Project Management Tools in Agile Embedded Systems Development

Master's Thesis
University of Turku
Department of Information Technology
Software Engineering
2014
Samuli Suomi

Supervisors:
Tuomas Mäkilä
Ville Leppänen

UNIVERSITY OF TURKU
Department of Information Technology

SAMULI SUOMI: Project Management Tools in Agile Embedded Systems Development

Master's Thesis, 92 p.
Software Engineering
June 2014

Agile methods have become increasingly popular in the field of software engineering. While agile methods are now generally considered applicable to software projects of many different kinds, they have not been widely adopted in embedded systems development. This is partly due to the natural constraints that are present in embedded systems development (e.g. hardware–software interdependencies) that challenge the utilization of agile values, principles and practices. The research in agile embedded systems development has been very limited, and this thesis tackles an even less researched theme related to it: the suitability of different project management tools in agile embedded systems development.

The thesis covers the basic aspects of many different agile tool types from physical tools, such as task boards and cards, to web-based agile tools that offer all-round solutions for application lifecycle management. In addition to these two extremities, there is also a wide range of lighter agile tools that focus on the core agile practices, such as backlog management. Also other non-agile tools, such as bug trackers, can be used to support agile development, for instance, with plug-ins.

To investigate the special tool requirements in agile embedded development, the author observed tool related issues and solutions in a case study involving three different companies operating in the field of embedded systems development. All three companies had a distinct situation in the beginning of the case and thus the tool solutions varied from a backlog spreadsheet built from scratch to plug-in development for an already existing agile software tool. Detailed reports are presented of all three tool cases.

Based on the knowledge gathered from agile tools and the case study experiences, it is concluded that there are tool related issues in the pilot phase, such as backlog management and user motivation. These can be overcome in various ways depending on the type of a team in question. Finally, five principles are formed to give guidelines for tool selection and usage in agile embedded systems development.

Keywords: agile development, embedded systems, project management tools, backlog

TURUN YLIOPISTO
Informaatioteknologian laitos

SAMULI SUOMI: Projektinhallintatyökalut sulautettujen järjestelmien ketterässä kehityksessä

Diplomityö, 92 s.
Ohjelmistotekniikka
Kesäkuu 2014

Ketterien menetelmien suosio on jatkuvassa kasvussa ohjelmistokehityksen saralla. Vaikka ketterien menetelmien ajatellaan yleisesti olevan soveltuvia hyvinkin erilaisiin ohjelmistoprojekteihin, niitä ei olla laajamittaisesti otettu käyttöön sulautettujen järjestelmien kehitykseen. Tähän on osasyynä tällaisten järjestelmien kehityksen luontaiset rajoitteet (kuten laitteiston ja ohjelmiston väliset riippuvuudet), jotka tekevät haasteelliseksi ketterien arvojen, periaatteiden ja käytäntöjen omaksumisen. Aiheen aiempi tutkimus on ollut rajoittunutta, ja tämä tutkielma käsittelee siihen liittyvää ja vieläkin vähemmän tutkittua teemaa: erilaisten projektinhallintatyökalujen soveltuvuutta ketterään sulautettujen järjestelmien kehitykseen.

Aluksi erilaisia ketteriä työkaluja käydään läpi fyysisistä työkaluista, kuten tehtävätauluista ja -korteista, web-pohjaisiin työkaluihin, jotka tarjoavat monipuolisia ratkaisuja ohjelmiston elinkaaren hallintaan. Näiden kahden ääripään välillä on myös laajalti erilaisia kevyempiä ketteriä työkaluja, jotka keskittyvät ketteryyden keskeisimpiin käytäntöihin, kuten tuotteen backlogin hallintaan. Lisäksi ei-ketteriä työkaluja, kuten bugienhallintajärjestelmiä, voidaan käyttää ketterän kehityksen tukemiseen esimerkiksi plug-inien avulla.

Selvittääkseen sulautettujen järjestelmien ketterän kehityksen työkalutarpeita kirjoittaja tarkkaili työkaluja koskevia ongelmia ja ratkaisuja kolmessa sulautettuja järjestelmiä kehittävässä yrityksessä. Kaikilla kolmella yrityksellä oli hyvin erilainen tilanne tapaustutkimuksen alussa, minkä vuoksi myös työkaluratkaisut vaihtelivat omatekoisesta taulukkolaskinbacklogista jo valmiiksi käytössä olleen ketterän työkaluohjelmiston plug-in-kehitykseen. Tutkielmassa esitetään yksityiskohtaiset raportit kaikista kolmesta tapaustutkimuksesta.

Ketteristä työkaluista ja tapaustutkimuksesta kerätyistä tiedoista päätellään, että esimerkiksi backlogin hallintaan ja käyttäjien motivaatioon liittyy ketteryyden pilottivaiheessa ongelmia, jotka voidaan selvittää eri tavoin riippuen siitä minkälainen tiimi on kyseessä. Lisäksi tutkielman lopuksi esitetään viisi periaatetta, jotka antavat yleisiä suuntaviivoja työkaluvalintaan ja -käyttöön ketterässä sulautetussa kehityksessä.

Asiasanat: ketterä kehitys, sulautetut järjestelmät, projektinhallinta, backlog

Acknowledgements

This thesis was conducted as a part of the AgiES (Agile and Lean Product Development for Embedded ICT Systems) project in Technology Research Center of University of Turku. The project was carried out in collaboration with Finnish Institute of Occupational Health and the industry partners BA Group, FiSMA, Lindorff Finland, LM Ericsson, Neoxen Systems, Nextfour Group and Nordic ID. The project was mainly funded by Tekes - the Finnish Funding Agency for Technology and Innovation.

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin OriginalityCheck service.

Turun yliopiston laatujärjestelmän mukaisesti tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -järjestelmällä.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

The purpose of this thesis is to map out what kind of agile tools exist and what the special needs are when the tools are put into use in agile embedded systems development.

Agile development methods are becoming increasingly popular in the software development projects [1]. While the Agile Manifesto emphasizes individuals and interactions over processes and tools, the tools still have a key role in agile development [2][3]. Many teams adopting agile methods may already have a tool for managing their work, but it may not be that compatible with agile practices. Also, using simple tools, such as physical wall and paper to manage backlogs, may not satisfy strict managerial needs for reporting in a company.

Teams and organizations can support their agile practices using an agile tool. There are a vast variety of agile tools to choose from. Picking up a suitable tool may be very problematic: On the one hand, different tools offer different kinds of options implemented in different ways, and on the other hand, often the exact tool needs are not clear even to the company itself. The needs may be very different between the management and the team itself and not equally supported by all tools. [3]

While agile practices are mostly put in use in software development projects, there

are also reported agile projects in the field of embedded systems development. While the agile practices in embedded development are mostly used on the software side of development process, some experiments can be found where the hardware is also developed in an agile way. [4][5][6] To gain more knowledge about agile and lean methods in embedded systems development, the subject was researched in the Technology Research Center unit of the University of Turku in a two-year project called AgiES (Agile and Lean Product Development Methods for Embedded ICT Systems). This thesis was written while working for the AgiES project and utilizes the tool related experiences gathered from the case companies.

The characteristics of embedded systems development bring many nuances to the problem domain of agile tools. Many of these problems root from the fact that agile practices are designed for software development. When adopting agile in embedded development, slight changes and interpretations may have to be made to the original practices in order to cope with the different characteristics of software and hardware development. When it comes to agile tools, this means that a tool must be flexible enough to deal with hardware–software co-design and other embedded system constraints. Many tools may not have enough out of the box features to support the characteristics of agile embedded systems development process, in which case heavy customization may have to be done.

This Master's Thesis is written in order to address these three research questions:

- **RQ1:** What kinds of tools exist that support agile development and how do they do it?
- **RQ2:** What special characteristics are there when tools defined in RQ1 are used in agile embedded systems development?
- **RQ3:** How should someone select a tool that supports agile embedded systems development?

To answer RQ1, different kinds of sources are used to map out common agile tool solutions. There is also discussion about how different tools are designed to support dif-

ferent types of work. In addition to the knowledge of the challenges in agile embedded systems development, RQ2 is addressed with case studies in three AgiES partner companies: During the adoption of agile practices as part of the AgiES project, each company also made different types of changes and improvements to the work management tool solutions originally in use. These experiences are then also used as a basis to answer RQ3.

The structure of the thesis is as follows: In this introduction chapter, the basics of agile development, embedded systems development and the problem domain of embedded agility are presented briefly. Chapter 2 is about different agile tools, discussing their categorization and usage. Chapter 3 discusses the agile tools in the context of embedded systems development and provides an introduction for the chapters 4–6 where the case studies are covered. Finally, the research questions are answered in Chapter 7 followed by Chapter 8 with the summary of the thesis.

1.2 Agile Development

There are many definitions for agile development and agility [7], but often *agile software development* is defined as a group of different principles, practices and methodologies that emphasize timeboxed *iterative and incremental development* (*IID*). Agile methodologies also highlight adaptive planning, evolutionary delivery and face-to-face communication with small co-located teams. [8] Although agile development is nowadays strongly associated with software development, it has roots in multiple industries. For instance, IID was used in the X-15 hypersonic jet project in the 1950s. [1]

Iterative development is a form of development where something (e.g. software) is built by dividing the lifecycle of the development process into many similar sequences back to back in time. In modern iterative methods, these iterations typically last between one and six weeks. Thus, rather than having big and long separate design phases back to

back with each other (usually referred to as the *waterfall model*), many of these activities like requirements analysis, design, programming and testing are now done repeatedly in every iteration. [8]

When building something iteratively, usually the system also grows incrementally as new features are added, which is known as *incremental development*. Being the combination of these two philosophies, iterative and incremental development is the basis of all agile methods. [8] Often agile development is seen as a revolutionary substitute for traditional waterfall-style development processes, although iterative and incremental development methods themselves are a lot older inventions. [1][8]

In this thesis, only the basic concepts of agile development are covered: the background of the Agile Manifesto and short introductions to the key agile methods.

1.2.1 Agile Values and Principles

The concept of agile development was aggregated in ”Manifesto for Agile Software Development” (usually referred to as the Agile Manifesto) written in February, 2001. The manifesto was signed by 17 software engineers and supporters of several already existent lightweight methodologies. They also founded the Agile Alliance group in the process. It mainly consists of a list of values and principles supporting iterative and lightweight development, as well as background and history of the manifesto. [2][8]

The four values of the Agile Manifesto are: [2]

1. **Individuals and interactions** over processes and tools.
2. **Working software** over comprehensive documentation.
3. **Customer collaboration** over contract negotiation.
4. **Responding to change** over following a plan.

It is also noted that while the items on the right have value too, the items on the left are valued more. [2]

Besides the values, the Manifesto also declares 12 principles behind the values. The principles describe a more detailed meaning of agility behind the values. They are as follows: [2]

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Each of these principles sheds light on one or more values behind them. For instance, the principle 2 welcoming changing requirements in the development process directly reflects values 3 and 4.

1.2.2 About Agile Methods

The Agile Manifesto mainly declares a philosophy and a set of goals to achieve agility, a concept defined by the signatories of the Manifesto. In order to organize daily development work there also exist concrete methods with sets of practices to achieve agility, usually referred to as agile methods [8] or methodologies [3]. It should be noted that many of these methods already existed before the Manifesto was written: the signatories themselves were representatives of many lightweight methods such as Extreme Programming (XP), Scrum, Dynamic Systems Development Method (DSDM) and Feature-Driven Development (FDD), among many others. [2]

Different agile methods provide different approaches to achieve agility: while Scrum provides a framework to manage many different kinds of development processes, XP mainly concentrates on a set of practices to be used. [8] Scrum and XP are the two most common agile methods [8][9] and therefore mainly these two methods are discussed in this thesis. Also a brief section about Lean Software Development and Kanban is included. It should be noted that each of these methodologies have many aspects and specific details to them that cannot be fully covered in this short introduction.

1.2.3 Scrum

Scrum is a holistic IID project management framework that puts emphasis especially on self-directed and self-organizing teams. [8] The history of Scrum roots back to an article ”The New New Product Development Game” released in 1986, in which a holistic set of best practices used in Japanese and American companies was described. [10] Then, during the 1990s, Jeff Sutherland extended and popularized the method with Ken Schwaber in its current form and name. [8]

Scrum consists of different events, roles, artifacts and rules of one or more Scrum teams [11]. The development is done in timeboxed iterations called *sprints* that last 30 calendar-days, although shorter sprints are also common. Timeboxing means that the

tasks and the length of a sprint are fixed after the sprint has started. [8][11] Each sprint starts with a *Sprint Planning*, where work is chosen for the sprint, and ends with a *Sprint Review*, where a demo is presented to external stakeholders. Other important meetings are the *Daily Scrum*, where each team member answers a set of regular questions, and *Sprint Retrospective*, where the team inspects itself and makes plans to adjust its behavior in the future. [11]

The key artifacts in Scrum are the *Product Backlog*, a prioritized list of product requirements (usually referred to as *backlog items*), and the *Sprint Backlog*, a subset of the first one containing estimated items chosen to be done during the current sprint. [11] From the viewpoint of agile tools, the backlog management is one of the most important aspects as the concept of backlogs and work item management in general is present in many different development processes, not just in Scrum.

The Product Backlog is managed by the *Product Owner*, a key person responsible for maximizing the value of the product. The Sprint Backlog is maintained by the self-organizing development team itself. Besides these two roles, also a person called the *Scrum Master* exists whose job is to make sure that Scrum theory, practices and rules are both understood and performed. [11]

1.2.4 Extreme Programming

The history of *Extreme Programming* (*XP*) roots back to 1996 when Kent Beck joined the Chrysler C3 project. During that project the XP practices were formed with some help from Ron Jeffries. [1] XP became a lightweight IID methodology intended to be used in small and medium-sized teams facing constantly and rapidly changing requirements. [12]

XP is built around these four values: [8][12]

1. Communication
2. Simplicity
3. Feedback
4. Courage

Communication is emphasized because many problems in projects are caused by team members not talking to each other. Simplicity is on the list since XP states that it is better to do something simple and change it in the future rather than to do something complicated that might not be used anyway. Feedback is needed to gather concrete information about the state of the system and to work together with communication and simplicity. Finally, courage adds an open-minded attitude on top of the other values. [12]

XP presents 12 practices to support the four values: [8][12]

1. The planning game
2. Small, frequent releases
3. System metaphors
4. Simple design
5. Testing
6. Frequent refactoring
7. Pair programming
8. Team code ownership
9. Continuous integration
10. Sustainable pace
11. Whole team together
12. Coding standards

Each of these practices are older than XP itself, and the key rationale behind XP is to wrap them all up in one clear methodology and to encourage to use all of them together. Besides these practices, there are important artifacts that XP introduces: *story cards* presenting *user stories* (clear description of a feature or fix from the viewpoint of a user) and *task cards* presenting *tasks* (often subitems of user stories). While these cards are constantly used and maintained during daily work, they are also used in planning games. The first cards are used in the planning game held every three weeks, while the latter cards are used in the iteration planning game to estimate the work for iteration. [12]

Sometimes Scrum and XP are used together. This is possible since most Scrum practices are compatible with XP. In fact, Beck himself encourages daily stand-up meetings to be held, which are direct derivatives of Daily Scrums. [8][12] The Sprint Review practice is well in line with XP's feedback and communication values, and the Backlogs are lightweight and simple enough tracking tools to be compatible with the simplicity value. The only contradicting aspect is the difference in the iteration lengths: XP encourages to

keep iterations very short (even one-week), while in Scrum the length is usually 30 days. [8] However, when applying agile practices in real life scenarios, teams may use custom iteration lengths, so this is not considered as a problem. In addition, the concept of user stories and tasks in XP can be combined with the concept of Scrum backlogs to provide flexible ways to manage work items of a project. This is a practice that many agile tools support as well.

1.2.5 Lean and Kanban

Lean Software Development is a concept that is closely related to agile software development, and nowadays usually counted in as one of the methodologies. Its roots are in Toyota Production System (later named Lean Production) and especially in its subcategory Toyota Product Development System. In the early 2000s Mary and Tom Poppendieck applied principles of the Toyota Product Development System to the software development domain. Lean software development consists of seven principles: [13]

1. Eliminate waste
2. Build quality in
3. Create knowledge
4. Defer commitment
5. Deliver fast
6. Respect people
7. Optimize the whole

When discussing Lean, often *Kanban* is mentioned too. Kanban can mean both the Kanban method and *Kanban cards*—the word ”kanban” itself is Japanese and translates into a ”signal card” [13]. Kanban method lays down five principles that largely overlap with the principles of Lean software development: [14]

1. Visualize the workflow
2. Limit Work In Progress
3. Measure and manage flow
4. Make process policies explicit
5. Improve collaboratively (using models and the scientific method)

A key artifact of Kanban is the Kanban board containing different Kanban cards [14] that can flow from one state to another. A Kanban board can facilitate at least the first three principles. When combining aspects of Kanban, XP and Scrum, different and more general task boards can be constructed.

1.3 Embedded Systems Development

An *embedded system* is usually described as a computer system that is specialized for a dedicated task and embedded within a larger system usually consisting of computer hardware, software and some kinds of mechanics. The number of different embedded systems is constantly growing, as nowadays microprocessors can be found anywhere from cell phones to cars and space stations. [4]

Embedded systems development is going through the same kind of evolution that software development has been going through over the last decades: The systems are more and more complex making it more and more difficult to develop these systems in an efficient manner. While agile methods have gained popularity in the software development domain to address these problems, agility is not a well researched subject when it comes to embedded systems development. Even though there are experiences about adopting agile methods both in embedded software development and embedded systems development, there are also some domain-specific issues that need to be taken into consideration when applying agile methodologies to embedded systems development. [4]

1.3.1 Agile Development of Embedded Systems

Embedded systems have some inherent characteristics that make them unique systems to be developed. For instance, they often have power limitations, real-time constraints, monolithic functionality, limited development tools and custom hardware. Maintenance of embedded systems can be far more difficult than software maintenance in situations

where the hardware hidden from the user will exist for a long period of time. [6]

When it comes to agile values, principles and practices, there are four key constraints that make it challenging to develop embedded systems in agile ways: [15]

1. **Need for system level documentation**: Especially in areas were standards and regulations play a major role, simply developing a working product can not be something to be aimed at to the detriment of important documentation.

2. **Hardware–software interdependences**: The development of hardware and software (and often mechanics too) are highly dependent on each other. Coupled with the long development cycles of hardware and mechanics development this makes iterative and incremental development challenging.

3. **Heterogeneous teams with different skillsets**: Traditionally software, hardware and mechanics developers have been very separated in the development process and furthermore each developer likely has their own area of expertise. Thus, developing the system in an agile way is difficult since knowledge transfer between developers is challenging.

4. **Inflexibility due to real-time functionality**: Real-time requirements of embedded systems mean that certain functions need to happen in a predictable time window on the chip. This makes the design flow very challenging, as the smallest features and bug fixes can significantly change the timings of the chip. Thus, modular and readable code can not always be achieved when speed and power consumption of the design matter more.

In [15], these challenges are discussed from the viewpoint of the 12 principles of the Agile Manifesto (see Section 1.2.1). The authors group agile principles into four categories: progress of product development (principles 1, 3 and 7), control of change (principles 2 and 10), people (principles 4–6, 8 and 11) and improvement of agility (principles

9 and 12). Almost all the principles concerning people or improvement of agility are well in line with the challenges of embedded development, while the first two categories are more problematic. To avoid this, authors have modified the incompatible principles to fit better to embedded systems development while preserving the philosophy behind agile manifesto.

The concluded proposal of new agile principles in embedded systems development are (redefined parts are italicized): [15]

1. Our highest priority is to satisfy the customer through early and continuous *demonstrations which lead to the valuable system*.
2. *Defer making restricting design decisions to allow changing requirements*, even late in development. *This way the change can be harnessed* for the customer's competitive advantage.
3. Deliver *demonstrations leading to the working system* frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. *Demonstrations and working system are* the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. *Balance between* simplicity—the art of maximizing the amount of work not done *in a short term—and generality—the art of minimizing the total amount of work to be done in a long term*—is essential.
11. The best architectures, requirements, and designs emerge from *co-operating* and

self-organizing teams.

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

The list of new principles clearly presents the differences between plain software development and embedded systems development. The principles concerning the progress of product development are modified the most and the concept of deliverable had to be presented in a loose way because of the long development cycles of hardware development. Similarly, the principles concerning the control of change have been modified because hardware cannot usually face huge changes in later parts of the development process—at least not in a cost-effective way. Other principles are more universal and therefore mostly left intact except a small addition done for Principle 11 to allow more emphasis on joint system designs and architectures that are necessary in embedded systems. [15]

The principles of the Agile Manifesto are envisioned to be applicable to embedded development with these modifications and as long as suitable practices are used to support them. [15]

Chapter 2

Agile Tools

In this thesis, an *agile tool* is defined as a project and work management tool that helps a team or an organization to support their use of agile practices, most importantly backlog management. This is a very broad definition, but on the other hand it gives a wider perspective on the field of different tools that could possibly be used to assist agile development. The tools form different categories and their features can be classified in different ways. This definition of an agile tool can also be interpreted in a way that it counts in physical wall and paper and tools that are not originally meant to be used in agile development processes. It is also important to notice that while the main focus of this thesis is in software solutions, tangible tools offer many good features that software solutions do not.

A concept closely related to agile tools is *agile project management*. Managing agile projects differs from traditional project management, but even very agile projects in real organizations may need some sort of traditional management to be done as well (e.g. managing the budget). Agile methods in general put emphasis on lightweight management, so the whole concept of agile project management is usually looked from a completely different viewpoint than traditional management. It can be said that traditional project management is a planning-and-control-biased model, while agile project management is an execution-biased model. [16] This is another reason why a more general term is used

in this chapter rather than the term *agile project management tool*: to clearly distinguish agile tools from traditional project management tools.

Another concept closely related to agile project management is *application lifecycle management* (*ALM*). The key goal of ALM is to integrate different workflows together, such as requirements management, architecture, coding, work tracking, and release management. It also aims to cover process automation and process traceability. [17] Many agile tools offer different ALM characteristics, and hence many of these tools could also be called *agile-ALM-tools*. [3] However, considering the strive for simplicity in addition to lightweight methods and tools of agile methods in general, intensive ALM may not be necessary for all projects.

In this chapter, the features and categories of agile tools are discussed with examples of real agile tool products. Then relevant agile tool surveys are reviewed to find out what kind of tool solutions are known to be popular in agile development. The chapter is crucial to gain knowledge about agile tools to provide answers to RQ1. These findings are later summarized in Section 7.1 along with answers to other two research questions.

2.1 Categorization of Agile Tools

Putting tools in different categories can be a tricky job. In this chapter, there are three initial general categories of agile tools discussed:

1. Tangible tools
2. Software-based agile tools
3. Non-agile software tools in an agile context

Different aspects of these tool types are discussed later. However, these categories are on a very high level, and especially since there are many different kinds of software-based agile tools that can be sub-categorized, more precise categorization might still be in place. Different categorization options are discussed here.

It should also be noted that, when looking at the big picture, different software agile tools can be seen as just a subcategory of different project management related software solutions. These other types of tools include traditional project management tools (e.g. Microsoft Project), issue and bug trackers (e.g. Mantis or Bugzilla), wiki systems, spreadsheets and office suites in general, even version control and build systems, among many others. [18][19][20][21] In addition, there are numerous tools that fit into more categories than just one, and many tools can be used outside of the original intended domain.

Putting agile tools into different categories would be easy if there was, for instance, a prominent feature or feature set that would only exist in one type of tools but not in others. For instance, if a categorization was done based on agile methods that different tools claim to support, a problem arises since most tools support the key practices of the two most popular agile methods Scrum and XP [18]. Hence, this does not provide a convenient categorization: most of the tools support several agile methods equally good. This usually applies if the categorization is done based on any other specific feature difference. Hence, the categorization options presented here tend to be more abstract.

2.1.1 Categorization by Feature Emphasis

Features that exist in agile tools tend to fall into a few categories. Whether a tool puts more emphasis on one aspect over another, it certainly falls into different category than another tool that puts emphasis on other aspects.

Here is one way of putting features in different categories:

1. **Daily work support:** Features that help teams to organize their everyday development work. Examples: task boards, collaboration features (or separate collaboration tools).
2. **Agile planning and management support:** Features that help an agile team to organize their agile work process. Examples: backlog management support, burn down charts.

3. **Traditional management support:** Features that originate from traditional project management and that are more aimed at team leaders. Examples: scheduling, detailed reports, forecasts and resource management.

Sure enough, there are features that fall into more categories than just one, but this still gives a useful viewpoint on tools. For example, agile tools that do not have much features from the last two categories are remarkably different (focusing on a small team of agile developers) than tools that have features from all three categories (focusing on a development process as a whole).

2.1.2 Dividing Agile Tools into Light and Heavy Tools

In [3], agile tools (called ”agile-ALM-tools”) are put into four different categories. The first two of these are *full life-cycle management applications* and *targeted planning applications*. The first ones are the all-in-one type tools, such as Rally and VersionOne, providing a large set of ALM features supporting both managers and developers. In contrast, the second category contains tools that are aimed to teams that want an agile tool but do not need a ”comprehensive integrated ALM platform”: For example, the tools in this category may have interactive card boards and other kinds of features to support daily work, and planning and review ceremonies, but nothing more complex. In addition to these two, there is a category called *Agile ALM as a Consulting Component*, which means that the ALM tool is just a part of a bigger consulting deal: the companies providing these have a business plan that is not mainly focused on stand-alone versions of their tool. Tools and companies called ResultSpace, ScrumWorks and Thoughtworks Mingle are mentioned as examples that have this sort of business strategy. Lastly, a category containing *ALM plug-ins* is presented: many non-agile tools have plugins that can bring different agile or agile lifecycle management features to the main product. [3]

The categorization presented in [3] is not entirely consistent: On the article, the categorization is introduced as ”a few examples of the tools”, and being merely headers on an

article, these categories are most likely intended to give different viewpoints on different agile tools and not necessarily model a fully thought out taxonomy. For instance, the third category, agile ALM tools as consulting components, can be seen as a subset of the first category, full lifecycle management tools.

The contrast between the first two categories does, however, provide a useful viewpoint on the scope and broadness differences between tools. To abstract the earlier categorization based on feature emphasis even more, there are two clear extremities: light and heavy agile tools. Their characteristics can be defined roughly as follows:

1. **Light agile tools:** Targeted planning applications and other agile project management tools that offer ways to organize the daily work of agile processes where intensive documentation and history tracking are not considered as top priorities.
2. **Heavy agile tools:** Broader agile tools that have more intensive features for tracking, documenting and estimating than the ones in the first category. Generally speaking, traditional project management and issue tracker tools have influenced the tools of this category a lot more than the tools of the first category.

2.1.3 The Broadness–Methodology Model

Previous categorization was based on an axis that has light agile tools in one end and heavy agile tools in another. However, in this definition light agile tools are quite exclusively ”agile”, meaning they only support agile development and management, while heavy agile tools also have traditional management aspect to them. Because of this, the aspect of agility can be distinguished from the aspect of broadness, and both aspects can be presented as their own axes of a coordinate system. This is exactly what has been done in Figure 2.1.

The diagonal line represents the previous definition of light and heavy agile tools—light tools focus on fewer features than heavy agile tools. The management features of light agile tools are designed to support mainly very agile processes, while heavy agile

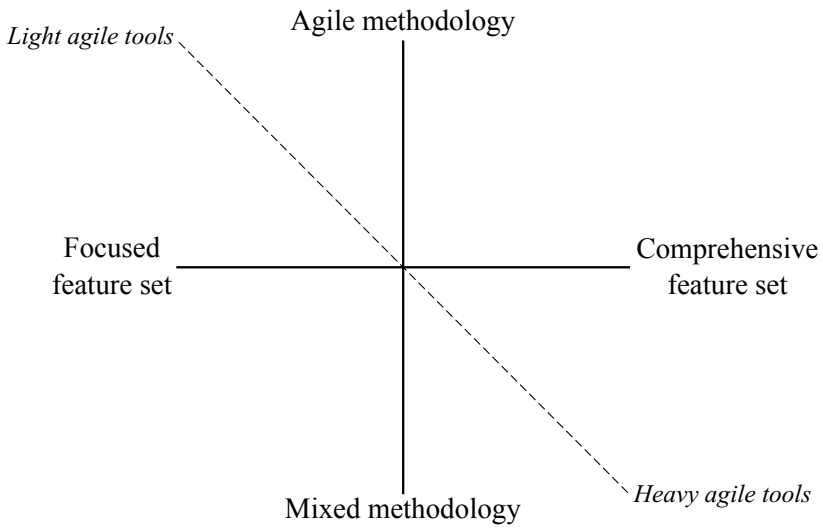


Figure 2.1: The relation between features and methodologies of agile tools

tools have features supporting both agile and traditional management—hence the term ”mixed methodology” is used.

Since individual tools can be located on different parts of Figure 2.1, this gives a graphical way to describe the type of a tool. In general, all sorts of tangible tools would be located at the upper-left corner since they are very agile and usually serve only a few clear purposes (and therefore have a focused feature set). On the other hand, software-based tools can be located anywhere depending on what tool is in question. *Agile* software-based tools are quite accurately somewhere on the line between ”light agile tools” and ”heavy agile tools”. The figure can be interpreted in a way that it also covers some non-agile tools, such as bug trackers. These tools have support for ”mixed methodologies” in a way that they can be possibly used in agile contexts. Then it depends on the individual tool whether it is located nearer to the lower-left or the lower-right corner.

Tangible tools, software-based agile tools and non-agile software tools in an agile context are discussed in their own sections next.

2.2 Tangible Agile Tools

As it was mentioned in the introduction, tools still have a place in agile development despite the Manifesto valuing individuals and interactions over processes and tools. For instance, the Scrum method has a lot of characteristics, such as backlogs, that require at least some kinds of tools to be used. This also applies to XP, which itself introduces and encourages two tangible tools to be used: task and story cards. Physical wall and paper or other tangible tools are often simplest tools to deploy, and they offer many other advantages over software solutions. Their simplicity also fits well together with the agile principles.

Task boards and Kanban boards were mentioned in the previous chapter in addition to plain task and story cards of XP. A lot of different variations can be possibly made to support different agile methods, such as XP, Scrum and their variations. However, in [22] a typical task board is defined, and it is used here to give a common example of a task board that is often replicated in software solutions. This is presented in Figure 2.2. This kind of a task board is used to manage sprint backlogs in Scrum and other methods. In the example there are four vertical columns. The first column is for sprint backlog items (in the example these are always assumed to be stories) and the second one is a list for ”To Do” items or tasks that are needed to be done to complete each backlog item on the left. Other columns are usually labeled ”In Progress” and ”Complete”. When a task is begun to work on, the corresponding task card or sticky note is moved from the ”To Do” list to the ”In Progress” list. When the task is considered to be done, the card is moved to the ”Complete” column. In Scrum, this sort of a task board is basically a way to present and manage Sprint Backlogs: When all the tasks next to a certain user story are moved to this column, the story is considered to be done and it is ready for the Sprint Review and for the Product Owner or customer to be approved. [22]

In the XP method, story cards describe a user story, which is a short description of a feature from the viewpoint of the user, or alternatively a fix or a nonfunctional requirement

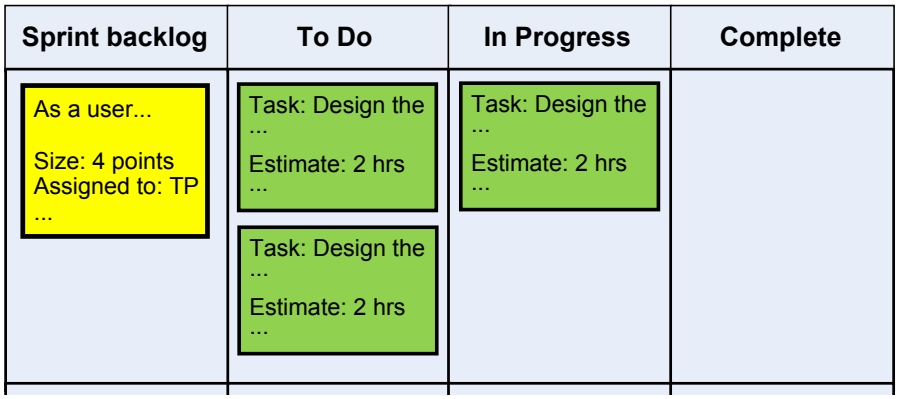


Figure 2.2: A typical task board (adapted from [22])

that is needed to satisfy the user. [8] A card can be either very minimal containing only a user story and an estimated duration (may vary between one day and three weeks) [8] or more diverse containing different additional information, such as an ID, the type of activity, a risk estimate and so on [12]. Tasks corresponding to stories have an effort estimate in ideal engineering hours (usually in the one to two-day range) [8]. They can be written on the story card itself [12] or written down on a separate list or separate cards. [8] If a task board is used, naturally the separate card approach is mandatory.

Because of the various possibilities of pencil and paper, a limited discussion of different tangible agile tools is presented here. The Agile Manifesto and agile methods courage people to interact and explore, and therefore sticky notes and other tangible tools can be helpful in many different situations during an agile development process. In addition to their advantages, physical tools obviously have limitations as well.

When it comes to distributed teams, there is an obvious restriction about physical tools: multiple teams cannot have an access to physical boards or cards. Hence the software solutions are generally used more by distributed teams than by co-located teams [18]. However, the software solutions are not just for distributed teams as co-located teams often need features that physical tools can not offer, such as various aspects of reporting and project tracking. Many of these needs are not necessarily from the core of the Agile Manifesto, but still relevant in real life organizations. A good software solution

may however be relatively easy to use, and, for example, a virtual task board can do many things better or more effortlessly than a physical one, such as automatic numbering and providing infinite amount of ”virtual paper”.

On the other hand, in co-located teams people working physically close to each other, physical tools may provide ease of access superior to software-based tools. Software tools may require laborious login procedures, though automatic login is usually possible. Another feature that tangible tools offer is the social aspect which is often restricted in software solutions. As from the Manifesto point of view individuals and interactions should be focused on, this should be considered as an advantage.

The deployment difficulties of software solutions are important to consider: it may take a lot of effort to set up the software system, input all the data, and familiarize the team with the different aspects of the tool. If the tool is not a free solution, someone also has to deal with licence payments, or even if it is, even more configuration might be needed, especially with on-site installations. Physical wall and paper can be put in use very quickly and do not cost a significant amount of money. While some familiarization might also be needed with a physical story or task wall to create a common usage practices, it can be far more straightforward.

These advantages and disadvantages are also discussed in the experience report [22], where a team switched from a physical task board to an electronic tool, and then back again. The final conclusions are well in line with the points mentioned here in the previous paragraphs. The pros and cons of both solutions are summed up in Table 2.1.

In conclusion, tangible tools have their own advantages and disadvantages. When picking a tool, physical tools should be considered as one of the options and not be ignored. When deciding between physical and software solutions, the actual needs for the software features covered in this thesis should be mapped out before dropping the physical tools off the list. Finally, while the broadness-methodology grid in Section 2.1.3 was mainly conceived with software tools in mind, tangible tools fit in it without any problems

Electronic task board	**Physical task board**
+ Distributed team support	+ Inexpensive
+ Integration with existing software tools	+ Provides a clear focus point
+ History tracking	+ Always present
− The focus not on interactions anymore	+ Minimal training required
− Invisibility of electronic information	+ Physical interaction improves learning
− Requires training	− Limited to line of sight
	− No history tracking

Table 2.1: Electronic and physical task board — pros and cons (summarized from [22])

in the upper-left corner.

2.3 Software-Based Agile Tools

If tangible tools do not offer a solution for an agile team, a software-based tool is the next logical candidate to be considered. Because the amount of software solutions is huge, a smaller set of popular agile management tools have been chosen to be discussed in this section. These tools are then discussed from the viewpoint of the earlier categorization models – not only to better describe their characteristics but also to verify the accuracy of the models themselves with real tool examples.

While Section 2.5 goes through all the relevant previous surveys about agile tools on a more detailed level, a particular agile tool survey from 2011 [18] was used as a basis to build up a list of tools. The original goal was to pick a subset of popular agile tools based on a few criteria and to test out these tools. In order to be chosen, a tool had to have some sort of a trial or free version available to be able to be tested in the first place. Obviously, it also had to have features to support agile process to qualify as an agile tool. This way, the initial list of tools consisted of six agile tools that had their usage percentages visible in the

survey: Rally [23], Mingle [24], ScrumWorks [25], VersionOne [26], Team Foundation Server [27] and Assembla [28].

Shortly after the list was formed some subsequent modifications were done to it. Since ScrumWorks never processed the original trial request, it was dropped off the list. Because there were only heavy or medium heavy tools listed, another tool was picked from the ”Others” category of the survey specifically to include a lighter tool on the list: tinyPM [29]. Also, in addition to the six tools chosen from [18], three other tools were also included in the final list—first, to have more variability, and secondly, (since the survey was conducted in 2011) to perhaps be able to give a more current listing of agile tools. The first two of them were picked by coincidence: First, a popular lightweight task list application Trello [30] was added since it was in a daily use in our project as a to-do list application. Secondly, Agilo for Trac [31] was added because it was used by one of the case companies (Nextfour; see the case study in Chapter 6) and since it became a logical addition to the original list. The last tool, ScrumWise [32], was picked based on overheard experiences and search engine results and because it brought a lightweight tool on the list specifically designed the Scrum method in mind.

The final list of tools is presented in Table 2.2. The list being relatively versatile should represent a realistic sample set of agile tools used in different sized companies and teams. However, there are two key similarities between these tools. First, all of them are web applications. That is, while many vendors also offer on-site licenses, the tools are still run in a web browser. Secondly, nearly all of the tools also have some sort of a ”freemium” model when it comes to licensing: there are extra features that cost more money. The only exception to this is ScrumWise, which (at the moment) only has a monthly fee per a licensed user and no alternative licenses or extra features currently available.

Putting these tools into discrete categories (e.g. ”light tool”, ”medium heavy tool” and ”heavy tool”) based on their broadness turned out to be a difficult task to do, as many tools would fall into more than one categories when trying to keep the amount of cate-

Name	**Version(s) or edition(s) tested**	**Rationale behind selection**
Agilo for Trac [31]	1.3.11-PRO	used in a case company
Assembla [28]	Free	1 % popularity in [18]
Mingle [24]	13.4	3 % popularity in [18]
Rally [23]	Unlimited (Demo) / Community Edition	5 % popularity in [18]
ScrumWise [32]	Demo	light Scrum tool
Team Foundation Server [27]	Visual Studio Online (formerly Team Foundation Service)	2 % popularity in [18]
tinyPM [29]	3.0	$<$ 1 % popularity in [18]
Trello [30]	(Only free version available)	used by the project team
VersionOne [26]	Enterprise (free trial)	2 % popularity in [18]

Table 2.2: The agile tools observed

gories low. As stated in Section 2.1.2, light and heavy tools are the extremities of one axis, so rather than trying to put these tools into separate categories, it is more appropriate to discuss the lightness/heaviness relatively to each other. It is fairly accurate to describe one tool being ”heavier” than another based on the amount of features it has from different categories (daily work, agile planning and management, traditional management), as noted in Section 2.1.1. The tools are introduced here in an approximate order, starting from the ”lightest” tools and ending with the ”heaviest” tools.

The short testing mainly consisted of backlog and task management: a group of backlog items were created along with a few tasks, inserting them into iterations and playing around with the features offered. The tool presentations are kept fairly short and therefore provide only an overview on the tools. There is no point to cover all the details of every tool as, in the end, every team has its own set of preferences to be taken into consideration.

2.3.1 Overview of the Tools

Trello

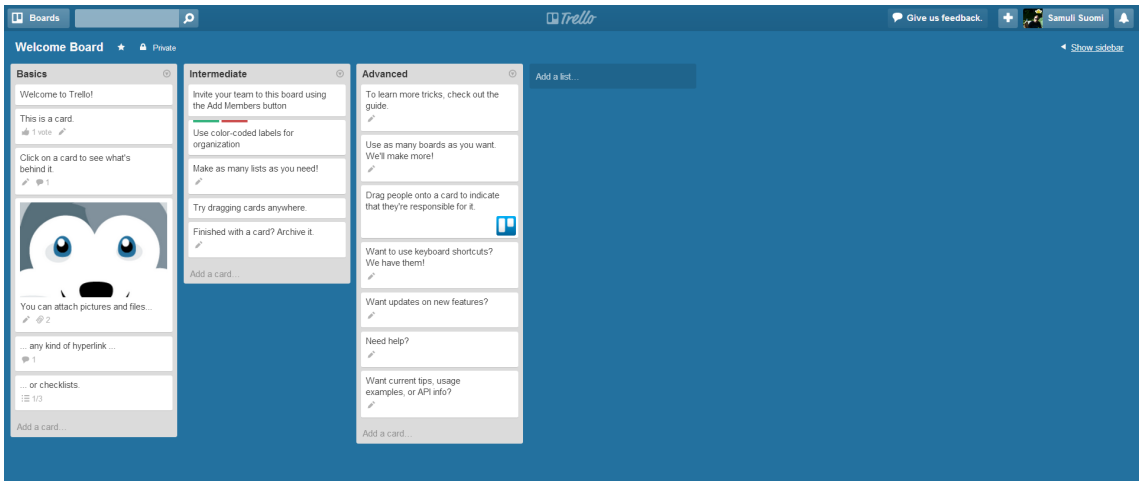


Figure 2.3: A screenshot of the ”welcome board” of Trello

The lightest tool on the list is clearly Trello: in fact, it is not even marketed as being

an ”agile tool”, however there are guides and reported experiences about using Trello for Scrum. [33][34] It is completely based around ”boards” containing ”cards” that can be assigned to different Trello users, and there is not much more other functionality. In Trello, cards are very simple and not restricted to any known definition of tasks or a user stories, so it is up to the user to decide what kind of scope is used on cards. Each card can also include checklists to provide a quick way to list tasks.

Trello’s features are all about daily work support: it is clearly designed to provide a simple task board application for self-organizing teams. Agile planning and management support are restricted to the user’s imagination about how to use the simple features of the tool to organize, say, a sprint planning or review. Trello does not provide estimating or hour tracking capabilities, but there is a browser plug-in available. [33] There are no particular traditional management features available.

ScrumWise

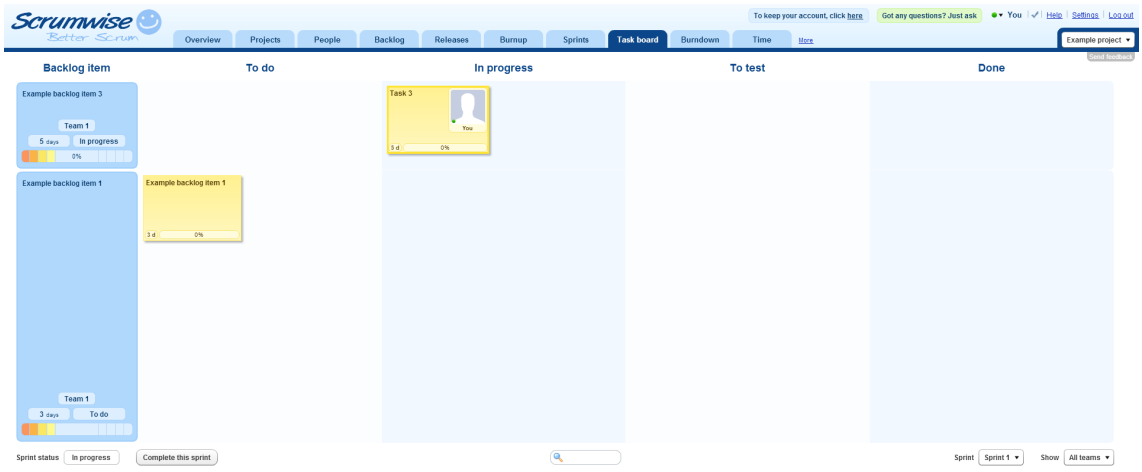


Figure 2.4: A screenshot of the task board in ScrumWise

The next lightest tool is ScrumWise. It is marketed as ”the most intuitive Scrum tool”, and the user interface is certainly kept very simple with clear tabs and no excessive menus.

ScrumWise runs on Adobe Flash, which is a big drawback if compatibility is wanted on many different platforms, especially mobile.

ScrumWise has features supporting both daily work and agile planning and management. ScrumWise seems to offer all the features needed to run a Scrum project. Backlog items can be input, prioritized and customized, sprints can be planned and all the planned backlog items and tasks can be observed and manipulated on the task board. There is support for multiple projects and users, and it also offers agile estimating capabilities and burn down charts. There is also an option to enable a more detailed time tracking functionality. In short, every feature of ScrumWise seems to have been built the Scrum method in mind. There are limited ways to look back at the history of project items.

tinyPM

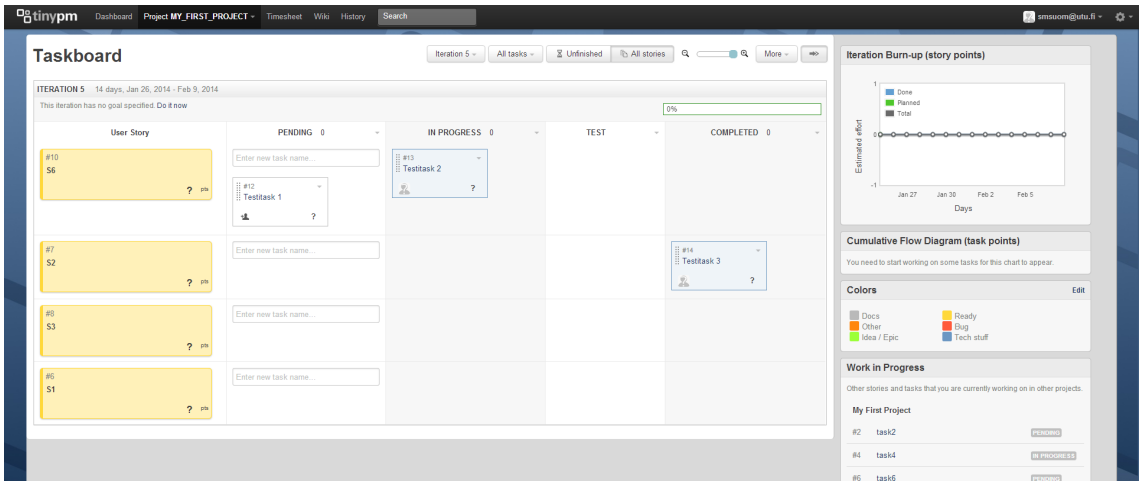


Figure 2.5: A screenshot of the task board in tinyPM

tinyPM is a lightweight agile project management tool such as ScrumWise. It provides a lightweight agile project management features while not being as Scrum-centric as ScrumWise. For instance, a more neutral term ”iteration” is used instead of ”sprint”, and the simple user interface does not specify specific places for Scrum ceremonies to be

centered.

The new 3.0 version of the tool is somewhat different than the previous 2.x versions, and some of the features, such as version control integrations, are not present in the new version. There is one feature that the previous ones do not offer: wiki. This can be used as a place to quickly input key information about the project or the product. Other than that, tinyPM also focuses features that support daily work, agile planning and agile management. Traditional management features are limited.

Assembla

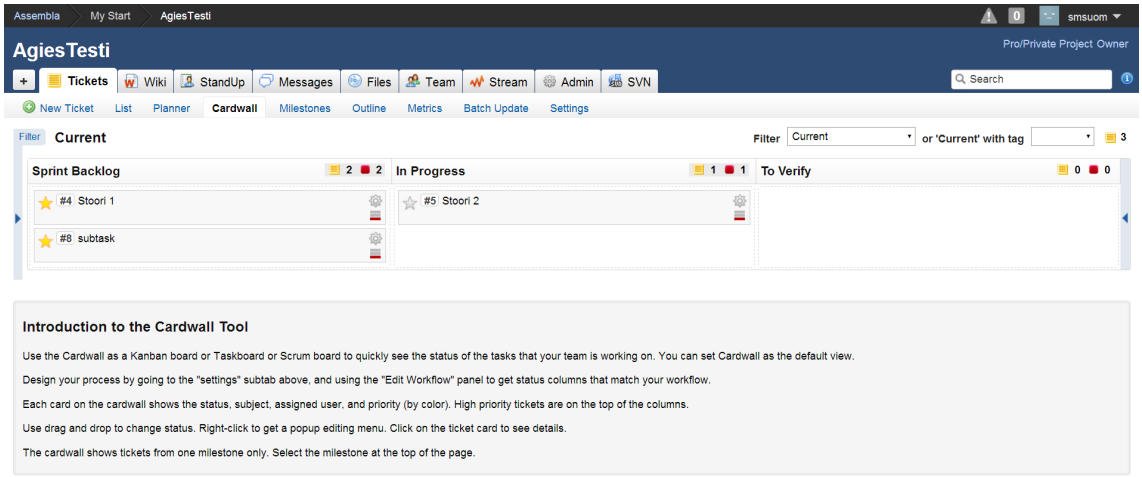


Figure 2.6: A screenshot of the ”cardwall” in Assembla

As a first small step to the heavier side of agile tools, Assembla offers more features than previous ones. The user interface of the tool is also slightly more filled with buttons and menus when compared to the previous ones, while still not utterly cluttered. Assembla also has a slightly different ticket design philosophy: the two main categories of tickets are stories and *epics* (epics are very big items that can contain multiple stories). There can also be tickets with no category set. Unlike earlier, the stories are actually the cards that are dragged on the task board (”cardwall”) during the iteration, not the actual tasks. The

task board has only three default columns, ”sprint backlog”, ”in progress” and ”to verify”. The stories themselves can contain subtasks that have their states to be set and changed, but not by dragging visually on any board. Tickets can be also labelled with custom tags and other information.

There is also support for milestones. Milestones are slightly limited though with no option to have iterations within them: they actually act as other iterations with only ending dates set and the current iteration is just a milestone called ”Current”. There is a separated task board for iteration and milestone planning. The tool also offers different metrics on the project. While some of the metrics are said to be on beta state, there is a solid set of metrics to do sufficient agile management from tracking velocity to investigating issues with progression of tickets. Assembla also has a lot of other functionality, such as support for messages, file uploads and an option to host SVN repositories within the system.

Mingle

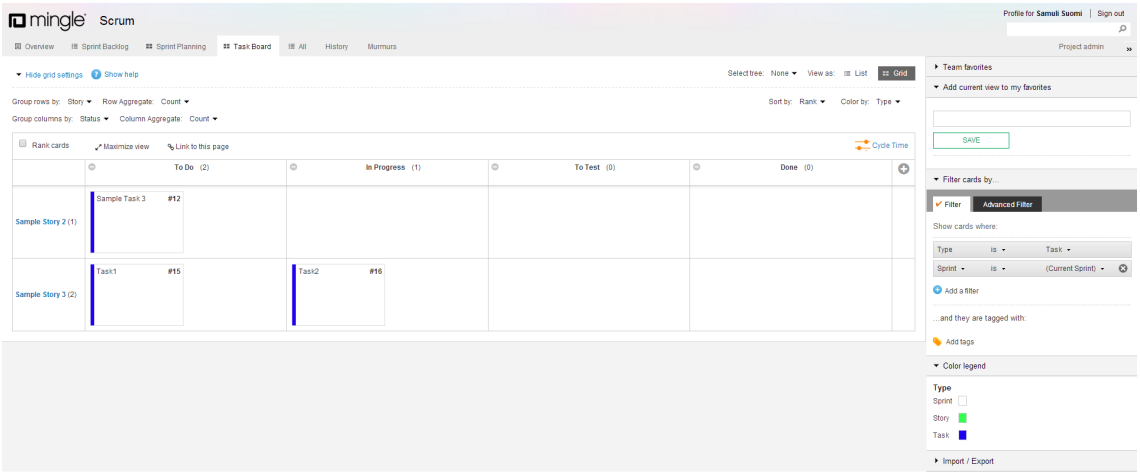


Figure 2.7: A screenshot of the Scrum task board in Mingle

Similar in heaviness with Assembla, Mingle is targeted both towards agile teams (the basic edition of Mingle) and agile organizations (Mingle Plus). In the basic edition, there

are project templates, workspace for team projects, issue tracking, card walls, basic reporting and customizable workflows. Mingle Plus also supports multiple teams each with different workflows and more advanced forecasting. The Plus edition was not experimented with.

The usability of Mingle differs very much based on the project template used. When creating a project in Mingle, there are a few template options to pick from and more can be downloaded from the community site. The original templates are for Scrum, Kanban and a general agile process. When the Scrum template is used, there is much of the same ticket functionality as with previous tools. There is a similar Sprint planning view as in Assembla and tasks can be seen only in the specific task board view. This makes assigning tasks to a specific story a little slower than in the previous tools. Also, prioritizing backlog items is not possible by dragging and dropping. Overall, the basic version of Mingle does not seem to have much other functionality than Assembla. There are both daily work and agile management functions, but reporting needs manual work in terms of custom charts on the overview wiki page. There is no particular functionality designed just for traditional management.

Agilo for Trac

Agilo for Trac is a free plugin for a free popular bug tracker Trac. As a bug tracker, Trac is heavily centered around ticket functionality which results in Agilo for Trac also being a ticket-centered tool. There are a few ways to install Agilo, the easiest of which is the virtual machine option that allows excessive configuration to be skipped. It is still not as easy to setup as the rest of the tools: for example, creating new projects and doing various other modifications to settings and plug-ins need to be done on the command line interface of the server.

Agilo offers no task board functionality, and different ticket types need to be managed

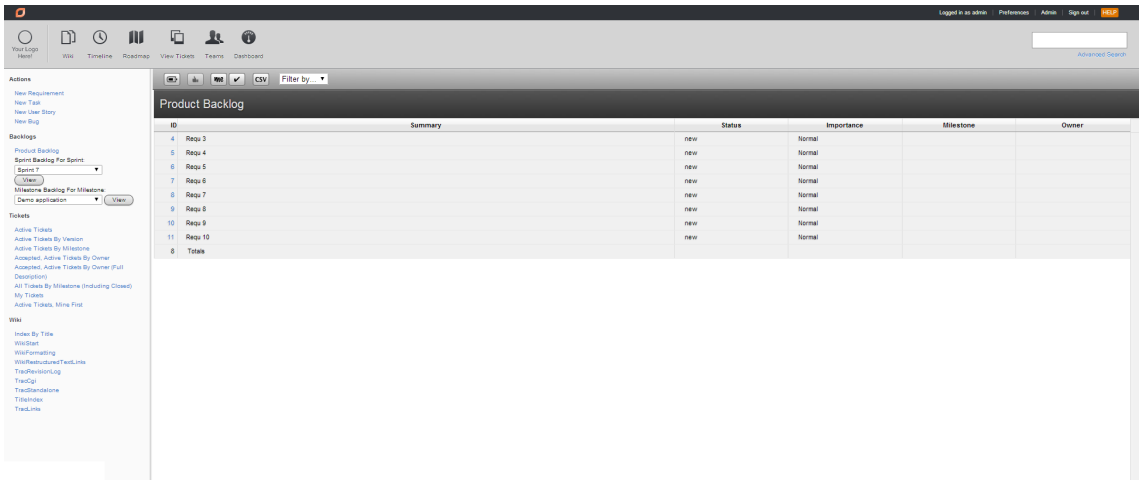


Figure 2.8: A screenshot of the product backlog page in Agilo for Trac

in lists. However, being internally a Trac bug tracker, Agilo offers many ways to customize the views to the work item data. Nearly everything ticket-related is customizable, from fields and their types to their hierarchy. Reporting can be achieved with different ticket fields and wiki plug-ins compatible with Trac, which again requires manual work to be properly configured. Agilo for Trac can be used for traditional management, but, again, it needs a lot of configuring. In Chapter 6, more light is shed on this tool since the case study is heavily focused on the customizations and other improvements done to Agilo for Trac in the case company.

Team Foundation Server

The last three tools are among the heaviest on the list, while Microsoft Team Foundation Server (TFS) differs the most from Rally and VersionOne. As any clear version of TFS was not mentioned in [18] and there were a few different Microsoft products labeled under the name TFS, the most easy to deploy equivalent was used: Visual Studio Online (formerly known as Team Foundation Service). It offers a full ALM platform with source control repositories (TFS or Git), cloud build service and tools for project planning and

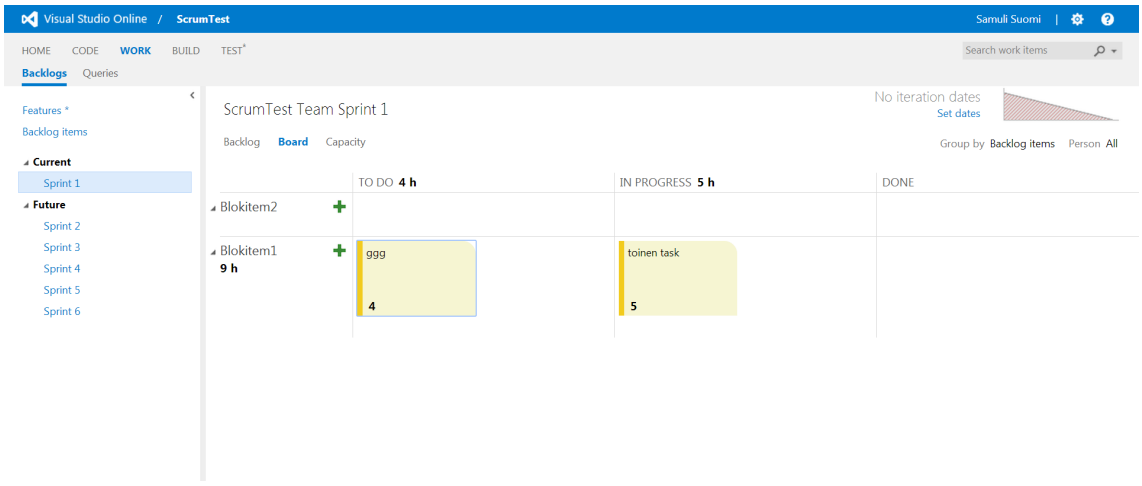


Figure 2.9: A screenshot of the task board in Visual Studio Online

tracking. The last category of features have both agile and traditional management capabilities. There are three process templates that new projects to be initialized on top of, including one for Scrum.

After finding all the necessary buttons on the comprehensive user interface, managing backlogs and tasks seem not to lack any important functionality. The philosophy of the tool is to centralize everything in one place from planning to code hosting, testing and building. The on-premises option offers a lot more customization, but the cloud platform is much more straightforward to set up. There are both agile and traditional planning and management characteristics in this tool, as well as daily work support.

Rally

At least during the recent years, Rally has been one of the most popular commercial agile tools (along with VersionOne) [18][19]. There are three editions, of which the free community edition was tested. The other two editions are Enterprise Edition (focus on multiple teams, more features such as Kanban support and shared backlogs between projects) and Unlimited Edition (focus on an organization level, more features supporting progress

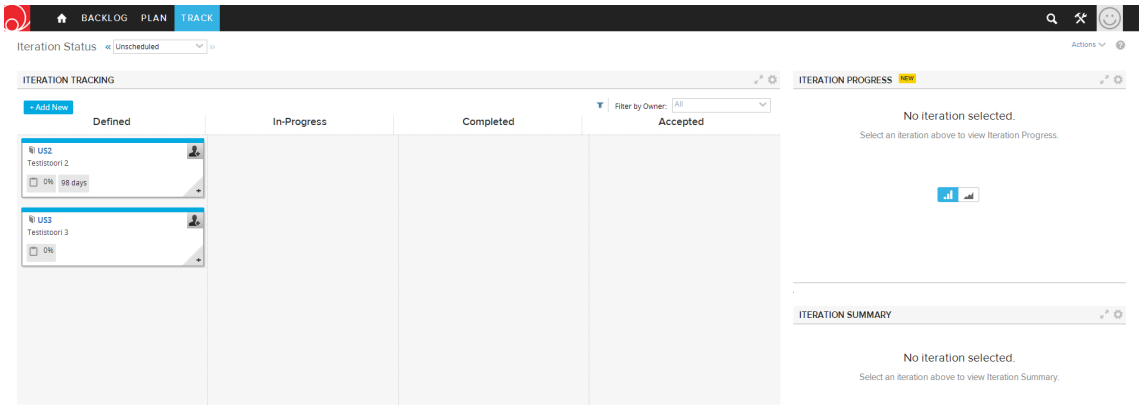


Figure 2.10: A screenshot of the iteration tracking page in Rally

and quality control). Especially the feature sets of Enterprise and Unlimited editions make Rally a heavy agile tool, though the community edition is more limited only to the agile planning and management.

After a little familiarization, creating a backlog and its stories follows a very similar pattern as the rest of the tools. One noticeable difference is that while tasks under stories can be easily created, quick editing of their states is not possible on provided task boards (one for sprint planning, other for story tracking—somewhat similarly to Mingle). However, custom pages can be added together with basic views, and various ”apps” can be inserted to them. One of these apps is a traditional task board to offer task moving functionality. Apps are designed in a way that it is possible to create a customized landing page with apps from Kanban boards to different reports and graphs depending the needs of the team. There are also customization options on the apps: for instance, unwanted fields can be filtered out from the reports. The free edition clearly has a lot of features supporting agile planning and management. Traditional management features seem to have more emphasis put on in the Enterprise and Unlimited editions.

VersionOne

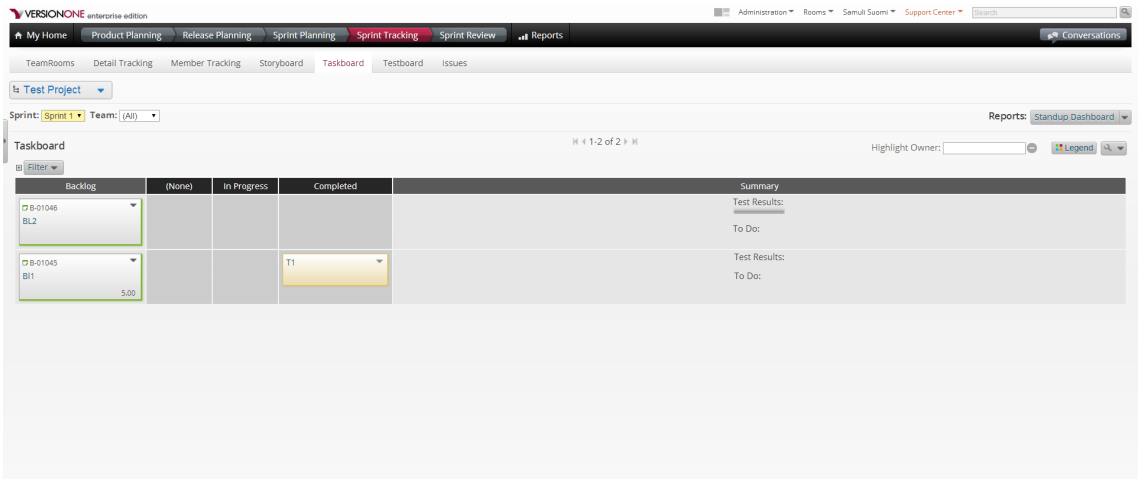


Figure 2.11: A screenshot of the task board in VersionOne

As said earlier, VersionOne is also one of the more popular agile management tools. There are also four different editions of the program, Free (maximum of 10 users), Catalyst (maximum 20 users), Enterprise and Ultimate. As Enterprise was marketed as the ”most popular” one, it was taken a look at. There are a ton of features from multiple teams and projects to reporting and forecasting. It also features a TeamRoom feature for distributed development.

VersionOne is very tab heavy with different pages for different Scrum-driven ceremonies from product planning to sprint review. The user interface has a lot of components, so familiarization is clearly needed to fully take advantage of the features. Backlog creation, task management and other basic agile planning does not seem to lack any important features. There are also customization possibilities and traditional management features, such as forecasting.

2.3.2 Summary with the Broadness–Methodology Model

After short testing it is clear that all of the tools provided capabilities to help managing agile projects and work items. However, there are many small differences that cannot be

covered in this short introduction: Since there are different ways to build features such as task boards, it is up to the requirements or personal preferences of a team to judge the best solution for a particular situation.

When it comes to the earlier model about broadness and agility defined in Section 2.1.3, the tools can be approximately inserted to the model: this has been done in Figure 2.12.

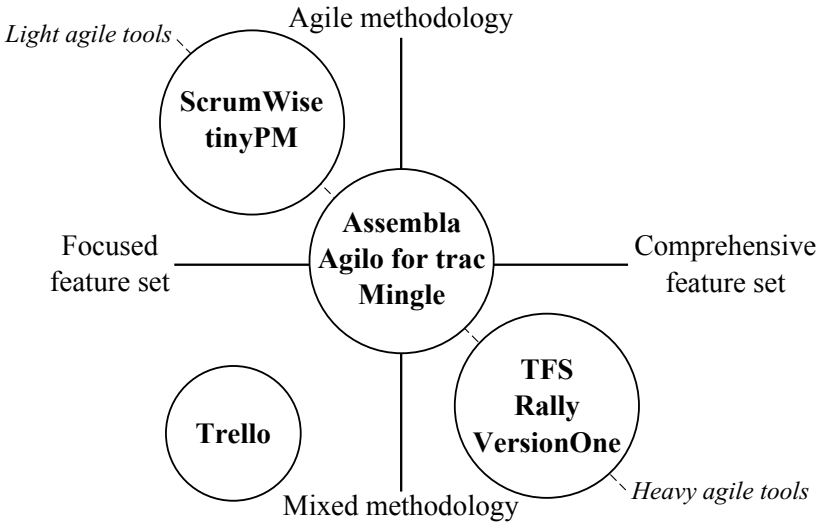


Figure 2.12: The different agile software tools located on the broadness–methodology grid

There are some difficulties when it comes to inserting the tools onto the figure. This mainly considers the three tools in the middle: for instance, Mingle could be considered a little more closer to the area where TFS, Rally and VersionOne are located, and Assembla could be moved to the opposite direction. The location of Agilo for Trac is highly dependent on the configuration, so it is left on the middle section. However, the rest of the tools are somewhat where they should be. Also, every tool except Trello follows the line between light agile tools and heavy agile tools which is logical since Trello is actually more of a non-agile software tool in an agile context.

It should be noted that the figure represents only an approximate and personal vision

about the possible tool locations. In order to accurately and realistically insert a tool onto the figure, it would need weeks or months of data gathered about the usage experiences of the tool in a real agile project.

2.4 Non-Agile Software Tools in an Agile Context

Often organizations already have existing tools when switching over to agile development from another process. These tools may include bug and issue trackers, traditional project management tools (e.g. Microsoft Project) and other software solutions such as version control, build, file sharing and different kinds of collaboration systems. In that context it makes sense to map out the possibilities of all the existing tools before rushing into purchasing possibly an expensive license for a new agile tool.

Microsoft Excel and other spreadsheet applications are typically already installed on workstations, and therefore offer the quickest way to start building and managing backlog. Many Excel backlog templates and examples can be found on the internet, and managing the backlog with them is not necessarily impossible. However, a spreadsheet backlog requires lots of manual work in order to work properly. These aspects are discussed more in the case studies covered in chapters 4 and 5.

It was mentioned earlier that a typical agile tool has characteristics of bug and issue trackers and their ticket functionality. Hence it might be also a possible solution to use bug or issue trackers to track agile work. For instance, most trackers feature custom fields, filters and milestones that can be taken advantage of in order to make product backlog and sprint backlog management possible. However, again, forcing one type of tool into another type of workflow to do agile management may require a lot of manual work, and it should be thought through whether or not it is worth the scrutiny.

When it comes to bug and issue trackers, one of the best possible ways to make an existing tool work in a new agile environment might be to transform it into an agile software

tool by enhancing it with agile plug-ins. Many bug and issue trackers have both official and third-party plug-ins for Scrum and other agile frameworks. The Agilo for Trac tool from the previous section is an example of this type of solution: a free plug-in that can be installed on top of Trac, a popular free bug tracker. On the other hand, agile functionality can be existent in a completely separate agile version of the tool. For example, JIRA [35] is a popular tool used for issue tracking and project management that is also available in a version called JIRA Agile [36] (formerly GreenHopper) that provides many features for agile planning, working and reporting. Using plug-ins or separate agile versions from the same vendor may help the deployment process if the users are already familiar with the earlier system. On the other hand it can sometimes make it harder, as especially plug-ins may require significant amount of work to be done on configuring them.

In general, when switching over to agile development it should be considered as a good opportunity to map out the existing tool needs and solutions. If using the old tool is worth the scrutiny, it certainly should be considered as one of the options.

2.5 Overview of Agile Tool Surveys

Five relevant surveys concerning agile tools, their usage and related topics are discussed here. All of the surveys have been conducted by privately held companies and three of these particularly by two tool vendors themselves. [20][19][37][18][21]

The oldest survey is from 2006: ”Agile Project Management (APM) Tooling Survey Results” by Pete Behrens from Trail Ridge Consulting [20] presents the results of a survey concerning agile method adoption and agile tool usage. The survey was conducted in 39 countries and there were a total of 525 responders of which over 50 % were from the United States. Although the gathered data is almost a decade old, the survey provides a snapshot of agile tool situation back in the day.

The next survey on the list was conducted by an agile tool vendor TargetProcess in

2008. ”Agile Tools. The Good, the Bad and the Ugly” [19] has limited data about agile tool types and products that were gathered with a free form question from 371 people who had requested a trial version of TargetProcess software. Categories have been formed based on the answers and different tools are listed but most of the survey discusses agile tools and their differences in general.

”4th Annual State of Agile Development Survey” from 2009 was conducted by the tool vendor VersionOne [37]. Covering 2570 responses from 88 countries, most of the survey focused on the state of agile development itself. Only a bar graph about different agile and non-agile tools usage percentages is included.

Conducted in 2011, ”Survey of Agile Tool Usage and Needs” by Azizyan, Magarian and Kajko-Mattson of Ericsson AB [18] is the most detailed agile tool survey of all these. It features 121 answers from 35 countries and, unlike other surveys, it also focuses on the needs behind tool selection. Also the agile methods in use were surveyed. The results consist of lots of data about different tool types and products in use, the least satisfactory aspects in tools and most needed features in agile tools.

Finally, the latest survey is also from VersionOne. Following very much the same pattern as the previous survey from 2009, ”7th Annual State of Agile Development Survey” from 2013 covers agile tools more. There are answers from 4048 people around the world, 60 % of which in the United States. In addition to a listing about specific tools in use, the survey report also features a list of preferred tool types and a list of tools that the respondents felt most satisfied with.

2.5.1 Tool Types and Products

The tool categorization varied a lot between the surveys (in a few cases, it was not used at all), but a conclusion can be made that spreadsheets and other office software are extremely popular tool solution in agile development: in [18] from 2011, spreadsheets were used in 23 % of the organisations and in [21] from 2013, Microsoft Excel was the most

popular individual tool product. This is not a surprise, since office suites are usually installed on every workstation anyway. It is hard to say exactly what type of tasks spreadsheets are mostly used for, since this question was only addressed in the Trail Ridge survey from 2006: office tools were much more popular when doing release tracking than iteration tracking (where agile tools were a little bit more popular).

While the actual agile tools were almost on par with spreadsheets and other office solutions back in 2006 [20], in [18] from 2011 they were the most popular single tool category. While these surveys are not necessarily comparable, this might indicate that the popularity of agile tools is, if not growing, at least not declining. Here are some examples of popular agile tools in the latest surveys (not in any order): VersionOne, Rally, XPlanner, ScrumWorks, Microsoft Team Foundation Server, Mingle and JIRA (presumably JIRA Agile—formerly GreenHopper) [18][21].

Different tangible tools have also been almost as popular as the previous two categories throughout all surveys, particularly in co-located teams. However, the different types of tangible tools in use have not been discussed in any of these surveys. In [18], physical wall and paper was used in 26 % of the surveyed companies. Also, the amount of co-located teams using physical tools was twice as big as distributed teams using them.

In addition to these categories, bug trackers (e.g. JIRA, Bugzilla, Trac) have also been popular. It is hard to tell much about the actual usage of bug trackers, since different surveys have acknowledged their existence and others have not: they were the most used tool type in [21] from 2013, while there is no mention of bug trackers in [18] from 2011. Other popular tools have been different wiki systems, traditional project management tools (Microsoft Project has been among the most popular individual tools in every survey) and in-house tools.

2.5.2 Tool Needs

In the ”Survey of Agile Tool Usage and Needs” from 2011, the most wanted agile tool feature was reporting: easy to use but still customizable agile-oriented reports were mentioned by many respondents. The next most wanted feature was integration with existing systems, such as version control and build systems—lack of integration was the most mentioned negative aspect of tools. The third on the list were virtual board features, and especially ones which would replicate physical boards properly and be easy to use. [18]

There were lots of new ideas of features that tools were lacking, such as test scenario creation directly from user stories and more flexible ways to make notes. Lots of comments were user interface and experience related: the ease of use was generally one of the most frequently emphasized aspect of agile tools. [18]

Interestingly, back in 2006 according to [20] the most popular reasons to select an agile tool were making the development process faster and more efficient in addition to have support for traceability and (project) tracking. While it is arguable if two surveys with different respondents are comparable at all, in the 2011 survey [18] these aspects were present but not that dominantly (8 % of the comments addressed the need for various project tracking options). This does not necessarily imply that tool users are not that interested in project tracking, but more likely these aspects have become ordinary and conventional or they do not stir up strong opinionated feelings similarly as something else, e.g. bad task board user experience.

In general, most users (93 %) in [18] had at least something negative to say about the tool being used. Many of these comments were also somewhat conflicting—while simplicity and ease of use were emphasized by some people, others wanted more comprehensive tools with a wide variety of features. The authors conclude that there is no silver bullet when it comes tool selection since the companies and their processes, stakeholders and teams are so different and result in contradicting needs. Hence, the tools that try to please many different customer types are bound to be more complex. [18]

Chapter 3

Agile Tools in Embedded Systems Development

Because agile methods have not yet been widely adopted in embedded systems development, practically all agile tools have been designed software development in mind. Most tools do not offer features that would help to manage and track the parallel development of software and hardware. Discussion around agile tools is very software centric: Not much has been written about agile tools in embedded development and many questions related to this subject have no answers to them. For instance, the constraints that software development centric tools cause, the most relevant tool features and the suitableness of different tool types are unclear in the agile embedded domain. To answer these questions, the author observed tool related challenges in the case companies of the AgiES project.

This chapter lists unanswered questions about agile tools in embedded systems and gives an introduction to the next three case study chapters.

3.1 Open Issues

RQ2 asked what special characteristics there are when an agile tool is used in the context of embedded systems development. Here the question has been divided into three more

detailed questions about tool-critical issues in agile embedded systems development. Answers to these eventually provide answers to RQ2 and also shed light on RQ3 (about how someone should select a tool that supports agile embedded systems development).

Based on the previous chapter we can assume that agile tools generally reflect more or less the principles of agile manifesto. We also know that these principles then need to be modified to be adaptable in embedded systems development [15]. This means that the tool being used to support this adapted process also needs to meet the modified principles. Luckily, most of these principle modifications are not particularly critical from the viewpoint of tools: most probable issues would seem to stem from the original constraints of embedded systems development, such as hardware–software interdependencies.

3.1.1 What Tool-Critical Issues Are There When Agility Is Piloted?

Since two of the companies had zero experience in agile methods before the pilot projects, this was a relevant question to ask. The issues and possible solutions were considered important factors to map out in case companies—both in the tool and outside of it: For instance, it was interesting to know if certain types of tools could counter possible resistance to change in the transition phase of agility. When it comes to agile embedded system development constraints, issues could also arise from heterogeneous teams. It was also clear that an answer to this question would provide extremely beneficial help for other companies and teams that might be piloting agile development of embedded systems.

3.1.2 How Should Backlog Items and Tasks Be Handled in Agile Embedded Development?

While complicated dependencies between requirements, features and tasks exist in plain software development as well, in embedded systems development these dependencies are always present to some extend because of the concurrent development of hardware,

software and mechanics. Real-time requirements might add even more uncertainty when building the backlog. This question needed to be answered to get guidelines for managing product and sprint backlogs in the context of embedded development.

3.1.3 Are There Tool Types or Features That Are Generally Favorable?

Combining the answers of the previous two questions with the knowledge gained in RQ1 (about different agile tools) it should be quite straightforward to say if there are some tool types or features in them that are generally crucial or favorable in agile embedded development.

3.2 Case Study Method

The AgiES project conducted three pilot case projects in the partner companies Ericsson, Nordic ID and Nextfour Group to research agile adoption in embedded systems development. All the cases were carried out in a similar structure, involving reviewing the initial state of the company, agreeing on new agile practices to be piloted and supporting the pilot project from the researchers' side during the whole project while also gathering research data.

Simultaneously with these cases, the author researched the tool usage, problems and improvements in the companies. The author first took part in the case projects mainly by joining meetings and getting familiar with the overall problem domains. The author also took part in helping the case companies to improve their tools. After the structure of the thesis formed, this was expanded to interviews and subsequent e-mail discussions about tool usage in the companies to gain more knowledge on the open issues listed earlier. This resulted in three reports about tool usage and improvements that were then converted into three chapters of this thesis after reviewed and approved by each company.

When it came to work management tools, each company had their own distinct situation at the start of the case: Ericsson did not have any particular tool in use, Nordic ID had been using an issue tracker mainly for bugs and Nextfour had already been using an agile tool for a few years. The agile pilot projects made the companies to develop their tool usage to support new practices. Because of the different starting points, the tool development was also very different ranging from the backlog spreadsheet started from scratch in Ericsson to plug-in development for the agile tool already in use in Nextfour.

The next three chapters summarize the tool utilization and improvements in these three companies during the pilot projects. The differences in how the agile tools were either introduced or improved in each company provided a good opportunity to examine how different tool solutions fit in different usage scenarios.

Chapter 4

Tool Case: Ericsson

The pilot project case was conducted in the digital team of Ericsson, a multinational company that provides various devices and services in the field of communications technology. At the time the research project was started, the team was part of ST-Ericsson, a joint venture of STMicroelectronics and Ericsson which dissolved in the August of 2013. This separation did not cause any big changes to the digital team where the case took place.

Located in Turku, Finland, the digital team is part of a bigger multinational RFIC (radio frequency integrated circuit) team. In addition to the digital team, the RFIC team consists of a system design team and another digital team (both in France), one distributed analog team (located in Turku and Sweden) and a software team that is also distributed (between Turku and other locations). During the AgiES case project, the digital team in Turku consisted of four smaller groups of team members: the front-end (FE) team, the verification team, the back-end (BE) team and the design for test (DFT) team. Most groups had approximately 4–5 members while the DFT team only had two. These four groups were managed by a team leader.

4.1 Overview of the Pilot Project

The case project was gradually started during the fall of 2013 since the transition to the new upcoming project where agile practices were planned to be piloted in was delayed a few times. During this transition phase, the team members were coached on agile practices and the form of the pilot project was shaped. The pilot project finally started in the end of November 2013 and it ended in the spring of 2014 approximately after 6 months of development. The project mainly consisted of upgrade work on older RFIC product designs.

Agile methods had not been used in the team before AgiES. After the results of the interviews and surveys held at Ericsson, three key improvement areas were formed: fluency and planning of the work, feedback enabling continuous improvement and internal communication to create transparency. To improve these areas, the team started adopting a more iterative development process.

Two-week iterations would start with a planning ceremony and end with review and retrospective ceremonies. Since daily meetings were not considered necessary, every other day meetings were held instead. Later these meetings were made even less frequent because the frequency of every other day seemed too overkill as well.

Additionally, possible spare planning ceremony was introduced to give the team a chance to react to sudden requests or changing requirements and possibly adjust the content of the ongoing iteration accordingly. Similarly, the last hour of every day was labeled as a support hour, where large support request would be handled rather than immediately, thus interrupting the work. If the support task could not be processed during a support hour, it was listed on the team backlog (or, in some special occasions, done immediately).

Besides new meetings, backlogs were also introduced. The creation and development of the backlog practices are the key topics of this report.

Right after the beginning of the pilot case, Ericsson launched its own agile adoption project which would eventually modify the team structure under the RFIC team into a

more cross-functional form. However, these changes were not put in action in the digital team during the pilot case.

4.2 Tool Situation at the Beginning

Before the case project, the tasks of the team were not explicitly listed or managed with any tool. Generally, there were requirement lists that propagated through various stages before reaching the RFIC team and finally the digital team. Each team would then form their own specification lists from these requirement lists. These lists were managed with Excel.

Otherwise, there were a lot of different development tools used, starting from scripts and ending with various hardware design tools.

4.3 The Deployment of Backlogs

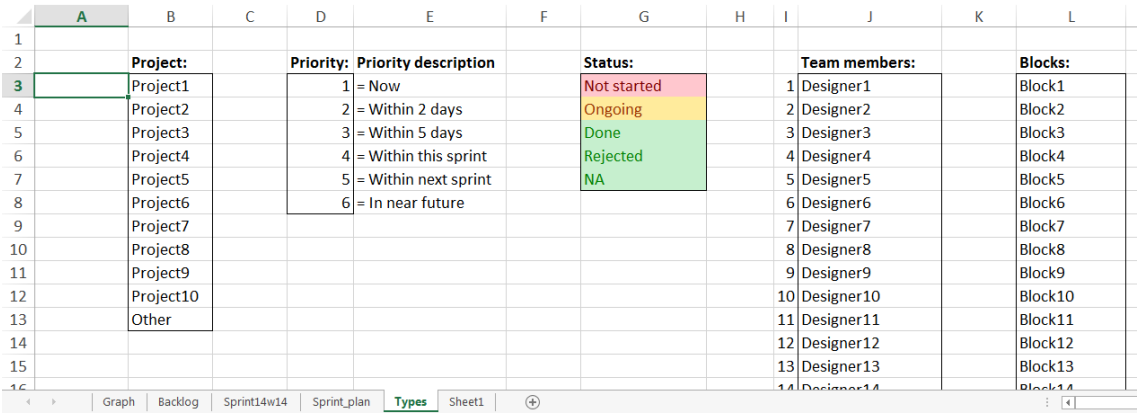
At the beginning of the case, different backlog management options were looked into. After some research it was decided that dedicated agile tools should be ruled out as too heavy: learning how to use one and getting the team familiar with it would have required a lot more effort than was considered appropriate.

The next logical candidate was considered to be Excel. Physical tools were ruled out since there were a lot of tasks to be tracked in addition to needs for sorting, filtering, backups and so on. One backlog template was first taken into consideration but then dropped after it was realized that it would need quite a bit of modifying to suit the current form of development. Thus, the backlog spreadsheet was built from scratch using Excel.

The top integrator of the RFIC team was the main person responsible for creating and improving the backlog spreadsheet. The team leader was also involved in this and the state of backlog development was often a topic in retrospective ceremonies where the entire team could present ideas on backlog improvement. In result, the backlog became

simpler, clearer and more user friendly during the case. For instance, the sheets were easy to sort and filter based on different criteria.

In the Excel file, there were two main sheets: ”Backlog” (containing the product backlog) and ”Sprint backlog”. In addition to these two, there was a ”Sprint planning” sheet where the incomplete tasks of the current sprint would be cut and pasted in the sprint review and an empty ”Graph” sheet that was created for future usage for burn down charts etc. and was left empty during the case. There was also a sheet called ”Types” where certain details about different backlog fields and their values were clarified. This is visible in Figure 4.1.



	Project:	Priority:	Priority description	Status:		Team members:	Blocks:
3	Project1	1	= Now	Not started	1	Designer1	Block1
4	Project2	2	= Within 2 days	Ongoing	2	Designer2	Block2
5	Project3	3	= Within 5 days	Done	3	Designer3	Block3
6	Project4	4	= Within this sprint	Rejected	4	Designer4	Block4
7	Project5	5	= Within next sprint	NA	5	Designer5	Block5
8	Project6	6	= In near future		6	Designer6	Block6
9	Project7				7	Designer7	Block7
10	Project8				8	Designer8	Block8
11	Project9				9	Designer9	Block9
12	Project10				10	Designer10	Block10
13	Other				11	Designer11	Block11
14					12	Designer12	Block12
15					13	Designer13	Block13

Graph | Backlog | Sprint14w14 | Sprint_plan | Types | Sheet1

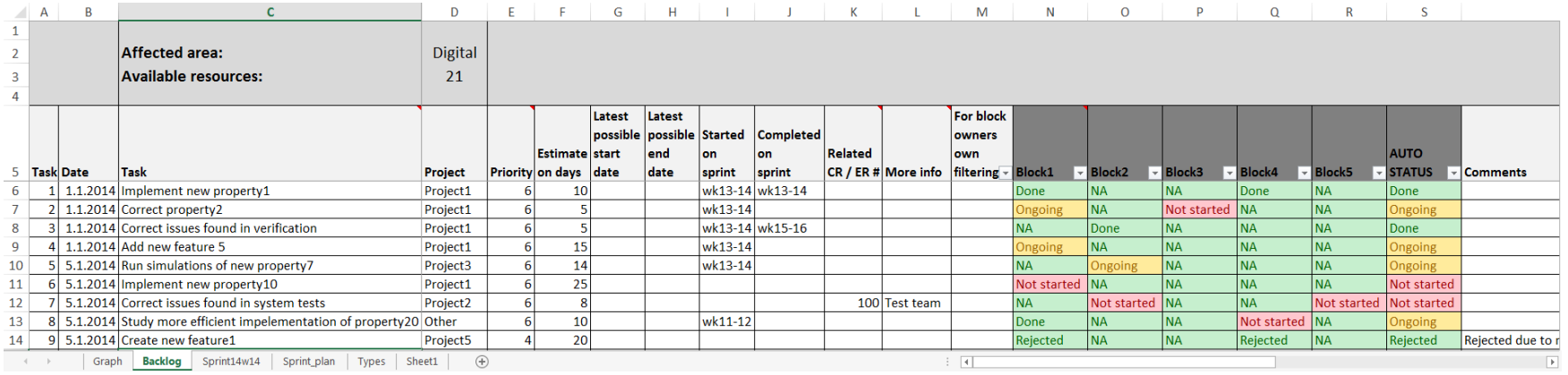
Figure 4.1: An abstracted version of the ”Type” sheet

The team leader managed the backlog spreadsheet on a general level but each team member was still responsible for adding and updating their own tasks directly on the product and sprint backlogs. There were guidelines about the minimum amount of information (e.g. team member in charge, priority, etc.) that people needed to fill in when adding a task.

The backlog spreadsheet was backed up and version controlled on the network file system. File locking ensured that the file was only modified by one person at a time. The team did try to use Git first, but the file formats of Excel were not suitable for this: binary formats could not be merged and textual file formats like XML removed all the color information from the sheets which was a major drawback.

4.3.1 Product Backlog

During the case, most of the time the product backlog consisted of way over a hundred items based on earlier specifications. The backlog items were on a task level and, in general, people referred to them as ”tasks”. There was no higher abstraction level in use, such as features, user stories or requirements. One reason for this was that the earlier requirements and specifications were also listed on a task level and this obviated the need for transition. On the other hand, a feature level abstraction was first experimented but it ended up creating too big features that would last too long. Also, there were subtasks on the sprint backlog so ultimately the product backlog was still on a higher level than the sprint backlog. A screenshot of the backlog sheet can be seen in Figure 4.2.



1																			
2		**Affected area:**	Digital																
3		**Available resources:**	21																
4																			
5	**Task**	**Date**	**Task**	**Project**	**Priority**	**Estimate on days**	**Latest possible start date**	**Latest possible end date**	**Started on sprint**	**Completed on sprint**	**Related CR / ER #**	**More info**	**For block owners own filtering**	**Block1**	**Block2**	**Block3**	**Block4**	**Block5**	**AUTO STATUS**	**Comments**
6	1	1.1.2014	Implement new property1	Project1	6	10			wk13-14	wk13-14				Done	NA	NA	Done	NA	Done	
7	2	1.1.2014	Correct property2	Project1	6	5			wk13-14					Ongoing	NA	Not started	NA	NA	Ongoing	
8	3	1.1.2014	Correct issues found in verification	Project1	6	5			wk13-14	wk15-16				NA	Done	NA	NA	NA	Done	
9	4	1.1.2014	Add new feature 5	Project1	6	15			wk13-14					Ongoing	NA	NA	NA	NA	Ongoing	
10	5	5.1.2014	Run simulations of new property7	Project3	6	14			wk13-14					NA	Ongoing	NA	NA	NA	Ongoing	
11	6	5.1.2014	Implement new property10	Project1	6	25								Not started	NA	NA	NA	NA	Not started	
12	7	5.1.2014	Correct issues found in system tests	Project2	6	8					100	Test team		NA	Not started	NA	NA	Not started	Not started	
13	8	5.1.2014	Study more efficient impelementation of property20	Other	6	10			wk11-12					Done	NA	NA	Not started	NA	Ongoing	
14	9	5.1.2014	Create new feature1	Project5	4	20								Rejected	NA	NA	Rejected	NA	Rejected	Rejected due to r

Graph | Backlog | Sprint14w14 | Sprint_plan | Types | Sheet1

Figure 4.2: An abstracted version of the product backlog

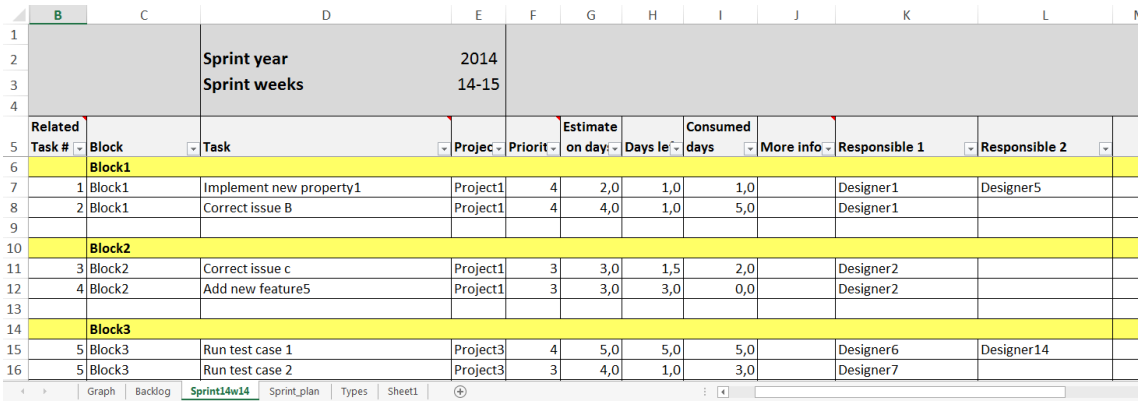
Each row consisted of the ID of the task, short description of it and some more details like the priority. There was also a comment field to present more information about the task or reasons behind changes. One of the most prominent aspects of the sheet was the division to different blocks: the rightmost columns were dedicated to reflect the work status of different tasks on different blocks on the chip or distinct groups of team members. Not all tasks were relevant to all blocks and groups, in which case the N/A status was used. Because one task could affect multiple blocks, there were no assigned owners on backlog tasks. Instead, they were assigned to subtasks on the sprint backlog.

Each task had certain fields such as ID, description and project, which were mandatory to fill in. The sprints where a task had been started and completed were also filled in their own cells. Estimating was considered challenging on the product backlog level because of major inaccuracies. There were also two columns spared for possible usage in the future: latest possible start date and end date. These were not used in the case project since there was not enough information available to fill them. Finally, there were own columns for providing more info on the task, additional comments about status changes and possible tags written by the owner of the block (the ”For block owner’s own filtering” column).

4.3.2 Sprint Backlog

The sprint backlog consisted of sprint-specific subtasks of the product backlog and was created during the sprint planning ceremony. Each subtask contained the ID of its parent task which made it possible to track the progression of the corresponding items on the product backlog level. Rather than having blocks and groups as columns (like they were in the product backlog), each subtask was placed under a corresponding header depicting the block or the group it belonged in. The structure of the sprint backlog sheet can be seen in Figure 4.3.

Similarly to the product backlog, the corresponding project was also present on the sprint backlog sheet. In addition to the designer or designers responsible, additional more



	B	C	D	E	F	G	H	I	J	K	L
1											
2			Sprint year	2014							
3			Sprint weeks	14-15							
4											
5	Related Task #	Block	Task	Projec	Priorit	Estimate on day	Days le	Consumed days	More info	Responsible 1	Responsible 2
6		Block1									
7	1	Block1	Implement new property1	Project1	4	2,0	1,0	1,0		Designer1	Designer5
8	2	Block1	Correct issue B	Project1	4	4,0	1,0	5,0		Designer1	
9											
10		Block2									
11	3	Block2	Correct issue c	Project1	3	3,0	1,5	2,0		Designer2	
12	4	Block2	Add new feature5	Project1	3	3,0	3,0	0,0		Designer2	
13											
14		Block3									
15	5	Block3	Run test case 1	Project3	4	5,0	5,0	5,0		Designer6	Designer14
16	5	Block3	Run test case 2	Project3	3	4,0	1,0	3,0		Designer7	

Graph | Backlog | Sprint14w14 | Sprint_plan | Types | Sheet1

Figure 4.3: An abstracted version of the sprint backlog

information and priority, the key columns were the estimation columns. Each subtask was aimed to be approximately a few days' work, even though both smaller and bigger subtasks were present. There were three columns in use for this: the first column described the original estimation in days, the second one was an approximate value of days left (updated by a responsible team member) and the last column was used to track actual days used for the task (also updated by the responsible team member). This way the accuracy of estimates was easy to track and adjust in the future sprints.

4.4 Conclusions and Possible Future Improvements

As no previous tool was in use before the case, the Excel provided an instantly available way to start tracking work items among new agile practices. In particular, the division by blocks (and groups) was very beneficial and would not have been present in already existing templates online. Also, getting the wanted basic functionality working on a dedicated agile tool would have required lots of resources. Still, there are a few areas of improvement that should be considered in the future.

People did not always update their own tasks on the backlog sheet as actively as it was desired. While people became more active to do this during the case, there still might be a need for further incentives. Estimating the tasks was also considered challenging, which

might relate to this problem.

The graph sheet was not implemented during the case period, so introducing burn down charts and some other metrics on the sheet would be a next logical step. Also, the hierarchy of tasks and subtasks is likely to be changed in future projects since if a new product backlog was built on a task level the amount of backlog items would be enormous. This is why the next project will probably have a feature-level product backlog—how this might affect the sprint backlog and its subtasks is unclear.

One feature request both from team leader and members was ways to make tracking of work items more release-driven. It would have been beneficial for each team member to see which blocks and features were mandatory for the next release and which were optional. Implementing something to cope with this need would require upgrades on the spreadsheet, such as starting to assign subtasks or tasks to release milestones on a different sheet and tracking the importance of each row in relation to different blocks.

There are a few areas of improvement that could be addressed by considering another tool than Excel, possibly an agile software tool. One of these is the need for manual work when using the spreadsheets. However, this mainly concerns the backlog development in general and sprint planning ceremonies where items are needed to be moved around the sheets. A dedicated agile tool could also provide easier ways to implement more levels of abstraction and possibly even the release tracking. On the other hand, these two areas in addition to now easily implemented status tracking system of different blocks might require varying amounts of customization depending on the tool in question. Selecting an appropriate tool that allows this might also be a challenging task.

Chapter 5

Tool Case: Nordic ID

Located in Salo, Finland, Nordic ID develops and manufactures mobile and fixed RFID and barcode reader devices and services that can be used in retail stores and warehouses, for example. The produced devices are typically used within a larger system integrated by another company, which means that the customer, from the perspective of Nordic ID, is typically one of the many integrators who then sells the whole system to the actual end user.

Nordic ID does both hardware and software development along with mechanics design, while the actual manufacturing of hardware and mechanics is outsourced. The hardware and software teams work quite independently and the team members are distributed into two locations: hardware is mostly developed in Salo, software mostly in Turku.

5.1 Overview of the Pilot Project

Lasting approximately six months, the case project was conducted in the 2013–2014 winter as a part of an already existing two-year new product development project in Nordic ID that had been started in the spring of 2012. The developed product was a new RFID reader and at the time the AgiES case project started the project had been already going on for over a year.

The rationale behind the case project was to put new practices in use towards the ending of the project and, at the same time, to make the overall product development process more agile in Nordic ID. After the initial interviews and surveys at Nordic ID, it was decided that the focus of the case project should be on four key areas of improvement: transparency, clarity of the development process, tool usage and documentation practices.

Before the case, the product development process was very waterfall-styled and based on milestones. From the developers' points of view, milestones were mainly seen as deadlines for documentation and the actual structure of the development process was considered a bit fuzzy. Basically the backbones of the development process were the highly experienced employees with intrinsic knowledge on what to do and when. In the case project, there were three software designers, three hardware designers, one tester and one project manager also responsible for mechanics. The team was intensively supported by the research and development manager.

The case period started in the October of 2013. Several agile practices were piloted during the Nordic ID case. New ceremonies (weekly stand-up, planning, review and retrospective) were introduced that partly overrode earlier meetings in order to make the development process more iterative and systematic. The iterations in use had a length of two months at the beginning and one month at the end of the case. The new practices did not particularly touch the existing timetables (milestones). Considering tools, the most relevant new practice was the backlog, which did not exist before the project. Introducing and improving the backlog made people think more about tasks, their sizes and their pacing when doing product development and improving the product development process. This also made people to think more about their personal work in relation to other team members.

The case ended in the April of 2014. After the case, the product development project continued on its own for a few months.

5.2 Tool Situation at the Beginning

At the beginning of the case, the Mantis bug tracker system was already in use but was rarely used by the hardware team. There were mainly bugs and other issues listed on Mantis, and there was no clear list of actual work items. The specifications of the product were listed in spreadsheets elsewhere and there was no actual product backlog in use. Mantis and spreadsheets were basically the only tools used for work management.

A PDM system was used for the archival of documentation and final designs. Other tools included the CVS version control system and many different software and hardware development tools, depending on the development platforms. Instant messaging was also used to communicate between the two locations.

5.3 Tool Development During the Case

There were several improvements done to the tools and their usage during the case. In a nutshell, the bug tracker Mantis was used to keep track of tasks (approximately 1–5 days' work) by inputting them into the system as issues and the product backlog (with backlog items of approximately one sprint's work) was created using an Excel spreadsheet. Tangible tools probably would not have done the trick because of the distributed teams.

5.3.1 Mantis

Both the Mantis bug tracker itself and its usage were improved during the case project. Being an issue tracker, Mantis does not have any particular support for iterative workflows. After some research, a Scrum plug-in was discovered online. The plug-in was very limited and only provided a task board as a separated page in Mantis. The page would show all the issues that were assigned to a selected sprint in a way that the plug-in interpreted product versions and their deadlines being equivalent to sprints. This was unacceptable in Nordic ID, where the versions could not be produced in the end of each

sprint, and the version numbering in use was different between software and hardware development anyway.

The plug-in was still taken into use. At the forum section of Mantis website there was a user who had made his own improvements to the original plug-in by adding support for estimations. This ”ScrumPatch” plugin was downloaded and, after some further modifications to its PHP code, it was put in use on the server of Nordic ID. These modifications were done both by the author and the software lead of Nordic ID. Instead of the version-based system, new tags were inserted in Mantis for sprints, e.g. ”Sprint1”, ”Sprint2” and so on. The description of each tag would tell the plug-in the end date of the sprint in question, following a ”DD-MM-YYYY” pattern. Then a corresponding tag was inserted to each issue which was assigned to that sprint (Figure 5.1), and the plug-in would get all the issues of the sprint from the database just like before.

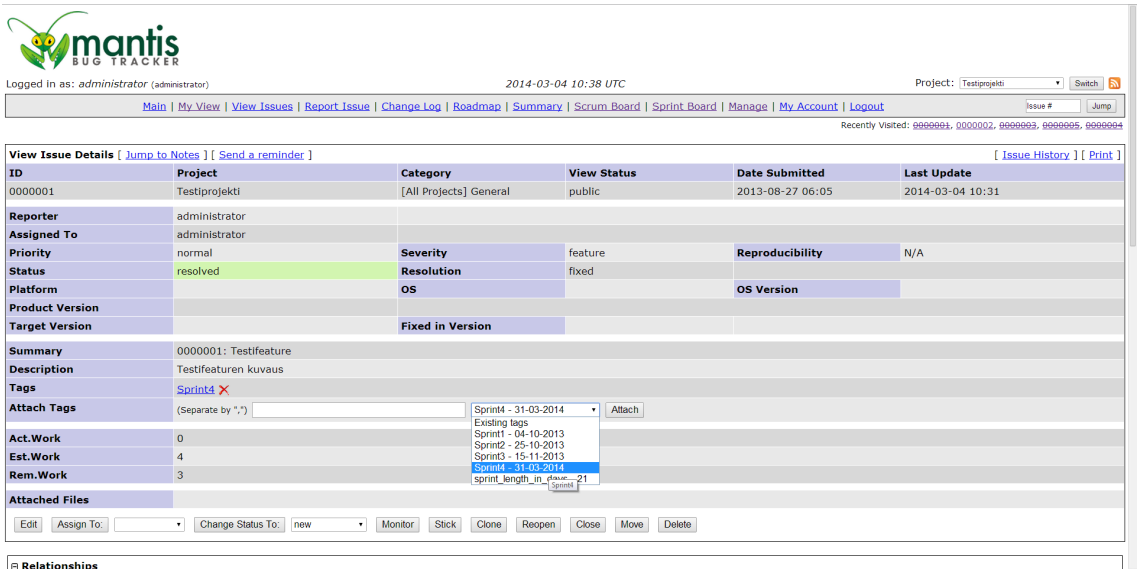


Figure 5.1: Attaching a task to a sprint

The original plug-in also did not list issues of subprojects: this was fixed. The length of sprints was hard-coded in one of the PHP files of the original plug-in. This was made a little easier to change, also with the usage of tags: the description of the ”sprint_length_in_days” tag in the database would tell the length. Later, one additional

feature was added: a checkbox on the sprint board page that hides all the issues that have not been modified during the last seven days. This was useful in weekly meetings. There were also minor tweaks done to the bar elements on the top of the board. The final form of the custom task board page can be seen in Figure 5.2.

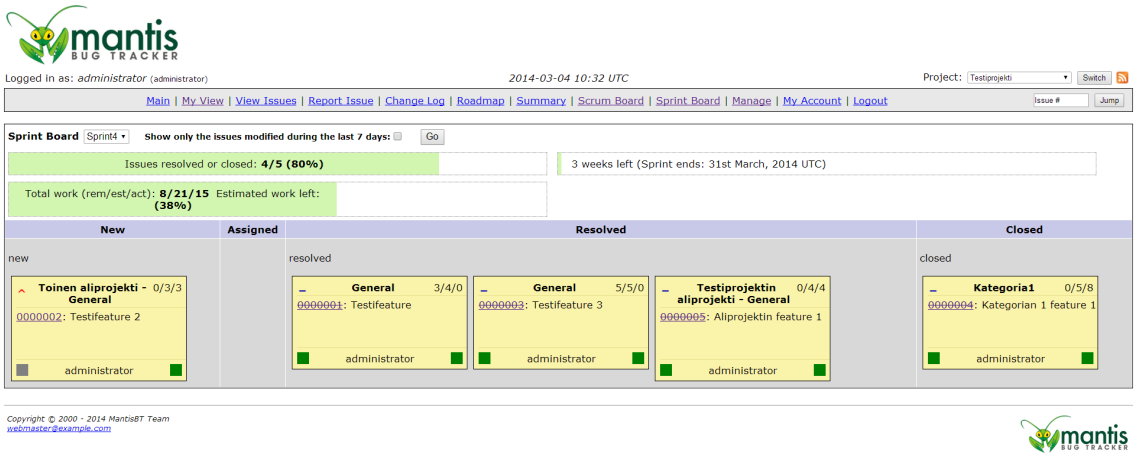


Figure 5.2: The custom sprint task board in Mantis containing test data

The plug-in shows the severity and other information of the issues on the side of the cards, but there is no way to interact with the cards other than pressing the ID link to get to the details of an issue: For instance, dragging and dropping to change a card state or prioritizing the cards by reordering them is not possible. Hence, this plug-in is not perfect, but provides a quick way to see all of the relevant tasks on one page.

Mantis was taken more into the development process than before. It became a regularity that all the team members reported their tasks on Mantis before sprint planning ceremonies. After a few iterations, the amount of issues reported on the system was highly increased and the on-going work could actually be tracked in Mantis unlike before.

Mantis and its usage was a recurring theme in the retrospective ceremonies. Several little improvements were done during the whole case based on the points that came up in the retrospectives. For instance, if people did not update their own tasks, the importance of updates was emphasized in a retrospective session. In addition, tasks themselves were often seen as inconsistent and too big, and this led to many conversations about the proper

sizing guidelines.

In weekly stand-up meetings, the sprint board page was used to show only the issues that were modified during the last seven days. If there were no big changes on an issue (e.g. the status field had been stayed the same during the whole week), the commenting function was used to shortly describe what had been done on the issue or if there was something blocking from proceeding with it: this made the issue pass the week filter. This workflow required that everyone went through their assigned tasks of theirs before the weekly meeting, which was a good way to encourage people to visit the issue tracker at least once a week.

In review ceremonies, Mantis was used as a tool to guide the conversation about the latest sprint. All the tasks of individual team members were systematically went through, both done (resolved issues) and unfinished. After this, the resolved issues were closed and the board was ready for the next sprint planning.

In planning ceremonies, Mantis was systematically started to being used for creating new tasks and directly assigning them to team members. Prioritizing was not explicitly done since the original prioritization functionality in Mantis (the severity field of issues) had never been used and since the plug-in did not support any prioritization functionality either.

Later in the case, the team also started experimenting with task estimation. The patched version of the plug-in, which was initially downloaded, required three custom fields to put in use in the system: actual work, estimated work and remaining work. The total amount of each of these is shown on top of the board page, along with the percentage of undone work which was calculated by dividing the total remaining work with the total actual work. At the beginning of a sprint, the ”estimated” and ”remaining” fields of an issue would be equivalent. Then during the sprint the value of the remaining work was decreased in relation to the completed work. Additionally, the ”actual” field was updated to track the time that it actually took to finish a task—the sum of ”remaining” and ”ac-

tual" did not need to be equal to "estimated". The "actual" field also served the purpose of making the issue pass the week filter.

The estimates were measured in days since no more accuracy was needed. Adding the estimates to all the issues was considered very difficult: not only was it challenging to size up the estimates, but it was also complicated because completion of hardware tasks was always very dependent on the progression of mechanics development. Also, simultaneous projects caused problems when trying to estimate employee resources. However, the sizes of tasks and backlog items got smaller and more controllable during the project and the work estimation itself also gradually got better towards the end of the project.

5.3.2 Product Backlog Spreadsheet

The actual product backlog was separated from Mantis since there was no functionality in the bug tracker to provide a higher abstraction level. The backlog was built using an Excel spreadsheet, and little modifications were done during the whole case. Since the case started in the middle of an existing project, the backlog was based on earlier listings about the features of the product. The first version of the product backlog was created by the project manager and the research and development manager.

Initially, the product backlog consisted of rows representing different high level features, such as "display". The columns next to the features described their statuses. The number of the states ended up being eight: "specification", "design", four different "verification" states, "production" and "documentation". Each state was also divided into three cells describing the individual state of hardware, software and mechanics development on that particular feature.

Not all the features involved all three types of development work, in which case the corresponding column was colored black. The cells were marked with an X if that part of the feature had been done before the introduction of the AgiES case project. Otherwise, there would be e.g. "S1" for "Sprint 1" and so on. The plain numbers in cells were

used for sub-prioritization of the features specifically from the viewpoint of hardware, software or mechanics development. These priorities would then be used in the next sprint planning as a basis to create tasks on Mantis. Two example rows of backlog spreadsheet are presented in Figure 5.3.

		Specification			Design		
Priority	**Feature**	**HW**	**SW**	**Mech**	**HW**	**SW**	**Mech**
1	**Bootloader**	x	x		x	x	
2	**Display**	x	x	S1	1	1	1

Figure 5.3: The backlog spreadsheet structure

The basic structure of the backlog table was kept quite the same during the whole case. In addition to the product backlog spreadsheet there was also a Word document produced where the details of the top priority features of the ongoing sprint were listed: for instance, there were details about what the software for bootloader actually meant in the specification phase. This was done because the original spreadsheet was not seen as self-describing enough. Due to the same reason, some additions were made to the original spreadsheet to generally describe the definitions of done for different phases from the viewpoint of hardware, software and mechanics development.

While many modifications were done by the project manager, the whole team was involved in making the backlog more concrete and better to understand. This was done a few times in separate backlog meetings after completed sprints.

The backlog reflected the earlier waterfall-like process but also provided a good hybrid between agile and waterfall-like development: there was a need for synchronizing the development of different features at least to some extent in order to make the product development possible as a whole. The spreadsheet provided a quick way to see the big picture of the development state to help tracking the project.

5.4 Conclusions and Possible Future Improvements

In Nordic ID, no excessive resources were put into tool development and the process of improving the tools consisted of learning-by-doing and developing the tools little by little. This was a good example of taking advantage of existing tools: rather than putting effort into finding, for example, a suitable agile software tool solution, the available tools were immediately put in use (Mantis and Excel). This type of approach reflects the philosophy behind agile methods and was particularly suitable when taking first steps towards agile development in Nordic ID.

The tools in use provided a decent way to manage the first steps of agility. However, there is clearly some room for improvement in the future.

One ongoing issue during the tool case was the difficulties in task sizing. There seemed to be two major reasons behind it. Inexperience was one reason, since agile methods were new to the people: this is something that can be solved in the long run just by practicing. The other reason is directly related to the tools in use: there were only two levels of hierarchy. The high level product backlog on the spreadsheet and the low level issues in Mantis seemed to call for something in the middle but there was no direct support for that.

The hierarchy in use could be interpreted as a one that agile tools typically use: the scope of the backlog items were similar to epics (while they were supposed to be one sprint's work, this varied more or less) and the issues in Mantis were similar to tasks. Hence, the lacking hierarchy level could be something similar to user stories. During the different AgiES cases, user stories have been considered artificial in many occasions when it comes to hardware development, so some experimentation and brainstorming should be done to find a useful language to put in use on this level.

If more hierarchy is wanted, this likely means that some other software tool should be considered to put in use instead of Mantis and its plug-in. Mantis still could be left in use for issue tracking purposes where it seems to function satisfyingly. This might also

improve the user experience, at least in a way that the task board plug-in does not allow any sophisticated interaction with the issues, such as dragging and dropping or sorting them in any way.

Subsequently, when the amount of issues per sprint increase, the plug-in board is not particularly readable. Finally, there is no link between items on Excel and tasks on Mantis in the current solution. While there might be a way to create some sort of linkage between the spreadsheet and the web browser, the flexibility of a dedicated agile tool may still be preferable.

Chapter 6

Tool Case: Nextfour Group

Founded in 2007, Nextfour Group is located in Turku, Finland. It develops embedded systems for medical, industrial and safety-critical markets based on their clients' requirements. Thus, the products are not typically intended for the mass market. The core competence is focused on authority regulated devices and thus the quality system and the certifications play a major role. Hardware and software development are done in-house while mechanics design is outsourced.

Before the AgiES case project, some agile practices were already taken into use in Nextfour, mainly from the Scrum method. These practices included sprints and regular meetings such as daily stand-ups. The developer teams were also already quite self-organizing and, while many team members had expertise in different areas, also cross-functional. The developers were also collectively in charge of the utilized agile practices.

6.1 Overview of the Pilot Project

Nextfour wanted to take their agile development further, thus taking part of the AgiES project along with other case companies. An adequate case project for this was decided to be a mini project that lasted approximately two months in the fall of 2013. The project consisted of internal platform development and there was no customer involved. While

the combination of team members varied a little bit during the case, most of the time there were a total of five team members in the project, two of which only worked part-time. Two team members were in the hardware team and three team members in the software team.

After the results of the interviews and surveys held at Nextfour, it was decided that the case project should focus on refining the existing process and agility in Nextfour in addition to testing and feedback practices. The team was given more freedom and responsibilities, thus making it more self-organized. Existing sprint planning ceremonies were replaced with sprint meeting days (containing review, retrospective and planning ceremonies). The planning poker was also introduced. The existing backlog practices were decided to be improved. Existing project phases were also reduced and simplified for the case project.

6.2 History of Agilo for Trac in Nextfour

Agilo for Trac is an agile project management plug-in built on top of the Trac issue tracker, which means that the customizations that can be implemented in Trac can be implemented in Agilo as well. The main feature that Agilo offers compared to a pure Trac installation is backlog functionality. This extends the ticket system of Trac to manage different types of items, such as user stories, requirements and tasks, relative to each other. A great amount of customizability is possible via settings, third-party plug-ins and completely own plug-ins written in Python.

While the pure issue tracker Trac was used in the early days, Agilo for Trac had already been in use for several years in Nextfour at the time of the case project. In the project, the running Agilo version was 0.9.12. Agilo for Trac is available in two versions, Free and Pro, of which Nextfour used Free. One instance of Agilo or Trac can run multiple projects but projects cannot interact with each other, which meant that basically every project of Nextfour had their own Trac server instance running. This had led to a situation

where one project could have a different Agilo for Trac setup than another.

Agilo and its customization facilitated many improvements to the agile practices of Nextfour. One of the desired areas of improvement was the estimation and predictability of agile projects, in which the tool played a major role. Some areas of Agilo were customized more than the others. For instance, while the backlog structure was kept very much the same as it was before the case project, the wiki system and the roadmap were customized a lot in order to improve visibility and predictability of the project status.

6.3 Handling of Work Items

Nextfour had already used the same backlog and ticket type structure for a few years at the time of the case, and this was not changed as it was not seen advantageous.

There were three backlog levels in use: product, milestone and sprint. A freshly installed Agilo project includes only product and sprint backlogs, but custom backlogs, such as milestone backlogs, can be added from the settings.

By default, a project in Agilo consists of requirements that may contain user stories that may contain tasks. User stories were not considered natural for embedded projects so a simple hierarchy of requirements and tasks had been chosen instead in Nextfour. There were other types in use too, but these were the only main building blocks of the backlog system itself. Requirements were used in the product and milestone levels and tasks were used in the sprint level. Requirements were split into tasks in sprint planning meetings to fill the corresponding sprint backlog. Every requirement had an owner responsible for making sure that every task linked to it was progressing. Tasks could have different owners than the corresponding parent requirements.

The way the work items travelled between the backlogs was somewhat different than in a typical Scrum tool workflow:

- **Product backlog** was basically a list that contained future work that had not yet

been accepted into any milestone. This meant that it was possible for the product backlog to contain requirements that were never going to be implemented. When a requirement was accepted to be implemented, it was moved to an appropriate milestone backlog.

- **Milestone backlog** was used more in a way product backlogs are typically used: a prioritized list of work items that are going to be implemented. While milestone backlog was on a requirement level, subtasks were still visible under each requirement in the milestone view as well. In this case project, there was only one milestone backlog for the requirements of the actual platform and another for the requirements of a demo application that was used to demo the platform itself. In a bigger project the milestone setup would have probably been different, for instance, including milestones for different phases of the development process.

- **Sprint backlogs** contained all the tasks assigned to the corresponding sprint. The parent requirements were also visible on the list. In sprint planning ceremonies, requirements were split into tasks and the created tasks would then be assigned to the next sprint. The target was to keep the task sizes at about two days.

The lifecycle of tasks consisted of states ”new”, ”ongoing” and ”done”. The lifecycle of requirements only consisted of states ”new” and ”closed”. Requirements were closed in sprint reviews. The default configuration of ticket fields was left mostly intact, even though not all the fields were being used.

There were custom fields in use that mainly dealt with estimation. When it came to requirements, the original estimation field provided by Trac was used to store the original time estimate of the requirement in days. Then the custom field called ”Spent” was used to track the actual billable work in days spent on the requirement. When it came to tasks, the ”Original Time” was used for the original estimate in hours (approximately two days of work) and the ”Remaining Time” field for the developer’s own estimate about how

much work was still approximately left to do on this task. In addition to these, a ”Spent” field was also present on tasks, where it was used to track billable work hours. This was then used as a basis to list work days on the ”Spent” field of the parent requirement.

The distinction of hardware and software requirements was achieved with the keyword field that worked similarly to tag functionality in other issue tracker software. Fields like ”component” could be used but only allowed one selection per time (e.g. the ”RFID_HW” component and nothing else). These were not visible on backlog listings though.

Milestone backlogs were prioritized by dragging and dropping the requirements into a wanted order directly in the backlog view, which made the priority field of tickets quite useless and was set to ”Normal” on almost every ticket. Prioritization was not used on the sprint level since all the work that was assigned to a sprint was usually considered equally important and developers knew what to do in what order anyway.

The backlog listings mostly ran on default settings. The milestone backlog view had only been customized in a way that it also included columns for the ”Estimation” and ”Spent” fields as well. This way the sum of ”Estimation” and ”Spent” values of all requirements was also visible at the bottom of the list, making it easy to observe if the work of the milestone was too overestimated or underestimated.

An illustrative example of the sprint backlog is presented in Figure 6.1.

6.4 Handling of Feedback and New Ideas

Customers and other project members could give feedback by creating new feedback tickets on Agilo. These tickets would then be converted into requirements or tasks by developers. Feedback tickets were mostly created in retrospective ceremonies and a majority of them concerned Agilo and Trac itself.

There was also an idea pool in use: new idea tickets could be reported that could be easily converted into requirements. However, later during the case project it was noticed

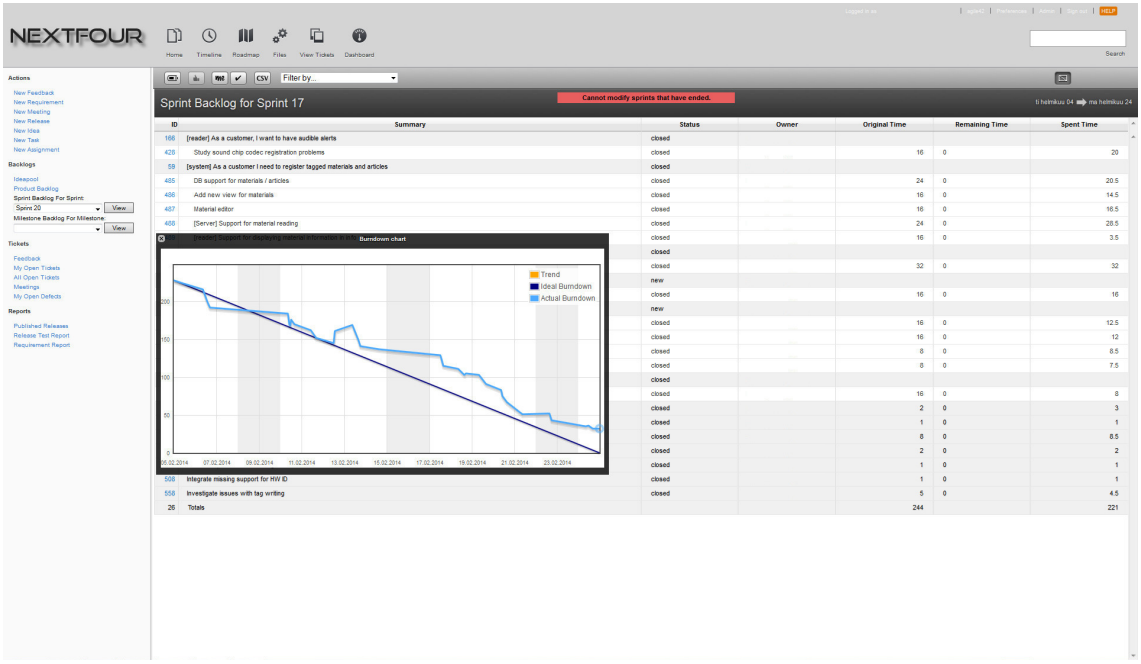


Figure 6.1: The sprint backlog page in Agilo containing mock data

that the product backlog offered very much the same kind of functionality for the team so the idea pool was rarely used.

6.5 Tracking, Estimating, Automatization and Plug-Ins

Tracking and estimating the development work and automatizing different things was considered key advantages of tool usage and were something that were developed most during the case project.

Improvements were achieved by taking advantage of the flexible plug-in system of Trac which is based on the component architecture which essentially comes down to interfaces and extending the existing functionality rather than rewriting it. The system not only allows own macros to be used on the wiki system but also makes it possible to write custom HTML almost anywhere on the Agilo pages. Lots of plug-in functionality was written by Nextfour itself rather than just using existing plug-ins available on the web.

6.5.1 Wiki System

Being essentially the same as in a pure Trac installation, the wiki system of Agilo for Trac allows different ways to document everything related to an ongoing project. It is also plug-in extendable like other parts of Trac: a key extension feature in this sense are macros that are essentially functions within wiki text. Self-built macros allow custom function calling directly from wiki pages, and this was taken advantage of in many occasions. For instance, the project plan was included in wiki, and the information and change history of this document were automatically shown on the project plan page.

One of the most important custom made functionality in the wiki system was present on the front page of the wiki. There was a table of different overall metrics of the project itself in addition to the server environment. Every metric had three possible states: ”OK”, ”WARNING” and ”ALERT”. First, the warning state would kick in, and, if nothing was done to it, after a certain period of time the alert state would be fired. The details column on the right described the cause of the alarm or the warning. This was all achieved with custom plug-in code. All in all, the table provided a clear overview to the system. An example of the front page of the wiki is presented in Figure 6.2.

6.5.2 Sprint Burn Down Chart

Agilo offers sprint burn down charts out of the box. These can be viewed on the dashboard page and also directly in sprint backlog views. Sprint burn down charts provided decent information about the progression of the sprint, although weekends could not be filtered out from the burn down chart: even though there was a setting for this, it did not work—this was also tested on Agilo 1.3 and the same bug still existed.

The sprint burn down chart is visible in Figure 6.1 where the sprint backlog is presented.

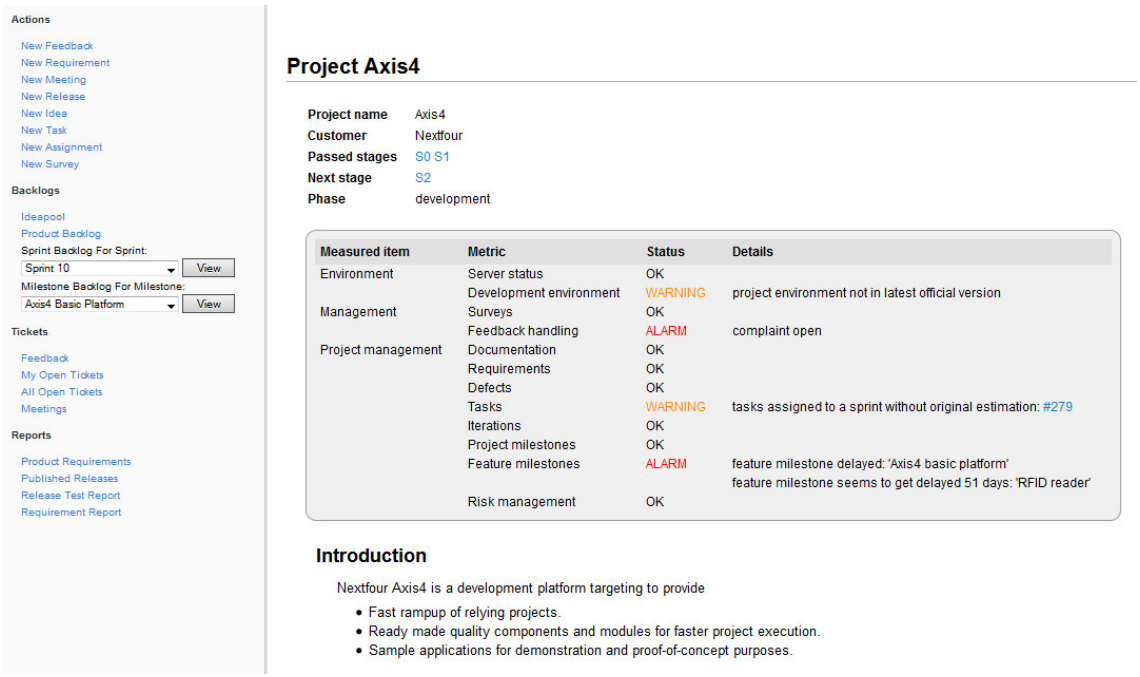


Figure 6.2: An example view on the home page of the project wiki

6.5.3 Roadmap Page and Milestone Burn Down Chart

The default roadmap page of Trac provides an overall view on the tickets and aims to help planning and management of the future development of the project. It consists of a list of milestones, details about them and, being also based on the wiki system, offers custom queries and other customization.

On the roadmap page of Nextfour, a lot of the existing functionality was overridden by custom queries and plug-in components during the case. In short, the roadmap page showed the progression of milestones and its sprints. The key parts of the page were the milestone burn down chart and sprint progression in percentages. An example of roadmap functionality is presented in Figure 6.3.

The default roadmap page tracked the progression of milestones by showing a progress bar of open and closed tasks. However, this did not take estimates into consideration. Milestone burn down chart replaced this sort of approach, and offered a relatively accurate way to see where the milestone was going with just one glance. The used burn down

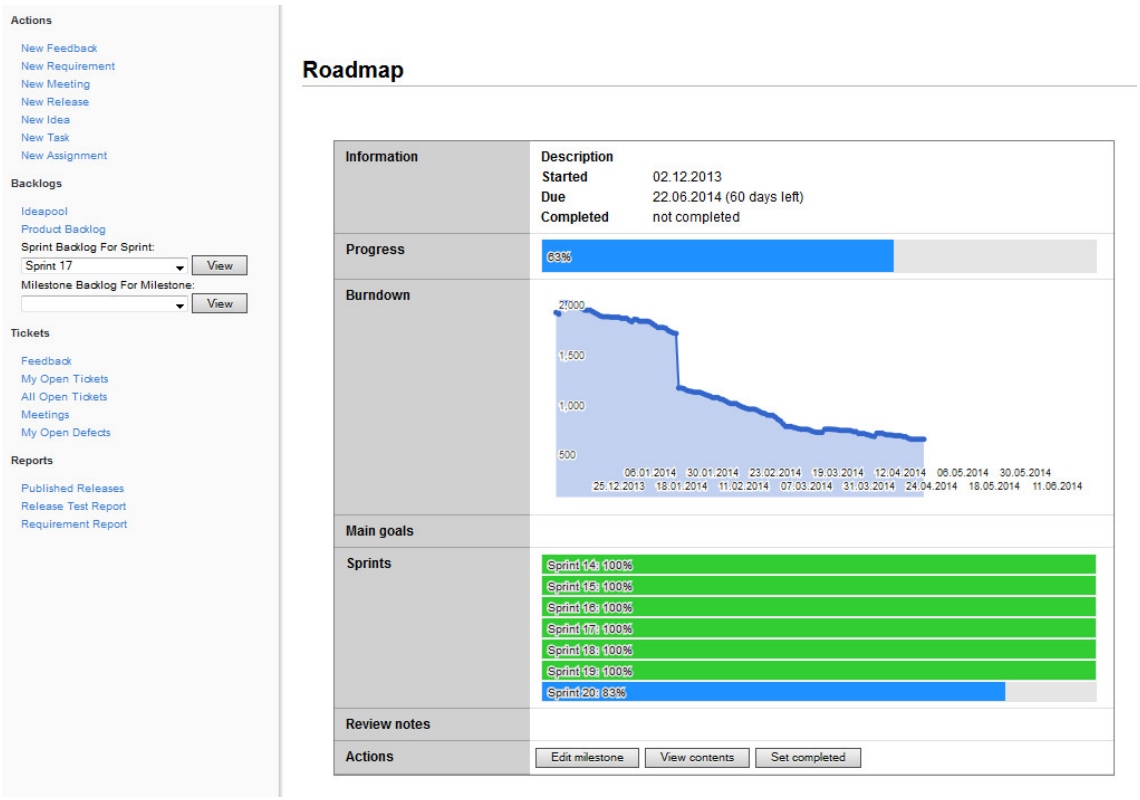


Figure 6.3: An example view on the roadmap page

chart was dynamically created with Google Charts. It filtered out weekends and counted in estimates both from requirements and their subtasks. Initially, all the requirement estimations were added together, and when a task under a requirement was eventually closed, a corresponding amount of hours (the estimated hours of the task) was removed from the burn down.

The default roadmap view presented the progression of sprints the same way it did with milestones: it tracked the open and closed tasks as progress bars. However, in the customized roadmap in Nextfour, the progression of sprints were shown as percentages, which again took the estimations of tasks into account, rather than treating all the tasks equally sized.

6.6 Conclusions and Possible Future Improvements

Agilo for Trac and all its customizations made the development work progression more predictable in the case project.

The development of Agilo had been conducted in quite an ad hoc way even during the case project. When a problem was discovered or a new feature was proposed, then it was looked into. After the case it was agreed that the development of Agilo for Trac would be changed into a more systematic form. This also meant that the customizations done in the case project were started to be spread into the Trac systems of other projects as well—at the same time with the improved agile practices.

The resulted Agilo for Trac configuration still has some areas of improvement. The first category of future improvements consists of bugs and deficiencies of Agilo itself. One clear bug is the faulty functionality with the weekend filter of sprint burn down charts. One useful feature that is particularly lacking in Agilo is the ability to filter and sort backlogs: It is only possible to filter the listings based on one field at most (the ”assigned” field was used in Nextfour to allow showing only the tasks and requirements of one particular person). Any other type of sorting or filtering was not possible. One of the smaller issues was that the ”Show or Hide closed tickets” checkbox on the backload settings only hid closed tasks but left closed requirements visible.

The free version of Agilo does not offer task board functionality but in the Pro version there is a feature called Whiteboard which should do essentially the same thing. This sort of dynamic ticket user interface might be something to look into in the future, but the list view on sprints has mostly been usable enough. If really wanted, the task board feature could also be coded as an in-house plug-in by Nextfour.

It was also speculated if the whole Agilo plug-in was actually needed and could this sort of a system be built right on top of Trac. Agilo offers a Scrum ticket hierarchy with backlogs and a layout that team members have liked but otherwise the actual value of Agilo has not been that enormous because requirements and tasks are mostly used

from the full possible Scrum ticket hierarchy and all the estimation capabilities have been overridden with custom plug-ins. Building the backlog hierarchy and other features in-house rather than using Agilo would probably require a lot of work though and it might not be worth the effort.

The last area of improvement has not that much to do with the tool itself but its users, particularly customers. It was realized that some sort of an incentive would be needed in order to encourage customers to use Agilo to give feedback and search through information. Usually very little feedback was inputted by customers through the system and detailed information that could be find on Agilo was regularly asked via email.

Chapter 7

Conclusions

This chapter addresses all three research questions formed in Section 1.1. In general, answers are provided to all three research questions while certainly leaving room for future research. Especially because of the nature of RQ2 and RQ3 and the major differences between companies adopting agile practices in embedded systems development, unambiguous answers cannot be produced.

7.1 Tool Types and Ways They Support Agile Development Work

Chapter 2 covered many aspects of both non-electronic and electronic tools that support agile development work to answer this research question. It was concluded that *one clear way to describe agile tools is to map them with two axes*: one that represents the line between mixed methodology and agile methodology and another that represents the line between focused feature set and comprehensive feature set. This model was then revisited with real world tool product examples.

Physical tools with their focused feature sets and agile emphasis are *cheap, easy to set up, simple and well in line with agile values and methods*. A typical physical tool is

a task board with separate columns for backlog item cards and task cards. This type of a board was noticed to have been replicated in many agile software tools as well to visualize sprint backlog usage. Physical tools face challenges when used in a distributed teams or in an environment where history tracking, back-ups or integration with other system are needed. When developing embedded systems, these aspects are often present and thus software systems need to be considered.

Various *software solutions* exist to support agile development process. In the Chapter 2, nine different agile tools were introduced and then mapped to the previously defined model of broadness and agility. These tools represented quite an equal variety of tools from different parts of the earlier grid between light and heavy tools. While the amount of features differed among the tools, all of the tools were capable of basic *web-based product and sprint backlog management*. This also turned out to be the *most prominent feature* of agile tools, as the case studies showed in the subsequent chapters. It is no surprise that tools are based around this functionality as it is one of the most important agile practices, particularly in Scrum.

Tool users are not only limited to products marketed as agile to support agile development. Other tools such as *spreadsheet software* and more *traditional project tracking software* can be used, possibly with *plug-ins*. Trello was actually an example of this tool type, even though it was introduced along with other software solutions marketed as agile. In the case studies there were two examples of this: the Scrum Board plug-in for the Mantis bug tracker in Nordic ID and Agilo for Trac running on top of the Trac bug tracker in Nextfour.

After going through five relevant surveys published in recent years, it can be summarized that *popular reasons* for teams to pick a tool are *reporting functionality, integrations with other systems* (version control and build systems) and *virtual board features*. Users would also like—not so surprisingly—these features to be in an *easy-to-use* form. Often people's needs are contradicting: on the one hand, lots of customization and features

might be wanted, but on the other hand, most people consider simplicity and ease of use extremely important. This is why tools that try to please many different customer types tend to be more complex.

Because this thesis mainly focused on agile project management tools, other more development-centric tools have not been discussed even if those tools definitely can have a massive impact on the development process. In [6], a claim is made that ”embedded systems frequently have limited development tools” while there is usually a lot wider variety of tools to choose from in most areas of software development. This is an interesting aspect from the viewpoint of agile tools, and it would probably be beneficial to research how some of the more advanced features like build and version control system integrations could be set up for hardware–software co-development.

7.2 Special Characteristics When Using a Tool to Support Agile Embedded Systems Development

The case studies brought out several characteristics related to tool usage in agile embedded systems and answers to the unanswered questions of Section 3.1. These are covered in their own sections next.

7.2.1 Tool-Critical Issues When Agility Is Piloted

In the pilot phase, there were three distinct tool related issues: creation and estimation of backlog items, motivating users to use the tool and the lack of best practices. These were mainly acquired from the experiences in Ericsson and Nordic ID where agile practices were just started to be adopted.

Creation and Estimation of Backlog Items

Because neither Ericsson or Nordic ID had backlogs or other agile practices in use before AgiES, there were difficulties when backlog items needed to be created and, especially, estimated. Product and sprint backlogs force teams to ponder how to divide their work into smaller segments rather than having big chunks of work constantly under development. Also, during the first sprints it was noticed that some key functionality had not been taken into consideration in the backlog. This is an issue that is likely to get better over time, as both companies more or less got the backlog flow going from the scratch already during the case projects. In the beginning, a team should just select one common language with the backlog items and improve the practices if necessary.

While tools cannot create or split backlog items on behalf of the user, they can lower the bar of tool usage to make the backlog management as painless as possible by, for instance, providing high usability. As the backlog management is one of the most important reasons these tools are used in the first place, this issue is very relevant.

Motivating Users to Use the Tool

When starting to use new tools, there is a risk of people not wanting to get on board especially in heterogeneous teams, like they often are in embedded systems development. This effect was present both in Ericsson and Nordic ID at the beginning of the cases. When people do not update their own tasks, a lot of time is wasted in unnecessary activities in review and other ceremonies, such as when tasks are updated in front of everyone else.

From the viewpoint of agile principles, the tool selection should stem from within the self-organizing team—in this sense, any attempts to increase the motivation in tool usage should also come from inside the team rather than from outside. However, in embedded systems development specialized team members or teams (e.g. hardware and software teams) are usually needed which means that, in addition to self-organization, there also needs to be co-operation between different teams—this was also emphasized in the mod-

ified agile principles in embedded systems development. The nature of heterogeneous teams make it very likely that there are team members that do not consider tool usage as a priority.

When this happens, it should be ensured that at least the majority of the team finds it beneficial to use the tool and thus also encourage other people to make their updates and find the inner motivation. This effect was present at least to some degree in both companies when the case projects reached their end. Obviously, it is also important to give people appropriate guidance on how to use the tool and to select a tool that is as easy to use as possible.

Lack of Best Practices

Because of the lack of research in tools in agile embedded systems development was so prominent, researchers could only provide tool guidance based on the general knowledge related to agile software development. This caused some frustration but also made the development teams of Nordic ID and Ericsson to come up with own solutions based on trial and error. In Nextfour, the tool practices have also evolved a lot during the years. Obviously, the fact that there were not any best practices available was one of the main reasons this thesis was conducted in first first place. With the findings of this thesis, it should be easier to start adopting agile practices in embedded systems development for other embedded development teams.

7.2.2 Managing Backlog Items and Tasks in Agile Embedded Systems Development

While universal guidance cannot be produced based on the case studies, the experiences did however shed light on backlog management in embedded systems development in a form of *three different approaches of agile tool usage*. These experience reports are beneficial from the viewpoint of other teams in similar situations where at least some

level of advice is sought.

The first example was the approach in Ericsson, where the progression of tasks were tracked on every block or team it considered. This sort of tracking might be needed in teams where the focus is mostly on hardware and there are team members developing isolated parts of the product. While the tasks and their subtasks could be inserted into an agile software tool as well, to properly visualize a distinct development process like this might need extra work, such as exporting the data into another system or setting up an appropriate plug-in. Being in a pilot phase, the development process might change in Ericsson in the future, which means that the backlog hierarchy is not necessary in its final form yet.

Nordic ID provided an example solution where a combination of spreadsheet backlogs and the Mantis bug tracker with a task board plug-in were used to track feature progression and low level tasks of hardware, software and mechanics development. This sort of development process is likely very common among different embedded systems development teams. This example also likely was not in its final form, as there was no automatic linkage between the high-level backlog spreadsheet and low-level tasks. In addition, a some sort of a middle level between backlog items and tasks could be profitable in the future. All of these features could be achieved with an agile software tool, but just like with the backlog sheet of Ericsson, visualization of the feature progression would need customizations.

Nextfour had a quite straightforward model of sprint, milestone and product backlogs used in a little bit similar of an environment as in Nordic ID. Hardware and software tasks were distinguished with a keyword filter and—partly due to the nature of the project—no more excessive progression tracking was done backlog item by backlog item, as it was done in the other two companies. The case companies and their projects and products differ so much from each other that it is hard to speculate whether or not this would have any value in Nextfour.

Interesting possible future research would definitely involve either Nordic ID or Ericsson in an experiment where a software agile tool (such as one of the products introduced in Section 2.3) would be used to track the progression of different blocks or features. The visualization of this progression could be achieved either within the tool with a plug-in (possibly with an in-house one) or by importing the data from the tool onto an external tool (either something like Excel or an in-house tool). This would likely give further insight into best practices of backlog usage in agile embedded development. Additionally, this would probably make people question the original backlog hierarchy and possibly improve it in the process: it would be interesting to know if the backlog flow would progress towards the one in use in Nextfour. On the other hand, it would also be interesting to see if, for instance, the Nordic ID type of visualization would be beneficial in Nextfour.

7.2.3 Favorable Tool Types and Features

General guidelines of favorable tool types could not be found in this thesis. The situation the companies and teams are in differ so much that it is fair to say that different tools are better at solving different problems. Even the physical tools, that were not used by any of the case companies, can probably be beneficial in other agile embedded development teams. When pondering between tool categories, the development team needs to find out which tools (summarized in Section 7.1) reflect their needs.

When it comes to favorable tool features, the case studies showed that in the pilot phase the most important reason to use these tools is backlog management. The way backlogs are handled between tools will have a big effect on which tool is eventually chosen to be used. Because development process is in a transition phase, the tools in use also need to focus on the key necessities rather than less important, yet still probably useful, functionality. That being said, there are probably teams that can jump right into a new heavy agile tool system in a middle of agile adoption.

After the team has gathered experiences on its agile tool needs in the pilot phase, then

later on all different kinds of tool approaches can be used as long as it supports the development process. In the case of Nextfour, the solutions have been an agile software tool coupled with own plug-ins and customizations to fit the product into the development process. Also features focusing on estimation, automatization and reporting might become more prominent, as they have been in Nextfour. The backlog management is still very important, as one of the most important tool related daily activities of members in any team is to input and update their own tasks on the system.

7.3 Principles for Selecting a Tool to Support Agile Embedded Systems Development

RQ3 asked how someone should select a tool that supports agile embedded systems development. It was also noted that this question is directly related to the special characteristics covered with RQ2. In light of the knowledge gathered in this thesis, selecting a tool for a real-life agile development process can be a difficult task to do. The amount of tools—especially software solutions—is very high and the idea about needed tool features might be very fuzzy, especially if the team is in the transition phase.

No particular selection process is presented here, even though there are systematic evaluation methods available for tool selection [38][39]. These types of methods were not considered necessary to be addressed in the scope of this thesis, as these types of selection processes can be very heavy and cumbersome. Instead, as seen in the case companies, the actual tool selection can be done in a more ad hoc way, at least as long as the needs of the team are somewhat clear and different options are tested, preferably in a real project. That is not to say a formal selection process would not provide help in other cases or even in the case companies in the future.

Instead of a selection process, for now this question can only be answered with general guidelines on tool selection and usage. Answers to RQ1 and RQ2 gave understanding on

what agile tools there are and what special characteristics there are when they are used in agile embedded systems development. To answer RQ3, this knowledge is presented in a list of guidance principles on what to focus on when selecting the tool. These are the principles and the rationale behind them:

1. **The development team needs to have the biggest role in tool selection.**

   Rationale: The best knowledge of needed features and how the backlog needs to be managed lies within the team itself—thus, the team should have the final word on what tool type or product is chosen. Since there will likely be a lot of resistance if a tool is forced upon a team, this also makes the tool deployment process much easier and utilizes the intrinsic motivation of team members to improve the tool and its usage over time.
2. **When in the middle of development process changes, be open-minded about new tool workflows.**

   Rationale: A very reasonable argument could be made that a tool should always fit in the development process rather than vice versa. However, when the development process is going through changes there is a possibility that all the tool needs have not been completely thought out. This situation should be seen as an opportunity to question the current workflow and to be open about new ways of organizing the work.
3. **A fluent backlog flow is crucial to the entire agile development process.**

   Rationale: It was realized in the case study that the backlog management is challenging in the pilot phase and the issues were discussed repeatedly in ceremonies taking time off more development-centric topics. In conclusion, backlog management seems to be one of the most important feature to be focused on when selecting the tool and it should be improved constantly when issues arise.
4. **The barrier of tool usage needs to be lowered as much as possible.**

   Rationale: If a tool is not used before, making it a natural part of every developer's

workday might be hard in some cases. To lower the barrier of tool usage, the tool needs to be as easy to use as possible. Besides choosing a tool with good usability, other things can be done to lower the barrier: For instance, inputting new tasks on the system can be made easier by giving the team members quick rules of thumb and other guidelines. On the other hand, if the tool needs to appeal to external people like customers to gather issues or feedback, the barrier needs to be even lower. Following this principle might be easier said than done and part of bigger problems in agile adoption in general—regardless, it is important to experiment with different ways to achieve this principle.

5. **A tool can help with estimation but the estimates will never be perfect.**

   Rationale: Even if estimating is done and improved for years, estimates will always be inaccurate and this is something that the development team and stakeholders need to be aware of. This was included in the list since, while some tools offer advanced ways to input and visualize estimations, in the end the estimates will always be only as good as people who make them.

Chapter 8

Summary

This thesis presented the research on project management tool related questions in an already very non-researched field of agile embedded systems development. Originally the idea for this thesis arose from the fact that there was very little knowledge on agile tools in the area of embedded systems development, and from that viewpoint the answers provided to the research questions can be considered relatively sufficient.

To know what kind of agile tools there are in the first place, several different tool types were discussed along with different categorization options. After some discussion it was concluded that in the scope of this thesis agile tools can be described in two axes representing the methodology (agile–traditional) and the feature set (focused–comprehensive). The discussion of different tool types, especially with nine different agile software solutions, should give an insight into the different options on how agile development process can be supported with tools.

The three case studies turned out to provide useful information about real life issues with tools that are used to support agile embedded systems development. Ericsson and Nordic ID were examples where agile practices was started to be adopted and piloted backlog management practices played a major role. In Ericsson, this was tackled with a spreadsheet built from scratch to obtain visibility to the different blocks in progress. In Nordic ID, an already existent tool was enhanced with a task board plug-in and high-

level features were tracked on a spreadsheet of its own. On the other hand, Nextfour had their agile adoption further and an agile software tool was already in use. Due to the customizations and plug-in development the work progression became much more predictable.

While universal conclusions cannot be made based on three different case studies, the pilot phase issues like backlog item creation and user motivation would probably be potential in other companies as well where the development process is going through a similar transformation. Furthermore, the solutions of Nextfour provide one example of a further developed software system where agile project tracking is present in an authority regulated environment. As such, the most valuable aspect of this thesis is to document these three distinct stories about project management tools in agile embedded systems development.

Formed in the light of all gathered knowledge and experiences, the five principles of tool selection and usage are on a very general level, and while they may seem a little bit like common sense they also compress the lessons learnt during the making of this thesis into an utilizable list. The list also shows that, at least in the transformation phase, there might also be a lot of tool issues that are very people-centric more than technical.

When it comes to suggestions for future research, trying to adapt one of the popular software-based agile tools (such as Rally or VersionOne) into agile embedded development process would probably help to form more concrete tool principles. The thesis did not quite get detailed answers on how to manage interdependencies of backlog items and tasks between multiple teams, projects and products, which would be an important question to ask when organization-wide agile practices are wanted to be adopted. This type of research would potentially involve tools that take into account different aspects of collaboration needs, version control and build system integration and application lifecycle management in general.

References

[1] C. Larman and V.R. Basili. Iterative and Incremental Development: A Brief History. *Computer*, 36(6):47–56, 2003.

[2] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for Agile Software Development, 2001. [Online]. `http://agilemanifesto.org/`. Accessed October 9, 2013.

[3] G. Goth. Agile Tool Market Growing with the Philosophy. *Software, IEEE*, 26(2):88–91, 2009.

[4] Matti Kaisti, Ville Rantala, Tapio Mujunen, Sami Hyrynsalmi, Kaisa Könnölä, Tuomas Mäkilä, and Teijo Lehtonen. Agile Methods for Embedded Systems Development: A Literature Review and a Mapping Study. *EURASIP Journal on Embedded Systems*, 2013(1):15, 2013.

[5] Pat Elwer. Agile Project Development at Intel: A Scrum Odyssey. Intel Corporation and Danube Technologies, Inc., 2008.

[6] Doug Dahlby. Applying Agile Methods to Embedded Systems Development, 2004. [Online]. `http://www.embuild.org/dahlby/agileEm/agileEm.pdf`. Accessed January 8, 2014.

[7] Maarit Laanti, Jouni Similä, and Pekka Abrahamsson. Definitions of Agile Software Development and Agility. In *Systems, Software and Services Process Improvement*, volume 364 of *Communications in Computer and Information Science*, pages 247–258. Springer Berlin Heidelberg, 2013.

[8] Craig Larman. *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley, 2004.

[9] P. Rodriguez, J. Markkula, M. Oivo, and K. Turula. Survey on Agile and Lean Usage in Finnish Software Industry. In *Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium on*, pages 139–148, 2012.

[10] Hirotaka Takeuchi and Ikujiro Nonaka. The new new product development game. *Harvard Business Review*, 1986.

[11] Ken Schwaber and Jeff Sutherland. The Scrum Guide, 2013. [Online]. `https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf`. Accessed November 18, 2013.

[12] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, first edition, 1999.

[13] Mary Poppendieck and Tom Poppendieck. *Implementing Lean Software Development - From Concept to Cash*. Addison-Wesley, 2006.

[14] M.O. Ahmad, J. Markkula, and M. Ovio. Kanban in software development: A systematic literature review. In *Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on*, pages 9–16, 2013.

[15] Matti Kaisti, Tapio Mujunen, Tuomas Mäkilä, Ville Rantala, and Teijo Lehtonen. Agile Principles in the Embedded System Development. In *XP 2014 Rome - 15th International Conference on Agile Software Development*, 2014.

[16] Jim Highsmith. Agile Project Management: Principles and Tools. *Cutter Consortium Executive Report*, 4(2), 2002.

[17] Jeong-Ah Kim, Seung Young Choi, and Sun Myung Hwang. Process & Evidence Enable to Automate ALM (Application Lifecycle Management). In *Parallel and Distributed Processing with Applications Workshops (ISPAW), 2011 Ninth IEEE International Symposium on*, pages 348–351, 2011.

[18] G. Azizyan, M.K. Magarian, and M. Kajko-Matsson. Survey of Agile Tool Usage and Needs. In *Agile Conference (AGILE), 2011*, pages 29–38, 2011.

[19] Michael Dubakov and Peter Stevens. Agile Tools. The Good, the Bad and the Ugly. Target Process, Inc., 2008. [Online]. `http://www.targetprocess.com/download/whitepaper/agiletools.pdf`. Accessed October 7, 2013.

[20] Pete Behrens. Agile Project Management (APM) Tooling Survey Results. Trail Ridge Consulting, December 2006. [Online]. `http://lithespeed.com/resources/2006AgileToolingSurveyResults.pdf`. Accessed October 7, 2013.

[21] VersionOne, Inc. 7th Annual State of Agile Development Survey, 2013. [Online]. `http://www.versionone.com/state-of-agile-survey-results/`. Accessed October 7, 2013.

[22] T. Perry. Drifting toward invisibility: The transition to the electronic task board. In *Agile, 2008. AGILE '08. Conference*, pages 496–500, 2008.

[23] Rally Software Development Corp. Rally Software | Enterprise Proven Agile. [Online]. `http://www.rallydev.com/`. Accessed December 2, 2013.

[24] ThoughtWorks, Inc. Mingle - Agile project management software. [Online]. `http://www.thoughtworks.com/products/mingle-agile-project-management`. Accessed January 8, 2014.

[25] CollabNet, Inc. ScrumWorks Pro - Powerful, Agile project management for Scrum, Lean and Kanban | CollabNet. [Online]. http://www.collab.net/products/scrumworks. Accessed January 8, 2014.

[26] VersionOne, Inc. VersionOne. [Online]. http://www.versionone.com/. Accessed December 2, 2013.

[27] Microsoft. Team Foundation Server 2013. [Online]. http://msdn.microsoft.com/en-us/vstudio/ff637362.aspx. Accessed January 13, 2014.

[28] Assembla Inc. Assembla - Task & Issue Management, Subversion & Git Hosting, Collaboration. [Online]. https://www.assembla.com/. Accessed January 8, 2014.

[29] ThoughtWorks, Inc. tinyPM - tiny effort, perfect management - agile collaboration tool. [Online]. http://www.tinypm.com/. Accessed January 8, 2014.

[30] Fog Creek Software, Inc. Trello. [Online]. https://trello.com/. Accessed January 8, 2014.

[31] Agilo Software, GmbH. Agilo for trac - Flexible Scrum & Kanban Tool Based on Trac. [Online]. http://www.agilofortrac.com/. Accessed January 8, 2014.

[32] ScrumWise. Scrum Tools | ScrumWise - The Most Intuitive Scrum Tool. [Online]. http://www.scrumwise.com/. Accessed January 13, 2014.

[33] Walsh, Kevin. Five Tips for Using Trello for Scrum | Kev Walsh | Civic Actions, October 10 2012. [Online]. http://www.civicactions.com/blog/2012/oct/10/five_tips_for_using_trello_for_scrum. Accessed January 27, 2014.

[34] Nervegna, Tommaso. 10 Tips for using Trello as an effective Agile Scrum Project management Tool - Way Out | Curated by Tommaso Nervegna, January 9 2014. [Online]. http://www.tommasonervegna.com/blog/2014/1/9/10-effective-tips-for-using-trello-as-an-agile-scrum-project-management-tool. Accessed January 27, 2014.

[35] Atlassian. JIRA | Issue and Project Tracking Software | Atlassian. [Online]. https://www.atlassian.com/software/jira. Accessed January 13, 2014.

[36] Atlassian. JIRA Agile | Atlassian. [Online]. https://www.atlassian.com/software/jira/agile. Accessed January 13, 2014.

[37] VersionOne, Inc. 4th Annual State of Agile Development Survey, 2009. [Online]. http://www.versionone.com/pdf/2009_state_of_agile_development_survey_results.pdf. Accessed January 13, 2014.

[38] B. Kitchenham, S. Linkman, and D. Law. DESMET: a methodology for evaluating software engineering methods and tools. *Computing Control Engineering Journal*, 8(3):120–126, 1997.

[39] N. Ahmad and P.A. Laplante. Software Project Management Tools: Making a Practical Decision Using AHP. In *Software Engineering Workshop, 2006. SEW '06. 30th Annual IEEE/NASA*, pages 76–84, 2006.