Aalto University
School of Science and Technology
Faculty of Electronics, Communications and Automation
Degree Programme in Automation and Systems Technology

Ilkka Lehto

# Using backlogs for linking long-term product plans and development tasks in agile software development

Master's Thesis

Espoo, May 5, 2010

Supervisor: Professor Tomi Männistö, D.Sc. (Tech.)
Instructor: Kristian Rautiainen, Lic.Sc. (Tech.)

Aalto University
School of Science and Technology

| | |
|---|---|
| Aalto University<br>School of Science and Technology<br>Faculty of Electronics, Communications and Automation<br>Degree Programme in Automation and Systems Technology | ABSTRACT OF THE MASTER'S THESIS |

Author: Ilkka Lehto

Title: Using backlogs for linking long-term product plans and development tasks in agile software development

| | | |
|---|---|---|
| Number of pages: v + 51 | Date: 5.5.2010 | Language: English |
| Professorship: Software engineering | Code: T-76 | |

Supervisor: Professor Tomi Männistö, D.Sc. (Tech.)

Instructor(s): Kristian Rautiainen, Lic.Sc. (Tech.)

Abstract:

Agile software development has been used for more than a decade in industry. Scrum, a popular agile method, introduces backlogs as a lightweight approach for managing software requirements and work tasks. The agile methods were originally intended for managing software development of a single team. However, large companies have started to adopt agile methods in large-scale development i.e. in a context where multiple teams and contributing to the same product simultaneously.

In this thesis the applicability of communicating long-term product plans to software developers and communicating progress information back to management, was studied in the context of large-scale agile software development. Existing research of large-scale agile software development is scarce and the role of backlogs has gained even lesser attention. However, some initial suggestions on how to manage backlogs in a multi-team setting were found.

The research problem was studied by conducting a case study in a large Finnish software product company. Research data was collected by conducting interviews and observations. The case company was given constant feedback and the case company reacted to some of the identified challenges. A framework that describes one possible way to link long-term plans with daily tasks, was constructed based on literature. The framework was used to analyze the results and also to collection of the research data.

Challenges were identified within the case company in the organizational structure, planning and progress monitoring practices and also in the tools they used. The case company took a number of corrective actions: An organizational rearrangement cleared off communication barriers, which in turn facilitated better planning practices; The backlogs were also renewed, which stressed the tools and that they had to be also changed.

Successful backlog management cannot exist without a well functioning organization, effective work practices and proper tools.

Keywords: agile, software development, backlog, product management, case study

Aalto-yliopisto
Teknillinen korkeakoulu
Elektroniikan, tietoliikenteen ja automaation tiedekunta
Automaatio- ja systeemitekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Ilkka Lehto

Työn nimi: Työlistojen käyttö pitkän aikavälin tuotesuunnitelmien ja työtehtävien kytkemiseksi toisiinsa ketterässä ohjelmistokehityksessä

Sivumäärä: v + 51 | Päiväys: 5.5.2010 | Julkaisukieli: Englanti

Professuuri: Ohjelmistotuotanto | Professuurikoodi: T-76

Työn valvoja: Professori Tomi Männistö, TkT

Työn ohjaaja: Kristian Rautiainen, TkL

Tiivistelmä:

Ketterä ohjelmistokehitys on yli kymmenvuotisen olemassaolonsa aikana saavuttanut suosiota ohjelmistoyritysten keskuudessa. Scrum, eräs suosittu ketterän ohjelmistokehityksen menetelmä, esittää ohjelmistotuotteen vaatimusten ja työn hallintaan käytettäväksi yksinkertaisia työlistoja. Alunperin yhden tiimin työn hallintaan kehitettyjä menetelmiä on alettu soveltaa myös suuressa mittakaavassa eli tilanteissa, joissa useat ohjelmistokehitystiimit työskentelevät yhtäaikaisesti saman ohjelmiston parissa.

Tässä diplomityössä selvitettiin työlistojen hyödyntämistä ja soveltuvuutta pitkän tähtäimen tuotesuunnitelmien kommunikointiin ohjelmistokehittäjille sekä ohjelmistokehityksen edistymisen seurantaan. Ketterää ohjelmistokehityksen käyttämistä suuressa mittakaavassa on tutkittu vain vähän ja työlistojen roolia strategian jalkauttamisessa tuskin lainkaan. Kirjallisuuskatsauksessa löydettiin kuitenkin muutamia ehdotuksia siitä, kuinka työlistoja voisi hyödyntää, kun tiimejä on hallittavana useita.

Tutkimusongelmaa selvitettiin tapaustutkimuksella suuressa suomalaisessa ohjelmistotuotteita kehittävässä yrityksessä. Tutkimusaineisto kerättiin haastatteluilla ja havainnoinnilla. Tutkimuksen tuloksista annettiin yritykselle palautetta, jota käytettiin apuna yhteistyön suunnittelussa ja siten edelleen tutkimuksen ohjauksessa. Havaintojen analysointiin sekä aineiston keräämisen suunnitteluun käytettiin kirjallisuuden perusteella tehtyä viitekehystä, joka esittää erästä mahdollista tapaa käyttää työlistoja strategian ja päivittäisten toimien kytkemiseksi toisiinsa.

Tutkimuksessa löydettiin haasteita niin organisaatiorakenteesta, suunnittelun ja seurannan työtavoista, että käytetyistä työkaluista. Tapausyritys ryhtyi useisiin korjaaviin toimenpiteisiin: Organisaatiouudistuksella luotiin edellytykset toimivalle kommunikaatiolle ja suunnittelukäytäntöjen täysipainoiselle hyödyntämiselle; Työlistat kokivat myös reformin, mikä aiheutti paineen ottaa uusia työkaluja käyttöön niiden hallitsemiseksi.

Työlistojen täysipainoinen käyttäminen suuressa ja kompleksissakin organisaatiossa vaatii niin toimivaa organisaatiota, hyviä työtapoja kuin oikeita työkaluja.

Avainsanat: Ketterä ohjelmistokehitys, työlista, tuotesuunnittelu, tapaustutkimus

## Contents

# 1 Introduction

## 1.1 Background of the research

In software product development, strategic long-term plans set direction for entire organizations but are ultimately realized by developers performing modest tasks on a daily basis (Christensen and Raynor, 2003). Many software companies need to bring strategies into action in a turbulent environment where nothing is permanent; customers change their mind and behavior as soon as the products reach them; technologies evolve rapidly; competitors are able to shrink time-to-market. Flexibility is needed to cope with the changes but at the same time control is needed to persistently head towards the long-term goals. (Rautiainen and Lassenius, 2004)

Agile software development is a counter to traditional software development methods that are seen as heavyweight and bureaucratic – a barrier for flexibility. The specific agile methods like Scrum (Schwaber and Beedle, 2002) and Extreme Programming (Beck, 2000) have gained the attention of industry and are being widely adopted (Ambler, 2006; Ambler 2007). The sweet spot in which the agile methods are most likely to succeed is where a small, self-managing team of experienced developers with continuous support from business representatives can concentrate on a single project at a time (Cockburn, 2002).

Today methodologists and software companies are pushing agile methods beyond their natural comfort zone. Practices are modified to match the demands of large enterprises. Some report of initial success (Schwaber, 2004) while some are headed in full speed into the panic zone (Lowery and Evans, 2008). Scaled agility has also attained the curiosity of researchers but empirical evidence is scarce (Dybå and Dingsøyr, 2008; Dingsøyr et al., 2008).

Much of the flexibility of agile methods stems from the agile requirements engineering practices. Scrum, for instance, is a framework completely dedicated to rapidly fulfilling customers' requirements and constantly eliciting them. Periodic software deliveries and continuous feedback keeps software development projects on track. (Schwaber, 2009) To achieve the ambitious goals, agile methods use different practices for managing requirements than "traditional" requirements engineering. Backlogs, user stories and iterative requirements among other engineering practices have replaced up-front requirements phases and comprehensive system requirement specification documentation. (Paetsch et al., 2003; Cao, 2008)

Agile practices are being experimented and used in large-scale agile software development i.e. multiple teams are simultaneously working on the same product release. Some practices, however, do not scale natively (Leffingwell, 2007). Scrum presented the role of product owner as the single accountable person for the project success. The product owner is supposed to gather customer requirements or product managers' grand visions into a prioritized backlog; decompose them on an ongoing basis; and communicate and clarify them to development teams (Schwaber, 2007). In a large

company a product owner might be struck by a backlog that grows in a steady rate of a few hundred user stories per week. The ability to either respond to change or steer towards the long-term goals is long gone and lost in the depths of the backlogs.

Recently, some literature has emerged to address large-scale agile development. In large projects the numerous teams need more than one product owner. It is naïve to think that a person working in close collaboration with development teams could also "represent all stakeholders" in a large project. The responsibilities need to be divided and coordination mechanisms created. (Leffingwell, 2008; Larman and Vodde, 2007; Schwaber 2007; Rothman 2009)

Are the agile practices fit for large-scale projects? Are backlogs a suitable solution for delivering the long-term plans in bite-size chunks for the development teams now that the plans are made as a collaborative effort by many product owners? Methodologies and agile consultants spread the word of successful agile adaptations of large enterprises. However, descriptions of how product owners and development teams cope with the backlogs tend to be vague. The case descriptions reveal some of the difficulties and pain companies are having with their spreadsheet-based backlogs (Moore et al., 2007; Lyon and Evans, 2008).

Finding the right granularity of planning on different organizational levels and providing visibility into the development work for both managers and developers is a true challenge for companies. Given the central role backlogs and requirements in general have in agile methods surprisingly little is published about how the backlogs could and should be used. The two interesting questions are how backlogs could provide the trace from the long-term plans to daily development tasks and how they could provide visibility into the progress so that refining the plans can be based on best possible information available.

This research presents a case study that was conducted in a case company that is doing large-scale agile software development. Earlier experiences presented in literature are contrasted to the situation in the case company. We see how long-term plans are gradually detailed and end up to the development teams. The progress information flow that starts from the teams is also followed back to managers. Challenges did arise within the case company but a number of improvements were made.

## 1.2 The ATMAN research project

This thesis was conducted as a part of the ATMAN (Approach and tools support for software development portfolio management) research project. The project is carried out in Helsinki University of Technology's Software Business and Engineering Institute and is funded by Tekes (The Finnish technology fund) and the participating companies. The ATMAN project studies how software companies can better link business strategies and long-term product development planning with daily work. A specific goal is to produce a framework that describes the elements constituting to this linking.

## 1.3 Research problem and scope of the research

The research problem of this study is:

> How are long-term product and business plans communicated to software developers that work iteratively and how is their progress monitored?

Companies using agile methods provide an interesting setting for studying the research problem for two reasons. Firstly, the agile methods are being widely adopted and the interest is still growing in the industry. Secondly, literature on agile methods provides conflicting and only little concrete guidance.

The focus of the research is on backlogs and the relating practices and roles. Scrum introduced the concept of backlog and is the most widely adopted agile method that defines practices for planning and progress monitoring.

Agile methods are tailored for small individual teams of experienced developers. But agile methods are trialed outside of their natural comfort zone by large companies. The research problem is most evident in large software product companies. But empirical research of large-scale agile development is scarce. Also studies of backlogs are almost non-existent.

The research problem is studied only in the scope of agile software development and moreover in the case of large-scale agile development i.e. organizations where multiple teams are simultaneously working on the same software product.

## 1.4 Research questions

The research problem is divided into five detailed research questions (RQ).

**RQ1. What are the suggested practices for using backlogs to communicate long-term plans to developers in agile software development?**

Literature addressing Scrum and large-scale agile development is studied to find out the suggested practices for communicating long-term plans to developers. Also the findings of existing research are studied. Identification of good practices helps to reflect the situation of the case company with the current knowledge.

**RQ2. What are the suggested practices for using backlogs to communicate progress information to managers in agile software development?**

In order to make good decisions the managers need information how the previous plans have been realized. To answer the second research question, literature addressing agile software development is studied to identify recommended practices for progress monitoring.

**RQ3. How is a product's goal and progress information communicated in the case company?**

**RQ4. What are the challenges the case company is facing in communicating the product's goal and progress information?**

The third and fourth research question is answered by conducting a case study. By describing the state of the practice this research contributes to the emerging body of knowledge of agile in large enterprise context.

**RQ5. What could be improved to overcome the identified challenges?**

The case company's improvement efforts are observed and reported to gain understanding of what can be done to overcome any possible communication challenges that are identified after answering research questions RQ3 and RQ4.

## 1.5 Research methods

Research questions 1 and 2 are answered by conducting a literature review. Constructing a framework that describes how backlogs provide a link between long-term plans and daily development tasks combines the results into a single view. The framework can be used as a guide for empirical inquiry and also for analyzing the resulting data.

Because the purpose of research question 3 is to describe a contemporary phenomenon within a real-life situation a case study is a preferred research strategy (Yin, 1994). The case company was identified among the ATMAN partner companies and a company that suited the purposes for this study was selected (Patton, 2002).

Besides describing the situation in the case company (RQ3) the aim of the study was also to identify possible challenges the case company is facing (RQ4) and describe any improvement efforts made to overcome them (RQ5).

The research approach in this study can be described as participatory action research, which is a kind of case study. In action research the case company and the researcher collaborate in solving a problem, and the researcher simultaneously makes observations in order to gather research data. The initial step in action research is to select the problem to be studied and solved. Next, different types of actions to solve the problem are considered. One of them is selected and its effects are studied. And finally, general findings are identified of the selected action and its effects. (Susman and Evered, 1987).

## 1.6 The case company and empirical data

The case company is a large Finnish software company. The company has several hundred employees. The case study was conducted at a research and development (R&D) organization that develops the software and has a few hundred people of which about half are software engineers. The company operates in global consumer and enterprise markets. It offers products and services in more than 20 language versions. The offerings are based on common components and platforms. Typically the R&D has more than 20 concurrent projects. Development teams are distributed on several sites that are located in a number of different countries. The case study was conducted entirely at the case company's headquarters in Finland.

Figure 1 presents a timeline of relevant events from the case study's viewpoint. In 2006 the R&D started a transformation to agile methods. The decision was made by management and was motivated by the need to hit market windows more reliably, get customer feedback more often and early and improve the capability of reacting to feedback. The transition was planned to be done in a fortnight during which all development work was ceased and the time was used for an intensive training period.

In February 2008 the research cooperation between the case company and ATMAN research project started with a current state analysis, which revealed challenges in the product owner role and in applying some agile practices. The results inspired further research. In May 2008 the case company representatives and ATMAN researchers agreed on conducting a study on backlog practices and relating roles and responsibilities and the results would be reported in this thesis.

From August 2008 to December 2008 research data was collected in seven interviews; in 23 observation sessions where the researcher was present; and in six observation sessions that were conducted by other researchers, which I listened from a recording afterwards. All sessions were digitally recorded and a diary was kept. Written feedback was provided to the participants after each observation session.

Within the case company purposeful sampling (Patton, 2002) was used to select the teams and managers for observations and interviews. The selected four teams and their managers were working on the most complex product family. It was assumed that they would need the most coordination and the requirements for long-term plan communication and progress monitoring would be the most complex ones. These teams are named as *the core teams.* In order to gain a broader and more representative view of the case company two more teams with distinctive qualities were selected: *the near-shore team* and *the startup team.* Team compositions and their characteristics are later discussed in detail.

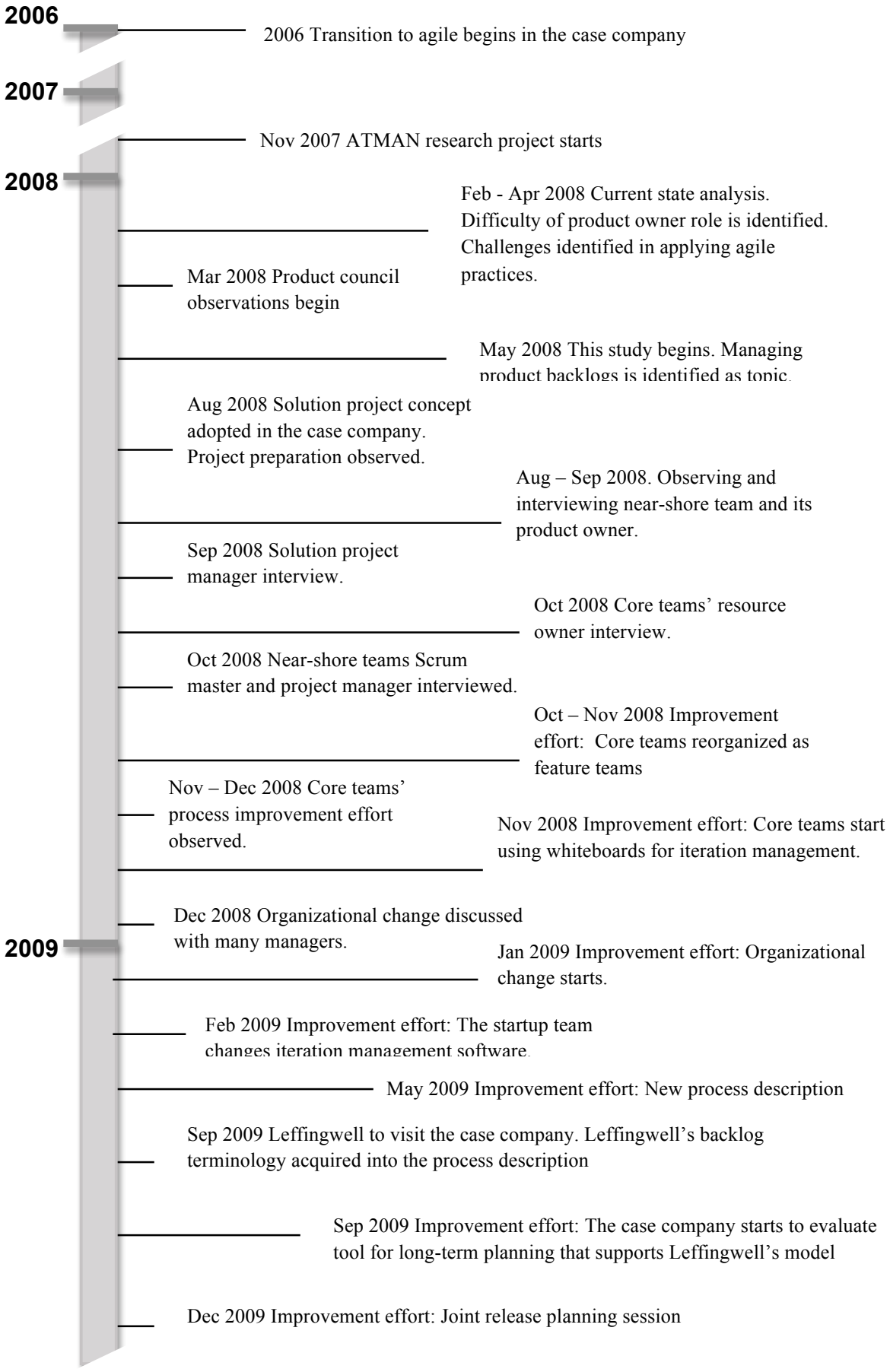


**Figure 1 Timeline of the research at the case company**

## 1.7 Structure of the thesis

This thesis consists of five chapters: introduction, literature review, case study results, discussion and conclusions.

2. Literature review presents the existing body of knowledge. The focus is on finding the suggested agile backlog-related practices for planning and monitoring progress. Two topics outside the mainstream of agile literature are the long-term planning aspects and use of agile methods in large enterprises. Finally, a framework is constructed that unifies the literature review results.

3. Case study results chapter presents the findings of the case study. The constructed framework is used for presenting the case study results.

4. Case study and literature review results are discussed. Threats to results' validity are also discussed.

5. In the final chapter conclusions are drawn from the results. Also further research is suggested.

# 2 Literature review

In this section the results of the literature study on using backlogs in agile software development for linking long-term plans with daily work are presented. The goal of the literature study is to answer research questions RQ1 and RQ2 that are presented in section 1.2. In order to compare literature review results against each other and the case results, a cohesive view of backlog management practices is constructed.

## 2.1 Agile software development

The term agile software development was introduced in a manifesto that claims that there is a better way of developing software (than the prevailing one). The signers of the manifesto demand a shift of focus from excessive planning, comprehensive documentation and command-and-control attitude to adaptation, the software itself and collaboration between developers, managers and customers (Fowler and Highsmith, 2001).

Larman and Vodde (2009) point out that the benefit of agile software development is not about delivering software faster, with higher productivity or with better quality but the competitive edge is gained by adaptability i.e. by being able to change and react faster than your competitors.

Agile software development is loosely defined and no single definition exists (Abrahamsson et al., 2002). The manifesto and the twelve principles published as a part of it provide some guideline for defining what is agile software development. Three of the principles are presented here because they have impact on how planning and progress monitoring should happen in agile software development. The agile manifesto and complete list of the principles is presented in Appendix A.

> *Business people and developers must work together daily throughout the project.*
>
> *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
>
> *Working software is the primary measure of progress.*

The principles of the manifesto are quite detailed but as a whole they do not provide an extensive guideline on how to do software development. For instance, it can be argued that not all teams can communicate face-to-face all the time and there might be situations where working software is not the best measure of progress. The manifesto sets some boundaries, but the implication of the manifesto might not be evident after reading it through. Cockburn (2002) explicates the impact of being agile by proposing an ideal context – sweet spot – where agile software development enhances the possibility of project success. He proposes to move a project as close as possible to the following conditions in order to be agile:

- *Two to eight people in one room*

- *Onsite usage experts*

- *One-month increments*

- *Fully automated regression tests*

- *Experienced developers*

In conclusion, agile software development probably suits best those who fit in the sweet spot and on the other hand it is a set of values and principles that support fostering change, being adaptable.

### 2.1.1 Agile methods

Agile methods are software development methods that share the agile values. First descriptions of those methods are clearly targeted for the Cockburn's sweet spot. The most well-known and widely adopted agile methods are Scrum and Extreme programming while many others exist. Agile methods have gained lot of industrial interest and are being widely adopted (Ambler, 2006).

Research of agile methods has focused mostly on Extreme programming and somewhat on Scrum. But it has been stated that research is lagging and empirical evidence is scarce on this subject. Agile methods are also been critiqued for the lack of explicit definition of underlying assumptions, which makes it difficult to evaluate their suitability in a particular situation (Turk 2005). While empirical evidence and rationale for selecting a suitable method are missing or scarce, decisions seem to be based on faith. (Dingsøyr et al., 2008)

The agile methods cover different phases of software development to a varying degree (Abrahamsson et al, 2002). Thus, it is no wonder that complementing agile methods are combined to meet industrial needs. Combining project management practices of Scrum and software development practices of Extreme programming have been reported in numerous cases (Sutherland et al., 2009).

As there is no clear description of methods' suitability and the state of practice is to combine practices of different methods, the term agile method has become to mean any set of practices that comply with the manifesto. In a synthesis on agile methods Abrahamsson et al. (2002) conclude that an agile software development method is:

- *incremental (small software releases, with rapid cycles)*

- *cooperative (customer and developers working constantly together with close communication)*

- *straightforward (the method itself is easy to learn and to modify, well documented), and*

- *adaptive (able to make last moment changes)*

### 2.1.2 Requirements in agile software development

The goal of requirements engineering is to ensure that the resulting software meets customer needs. Requirement engineering deals with understanding customer needs and managing requirements during the development so that the needs are eventually fulfilled (Pressman, 2004). The sequential model for software development suggests that requirements are defined and documented in the beginning of a project (Royce, 1970). In his paper Royce describes such linear approach only to be the simplest description and not the actual recommendation (Larman and Basili, 2003). But it has been claimed that the model has been applied in such oversimplified manner in the industry (Leffingwell, 2007).

Leffingwell (2007) argues that doing requirements engineering as an isolated phase in the beginning of a project and managing the changes with some change management practices has major shortcomings. Firstly, he points out that such approach assumes that there exists a reasonably well-defined set of requirements that can be defined in the beginning of a project. He claims such assumption to be false because the intangible nature of software cannot be captured in documents that would be understood in the same way by the customer and the developers; user needs are ultimately revealed only after the user has tried out the software; and the delivery of software changes user behavior and results in new requirements. The other false assumption, he points out, is that change would be small and manageable. He argues that contextual factors like user needs, technologies, markets and competition keeps changing during the project. The conflict between the requirements defined in the beginning and the requirements existing in the end of the project is revealed only in a late phase, making the situation even worse.

In agile methods software is delivered incrementally, early and often to maximize customer feedback. In order to get the most relevant feedback the customer needs are prioritized promptly, and the most valuable features are delivered first. The delivery and the feedback enhance both customer's and developers' understanding of the requirements. Instead of an upfront requirements elicitation phase, the requirements are gathered and analyzed in small portions.

In the beginning of a project there might be only a vague vision of the desired direction and it becomes clarified as the most important features are developed. In agile methods development is time-boxed, which means the schedule is fixed. If something changes it is the scope i.e. the requirements. They are constantly changing and therefore need to be easy to work with. Two widely used practices that address the need for lightweight and just-in-time elicitation are user stories and backlogs.

User stories were presented in Extreme programming (Beck, 2000) and popularized by (Cohn, 2004) to provide a lightweight and just-in-time practice for managing requirements. A user story is a free-form expression of software functionality as customer or user of the system views it. A suggested way to write user stories is to use a format like: "As a <user type>, I want <description of the desired functionality> so that <business value>". The user stories are compact but not comprehensive. They are not intended to provide the full specification or capture all possible aspects of a requirement.

Instead, they are reminders of requirements that need to be further elaborated and discussed or even documented. (Cohn, 2008).

The concept of backlogs for requirements management was introduced in Scrum (Schwaber and Beedle, 2002). Backlog is a prioritized list of backlog items that might be presented in the form of user stories. The items represent work to be done that provides value to the customer. The less important items are allowed to be vague and large in terms of needed development effort. The top priority items need to be elaborated in more detail and split so that they fit in a time-box and can be more accurately estimated.

Cohn (2008) discusses splitting user stories in detail. His advice is to split stories according to the boundaries of the data in that story; operations that are performed in that story; by crosscutting concerns; by functional and non-functional aspects; if a story contains distinguishable priorities; and finally not to split a user story into development tasks but into end-to-end capabilities.

### 2.1.3 Scrum

Scrum is an agile method for project management. It was invented and used for the first time in 1993 and published in 1995 (Schwaber, 1995). According to a recent description Scrum is an iterative and incremental process framework that addresses the complexity of software development projects by empirical process control. (Schwaber, 2009).

With empirical process control Schwaber means a set of practices, rules and responsibilities that promote transparency, inspection and adaptation. Complexity is addressed by inspecting aspects of the process so that deviations from the desired outcome are detected. For this transparency – visibility to the process – is needed. And finally, adaptation is used to change the process to reach the desired outcome. (Schwaber, 2007).

The rules, practices and roles that are supposed to promote empirical process control are defined in detail but do not address all software engineering practices but only those of project management. Scrum includes project management, release planning, iteration planning and daily practices but excludes for instance product management, roadmapping and strategic management.

Scrum introduces the concepts of backlogs and burndown graphs that provide visibility. Items in backlogs are created and updated in order to define scope for releases and iterations. The items contain estimates that form the basis for progress monitoring. In addition to backlogs and burndown graphs visibility is enhanced by frequent demonstrations of the product and close collaboration of team members.

**Roles**

Scrum defines three roles: the product owner, the scrum master and the team. The product owner is responsible for project outcome and represents the interest of all stakeholders that relate to the project. He is responsible for achieving funding and for this needs to define a release plan, initial requirement, and return on investment objectives.

After the project has started the product owner ensures that the most valuable functionality is built first. Despite the plethora of responsibilities the product owner is a single person and not a group of people.

Scrum does not provide specific guidelines for product owner how to handle all responsibilities except for how to ensure that functionality is built in order of the greatest value. For this, the product owner uses a product backlog, which is prioritized list of everything that might be needed in the product.

Scrum master is a process coach who helps the team to adopt Scrum. Scrum master might or might not be part of the team but Scrum master should never act as product owner. Scrum master also coaches and helps management select the product owners.

The team itself is self-organizing and responsible for managing its work. The suggestion is that a team would be a cross-functional feature team i.e. the team delivers features that the customer demands and has all the relevant skills to do this by themselves. There is no hand-off to other teams of partial solutions. Scrum master does not manage project and product owner only commands the order in which work is carried out. Managing the project and implementing the items in the product backlog is up to the team. They are also required to do this iteratively and incrementally.

The optimal size for a team is five to nine people. The team must be capable of delivering the features or other things in the product backlog from start to finish. It is also explicitly prohibited to have sub-team specializing on a certain domain like testing.

**Practices**

When project starts it should have a vision defined by the product owner. In release planning the vision is turned into a plan. This plan includes possibly dates and budget but the most important outcome is the product backlog. The product backlog is re-planned, estimated and prioritized throughout the project. The planning is done just enough to get iteration planning started.

Scrum's iterations are called sprints. They are one-month long or shorter time-boxes that produce an increment to the product under development. A sprint starts with a sprint planning meeting where the team and the product owner decide what is done. The product owner uses the product backlog and describes and explains the most important work that needs to be done. The team estimates how much work it can complete from the product backlog. The team plans how to accomplish the selected work. The output of planning is a task list, which is called sprint backlog. It can and should be refined during the sprint as more tasks are figured out or old ones become obsolete.

During the sprint it is prohibited to interrupt the team e.g. by demanding new features. The team manages the sprint and to keep everybody up to date it has a short daily status meeting. If it becomes obvious that planned work cannot be done changes must be negotiated with the product owner. At the end of the sprint the team demonstrates the produced increment and finally holds a retrospective meeting to improve its work in coming sprints.

**Artifacts**

Product backlog presents all work planned for a product in priority order. It is a constantly updated list, where the top priority items are described in more detail than the less important. As things get done the new top priority items are planned in more detail. This is product owner's responsibility but it is recommended to do this in cooperation with the team.

The items in the product backlog include but are not restricted to features, functions and defect reports. The use of user stories or use cases is encouraged in the product backlog. These artifacts entail also the non-functional requirements of a feature. The product backlog items should have also an estimate.

A release burndown graph depicts the progress of the release. The estimated effort left in the product backlog items that are planned for the release are drawn at least at the end of each sprint. This graph gives rapid visual feedback how much product backlog has grown and how much effort is left. The graph also shows team's velocity i.e. how much estimated effort a team can carry out per sprint. Keeping the release burndown graph up to date is product owner's responsibility.

Sprint backlog is team's tool for managing a sprint. The sprint backlog contains tasks that are needed to accomplish the selected product backlog items. Tasks are constantly updated so that the sprint backlog provides constant visibility to the work the team is doing. The tasks have status to indicate what has been started and what has been done. Task have also estimates which provide the data needed for sprint burndown. It is a similar graph to the release burndown only that it is updated more often, at least daily.

## 2.2 Cycles of Control: A temporal-pacing framework

Cycles of Control (CoC) is a framework that helps a software development organization to combine software engineering practices into a functioning whole by establishing a rhythm. The framework is based on the concept of time pacing; goals are allocated into a fixed-length time periods, cycles. Each cycle starts and ends with a control point, where decisions are made. Adaptability is achieved by reacting to changes at regular time intervals, while persistence is guaranteed during the cycle because changes are made only at the control points. (Rautiainen et al., 2003)

A key in creating an efficient rhythm is to have proper length cycle for goals. Long-term decisions have no effect if they are constantly changed, while more specific, short-term goals cannot be accurately defined for months ahead. Thus, the framework defines four different length cycles. The longer cycles include many shorter ones and set direction and constraint for them. In Figure 2 the cycles are drawn to a timeline.

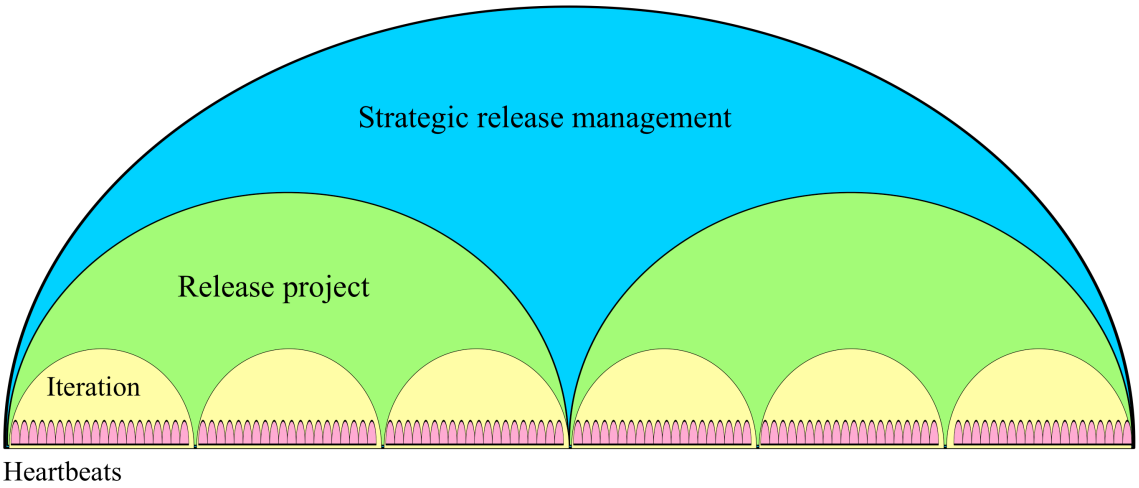


**Figure 2 Cycles of Control on a timeline**

The longest-spanning cycle is strategic release management, where long-term plans are made and monitored. Its recommended length in turbulent environments is 6-12 months. In this level the direction of product development is set. The starting control point is the planning of future releases i.e. roadmapping. The resulting document, roadmap, defines 2-3 next releases and their main goals at a coarse level. In the closing control point the outcomes of releases are inspected and the roadmaps are adjusted if needed. It is suggested that plans should be updated also after every iteration, or at least it should be considered.

The next cycle is release project that aims to produce a working version of the product to customers. The cycle length is 3-4 months and it starts with release planning and ends with the decision whether release is successful and can be launched to the target market. The release's goals are defined in the release planning, which is guided by long-term plans in the roadmap.

The iteration cycle aims to produce an increment to a release i.e. a set of features that are build on top of the previous increment. An iteration should last no more than one month. The cycle starts with iteration planning that is guided by the release goals. The ending control point is a demonstration the capabilities of the produced increment.

Heartbeat is the lowest level and it is where actual development work is controlled. Its length varies from a day to a week. The progress of iteration goals is monitored and managed in this cycle. The starting and ending control point can be a meeting where the achievements of the previous and plans for the next heartbeat is reported. The heartbeat cycle provides a steady status information flow on how development work is progressing.

By defining length for the cycles on all levels an organization would create a rhythm for its development that provides both flexibility and control. The rhythm is a backbone that is used when practices are designed. For instance quality assurance or backlog management practices and their interplay would be considered on all four levels. This way the framework can help in analyzing the appropriateness of the current practices or it might reveal missing links.

### 2.2.1 Backlogs in the time horizons

Mapping the backlog practices suggested in Scrum to the CoC framework is a straightforward task, because CoC is based on Scrum. Figure 3 shows an overview how backlogs fit on the levels of CoC framework.

The product backlog contains all planned work for a product. The short-term goals should be detailed and on top of the backlog as their priority is highest and the long-term goals are located at the bottom and they are coarse grained. The bottom part of product backlog and its management fit in the CoC frameworks strategic release management level. In Scrum there is no further guidance how to manage the long-term plans in the product backlog; its mentioned to be done by the product owner.

As the top priority items get finished the long-term goals move closer to the top and need to be split into smaller and more detailed items that can be allocated into releases. Such release goals form a subset of the product backlog called release backlog and it maps directly to the release project level. Scrum suggests release burndown graphs for monitoring releases.

Iteration backlog (Sprint backlog in Scrum terminology) contains a subset of the release backlog and they fit into the iteration level. Items in the release backlog might need splitting prior to allocating them into iterations. Once items are allocated into an iteration they become goals for that iteration. The development team splits them further into tasks, which are also kept in the iteration backlog. The tasks should be small enough to be completed during a heartbeat. For progress monitoring on heartbeat and iteration level Scrum suggest using the sprint burndown graphs.

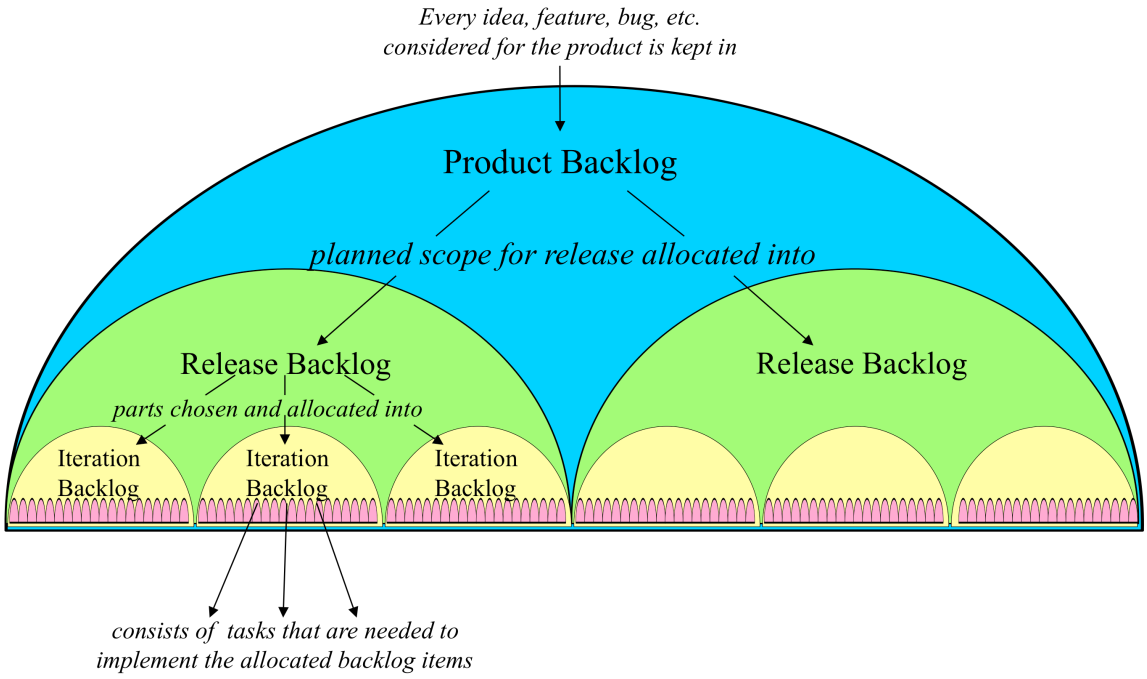


**Figure 3 Backlogs in Cycles of Control framework**

## 2.3 Agile in large enterprise context

The sweet spot of agile (Cockburn, 2002) is far from reality for many companies. Software projects are being developed – instead of 2-8 people working in the same room with constant access to a customer – by hundreds of people distributed on different geographical locations and time zones and no single customer exist. But the success of agile projects (Sutherland et al., 2009) has raised a question can the agile development be scaled to match the large and complicated situations. Some literature has emerged in recent years to address agile in the large enterprise context, e.g. (Leffingwell, 2007; Larman and Vodde, 2009; Schwaber, 2007) but empirical research is still scarce on this subject (Dingsøyr et al., 2008). When whole enterprise adopts agile methods it needs to consider how large projects with multiple teams are coordinated and also the long-term planning practices should work together with the agile practices.

### 2.3.1 Schwaber model

Use of Scrum in a large enterprise was reported by (Schwaber, 2004). The project employed multiple collocated teams. This lead to a need for synchronizing their work and creating an architecture that allowed the work to be divided among the teams. The architecture needs were addressed so that one team worked for four sprints to produce an infrastructure for other teams. For coordination a practice called daily Scrum of Scrums was invented. After the teams had had their daily meeting each team would send a team member to participate in the Scrum of Scrums meeting, where each team representative reported his team's status.

For backlog management Schwaber (2004) reports that spreadsheets were functioning well enough even for large companies, though he notes it required quite a lot of cooperation and communication. His approach was to form a hierarchy of product backlogs. Customer product backlogs were composed into district product backlogs, which were composed into regional product backlog, which were finally composed into an overall product backlog. This helped prioritizing customer needs in the right context and combining all items into a single backlog where they were allocated for teams' sprint backlogs.

In a more recent book Schwaber (2007) suggest the same approach of combining product backlogs to be done for all product backlogs of an enterprise. The enterprise product backlog is used for prioritizing and managing all development work of an enterprise. The enterprise product backlog is described by few examples where the backlogs are presented as tables. But Schwaber does not provide concrete advice how to manage them and with which tools.

In a recent report of using Scrum in a large scale project the Scrum of Scrums and combined product backlog were tried out (Lyon and Evans, 2008). Challenges with product backlog is described as follows:

> *One of the key problems was the Product Backlog. It was the right decision to have a single product backlog but our problem was the level of detail in the backlog. By combining the individual teams' backlogs we had created an*

*unwieldy monster. It was impossible for a single Product Owner to come to grips with or to use it to communicate to anyone the scope and priorities of our work. It was also rapidly ceasing to be of any use to the teams both in terms of a tool to direct their work and sprint planning and in the sheer mechanics of trying to share a single Product Backlog between nine teams.*

Having different granularities between different types of backlogs solved the issues. They report that the teams grouped items into larger themes and also allocated items into releases. The teams' product backlog items had to be linked to the combined product backlog. Prioritization of the combined product backlog became possible and progress could be monitored because of the links.

Schwaber's suggestion to use an enterprise backlog follows directly from the way he suggest organizing people into Scrum teams and organizing the teams on different integration levels. An integration team coordinates the development teams' work. The integration team has developers, a Scrum master and a product owner that uses the Scrum practices to develop infrastructure for integrating, building and testing the development team's work. If multiple integration teams are needed to coordinate the development teams, an additional integration level is added that is managed by a similar integration team. The hierarchy is as deep as needed. The integration team's practices are not described, only the structure and their intent. The actual practices are formed as the teams self-organize. (Schwaber, 2007)

### 2.3.2 Leffingwell model

In his book Scaling software agility and whitepapers Leffingwell (2007, 2009a, 2009b) describes what a large organization might look like if it would make an agile transition and he calls it a "Big Picture". He gives a concrete suggestion how an enterprise should be organized; what the development process is like and what are the necessary roles. He also stresses the importance of requirements practices: "- - Big Picture should highlight the requirements practices of the enterprise agile model, because those artifacts uniquely carry the value stream to the customer" (Leffingwell, 2009a).

At the heartbeat and iteration level Leffingwell's model does not differ from Scrum, only that iteration backlog items are called stories. Stories are implemented by completing tasks. Leffingwell suggests using a backlog item hierarchy with four levels: epic, feature, story and task. Epics and features are similar to stories only that features are bigger than stories and epics even bigger. An example of Leffingwell's hierarchy and its relation to different length time periods – program, release and iteration – is depicted in Figure 4.

Leffingwell claims that the teams working on a large project or program tend to cause "pods" of developers that are organized around different implementation domains. Pods are typically about of size 3-10 teams or 50-100 people. For coordinating teams within a pod he suggest the Scrum of Scrums meeting and keeping each team's iterations synchronized with the others.

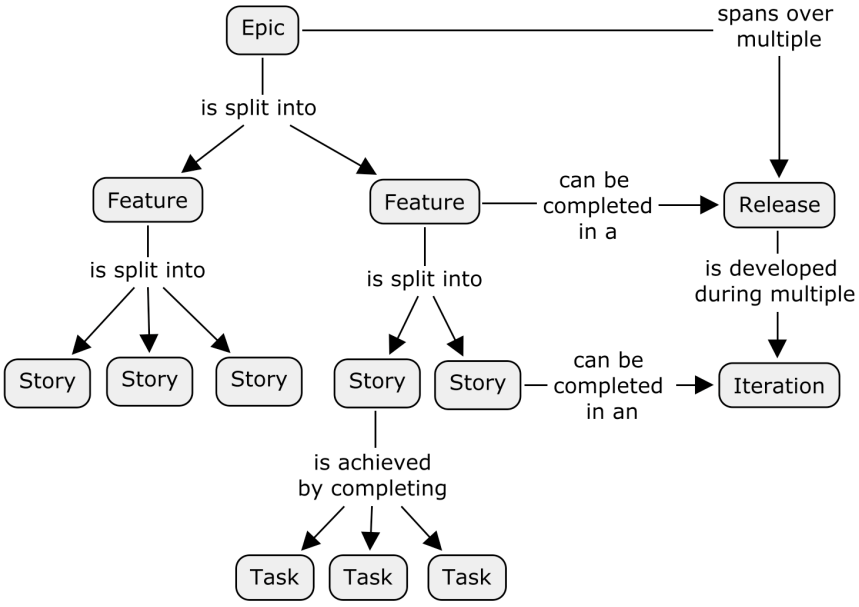


**Figure 4 Leffingwell's model of backlog item hierarchy and its relation to programs, releases and iterations**

A new role is presented in the release level. Product owners are part of the development teams and he owns the iteration backlogs. A product manager owns the release backlog. Dividing responsibilities between product owners and product managers is reasoned by the fact that senior managers with understanding of the markets are too few to be involved with all the teams and yet the teams need a single person to steer in the right direction by prioritizing the iteration backlog.

Product manager is solely responsible for release success but is aided by a release management team. It includes a covering mix of representatives from the enterprise such as product manager, senior manager, system level quality assurance expert, system architecture and even agile coaches. This team has the understanding and capability how and when to deliver a solution to the end users. The team has a weekly meeting where release progress is monitored and corrective measures are taken without jeopardizing the release's overall goal.

A suggested practice is also to have another special team – system team is essentially the same thing as Schwaber's integration teams. They provide infrastructure, manage integration and do system level quality assurance.

Stories are too detailed and numerous to be useful for release management in large projects. A higher level of abstraction is needed. Release backlog contains features that describe a service provided by the system that fulfill a user need. Basically features are user stories with higher level of abstraction, though any format other than user story can be used. For describing a system of arbitrary complexity 25-50 features are supposed to be enough so that releases can be planned and managed.

Features in release backlogs are supposedly too large to fit in an iteration backlog. In release planning the features in the release backlog are split into stories that are allocated into iterations. Features can be estimated roughly but the details for well-grounded

estimates are gained only when story splitting is done. Suggested practice for release planning is to gather the whole pod to do it.

Besides the release backlog a product manager owns also a vision and a roadmap. The vision gives an overall goal for the release and it guides release planning. The roadmap has two parts. The first one contains the current release and its features. The latter part describes few upcoming releases and their goals, which probably are still vague and should not be considered as commitments. The roadmap's latter part fits on the strategic release management level in the CoC framework as it has a long-term view on development work.

On top of the program level is a portfolio level. On that level portfolio managers have epics in a portfolio backlog. Epics are the highest-level expression of a customer need. They might not fit in a single release and need to be split into features before release planning. The epics describe a strategic intention and even their mutual priorities might not be clear. The portfolio managers can set a relative investment level for epic to communicate how effort should be divided among the epics.

### 2.3.3 Larman model

Larman and Vodde (2009) discusses the ill effects of component teams, which refers to an organizational structure where a product development is divided into functional aspects like graphical user interface or databases and a team is assigned to handle such aspect. Some of the issues, raised because of the component team structure, are complicated planning; tendency to work according to team preferences rather than customer's priorities; work slowing down because of hand-offs between teams. The presented solution is to transform the teams from component teams into feature teams that are capable of producing end-to-end features for customers.

In large-scale product development the feature teams are assigned into requirement areas. Each requirement area deals with a set of end-to-end features that provide value to a customer and together the areas cover all product requirements. A product owner is responsible for the product and area product owners are responsible for the requirement areas.

Product backlog describes the requirement areas that are split into product backlog items. A product owner owns the product backlog. An area backlog is a subset of the product backlog and it also contains area backlog items that are split from the product backlog items. Area product owners own the area backlogs. The product backlog and area backlogs are prioritized independently. Because of the hierarchy of backlog items the priorities might be different. Larman realizes the conflict and gives an advice: "When this difference [of priorities] is big, then it must be reflected in the product backlog so that it is visible to the product owner". But no practical guidance is given how to actually do this.

Larman does not discuss how release planning or long-term planning is affected by the requirements area approach. In the iteration and heartbeat level, teams can work

according to the basic Scrum practices. In sprint planning, area backlog items are split into sprint backlog items, which are further split into tasks.

Scrum of Scrums practice is not recommended without considerations. The feature teams are supposed to be self-organizing and thus need to decide by themselves whether they need such coordination practice or not. And if the Scrum of Scrums is organized it certainly should not be a meeting of Scrum masters or a relabeled meeting that replaces some existing managerial meeting. The recommendation is to use rotating representatives from teams to attend the meeting.

For managing large backlogs the use of spreadsheets is encouraged, not because they are the best choice but because the alternatives are probably worse. The problem with traditional requirements management tools is that they might need too much effort to support the prioritization needed by agile methods. A problem with even tools aimed for agile methods is that many of them are optimized for reporting and not for planning, which slows teams' and product owner's work.

## 2.4 Framework for agile product management

A framework that presents the artifacts and concepts for planning and monitoring agile product development was created from the results of the literature review. The framework is presented in Figure 5. It was created to analyze how the different artifacts might link long-term plans with development tasks. The framework "normalizes" vocabulary so that it is easier to compare literature review and case study results. It can also be used to reveal possible missing links when the flow of requirements and progress information are followed in the case study.

In agile software development initial plans are coarse grained and ruthlessly prioritized according to perceived customer value. Only the few most important goals are taken into consideration at a time. The goals are split into sub-goals that are small enough to be completed in a time-box. The prioritization and splitting occurs over and over again until the shortest time-box and granularity of actual development task is achieved.

A decision on when to split the next most important goal is based on the progress of lower levels. The velocity, i.e. the amount of work a team can complete, of previous time-boxes tells whether more planning and splitting is needed or are the current plans enough to accommodate the next time-box.

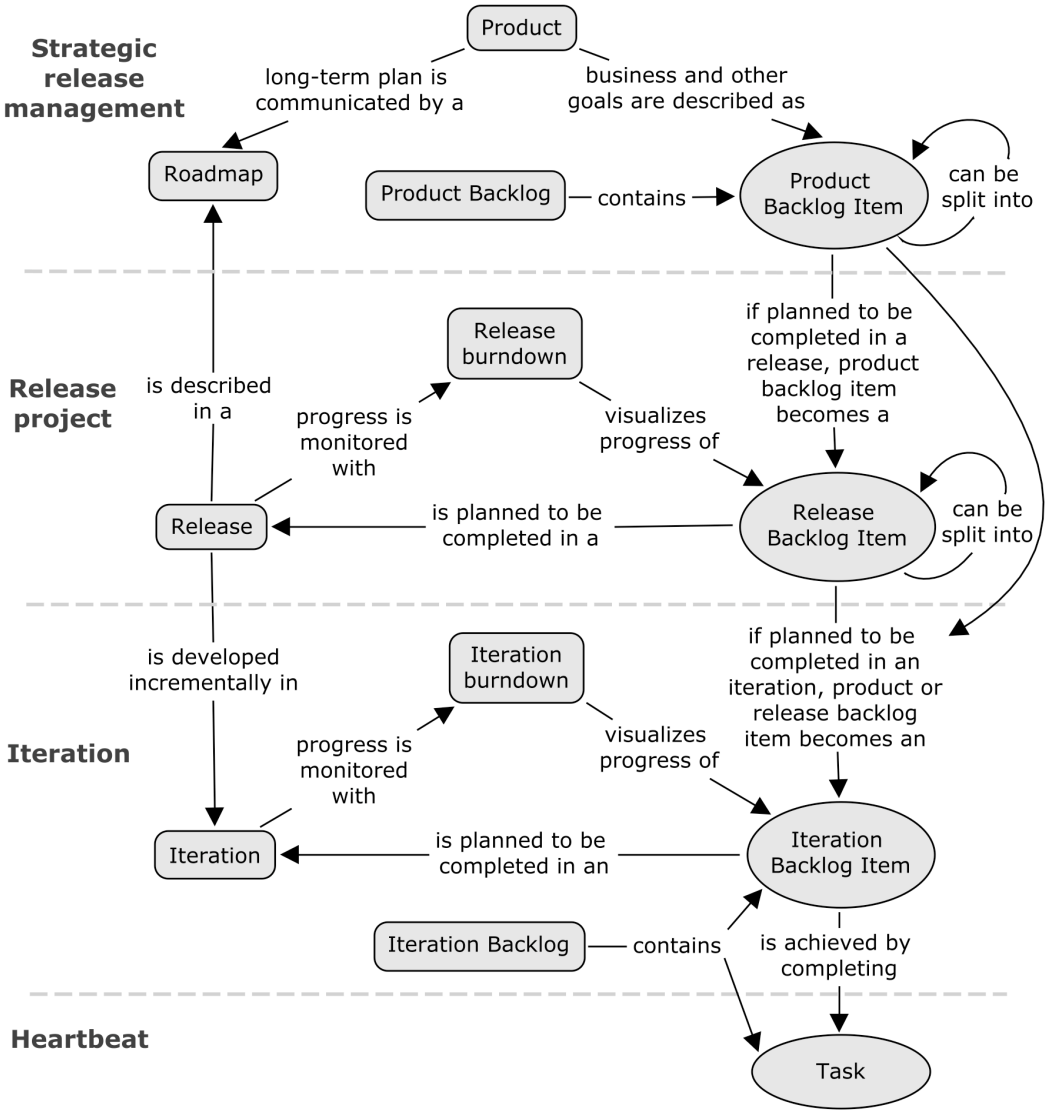


**Figure 5. Framework for agile product management**

Splitting and planning is only done barely enough to keep the time-boxes going, no more. It is probable that changing priorities and increased knowledge makes the previous plans obsolete. Keeping the plans as light as possible and making decisions just in time provides flexibility while maintaining control of the development work.

In the framework the time-boxes, or levels for planning and monitoring, are adopted from the CoC framework. In Figure 5, titles of the four levels are presented on the left hand side and a dashed line separates the time-box levels. The rounded rectangular shapes present the relevant concepts and an oval shape distinguishes items that are kept in a backlog. The lines and the associated text describe the relations between the concepts.

Long-term planning of a software product happens at the highest level, strategic release management. Product planning starts with high-level goals, such as business or technology goals, which are recorded as highest-level goals into a product backlog. A roadmap is used for communicating product's long-term plans. It describes the next few releases and their main goals, or high-level backlog items.

The backlog items are split into smaller backlog items, which can then be further split as many times as necessary to provide goals for a release. Such backlog items are now called release backlog items. Estimates of remaining effort of release goals provide the data needed to visualize release's progress in a release burndown graph.

Teams develop a release incrementally during iterations. Iteration goals are split from the release backlog items and maintained in a team's iteration backlog as iteration backog items. The team plans tasks for achieving the iteration goals. The tasks are also put in the iteration backlog. Iteration progress is visualized in an iteration burndown graph that is drawn based on estimates of remaining effort of the tasks.

The hierarchy of backlog items also provides a "big picture" from strategy to action, and feedback on progress from action back to strategy based on the completion of tasks and backlog items. Unfortunately hierarchies are detrimental is in presenting priorities, which is a major concern in agile software development. Therefore, two different views are probably needed: one for the hierarchy and one for presenting the lowest level of the hierarchy in a rank ordered list. This of course stresses the need for a tool that is capable of presenting both of the views (Szalvay, 2006).

# 3 Case study results

In this section the case study results are presented. The case company's research and development organization is described in chapter 3.1 in order to provide an overview of situation in which the case study was conducted. The observed teams and their current work were studied by interviews and observations from August to December 2008 as described earlier in Figure 1.

Chapter 3.2 describes the planning practices and related roles in the case company. The roadmaps and backlogs are followed from the strategic release management level all the way to the development teams' daily tasks. Chapter 3.3 describes the flow of progress information from the daily work tasks up to the level of roadmaps. In chapter 3.4 the big picture of the observed teams are drawn and contrasted to the framework for agile product management, which was described in chapter 2.4. Finally, the case company's improvement efforts are presented in chapter 3.5.

## 3.1 Research and development organization

This study took place in the case company's research and development organization (R&D) that develops 34 different software components and platforms. The case company has seven distinct business areas that produce solutions for customers. The solutions are compositions of services and R&D's software components and platforms.

Figure 6 depicts the case company's business areas and their relation to the R&D's components and their relation to development resources as they were in April 2008. In the figure, business areas are in the left hand side; components in the middle column; and development resources in the right hand side.

Each business area is lead by a solution manager (denoted with a capital letter in Figure 6). The components are grouped according to their responsible manager, product owner. The same product owner is responsible for the first and last component, so there is a total of nine product owners.

Figure 6 also shows six developer "pools" and the relations how they are used to resource the development of the components. Each pool is managed by a resource owner (I to V) and consist multiple development teams. A resource owner is responsible for resource management and coordinating the teams in his pool. There are four components that have no relation to any developer pool. Their development is resourced with external developers that are purchased on-demand basis.

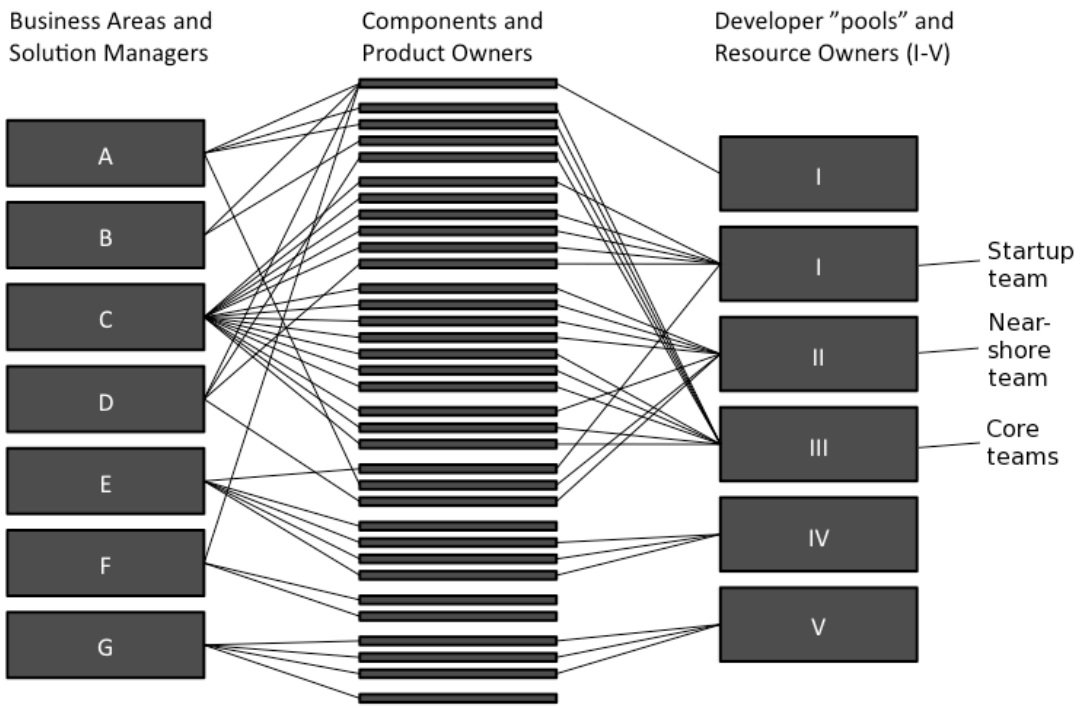


**Figure 6. Relations between business areas, components and development resources**

In Figure 6 the fourth developer pool (resource owner III) is the largest one in the company. The pool consists of five teams – *core teams* – each of which have about nine members on average. They develop so-called core components for all the major solutions. The core components share roughly 90 percent of their binary files. Thus, features developed for one core component are available for all the other core components. Because the components share a majority of their binary files they are not as independent as other components are.

Two other teams are also discussed in this study. They both have about eight members. They are both developing components that have independent binary files i.e. no other team touches their source code. Thus the teams are independent of other teams in their developer pool. The team in the second pool – *startup team* – is developing components for emerging markets. Their situation is special because no one else uses their components at the moment i.e. they are actually developing two complete solutions on their own.

The team in the third pool – *near-shore team* – is developing a component that is used in many solutions. The team is located abroad, while their product owner works in the case company's headquarters in Finland. Their distance – measured in travel time – is few hours. Thus, the team can fairly easily travel to the headquarters.

## 3.2 Practices and roles for planning in different levels

In this section the case company's planning practices are described starting from the business owners planning their solutions; products owner planning their components and their releases; and teams doing iteration planning and managing their daily work in the heartbeat level. The roles discussed in this chapter are presented in Table 1. The table also maps the roles on the levels of the agile product management framework and also summarizes each role's main responsibilities.

**Table 1. Roles and responsibilities in the case company**

| Level | Role | Main responsibilities |
|---|---|---|
| Product | Executive team | Approves roadmaps |
| | Solution manager | Prepares roadmaps and projects, defines business goals |
| Release | Product council | Approves project scope, schedule and resources |
| | Project Steering Group | Manages a project within an approved mandate |
| | Project manager | Prepares and coordinates release projects |
| | Resource owner | Manages resource allocation and coordinates teams |
| | Product owner | Defines technical/implementation goals as product backlog items |
| Iteration | Development team | Delivers product increment every iteration |
| Heartbeat | Scrum master | Team's process and methodology coach |
| | Developer | Develops software |

### 3.2.1 Long-term planning

A solution manager is responsible for creating and updating three-year solution roadmap. The roadmap is the most essential part of a business plan for R&D because it sets direction and boundaries for development work in the highest level. The roadmaps contain a schedule of upcoming projects and their high-level business goals, which are called business themes.

The case company's executive team approves roadmaps as a part of the business plans. There is a company policy that three-year roadmaps are confidential and accessible only by executives and solution managers. Because of their sensitive nature I was not able to see either the three-year roadmaps or observe the executive team's meetings.

In addition to the three-year roadmaps there are less sensitive one-year roadmaps, which are created by solution manager in cooperation with other stakeholders within the case company. The one-year roadmaps contain a more detailed release schedule, which contains information also of internal releases, such as alpha, beta and customer pilot releases. The one-year roadmaps do not include market information or resource usage plans. The roadmaps are simple time-line graphs presented in a slide set and they are available in the company's wiki.

The solution managers have a business-oriented view to the solutions. The components that are the building blocks for the solutions have their own technical experts, product owners. The Product owners used to be product managers working in a product management function in the case company. The product managers took care of the long-term planning of the components and worked in close collaboration with the solution managers. But the product management function was discontinued in 2008 and product managers were re-titled and started to work more closely with development teams. The amount of technical long-term planning decreased although some product owners keep their own private backlog of essential features and user needs of their components.

### 3.2.2 Release project preparation

The concepts of project and release are interchangeable in the daily speech in the case company and sometimes mixed to each other. In this study I use the term release to describe a functioning version of software that can be sold to customers. And by the term release project I refer to the planned activity that aims to produce a release.

The case company manages its releases within release projects. The release project preparation starts early enough to get the project started as planned in the roadmaps. A project needs approval for its release project plans from a managerial board called product council.

Figure 7 is a picture that the case company uses to describe how release projects have changed by agile transformation and what is the role of the product council. Previously, resources and time were variables and target contents were "locked" by the product council. After the transition, the product council locks the resources and schedule leaving content as the only variable. The product council sets constraints also for the content but only in terms of high-level business themes or project vision.

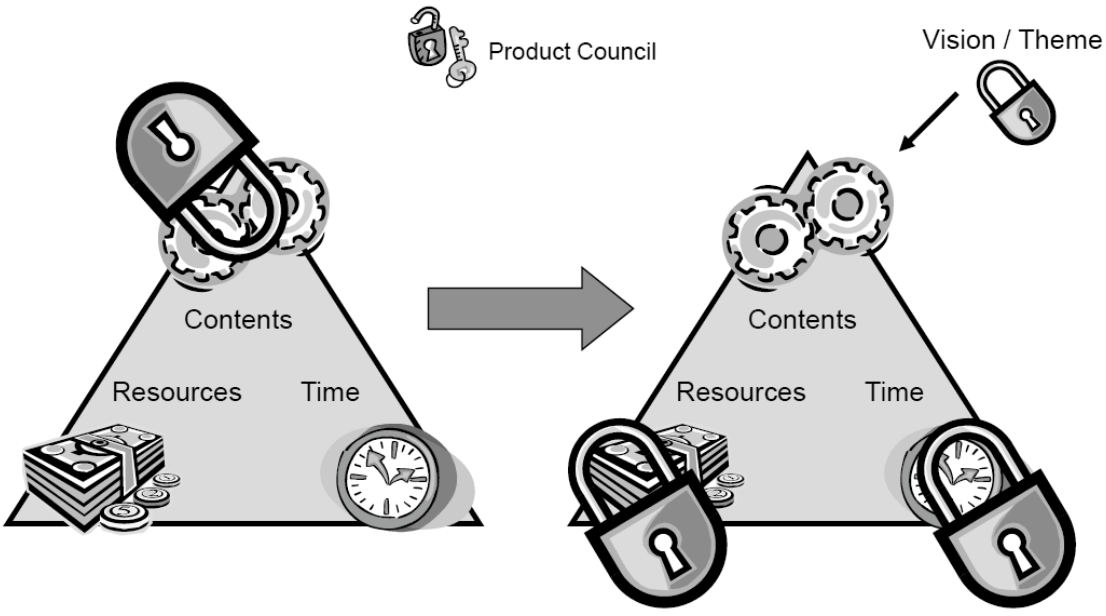


**Figure 7 Release project planning aspects and the role of the product council**

The release project preparation involves the solution manager, a resource owner, a project manager and product owners. In this phase the budget, resources and contents of the release are planned. The initial project plan is presented to product council. It verifies that the planning has been done in a satisfactory manner. The project budget and how it is divided between the business themes is approved. Also the proposal for responsible managers including a project steering group is approved. In addition to the initial planners more managers and experts are appointed to the project steering group.

The product council also monitors that resource plans are balanced and up to date. The release project's resource usage estimates are placed in a spreadsheet that has information of all the ongoing and upcoming release projects and those teams contributing to them.

### 3.2.3 Release project planning

Release projects' management responsibility is divided between project steering group, project manager and product owner. Typically, a solution manager, a project manager, a software architect, a chief quality engineer and project stakeholders outside the R&D form the project steering group that tries to keep the release project within the boundaries set by the product council. They seek ways to keep the project within the approved budget and schedule; manage the project scope so that the business themes are achieved; and they monitor quality of the software. When product managers were re-titled as product owners their responsibilities became more operational and started to overlap with project managers' responsibilities. An example of such overlap is that a project manager is always a member of the project steering group but the product owners usually are not.

The product owner has authority over the actual scope of the software. He defines the technical goals that are called product backlog items and they are kept in a product backlog. In the majority of cases, the product backlogs are spreadsheets kept in a network hard drive. Product backlogs are mainly used and maintained by product owners. The backlogs are accessible for the whole research and development organization.

The product backlogs contain all sorts of information about the product backlog items, such as status, priority, originator, estimate in story points, estimate in hours and risk. For business theme names there is a dedicated column, where the name of the solution theme is written. For the backlogs in spreadsheets there is no support for linking two data items (solution theme and product backlog) or at least it is not used in the case company. Because of this the solution theme name does not update in the product backlog if the solution theme is updated in the solution backlog.

### 3.2.4 Solution project planning

A new concept – solution project – was introduced and taken into use in August 2008. Solution projects were created because the integration of multiple software component releases into a single solution was problematic and responsibilities for integration work were unclear. The concept and relating roles were only partly defined and expected to become better clarified as the first solution projects were carried out. During the observations the core teams and the near-shore team took part in a development effort that was coordinated as a solution project.

A solution project resembles closely a release project only that it is bigger. A solution project has an appointed project manager, a project steering group, and many product owners are involved. One noticeable difference to release projects was that a new type of backlog was introduced. The newly appointed project manager started managing his solution project with a solution backlog. The business themes were split into solution backlog items with assistance from an architect.

A solution backlog is a spreadsheet very similar to the product backlogs. The relation between solution backlogs and product backlogs was unclear at the time of the observations. The product backlog items are split from the business themes as before or in some cases the splitting is done from the solution backlog items. The higher-level object name in the product backlog is written to a column labeled as *category*. The category refers to a business theme, a solution backlog item or an arbitrary product backlog item category that is used for some product backlog items that are related, but there is no precise way of knowing what the type of the high-level object is.

In most cases it is clear to the user what the higher-level object is but in some cases it cannot be said for sure which exact business theme or solution backlog item the product backlog item is linked to because the textual presentation has changed in the solution backlog. Another drawback of this loosely done linking is that only the "parent item" can be seen from the backlog but not the ones that are split from it. Thus it is not easy to see the state of any single backlog item.

Solution backlog items are written from a technical point of view. This lessens the product owners' responsibilities because the technical considerations used to be product owners', but they are not part of the project steering group. When the first solution project started, which involved three major software components and thus three product owners, two of them left the case company. I had no opportunity to interview them so it remains unclear whether their resignation was influenced by changed responsibilities induced either by the new solution project approach or by shift from product manager to product owner.

### 3.2.5 Iteration planning

In iteration planning a development team plans its work for the upcoming iteration which is a 2 – 4 week length time period. The top priority product backlog items are taken from the product backlog and put into an iteration backlog. Product backlog items can be split and only some of them are taken into the iteration. It is product owner's decision what product backlog items are developed in an iteration and his presence is needed in the iteration planning meeting.

Iteration backlogs that are in spreadsheet format are very similar to the product backlogs. Iteration backlog items and product backlog items can be linked by typing the name of a product backlog item into a cell in the iteration backlog. Some teams use a dedicated backlog management tool. When the team's product owner maintains the product backlog in the same tool the linking between the backlog items is automated.

The development team needs to estimate the size of the backlog items so that they can select such an amount of backlog items that they feel confident at completing them in one iteration. Data from previous iterations helps the team to select an appropriate amount of items.

The team needs to be familiar with the top priority product backlog items in order to keep the time frame of the iteration planning meeting feasible. The product owner needs team's input for planning the product backlog items in more detail so that he has enough knowledge of the contents, complexity and size of the product backlog items. In order to meet these requirements the product owner's responsibility is to prepare the product backlog. The activity is referred to as backlog grooming and the meeting is called 5% workshop, which refers to a rule of thumb that 5% of team's time should be used for preparing goals for the upcoming iteration.

### 3.2.6 Heartbeat level

Task planning is done in iteration planning after the iteration backlog items are selected. The development team plans in detail the necessary tasks needed in order to achieve the selected iteration backlog items. In this phase the product owner is no more needed except if the team realizes after task planning that it has too much work to be done. Then the scope is re-adjusted with the product owner. The initial task list and tasks' hour estimates provide the basis for managing the iteration scope. Daily progress is also discussed in daily meetings. The tasks are added, updated and removed as the iteration progresses

## 3.3 Practices and roles for progress monitoring

In this section the practices how progress information is gathered and used is described. The subsections are structured according to the levels presented in the agile product management framework. First subsection describes the heartbeat level where the progress information flow starts. The following subsections describe the higher levels.

### 3.3.1 Heartbeat level monitoring

Heartbeat level progress is monitored in each team's daily status meetings. A scrum master facilitates the meeting. The team members report the tasks they have completed and are working on. They are also expected to report any problems they have encountered. Even though daily status meeting is a simple practice, it is the one that assures that progress gets reported.

### 3.3.2 Iteration level monitoring

In the iteration level progress is visualized with iteration burndown graphs. The burndown is updated when task estimates are changed or tasks are completed. Updating needs to be done manually if the iteration backlog is a physical task board. If spreadsheets are used the burndowns are also updated manually every day because the spreadsheets do not contain enough information e.g. timestamps for every status change or re-estimate.

If a dedicated backlog management tool is used it provides automatically generated iteration burndown graphs. For instance the XPlanner (Morel et al., 2004) provides an overview of iteration progress by listing the iteration backlog items and visualizing the progress of the corresponding tasks.

### 3.3.3 Release project level monitoring

The project steering groups need iteration progress information because they are responsible for managing the release projects and solution projects. But backlogs do not have practically any role in progress monitoring. The project steering groups do not use iteration burndown graphs or velocity calculations to aid their work, mostly because the core teams' do not have them available. Also the information that is in the backlogs is not used. Instead, the project managers gather the progress information and present it to the project steering group.

A practice for visualizing release progress is to maintain a burnup or burndown chart based on release goals i.e. product backlog items. In the case company the information in the product backlogs could be used to generate these graphs if the product backlog items would have estimates, which is not the case in a majority of cases. There is no company policy that requires progress charts or any other progress information; rather it is defined as project managers' responsibility to communicate the project status.

The product council has a tool for tracking resources. It is a spreadsheet that describes the workload of each team at current moment and for about 3-12 months. In theory the workload could be cumulated from the task estimates in the iteration backlogs. But the reality is that the project managers update the spreadsheet manually.

### 3.3.4 Monitoring long-term plans

During the time of conducting this study the product council established a practice called weekly reports that had before been in use in the case company. The ongoing release and solution project status can be checked from the weekly reports, and they provide the highest-level of progress information that was observed during this study. The project managers perform this practice manually and thus the information might or might not be based on information that is available in the backlogs.

The case company's policy is that if a project steering group is not able to keep the project within the mandate that the product council has given to it, the project plan is changed and it needs to be re-approved by the product council. Likewise, roadmap updates needed to be approved by the executive team.

## 3.4 Teams' perspectives on agile planning and monitoring

In this section, the big picture of planning and monitoring agile software development in the case company is outlined from three different perspectives: the startup team's, the near-shore team's and the core teams'. Despite the fact that the case company has a defined process for software development, the actual situation differs in all the observed

teams. The planning and monitoring practices of the teams are described using the same notation that was used in the agile product management framework,

### 3.4.1 The startup team

The startup team's setting closely resembles the agile sweet spot. The team is collocated; the team has one product owner; their development is independent of other teams. Thus, it is not surprising that the agile product management framework has lots of similarities compared to their situation, which is presented in Figure 8.

The startup team is mainly developing a complete solution that has a roadmap approved by the executive team and solution project business themes approved by the product council. They also do smaller maintenance releases for another solution. The startup team develops both of these solutions alone so they do not have dependencies to other teams. The team has access to the one-year roadmap and the business themes but rarely uses them for anything.

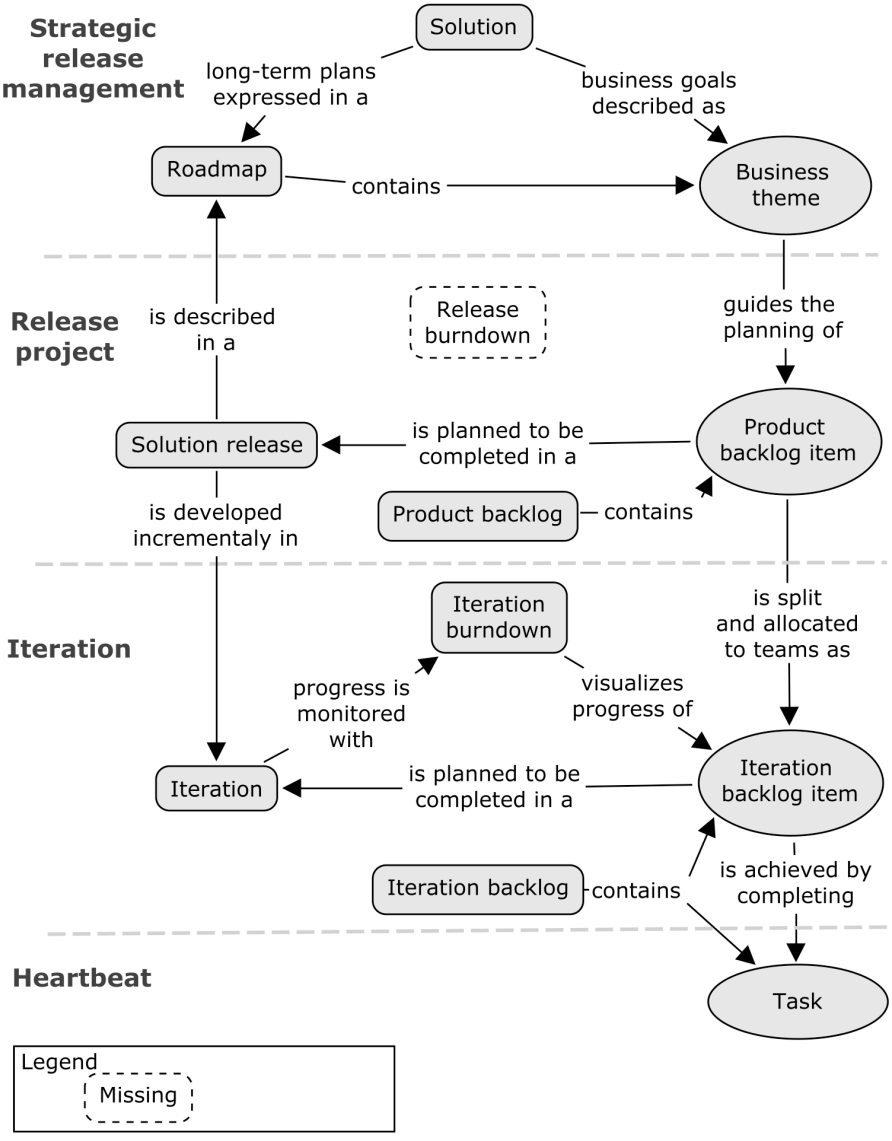


**Figure 8 The startup team in a release project**

The team has a dedicated product owner and their cooperation is frequent. The product backlog is maintained in a spreadsheet but no release burndown graph existed during interviews. The startup team suffers from poor quality of their estimates. Their iteration velocity varied a lot – from 3 to 56 story points. The team thinks that the estimates will settle as they learn more of the new project but at the moment the progress information is quite useless because the estimates cannot be trusted.

The startup team has two-week iterations. Iteration planning is prepared in advance in five percent workshop, in which the top priority backlog items are worked on. The team uses a dedicated backlog management tool – XPlanner – for planning iterations and managing the iteration backlog. In iteration planning the team has to copy items from the product backlog spreadsheet. Thus, there is no links between the iteration backlog items and the product backlog items.

The actual iteration backlog management is easy with XPlanner. The team has tasks planned for the iteration backlog items. The tool automatically provides progress information of each backlog item and of the entire iteration in form of an iteration burndown chart.

The startup team actively manages their daily work. Status meetings are held daily; the status of tasks and stories are constantly updated; work can be mainly carried out as planned during iterations and there are little interruptions. An advantage of using a backlog during daily meetings is that nothing that is put in it gets forgotten in the status meeting.

The team discarded XPlanner and started using Agilefant (Agilefant, 2009) in February 2009. ATMAN researchers assisted and coached the team in using the new tool. Also the product owner discarded the spreadsheets and adopted Agilefant for managing the product backlog. The team and the product owner have been satisfied with the tool.

### 3.4.2 The near-shore team

The near-shore team is rather similar to the start-up team, which is also reflected in the picture (Figure 9) that gives an overview of the team's planning and monitoring practices. A major difference is that they are not developing a complete solution like the start-up team but are developing a component that is used by many of the company's solutions. Therefore the near-shore team has more dependencies to other teams. But on the other hand the dependencies are not that significant as the component is an isolated entity in a technical sense. It is also characteristic for this team that the team and their product owner are not collocated. The setting emphasizes the importance of communication tools, including all backlog management tools.

The component that the near-shore team is developing has a roadmap and business themes that are approved by the product council. The related solutions also have roadmaps and business themes but those are insignificant for the near-shore team because the relevant information should be expressed in the long-term plans of the component.

On the release project level the team's product owner manages product backlog with the XPlanner tool. The tool supports only one kind of backlogs, which forces the product owner to maintain the product backlog as an iteration backlog, which is labeled as product backlog. The backlog items must be copied from the product backlog to the actual iteration backlogs and therefore there is no release burndown graph.

Iteration planning is a straightforward activity for the near-shore team because they use XPlanner for both product and iteration backlog management. Together with the product owner the team selects the top priority product backlog items from the system and moves them into an iteration backlog. The team and the product owner use the five percent workshops to prepare for the upcoming iteration.

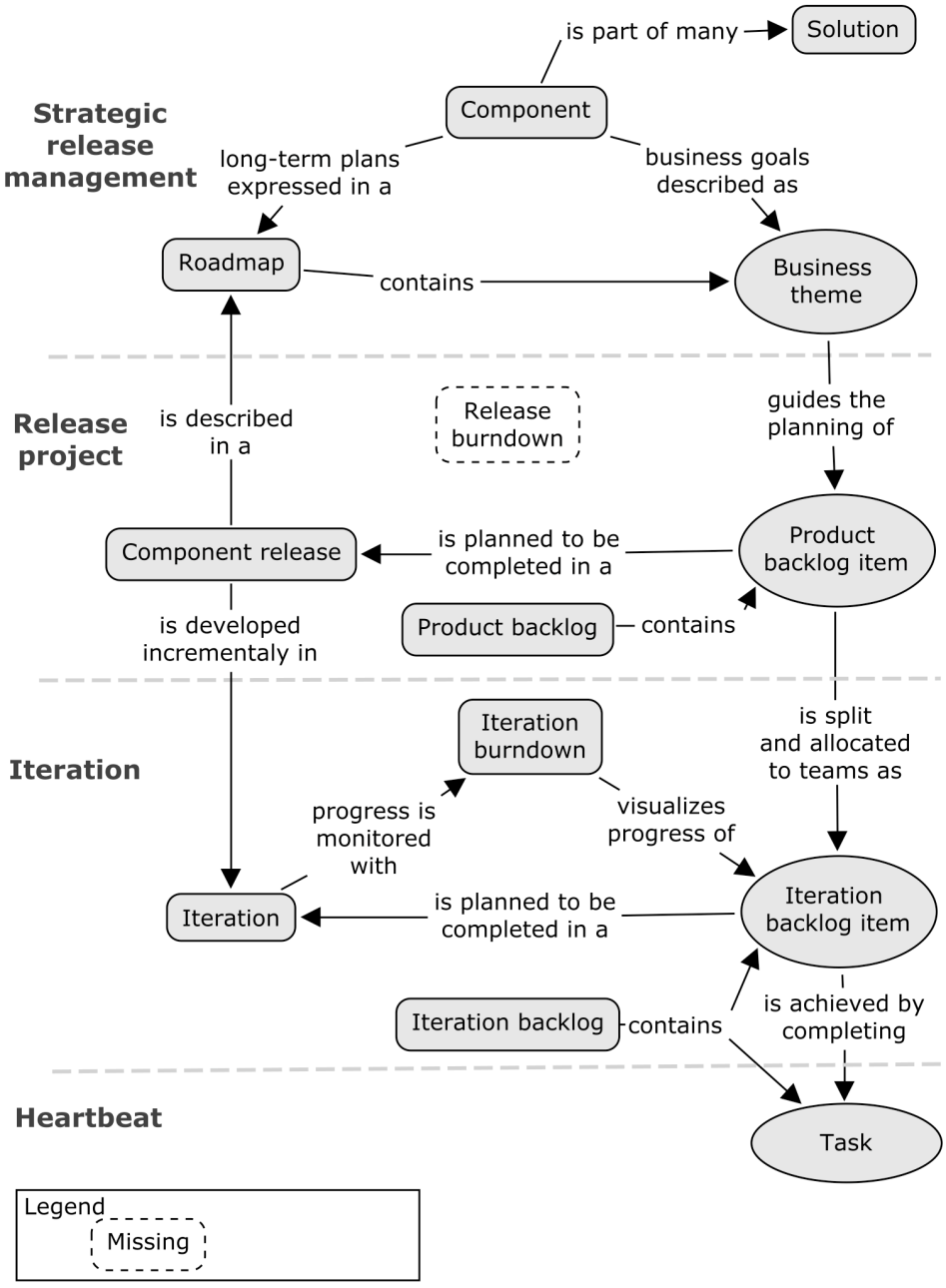


**Figure 9 The near-shore team in a release project**

The team has two-week iterations and they actively use and maintain the iteration backlog throughout the iteration. The team has split the backlog items into development tasks and their status is constantly up to date. The tool automatically updates their iteration burndown chart. The tool and the active usage of it are important because the team is located in the near-shore office and the product owner works in the headquarters in Helsinki. The tool is one essential part of their communication.

Because XPlanner is used for managing both the product backlog and the iteration backlog the team has visibility to the release plans and the product owner has visibility to iteration and release project progress. Even though the team has best visibility of the studied teams it is not enough. The team members are asking to get more visibility into the long-term plans that the product owner has and also are asking to get information of other teams in the organization so that they could better plan their work. There has been some amount rework that the team considered unnecessary and thought it could have been avoided if they had had more information available.

### 3.4.3 The core teams

The core teams' situation is different from the start-up team and near-shore team because there are five teams that have shared responsibility of multiple components. Depending on the situation more than one team might be working on a same component. A team also might be responsible for developing more than one component at a time. Furthermore, the components they are working on are so called 'core components'. These components form the foundation for the case company's solutions. Therefore, the planning boundaries between solutions and components are not clear within the core teams. The multi-team and multi-component setting causes complexity that affects all the levels presented by the agile product management framework. The situation in which the teams are working is presented in Figure 10. Only one team is presented in the figure for clarity. Nevertheless, the complexity of the overall context comes across.

The long-term plans, roadmap and business themes, are approved for a solution by the product council. There are no roadmaps or business themes for the core components, because they would contain mostly the same information as the solution's long-term plans.

In the release project level two sorts of releases are coordinated: the solution release, and the component releases that, when integrated, form the actual solution. To manage the solution release a project manager maintains a solution backlog that consists of solution backlog items. At the time of conducting this study the solution backlog was in its infancy and it was unclear whether the solution backlog would eventually replace product backlogs that the product owners use to manage component releases. Neither the project manager nor the product owners had a release burndown graph. The solution backlogs and product backlogs were spreadsheets that were stored in a network hard drive.

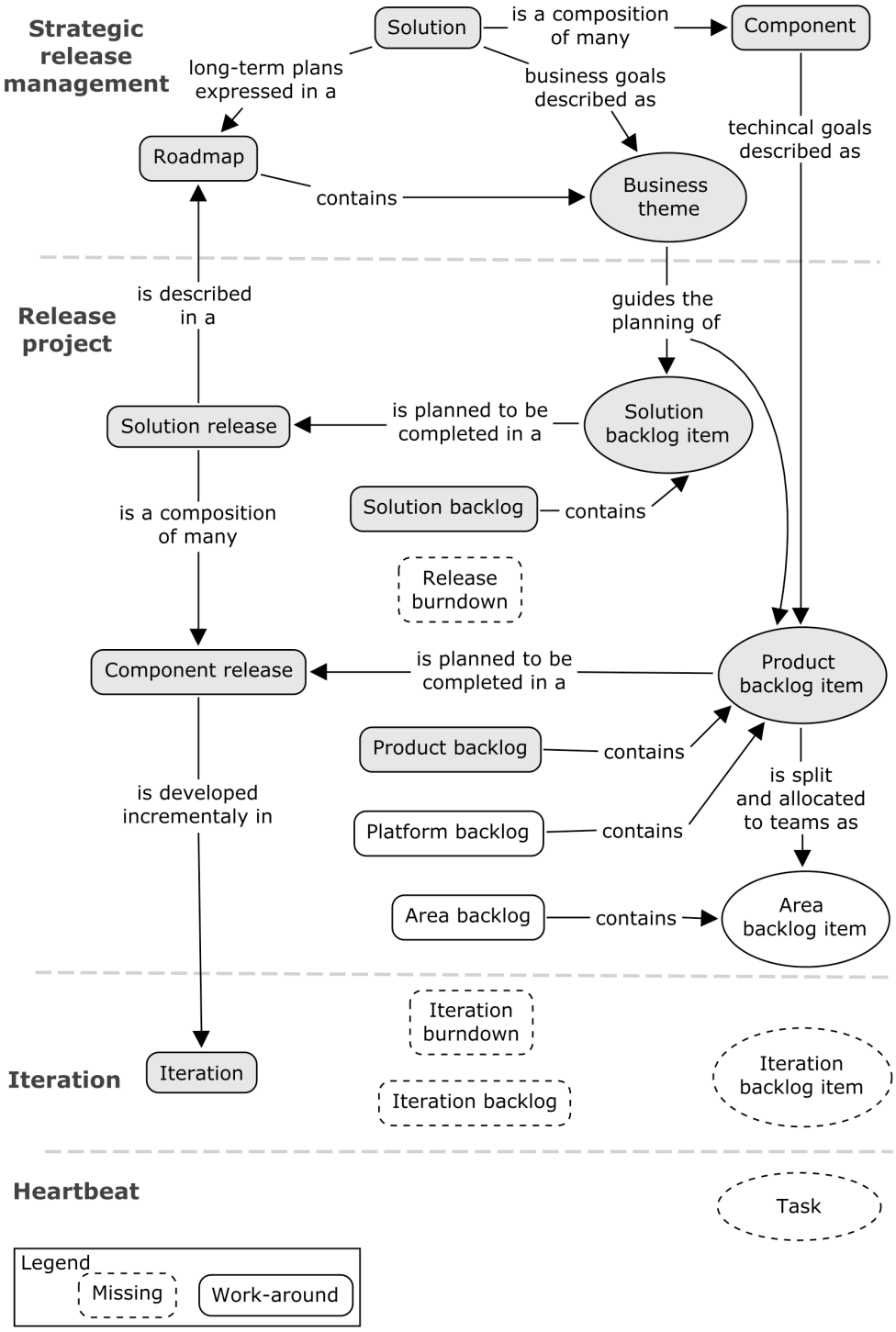


**Figure 10. The core teams in a solution project**

The core teams' resource owner, who is a line manager for the core teams, maintains a platform backlog in cooperation with the product owners to manage release planning that is a complicated effort because the core teams are working on multiple software components simultaneously and thus many product owners are involved. Each product owner provides product backlog items to the platform backlog, which is then prioritized by mutual agreements. The platform backlog is a spreadsheet in company network.

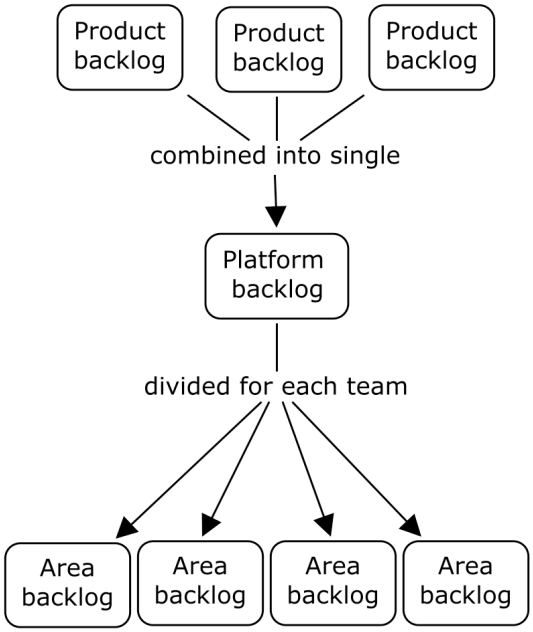


**Figure 11. An example of a backlog structure used by the core teams**

Because each team is specialized in some areas such as user interface development or database development, the teams cannot complete platform backlog items in priority order. The platform backlog is therefore divided into so-called area backlog that match with teams' competencies. Also the product backlog items in the platform backlog must be split into area backlog items that match with the teams' competencies. The relation between product backlogs, platform backlogs and team backlogs is illustrated in Figure 11.

The core teams have four-week iterations. And in iteration planning the product backlog items should have been taken from an area backlog and put into an iteration backlog. But at the time of conducting the case study the core teams had decided not to maintain a separate iteration backlog or do any task planning at all. A scrum master commented this decision:

> *"It was considered as waste. And besides, everyone voted on this [abandoning the iteration backlog and task planning] in the iteration retrospective."*

In order to manage their iterations the teams now verbally agree on which product backlog items are taken from the area backlog or straight from the platform backlog. The scrum master reasoned:

> *"One can easily memorize a few backlog items and thus there is no need for spreadsheets or post-its"*

The core teams are not cross-functional feature teams as suggested in literature; they do not have only one product owner and they do not have a clearly dedicated product owner but a group of managers coming and going. Apparently, the platform backlog related

planning practices were complicated and took so much core teams' time that it discouraged the teams to do any further planning.

Naturally, there were no iteration burndown graphs as there was no proper iteration backlogs, tasks nor estimates. Also links between different backlog items were non-existent. The linkage between the product backlog items in the platform backlog and the area backlog items in the area backlogs could be provided with textual links in the spreadsheets. But the inevitable changes in the product backlog items would reflect to many area backlog items in many area backlogs and the update would require a lot of manual work. This is simply too laborious and therefore there are no links between backlog items in the team backlogs and the original product backlog items.

Because there are no tasks the core teams' status reporting is based on mutual reporting in meetings. The progress information is needed not only by the team itself but the other teams that are dependent on each other's work. Project managers, product owners, architects and quality engineers also need this information. Because of the amount of stakeholders there are lots of meetings. During the observations the most repeated single phrase by frustrated managers and developers was:

*"There are so many meetings that I don't have time to do my work"*

The core teams' and other stakeholders' understanding of iteration progress is based on the daily status meetings. It seemed that the whole concept of iteration had become blurred because the progress was not monitored nor actively managed. During a lecture held by Mr Craig Larman in the case company, a member of the core teams commented after Larman had presented his view of backlogs, user stories and tasks:

*"I think that we are not agile after all"*

Later, the situation of the core teams was referred to as "an illusion of agile" within the case company.

## 3.5 Improvement efforts

The case company made seven improvement efforts (IE1 – IE7) that potentially impact backlogs and their usage in linking long-term plans with daily development tasks. The improvement efforts are presented in order of appearance. They are also presented on a timeline in Figure 1.

**IE1. The core teams were re-organized as cross-functional feature teams**

During October and November 2008 the core teams were reorganized as cross-functional feature teams. The teams were responsible for forming five teams that could deliver end-to-end features i.e. work on a backlog item from start to finish. The core teams' release and iteration planning became much more straightforward. Firstly, they got rid of the platform backlog. Secondly, the product owners could prioritize product backlog items and teams could work in priority order. Thirdly, less splitting of backlog items is needed,

which causes the backlogs to be simpler. And fourthly, less coordination is needed between the core teams.

**IE2. The core teams re-introduced agile practices in a process improvement effort**

After been organized as feature teams the core teams had an agile awakening. They had dropped many of the agile practices but now the basic planning and monitoring practices were re-introduced and the teams even started to look for additional agile practices.

During November and December 2008 the core teams went through a training period where agile working practices were recapped and trained. The focus was on iteration level and daily practices. A consultant was hired to help the teams. I attended weekly status meetings that were held during the improvement period. The initial goal was to learn test-driven development and start using continuous integration. But soon the focus was shifted on learning Scrum practices. This was because the teams felt that they could not properly plan their iterations. It turned out that the underlying problem was absence of the product owners. Even thought the teams started to use agile planning and monitoring practices the surrounding organization was not as fit as the teams would have liked. The core teams chose a new tool for managing the iteration backlogs: a white board caused far less frustration than the spreadsheets had.

**IE3. Organizational change affected especially the product owner role**

In December 2008 the case company started to prepare for an organizational change. ATMAN researchers took an active role in commenting and advising the planning of the new organization. Especially, the product owner role was taken into careful consideration, mostly because of the core teams' needs. In the beginning of 2009 the organizational change was rolled out. The core teams gained a better access to product owners. Even though the new organizations is such that a product owner might have two teams to work with at the same time, each team has only one product owner per release project.

**IE4. The startup team and their product owner chose a common tool, Agilefant, for managing the product backlog and the iteration backlogs**

The findings made when studying the startup team and the resulting feedback lead the startup team and their product owner to choose a common tool for managing their product backlog and iteration backlogs. In February 2009 the team and the product owner started piloting and eventually took in use Agilefant. Another ATMAN researcher assisted the team and the product owner in the adoption. Compared to the other improvement efforts this one does not have as great an impact. It has, however, simplified iteration planning and the product owner has considered the tool as a better option than the spreadsheets. XPlanner was not considered as a viable option because its active development has ceased back in 2002.

**IE5. R&D organization's software development process was updated and Leffingwell's hierarchical backlog item model became part of it**

The R&D organization realized that the current process for software development had outdated. Also the organizational change impacted software development practices. In May 2009 the work for updating the case company's process description started. The new process model describes in more detail the role of backlogs in the software development process. Again, the ATMAN researchers were closely involved in the preparation of the new process model.

The ATMAN researchers found out that Mr Dean Leffingwell had carried out work (Leffingwell 2007, 2009a, 2009b) that might be beneficial to the case company's R&D organization. The matter was arranged so that in September 2009 Mr Leffingwell visited the case company and gave a presentation and training session of agile methods in large scale software development. As an outcome the case company adopted Leffingwell's hierarchical backlog item model as a part of the new process model

The case company ended up in a process model that is, at least in terms of backlogs and requirements, markedly more complicated than the Scrum model, especially so in the case of the hierarchical backlog items. The case company sacrificed simplicity but considered it necessary in order to match the development process with their current organization's needs. The new backlog item model provides more unambiguous priorities and enables coordination of multiple component releases that are developed for a larger solution.

**IE6. A common tool, VersionOne, was piloted for all long-term planning and highest-level backlog management in the R&D organization**

Immediately after the hierarchical backlog item model was adopted, the case company started to evaluate tools that support the new process model. The spreadsheets had been used for managing backlogs but they were not very good at it. When the hierarchical backlog items were introduced, the spreadsheets were not even seriously considered to cope with them.

The ATMAN researchers assisted in analyzing the requirements for the tool. As a result the case company chose VersionOne (VersionOne, 2009) to be used for managing long-term planning and release planning. The tool was not recommended for iteration planning since licenses for the tool were considered pricey and nobody was sure whether the tool could be used in the long run.

**IE7. All development teams participated in a joint release planning session in the beginning of a solution project.**

In December 2008 when the organizational change was in preparation phase the ATMAN researchers suggested arranging a joint release planning session. The idea was to bring all the teams together in the beginning of a solution project. The goal was to create a shared vision to the teams and create synchronized iteration plans at least for the initial iteration.

A year later such event was arranged and Mr Leffingwell was invited to facilitate the session.

The joint release planning practice is a promising and interesting practice for large-scale agile development. The goal is to improve communication and cooperation between the development team so that dependencies are better managed. No matter how good tools and backlog management practices the case company has, the bottom line is that mutual understanding of the work at hand is needed. And the joint release planning is just for that.

# 4 Discussion

In this section the case company's challenges in linking long-term plans and daily development tasks are discussed. The interplay of organization, practices and tools are discussed. Also the improvement efforts and their impact are talked about.

## 4.1 Challenges in visibility

The case company's management had poor visibility to development teams' progress. The development teams had a very limited view of the long-term plans and to other teams. It is difficult to separate whether the organization, the practices, or the tools cause the visibility challenge because they all affect each other. It looks like the case company was struggling in a vicious cycle that reinforced itself. Inappropriate planning practices and the complicated relations between product owners and the core teams caused the challenges in visibility and were further worsened by the spreadsheets that were used for backlog management. All this demoralized the developers, which in turn caused the planning and monitoring practices to corrupt, ultimately leading to abandoning iteration-level planning and monitoring practices.

The core teams and the startup team seemed to work in isolation of the long-term plans; they worked based only on the information provided by the product owners. The core teams had sometimes difficulties even having access to a product owner. The near-shore team was the only of the observed teams that demanded visibility to other teams and tried to understand the bigger picture.

## 4.2 Challenges in roles and responsibility

It seems that in large-scale software development the amount and quality of communication that a product owner can have with developers is reduced significantly by coordination work that stems from working with multiple teams. The case company had realized the challenge and had divided the product owner responsibilities among a number of managers. But they had not considered the team structure that much. As I interviewed the managers and teams, I got the picture that they all are hard working and doing good job that was relevant for the success of the company. It all sounded logical. But when put together i.e. to form a big picture, the pieces did not completely fit together.

The product owner role described in literature was split among four different roles in the case company: solution manager, resource owner, product owner, and project manager. Since each manager was responsible for a specific area, responsibility of overarching issues, such as visibility of plans, were not clearly appointed. In addition to these four managerial roles, three different types of councils – the executive team, the product council and the project steering groups – had taken some of the product owner role's responsibilities.

The core teams' complicated backlog structure was against the suggestions found from the literature. But it actually made sense because of the way the teams were organized around the specific functional domains. This arrangement had two major downsides. It

made collaboration of the product owners and the teams difficult. And furthermore, the resulting practice of working with the platform backlog made little sense to the developers, which gave them a negative image of agile practices. The transformation of the core teams into the cross-functional feature teams had all the difference to them. It was as if the teams were unlocked and they started to progress towards an agile way of working.

Agile literature emphasizes that teams should be self-organizing and have the responsibility and duty to improve their own work. I think it is fair to say that the core teams misused this power when they abandoned the iteration and heartbeat level practices. The teams had created practices that minimized the need of the product owners. When the core teams started to work as feature teams and agile practices were enforced, they started to use their power in the correct way. They started to demand better access to the product owners. This lead to the introduction of a renewed organization and it seemed as if the puzzle had started to untangle itself.

After the introduction of feature teams, the core teams could develop a feature on their own. In addition, the product owners' work should be simplified because they can work with one or two teams at a time. Being a feature team does not remove the need for multi-team coordination. But it does not need to be done on a daily basis for every single feature; the focus of the coordination moves to the iteration and the release level. The teams should have more time for coordinating themselves because the daily hassle has reduced.

### 4.3 Planning practices

A backlog is a useful and lightweight tool for prioritization and planning, but it is not a silver bullet that would automatically communicate customer's needs. A backlog item is only a one-liner that possibly is not self-explanatory for outsiders. Beneath every backlog item there should be a more detailed conversation, additional written specification, or whatever necessary to understand the work at hand. At least a backlog item should denote that such planning and conversations should be done.

The complicated backlog structure that the core teams had, amplified the visibility challenges caused by the lack of linking in the spreadsheets. Although the practice of having the platform backlog and the area backlogs helped the teams and product owners in the planning phase, it did not provide virtually any benefits in progress monitoring and in adjusting the plans. In fact, the planning practices were so complicated that it took too much of the developers time. The underlying cause was the dependencies between the teams.

Improving the organization can reduce dependencies, but they will not disappear completely because the teams are still working on the same software. Practices for managing the dependencies and effectively coordinating the teams' and product owners' work are still needed. The joint release planning practice is one of the few practices that are specific for managing large-scale agile software development. Again, it is not a silver bullet but when used in an appropriate organization this is a practice that should

effectively enhance understanding early in a release project. The joint planning session should lay a foundation for the teams to understand their work in a larger context. Knowledge of the dependencies is surely needed if they are to be managed.

## 4.4 Tools for backlog management

In the case company the backlogs were created and maintained so that relevant information was scarcely available for planners or developers. The backlogs did provide some but not adequate visibility. The spreadsheets had some links between the backlog items, but the links were broken in many cases. The backlogs did not provide visibility to plans nor automatically escalated relevant progress information.

The XPlanner tool seemed to provide better visibility to plans and progress information than the spreadsheets. The near-shore team had the best visibility since their product owner used the same tool for managing the product backlog. But still the team demanded for even better visibility. The startup team and their product owner adopted Agilefant to gain the same benefits the near-shore team had.

A big part of the case company had learned to use spreadsheets for backlog management. There was no company-wide instruction or standardization for effective use of the backlogs. To me the usage of spreadsheets looked very limited and a lot of manual work needed to be done to gather progress information. It is not known why the spreadsheets were taken in use in the first place; probably the spreadsheets are considered a de-facto solution for backlog management.

The spreadsheets were of little use to bringing the long-term plans to the development teams and in providing progress information to management. The vast majority of examined backlogs were spreadsheets. The weakness compared to dedicated backlog management tools is that creating a link between, say, product backlog item and iteration backlog item is too hard for every day use. Instead of creating and maintaining references or unique identifiers in spreadsheets users copy and paste backlog items' names from one backlog to another.

At the time of iteration planning the copy-and-paste practice might seem appropriate but once the inevitable changes occur and either the product backlog or iteration backlog is updated they become unsynchronized. If a product backlog item name needs to be updated the product owner could do the update in both backlogs. Without the link it is of course more difficult to find the relating iteration backlog item to be updated. There might also be multiple iteration backlog items related to a single product backlog item and they can be scattered around many teams' iteration backlogs.

The core teams' improvement efforts (IE1 and IE2) and the organizational change (IE3) brought their focus back on the backlogs and tools. The platform backlog and area backlogs were abandoned. The product owners continued to use the spreadsheets but the core teams took the most simple tool for managing their iteration backlogs: a whiteboard.

It seems that in large-scale development, where multiple teams need to be coordinated in order to produce a larger whole, the simplest solution for managing requirements is not enough. It is probable that the spreadsheets are about to disappear in the case company. Because the new process model introduced a more complicated backlog item structure, a dedicated backlog management tool is needed. VersionOne is a tool that copes with many of the case company's requirements. The key feature over the spreadsheets is that VersionOne can manage backlog items that form a hierarchy, and it still works for prioritizing. However, the tool was not recommended for development teams to use in iteration management but was used for managing long-term and release-level plans. Even this tool is not the ultimate solution for the case company if they cannot use it for iteration management.

## 4.5 Limitations and threats to validity of the results

Being a single case study the external validity is not strength of this work. It is questionable whether the results of this study can be generalized to any other context. Also similar work has not been carried out previously so there is nothing to compare the results to. The study can be considered as a descriptive case study, which provides knowledge of this particular context. Only further research on the same topic can use this study as comparison.

A number of different settings within the case company were studied in this study. This deducts the risk that research data would have been gathered from a special case that does not apply with the rest of the case company. It is probable that the three rather different settings – the core teams, the near-shore team and the startup team – provide a good sample of the case company.

The external and internal validity is compromised because of limited access to development teams. The startup team was not observed or interviewed, only their Scrum master and product owner. The case company was unwilling to disturb the development teams. So, the core teams were observed only during their process improvement effort.

Internal validity is threatened if the researcher did not understand the situation in the case company correctly. This risk is reduced by the fact that the researcher has education on the field of software engineering and has previous knowledge of agile software development. Furthermore, if the study has flaws in the internal validity then the construct validity is also compromised. Any misunderstandings have ended up into the results section of this study.

The risk of subjective bias is reduced because other ATMAN researchers are also studying the case company and we have a shared understanding of the situation. The risk is further reduced by triangulation. The data was gather with interviews, observation of many different events and by inspecting the documents that were discussed in the interviews i.e. the backlogs.

# 5 Conclusions and future work

In this final chapter I draw conclusions based on the result and present how the research questions were answered. I also suggest directions for future research.

## 5.1 Conclusions and answering the research questions

Agile software development has reached organizations and projects that are too large to fit in the agile sweet spot and yet the organizations seek ways to benefit from the flexibility that agile methods are claimed to foster. Studies on this topic are scarce but some reports have emerged. In this study the focus was on backlogs and how they could be used for communicating long-term product and business plans to software development and how they can be used for progress monitoring, especially in large-scale agile development.

**The research questions RQ1 and RQ2**

The research questions RQ1 and RQ2 were about discovering suggested practices for using backlogs to communicate long-term plans to developers; and to communicate progress information to managers in agile software development. A major contribution of this work is the agile product management framework (Figure 5 at page 21) that was constructed from the results of the literature study. The framework is based the Cycles of Control (Rautiainen, 2004) framework that presents four time horizons of iterative product development; and on three different models for scaling the backlogs and relating practices: Schwaber's (2007), Larman's (2009) and Leffinwell's (2007, 2009a, 2009b). The concepts of backlog management were plotted on the time horizons and the relations of the concepts were presented. The framework presents one possible interpretation of what backlogs are and what is their role in planning and monitoring agile software development.

**The research question RQ3**

The research question RQ3. How is a product's goal and progress information communicated in the case company was answered thoroughly by reporting case study results in chapter 3. In the case study I used the framework to guide data gathering and also in analyzing the data. With the startup team and the near-shore team the case study results were reasonably well in line with the framework. But in case of the core teams, that present the true scaled agile development, the big picture in Figure 10 at page 35 showed cluttering of the release project level with workarounds. The framework showed that the iteration and heartbeat levels were practically missing. The framework revealed that the core teams lived in an illusion of agile, which they also found out by themselves. In this study I found out the framework to be useful, and I recommend it to be used in a similar way in other cases too.

**The research question RQ4**

The research question RQ4 was about identifying challenges in communicating the product's goal and progress information, which was done during the case study. The case

company faced challenges in bringing the long-term plans gradually down to the development teams in a form that they could understand and use effectively. Another challenge was the lack of progress information. These challenges were coined from three concerns: the organization of the development teams and the product owners, inadequate planning practices and tools that amplified the challenges. Based on the results of this study it seems that all of these three aspects need to be considered together if the challenges are to be successfully acted on.

In agile software development the plans are communicated effectively with close collaboration of product owners and development teams. Collocation of a team improves its internal communication and knowledge sharing. Backlogs are merely a simple tool that keeps discussions focused during planning. It is an effective tool if some basic practices are used. Should there be lots of collaboration between managers and teams I see no obstacles for effectively bringing the long-term plans to the developers and progress information to the managers. However, when agile is used in a large-scale development the managers and teams are burdened with additional coordination work that is caused by dependent work carried out in many teams. Anything additional is of course away from the communication of managers and teams. It might even cause such a chaos that the developers try to get rid of agile instead of trying to improve it and modify it to match with challenging situation. This was the case with the core teams. The all too difficult planning practices that were in use because of the organization of the teams and the product owners had such a de-motivating effect on the core teams that led them to abandon many of the agile practices.

**The research question RQ5**

There are few options when a company hits the panic zone of agile software development. It could drop the agile and do something else; it could try to get rid of the dependencies between teams by developing smaller projects or developing products slower; or it might as the case company did, do a complete transformation which includes the organization of roles and responsibilities, planning and monitoring practices and tools. The case company went thorough an organizational change where the product owners were brought closer to the development teams. They took a more complicated but flexible view on backlogs by adopting Leffingwell's model of hierarchical backlog item. And by doing so, they were practically forced to seek a tool to cope with the situation. They chose, at least for the time being, VersionOne because it was evident to them that the spreadsheets had met their limits. All the improvement efforts were reported in section 3.5, which provides answer to the last research question RQ5 What could be improved to overcome the identified challenges.

## 5.2 Future work

The case company went through a series of improvement efforts. In this work I described the situation, analyzed it and described the case company's improvement efforts but I was unable to follow-up their effects. In order to understand what improvement efforts have impact on communicating the long-term plans and progress information more inquiries

should be made in the case company. Especially, the joint release planning should be studied because it is one of the few practices that are specific for large-scale agile software development that could have potential to increase communication. The case company provides an appealing setting for studying tool requirements; how they excel and what are the shortcomings.

Knowledge of the product owner role in large-scale agile software development is scarce. It clearly requires further research to find out different approaches to organize product ownership and to compare their suitability in different contexts. It is probable that any company doing large-scale agile software development will face challenges on organizing product ownership.

The framework constructed in this study might be used within the same case company to examine if and how backlogs have changed. Applying the framework in other cases probably would shed more light on how backlogs are and can be used in large-scale agile software development. Additionally, more case studies should be conducted with the framework in order to validate it further.

## References

ABRAHAMSSON, P., SALO, O., RONKAINEN, J. and WARSTA, J., 2002. Agile Software Development Methods: Review and Analysis. VTT Publications 478, VTT, Finland.

AGILEFANT, 2009. Agilefant. Available at:www.agilefant.org (accessed 17 November 2009).

AMBLER, S., 2007. Survey Says...Agile Has Crossed the Chasm. Dr. Dobb's Journal: TechWeb, United Business Media LLC. Available at: http://www.ddj.com/architect/200001986?pgno=1 (accessed 17 November 2009)

AMBLER, S., 2006. Survey Says: Agile Works in Practice. Dr. Dobb's Journal: TechWeb, United Business Media LLC. Available at. http://www.ddj.com/architect/200001986?pgno=1 (accessed 17 November 2009)

BECK, K., 2000. eXtreme Programming eXplained. Addison-Wesley.

CAO, L. and RAMESH, B., 2008. Agile Requirements Engineering Practices: An Empirical Study. *Software, IEEE*, 25(1), 60-67.

CHRISTENSEN, C.M. and RAYNOR, M.E., 2003. The Innovator's Solution: Creating and Sustaining Successful Growth. Boston: Harvard Business School Press.

COCKBURN, A., 2002. Agile software development. Pearson Education.

COHN, M., 2008. Agile Estimating and Planning (Robert C. Martin Series). 7th edn. Prentice Hall PTR.

COHN, M., 2004. User Stories Applied: For Agile Software Development (The Addison-Wesley Signature Series). Addison-Wesley Professional.

DINGSØYR, T., DYBÅ, T. and ABRAHAMSSON, P., 2008. A Preliminary Roadmap for Empirical Research on Agile Software Development, T. DYBÅ, ed. In: *Agile, 2008. AGILE '08. Conference,* 2008, pp83-94.

DYBÅ, T. and DINGSØYR, T., 2008. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9-10), 833-859.

FOWLER, M. and HIGHSMITH, J., 2001. The Agile Manifesto. *Software Development*, 9(8), 28-32.

LARMAN, C. and BASILI, V.R., 2003. Iterative and incremental development: a brief history. *Computer*, 36(6), 47-56.

LARMAN, C. and VODDE, B., 2009. Scaling Lean & Agile Development: Thinkingand Organizational Tools for Large-Scale Scrum. Addison-Wesley.

LEFFINGWELL, D., 2009a. The Big Picture of Enterprise Agility. Available at: http://scalingsoftwareagility.files.wordpress.com/2007/03/the-big-picture-of-enteprise-agilitywhitepaper.pdf (accesed 17 November 2009)

LEFFINGWELL, D., 2009b. A Lean and Scalable Requirements Information Model for Agile Enterpises. Available at: http://scalingsoftwareagility.files.wordpress.com/2007/03/a-lean-and-scalable-requirements-information-model-for-agile-enterprises-pdf.pdf (accesed 17 November 2009)

LEFFINGWELL, D., 2007. Scaling Software Agility: Best Practices for Large Enterprises. USA: Addison-Wesley.

LOWERY, M. and EVANS, M., 2007. Scaling Product Ownership, *AGILE 2007,* 2007, pp328-333.

LYON, R. and EVANS, M., 2008. Scaling Up Pushing Scrum out of its Comfort Zone, *Agile, 2008. AGILE '08. Conference,* 2008, pp395-400.

MOORE, R., REFF, K., GRAHAM, J. and HACKERSON, B., 2007. Scrum at a Fortune 500 Manufacturing Company, *AGILE 2007,* 2007, pp175-180.

MOREL, M., SUN, G., MOWER, K., PROKIPOWICZ, M. and SIWIEC, T., 2004. XPlanner. Available at:www.xplanner.org (accessed 17 November 2009).

PATTON, M., 2002. Qualitative Research & Evaluation Methods. 3 edn. Sage Publications, Inc.

PAETSCH, F., EBERLEIN, A. and MAURER, F., 2003. Requirements engineering and agile software development, *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on,* 2003, pp308-313.

PATTON, M., 2002. Qualitative Research & Evaluation Methods. Third Edition edn. Sage Publications, Inc.

PRESSMAN, R., 2004. Software Engineering: A Practitioner's Approach. 6 edn. McGraw-Hill Science/Engineering/Math.

RAUTIAINEN, K. and LASSENIUS, C., 2004. Pacing Software Product Development: A Framework and Practical Implementation Guidelines. Technical Report 3. Espoo: Helsinki University of Technology, Software Business and Engineering Institute.

RAUTIAINEN, K., VUORNOS, L. and LASSENIUS, C., 2003. An Experience in Integrating Strategic Product Planning and Agile Software Development Practices, *Proceedings of 2003 International Symposium on Empirical Software Engineering*, 30.9.-1.10. 2003, IEEE Computer Society pp28-37.

ROTHMAN, J., 2009. Manage Your Project Portfolio. 1st edn. Pragmatic Bookshelf.

ROYCE, W.W., 1970. Managing the Development of Large Software Systems: Concepts and Techniques, *Proceedings of Wescon*, August 1970, IEEE pp1-9.

SCHWABER, K., 2009. Scrum Guide. Available at: http://www.scrumalliance.org/resource_download/598 (accessed: 17 November 2009)

SCHWABER, K., 2007. The Enterprise and Scrum. USA: Microsoft Press.

SCHWABER, K., 2004. Agile Project Management with Scrum. Microsoft Press.

SCHWABER, K., 1995. SCRUM Development Process, Proceedings of the 10th Annual *ACM Conference on Object Oriented Programming Systems, Languages, and Applications* (OOPSLA, 1995, pp117-134.

SCHWABER, K. and BEEDLE, M., 2002. Agile software development with Scrum. Prentice Hall.

SUSMAN, G.I. and EVERED, R.D., 1978. An Assessment of the Scientific Merits of Action Research. *Administrative Science Quarterly*, 23(4), 582-603.

SUTHERLAND, J., SCHOONHEIM, G. and RIJK, M., 2009. Fully Distributed Scrum: Replicating Local Productivity and Quality with Offshore Teams, *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on,* 2009, pp1-8.

SZALVAY, V., 2006. Hierarchical Requirements and Scrum. Available at: http://blogs.danube.com/wp-content/uploads/Hierarchical-Requirements-and-Scrum-blog.pdf (accessed: 17 November 2009).

TURK, D., FRANCE, R. and RUMPE, B., 2005. Assumptions Underlying Agile Software-Development Processes: *Journal of Database Management*, 16(4), 62-87.

VERSIONONE, 2009. VersionOne. Available at:www.versionone.com (accessed 17 November 2009).

YIN, R.K., 1994. Case Study Research: Design and Methods. Second edn. London: SAGE Publications.

# Appendix A

## Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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.

**Principles of the Agile Manifesto**

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.