already known about the product and let it evolve according to customer requests the requirements are explored deeper and deeper over time.

• *Throw-away prototyping* is about experimenting with the unknown requirements, while the known are kept unchanged. Showing the customer versions of the system with different features that are later thrown away if not accepted.

Some examples of this kind of empirical approaches to software development is covered by Agile methods and Scrum that are explained in detail below.
3.6 Software process improvement

The above mentioned software process models are only a scratch on the surface of theory behind these complex matters. The area of improving such activities would of course be judged impossible at first sight. This is explained by Caputo [16]: 

“To perform software process improvement, you need models and experience. Reading about it isn’t enough to learn how to do it. You learn by practicing.”

While Khaled El Emam [26] states The SW-CMM is Common-sense engineering, Ljungberg and Larsson [49] point out that Common sense is not so common when talking about process improvement. These two statements might give even more understanding that process improvement is a complex matter.

When explaining why you should improve your software process, Wiegers [83] states:

- To correct problems you have encountered on previous or current projects.
- To anticipate and prevent problems you might encounter on future projects.

When planning improvements of already existing software processes, you have to consider the known facts and premises of the projects, staff and organisational possibilities [49]. Sommerville [77] mentions some key stages in a process improvement process:

- Process analysis involving analysing the existing process used, for present understanding. If possible this analysis can be made quantitative with measurements. This way it is objectively comparable with the result of the improvements.
- Improvement identification is done according to the results of the analysis concerning process quality, cost or schedule bottlenecks that have impact on the product quality. New procedures or tools can be suggested in order to remove these bottlenecks.
- Process change introduction is where the new procedures and tools are put into the organisation and integrating with the rest of the processes and resources. This activity requires time for the new procedures to be adapted into the organization. It is also of essential value to have the changes approved with upper management [49].
- Process change training is very important for the improvements to stay and be accepted by the organisations. Both engineers or upper management would without proper training to the new procedures, ignore the changes, with a complete mess as a result. This way the improvements would become impediments.
- Change tuning for adapting and tailoring the new procedures to fit the process is needed. No improvements are perfectly effective when introduced, and minor changes could improve the improvements.

Ljungberg and Larsson [49] summarize many years of publications about the re-engineering process (process improvement process) and converges this into four stages, which are quite similar to Sommervilles key stages. The first two activities are not very hard to achieve, whereas the last two are really challenging to any organisation.

- Mobilize (“get serious”). The understanding of the need for process improvement within the organization.
• **Analyze (“get focused”).** Create understanding for the current process and identify needs and strengths for the improvement.

• **Reconstruct (“get crazy”).** Search for completely new ideas and build a new process model on these ideas.

• **Realize (“get real”).** Develop a plan and strategy to implement the new process.

When performing process improvement it is very important not to stop halfways, as figure 17 shows. Productivity can, for any process improvement, be predicted to be reduced initially, and then risen to a new all time high after a certain amount of time. Process change training, as described by Sommerville [77], is crucial to not get stuck in the low level of productivity.

![Process improvement learning curve](image)

3.6.1 Process and product quality

When talking about process quality, Sommerville [77] mentions a number of characteristics that are of special interest. These are listed in table 8 and can be used to validate process quality.

Although much of process improvement is about adopting certain methods and processes, they always have to be tailored into the organisation, with more focus on some of the characteristics rather than others. The saying that *product quality relies on the quality of the process models* has been thoroughly investigated in many related articles [24] that mostly conclude the relation is of essential matter. This applies more to products developed in largescale organisations or where the product does not rely too much on human innovation and intellect. Sommerville [77] extends the product quality to be a mixture of the four factors presented in figure 18: process quality, development technology, people quality and cost, time and schedule.

The weight of influence of each factor is very much depending on the product and the organizational structure surrounding it. When working with very large systems the process quality is the most important, but when looking at small and medium sized projects, the people skills and experience is much more important. Development technology is more important in small and medium sized projects, since much of the time of people in larger projects goes to understanding other parts of the system, giving less time doing actual development activities. The forth factor of time and money is the most critical factor in projects, and often the factor to undermine the whole structure of
<table>
<thead>
<tr>
<th>Process characteristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understandability</td>
<td>To what extent is the process explicitly defined and how easy is it to understand the process definition?</td>
</tr>
<tr>
<td>Visibility</td>
<td>Do the process activities culminate in clear results so that the progress of the process is externally visible?</td>
</tr>
<tr>
<td>Supportability</td>
<td>To what extent can the process activities be supported by CASE tools?</td>
</tr>
<tr>
<td>Acceptability</td>
<td>Is the defined process acceptable to and usable by the engineers responsible for producing the software product?</td>
</tr>
<tr>
<td>Reliability</td>
<td>Is the process designed in such a way that process errors are avoided or trapped before they result in product errors?</td>
</tr>
<tr>
<td>Robustness</td>
<td>Can the process continue in spite of unexpected problems?</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Can the process evolve to reflect changing organizational requirements or identified process improvements?</td>
</tr>
<tr>
<td>Rapidity</td>
<td>How fast can the process of delivering a system from a given specification be completed?</td>
</tr>
</tbody>
</table>

Table 8: Process characteristics [77]

Figure 18: Principal software product quality [77]

development. Without time and money there is no possibility to achieve any product quality or even product for that matter. This matter is described more detailed in section 3.7.4.

3.6.2 Process measurement

To be able to evaluate process improvement in an objective way, some measurements need to be done. Ljungberg and Larsson [49] describe the process measurement as a complex thing to do, since you not only should measure the right thing, but also in the right way. Specifically within software processes, Humphrey [38] describes three classes of metrics to be collected:

- *The time taken for a particular process to be completed*. Any time metric involving specific people in the project, calendar time or process time.
- *The resources required for a particular process*. Total effort in work-days, travel costs or equipment resources are some examples.
The number of occurrences of a particular event. This could be number of defects found during inspections, requirements changes or lines of code to change for bugs.

Basili and Rombach [9] developed another method to know what to measure in a process, Goal-Question-Metric (GQM) paradigm. This method was later combined into the SEI’s Capability Maturity Model described below.

3.6.3 Defined process improvement models

The most comprehensive formal model for software process improvement is called the Capability Maturity Model for Software (CMM) [60]. Assessors outside of the company investigate many areas of software development, management and related activities - not only requirements engineering. CMM is based on experience from successful organizations put down as guidelines how others can improve their work [16]. There are five capability maturity levels that describe how well a company can predict the results of following the process:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initial</td>
</tr>
<tr>
<td>2</td>
<td>Repeatable</td>
</tr>
<tr>
<td>3</td>
<td>Defined</td>
</tr>
<tr>
<td>4</td>
<td>Managed</td>
</tr>
<tr>
<td>5</td>
<td>Optimizing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unpredictable and poorly controlled.</td>
</tr>
<tr>
<td>2</td>
<td>Can repeat previously mastered tasks.</td>
</tr>
<tr>
<td>3</td>
<td>Process characterized and fairly well understood.</td>
</tr>
<tr>
<td>4</td>
<td>Process measured and controlled.</td>
</tr>
<tr>
<td>5</td>
<td>Focus on process improvement.</td>
</tr>
</tbody>
</table>

Table 9: Maturity levels in CMM [16]

Each increase in level can be said to represent the degree of conscious effort to produce results and how this effort is managed and controlled. It also includes the degree of involvement from everyone in the organization to produce and manage this effort [16].

CMM has been superseded by Capability Maturity Model Integration (CMMI) [73] where organizations can determine how well they conform to best practices of CMMI. It is a collection of best practices and a framework for organizing and prioritizing activities. There is also support for coordination of multi-disciplined activities. CMMI emphasize on aligning process improvement goals with business objectives. For requirements, the R-CMMI has been developed [12].

Wiegers [83] states that process improvement should be evolutionary, continuous and cyclical, with small steps at a time and not expecting all process changes to be adopted at the same time. People in the organization change only if they have incentive to do so. Later [83] he presents a cyclic definition of four steps slightly different from Sommervilles steps, figure 19. It is a goal oriented improvement model which handles changes as individual projects.

3.6.4 Empirical process improvement models

The general idea with empirical processes, as stated above, is to adapt the development to how the iteration was experienced. Because of this exploratory nature of processes, it usually means involving the process improvement in the process itself. This is described by Schwaber [72] in figure 20 where the process model shown is based on the Scrum methodology, which is explained further below. In short, this is how empirical processes are improved. The output represents not only the product developed and customer satisfaction as described in section 3.5, but also feedback from last iteration of the process. This feedback is analysed and results are used in the next process iteration. The solutions could handle process adaptation to the product development or, the other way around,
with product development adapting to the process. This picture can be compared with figure [14] which shows the definition of a process model. The empirical process improvement hence takes a standard process and iteratively evolves it by using the output as input to the next iteration. This was evaluated at Google when they enforced a process to be followed in an organization usually without outspoken processes [80].

### 3.6.5 Requirements Engineering process improvement

**Assess current practices**
Following the fundamental process improvement schemes described above the first step to improve a requirements engineering process is to assess current practices. The objective is to locate which part of the process is in most need of improvement [83]. A questionnaire can be used where participants cover different areas by marking which answer is most similar to how work is currently carried out. One example questionnaire question can be seen in table [10]. As intuitively can be seen, the early replies might indicate a problem in this area. Other ways to achieve current process knowledge are for example interviews or observations.

**Plan improvement actions**
When problematic areas have been identified, the next step is to plan improvement actions. The plan to improve requirements process is often part of a larger strategic plan on how to improve overall software processes. The plan should include goals for all improvement activities as well as participants and assigned individuals. The plan should be kept rather simple and not contain more than 10 action items [83].
3. **How do you obtain voice-of-the-customer input?**
   a. Development already knows what to build.
   b. Marketing feels they can provide the user perspective.
   c. Focus groups of typical users are surveyed or interviewed.
   d. Specific individuals who represent different user classes participate on the project, with agreed-upon responsibilities and authority.

Table 10: Example question from requirements process survey, extracted from [83]

**Pilot creation and process implementation**
The hard part of requirements process improvement is to actually implement the action plan. Since it is unlikely that the new process will be perfect the first time a good idea is to first create a *pilot* for most new procedures or document templates. The pilot gives opportunity to gain additional knowledge that can be used to make adjustments that will improve chances of success when the changes are rolled-out.

**Result evaluation**
The last step is to evaluate activities and results. The most important part is to conclude if the newly implemented procedures are yielding the desired results. This can be checked against goals stated in the process improvement plan. One very important aspect is that of the *learning curve* (see figure 17 in section 3.6.4). It might be that evaluation of the results shows the new process does not yield satisfactory results. This could be an indicator that the personnel currently are at the productivity drop, because not enough time has passed to really understand the new process. The short-term productivity loss is a part of investment in the process improvement and it is very important to not be discouraged and abort the process.
3.7 Agile development

Agile development is a set of frameworks for developing software in shorter life-cycle projects in contrast to the traditional waterfall models. Agile methods have their foundation in the Agile manifesto (figure 21) created in 2001 by prominent software developers that later formed the Agile Alliance. The Agile manifesto points out the heart of Agile thinking, which is individuals, working software, customer collaboration and being able to adapt and respond to change. The manifesto still recognizes the value of processes, tools, documentation, contracts and plans in some situations - but the left side of the figure contains what is really important.

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Figure 21: The Agile manifesto

3.7.1 Principles and practices

These values have evolved into the following 12 principles:

- **Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.** By having less functionality in an early delivery of a system and a continuous delivery of functional software, the final release is often of higher quality. Using these early deliveries, the customer has a good overview of the development, and can tell if the product developed is the right product early on.

- **Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.** It is an attitude statement in the Agile team to see changing requirements as a good thing. Whenever requirements change it is because the team has realized another angle to the problem and will be developing better and more competitive software for the customer.

- **Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.** The team focuses on delivering software instead of documents. This software should be working to be worth delivering. It is prefered to deliver small, but working, parts of the system instead of one large delivery, and instead deliver more often.

- **Business people and developers must work together daily throughout the project.** All stakeholders should have frequent interaction and communication to handle the turns of the project.
• **Build projects around motivated individuals.** Give them the environment and support they need, and trust them to get the job done. This is one of the most important factors for an Agile project to be successful. Any obstacles that prevent the individuals in the team to achieve working software should be taken care of. This involves for example office environment and process implementations.

• **The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.** The most important way of communication in an Agile project is to talk to your team members. Written documentation is avoided if not really necessary.

• **Working software is the primary measure of progress.** The progress in an Agile project is measured in functionality, nothing else, like documentation created or time passed. When the software is working, the project is done.

• **Agile processes promote sustainable development.** The sponsors, developers, and users should be able to maintain a constant pace indefinitely. The work environment should not enforce overtime, but rather encourage team members to work efficiently, but in a sustainable pace to prevent burnouts.

• **Continuous attention to technical excellence and good design enhances agility.** Keeping the code clean is essential for Agile teams. High quality code is expected from all team members that are required to clean up any mess before the end of a working day.

• **Simplicity—the art of maximizing the amount of work not done—is essential.** The team is not supposed worry about tomorrow’s problem, nor try to solve them before the problems arise. Today’s problems are solved in the best and high-quality way as possible.

• **The best architectures, requirements, and designs emerge from self-organizing teams.** This means no team member is responsible for certain areas within the project. No tasks are given to any individual in the team either, but to the team as a unit. Then the team decides itself how to achieve the tasks in the best way.

• **At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.** The organization and rules of the team has to be adjusted over time for the team to stay Agile within the changing environment.

These principles require certain prerequisites on the project team and management. For example most Agile teams need to be relatively small and based on individual skills to adapt, and the project has to be adapted to the Agile methods as well. This is mostly done by breaking down big tasks into subtasks [72].

There are many different Agile methods that adopt the principles in different ways, but still there are some main characteristics that are the same according to the principles [35]. Mostly the development is iterative, meaning short iterations delivering working software after each iteration. Each iteration, on the other hand, can consist of all or some of the process components of the normal waterfall model. Since the developed software is working and already has been quality assured within the iteration no, or very little, documentation about it is created. Instead this information is transferred within the team with face-to-face communication. To be more secure about delivering the right software, a close contact with the customer is essential [51]. This is also stated as one of the fifteen key principles within software engineering as a whole by Davis [23]. The main advantage with small iterations is that it minimizes risk by developing small pieces of software that potentially can ship when the customer is satisfied. It also decreases time of feedback cycles.

Compared to other iterative development methods, Agile methods measure their project frames in weeks rather than months or years. The time frames are also strictly time boxed. Some Agile methods implement smaller versions of the waterfall model within these time frames, while others stay
Agile with for example methods like eXtreme Programming [10] or practices such as Test Driven Development [54]. When comparing with the waterfall, we see the difference in that waterfall model is predictive while Agile methods are reactive. Dealing with changing requirements over time forces the process to adapt to the changes. In that perspective reactive methods fit best.

There are many different approaches to these Agile principles, described in own process descriptions. Some of these are Scrum [72], eXtreme Programming [10], Crystal [77], Feature Driven Development [59], Test Driven Development [54] and Adaptive Software Development [36]. We will further describe the model Scrum.

3.7.2 Requirements Engineering for Agile Methods

There are many differences how requirements are handled in traditional RE and Agile Methods (AMs). Some of them are that AMs do not use formalized standards (such as suggested by IEEE) for eliciting and managing requirements as well as letting the whole development team be involved in elicitation. The ability to react to changes is, as described in the previous section, fundamental in AMs and requirements variability is a constant problem in almost all software projects [76]. That is one of the major advantages of AMs. One drawback is that the size of the problem has to be limited, or the team will not be able to elicit requirements, nor have close interaction with the customer.

The customer is a very important role in AMs. One single person takes the role of customer and is available on-site to answer any questions the developers might have. Since the customer represents all stakeholders he must have complete knowledge to be able to give good answers. The customer is in this aspect a domain expert that knows how the application should work and how in/ out data should be handled. The customer also has decision power and is capable of making decisions and commitments. Changes in requirements, acceptance of implemented features and prioritization are all handled by the customer. Obviously, having access to a customer with all these qualifications can be difficult [58] since the task requires comprehensive knowledge about engineering, marketing and business processes.

One aspect of AMs is to reduce waste, and that only things that create business value should be worked on. This originates from Lean development [62] and ultimately the manufacturing process of Toyota Auto Company [56]. Waste always produces more waste [62], that is why it is very important to get rid of waste amongst the requirements as soon as possible. Reduction of waste from requirements is one important activity in AMs since it deeply influences the development process and the ability to deliver a satisfying product. Sillitti and Succi [76] list the following effects of waste amongst requirements:

- More source code to write and higher cost
- Increased complexity of the source code
- Delayed delivery of the final version of the application with all functionalities
- More complex and costly maintenance
- More resources required by the application including: memory usage, processing power, network usage, etc
- Increased complexity of the application from the customers point of view (e.g. more complex user interface, more effort to learn how to use the application, etc)
- Savings produced by the application in the production process of the customer are delayed
To reduce the waste, Sillitti and Succi [76] describe a number of techniques used in AMs:

The whole team gathers requirements from the customers, which in effect spreads knowledge and the probability of misunderstandings decreases. Requirements are elicited using a common language, which is the language of the customer and not a formal language. This way, developers gain valuable domain knowledge. Interactions between customer and team are direct, which means there is no man in the middle. This reduces the number of formal documents and leads to less misunderstandings. If the team considers a requirement too complex, it is possible to split up requirements into simpler ones. This helps developers to understand the functionality wanted by the customer. It should be noted that the previously described techniques do not scale and is only feasible if the size of the development team is limited. Sometimes customers over-specify and include a wide range of detailed features that are not providing real value to their business. Such requirements are useless and a source of waste. Two techniques to handle this is to prioritize requirements or use incremental releases. Releasing functionality in small but frequent packages helps collect feedback from the customer. Elicitation is repeated in each incremental release and this way it is possible to identify requirements that do not provide enough value for the customer.

Another important activity in RE for AMs is managing the requirements evolution [76]. It is often assumed that it is very hard to elicit all requirements at the beginning of the project and they change over time as the customer changes his mind or technical obstacles arise. Embracing change is a fundamental Agile concept and as a consequence requirements collection and management are based on these three hypothesis [76]:

- Requirements are not well known at the beginning of the project
- Requirements are subject to changes
- Making changes is not expensive

![Figure 22: The cost of change in Agile (eXtreme Programming) software development](image)

This contradicts the traditional view that the cost of implementing changes grows exponentially over time like described by Davis [22]. Instead, as can be seen in figure [22], the cost for implementing changes is almost constant over time in AMs, because of the iterative process and because binding decisions are taken as late as possible. This is not always true in every context, but nevertheless a good guideline. A common procedure in Agile projects is to use scope-variable price contracts [76].
which means that requirements are not specified in details at contract level (see section 3.7.4). By decoupling requirements the order of implementation becomes irrelevant. The challenge is to identify which requirements are dependant of other requirements. Having independent requirements allows for more variability. Requirements elicitation and prioritization is performed at the beginning of every iteration, where new requirements are identified and analyzed. Often requirements are chosen for implementation or put "on hold", to (possibly) be included in a future release. Using this approach the project team is able to identify the most important requirements during the whole project, not only at the beginning [76].

3.7.3 Synergy between Requirements Engineering and Agile Methods

Paetsch et al [58] have investigated how requirements engineering and Agile software development can interact and found some common approaches, differences and synergy effects.

Customer involvement is important in AMs and for RE the same can be said with stakeholder involvement. Traditionally stakeholders are mostly present in the inception and elicitation phases, but with AMs they are present throughout the project as the customer.

The main elicitation technique used in Agile environments is interviews, since customer is always present and accessible. Chains of knowledge transfer increases with number of emissaries which in turn increases risk of misunderstandings, but in AMs this chain is kept to a minimum.

Prioritization is found in both RE and AMs. Agile prioritization changes during development, as further knowledge is gained and as new requirements are added. The highest priority features are implemented first, in order to deliver most business value.

Modeling in AMs is used to communicate understanding of a small part of the system. In contrast to traditional RE the models are often thrown away after usage, and simply drawn on a whiteboard or piece of paper. Models do not act as documentation for the system as they do in RE.

Documentation, in terms of creating complete and consistent requirements documents, is not seen as cost effective in AMs. Some methods, like Scrum, use a kind of documentation as part of the model, but it is up to the development team exactly how this is to be done and to which level of detail. The lack of documentation might cause long-term problems for Agile teams:

- **Documentation can be used to share knowledge** between people, and is particularly valuable for new team members. Asking other team members might slow down work if the project is complex.
- **Documentation reduces knowledge loss** when people leave the team. This is an important issue if software needs to be maintained over a long period of time.
- **Customers often ask for documentation to be produced.** It is often seen as a certification of quality towards the customer.

The other side of the coin is of course the lost productivity if the team has to spend too much time on writing documentation. It should also be noted, that with more comprehensive documentation further resources are drained on keeping it up to date. More documentation is important as team size increases since it is more likely to have to explain details many times, but most Agile teams are often kept small.

Validation is performed in AMs through frequent review meetings and acceptance tests. Review meetings let the customers try the software and experience how the system and implemented func-
tionality work. This allows for face-to-face contact and customers can ask the developers questions or maybe ask for changes in requirements. Customers can also run acceptance tests to validate system behaviour. Agile development can be compared to evolutionary prototyping in traditional RE since both use fast deliveries of working code to enhance feedback, but AMs have a deep focus on testing as well.

Management should be able to get overview of changes made to requirements, design or documentation and why those changes were made.

### 3.7.4 Agile contracts

Poppendieck [62] introduces her section on contracts by discussing the possibility of trust between professional firms and companies. The sole purpose of a contract is actually to take away the need for trust between business partners, that is contradict a positive atmosphere between firms by stating an outspoken confidence in the other party fulfilling what has been promised.

As discussed by Ambler [5] in his Agile newsletter, projects are constrained by the Iron Triangle (see figure 23). The scope reflects what users need, resources (or budget) is a financial issue and schedule is related to management. All three of these cannot be fixed, or the project is doomed to fail because of quality concerns. This will happen because there is no room to maneuver, except to adjust quality. The only situation where a project can succeed in this scenario, Ambler argues, is if scope is very small, budget very large and schedule long. Unfortunately most organizations desire the opposite situation. That means that in the Iron Triangle one of the three corner factors must be allowed to be variable.

![Iron Triangle of constraints](image)

**Figure 23: The Iron Triangle of constraints** [5]

#### Fixed price contracts

This is the traditional way of setting up an agreement between a customer and a supplier, designed to totally protect the customer. Governmental agencies and bigger corporations sometimes often have to use these contract forms and usually sign the contract with the lowest bidder. In the software industry this makes the vendor taking all risks, often causing him to strictly hold on to the initial requirements and make benefits on any change requests from the customer. The result ends up as high costs for the customer anyway, or a non-satisfied customer, like in the quote below.

"I ran a software development company which prided itself in not exceeding the price and schedule quoted at the beginning of an engagement. In a three-year period, we had 78 projects, and 77 of them were delivered on
time, on budget, and in scope. Then we surveyed the customers and found out that none of them was happy! The systems that we delivered did not solve their problems. Sure, we had protected ourselves by being on time, on budget, and in scope, but in doing this, we could not deliver what the customers really wanted. That is why I sold my business.” - Anonymous colleague to Poppendieck [62].

The dissatisfaction is often because the delivered system does not supply solutions for the real needs of the customer, but instead includes features that are not wanted. This is visualized in the Standish Group report [34] where the report shows that features never or rarely used in a system covers 64% of the whole system. This motivates some kind of ways to change the scope.

Schwaber [72] describes his experience with fixed-price contracts as kind of problematic. After long considerations he concluded that fixed-price contracts can be used together with Scrum, but the problem is the initial phase where requirements, design artifacts and architecture must be analyzed enough to provide time and cost estimates. This goes against the Scrum philosophy where the principle is “the art of the possible”, not “you give me what I paid for, when you said that you’d deliver it”. But even though the Scrum process would be “corrupted” in this way it is still possible to gain some advantages over competitors by using Scrum. The foremost of these is that the solution is delivered in iterations, where the customer can influence how the project evolves and decide to stop when enough business value has been added, even if there is additional “payed” time left for the project. There would be a penalty for stopping early, but it would be less than the cost of continuing until deadline and implement unnecessary requirements.

**Time and materials contracts**

The complete opposite of to the fixed-price, is a flexible-price contract that focuses on the time and expenses used for the project as a base for payment [61]. The risk is in this case completely on the customers side that has no control or safety in getting something delivered for the money spent. The ability to push change into the product at a later stage is easier however, giving the customer a better chance of getting a usable product that solves the problems needed in the end. Even if this type of contract also gives some risk to the supplier, they are good as long as they last. The risk is that the contract and earnings can stop whenever the customer wants, creating an environment with no incentives for the supplier to be efficient at work.

**Multi-stage contracts**

These contracts are also called progressive contracts [61]. They can be of two different characters. The first one is iterating fixed-price models in smaller steps, usually ending up in a bigger fixed-price contract if both parties are confident in the product after the smaller iterations. The other model is of completely iterative nature, fitting into Agile development. The biggest risk in this case is the uncertainty of both parties staying in the relationship.

**Target cost contracts**

Target cost contracts is a middle way between the fixed-price and the time-and-materials contract forms. This type of contract provides a base of trust between the companies and gives incentives for both parties to see to the best of the project. In the contract there is an outspoken target-price, but also rules about the consequences of not following this target. With a shared benefit if the project finishes earlier and a shared cost when late, it makes supplier and customer both take the risks.

Eckfeldt et al [25] has evaluated the target cost model in real situations at their company and found a way to integrate it into the Agile processes. They started this process by using a target-scope approach, similar to the target-price, but with a fixed price and the scope changing. During development, they were in close contact with the customer, using a model where the features all had time estimations. To get the incentives for both parties in the contract, when a feature was finished early, the customer could add features to the scope, and when something was late remove scope. The incentives were given in on a 50% basis, removing and adding scope was always 50% of the time

65
late/early. That is, if the supplier was 3 days early with a task, the customer could add 1.5 day of new scope and when 2 days late, only remove 1 day of scope.

After some time with the target-scope, the customer would get more and more confident in the supplier, being able to move to a target-cost solution. Eckfeldt describes the mechanics of their contracts. All stories are given time estimates according to their difficulty, 1/2, 1, 2, 3, 5 days of work for a development pair. Then some days are added for meetings, set-up, interface design and development and summed up as initial development time. On top of this a 10 - 25 % contingency cost is added depending on the clients and projects reliability. These hours are counted as the target-cost for the project. On top of this, a fixed profit is based on the development hours to become the real price for the customer. An example is seen in table 11. During development they strictly tracked all stories and their estimations and outcomes, and this summary was an essential tool for everyone when discussing changes in the scope.

<table>
<thead>
<tr>
<th>Line Item</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total estimated development days</td>
<td>50</td>
</tr>
<tr>
<td>Meetings</td>
<td>4</td>
</tr>
<tr>
<td>Setup</td>
<td>4</td>
</tr>
<tr>
<td>Interface design work</td>
<td>6</td>
</tr>
<tr>
<td>Day subtotal</td>
<td>60</td>
</tr>
<tr>
<td>Contingency/buffer (20%)</td>
<td>12</td>
</tr>
<tr>
<td>Total project days</td>
<td>72</td>
</tr>
<tr>
<td>Cost (developer/day)</td>
<td>$1000</td>
</tr>
</tbody>
</table>

| Total cost                       | $72,000  |
| Fixed profit (25%)               | $18,000  |
| Total estimated price            | $90,000  |

Table 11: Typical target-cost pricing structure [25]

When a change is to be made on the scope, it was categorized as fix, clarification or enhancements. Fixes and clarifications did not generate more profit, but were billed as additional hours. Enhancements were billed extra according to table 12 since these were different from the scope.

<table>
<thead>
<tr>
<th>Line Item</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total estimated development days</td>
<td>50</td>
</tr>
<tr>
<td>Additional development days</td>
<td>+5</td>
</tr>
<tr>
<td>Meetings</td>
<td>4</td>
</tr>
<tr>
<td>Additional meeting time</td>
<td>+1</td>
</tr>
<tr>
<td>Setup</td>
<td>4</td>
</tr>
<tr>
<td>Interface design work</td>
<td>6</td>
</tr>
<tr>
<td>Revised day subtotal</td>
<td>66</td>
</tr>
<tr>
<td>Contingency/buffer (20%)</td>
<td>13.2</td>
</tr>
<tr>
<td>Total project days</td>
<td>79.2</td>
</tr>
<tr>
<td>Cost (developer/day)</td>
<td>$1000</td>
</tr>
<tr>
<td>Revised total cost</td>
<td>$79,200</td>
</tr>
<tr>
<td>Revised fixed profit (25%)</td>
<td>$19,800</td>
</tr>
<tr>
<td>Original fixed profit (25%)</td>
<td>$18,000</td>
</tr>
<tr>
<td>Additional fixed profit to be billed</td>
<td>$1,800</td>
</tr>
</tbody>
</table>

Table 12: Calculation of a change [25]
For this approach to work in the best way, both client and vendor need to be a little risk adverse and trust the other partner some. Since both have their bacon on the line, this is also creating a good atmosphere for trust. The time tracking and reporting is also essential, since this is the subject of billing. It also creates a transparency towards the client if this summary is easy to understand. This kind of estimation and reporting is also an important part of any Agile methods, which is also why they work together.
3.8 Scrum

Scrum is a simple framework for an Agile software development process developed from empirical studies of how companies within the software development business actually worked with their daily projects in the 1980s and early 1990s. The approach builds on using small, cross-functional teams for the best and most efficient results to complex problems. The name Scrum comes from the rugby term that was used in the first articles about the method by Takeuchi and Nonaka [82]. It is a very easy to use framework that can be implemented within days, but takes some practice to master completely. The framework involves close contact between developers and customers for best result. The framework involves many different parts, which are of different importance within the process. Together these parts bring out the best of development teams.

Scrum follows some basic rules [70]:

- Always have a product that you theoretically can ship.
- Speak a common language on every single development site.
- Continuously test the product as you build it.

Key concepts or principles for Scrum development are [70]:

- Small working teams that maximize communication and sharing of informal knowledge, while minimizing overhead.
- Adaptability to changes for ensuring the best possible product to be produced.
- Frequent builds that can be inspected, adjusted, tested and documented.
- Partitioning of work and team assignments into clean, low coupling partitions.
- Constant testing and documentation of product as it is built.
- Ability to declare a product as done, when it is required.

3.8.1 Roles

Schwaber [72] defines two kinds of overall roles within the Scrum framework. These are called pig roles and chicken roles. The pigs are the ones that have **their bacon on the line**. In traditional RE these would be called the direct stakeholders. These include the team, Scrum master and Product owner. All other people that can be involved in the project are indirect stakeholders, like business managers, external experts and end customers, are called chickens.

**The Scrum team**

The Scrum team is usually a small group of people that are problem solvers and designers. The basic idea is that there are no set project roles, so everyone should be able to swap tasks with each other. Research [19] has shown that a group size of 5-9 (7 +/- 2) people works good. More, or less, than that is not viable. One key element of Scrum is that teams should be self organizing and Schwaber [71] goes as far as to call the team a *manager* in itself. The team will alone decide how to turn selected requirements into a functioning and shippable product.

**Product owner**

The voice of the customer is represented by the Product owner [19] who also manages things from a
business perspective. The main task is to administer the Product backlog, which is a current "TODO-list" that is prioritized according to profitability. The Product owner fulfills the Agile practice of customer in team. The Team and Product owner are brought together and synchronized by the Scrum master.

Scrum master
The Scrum master has the role of running the Scrum team [19]. It is not to be seen as a project manager, as it in many ways is fundamentally different. The authority of the Scrum master is for the most part indirect, and comes from knowledge about the Scrum process and how work is performed at the local development site. The Scrum master can be seen as a combination of coach, fixer and gatekeeper. He is responsible for the success of the project, and the major activities to ensure this is to help the Product owner select items from the Product backlog and to help the team to turn it into functionality. The Scrum master will also assist in improving team productivity in any way possible, like improving engineering practices, keeping information up-to-date, remove impediments from the team and keep the product in a state that is potentially shippable.

3.8.2 Scrum process

The Scrum project starts with a vision of what is to be developed. The vision can be seen as overall business goals. The Product owner is responsible for transforming the vision into something that will maximize return on investment. The Scrum process used by Softhouse [19] is shown in figure 24 and it starts with the product backlog, continues into sprint backlog, the sprint cycles and deliverable product. Each of these steps are here discussed in detail.

![Image](image.png)

Figure 24: The Scrum process, ©2008 Softhouse [19]
Product backlog

All requests and specifications for functionality or bug fixes are compiled by the Product owner and then broken down into segments. Each segment contains something that adds business value, and in itself is deliverable. The product backlog can be seen as a kind of requirements specification and contains functional and non-functional requirements. It can also include items required by the team, but only those that eventually bring value to the customer, like using an integration server to gain higher product quality. It should be noted that everyone can contribute to the Product backlog, but it can not contain low level tasks. The items in the log will be prioritized according to what is most likely to generate business value and this is also the responsibility of the Product owner. Scrum points out that items in the Product backlog are expected to change, and the log is constantly reviewed to make sure content, priorities and segments are up-to-date. When it is time to start a new sprint, the Product owner freezes the foremost items on the list and summons the team to a meeting.

Sprint backlog

The Sprint planning meeting should be a timeboxed meeting where the Product owner first presents the highest priority Product backlog items and the team can ask questions regarding content, purpose, meaning and intentions. When the Team has gained enough knowledge, they estimate the required time to develop these highest priority items, and decide what items from the backlog they can turn into functionality during the coming sprint based on these time estimations. The second part of the meeting is spent planning the sprint. The items that were selected from the Product backlog are put into another log called the Sprint backlog. This log is a highly detailed document containing information about how the team is going to implement the requirements. The most important issues to decide during the sprint planning meeting are:

- **Sprint goal.** The business goal to strive towards the next weeks. This goal does not have to be a product goal, but can also be about managerial or personal goals.

- **Sprint backlog.** Make sure that there is a sprint backlog to start working with when the sprint starts.

- **Demo date and time.** Decide exact time and date for the demo. This creates a motivation in the team to be ready at the specified time.

- **Time estimates.** All stories and tasks should be time estimated for use in the burndown chart.

- **Sprint stories to include this sprint.** Stories are taken from the product backlog and put in a sprint backlog. These should be time estimated to fit into the sprint in the sense of workload and time in sprint. Stories that are too big to fit into sprints are called epics, and are divided into smaller stories before included in sprints.

- **Time and place for the daily scrum meeting.** An important issue that can be decided by the scrum-master himself and emailed to the team if it does not fit into the planning meeting.

Sprint phases

The sprints are the time of actual work, and this is the main part of the Scrum framework. A sprint can be seen as another word for a month of working, but the actual length of the sprint can be anything between 3-6 weeks, depending on the team and the product developed. Recently it has been advised to use as short iterations as 1-2 weeks, if the project is small. The team will focus on the product being developed and no other disturbances are allowed. To enable the team to work undisturbed, the Scrum master shields the team from all incoming disturbances like a protecting wall. This could be any incoming requests from the Product owner or other stakeholders. One common problem is for example that upper management want certain team members to jump in and work on other projects part time. If the team is falling behind on the stories, the Scrum master and the Product owner can
together decide what stories to drop for the next sprint. If progress is quicker than expected, extra stories from the product backlog can be put into the sprint backlog. Every sprint starts with a sprint planning meeting and ends with a sprint retrospective.

Every day there is a meeting with the whole team and any other interested people. Only the pigs are allowed to talk though. This meeting should be timeboxed to 15 minutes and everyone summarizes what has been done since the last meeting and what will be done until the next meeting. The three questions to be answered are:

- What have you done since the last Daily Scrum regarding this project?
- What will you do between now and the next Daily Scrum meeting regarding this project?
- What impedes you from performing your work as effectively as possible?

The daily Scrum meeting gives awareness of what is going on and how the project is progressing. The purpose is also to eliminate any speed impediments that slows down the work.

When the sprint is finished the functioning software is shown to a larger group of people, often the Scrum team, Product owner and upper management, as well as other interested people from the rest of the company. Having this demo encourages team members to work hard to actually have something that works at the end of the sprint. It is important though to not spend too much time on tuning the product for the demo, but find a good level without too much work on tuning and refinements. The sprint is then summarized and evaluated. The focus lies on the process perspective and not very much about the product itself. Everyone in the team should get some time to express their thoughts about the last sprint and summarize if there is any better way of doing it.
3.8.3 Tools

During the sprint, all team members make sure to update the progress of the sprint stories. This allows for visualizing progress in a burndown chart for everyone to see if the sprint is on schedule. The remaining work is tracked on the vertical axis and the time periods tracked on the horizontal axis. See figure 25 for an example of how a burndown chart might look like.

![Burndown Chart](image)

Figure 25: An example of a burndown chart

The task board is a physical board used for task management and status monitoring. It is important that the task board is easily accessible by all team members. The board contains sprint goals, a burndown chart, tasks and stories. The stories and tasks are positioned depending on status and priority. Higher priority is indicated by placing the item higher up on the board. There is different slots to put items. The category not started is the first place to look for new tasks to do. When a task is finished it is moved to Done. See figure 26 to see a sample taskboard.
It can often be valuable to know how the team members are doing, on a personal level. A tool that can help here is a Niko-niko calendar. This is a simple board that looks like a calendar. At the end of every day each team member puts a little sticker in current day. The sticker will show either a happy, neutral or sad smiling face. This will indicate how the person is feeling at the end of the day. Many sad smileys can indicate that something is wrong with the project, like many members working overtime or under time pressure.
4 Case study

This section concludes observations, interview sessions and archive analysis into actual findings describing how the current situation at Mobical actually looks like. The first part of the section gives an overview of Tactel AB in general, the products it offers and the Mobical team in particular. The section continues with describing current work process in Mobical, including quotes from interviews, and the issues we have identified.

4.1 Description of the company Tactel AB

This master thesis is being performed on - and requested by - Tactel AB, which is a software house in Malmö, Sweden. The overall business idea is formulated as:

Tactel is a leading developer of mobile applications, providing solutions and consulting services to many of the world’s major network operators and mobile handset vendors. We help these customers stay ahead of the game by designing genuinely original applications, and making sure they work on all networks and with all handsets. [81]

4.1.1 Organization

The company Tactel was founded in 1995 and is still privately held by its founders. It has been profitable from the start and has had a steady growth ever since. There were only 4 employees in the beginning, but this has now risen to over 300. The headquarters are located in Malmö, Sweden with four other offices in Sweden and two in San Diego and Kiev respectively. The Tactel group also includes the subsidiaries Southend and Letcat as seen in figure 27. Tactel Sweden (figure 28) has some key individuals handling sales and administration of the company, but the main focus of resources is on the internal and external projects carried out in the region of Malmö and Lund. It should be noted that project 1-3 are not three separate projects, but departments containing more than one project. The Mobical team is a part of the Project 2 department. Sales for Mobical is mostly carried out by Fredrik Wångberg with the help of Jonas Falk who is situated close to the development team. Jonas Falk is the Product owner of Mobical as stated in Scrum (section 3.8).

![Figure 27: Tactel Group organization](image)
4.1.2 Business area

The business area of Tactel is about developing mobile applications with three main types of customers, handset vendors, content providers and network operators as seen in figure 29. Examples of applications are GPS solutions, MobileTV, content clients like CNN client, Windows Mobile applications, MMS and Bluetooth solutions, SonyEricsson specific SEMClet solutions and SyncML solutions. All projects are run differently as insourcing, outsourcing or solutions projects depending on the nature of the project and the wish of the customer. As an answer to the market needs and specific competence within the areas of embedded systems, graphical interfaces and user friendliness in the company, the subsidiaries Southend and Letcat have more specific focus within the area of business.
4.2 Description of the project Mobical

*Mobical* is a reliable, flexible, and interoperable synchronization platform that enables network operators and service providers to make life easier for their users by launching Mobile Backup/Restore and Central Address Book services. - Mobical sales documentation

4.2.1 Product

Mobical is a SyncML [2] solution for synchronizing and backing up personal mobile phone data over the air to a remote server. This personal data can be address book contacts, calendar events or SMS messages. It supports end customers that have multiple mobile devices but wish to have the data synchronized between them and as a security measure for having a backup if something happens to the mobile device. By using the SyncML protocol, already built-in in the most common phones and settings possible to preset from factory or edited with an SMS, Mobical offers a very easily used way of backing up contacts, calendar, notes, bookmarks and SMS. This information is then editable in any Internet browser for later synchronization into the mobile device.
The solution is mostly sold to the customer segment of network operators that get a customized version of the generic product. The generic product Mobical.net is a concept platform where the project itself handles the user accounts, usually handed out for free, where a lot of feedback on bugs and new feature requests are collected from the users. Apart from this channel of elicitation, customers usually request for new features in their solutions. These are often used to upgrade the concept platform as well.

4.2.2 Team

The Mobical team, as it was, consisted of 12 members with various backgrounds and skills. Depending on these skills and organizational decisions these persons include the roles of one Product owner/sales, one project manager/Scrum master, two part-time key account managers/developers, three testers including a test leader and four full-time developers.

Project work is being performed using the Scrum methodology described in section 3.8. When the work with this thesis started, the team was using a modified version of Scrum as the framework for development. During our presence the development process was restructured and improved to fit better to a real version of Scrum dividing the big team into two smaller and somewhat equal teams in the sense of competence.
4.3 Current process description

The following section will outline the current process that the Mobical team is following, and what shortcomings we have identified during the empirical studies. The data is based on interviews and observations as described in the method section (sections 2.2.2 and 2.2.3) of this thesis report. We will also use the term "smell" as described by Amr Elssamadisy [27] to point out specific areas of concern that are related to Agile methodology. It is essential to remember that the role project manager in the following text refers to all management roles in the team including Product owner, project leader and customer project leaders.

![Diagram of the current development process at Mobical]

Figure 30: The current development process at Mobical

4.3.1 Project inception and elicitation

Development at Mobical can be seen in two different approaches. There are base products, such as Mobical Core and AirCal, that are market-driven with no specific customer, but development is driven forward based on general needs and what customers might want to have. The other path is specific projects for a customer. Often this is a customization of one of the base products. This devel-
development is contract-driven. A project manager explains that “we want to sell as much standard solutions as possible, that will not require much new development” since this is most cost effective. Development of new features or products "should be sellable to multiple customers". There is no clear or structured elicitation process at Mobical. A project manager states "elicitation is very informal. We are bad at these kind of things". A tester describes how he actually elicits requirements to create the test plan: "I have to get hold of customer and project leader to get all information". One developer says: "Requirements are fuzzy at start, and with time they clear out.” which shows how more information about customer wishes are coming in as the project progresses. This is a clearly Agile method of working, as described in 3.7.1 and related to embracing change and collaborating with customer. The usual business development process follows a standard chain seen in figure 30. The customer first requests information about the product (RFI) and a customer project manager replies with description, alternatives and business solutions. If the customer is still interested a RFQ (Request for Quotation) is sent back to Tactel. This contains requested features (can be seen as early requirements), price range and preliminary schedule. A sample of an existing RFQ at Tactel can be seen in appendix F. An RFQ is sent to multiple suppliers in a tender process. Mobical replies with their solution to the problem, which is a Software Requirements Specification (SRS). This SRS is negotiated if necessary, and signed by both parts. Simultaneously a project plan containing schedule, resources and other information, is constructed and sent to the customer. A contract is then established.

4.3.2 Documentation

There is lots of different kind of documentation available, such as:

- Project Plans and different kind of Software Requirements Specification, in Word-format
- Test Plans, Master Test Plans and Acceptance Test Plans in Word-format and Test-cases in Excel-format
- Screenshots and basic prototypes
- Email conversations
- Wiki pages containing various information
- Tasks in the Scrum tool
- Bugs in the BugZilla database
- Bugs reported by customers in the SupportTrio database

Different projects contain different documents. Sometimes the customer will include specific screen-shots or other forms of requirements in their specifications. There is little or no traceability between the documents, so for example it is not possible to see which requirement a task comes from. A lot of the documentation is also not present in written form at all, such as phone communication with the customer which might lead to changes in the project. A tester points out how “documentation is badly structured” and “we have a lot of information in our heads.”

The requirements documents do not follow any standard template since, as a project manager says, "the projects are divided between project managers”. The SRS documents are often not very clear and do not follow the quality criteria like we described in section 3.1.3. This has often actually caused problems in the past as a tester says “We have had one or two projects that have failed, which was due to fuzzy requirements”. It is also evident that the SRS documents do not contain all necessary information: "We test a lot more than is specified. We have an idea what the customer wants and tests that even if
there are no requirements for it. ”, as a matter of fact “if someone had asked me for a description what [the system] can do, I would have sent the test cases”. This shows the test documentation as the most detailed description of the system: “In my eyes it is the most detailed specification of the system. I try to write everything so it is exact.”. The blame for lacking requirements documents should not only be blamed on the authors as a project manager says how “sometimes it would be nice if testers and developers read through the system description to see if there are loopholes or something that needs to be clarified first”. The fact that documentation is lacking has also caused problems such as the one described by a developer: “I worked on a feature for two weeks, then it turned out it was not needed in the first place. That makes you a bit disappointed”. We have also noticed that many developers do not enjoy looking through lots of documentation and one of them points out that “[the SRS] is nothing you browse through voluntarily. None of us would touch it unless necessary”. The SRS requirements are rarely numbered, which makes it difficult to provide traceability back to original requirements. A portion of a sample SRS can be seen in appendix E.

There is very rarely non-functional requirements - such as performance, security, reliability and usability - present in requirements documents at Mobical. They can be informally expressed as a tester describes it “sometimes we see something coming in looking really ugly, but it is not anything that is written down in the requirements”. There is sometimes a wish for security in form of having a firewall installed, but not expressed as a specific criteria. In all, non-functional requirements are very informal if they are present at all. One tester concludes “internally we testers can comment, but there are not any test cases for [non-functional requirements]”.

4.3.3 Team composition

The traditional Scrum approach has a development team with a Scrum master who helps by acting as a gatekeeper and clearing impediments. There should be a Product owner whose responsibility is to prioritize the product backlog. The Mobical team has diverted from this approach, mainly in the sense of the Scrum master being more of a traditional project manager, and there are two additional project managers that work more closely to the customer, but partly do some development work. One of these says “We are a small team working with big customers, and the role splitting is unproportional. Sometimes it would have been better with a bigger organization. As it is now, I sometimes visit the customers as the project leader and sometimes as a developer”. The Scrum master’s multiple role has rendered complaints in the interviews, for example a tester says “I would like more order and that would mean more time on purely being Scrum master”.

Despite this fact the team is otherwise not cross-functional, with the strictly divided roles of developers and testers. There is clear support from management and team to make the team cross-functional and let team members perform different tasks. A project manager says “it is outspoken that we want to widen this, but there is no goal or process formulated for reaching there. There is still a lot to do.”. A tester claims “I would like to have people more involved in all areas. If developers understand a bit about testing, and testers a bit about development and core developers knows a bit about GUI, and so on.” and another one states “It would have been insanely fun to learn more and be able to alternate work”. The developers are not quite as enthusiastic: “I’m quite happy with what I do” but most still think “it can definitely be done”. Additionally, one member points that “the employees’ knowledge should be a resource of investment, and in that area I think Tactel are quite bad”. The divided roles in Mobical leads to the smell “Us vs Them”, which can cause problems such as working for ones own subgroup’s best instead of what is best for the whole team. Although we have not seen much evidence for it, the situation can risk “blame games” where i.e. the developers blame the testers for problems and vice versa.

The developers are simultaneously working concurrently on multiple projects and with multiple tasks. This means that the sprint backlog is filled with items from many projects and makes it very
difficult to get an overview. One developer describes this: "In the Scrum tool we now have a list with like 500 tasks. I would like to see only my own tasks". This is an example of the team members desire to reduce the amount of work items visible. The situation of having developers working in multiple projects leads to the process smell "Bottle-necked resources" as the developers are tied up to multiple projects.

### 4.3.4 Organized chaos

It is quite common that companies switch to using Scrum after using a very strict process with detailed plans, Gantt charts and work schedules [71], which means a more loose approach to the development. The Mobical team on the other hand, what we discovered by our archive studies, have been trying to have these strict process tools, but failed in doing so and the result has become total chaos. Mobicals migration to Scrum has hence been from total chaos to Scrum, which became something more organized. Sadly it is evident that for many team members Scrum is only seen as a computer tool for keeping tasks and not a work process or philosophy. Some people in the team find it frustrating with the lack of structure. A tester says *When we are stressed it would have been great with more structure* and another one also wants improvements: *"I would like to have some kind of project model for how to manage the limited project. I try to run a controlled test process and I have to push that into the project every time"*. There is, however, a balance between order and chaos. Many enjoy the relaxed atmosphere and one developer even considers the best things with working at Mobical to be "*No really hard demands and lots of freedom*". It is rather obvious how Scrum as a process, is not quite working. A tester says he *"I would like to see more emphasizing on getting everyone to work better with Scrum. Everyone have probably not quite understood Scrum and do not see the point of using it"*.

### 4.3.5 Development

The actual development starts as soon as there are stories present in the Scrum tool. The basic idea in Scrum is that all project members can choose which story to work on, and preferably pick a story that is highly prioritized. If a story is very large in extent and complexity it is broken down into sub stories, or tasks. This is sometimes done in advance prior to start of the sprint, but in some cases developers do not actually do a break down except in their head or on a post-it note next to their computer monitor.

In Mobical the developers often know in advance which is "their" area of expertise. Certain developers are skilled in the database backend and others in the GUI layer and they tend to stick to their areas. Test-driven development [54] is encouraged but not actively enforced. There should be JUnit tests for all new sections of code, but there are still vast amounts of old code that are not covered by unit tests. Developers rarely actually write tests first, which has been identified as a problem. as a project manager says *"We want more Agile processes, like JUnit tests, into the Scrum framework."* Studies [13] have shown that pair programming is the least adopted Agile technique in the industry, even if it usually pays off when talking about the quality [18]. When asked, many developers in Mobical are also very hesitant about this. For example one says *"I cannot handle having someone else sitting next to me and coding. Someone taking over the keyboard writing faster than me so I cannot catch up."*.

One of the most appealing business situations for Mobical is when a customer wants to use an existing product and have it rebranded or some minor functionality added. This is supposed to be simple modifications to an existing demo platform, but they take longer than they should to implement. This leads to the business smell "Delivering new features to customer takes too long".

When developers have completed their code it is handed over to testing.
4.3.6 Testing

The test process is a bit separated from the main development cycle. The most important document is the test plan, which describes acceptance criteria, tools and methods used for testing. This is written just before, or concurrently with, the start of development. The next step is creation of test cases. Very often test cases can be reused from earlier projects. There are few guidelines how test cases are written and they cannot be traced to original requirements. A tester explains: Requirements are not labeled with version or number, so it is not possible to create a test plan based on it. When asked if he would reference test cases to requirements if they were numbered the response is "Yes, I would do that. I know that is the way it should be done". The current method leads to the unpleasant situation where requirements are not verified, and thus it is not known if the customer actually gets what he asked for. Since requirements are not validated there are ultimately problems that show up when a test case fails. Maybe the test case was based on a requirement, but the developer did not look at the requirement as he got the information needed elsewhere. Perhaps the test case was not based on the right requirement and thus the test case is wrong. The third alternative is if the code and test was based on the correct requirement, which would point at implementation error. It could also be that the requirement was misinterpreted by either tester or developer. Regardless of the reason, failed test cases leads to bug reports, which are stored in the bug repository tool BugZilla [57]. The bugs are then fixed and tests are once again run. This will continue iteratively until the tests go through. In the bug report it is described how the bug was found with step by step instructions. The last step is to run an acceptance test, which often the customer allows Mobical to take care of. A report of the acceptance test is sent to the customer. Overall the test process, as expressed by a project manager, "doesn't work at all as it should". One reason for this could be that very few people, except testers, are aware of the test process. When asked about the test process a developer replies "Nothing I think. I never see the test cases because I don't look for them." and another "They have some kind of test plan with test cases that they go through and click".

4.3.7 Awareness

The standard Scrum method for estimating time schedule is based on team members continuously updating status of progress left for a story. Mobical does not use this approach, which makes time estimation very difficult. Developers are unsure about how long a task takes and one big reason for this is the widely different definition of when a task is done. Some developers claim their task is done when "it has passed junit tests" others simply "...mark it as done when it is implemented". The official definition of done, that Mobical uses but has not been communicated enough, is when a task is "documented, unit-tested, developed, integration-tested, acceptance-tested". There are problems with this definition as well.

A project manager concludes "planning is horrible". The lack of updated estimations of effort left makes it impossible to use a burndown-chart to show project velocity. The lack of a task board (which in Scrum can be used as a status indicator) makes it additionally difficult to get overview of what stories are worked on, in testing and done. This situation is an indicator of the process smell "Management is surprised".

4.3.8 Communication

The team is quite large with 11 people and it has shown to be a challenge to manage this large group. Scrum recommends, or even dictates, a team size of 7 +/- 2 people (see section[3.8.1]) so in this aspect Mobical has a problem. As a tester points out "We might be a bit too large group for Scrum
and we have too much going on at the same time”. The team is physically located in one big room, with several groups of desks. This seems to encourage communication and interaction, but despite this there seems to be complaints about information not coming through and some people want more meetings to remedy this. A developer states “I would love to have weekly meetings in addition to the daily Scrum meetings. One hour on fridays as a kind of part-sprint meetings. As it is now we are clueless about the status until the end of the month. There are so many tasks that you cannot get an overview”. Another issue that complicates teamwork is that one team member is not physically present, but working off-site, and must be contacted by phone, mail or instant messenger.

The most appreciated aspect of Scrum is the daily standup meetings. One developer says “the daily Scrum meetings are really good! When I joined [the team] I never knew what the others were doing. When we started with [Daily Scrums] I had full insight after just a few days.” The daily Scrum is a rather short meeting which allows everyone to speak and say what they have done. In standard Scrum the daily meetings should only address the three questions: What have you done? What are you going to do? Does anything impede your work? Mobical has extended this and will discuss general areas of interest such as discussions about implementation alternatives and status of different bugs.

During the interview sessions we noticed another issue related to communication. Team members use different terms when describing items. Some people call the requirements specification SRS, some call it FRS. The work items are sometimes called stories, sometimes tasks and sometimes items. To develop a common vocabulary as a basic Agile practice, as described in 3.8 in relation to Scrum, is the standard procedure to overcome this problem.

### 4.3.9 Release management and time planning

At Mobical they have tried different lengths for sprints, 3-5 weeks, and settled with 4 week sprints. Each project has a release date fixed in the project plan. This date does not coincide with end of sprints, except in rare occasions. Time planning from a sample project plan is show in figure [31] which clearly shows how the project is developed in a rather traditional fashion despite the fact that Agile development methods should be used. In the case of customer projects (contract driven) releases do not represent an iteration, but the completed project, which is often not longer than 8-10 weeks according to a project manager. With projects such as Mobical Core and AirCal (market-driven) the releases represent an iteration, with increased functionality, but they are not released at the end of sprints but rather when it is deemed suitable for management. Due to this Mobical reaches for stability by introducing the smell “Hardening (freeze) phase needed at the end of release cycle”. This is by some people not considered a smell, but as a matter of fact each iteration should in theory produce a working, integrated and tested system [27]. The code freeze indicates that iterations are not really iterations, but rather time blocks of work. At certain time intervals released projects are merged together with the latest release of Mobical Core. In general we have observed the process smell "Integration is infrequent", due to the fact that branches are rarely merged.

### 4.3.10 Product quality

The products that the Mobical team ships are not free from errors. In fact, there are a number of bugs in the bug database. Many of the bugs are old and have unknown status and it is in general very unclear what bugs have been fixed and what are still active. However, bugs that are reported from the customer have higher priority and are handled more carefully. There can also be difference between bugs in branches and delivered products. "Hundreds of bugs in bug tracker" is a process smell and the low visibility for bug status increases "Management is surprised". This is an indicator of a significant investment in work that is never released to the customer and thus has zero value.
Figure 31: Time planning from a sample project plan

One developer expresses his frustration with quality as "It was released half-finished. We could have worked really hard with it until everything worked. Perhaps it can’t be done but I would appreciate higher quality and more time on projects". This statement reflects how time is running out and products are released before they actually work. The "Hundreds of bugs in bug tracker" smell can be countered by different testing techniques [27], and testing is, as described in section 4.3.6, quite a problem.

4.3.11 Current tools

When the work with the thesis started, the Mobical team was using the tool Scrumworks, to help them with tracking their Scrum stories. This is the tool they had been using since they started using Scrum. It is a very basic Windows based tool adapted to Scrum development. Some months before, they had also tried the Pro-version of Scrumworks, that provided some extra features to the basic tool. During the interviews there were many complaints about the tool. One developer describes how "I do not go and change status in Scrum, I do not like to use it." and management is aware of that, "People find it very hard to try. There is documentation to read.". Another developer stated "I think it’s hard to find the tasks in Scrumworks", which in fact goes back to the problem with people having their own tasks as shown in section 4.3.5.

During the thesis work, the team decided to switch tool, since they thought Scrumworks was insufficient. They started to use Hansoft that is described in section 5.5.1 that is also used by Southend. According to conversations with the team, Hansoft is a better tool that provides a better overview and features that was missing in Scrumworks, like a developer stated, "The ideal case would be to have everything in one place, with references and everything.". Hansoft provides this better than Scrumworks.
5 Analysis

In this section we conclude findings from the case study and point out the main issues addressed in our solution. Later we present our solution in a generic model for working with requirements in an Agile environment, first in a summary and then into more detail. Next our tool evaluation is presented and concluded and then we have evaluated our generic model in an internal and external evaluation.

5.1 Case study conclusions

In the current process description (section 4.3) many different issues related to the requirements engineering and development processes, and hence in the end related to the product quality 4.3.10, at Mobical were identified. The most important of these are concluded in this section. We noticed that Mobical have both contract-driven and market-driven development, but the most common area is contract-driven, which also is most in need of improvements. Our solution will hence focus on contract-driven development.

5.1.1 Project inception, planning and requirements engineering

**Elicitation** is not carried out sufficiently in the beginning of the projects. This results in many essential requirements missing when development starts, which in turn leads to missing functionality, rework and missed deadlines.

**Requirements are not validated** when elicited and specified. The result of this may be requirements that do not meet customer wishes, maybe not testable, ambiguous or any other bad requirements characteristics.

**No documentation standards** are used for requirements specifications. This is because of the fact that there are multiple project managers for the different projects and sometimes the customers have demands of certain documents for their specific project. This often causes confusion among the other team members.

**No traceability** is present within the documentation, partly because of the lack of numbering of requirements. This causes problems in the testing, since the test specifications cannot be backwards traced to any requirements.

**Information overflow.** With documentation at various places in different systems, it is hard to keep track of changes, since these have to be updated in all places. This makes the information in some places not to get updated at all times, making uncertainty in the validity of all information. The team often keeps important information from telephone conversations and emails in their head, creating opportunities for human errors.

**Prioritization** is problematic due to requirements that are hard to compare. In addition, all projects are run traditionally and since the contracts state that all features shall be implemented for the releases prioritization is not used for scoping. Related to this project managers also sometimes override prioritization during sprints by pushing new tasks or changes over what has been committed.

**The time planning and estimates** are carried out traditionally even if the goal is Agility. This causes a conflict when a project plan (visible for customer) dictates what work should be performed when, and the Scrum team has their own planning.
5.1.2 In-project, workflow and iterations

**Visibility** within the project in the sense of bug tracking, time left of project, progress status and product quality is one thing that is important for the team to have. Without being able to visualize statuses, it is hard for the team coordinate the workflow.

**Sprint concept and length** is a huge problem at Mobical. One symptom of the limping Scrum implementation is a sprint length of 4 weeks, which does not relate to any customer project deadlines or internal releases in any way. As for now the sprints at Mobical are only boxes of time, and not time-boxed workflow. Since the projects usually are short (about 8-10 weeks), the 4 week sprints cannot several create iterations of working software.

**Cross-functional team.** There are certain key experts within the team, that know certain areas of development better than the rest of the team. When any of these are gone, some work can be stagnated for days without progress.

**Parallel projects.** The same team has several projects running at the same time, with different people active in all projects. This creates a confusing environment for the team that has to adapt to new tasks whenever they switch between projects.

**Team size** is not the ideal for Scrum. The Mobical team consists of 11 people, divided unevenly between all the different projects. Since Scrum works best with smaller teams they are divided into two teams, with the effect that some people get locked from use of the other team.

**Team composition.** For running Scrum, Mobical has too many management roles, partly since they are stuck between Scrum and the traditional roles. Apart from the Scrum master, who is entitled project leader by management, they have a Product owner and two customer project managers. There are also the outspoken roles of developers and testers which is not in line with Scrums cross-functionality mentioned above. This can lead, as we have also witnessed, to the situation where the project managers influence the Scrum team in midst of sprints by putting extra demands or transferring resources.

**Bugs** in Bugzilla. There are hundreds of bugs in the bug tracking tool of which many have unknown status. They remain in the tool because it is too much work to go through the database and will likely never get fixed or investigated. The vast amount of bugs, many of which have unknown status, is a clear indicator of the smell "quality delivered to customer is unacceptable" and a sign of the test process which isn’t quite working.

**Integration** is done infrequently between branches that are unclear within the team. The customer projects have different branches and settings, causing confusion for example when testing. Because of the infrequent integration, testing cannot be done when new features are added which reduces product quality.

**No verification** of if the system actually fulfills the requirements. Because of the unclear and lacking requirements, the testers elicit their own testable requirements when creating test plans and test cases. This allows for some testing, but the original requirements, which contract is based upon, are not verified.

5.1.3 Tools

The case study has investigated if a tool can assist in requirements engineering and project management. The interviews revealed many team members being very unhappy with the tool currently in use, but there were very varied opinions about whether a tool would actually improve the situation.
We have not found any evidence of a tool being solution to the above mentioned issues, but have
never the less identified a number of quality criteria that describe what is important to look for in
a tool. These criteria can be found in appendix C and were used to evaluate a number of different
tools. Results from the tools evaluation can be found in section 5.5.
5.2 Introduction to solution

In section 3.7.2 we described previous work of how requirements engineering can be handled in projects using Agile methods. This gave very much inspiration in the development of our new process. In addition, we have identified that Agile methods traditionally break down stories into smaller sub-tasks that fit better for technical implementation and are easier to estimate. During our literature studies we found that the Requirements Abstraction Model (see 3.3) takes a very similar approach to requirements. Requirements are placed at an initial abstraction level and abstracted and broken down, and are hence comparable at different levels of detail. Just like Agile methods discovered that broken down stories are easier to time estimate, one motivation behind RAM is that it is easier to compare requirements at the same level. RAM will help product management with visualizing requirements and make it easier to prioritize backlog items. Gorschek presented more methods for process improvements in small and medium sized businesses, but these have not been taken into account in our solutions.

There is a kind of conflict between traditional and Agile methods in project management. The old way of running projects means to plan in advance, strictly specify what should be included and use tools like project plans, gantt charts and rather extensive documentation. In addition, contract is fixed and allows for very little flexibility. The Agile way, as described in section 3.7, is to choose people and interactions over tools and processes. Change is embraced throughout the whole project. Agile methods try to avoid too much documentation and strict processes. We aim to, in our model, be able to bridge these two worlds by allowing a project plan and contract, but still keep time-boxed iterations where the development team can work freely on their sprint backlog items.

For this reason, the new process does not interfere with the Scrum sprint, which can be seen as a black box, but rather focus on making the pre-project requirements engineering as efficient as possible. The ultimate goal is to produce a product backlog filled with testable requirements of high quality that actually reflect customer needs. This solves many documentation issues identified in the case study, and will make the requirements easier and more structured to test.

Customer involvement is an important Agile aspect and one path of our solution relies on the fact that the customer is trusted, trusting and willing to cooperate throughout the requirements engineering and development phases. However, it is far from certain that Tactel’s customers are willing to try out this novice approach to working with requirements and contracts. Therefore we have also made a traditional path available, that still fits into the process and enables the project team to be Agile using Scrum.

Many of the major issues identified in the case study are related to requirements engineering, and these will be addressed by implementing our new process. There are additional issues that are not related to requirements engineering, but rather to the Scrum process or project management and those are described in section 5.7.

As discussed in the case study conclusions (section 5.1) the solution will focus on contract-driven development, but it is still applicable to market-driven as well. With market-driven development there is no customer so many of the problems of contracting and strict requirements specifications are likely to disappear. We believe the market-driven context is ideal for Agile methods such as Scrum since they expect a vast amount of requirements that are implemented in order of priority. As such we recommend following the Agile path of our solution if development is market-driven.

The next section describes our new process briefly, followed by an in-depth analysis of its steps.
5.3 Agile Requirements Abstraction Model (ARAM) summary

This section is a quick summary how to use the new requirements process. There are two different paths to follow in these phases, depending if the customer is traditional or willing to be more Agile. Figure 32 shows a graphical representation of this process. A more detailed description of the process is found in section 5.4. We have also developed two short guides to complement this process description for ARAM to be easy to use. These are presented in Appendices K and L and are guides for project management on the ARAM process and ARAM requirements process.

![Diagram of Agile and Traditional paths]

Figure 32: The two paths to choose between in the new requirements process

5.3.1 Inception

Agile

A customer contacts the vendor and discusses the business values she is interested in. The vendor describes the Agile nature of projects being carried out and how trust and customer collaboration is important.
Traditional
A customer contacts the vendor and requests product information, which is sent back along with information about the Agile way of working at Mobical. The customer reviews the information and sends a Request for Quote (RFQ) document to the vendor, containing requirements and wishes. The Agile way of working is not accepted by the customer.

Agile/Traditional
The inception step is obviously skipped if it is an ongoing project and new requirements arrive.

5.3.2 Elicitation

Agile
The customer and vendor collaboratively creates a Vision Statement, which expresses who the target end-user is, what needs are to be fulfilled and how this differentiates from competitors by certain advantages. The Vision contains the first high level requirements. At the same time a target-cost contract, that allows scope to be variable during the development, is signed by both parties. If development is market-driven elicitation should be performed using workshops, interviews, market analysis and other methods instead.

Traditional
The vendor goes through the RFQ and checks which requirements they can deliver. In some cases the vendor will suggest additional requirements that the customer didn’t think about and have been elicited previously in other projects or through workshops.

Agile/Traditional
Attributes such as status, description and value are added to each requirement. The fields for abstraction level and ID are kept empty for the moment.

5.3.3 Placement

Agile/Traditional
The requirements are analyzed and placed at an appropriate abstraction level that can be either Product, Feature, Function or Component-level, where Product level is the most abstract and Component level is concrete and closer to, example of, or describing how something should be solved.

5.3.4 Work up

Agile/Traditional
The next step is to work up the requirements, which means to either connect each requirement to other requirements or create new requirements at other abstraction levels. The results are unbroken chains of requirements from product level to (at least) feature level.
5.3.5 Documentation

**Agile**

Finally the requirements are inserted into the product backlog as backlog items, which are prioritized as usual by the Product owner. The product backlog acts as a "living" requirements specification, that can be modified and reprioritized at any time.

**Traditional**

The requirements are used to write a standard software requirements specification (SRS). The high level requirements are used to categorize and structure the document. The SRS is sent to the customer for acceptance, and after agreement and contract has been signed the requirements are inserted into the product backlog. The contract is a fixed-price/fixed-scope contract which allows time to be variable by keeping a time buffer prior to release date. Project plan includes a schedule with milestones that represent releases at the end of every sprint.

5.3.6 **Scrum**

When items are in the product backlog they will then be prioritized by the Product owner, who will consult the customer and other stakeholders. Backlog items that are highly prioritized will be worked on first and will need to be broken down to function or component level. These items will be given rough time estimates by the team. Other items, that are planned for a later sprint, can remain at feature level until it is further details are known. Items selected for next sprint will also be given acceptance criteria to make them easily testable.
5.4 Agile Requirements Abstraction Model (ARAM) details

This section provides a detailed description of the new requirements engineering process, with examples of how it can be used. Each step of the process seen in figure 52 will be explained in its own subsection.

5.4.1 Inception

In a standard contract-driven business scenario, as identified in section 4.3, a potential customer contacts the software vendor and requests for product information. The customer expects to receive a very overall picture of what business value is offered and how this can provide commercial opportunities.

The response to the customer at this stage is an easy-to-read product overview containing enough information for the consumer to know what the product can do, and what value it offers, but still limited to be quick to browse through. At Mobical this is common marketing material and samples of how existing customers successfully use tailored solutions. Our model does at this stage introduce additional information in the Product information about the Agile development methods used at Mobical. This is to allow the customer the choice of participating in the Agile process from the start, and in particular have a special kind of contract written. The Agile information will include an overview about Scrum, target-cost contracts and how the Agile path that will be described shortly.

The contract and requirements processes will ultimately end up in the product backlog, but there are two different ways towards that goal.

If the product is market-driven there is no direct customer to contact, and as such inception usually starts with some kind of innovation meeting at the producing company where high level goals are derived.

Agile

Our Agile alternative is to select a target-cost contract (section 3.7.4) with variable scope. The delivery date and budget are known in advance and requirements will be implemented in an order depending on priority. There will be a release at a determined date, and it will contain the most important features. This is the recommended method of working in Scrum. Changes are allowed, and encouraged, throughout the development process and new versions of working software will be released at the end of every iteration, to allow for feedback from the customer.

Traditional

The traditional customer prefers, or requires, a standard fixed-price contract (section 3.7.4). This is the current de-facto situation at Mobical. The key advantage for the customer is a contract that dictates total price and time of delivery. One disadvantage with fixed price contracts is that the tender procedure is likely to favor the cheapest, or most desperate, alternative. Another problem is that changes cost money and need to be handled strictly through change requests. Fixed-price contracts put the supplier in a very defensive position and if margins are low, it is highly tempting to press change requests in order to gain extra income. The main down-side for the supplier is that it has to take large risks. The customer might not agree that the product fulfills requirements and hence refuses to pay. The customer could also agree that requirements are fulfilled, but is still unhappy because the requirements did not reflect what was desired in the first place, as described in section 3.7.4. Fixed-price contracts require that cost is estimated up front, and this estimate is only possible to do with at least some high level requirements available.
The Iron Triangle shown in figure 23 (in section 3.7.4) dictates that at least one factor must be kept variable for a successful project. We believe, from reviewing old contracts at Mobical (see figure 31), that the schedule can be allowed to be variable. That means the customer will receive a fixed price for a specified number of features. The fixed scope on requirements is not desirable from an Agile perspective since changes must be performed via change management procedures that will cost money. Nevertheless, this is often a real world scenario with the customer at least wanting a fixed price for a fixed amount of features. If time also is a very important factor, the number of implemented features must be reduced. Our case study has shown that the least variable factor is the price. Even if the customer gives more funding to the project (for example to meet a deadline) it is unlikely that there is enough knowledgeable personell available to transmute the money into work done. With schedule variable the delivery date will be estimated up front, but updated as project evolves depending on sprint velocity. From a customers point of view it might actually be fixed scope, price and schedule, but the contractor adds a safety margin to the delivery date, making the time factor variable in practice. In worst case the customer is not willing to sacrifice any of the factors, and hence tries to keep schedule, cost and scope fixed. This cannot be done without lowering product quality, which is acceptable. In this case a time buffer can be used to make schedule at least variable to some extent. For example, keep one release date communicated with the troublesome customer and one earlier release date for the team.

Note that the contract is not actually signed yet, until the requirements specification is created and customer knows what he will receive.

5.4.2 Elicitation

After project inception, the next step is to elicit requirements. The continued process for how a requirement is handled is visualized in figure 33. This is very similar to RAM, with the steps specify (which, in this context, is the same as elicit), place and work up.

Agile

For the Agile customer the next step is to cooperate with supplier to formulate a product vision statement. Geoffrey Moore [53] proposed a product vision model that aims to enable team members to pass the "elevator test", which is the ability to explain the project to someone within two minutes (the time it would take to travel with an elevator in a tall building). The form follows:

- For (target customer)
- Who (statement of the need or opportunity)
- The (product name is a product category)
- That (key benefit or compelling reason to buy)
- Unlike (primary competitive alternative)
- Our product (statement of primary differentiation)

A made-up example of the elevator test in the case of Mobical could be: "For mobile phone users who risk losing their valuable personal data, the Mobical platform is a synchronization service that lets users to retrieve what is stored on their mobile phone in case they loose it. Unlike using a cable-based sync solution, our product lets the users synchronize wherever they are." This simple sentence could easily be communicated within two minutes, and it contains what is most essential about the product. This vision
helps development focus on what is judged to be most important. The vision statement should contain overall business values. If the customer is unsure exactly what is wanted a simple technique is to have a brainstorming session to come up with requirements or features and then use any of the methods for prioritization described in section 3.1.4. The vision statement will contain requirements, but at a very high level. Even with market-driven development this step of creating a vision statement should be followed, but elicitation is rather performed using workshops, interviews, market analysis and other methods instead.

**Traditional**

The traditional customer would choose to submit a Request for Quote document instead of a Vision Statement. This is a tender proposal that will contain requested features and can be quite detailed, depending on customer needs and habits. The document can be seen as the first elicited drafts of requirements, or at least customer wishes. It will be analyzed by Mobical and compliant features are marked. A proposal is sent back to the potential customer, who will compare other offers and decide who gets the contract.

**Agile/Traditional**

When requirements are available in RFQ or Vision statement, some attributes are added for each requirement. We have selected just a few of the proposed attributes (section 3.3.3) described by RAM that will be used as requirement attributes, seen in table 13. While implementing this process, more attributes may be added to support the specific project.

While inserting attributes it makes sense to decide if the requirement should remain or be discarded, so no further time is spent on unnecessary work. If the requirement is not selected to remain in scope, the status is changed to "Rejected" and the reject reason is updated with a short description why this decision was made.
Description is a natural language formulation of the requirement. If additional information about the requirement is available, such as use cases or screenshots, it is noted here with a reference to the external location. The ID number will indicate abstraction and connections to other requirements, and will be further described in section 5.4.4. The level attribute will be filled out in the next step, and kept empty for the moment. Value is the reason why this requirement exists and what value it adds to the user. The source attribute shows where the requirement has originated, which is usually "RFQ" or "Vision". The status alternatives are the ones suggested by Wiegers and described in table 4. Owner attribute indicate who is responsible for the requirement, and in the case of Mobical this is almost always the Product owner.

It should be noted that at this stage it is useful, or practically necessary, to insert the requirements in some kind of repository. This could be a spreadsheet or a real requirements tool. We suggest that requirements are entered into the tool that is used as product backlog. If possible, create a category in the tool called "ARAM in progress" or similar and put the requirements there while working with them.

### 5.4.3 Placement

Regardless of if the customer is Agile or traditional, by this stage the first set of requirements have been specified. In the case of the traditional customer the requirements are most likely numerous and at varied complexity level. The Agile counterpart has few requirements and at a high abstraction level. This is to be considered specified requirements as stated by the first action step of RAM (section 3.3.2).

The next step is to place the requirement at an initial level. The levels presented in table 14 are the same as described in section 3.3.1 with the difference that the business level has been removed since it is too abstract to work up from. This is also recommended in the RAM implementation guidelines.

Deciding at which abstraction level a requirement should be placed can sometimes be challenging and require additional analysis. The following questions can assist in this work:

<table>
<thead>
<tr>
<th>Attribute title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Text for labeling the requirement.</td>
</tr>
<tr>
<td>Description</td>
<td>Text with an informative description.</td>
</tr>
<tr>
<td>Level</td>
<td>Which abstraction level this requirement is at.</td>
</tr>
<tr>
<td>ID</td>
<td>An identification tag in the form a.b.c.d</td>
</tr>
<tr>
<td>Value</td>
<td>Text for rationale or benefit behind the requirement.</td>
</tr>
<tr>
<td>Source</td>
<td>From where does this requirement derive.</td>
</tr>
<tr>
<td>Owner</td>
<td>Who is responsible for this requirement.</td>
</tr>
</tbody>
</table>
| Status          | Proposed  
 Approved  
 Rejected  
 Implemented  
 Verified  |
| Reject reason   | (only if rejected) |

Table 13: Proposed requirement attributes
<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Product level</td>
<td>Abstract and goal-like requirements</td>
</tr>
<tr>
<td>2</td>
<td>Feature level</td>
<td>Features that the product supports, but not too detailed</td>
</tr>
<tr>
<td>3</td>
<td>Function level</td>
<td>Functional requirements, actions that are possible to perform</td>
</tr>
<tr>
<td>4</td>
<td>Component level</td>
<td>Detailed description of how something should be solved</td>
</tr>
</tbody>
</table>

Table 14: Proposed requirement abstraction levels

- Is the requirement comparable to product strategy? (product)
- Does the requirement describe what the system should include/support? (feature)
- Is the requirement functional and testable? (functional)
- Does the requirement suggest how to solve something? (component)

It can also be helpful to consider figure 34 to see how the levels are connected to requirements in business plan, road map and story repository (backlog). Connections also show how each level corresponds to acceptance, system, integration and unit testing. For example, if a requirement obviously seems to be tested at unit level, it most likely belongs at component level in the RAM framework.

Note that the business level is not present in our model, but shown in the figure to reflect the fact that the business plan will contain very abstract high level goals.

Figure 34: Requirements abstraction levels in our model

5.4.4 Work up

The next step after placing the requirement at the appropriate abstraction level, is to work up the requirement. This means creating new requirements, or link to existing requirements, that are connected to the initial requirement upwards or downwards in the abstraction level structure. The first work up rule of RAM (section 3.3.2) states that all requirements must have a connection to the product level, and our model abides to this. It means there will be no loose requirements floating in the
air. Rule 2 of RAM (section 3.3.2) demands that requirements have to be broken down to function level. We are changing this rule so that requirements only need to be broken down to feature level. The reason for this is that it can very well be that a feature requirement is elicited, but details for being able to break down further are unavailable at the time. In contrast to traditional requirements engineering, Agile concepts accept that there will be unknown factors that will get clearer at a later time. Hence, it makes sense to be able to keep some requirements at a rather abstract feature level. In addition, when abstracting requirements, any "empty" requirements should be filled out, so no links are broken. The rules for working up requirements can be summarized as:

- Rule 1: All requirements should be abstracted upwards to reach Product Level.
- Rule 2: All requirements should be broken down to reach Feature Level.
- Rule 3: There may exist no broken links in the requirements abstraction chains.

We exemplify working up requirements by describing how the attribute ID number is constructed. The ID number is composed in a similar way as an Internet Protocol (IP) address in the form a.b.c.d where each letter represents a link to a requirement. We provide some sample requirements to illustrate construction of ID numbers:

**User value:** It should be possible to retrieve data if a user's phone is lost

1 - Product level - It should be possible to sync personal information over the air.
2 - Feature level - It should be possible to sync the address book
3 - Function level - It should be possible to update a contact
4 - Component level - There should be a button for confirming change

For the first requirement we simply use the first available integer at level 1 as ID. If the requirements repository is empty it will be 1. The second requirement gets the ID 1.1. The third one gets 1.1.1 and fourth one gets 1.1.1.1. The number of digits will indicate at which level a requirement is.

The requirements exemplified above are all connected in the structure shown in figure 35.

![Figure 35: Example requirements and connections](image)

In the situation where a new requirement is added it has to be analysed and inserted into the structure. It could look like:

2 - Feature level - It should be possible to sync the calendar
This requirement is at abstraction level 2 and has the parent "Sync over the air". Inserted into the repository it receives the ID 1.2, hence the graph now looks like in figure 36.

Figure 36: Example requirements and connections, with new requirement added

As seen, requirements will form a tree-like structure where a requirement at high abstraction level typically will have many children and grandchildren. This also implies that all requirements will have an ancestor, unless it is at the highest abstraction level. It is also possible to have requirements that do not have a parent yet. An example could be a very detailed requirement at abstraction level 4, where requirements at higher level not yet have been identified, except that it is still attached to the most top level requirement at product level. The ID for this kind of requirement could be 1.0.0.2 where the zeroes indicate that there is no requirement at that specific level yet. Notice that work up rule 3 specifies that there may exist no broken links in requirement abstraction chains, so the empty levels are only present while specifying the requirement with attributes and filled out while doing work up.

Our objective is to have requirements that are suitable for placing in a product backlog and that can be compared for prioritization purposes. Now we have requirements that can be compared at different levels and this is where the model really shines. In some cases there might be requirements that describe functionality at a very low level and other non-related requirements of which details are still unknown. An example to illustrate this scenario would be as follows:

2 - Feature level - It should be possible to sync the calendar

to be compared to

4 - Component level - User account should automatically be deleted after 180 days with a cron script

These two requirements are very hard to compare, and hence it is difficult to prioritize them in a meaningful way. We now create two new requirements at level 3 and 2 for the later requirement.

3 - Function level - It should be possible to automate administration of user accounts
2 - Feature level - It should be possible to administrate user accounts

Now the requirements can be compared at feature level and Product owner can prioritize if calendar or user admin support is the most important.
5.4.5 Documentation

From the Requirements Abstraction Model’s point of view we are now done. The requirements are elicited, specified, given attributes, placed at appropriate abstraction levels and abstracted through work up to fill out gaps.

Agile

In the case of the Agile customer the requirements do not have to be processed any further, and will likely be at a high abstraction level suitable to be inserted into the product backlog as epics and stories (see section 3.8 for description of terminology about items in Scrum). A fundamental difference between Agile project planning and traditional project planning is that Agile project planning is very collaborative in nature [4]. It is not necessary with further written documentation, as the Agile customer is aware of what is in the product backlog and will, through contact with the Product owner, help prioritize all items.

Traditional

In the case of the traditional customer the requirements are now inserted into a standard Software Requirements Specification. Typically the abstraction levels are used for sections and categories. For example the sample requirements above could be expressed as in figure 37.

1. User account administration

This section describes requirements related to how user accounts are administered.

1. It should be possible to automate administration of requirements
   1. User accounts should automatically be deleted after 180 days using a cron script
   2. xxxxxx xxxxxx
   3. yyyyyyy yyyyyy

Figure 37: Example requirements in a SRS

The details of requirements abstraction is not necessary to communicate to the customer, and in many cases it would not be possible since some companies require vendors to use specific templates for requirements documents. The requirement hierarchy is of course saved in the local requirements repository anyway. From the customers point of view the requirements look like ordinary requirements except they are unusually well structured.

The traditional customer will also expect to receive a project plan that details time, scope, resources and additional information. This document can be standardized. It is common to include milestones as descriptions of a baseline reached at a certain date. An example of this was displayed in section 4.3.9 The milestones will represent iterations of working software at the end of each sprint. So, even if the customer prefers a standard requirements specification and contract, the software will be delivered in increments as Scrum dictates.
5.4.6 Scrum

When requirements have entered the product backlog as backlog items they need to be prioritized according to what holds most value to the customer. The big advantage now is that all requirements can be compared. If a feature level requirement is to be prioritized it can only be compared to other requirements at the same abstraction level. The good news now is that all requirements can be expressed at feature level because of the previously performed work up step.

After all items have been prioritized they will be analyzed as usual at the next sprint planning meeting. The high priority items will be selected for the coming sprint, and while estimating effort for each story they need to be broken down to at least function level or else it will be very hard to make good time estimates. In some cases it may even be desirable to break down to component level, but usually that is done during the sprint, by the development team. One Agile practice, and basic rule of Scrum, is to develop a common language (see section 3.8) so all team members can refer to artifacts using the same name. This has also been pointed out in the case study (4.3.8) as a problem. For this purpose it is highly desirable to have a common name for each kind of requirement at each abstraction level like the example in figure 38. These names derive from what is currently used to describe items at different levels. The product level name, goal, can also be seen as “project name”.

![Abstraction level and Scrum term diagram](image)

Figure 38: The terms used in Scrum for requirements at different levels

While breaking down epics into stories and tasks it is important to write acceptance criteria, which will make the items testable. As much testing as possible should be automated, depending on what kind of software frameworks and environments are used.

When a sprint has passed it is time for another planning meeting, and then the backlog items at high abstraction level are once again broken down into stories and tasks that can be worked with.
5.4.7 Process artifacts

As mentioned in the sections above, there are several artifacts present throughout the project inception phase, requirements engineering process and Scrum. A summary of these artifacts is presented below.

Initially the potential customer contacts vendor and requests information in a RFI document. This is replied to with product information, where the two paths traditional or Agile are presented to the customer. Preferably the customer chooses the Agile path.

Figure 39 shows the artifacts as in the case of a traditional customer. The customer then presents a Request For Quote document which asks for a proposal to the drafted requirements and wishes. These requirements are analyzed and worked up to different abstraction levels in the form of an initial product backlog. For this purpose a tool is needed, preferably that can continue to be used in the continued work with requirements in Scrum. This will be discussed more thoroughly in the tools analysis in section 5.5. When requirements have been well-structured they are transformed into a Software Requirements Specification and sent back to customer together with a Project plan that contains schedule, milestones and costs. If the customer is satisfied with project plan and SRS a formal fixed-price contract is established. The product backlog is prioritized and finally turned into a sprint backlog which the development team can begin working with.

The other option, an Agile customer as shown in figure 40 follows the same initial steps, but after receiving product information the vendor and customer will collaboratively construct a Vision statement containing overall business needs and goals. At the same time a variable-scope, target-cost contract is signed by both parties. This will allow for continued cooperation with requirements that are analyzed, broken down to lower abstraction levels and inserted into the product backlog. Following standard Scrum procedure the product backlog is prioritized and acts as a basis for the sprint backlog.
5.4.8 Special cases

There are a number special cases we have identified that deserve extra attention since they do not fit into the model straight away.

- **Non-functional requirements** are hard to put on a specific abstraction level, and often belong to several other requirements at higher levels. For example, a usability requirement might apply to several functions in a system. Our suggestion is to abstract a requirement called "Non-functional requirements" at product level, which can contain all non-functional/quality requirements at feature level. As more details about non-functional requirements arrive at lower level they can be attached to existing requirements at feature level. There can, however, be other similar situations where this approach is not applicable. If a requirement states a quality attribute, such as reliability or security, it can sometimes be “attached” to another requirement. Example: “The MySQL connection should be secured using industry standard encryption” could be written together as description text for the requirement "A MySQL database should be used for storing user information”.

- **Use cases** can be taken advantage of in two ways. They can provide an overall picture of how the system should work and can be included in the Vision statement or SRS as a part of introduction to the system, and as support for performing requirement abstraction. It can also be detailed usage scenarios at low level. It can then be included in the SRS if customer requests it, or simply provided to developers in the Agile case. The use cases will then be a help when breaking down the committed requirements in the sprint backlog once development starts.

- **Screenshots and pictures** is the same situation as use cases, but it can be meaningful to actually attach a screenshot or picture to the story it applies to. For example, a picture showing a prototype of a user interface window can be attached to the story that deals with implementing that user interface as support for the developer assigned to that story.

- **Managerial requirements**, such as legal text, delivery time and development process to use, are not handled by ARAM. These requirements must be specified in a contract, which is also what is desired by Tactel.
5.5 Tools analysis

The tools were evaluated according to an evaluation plan we constructed and can be found in appendix B. Using a standardized plan to evaluate each tool gives higher probability of treating the tools equally when running use cases. A detailed description of how the tool evaluation was performed can be found in section 2.2.5.

In the new process described in previous section tools are not mentioned specifically, but as mentioned in section 3.2.1 in the context of market-driven requirements engineering it is important to have a good place to store requirements in. The tool we are looking for should be able to act as a requirements repository. The tool evaluation was performed before the final process was constructed, so it was deemed very important to have a flexible tool that is able to adapt based on our findings.

The tools were evaluated according to a number of quality criteria (see appendix C). The complete grading is shown in appendix D and a short summary of our thoughts for each tool is provided below.

5.5.1 Hansoft

Hansoft is a Windows based tool that is currently used by Southend, which is an independent part of Tactel. Because of the close connection to Southend, Mobical decided to try the system for one sprint at least. The user interface supports the most common features expected such as drag and drop and basic import/export options. In our evaluation the system got a decent score compared to the other tools, not the worst but not the best tool, with heavy points in prioritization (best prioritization tool), being completely Scrum compliant and good plug-in capability to Bugzilla. Among the cons with this tool we can emphasize the platform independency (server and client only in Windows), plug-in capabilities other than bug reporting and lack of abstraction levels for requirements.

5.5.2 Mingle

Mingle is completely platform independent in both server and client side. The client runs in any browser and shows a very clean and attractive working area for any users. This tool got a very high average score due to its powerful customization capabilities where the complexity was helped out with the support of existing templates. Simple drag and drop, export/import capabilities and feed alerts were other features that impressed. The cons include the complexity in customizing and that it sometimes could be a little slow to use. Even if Mingle did not get the highest score rating, this was our favourite tool due to its ease of use, customizability and many good features.

5.5.3 ExtremePlanner

ExtremePlanner is a web based tool that according to their webpage focus more on features than task lists as most other management tools. In our evaluation is was rated as a decent tool with eclipse plug-in capabilities and test tracking as the better features, whereas the Scrum capability and usage affordance for our purposes were not as we would prefer. The other areas were mostly as expected by these kind of tools, but not more than that.
5.5.4 ProjectCards

ProjectCards is a platform independent client/server solution where the only thing that impressed was the built-in dashboard that looked like a Scrum whiteboard with the three To Do, In progress and Done columns. With the plug-in for eclipse and decent price tag it would qualify for testing at Mobical, but the lack of overview visualizations and other weaknesses did not impress on us. Another decent-but-not-more requirements repository.

5.5.5 Xplanner

Xplanner was the only open-source tool we evaluated. This is also its only strengths, the platform independency and price tag. In all the other aspects of our evaluation, it was weak in comparison to the other tools. Especially the Scrum compatibility was weak, it was more of an XP tool. We did not like this tool at all for our purposes.

5.5.6 VersionOne

VersionOne is a very complete client/server based tool that got the most points in our evaluation. The strengths is many usable features within the tool and at the same time having full compatibility support for other applications like Eclipse, Bugzilla and Subversion. There is support for all our specific criteria as drag and drop and project visualizations. All these features are also a part of our cons with the system. In some aspects it was hard to get the overview of all the features. Other bad things include the platform dependency to windows on both client and server side and the very high price tag.

5.5.7 Physical tools and spreadsheet

Since Scrum by the book only mentions physical tools like whiteboards and post-it notes as the primary planning and management tools, we decided that this should be included in our evaluation. To make it more fair between the physical tools and the fact that this would never be the only repository of information, we have added the functionality of simple spreadsheets to this kind of approach. Of course this is also a matter of subjectivity, but the conclusion we can make is that it was just as good as some of the other tools in our evaluation. Price tag, platform and technical independency and customizable to the team are key points of this grading. The obvious lack of plug-in capability to other systems, traceability and distributed teams issues are the cons in this approach. In the long run there could be even more issues with this approach thinking about legacy documentation and so on, but this is not taken into our judgement right now.

5.5.8 Conclusions

In the beginning of the thesis we actually thought that a good tool was the solution to most of the problems we were trying to solve. This was probably exaggerated by the presence of the, according to Mobical team, useless tool Scrumworks that was used by the team at that time. According to the criteria set up in the beginning of our tool evaluation, the two best tools were considered to be VersionOne and Mingle. They were both very complete with many features available that fit our purposes known at that time. The platform dependency is, however, not as important as we first
thought, because of several good ways of running Windows applications on other systems as well, making Hansoft (the tool currently in use) a better candidate.

After some time working with the process, we actually realized that the problems were far deeper than that of a good tool, setting the tools aside for a while. That is, for a working Scrum process to function, the tool is actually not very important. Good teamwork, collaboration and acceptance and awareness of how to work is of far greater importance. This goes back to the Agile manifesto in section 3.7 - people and interactions over processes and tools.

As we further developed our process contribution, we realized that we now are relying on having a good repository for our abstracted requirements backlog and being able to store attributes like acceptance criteria and abstraction levels on story cards. We have also identified the possibility of having requirements templates in a tools as a valuable solution to minimize the overhead work. With these newer insights in our own process model and the needs for abstraction levels in the tools, both VersionOne and Mingle become stronger candidates to be the optimal tool to use in the team.

The question of adapting a new tool is a complex matter. New tools create new ways of work to be carried out by the team, which in fact creates a slight change in the productivity early on as shown in the process improvement learning curve, figure 17. The next question is if the tool itself would create a future state of productivity that is higher than the initial. This is not an obvious matter and would probably only cause the team to get stuck somewhere in the first half of the learning curve. The actual need for a tool should be decided by the process improvement structure, and economically motivated as a contributing part of this process. The return on investment should according to this not only be calculated on the tool investment, but on the investment of the whole process improvement. In that sense, any of the evaluated and suggested tools are economically feasible for the Mobical team to adapt to in the process improvement suggested by us.
5.6 ARAM implementation guidelines

As described in section 3.6, there are certain steps that should be included when improving processes, Process change introduction being the previous sections. To be able to adapt to our new proposed process model, there are certain things that have to be changed in the teams way of working, according to the process model description. As an implementation help, we have identified some guidelines for adapting the process, a part of the process change training and initial investments. The goal after this is evolving the change tuning process into the empirical process improvement described in section [3.6.4] This is where the process is tweaked for fitting the team.

The following guidelines are essential for the new process to be implemented.

5.6.1 Documentation

Templates for System Requirements Specifications for the traditional customer should be set up. The team should all be aware about what information is available in the new SRS, and of what quality the requirements are supposed to be.

Templates for requirements should be set up to fit the teams working process. In particular, attributes should be reviewed. This should be a task for the whole team in a workshop, to be able to decide the best set of requirements attributes possible for the team. An example of a requirement template can be seen in appendix H. This template could contains a basic set of attributes and could be used as a starting point.

The product info, marketing material, usually sent to potential customers should be updated with relevant information about working Agile and how this can have big positive effects on the quality of developed software. It should also point out the positive effects with satisfying customer needs and win-win contract solutions like for example target-cost contracts.

5.6.2 Team education

Scrum workshop for the whole team. Since our process implementation requires a working Scrum environment for receiving the backlog and proceed with the work, it is essential that all team members really know the basic Scrum principles and understand the importance of all parts in the methodology.

Common language workshop for the team to settle critical words used in for example work-ups, so the team uses the same vocabulary throughout the development process. For example should everyone know what “Done” criteria and what lies behind certain document names.

Abstraction level workshop for settling the different abstraction levels in the project. This should be done at the beginning of this process implementation in a new project of environment, where the abstraction levels are fitted in the project.

Requirements attribute workshop to settle what attributes should be tagged for requirements in the project. This should be a team decision and discussion, since too much detail will confuse the team while too few attributes will restrain the work. This was done in the empirical evaluation of RAM [31].

Pilot project for implementation of the process. If the real project in the team is too complex or too crucial to be experimented with, there should be set up a pilot project for tweaking the process and
fit to the team. Preferably though, the process should be implemented as is on a real project as fast as possible.

**Team composition.** To actually be able to use our process model the extra overhead compared to before must be carried out by someone, to enable the rest of the process to run smoothly. Since our model mostly focuses on tasks that are now carried out by the Product owner, this role needs help doing all of the steps described. In the current process, the obviously similar activities are done by the testers. This is why we suggest that the Product owner should get help with the ARAM activities by other team members. At the same time the requirements are validated through reviewing by the this person/persons. The team should of course discuss what resources are best suitable for this task.

### 5.6.3 Tools

The process implementation obviously requires a tool for keeping track of the requirements, as stated in section [5.5.8](#). The most important functionality, besides being affordable and easy to use are:

**Support for abstraction levels.** Preferably is this done in a hierarchial view of epics, stories and tasks or similar. Another alternative, but not the optimal, would be to include the level as an attribute.

**Support for the attributes needed.** The requirements attributes decided in the workshop (section [5.6.2](#)) and specified in templates (section [5.6.1](#)) should be supported in a good way without having them specified too much in ordinary text boxes. This gives support for structure and better overview.

**Support for Scrum development.** More exactly it should support creating a backlog from the specified requirements. Preferably this list should be visualizable in different ways for prioritization, project and story status and task management.

As we found during our tool evaluation [5.5](#) there are many other aspects to consider when a tool should be implemented, and the important thing is actually having the team decide what other features to have. This can be done in a tools workshop deciding what tool would fit the project and team the best.
5.7 Scrum and project management issues

We identified several problems in the case study that were not related to requirements engineering, but rather to how the Scrum method is being used and what impacts this has on project management. This is not directly a part of our problem formulation described in 1.1. We still, however, feel that it is necessary to present some solutions to these issues, which are available in specific project management theory about Agile and Scrum methods.

As described in [27], there are a number of Agile smells that indicate problems in a business organization or development process. The smells found in Mobical are described in the case study (section 4.3). In addition, we have located a number of other issues also related to Scrum and project management. Brief solutions for these smells and issues will be presented below.

Visibility is connected to the "Management is surprised"-smell and can be increased by more communication and shorter sprints that allows for better and faster status reporting. Better functional testing, with acceptance criteria attached to each story, will also increase visibility.

Sprint concept is not used as it should. In Scrum each sprint should lead to a release of working software and the team members should experience the sprint as a time-boxed session of work where the team can work undisturbed and self-organized. We recommend that all team members participate in a Scrum education program, and that management is pressured to leave the team members alone during sprints. The education program would also allow for a common vocabulary to be evolved in the team. The sprint problem is related to the long sprint length currently in place. If the sprint length would be reduced to 2 weeks or even 1 week it would allow for faster feedback loops and real iterations. The sprints should always end with sprint demos, that are quite formal and include people from outside the team. This will encourage team members to actually have a working piece of software at the end of each sprint.

Cross functional team is something that should be aimed for and the current situation is a sign of the smell "Bottle-necked resources". The Mobical team consists of bright people that have experience and education to allow for adapting to multiple tasks and roles. It is still perfectly fine, in a Scrum perspective, to keep certain individuals as experts for i.e. database backend or system testing, but all team members should be able to participate in all areas of the development process. From Tactel’s side allowing people to learn outside their traditionally designated role is an investment that will pay off in the long term. The investment could be to allow testers to participate in software development courses and developers to take part in testing courses. The pay off would be a cross-functional team that is much more likely to be able to adapt to different situations. The risk of having key personnel leave the project will also be reduced. A cross-functional team is also likely to remove the "Us vs Them"-smell. At Mobical many projects are run in parallel. Often several projects need the assistance of an expert, which leads to task switching. This can also, at least partly, be resolved by a cross-functional team where experts are less likely to be needed in multiple projects.

Team size has already been reduced when the 11 people were split into two Scrum teams. Team composition, however, remains to be a problem. There are several people that are both in the Scrum project as developers and outside of it as project managers. Scrum has only a handful predefined roles: Product owner, Scrum master and team member. There are no project managers, since the project teams are supposed to be self-organizing. Tactel should investigate what advantages there are, if any, with having people as both software developers and customer contacts or sales personnel. This investigation falls outside the scope of this master thesis. What we can recommend though is discussions, and possibly education, about the importance of allowing team members to work on their own during the sprints, and make sure that changing resources will have to wait until the sprint is over.
Hundreds of **bugs** in the bug tracking tool is a serious smell that is mostly related to testing. First of all, the current situation must of course be addressed and resources should be spent on going through the bug database and investigate which bugs are still active and solve them. To prevent the situation to arise in the future automated testing, more junit tests and functional testing will locate bugs early and allow for fixes while the system is still under development. Continuous **integration** is another pattern that will help locate bugs early and raise overall quality of the released product. Adopting continuous integration relies on having an automated build process, so Mobical should investigate which parts of building remain manual. Continuous integration will also remove the need to have a “code freeze at the end of release cycles”, which is a process smell.

The most serious problem we have located is that there is **no verification** of requirements. The new requirements process we have described in section 5.4 aims to create requirements that are testable, validated and at different abstraction levels. Acceptance criteria will also be noted on each story in the product backlog, which finally makes acceptance test a verification of that the specified requirements have been implemented as working software. We recommend testing at lower levels also be strictly connected to stories and tasks, and that all component level code have corresponding unit tests.
6 Evaluation

The new requirements process suggestion is here validated through internal and external evaluations. The internal evaluation was performed during multiple iterations through numerous meetings and a workshop. The external evaluation was performed by experts in the field reviewing our work and comment.

6.1 Internal evaluation in the Mobical team

In section 2.3.1 we described how internal validation was performed and we here conclude the results.

The internal evaluation provided very helpful feedback and was used to fine tune our solution. The first draft of the process was shown quite early to the testers in Mobical and they could right away see the advantages of connecting tests to requirements. Also appreciated was the fact that requirements now are likely to be well-structured and validated to be testable. We discussed the concepts of traditional and Agile path choice and concluded that most Mobical customers would prefer the traditional method of communicating, but there would be advantages with the Agile and worth trying to persuade them.

We decided to present the same material to the Product owner and Scrum master as we had provided for the test meeting. They liked our ideas and we decided to keep building the process in the way we had initially outlined. In particular we came to think of having the choice of being Agile or traditional in relations to the customer.

The meeting we had with only the Scrum master was mainly a check that we were right on track and that the new process was something that could actually be used without influencing the Scrum process too much.

There was also a meeting with the Product owner, Scrum master and test leader present. This was a good meeting, and one of the reasons behind this was that the Product owner had just attended a Scrum course and was very motivated towards Agile thinking. We illustrated the new process using figure 32 and figure 40 as the basis for discussions. They all thought that the new process would solve problems such as requirements not being validated or verified. We also discussed the choices of Agile and traditional paths, where the most likely scenario would be the traditional path. The Product owner pointed out how he had thought about how the traditional interaction with the customer seems to influence how work is being performed at Mobical. Since the customer receives a project plan with dates for delivery it is just too easy to go into the waterfall requirements, design, code, test way of thinking even though the team is supposed to use Scrum. The Product owner stated that it is probably hard to get the customer to completely abandon the approach with RFQ and other standard documents, but also stated that there is a possibility that customers might be willing to try the Agile approach if the contract is restructured. This made us think about contracting and as a result we wrote the theory section about Agile contracts. It was also concluded that it is hard to sell a vision, whereas the Vision statement document might perhaps not be a good idea. The Scrum master commented about customer involvement and that it might lead to the unpleasant situation where the customer exploits his position to get maximum amounts of changes that he does not have to pay for and that the code will never reach stability. We responded that this is remedied by having shorter sprints and allowing customer feedback at the sprint demo but not make changes during the actual sprints. That way iterations are kept stable. The Product owner thought that it should be possible to sign the scope traditionally, but keep the project plan Agile and invite the customer to provide feedback after each sprint. This feedback led to our decision for allowing schedule to be kept variable for contracts in the traditional path of our new process. The contract should also contain an overall
description of what it is that should be built. RAM leads to a very well-structured description of the system, and could be used also in the contract by including product and feature level requirements to show a very overall picture of what the system should support. The Product owner talked about how the team is often not left alone during sprints, and this can be solved by having shorter sprint length, something the team has tried before but abandoned because the planning meeting took too long. We concluded that this should be tried again. The participants also requested that we provide quick guides to using the process. We discussed the aspect of tools since it seemed that a tool would be very helpful in structuring the requirements. The current tool in use, Hansoft, might be sufficient but had to be tested to see if it suited to the new process. Everyone was eager to try out the new process in "real life" and this led to a new workshop meeting two days later.

The final part of our internal evaluation of the new process was to apply it to a real life project. The participants were the Product owner, Scrum master, test leader and two customer project managers. The project was a rebranding of the Mobical core product and the customer was a mobile phone operator. There was an initial requirements specification and project plan available, and the requirements were quite typical for a Mobical rebranding project. Our process description for the traditional path expects the initial requirements to be present in a RFQ, but it is close enough for evaluation purposes to use a SRS. The objective of the workshop was to use the new process model on the available requirements and insert them into the Scrum tool Hansoft. The participants discussed each requirement and determined abstraction level with our help. They were then worked up to create new requirements or connect to existing ones. The result was a hierarchy of requirements in Hansoft that represented the entire system to be built. There were a number of reactions while performing this work. Using the model we located a number of redundant requirements and also a handful requirements that when analyzed were not useful at all and could be discarded. Hansoft worked quite satisfactory since it can handle requirements (or stories) in hierarchies and at different levels. It was also possible to insert attributes (see appendix [H]) into the free text field, but we realized that it would be better to have them as fields in the tools’ story listing. That way it would be possible to i.e. sort by any one of the attributes. Hansoft has support for custom columns so it should be possible to have the most attributes available there instead of the free text field. We did, however, not have time to create these fields during the workshop. The participants were overall very satisfied with the new process and could see the immediate benefits of having more well analyzed requirements that allows for good prioritization and proper verification.

To follow up the workshop, a questionnaire was sent out to the participants that filled it out and returned it to us. The questions and a summary of responses are described below:

In general, how did you feel about working with the new requirements process?
Participants expressed a positive view about working with the new process. In particular they appreciated the break down to requirements that are testable and that the requirements are separated from the sales documentation. It was also good that the process connects early project work with development.

How did it work to place requirements at different abstraction levels?
Several participants had expected it to be more difficult but realized it is not so hard and also that the process is quite well thought out. Since working with requirements will take more time than it used to do, some participants pointed out that they might have to wait with using the requirements process until the contract has been signed with the customer, to minimize unnecessary work in case the contract is not won. In most situations the customers sign the requirements as a part of the contracting, and that would mean two sets of requirements: the ones used to sign the contract and the (good) ones extracted using the new requirements process. This is not good and might affect managements’ willingness to use the new process at all. We, however, pointed out that since the Mobical projects often are very similar it might be possible to create a template and thus only have to do the major part of working up requirements and analyzing once only.
How do you think the test process will be influenced by having acceptance criteria on each story card?
All participants recognized the obvious advantages of having acceptance criteria for each requirement. It means that it is much easier to verify each story instead of just running one big acceptance test for the entire project. More work has to be carried out to learn how to best write these acceptance criteria.

Which path do you think most Mobical customers will choose? Why?
Mobical’s customers are mostly big mobile phone operators that are quite rigid and formal in their purchasing procedures so the participants agreed on that it is much more likely that the traditional path is chosen. In particular the Product owner was open to the idea of at least trying to communicate the Agile option, perhaps as a check box in the sales documentation. We also concluded that Scrum and other Agile methodologies clearly are on the rise and could soon become acceptable even in this area of business. One project manager noted that it might be possible to get OEM partners (which often are long term customer relationships) to use the Agile path. However, the Agile path will most likely require more resources from the customer, in particular if they are to participate during sprint demos etc.

How well did Hansoft perform as a tool for the new process?
There were mixed opinions about using Hansoft for the new process. Initially most had thought it would not be possible to use at all, but that turned out to be false. It was possible to structure the requirements in hierarchies of abstraction levels and quite easily move them around using drag and drop, but some parts are lacking. It includes few opportunities for following up requirements easily and can not handle test cases. It would have been very nice if the tool could do some kind of export of the data to Word format, as it then would be possible to almost automatically generate a SRS from the tool itself. Sadly, this feature was lacking. There were export capabilities, but to a rather obscure XML format.

What problems will the new requirements process solve?
In particular the participants thought the new process would make requirements verifiable but also that customer communication is improved. One team member expressed that many of the problems are not only due to a lacking requirements process, but a lack of resources in general. This is a management problem, and will not be addressed by the new requirements process. In general participants considered the new process to address the problems we had listed.

What benefits do you see with the new requirements process?
The strongest incentive to use the new process is how it will improve the support for testing and, in the end, product quality. It will also lead to validated requirements that are easier to read, less ambiguous and less likely to contain errors. The benefit of improved prioritization was also pointed out by participants as one big reason to start to use the new process.

What are the drawbacks with working with the new requirements process?
The big drawback, that cannot be prevented, is the fact that more time and resources must be spent on requirements engineering. Participants believed that the advantages, described above, will cover these extra expenses. As already discussed it might not be financially liable to use the process until a contract has been signed, but this may be solved by using a template to minimize the rework. It was noted that customers might be unwilling to have requirements structured in the way we propose, and might have their own template for requirement specifications. In that case some extra effort must unfortunately be spent on translating our requirements to the customers format.

How will your way of working change with the new requirements process?
It will take longer to start up and plan projects, but most likely this loss in time will be regained during development. The participants realized that their workload with requirements would increase, particularly while the new process is being learned.
How would you like to introduce the new process at Mobical?
One project manager pointed out that he wants really good tool support before the process can be used, but others were more optimistic of being able to introduce it with a new development project and see how it performs in the real world throughout planning and development in Scrum. It was also requested that people that will work with the new process will be given some informal documentation how to use it, such as "cheat sheet" or "quick start guide". We accepted to create some material that will make it easier to introduce the new process. Some of the participants were worried that the new process would risk a lot of unnecessary work if a well-worked SRS is produced to the customer who then rejects it and chooses another supplier. The solution would be to create a SRS using the old way of working and then either apply the new process mainly to structure requirements internally or send out a new and improved SRS for customer approval.

6.2 External evaluation

How the external evaluation was performed is described in section 2.3.2. Here we describe the answers of the questionnaire. The reviewers have had some different approaches to answering the questions, whereas we try to get the general picture in all answers rather than cite all answers. This approach was expected when the questionnaire was sent out.

To what extent does the new process solve the issues listed in sections 5.1.1 and 5.1.2?

Since the ARAM process focuses mostly on the pre-Scrum activities the answers on these questions were almost as expected. The reviewers thought most of the pre-project problems in 5.1.1 would to be solved since this is what the process is aiming the most at doing, while the process helps with only some in 5.1.2. Pendleton was really satisfied in our analysis of the problems and thought we were "spot on" many of the RE problems experienced in the industry.

Elicitation in the Mobical case is usually not performed as in a market-driven context, but as contract-driven projects. While having attributes for the requirements, the elicitation is easier but not completely solved since we do not explain detail how it is done, according to Gorschek. He also states that validation of the requirements is solved by manually processing them in placement and work-up phases, where the attributes have to be added. Document standards are solved by the project managers having to use the same type of process with the defined attributes and adapting to all requirements. Traceability is easily solved with the explicit numbering of the requirements, but also adding the attributes of sources between requirements in the model makes it even easier. Information overflow is according to Gorschek solved by using the process as a pre-Scrum screening of the requirements (which is not really our perception of the models use), but also here the numbering is said to be a way to improve some according to Bendix. Prioritization is much easier since there is traceability between abstraction levels, which gives fewer requirements to prioritize at each level (Gorschek). The overall comment from Granvik was that of investigating the prioritization part of our process, and maybe evaluate if this could be done differently for quicker prioritization. Estimation is much easier in this model according to Gorschek, since product management and developers use the same information but at different levels.

What is your opinion on allowing both traditional and Agile paths?
The general opinion about this was that it is good to have multiple options (tools/processes) to the way of working, using the best for all situations, mainly because there are always different things that play in when it comes to customers. Sometimes the best way will be one path and sometimes the other, and the experts do not see any obvious contradictions in the ways they work and proceed. As expected the Agile way is the most preferred way ("half-baked is half-baked, I prefer the Agile path" - Granvik). As we have analysed, this is also going to be the most difficult way to be able to go when it comes to customers that want a fixed-price contract. Gorschek describes that in the market-
driven perspective, as in the context RAM was built, it would be easier to go the Agile path. This would of course require a product manager that knows the market and what is expected by the end customers. Pendleton likes our initiative towards Agile contracts when describing the bad case of fixed price contracts. This is an important thing to create the incentives with the customer to start the use of the Agile path.

**Will the new process make it possible to bridge traditional requirements engineering with Scrum?**

Also here, the view of our process is very positive, and our experts do not see too much contradiction between the two. A statement from Bendix is that our process teaches Agile people to perform good and lean RE. We also get an explanation from Gorschek about traditional product management vs. project management, and how our process actually realizes both of these. Pre-scrum process being product management deriving requirements from a larger perspective and Scrum being the project receiving them, and redefining into tasks and user stories for the programmers. But product management would also need help from the team doing the estimates. It is also expected to be good to be able to work on requirements in an iterative way, both pre- and in-project like we suggested. There was one hesitant comment by Granvik about working with really scared customers, where a better way to go would be a smaller pilot project instead of our pre-project activities, to minimize the risk of wasting resources.

**Does the new process constrain "Agile philosophy"?**

Our experts from the academia had no problems with our process being Agile, while the industry experts were more reluctant. We are not doing anything that is not useful, no waste, and we are embracing change. Also here we got the parallel to product and project management by Gorschek. What is waste from a project view does not have to be waste from a product perspective, meaning that documentation to some extent is necessary for the product to be delivered completely, while the project itself is not in need of this documentation. It is important to realize these differences, and that is one part that the industry experts maybe have not taken into account. Bendix emphasizes having good people in the organization (that follow the process anyway) is better than introducing this good process (that bad people are not going to follow anyway). Granvik however, lacks the Agile approach to prioritization, and thinks our process is somehow creating too much of waste according to what is needed in the beginning of the project, not enabling the prioritization to be done with only user values (business goals). He does, however, agree that this might be the compromise that we have to make.

**How difficult do you consider the new process is to implement and use? Why?**

Our process is overall evaluated to be easy to use, but with an obvious initial learning curve and initial sacrifices in velocity. Bendix describes this as a reasonable way of working, no more nor less than what is required to make the job done anyway. And that it is described in a fairly detailed way. Granvik describes the cons of having to do more work in the start with less velocity, but in an easy-to-follow way. Gorschek also describes the needs for easy to use support manuals and tools to use within the working process. He also asks for thorough static and dynamic validations in the industry before the question really can be answered. Pendleton likes our good description and support questions presented of the abstraction levels, since this is our uttermost tool in the process. Anyway there is a request for an "exit" in this stage of the process, to avoid getting into time consuming discussions if the initial level of abstraction is not obvious.

**How do you feel about the need for a tool to support the process?**

The need for a tool is obvious for both Bendix and Granvik. Bendix emphasizes the merge problem without having for example CVS or Subversion while Granvik sees the problem of having too many requirements for keeping them on paper or post-its. Gorschek, however, discusses the need for the process itself to be good, and then implement a tool that supports this process. The tools should be integrated and tailored to fit the process, not the other way around.
From the angle of your professional expertise what problems exist in the process?
Bendix brings up questions from configuration management and asks himself if the sprint planning is anything more than a CCB meeting in a more lean way and with the Product owner present. Change requests are also handled in Agile, but without the involvement of money. The drawbacks on implementing changes are instead of money made on the number of planned requirements, that are reduced if a change accepted. By not involving money, change management is more lean. Pendleton also states the need for a working change management, as Gorschek discusses not being waste in product perspective, and that there are many stakeholders that have incentives for different things in the product perspective. Granvik and Gorschek bring here up the problems specified above, with the prioritization coming too late and the misconceptions about product and project management and their differences. Pendleton asks for a dependency analysis to be included in the process, for improved transparency and awareness within the project. He also states that there could be biases involved when creating the different levels of requirements, people creating more “selling” requirements for the ones they feel is more important. Pendleton suggests to at least create an awareness of this, and maybe include some kind of review of this activity.

Are there more special cases that can emerge that we haven’t thought about?
Gorschek does not completely agree on the fact that our process supports completely market-driven product development, since in those cases there is no customer available. In our process we can motivate this by always having a Product owner as a proxy towards the open market. This of course demands for a Product owner with complete understanding of the current situation. Our case was maybe not presented enough in the evaluated material to get this information. Granvik has some thoughts about the pre-scrum process being iterative in nature, for emergent requirements after or in between deliveries/iterations. This is something that we need to specify better in our detailed description of the process.

Are the process instructions detailed enough? What kind of help materials would you need? (checklists, diagrams etc)
Bendix thinks the process material is fairly complete with detailed descriptions for the newcomer and figures/tables to use as checklists when getting more experienced. Granvik wants more examples while Gorschek, wants more user manuals, training materials and templates and tool support. We already decided to provide more templates for story cards, guidelines for implementing the process and tool support guidelines, but these were not included in the evaluation.

How generalizable do you think the new process is?
Gorschek emphasizes the generalizability, but only for bespoke products, not for market-driven as described above. Bendix and Granvik are, however, more positive to the generalizability, describing the goal of finding the most and the right requirements for a product in a sufficient way, being both traditional or Agile.

What other comments about the new process do you have?
Granvik and Gorschek both ask for more thorough evaluation of the process to be able to evaluate its adaptability in general. The external and internal evaluations are the first steps towards this and further empirical studies is not included in our scope. Overall we got very good feedback and the experts in the evaluation have all said that it is a very interesting and good work that we have provided.
6.3 Evaluation conclusions

6.3.1 Problems solved

A number of key issues were identified during interviews and observations. These issues represent areas of interest that we believe can be improved by implementing a well-defined requirements engineering process and streamlining the existing Scrum process.

The issues that are apparent in early phases of the project, before any development has started, are presented in the table below. These are issues that will be addressed by the new requirements engineering process.

**Insufficient elicitation**
Solved by thorough analysis when doing the RAM steps. If requirements are missing they are created, while still keeping customers vision intact. Elicitation for the market-driven context is not included in the process model.

**Low validation of requirements**
There are a number of quality criteria for requirements mentioned in section 3.1.3. Requirements are validated in two ways. Firstly, just following ARAM means that requirements are analyzed so they are correct, unambiguous, consistent and traceable. They are ranked for importance when having entered the product backlog. Secondly, requirements are validated from developers and testers point of view at the standard Scrum sprint planning meeting where all team members are allowed to give feedback on requirements and clear out any problems. This also validates that requirements are verifiable. The nature of Agile development means that the requirements are modifiable and expected to be unstable. Naturally we cannot expect the requirements to be complete at the start of the project. We embrace change. The Scrum sprint planning meeting can be seen as the requirements review, in traditional RE.

**No documentation standard**
We do not solve this issue completely since we do not provide a template for creating SRS documents. The reason for this is that we have identified how many customers keep their own templates that they want their suppliers to use. ARAM does help a bit here though. By following the process model requirements are derived in the same, structured way and end up in the product backlog before being transformed into a SRS. This way it is much more likely that SRS documents have a similar appearance.

**Lacking traceability**
This issue is solved by letting pre-project requirements transform into work items in Scrum. It is always possible to trace a business goal with any work item since all requirements have connectivity upwards to requirements at higher abstraction levels.

**Information overflow**
We partly solve this by having requirements located in one repository only. There may be a SRS document created for the customer, but the development team does not really have to see it since the same requirements are present in the sprint backlog. Acceptance criteria is also present attached to the work item in Scrum, which makes testing require less extra documentation. There is still the problem of having several information sources such as a bug database and a support database. When the number of bugs has been reduced and new Agile practices, such as continuous integration, been introduced it might be possible to keep bugs only as work items.

**Problematic prioritization**
This is one of the main incentives to use ARAM. By structuring requirements at different abstraction
levels all requirements are comparable, regardless of detail level. It is then much easier to prioritize requirements in the product backlog. The backlog should always be prioritized, even if following the traditional path with a contract that states that all requirements should be implemented. Prioritization guides what requirements should be implemented first.

**Conflicting time planning**

This issue is partly solved. The Agile path supports Agile contracts, in which the scope is allowed to be variable. This way the team can work towards a deadline which will include all current functionality. If the traditional path is chosen time planning is still not satisfactory, but the schedule will be kept variable. Some customers are likely to force schedule, cost and scope fixed, which can only be achieved by lowering quality. Since this is unacceptable it is advisable to keep a time buffer before deadline that in practice can be used to make schedule variable, or at least a bit flexible. The traditional project plan will include milestones that will directly correspond to sprints in Scrum, which makes time planning match better with Scrum.

The issues that have been identified to cause problems during daily work of working in the project are mostly related to Scrum and will not be solved by our requirements engineering process in itself. We have, however, provided a number of suggestions for improvements (see section 5.7) that will address these issues as well. Below we show the Scrum and project related issues together with the evaluated solutions:

**Low visibility**

Increased by better communication and shorter sprints.

**Vague sprint concept and unsuitable sprint length**

Improved by having shorter sprint length and educating the team about Scrum.

**No cross-functional team**

Solved by educating team members so they can perform tasks within multiple areas of knowledge.

**Projects are run in parallel**

This can be partly solved by promoting the cross-functional team, but not really covered in our solution proposal.

**Team size is not ideal**

We recommended smaller team size, which can be achieved by splitting the team in two separate entities. This has already been done during our time with the project.

**Team composition**

This is not solved by our process or improvement suggestions.

**Large amount of bugs**

The situation can be improved by automating testing and having continuous integration.

**Infrequent integration**

This can obviously be solved using more continuous integration.

**No verification of requirements**

The new requirements process will create requirements that are testable and the fact that acceptance criteria will be attached to the backlog items will make requirements verified.
6.3.2 Advantages

The new requirements engineering process solves a number of the issues described in the case study and also brings a number of new advantages:

- All requirements are comparable, which makes prioritization easier.
- More structured requirements and less risk for redundancy.
- Easier to make Software Requirement Documents to follow a standardized template.
- Requirements are actively validated for quality, due to the RAM steps.
- Requirements are traceable from SRS to stories in Scrum.
- Acceptance criteria are attached to each requirement, which makes testing actually verify the agreed requirements.
- Information overflow is partly solved by introducing a common workspace and keeping requirements in the same tool from elicitation to implementation and testing.
- Time planning issues are solved by choosing the Agile path.

6.3.3 Disadvantages

We have also identified some drawbacks with introducing the new requirements engineering process:

- The process may be too expensive to use before a contract and SRS is signed.
- A tool is needed to support the process.
- It will take longer to start-up new projects since more time must be spent on requirements engineering.
7 Conclusions

This section concludes the master thesis with summaries from methodology, case study and analysis sections. We will present the most important experiences and findings and point out how the new requirements engineering process can be used to help companies increase productivity and quality.

7.1 Results

Tactel is a developer of mobile applications and provider of solutions and consulting services. The major customers are network operators and mobile handset vendors. Mobical is a project team within Tactel that works on development of a synchronization platform that allows customers to backup and synchronize personal mobile phone data to a secure centralized server. The Mobical project is the working context for this master thesis and the objective is to improve requirements engineering in an unobtrusive way. Mobical has a standard business process for contracting, specification and communicating with the customer, but uses the Agile method of Scrum for actual development and internal planning.

Our solution and contribution consists of an adaption of the Requirements Abstraction Model (RAM) that was invented by T. Gorschek. There are two different paths that can be followed, depending on the nature of the customer on the specific projects. The standard procedure would be to choose the traditional path, which means that work carried out as usual from the customer's perspective. The new option is to go for the Agile path which involves a much closer collaboration between the customer and supplier. At the same time less documentation is needed, since communication and collaboration continues throughout the project. Change is embraced instead of opposed and resisted. Even if the traditional path is chosen a solid requirements engineering process is applied, but the standard documents, such as SRS and project plan, are still constructed.

The process model follows a number of steps, regardless of if the Agile or traditional path is chosen, and these were described in detail in the analysis section. The first three steps are specify, place and work up and can be said to relate to requirements engineering. The last two steps are parts of the normal way of work with Scrum and labeled as putting into product backlog and prioritizing. The traditional path includes a number of artifacts that the customer expects, such as an SRS and Project Plan. The Agile path keeps less documentation and the most important initial document is the Vision Statement, in which the customers needs, business values and ideas are expressed on a high abstraction level. When following the traditional path initial requirements should first be analyzed according to the steps mentioned above and then placed into the product backlog. The SRS is then created by filling in the created requirements, preferably by using the abstraction levels to make sections and headers but the exact appearance depends on the document standard used. For the Agile path requirements reside only in the product backlog and collaboration with customer is close during the sprints of work.

7.2 Fulfillment of purpose

7.2.1 Goals

At the start of the project we identified a number of goals for being able to answer the research questions. Here we describe how these goals have been fulfilled.

- G1: Compilation of current academic work in the area.
• G2: Investigation of current work processes at Mobical.
• G3: Comparison of existing and reasonable priced requirements and project management tools compatible with Scrum.
• G4: Construction and formalizing of a process that can aid the work with requirements without interfering the overall project management method.
• G5: Evaluation of the implemented process at the company.

G1 has been performed extensively and the results were summarized in the theory section. We looked at many relevant areas, such as traditional requirements engineering, testing, Agile methods, Scrum and software process improvement. Theory was used as background material for our case study and solution proposal.

G2 is fulfilled by the detailed case study present in section 4. Our means for collecting information were interviews, observation and archive analysis of material such as software requirements specifications and project plans.

G3 has been reached by the tool evaluation presented in section 5.5. A number of tools were compared and rated according to criteria derived from the case study.

G4 is our main contribution in the master thesis. The new process, which we call ARAM, can be used for working with requirements before they enter the product backlog in Scrum, and will not interfere with the Scrum process in any way.

G5 can be said to be fulfilled. The new process was evaluated internally at Mobical through several meetings and one workshop where the process was tested on a real life project. We also sent out the process description for evaluation to four external experts on Scrum and requirements engineering.

7.2.2 Research questions

The research questions from the introduction section are here shown again to describe how the purpose of the master thesis was fulfilled.

• RQ1: How is it possible to bridge the traditional aspects of requirements engineering with the modern and Agile methods of Scrum?
• RQ2: Is it possible to be Agile throughout contract and planning, while still maintaining good requirements engineering practices?
• RQ3: Can product quality in Agile software development be increased by introducing a requirements engineering process?

RQ1 deals with the clashing worlds of traditional software engineering and the modern and Agile approach used by Scrum. Scrum is a methodology that does not dictate how requirements engineering is performed so it is quite easy to apply any RE process before Scrum takes over. The traditional RE process starts with elicitation and ends with the Software Requirements Specification, which Scrum considers to be the product backlog. The problem is getting requirements to arrive into the product backlog in the right form. Scrum does not decide exactly how the product backlog should be structured so we were quite free to use any model here. We decided to use RAM to structure requirements at different abstraction levels, which makes requirements appear in the product backlog
in a hierarchy. This makes prioritization of requirements much easier since they can be compared at the same level. The internal and external evaluations strengthened our opinion that it is indeed possible to bridge the traditional aspects of requirements engineering with Scrum.

RQ2 takes a look at how contracting and planning can be made while still keeping the principles of the Agile Manifesto in mind. It also relates to how contracts and customer communication can support the iterative nature of Agile software development where big up front requirements and design is to be avoided. We have realized through interviews and workshop that in the real world it is unlikely that most customers will participate in being the “Agile customer” because they want to have a fixed price contract that dictates which features that should be included when the project is completed. We have included the option of being able to let the customer have the traditional artifacts such as SRS and project plan, but still keep everything from the suppliers point of view Agile. The other alternative is when the customer wants to be Agile from the start and be more active in participating in the development project. For this path we have provided information on how to write a Vision Statement Document that contains the first early requirements. We still maintain good RE practices such as traceability, validation and prioritization of requirements. It has been difficult to evaluate RQ2 since it needs to be further researched in real world projects using real customers. We have noticed that Mobical already have started to try convincing customers choose the Agile path and use documents such as the Vision Statement. This indicates that we might be on the right track.

RQ3 is a relevant question because it asks if it really is any point in actually introduce requirements engineering when already using Agile methods for software development. We believe the answer is yes. A defined requirements engineering process will help make the requirements better in several aspects and better requirements in the product backlog will lead to less misunderstandings, rework and better estimations. While Agile methodology most likely is right in that requirements are rarely correct or stable from the start it can still provide overall higher quality by making sure good requirements are available from the start. The risk is to do too much work pre-project that might change anyway. That is why our process model encourages to keep requirements at high abstraction levels at start, and then become more detailed as project progresses. If a customer comes up with a vast amount of requirements, many of which are at detailed abstraction level, then Agile methodologies would have problems. With our process the requirements will be analyzed and structured so that it becomes clear what requirements are subsets of others. The detailed requirements can be discussed with the customer and perhaps be discarded or kept flexible (Agile). If so, then feature level requirements can be kept in the product backlog and the details are allowed to change or emerge over the iterations of the product. This is a big advantage and a reason for incorporating ARAM in Agile development.

7.3 Generalizability

The question of generalizability is interesting since it is not something we had planned for from the start of the thesis. The goal was a requirements process made specifically to fit the Mobical team and their way of working. As work progressed it became clear that the new process actually may be able to be applied generally. Several of our external aspects also believed the process to be general. Agile methods do not dictate how pre-project work should be performed, so the choice of Agile method to use does not really constrain our process. We know that the new process seems to work at Mobical and should at least function at companies with similar prerequisites, middle sized software engineering companies with contract or market driven (or both) product development.

The process model ARAM is based upon RAM, which according to its inventor needs to be tailored to each organization before it is used. ARAM is a customization of RAM, and may be subject to the same. We have not investigated this issue thoroughly to reach a conclusion about the models
generalizability. This is something that can be included as part of further evaluation and future work.

7.4 Discussion

Having the assignment of improving something as abstract as a requirements engineering process yields questions on how much focus should be spent on how to construct specific requirements or if the important area is the way of working in general. The intuitive approach would be a middle way, adapting the two for the best result and provide means for both specific requirements and the way of work. It can differ from organization to organization if the main area of concern when improving the requirements engineering process is the requirements themselves or the way of working with them. If Agile development is involved there are other perspectives that needs to be taken into account, such as how much formalized documentation or processes that can be introduced without loosing people’s motivation or effectiveness.

During the master thesis work we evaluated a number of tools, but had problems naming a clear winner. There is a classic saying called “a good workman is known by his tools” and this is often true within software engineering as well. A tool can be a blessing and a way of guiding team members and helping them to follow a process. It could also be a curse if its hard or annoying to use. Our suggested requirements process does need a tool to be effective, but serious consideration needs to be taken before choosing one.

When stating an improvement to be carried out, there is also a postulate of a process already being used. In our case the development process was carried out with Scrum, but the provided solutions are actually only partly improvements to this. We mainly bring a new contribution to the work process carried out with requirements, which we hope will improve following the basic business strategies of providing products of good quality. There was an existing requirements engineering process when we started, but it was not defined, not repeatable and differed greatly depending on who was in charge of the specific project. Our improvement was in reality a completely new process. The question is whether the organization is mature enough to actually start using a formalized requirements process, in particular when there are several people involved in requirements engineering at Mobical and they have different background and view on requirements.

The Agile manifesto and its principles derives from research and real life experience from experts of software development. It is a collection of best practices that by some are described to be “rational” and “obvious” way of working. Some of the most important aspects in Agile methods is to have an iterative process that is lightweight and easy to follow. As we have seen in this report it is sometimes necessary to introduce formal processes or documentation even in an Agile environment. Our solution was to put a process in front of the Agile counterpart, which means that we do not influence the daily way of working. Another solution, which we have barely considered (because of the premises of the thesis) would be to instead introduce the formality inside the Agile process. Then requirements engineering would be something that has to be considered when performing daily work and there might have to be rules and guidelines for handling work items in this regard. We believe it is better to let requirements engineering be applied to work items before they enter the Agile process. In other software engineering areas this might not be possible. For example, the field of Configuration Management has practices that need to be applied in-project, for example rules or strategies when committing or merging source code in the code repository. Putting demands on how to perform work inside the Agile process will in some ways most likely cripple it and make it slower and less likely to be able to embrace change. There is a fine balance between harvesting rewards of formal processes and impacting the advantages of using Agile development.

122
7.5 Future work

During this master thesis we noticed that there are many areas of interest that fell out of the scope that we were able to include. These areas can be further academic work in the future to investigate, in other master theses or perhaps research by i.e. PhD students.

First of all we would like to point out the need for **further evaluation of our process model** using real life projects. These could include both market driven or contract driven contexts and a comparison between the two would be very interesting. We also do not know if the model scales well and if there is an upper size limit where the steps will prove to be too cumbersome.

Another area of research that we have only touched very lightly is **testing in Scrum**. This would be a very interesting master thesis that could investigate how Agile testing differs from traditional testing and if traditional testing techniques still can be used. Agile testing focuses a lot on keeping tests as automated as possible (for use with continuous integration) and one part of the thesis could be to compare different frameworks for automated testing at different levels of detail, i.e. unit testing, function testing and acceptance testing. This relates to our thesis and the result that we want to keep requirements at different abstraction levels.

Something that needs further research is the aspect of contracting in an Agile context. We described this briefly in our theory section and one part of our solution allows contracts to be quite informal, but how should the contracts be written in practice? The traditional fixed-price contract is not desirable in Agile development so something else is needed. **Agile contracts** could be an interesting master thesis for engineering students following the Technology Management program or business students with an interest in technology and software business.
References


Appendices

A Interview questions

The structure of the interview is to closely follow these questions but let the participant talk as much as possible. Semi-structured interviewing. The parts called overview, basic requirements engineering, SCRUM and summary are answered by everyone, while the two other parts are directed to product management and developers/testers respectively.

Overview

1. Name:
2. Title:
3. Tell us about your position in the company and what you do in projects
4. Tell us about the Mobical projects (time on market, typical customer/end-user, size of product projects, etc)
5. How many projects is Mobical carrying through annually?
6. How many other projects have you participated in?
7. What kind of different development processes have you worked with/experienced?

Basic requirements engineering

8. From your point of view, how is a product developed? (kind of products, activities performed, documentation developed, special cases, evaluations performed, etc)
9. What is a requirement to you?
10. What is a "good requirement" to you? And to the company? Is the quality of the requirements assessed? How?
11. What challenges do you face when working with requirements? What has been successful regarding requirements engineering?
12. What needs are there for requirements traceability? That is, how requirements, customer needs and work products are linked. How can this make your work easier?
13. What kind of dependencies between the requirements have you come across? Are the dependencies documented? Are dependencies actively looked for? How do they affect product development? How is it handled?

Scrum

14. Do you have any description of the SCRUM process, any instructions how to handle the process? If there are:
(a) How old are the instructions?
(b) What changes have been made since the start?

15. How do you know when a task is ‘done’? Who decides?

16. What do you think about the sprint length used now? Should it be longer? shorter? Fit to project deadlines?

17. Is the SCRUM process easy to follow? As it is implemented in Mobical?

18. How are stories and tasks related to requirements? Are these relations documented?

19. Are tasks/stories changed throughout the development process? If so, how are you responding to this? ex. are test documents updated?

Current requirements process


21. How much resources are spent on requirements engineering? Continually or in the beginning? How much time would be optimal? Which roles are involved in the process?

22. How are requirements documented? What information and attributes are documented about the requirements? (Traceability, requirement owner/manager)

23. Who are the stakeholders/readers of the SRS/FRS (System Requirements Specification)? How do you use it?

24. What support and what tools do you use in the requirements engineering process (task management, development, testing etc)? On a daily basis? What pros and cons do these tools have?

25. How many requirements are handled in a typical project?

26. Do you group the requirements? How are the requirements groups handled during development?

27. How do you handle non-functional requirements such as performance or stability?

28. How are the requirements prioritized? What is difficult when deciding what to include in the product? Is there any following up on the prioritising of the requirements? How is the order of implementation decided?

29. How are change requests handled in a project? How is this communicated within the team?

30. What kinds of decisions are taken during the development of a product? What kind of support is needed in those decisions? What is the effect of a very late/too late decision?

31. How are the requirements validated? (How do you know a requirement is “good”?)

32. When and from where is the acceptance criteria involved in the process? What roles are affected?

33. Where in the current process is testing performed? Function test, Acceptance test?

34. How are test cases related to requirements? Are these relations documented?
35. How are bugfixing activities prioritized? Who decides?

Summary

36. What would you like to change in the development process if you had enough time and money? Tools? Process?

37. How could a tool support the requirements engineering process?

38. Do you see any other general problems or difficulties with the development process?

39. Prioritize which problems that are most urgent to be solved and which possibilities that would be most valuable to carry through.

40. Do you have any ideas on how to improve product process in Mobical in general?
B Tools evaluation plan

Evaluation plan

Introduction

This document describes evaluation of different project management tools for use in a Scrum-based environment. It is a part of our thesis work with the goal of fitting a tailored process model to the existing project of Mobical. As a part of this process improvement, our interviews pointed out that possibly a new and better tool would improve and support a new process implementation. Since the most important thing is the process itself and the way of working, we have to be extremely cautious about how the tool can help, and how it should be used. A part of the process implementation is to get it accepted by the people working with it, and if the tool that is supposed to support this is bad to use, the tool itself can create impediments in the work. This is why we have to evaluate the tools in a good way, according to the quality criteria and wishes found during interviews.

Quality criteria

The most important criteria as requested by Tactel are:

- Platform independent
- Can show project overview
- Has a repository for requirements
- Can prioritize stories and tasks
- Integrated with eclipse-plugin if possible

Additionally, we have identified the following necessary features:

- Support for the common aspects of Scrum
- Can handle requirements and testing
- Support for "abstraction levels"
- Good support for bug management, or connection to BugZilla
- The licensing price tag should be acceptable for Mobical
# Evaluation template

<table>
<thead>
<tr>
<th>Tool name</th>
<th>Date evaluated</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - Cost</td>
<td></td>
</tr>
<tr>
<td>1 - Installation</td>
<td></td>
</tr>
<tr>
<td>2 - Administration</td>
<td></td>
</tr>
<tr>
<td>3 - Product backlog</td>
<td></td>
</tr>
<tr>
<td>4 - Prioritization</td>
<td></td>
</tr>
<tr>
<td>5 - Sprint backlog</td>
<td></td>
</tr>
<tr>
<td>6 - Daily work</td>
<td></td>
</tr>
<tr>
<td>7 - Daily Scrum</td>
<td></td>
</tr>
<tr>
<td>8 - Project status</td>
<td></td>
</tr>
<tr>
<td>9 - Report bug</td>
<td></td>
</tr>
<tr>
<td>10 - Test cases</td>
<td></td>
</tr>
<tr>
<td>11 - Change management</td>
<td></td>
</tr>
<tr>
<td>12 - Plug-in capability</td>
<td></td>
</tr>
<tr>
<td>13 - Version control</td>
<td></td>
</tr>
<tr>
<td>14 - Reporting</td>
<td></td>
</tr>
</tbody>
</table>

Additional comments:
Task descriptions

We will follow these descriptions when evaluating the tools:

<table>
<thead>
<tr>
<th>Task:</th>
<th>0) Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Evaluate if the price tag is feasible for mobical</td>
</tr>
<tr>
<td>Trigger:</td>
<td>-</td>
</tr>
<tr>
<td>Frequency:</td>
<td>Startup and Annual costs</td>
</tr>
<tr>
<td>Critical:</td>
<td>Price should be reasonable</td>
</tr>
<tr>
<td>Completion:</td>
<td>Costs for mobical group are calculated</td>
</tr>
</tbody>
</table>

0.1) Costs derived from homepage
0.2) Calculation of startup and annual costs
0.3) Costs are used when evaluating if the tool is possible to implement

Task: 1) Installation

Purpose: Make sure the software is installed and operational.
Trigger: Software is acquired.
Frequency: Typically, once.
Critical: -
Completion: Software is installed and working.

1.1) Download software
1.2) Read installation documentation
1.3) Prepare target platform
1.4) Perform installation

Task: 2) Administration

Purpose: Create users and sample projects
Trigger: Software is installed.
Frequency: Once per month.
Critical: Many new projects at the same time.
Completion: Users and projects created.

2.1) Login as administrator
2.2) Create 3 users: developer, productowner, tester
2.3) Create 2 projects: mobicalcore, customercustomization
### 3) Fill product backlog

<table>
<thead>
<tr>
<th>Task:</th>
<th>3) Fill product backlog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Create stories from business goals</td>
</tr>
<tr>
<td>Trigger:</td>
<td>Admin user is available</td>
</tr>
<tr>
<td>Frequency:</td>
<td>Frequently during project</td>
</tr>
<tr>
<td>Critical:</td>
<td>Many items, overview</td>
</tr>
<tr>
<td>Completion:</td>
<td>Backlog created</td>
</tr>
</tbody>
</table>

3.1) Login as product owner  
3.2) Extract 10 stories from existing contract document  
3.3) Check the overview  
3.4) Attach file to one story

### 4) Prioritize backlog

<table>
<thead>
<tr>
<th>Task:</th>
<th>4) Prioritize backlog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Prioritize stories in the backlog according to business value</td>
</tr>
<tr>
<td>Trigger:</td>
<td>Product backlog filled with stories</td>
</tr>
<tr>
<td>Frequency:</td>
<td>Once every sprint</td>
</tr>
<tr>
<td>Critical:</td>
<td>Many items from different projects</td>
</tr>
<tr>
<td>Completion:</td>
<td>All items prioritized</td>
</tr>
</tbody>
</table>

4.1) Login as product owner  
4.2) Select product backlog  
4.3) Prioritize all stories

### 5) Create sprint backlog

<table>
<thead>
<tr>
<th>Task:</th>
<th>5) Create sprint backlog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Select items to include in the next sprint</td>
</tr>
<tr>
<td>Trigger:</td>
<td>Backlog prioritized</td>
</tr>
<tr>
<td>Frequency:</td>
<td>Once every sprint</td>
</tr>
<tr>
<td>Critical:</td>
<td>Many items from different projects</td>
</tr>
<tr>
<td>Completion:</td>
<td>Enough items selected to facilitate for a sprints work</td>
</tr>
</tbody>
</table>

5.1) Login as product owner  
5.2) Create new sprint  
5.3) Select 8 items to include in the sprint
### 6) Daily work as developer

<table>
<thead>
<tr>
<th>Task:</th>
<th>Purpose: Daily work with the tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger: Backlogs created correctly</td>
<td></td>
</tr>
<tr>
<td>Frequency: Daily use</td>
<td></td>
</tr>
<tr>
<td>Critical: Ease of use, missing features</td>
<td></td>
</tr>
<tr>
<td>Completion: Satisfying UI and complexity. Satisfying reporting possibilities</td>
<td></td>
</tr>
</tbody>
</table>

6.1) Login as developer  
6.2) Choose a story to work with  
6.3.) Check that the story is included in "My work" or similar  
6.4) Change effort left

### 7) Daily Scrum

<table>
<thead>
<tr>
<th>Task:</th>
<th>Purpose: Manage report from daily Scrum meeting with team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger: A daily Scrum meeting has been performed</td>
<td></td>
</tr>
<tr>
<td>Frequency: Once every work day</td>
<td></td>
</tr>
<tr>
<td>Critical: No Scrum meeting held, too much decided</td>
<td></td>
</tr>
<tr>
<td>Completion: The daily Scrum meeting logged</td>
<td></td>
</tr>
</tbody>
</table>

7.1) Login as scrum master  
7.2) Select daily Scrum or similar  
7.3) Log that 1 developer is impeded and 1 tester is home sick

### 8) Project status

<table>
<thead>
<tr>
<th>Task:</th>
<th>Purpose: Show status of a project as how much work is left</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger:</td>
<td></td>
</tr>
<tr>
<td>Frequency: Several times per week</td>
<td></td>
</tr>
<tr>
<td>Critical: People haven't updated their tasks</td>
<td></td>
</tr>
<tr>
<td>Completion: Burndown chart visible</td>
<td></td>
</tr>
</tbody>
</table>

8.1) Login as developer  
8.2) Select customer customization project  
8.3) Show burndown chart for a specific task

135
8.4) Show burndown chart for entire project

<table>
<thead>
<tr>
<th>Task:</th>
<th>9 ) Report bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Bug management, trace to requirements</td>
</tr>
<tr>
<td>Trigger:</td>
<td>Bug found in system</td>
</tr>
<tr>
<td>Frequency:</td>
<td>Often</td>
</tr>
<tr>
<td>Critical:</td>
<td>Missing reporting functionality</td>
</tr>
<tr>
<td>Completion:</td>
<td>Bug reported and tracked to requirement</td>
</tr>
</tbody>
</table>

9.1) Login as developer
9.2) Select customercustomization project
9.3) Report bug, include customer bug ID
9.4) Choose to connect it to a story or task
9.5) Check connectability with other bug reporting tools

<table>
<thead>
<tr>
<th>Task:</th>
<th>10) Create test cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Add test cases connected to a specific story</td>
</tr>
<tr>
<td>Trigger:</td>
<td>-</td>
</tr>
<tr>
<td>Frequency:</td>
<td>Several times per week</td>
</tr>
<tr>
<td>Critical:</td>
<td>Unclear stories</td>
</tr>
<tr>
<td>Completion:</td>
<td>Burndown chart visible</td>
</tr>
</tbody>
</table>

10.1) Login as developer
10.2) Select customercustomization project
10.3) Show burndown chart for a specific task
10.4) Show burndown chart for entire project

<table>
<thead>
<tr>
<th>Task:</th>
<th>11) Change management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Be able to manage changes in a good way</td>
</tr>
<tr>
<td>Trigger:</td>
<td>Incoming change request</td>
</tr>
<tr>
<td>Frequency:</td>
<td>Several times per sprint</td>
</tr>
<tr>
<td>Critical:</td>
<td>Dependancies, regression testing.</td>
</tr>
<tr>
<td>Completion:</td>
<td>Visualize changes</td>
</tr>
</tbody>
</table>

12.1) Change request
12.2) Change affected story/task
12.3) Evaluate the visualization of the change made
<table>
<thead>
<tr>
<th>Task</th>
<th>12) Plug-in capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Eclipse plug-in evaluation</td>
</tr>
<tr>
<td>Trigger:</td>
<td>-</td>
</tr>
<tr>
<td>Frequency:</td>
<td>All the time</td>
</tr>
<tr>
<td>Critical:</td>
<td>Integration with IDE</td>
</tr>
<tr>
<td>Completion:</td>
<td>Plug-in shown in eclipse</td>
</tr>
</tbody>
</table>

12.1) Install plugin
12.2) Configure plug-in for project
12.3) Open eclipse and evaluate plug-in
12.4) Look at what is shown in eclipse compared to the tool view

<table>
<thead>
<tr>
<th>Task</th>
<th>13) Support for version control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Document and code version control</td>
</tr>
<tr>
<td>Trigger:</td>
<td>-</td>
</tr>
<tr>
<td>Frequency:</td>
<td>All the time</td>
</tr>
<tr>
<td>Critical:</td>
<td>To enable working with the latest release of everything</td>
</tr>
<tr>
<td>Completion:</td>
<td>Enable SVN or CVS, visualize changes</td>
</tr>
</tbody>
</table>

13.1) Login as developer
13.2) Update a story with new information
13.3) See if it’s possible to revert back to old version
13.4) See if it’s possible to see difference between versions

<table>
<thead>
<tr>
<th>Task</th>
<th>14) Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Generate a project report</td>
</tr>
<tr>
<td>Trigger:</td>
<td>-</td>
</tr>
<tr>
<td>Frequency:</td>
<td>Mostly once per sprint</td>
</tr>
<tr>
<td>Critical:</td>
<td>-</td>
</tr>
<tr>
<td>Completion:</td>
<td>Project report generated</td>
</tr>
</tbody>
</table>

14.1) Login as scrum master
14.2) Go to project reporting, or similar
14.3) Generate report containing following info:
14.3.a) Effort left on stories per user
14.3.b) Impediments
14.3.c) Status for all stories
14.3.d) Complete project effort left
14.3.e) Bug status
### C Quality criteria for tools

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform independent</td>
<td>Server platform - not as important as the client</td>
</tr>
<tr>
<td>Can show project overview</td>
<td>Client platform</td>
</tr>
<tr>
<td>Reporting possibilities</td>
<td>Visualizations</td>
</tr>
<tr>
<td>Has a repository for requirements</td>
<td>Database visualization</td>
</tr>
<tr>
<td>Has a repository for requirements</td>
<td>Customized card types</td>
</tr>
<tr>
<td>Can prioritize stories and tasks</td>
<td>Easy to use prio</td>
</tr>
<tr>
<td>Integrated with eclipse-plugin if possible</td>
<td>Prio visualization in the rest of the project</td>
</tr>
<tr>
<td>Integration with eclipse-plugin if possible</td>
<td>General plugin and connectivity</td>
</tr>
<tr>
<td>Integration with eclipse-plugin if possible</td>
<td>Import/export to xls = 1</td>
</tr>
<tr>
<td></td>
<td>Full eclipse support = 5</td>
</tr>
</tbody>
</table>

Table 15: The most important criteria requested by Tactel

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for the common aspects of Scrum</td>
<td>Visualization of the Scrum process</td>
</tr>
<tr>
<td></td>
<td>Scrum terminology supported</td>
</tr>
<tr>
<td>Can handle requirements and testing</td>
<td>Supporting Scrum workflow</td>
</tr>
<tr>
<td>Integration with eclipse-plugin if possible</td>
<td>Traceability to test cases</td>
</tr>
<tr>
<td>Integration with eclipse-plugin if possible</td>
<td>In task vs own task documentation of tests</td>
</tr>
<tr>
<td></td>
<td>Test visibility</td>
</tr>
<tr>
<td>Integration with eclipse-plugin if possible</td>
<td>Customized card types</td>
</tr>
<tr>
<td>Support for &quot;abstraction levels&quot;</td>
<td>Story, task, epic stories, card connections other than story-task</td>
</tr>
<tr>
<td>Good support for bug management, or connection to BugZilla</td>
<td>Plugin-capability</td>
</tr>
<tr>
<td>Integrated with eclipse-plugin if possible</td>
<td>General plugin and connectivity</td>
</tr>
<tr>
<td></td>
<td>General traceability</td>
</tr>
<tr>
<td></td>
<td>Bug reporting within tool</td>
</tr>
<tr>
<td>The licensing price tag should be acceptable for Mobical</td>
<td>Graded according to a subjective scale. Free=5, &gt;50000SEK/year = 1</td>
</tr>
</tbody>
</table>

Table 16: Additional criteria needed according to our preferences
<table>
<thead>
<tr>
<th>Quality Criteria</th>
<th>Hansoft</th>
<th>Mingle</th>
<th>AtlassianPlan</th>
<th>ProjectCards</th>
<th>AgilePlan</th>
<th>VersionOne</th>
<th>Physical tools + spreadsheet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affordance: Likelihood of being used</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Platform independent</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Can show project overview</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Has a repository for requirements</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Can prioritize stories and tasks</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Integrated with eclipse plug-in if possible (score/report do=?)</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Support for the common aspects of Scrum</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Can handle requirements and testing</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Support for “abstraction levels”</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Good support for bug management, or connection to BugZilla</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>The following print log should be acceptable for Merced</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Price/year</td>
<td>9700/year, 7500 next</td>
<td>17000</td>
<td>11200, 3500, one-time fee</td>
<td>0</td>
<td>27500/year, 28000 next</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SUMMARY</td>
<td>36</td>
<td>40</td>
<td>35</td>
<td>33</td>
<td>29</td>
<td>41</td>
<td>34</td>
</tr>
</tbody>
</table>
E Sample Software Requirements Specification

Included is a portion of sample SRS that has been used in a project at Tactel. The identity of the customer has been masked for confidentiality reasons. The sample shows requirements for the e-mail subsystem.

4 Functional requirement

4.1 Following functionality describes in detail the in Scope Requirement ID: New  for Webportal:

The system shall support Microsoft Internet Explorer, Netscape Navigator, Opera and Mozilla.

The main browsers for acceptance testing will be Microsoft Internet Explorer and Mozilla.

For sync, there shall be an interface provided, that allows customers to set the default rule "client wins" to "server wins" in the self-care UI.

E-Mail

- Send/Receive/view/Forward/Reply email
- Receive/View/Forward voice, Fax and Video Mail (to view attachments like videos, images etc. they must be downloaded to the local computer)
- Delete e-mail/move/resize/make/move to folder or move to folder
- Create, delete and rename folders and move any kind of messages from one folder to another
- Persistent folders shall be inbox sent items, deleted items, spam, draft, which cannot be deleted or renamed
- Selected emails or all emails deleted from the SPAM folder shall be deleted permanently. There shall be a restore selected functionality for the SPAM folder.
- Restored emails will end up in inbox folder.
- Trashcan folder shall have the possibility to empty trashcan and to restore selected mails. Restored emails will end up in the inbox folder.
- Move selected emails to folder
- Refresh inbox screen
- Use of and view attachment (to view attachments like videos, images etc. they must be downloaded to the local computer)
- Read HTML mail:
  - HTML mails shall be readable and all pictures will be shown. Javascript code will be removed from the email prior to presentation.
- Reply/Forward HTML mail:
  - HTML mail must be editable and stays in HTML format. To start with replied/forwarded HTML emails, you will be asked as an attachment to the new plain text email message.
### 3.8 Technical Features

<table>
<thead>
<tr>
<th>ID</th>
<th>Short description</th>
<th>Long Description</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>SMS-triggered startup</td>
<td>It should be possible to start up the mobile client by sending an SMS to the terminal (for example, by using the Wireless Message API for Java MIDP)</td>
<td>M</td>
</tr>
<tr>
<td>60</td>
<td>Autostart when starting terminal</td>
<td>It should be possible to configure the mobile client to automatically start up when the terminal is switched on.</td>
<td>M</td>
</tr>
<tr>
<td>61</td>
<td>Home or roaming detection</td>
<td>The mobile client shall be able to detect whether the end-user is in home network or roaming in a foreign network</td>
<td>M</td>
</tr>
<tr>
<td>62</td>
<td>Detect de-installation</td>
<td>The server shall be able to detect whenever a mobile client is uninstalled. This is for statistics and to prevent unnecessary notifications/remote mobile client startup.</td>
<td>M</td>
</tr>
<tr>
<td>63</td>
<td>Battery life</td>
<td>The battery life time shall not be noticeably affected when using the Mobile client – the Tenderer shall describe how this is achieved</td>
<td>M</td>
</tr>
<tr>
<td>64</td>
<td>Valid SIM subscription</td>
<td>The mobile client shall only be able to connect to the server when the terminal has a valid SIM/subscription</td>
<td>M</td>
</tr>
<tr>
<td>65</td>
<td>APN profile setting</td>
<td>The Network Access Point (APN) profile for the mobile client to use must be pre-configurable and/or “auto-configurable”, i.e. the user shall not be forced to choose APN profile each time the mobile client is started, except the very first time. The choice made at that time shall be stored in the configuration settings.</td>
<td>M</td>
</tr>
<tr>
<td>66</td>
<td>Configuration after installation</td>
<td>It must be possible to manually configure the mobile client (APN profile to use and other relevant configurable elements) after installation of the mobile client (e.g. using a configuration file or registry settings)</td>
<td>M</td>
</tr>
<tr>
<td>67</td>
<td>GPRS connection</td>
<td>The mobile client must be able to communicate with external systems using a GPRS (packet switched) connection.</td>
<td>M</td>
</tr>
<tr>
<td>68</td>
<td>Open protocols</td>
<td>The communication between the mobile client and the server shall be based on open-standart protocols</td>
<td>M</td>
</tr>
</tbody>
</table>
G Requirements Abstraction Model guides

Figure 41: RAM how-to guide part 1: Initial placement of requirements
Figure 42: RAM how-to guide part 2: Abstraction and breakdown of requirements [32]
H   Story card template

Description.......: xxxx
Level..............: x
Business value....: xxxx
Source.............: xxxx
Owner..............: xxxx
Status...............: xxxx
Reject reason.....:
Accept. criteria..:
# xxx
I Internal evaluation questionnaire

- In general, how did you feel about working with the new requirements process?
- How did it work to place requirements at different abstraction levels?
- How do you think the test process will be influenced by acceptance criteria on each story card?
- Which path do you think most Mobical customers will choose? Why?
- How well did Hansoft perform as tool for the new process?
- What problems will the new requirements process solve?
- What benefits do you see with the new requirements process?
- What are the drawbacks with working with the new requirements process?
- How will your way of working change with the new requirements process?
- How would you like to introduce the new process at Mobical?
J External evaluation questionnaire

- To what extent does the new process solve the issues listed in 5.1.1 and 5.1.2?
- What is your opinion on allowing both traditional and Agile paths?
- Will the new process make it possible to bridge traditional requirements engineering with Scrum?
- Does the new process constrain "Agile philosophy"?
- How difficult do you consider the new process is to implement and use? Why?
- How do you feel about the need for a tool to support the process?
- From the angle of your professional expertise what problems exist in the process?
- Are there more special cases that can emerge that we haven’t thought about?
- Are the process instructions detailed enough? What kind of help materials would you need? (checklists, diagrams etc)
- How generalizable do you think the new process is?
- What other comments about the new process do you have?
ARAM Process Guide

1. Include Agile information in product information. (Inception)
2. Contract based on as much Agile values as possible. (Inception)
3. Create Vision statement or RFQ with customer from template. (Elicitation)
4. Follow ARAM Requirements Guide.
5. Extract SRS from the worked-up requirements for the customer.
6. Develop prioritized backlog items according to Scrum values.
7. Acceptance criteria is the basis for testing.
ARAM Requirements Guide

1. **Elicit** from input source - RFQ, Vision statement, SRS etc.  
   *Attributes: Title, status, description, value, source*

2. **Initial placement** according to the question:  
   *What level to place requirement?*  
   - Is the requirement comparable to product strategy?  
   - Does the requirement describe what the system should include/support?  
   - Is the requirement functional and testable?  
   - Does the requirement suggest how to solve something?  
   *Attributes: Abstraction level, ID, owner*

3. **Work-up** for all requirements. All requirements chains should at least be represented at least at Product and Feature levels. Create new and connect already existing ones.  
   *Attributes: Source, abstraction level*

4. **Insert** into product backlog and add acceptance criteria and rough estimates.  
   *Acceptance criteria = Testable*  
   *Attributes: Acceptance criteria, estimate*

5. **Prioritize** backlog according to value and estimate.

### Attribute title | Description
--- | ---
Title | Text for labeling the requirement.
Description | Text with an informative description.
Level | Which abstraction level this requirement is at.
ID | An identification tag in the form a.b.c.d
Value | Text for rationale or benefit behind the requirement.
Source | From where does this requirement derive.
Owner | Who is responsible for this requirement.
Status | Proposed  
Approved  
Rejected  
Implemented  
Verified
Reject reason | (only if rejected)