TAMK University of Applied Sciences
Business Information Technology
Jenna-Riia Karhunen

Final Thesis

Scrum Quality Management: an Empirical Study

Supervisor Lic. Phil. Paula Hietala
Commissioned by Nokia Corporation, supervised by test manager Jussi Kuisma
Tampere 06/2009

TAMK University of Applied Sciences
Business Information Technology

Author:	Jenna-Riia Karhunen
Thesis:	Scrum Quality Management: an Empirical Study
Pages:	50 + 3
Graduation time:	June 2009
Thesis Supervisor:	Paula Hietala
Co-operating company:	Nokia Corporation, Tampere

ABSTRACT

This thesis was commisioned by Nokia Corporation and the research problems are related to the issues in one of its development teams. The team has utilized agile methodologies and new methods such as Scrum have brought along new requirements related to flexibility, scalability and transparency of development and software testing processes. The test environment currently in use is very complex and designed for more traditional, test case-based testing and does not respond well enough to the new, agile ways of work. In addition, the development team has issues related to the internal flow of information and test planning.

The original goal of this thesis was to explore, study and present alternative test management tools and to evaluate how usable they are in a Scrum team. By comparative study, the tools were tested and evaluated by using predefined requirements based on the criterion set by the development team and on the personal experience of the author of the study. The outcome of this comparative study was an Excel-spreadsheet that was in accordance with the requirement specification.

During the study the severity of the issues related to the flow of information and test planning in the Scrum teams became more distinct. To answer this transition, the research problem of the thesis was changed so that it would take into account also the newly emerged problems. This was achieved by further developing the outcome of the comparative study.

The final outcome of this thesis is the improved version of the Excel-spreadsheet. The current version has been compiled especially concerning the different roles of the Scrum team: it provides an easily accessible user interface for developers, extensive features for test planning and formability for the test engineer. For the entire team, it gives more flexibility and transparency. At the moment the spreadsheet is used by the author of this thesis and in the future it should be tested in a Scrum team.

The problems highlighted in this thesis and especially their impact on quality has been identified and the work to improve the process in order to solve these issues continues.

Keywords: Agile, SCRUM, Software Testing, Software Test Management, Exploratory Testing

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä:	Jenna-Riia Karhunen
Työn aihe:	Empiirinen tutkimus SCRUM-kehitystiimin ohjelmistotestauksesta
Sivumäärä:	50 + 3
Valmistumisaika:	Kesäkuu 2009
Työn ohjaaja:	Paula Hietala
Työn toimeksiantaja:	Nokia Oyj, Tampere

TIIVISTELMÄ

Tämä opinnäytetyö on tehty toimeksiannosta Nokia Oyj:lle. Työn tutkimusongelmat käsittelevät Scrum-kehitystiimien työssä ilmenneitä ongelmia. Ketteriin menetelmiin siirtyminen on tuonut kehitysprosessiin ja testaukseen uudenlaisia, prosessin läpinäkyvyyteen ja joustavuuteen liittyviä vaatimuksia, joita käytössä oleva, yksityiskohtaisiin testitapauksiin vahvasti perustuva järjestelmä ei kykene täyttämään. Tilannetta huononsivat entisestään kehitystiimien sisäisen tiedonkulun hitaus ja riittämätön testaussuunnittelu.

Työn alkuperäisenä tavoitteena oli etsiä, tutkia ja esitellä vaihtoehtoja käytössä olevalle testaustyökalulle sekä arvioida vaihtoehtoisten työkalujen soveltuvuutta Scrum-kehitystiimin työhön käyttämällä apuna vertailevaa tutkimusta. Tutkimuksen vertaileva osuus perustui yksinkertaistettuun, toimeksiantajan toiveiden sekä työn kirjoittajan kokemusten pohjalta muodostettuun vaatimusmäärittelyyn. Vertailevan tutkimuksen tuloksena syntyi vaatimusmääritelmän mukainen Excel-työkirja testauskirjanpitoa varten.

Tutkimuksen aikana kiinnitettiin erityisesti huomiota lisääntyneisiin ongelmiin Scrum-tiimien tiedonkulussa ja testaussuunnittelussa. Ongelmien luonteen muuttuessa myös työn tavoitetta muutettiin siten, että työ käsittelee nyt myös tutkimuksen lopputuloksen jatkokehitystä sekä ottaa kantaa tiedonkulussa ja testaussuunnittelussa ilmenneiden ongelmien laatuun.

Työn lopputulos on jatkokehitelty Excel-työkirja, jossa on otettu huomioon erityisesti Scrum-tiimin eri roolien vaatimukset. Työkirjasta on tehty kehittäjille nopeakäyttöinen, testaussuunnitteluunkin riittävän kattava sekä testaajan tarpeet huomioiden muokattavissa oleva versio. Tiimin tarpeisiin se tuo lisää läpinäkyvyyttä ja joustavuutta. Tällä hetkellä työkirja on työn kirjoittajan henkilökohtaisessa käytössä ja tulevaisuudessa sitä on tarkoitus koestaa myös Scrum-tiimin käytössä.

Työssä esitetyt ongelmat ja niiden vaikutus laatuun on toimeksiantajaorganisaatiossa tunnistettu ja työ prosessin parantamiseksi sekä ongelmien ratkaisemiseksi jatkuu edelleen.

Avainsanat: Ketterä ohjelmistokehitys, SCRUM, Ohjelmistotestaus, testauksenhallinta, tutkiva testaus

Table of contents

LIST OF TERMS AND ABBREVIATIONS

Conditions of Satisfaction

Conditions of Satisfaction is a User Story-specific list of terms and conditions. The list is used in sprint review to determine the User Story's done-status.

Definition of Done Definition of Done (or DoD) is a checklist of activities that add value to the product under development. With DoD, Scrum team can verify that the delivering features are truly finalized. (Panchal 2008)

DSDM DSDM (Dynamic Systems Development Method) is an iterative, incremental software development model. DSDM is one of the many agile software development models.

Exploratory Testing

Exploratory testing or ad-hoc-testing is a form of manual testing. In Exploratory Testing, the test engineer does not follow a predefined script but with his creativeness and experience searches the program for defects.

Extreme Programming

Extreme programming (or XP) is an agile software development model which consists on numerous practices known as XP Values.

Feature-Driven Development

Like Scrum or DSDM, Feature Driven Development is one of the iterative and incremental software development models.

Requirement Requirement (in S60) is a concept used by the S60 platform to define the new functionality and features for the upcoming products based on S60 platform

S60 Is a mobile software platform using Symbian OS. It is developed and licensed by Nokia.

Scrum Scrum is an iterative and incremental agile software development model. Its modern version was developed mainly by Ken Schwaber and Jeff Sutherland and it's one of the most popular agile development models.

Session-Based Test Management	
	Session-Based Test Management is a software management method developed by Jonathan and James Bach to especially manage Exploratory Testing.
User Story	User Stories are used in some agile methods to present software requirements. A typical user story consists on few sentences and it describes the designed feature in general and understandable way. The S60 development team uses the User Stories to further specify the Requirements set by the S60 platform.

1 PROLOGUE

This thesis is commissioned by Nokia Corporation mobile software development team in which I work as a test engineer. Our current program has severe quality related challenges inherited from older projects; refactoring, implementing new features, errors, old test assets and constant rush create a very demanding environment for quality management. Working with Scrum – an agile software development method – gives us new ways to tackle the constantly changing demands and qualifications but also brings along new issues to solve.

One of the issues at hand is test management. The nature of our product requires a lot of manual testing and just managing a traditional script-based test asset requires a significant amount of work. While Scrum is widely used at Nokia, there is no standard on how the software testing should be managed within the teams. Similarly, there is no Scrum team –centric tool for test planning and managing available. Lacking the proper tools, the process has insufficient visibility on what is the actual plan for testing and how the test process proceeds.

Exploratory testing and session-based test management have been introduced to our program, but full utilization of these methods has not yet been achieved. Hands-on experience has shown that new tools or methods are needed so that the testing effort can be more agile and bring more value to the whole development process. The theoretical part of this thesis will explain the basics of the agile methods mentioned, clarify our working environment as well as the use of different agile methodologies in our team.

The final aim of this thesis is to improve the flow of information in the Scrum team and give the team a framework for planning, managing and executing test effort so that the output is usable by the team and the higher level management. Through a practical study, I compare different tools for managing software testing and evaluate their usability for Scrum team work. The prospective output of this study is my proposition about the most efficient and usable test environment considering our ways of working.

A reader of this thesis is recommended to have a basic knowledge about agile methodologies and software testing.

2 AGILE & SCRUM

2.1 Agile

Agile methodologies continue to be a hot topic in software development. Growing number of industry leaders (Oivo 2008) embrace the values and new ways of working in order to gain more flexibility and ability to respond to the constant change so tenaciously present in software industry. Agile values have been under development from mid-80's but the basis of modern agile values lies in Agile Manifesto from 2001. Agile Manifesto – or Manifesto for Agile Software Development – is a set of values combined by a group of software industry experts in 2001. The manifesto is presented in Figure 1.

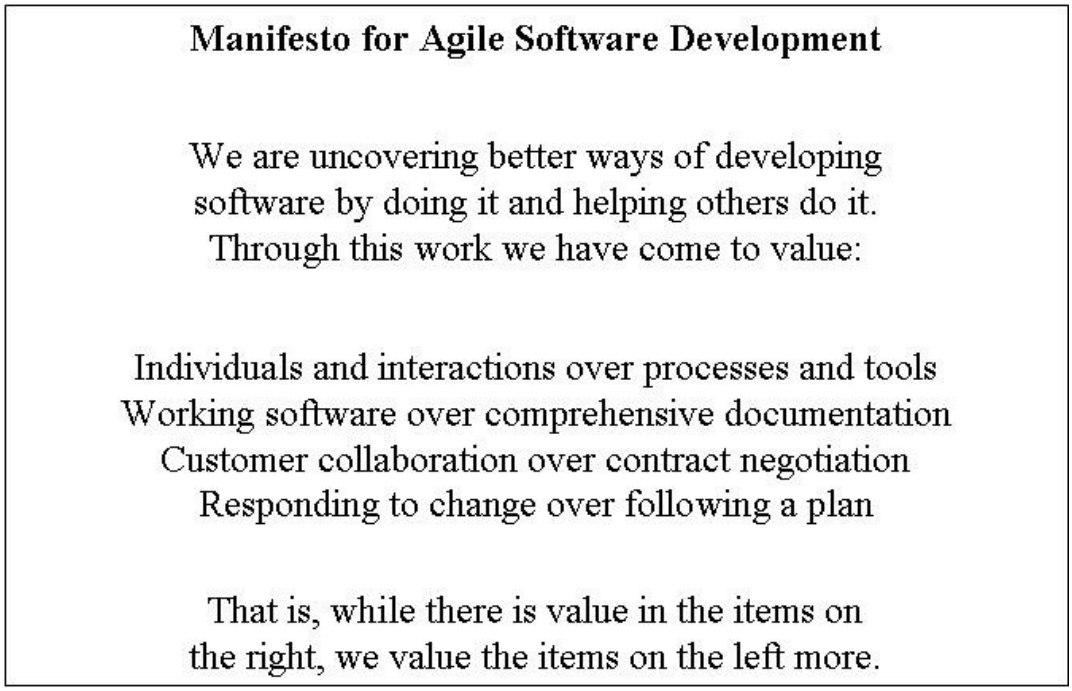


Figure 1. Manifesto for Agile Software Development. (Agile Alliance 2001)

Individuals and interactions over processes and tools.

The first of the four agile principles encourages software projects to give a high priority to the participating people in the product chain – as well as their way to interact, work and make decisions. It simply means that every decision related to the process should take into account the development personnel and the possible impact upon them. (Hazzan & Dubinsky 2008, 5)

Working software over comprehensive documentation.

Second principle of the Agile Manifesto states very firmly that the main goal in every software project should be a working, high-quality product. In practice, this means focusing on the development of the product and producing only the documents that are absolutely necessary for the project. Leading to earlier start on the actual development work, this approach also improves quality as the time previously used for extensive documentation can now be used to produce higher quality software. (Hazzan & Dubinsky 2008, 6)

Customer collaboration over contract negotiation.

Third of the Agile Manifesto's principles takes a novel look at the customer's role in the development process. It embraces constant interaction with the customer – giving the customer a chance to understand and survive the constant change present in the software projects. (Hazzan 2008 & Dubinsky, 7)

Responding to change over following a plan.

Last of the Agile Manifesto's principles underlines the meaning of a working process. A working software development process can cope to the emerging changes: such as changing customer requirements. The agility of the process helps to introduce these mostly unavoidable changes without necessarily increasing the final cost of the whole development process. (Hazzan & Dubinsky 2008, 7)

Agile is all about the ability to respond to the constant change. Being incremental, iterative, co-operative, straightforward and adaptive, agile methods bring different approaches to vastly changing, turbulent and complex processes that are typical in modern software development. There are several different methodologies that share the core principles mentioned above: Extreme Programming, SCRUM, DSDM, Feature-Driven Development and many others. (Hazzan & Dubinsky 2008, 5) One of the most popular methods today is SCRUM, used for example by Philips and F-Secure (Oivo 2008) as well as Nokia.

2.2 SCRUM

Modern day Scrum dates back to 1993 when Jeff Sutherland created the process borrowing the terminology of high-performing, cross-functional “scrum”-teams from 1986 Harvard study by Hirotaka Takeuchi and Ikujiro Nonaka. On 1995, Ken Schwaber took a step forward and in co-operation with Sutherland, formalized the process in the first published Scrum-whitepaper at OOPSLA 1995. In 2001, Schwaber and Mike Beedle published a book “Agile Software Development with SCRUM”. (Scrum Alliance Inc. 2009)

Nowadays Scrum is one of the most popular agile development methodologies, used worldwide in projects off all sizes. Its popularity is probably best explained with its simplicity – the core practice itself defines only three roles, two essential documentaries and a three-tier iteration process. Yet, Scrum might be easy to understand but mastering it can be very difficult. (Oivo 2008) Scrum is often combined with other agile practices. For example at Nokia Extreme Programming is not used as such, but many of the 12 core principles – like the planning game or pair programming – are adapted and used widely by individual Scrum teams. (Kontio 2008)

According to Scrum Alliance Inc (2009), Scrum defines three roles: a product owner, a scrum master and the scrum team. Product owner is responsible of the value of the product – ultimately the decision about what is important and what is not is one for the product owner to make. The product owner defines the product backlog solely and is officially responsible of the project. (Schwaber & Beedle 2002, 34)

Scrum team as ideal is a self-organized, cross-functional team of experts, who together can provide a working piece of software after each of the iterations. In Scrum, these iterations are called Sprints. Sprint is a short period of time, lasting often from one to four weeks. Before a sprint, the scrum team meets in a sprint planning session where the product backlog items are evaluated and the sprint backlog is formed from the evaluated items. The Scrum Master is responsible for the functionality of the team. Should there be any impediments or problems; the team takes them to the scrum master who makes sure that these impediments are removed. (Schwaber & Beedle 2002, 36-37, 47-48)

Scrum defines also two backlogs, which are the two most essential documents in Scrum. The product backlog is a defined set of features of the final program. The product owner evaluates the list and defines the order or value of each item, thus defining the order in which the items are done by the scrum team. The sprint backlog defines the items that the scrum team has chosen to work on during that particular sprint. Figure 2 displays the lifecycle of Scrum-implementation.

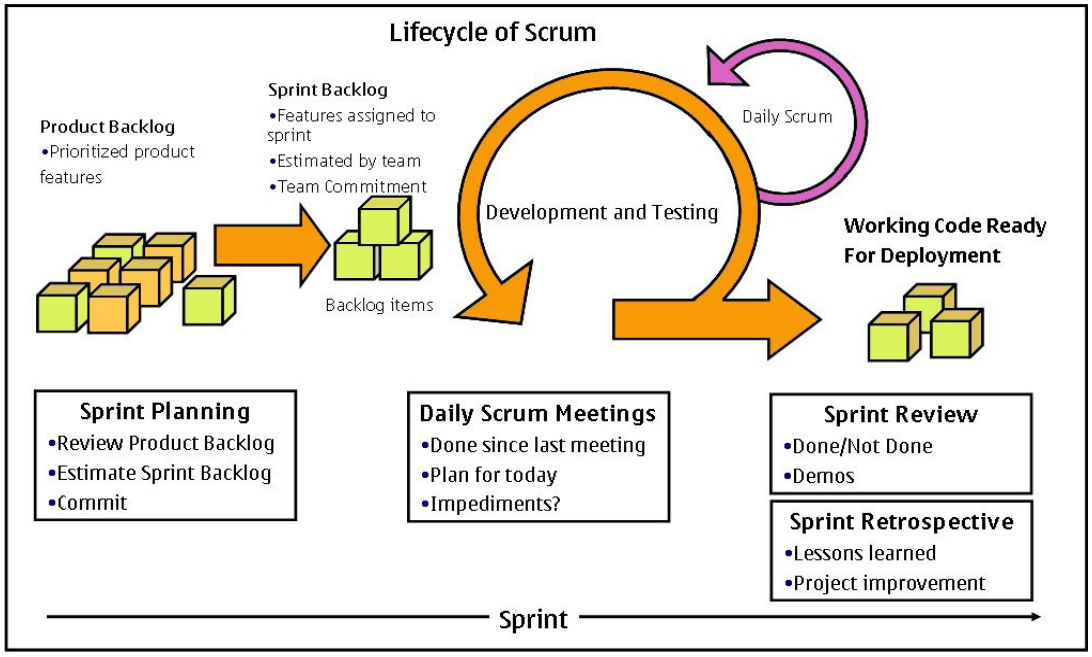


Figure 2. Scrum Lifecycle. (Retold from Battue Inc 2006)

After a sprint, the team, product owner, customers and others meet on a Sprint Review to evaluate what the team has accomplished during the Sprint. Teams usually hold demonstrations about the working code or use some other way to display what they have been able to produce during the last sprint. (Schwaber & Beedle 2002, 54-56) In Sprint Retrospective, team and Scrum Master together evaluate the last sprint and bring up any issues regarding for example the agility of the team.

Using scrum in large corporations often leads to a number of Scrum teams working on the same product. With these large programs, a common approach is to use a practice called “Scrum of Scrums”. In Scrum of Scrums, the scrum masters from each of the teams form their own scrum meetings, in which they take notice the more extensive, program-wide issues. (Kontio 2008)

2.3 Agile methods and software testing

Agile methods promote the meaning of built-in quality. When traditional software development processes tend to split the development process to different stages (like designing, developing and then testing, as for example in the traditional waterfall model) and assigning different tasks to teams responsible of every stage, agile methodologies take a different approach to the roles of the working team. In an ideal agile project responsibility is not transferred step by step in the production chain – instead, all the team members are responsible for the quality of the product. (Hazzan & Dubinsky 2008, 117-118)

One of the means to produce built-in quality is Test-Driven Development. Scott Mark (2007) wrote: "Agile methods move quality assurance upstream in the software development process and the most relevant of these methods is the principle of test-driven development (TDD)." TDD is a radically different approach to software testing and quality assurance: it implements executable unit-level test cases before the actual functional components. The basic idea is that the test case propels the implementation of the functional component – therefore the TDD test cases are authored and ran by the developer. (Mark 2007, 206 – 207)

But, if the case is like Hazzan & Dubinsky state: "quality refers to the entire team during the entire process of software development" (Hazzan & Dubinsky 2008, 118), what kind of role does agile software development give to the test engineer? Pettichord (2004) states that in agile, testing acts as the headlights of the project, providing information of the current status of the software to the development team. When the traditional waterfall model placed the test engineer to the end of the product chain, in agile test engineer takes an informative role, helping the rest of the team to achieve the common quality target during the whole project. As Pettichord sums up: "Testing provides information about the status of software under development to inform decisions." (Pettichord 2004)

So, regardless of the emphasis on Test-Driven Development and test automation, transferring to agile does not necessarily mean that the project will have to automate all of the testing. Only the parts that are reasonable to automate and where the test automation can bring more value to the development process (Duarte 2008). The role of

the test engineer remains, but there are now new tasks and methods for the testing work. One of the methods used in agile testing is Exploratory Testing (or ET).

Exploratory Testing

"*Exploratory testing is simultaneous learning, test design, and test execution." (Bach 2000)*

The quote above is from an article in which James Bach defines Exploratory Testing to be a brain powered, question based ad hoc-process that promotes continuous learning, test design and planning during the actual testing process.

According to Bach ET is among the most powerful solutions when test engineers are not exactly sure what has to be tested, when test cases are not necessarily fully ready or when test results are needed very quickly. It is also an ideal approach when test engineers have to learn the software under test quickly and when the testing needs to have some diversity from the scripted test execution (Bach 2002). A study conducted at Helsinki University of Technology in 2007 even suggests that when comparing ET and test case based test execution, ET produces more accurate results (less faulty defect reports) significantly faster (Itkonen, Mäntylä and Lassenius 2007).

ET is a technique that gives more power to the test engineer and relies more on his abilities to discover what is hidden deep in the software – rather than to just follow a guideline of a test case. The test engineer has the control over the test situation and it is up to his proficiency to attain the needed information about the current maturity of the software under test. (Bach 2002)

The basic idea of ET is quite simple: over time, the test engineer uses the product under test to achieve the predefined testing target without exact test steps and reports the findings. Usually the targets for a test session come from some kind of charter (Bach 2002), but even XP-style User Stories can be used as targets for ET.

Explaining ET is easy, but the real challenge is trying to understand and explain the process that goes in the mind of a test engineer doing ET. James Bach has defined five basic elements to consider when trying to understand the core of ET: Test Design, Careful Observation, Critical Thinking, Diverse Ideas and Rich Resources. (Bach 2002) This "inner structure" of ET requires skill from the test engineer and gives him whole new roles in addition to just executing the predefined test cases. First, the ET test engineer becomes a test designer – it is now the test engineer himself who must find the ways to explore the product systematically. Careful observation of all unusual behavior and critical thinking are essential characteristics of a good ET engineer – as are the ability to produce diverse, new ideas and capability to gather essential test resources like information about test data, different test environments or test tools. (Bach 2002)

Session-Based Test Management

Overall, ET can be a very efficient way to cover the gaps left by scripted testing. It gives the test engineer a lot of responsibility and thus requires very professional testing personnel. For test management, ET sets new requirements considering the output and follow-up of the whole testing process.

Because ET is a very quickly re-focusing process that uses significantly less time on documentation compared to most traditional models, it can be challenging to manage. Test management and test engineers might face new problems for example on how to track the overall maturity of the software or progress of a single tester. These issues do not limit only to the test engineer or test management – the new practices will affect the interaction with the developers and possible higher level management as well. Fluent bug tracing is essential to quality and summarizing the software maturity on a legible form is often required in order to keep trace on the overall progress of the whole project. (Bach 2008)

Maintaining a fluent flow of information is essential, but how to communicate between all the groups without interrupting the flexibility of the testing teams? Jonathan and James Bach, software testing specialists who have done a lot of work with agile methods and ET, have developed a new approach for managing ET. This Session-Based

Test Management (or SBTM) seeks to tackle the problems that testing team faces when starting ET. (Bach 2000)

SBTM uses a session as a basic unit of work. According to Jonathan Bach, a session in SBTM is an “uninterrupted block of reviewable, chartered test effort” (Bach 2000). In other words, a session must not be interrupted by phone calls or emails; it must have a predefined goal about what to test and what kind of defects to search and it produces an output – session sheet – for test management about what has happened during the session. (Bach 2000)

Each session is followed by session debriefing. Debriefing is a short meeting in which the test engineer explains the session report to the test manager – who also has an opportunity to give feedback or coaching to the test engineer. At best, session debriefings give the management a good understanding of the effort involved in the test cycle and thus help to predict the amount of time needed for testing without a detailed test plan. (Bach 2000)

Overall, SBTM adds more accountability to ET. By adding some framework like predefined goals and required documentation, the dynamic elements of ET do not get out of hand. Considering traditional test case based test execution, sessions take a lot less time to setup and leave much less legacy than a suite of test cases – without a strict test suite, the test team can also achieve a whole new level in flexibility. (Kalman 2007)

While saving time on test planning, project setup and so on, SBTM can also be a very time-consuming approach. Test engineers do not produce good sessions by default – it takes time and effort to learn what is a reasonable way to execute and plan a session. The reporting and analyzing the results are another time consuming elements, especially when starting to utilize SBTM. (Kalman 2007)

Sam Kalman (2007) suggests that rather than considering SBTM to be “a practice”, it should be regarded as a bundle of different practices, or components. A project team considering SBTM can review and compare these components and use only the ones they think are suitable for their environment. This approach is called SBT Lite. (Kalman 2007)

3 WORKING ENVIRONMENT

3.1 Our organization in Nokia

This thesis was commissioned by Nokia and was partly done besides my daily activities as a test engineer. I work in a development team that does messaging-related work for S60 product platform utilizing the agile principles and above all, Scrum. Personnel in our program consist of several Scrum teams working on development, testing and integration of new features as well as maturization of the existing software. In addition to the scrum teams, our program also has personnel in management and support tasks.

Requirements

The Scrum teams work independently, but all the work in our program is in the end done to achieve one common goal – meeting the requirements. Requirement is a concept used by the S60 platform to define the new functionality and features for the upcoming products based on S60 platform. Requirements provide information for different interest groups about the dependencies of planned functional and non-functional requirements and the planned time scale of their implementation. Requirements are also used to prioritize the implementation of planned functionality for different platforms and actual products. For example, the S60 3rd Edition Feature Pack 2 contains several requirements working as a product backlog for one release.

Requirements are usually planned for a specific product or S60 platform release. In this context, release means a package offered to different kinds of clients: such as mobile operators or external software developers. For example different SDK's released for Forum Nokia-developers are customer releases.

A development program manages and supervises the status of their requirements. When a requirement is considered to be ready for the product, it is inspected and approved by the platform it is targeted to. Before being accepted as a part of a release of a specific product, requirement has to pass a number of quality gates. Gates work as quality checkpoints and to pass, the program must be able to prove that the requirement meets the quality gate criteria – for example by providing sufficient test results for the requirement. Lifecycle of the requirement is displayed in Figure 3.

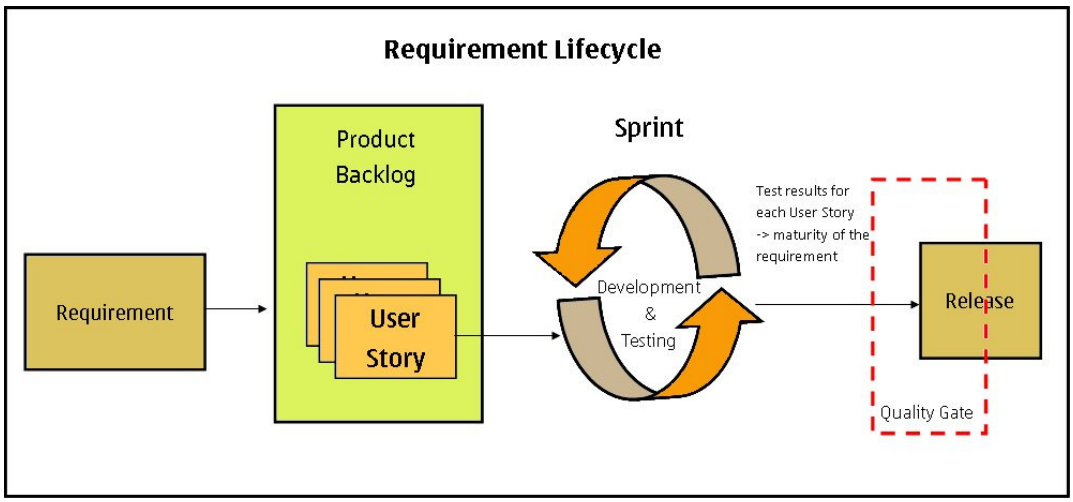


Figure 3. Lifecycle of a Requirement

Lifecycle of a requirement starts in the definition. After definition a requirement is divided to multitude of user stories to the product backlog. The user stories are taken from the product backlog to the sprints (sprint backlogs) for implementation. During the sprint the new implementation is tested and test results of these user stories are combined to present the maturity of a requirement. After the sprint, the user story can be marked as done if it meets the criteria set for the done-status. When all the user stories of a single requirement are done, the maturity of the requirement is compared against the quality gate criteria. If the maturity of the requirement is acceptable, the requirement can be added to a release.

3.2 Agile in our organization

Nowadays, agile methods are widely used at Nokia. The company has its own agile trainers and experts, but there is no common standard on how a specific program or team should utilize the different methodologies – different organizations decide by themselves how to be Agile. In our organization, the main method is Scrum – we organize ourselves and our work according to the Scrum principles but for example our way to divide requirements to User Stories is an adaptation from Extreme Programming. When a requirement is taken to development, it is divided to multiple User Stories and added to the Product Backlog which is common for the whole program. In each bi-weekly Sprint planning, a User Story is taken to one of the team-specific Sprint Backlogs. The nature of user stories can vary – they can involve

implementation and testing of a new functionality, bug fixes, platformization or program-wide testing tasks.

User Story

User Story acts as a basic unit of work in our Scrum Teams. It is typically an end-user view to a new functionality but, as mentioned before, user stories can also relate to maturization of existing software like bug fixes or to program-wide testing. User Story description should be as concrete and practical as possible, so the team could easily understand its goal and what tasks are needed to get the User Story done. For instance, User Story for using a search engine could be: "As a user, I am able to enter characters in the search field."

The requirements are divided to user stories by product owners. Product owners also rate and prioritize the user stories in the product backlog. Local product owners also communicate with the scrum teams about the User Story content and of the tasks related to the user stories. User Story is divided to tasks in the sprint planning and it is very common that a single User Story has tasks involving development and testing. For example when implementing a bug fix, developer has a task related to studying and solving the problem. The test engineer of the team then verifies the fix and if both tasks are ready and acceptable when the sprint ends, the User Story can be marked as done in the sprint review.

3.3 Testing in our team

Before user story can be marked as done or even before any code can be released to the unified database, it has to pass the quality criteria set for that specific user story. This criterion – Test Objective – is a user story specific list of testing tasks, which should be collected up in sprint planning. Figure 4 shows the lifecycle of a User story from the test engineer's point of view.

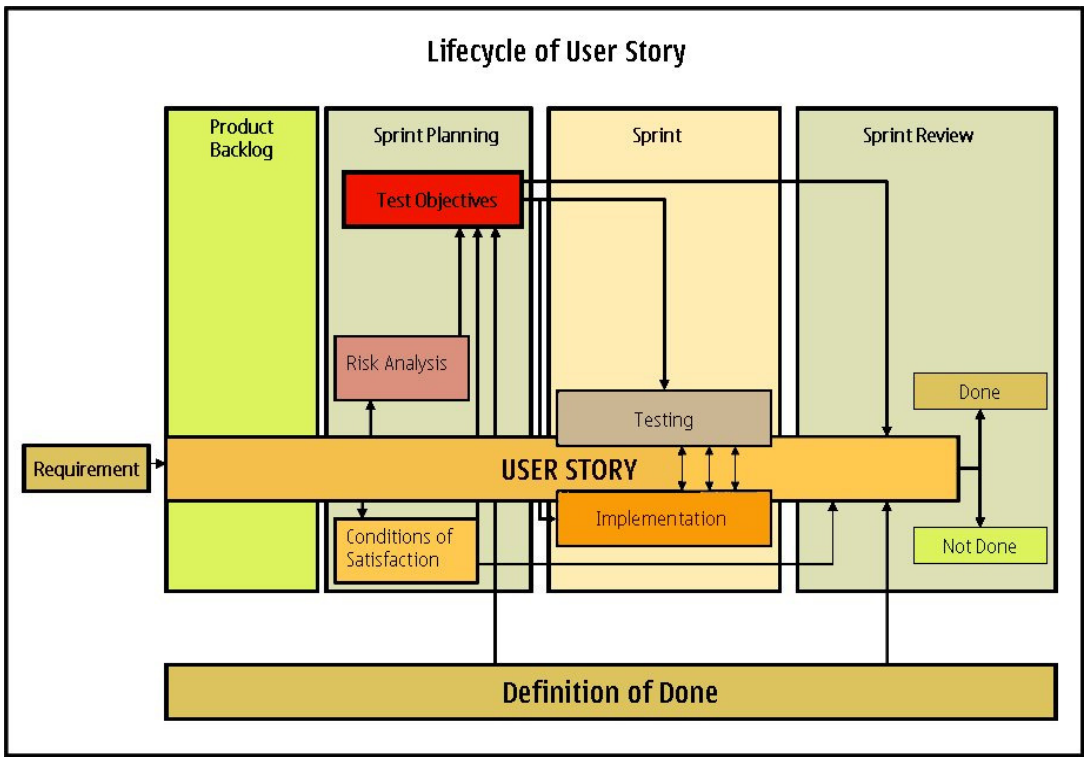


Figure 4. Lifecycle of a User Story

As mentioned before, User Stories are formed to the product backlog from the requirement database. In sprint planning, the User Story specific Conditions of Satisfaction are defined by product owner and the User Story risk analysis is done in co-operation of product owner and the scrum team. Considering the CoS, the Risk Analysis and the product wide Definition of Done (or DoD), the team and product owner agree about the tasks related to the testing and verification of this specific User Story – the Test Objectives.

Test objectives have a direct impact on the Scrum team's testing and implementation. Both the tester and the developer involved in User Story have to take the task list into consideration. For example, the test objective list can contain very specific information about the nature of testing – is it more non-functional or does the new implementation involve a high risk of regression (disintegration of existing, tested functionality).

DoD and CoS are also some of the key elements in the Sprint Review –meeting that is held after a Sprint. At the Sprint review the product owner should use these predefined targets as a guideline when he decides whether a user story is done or not.

Testing in Scrum Teams

Actual test engineer duties in our organization can be roughly divided into two sections: To the team-specific testing and to the program-wide testing tasks. The program-wide testing tasks involve the maintenance of test assets, performing test runs and reporting the results of the functional, regression and non-functional testing done for the entire program.

As stated in our DoD, the Scrum teams are responsible for their own testing. In practice, this leads to the fact that the quality of the program or the responsibility of it does not lie on the shoulders of one solid test manager, but that every team is in liability for the quality of their own work as well as its effects on the program's joint code line. For the Scrum teams this means that before any new functionality, bug fixes etc. can be added to the joint code line, they must be tested in developers custom builds by the team's test engineer. Having specific test engineers in teams is a departure from the traditional Scrum model and the idea of universal team roles, but it occurs from the specific nature of our product and especially from the need for wide testing focusing on the end-user point of view.

Testing tools

The testing tasks in teams vary quite a lot and so do the tools used in testing. For tracking of the requirement status, we have the test assets in HP Quality Center. Running these assets generates metrics about the program-wide run rates and pass rates used for example in the Quality Gate reviews. Program-wide testing tasks usually involve use of these assets and running the pre-defined test sets to track for example the regression in the joint code line.

The testing regarding new implementations is done by Exploratory Testing. Tracking of Exploratory Testing is currently partly unspecified and almost every test engineer has his own way to utilize Excel-spreadsheets, wiki pages and other similar tools to control their own work. As mentioned before, we try to avoid the model of a one test manager – ideally, a test engineer's "test manager" is his Scrum team and the primary way to communicate defects is to go and have a word with the developer who is responsible for that specific action. In addition to the Scrum team interaction, our test engineers meet in recurrent bi-weekly sessions to share the knowledge of the general issues concerning the

product quality and issues that might affect test engineer work. This model of constant interaction and discussion has been identified to have some features of the Session Based Test management described in Chapter 2.

3.4 Issues in Agile Quality Practices

Agile theories might make the "ideal" process to sound very easy. Preparing and adapting to constant change, considering quality issues through the development and focusing on the working product instead of specifications and documentations sounds very reasonable.

During this study I have had a chance to see a lot of changes in a very short period of time. Changes in my team, changes in the product we are making, changes in the way we are working and even some major changes in the whole economical climate around our business: and still, despite the fact that our software development is agile, all of the changes have brought along a lot of problems.

When I started this study, I thought that the most critical problems regarding our testing process were in the test execution: slow environment, bad test cases, unknown product which had a lot of defects inherited from other projects and so on. During the study it became quite clear that more severe problems lie in our ways to plan testing. In Chapter 3.2 I introduced our theoretical way of doing test planning: setting test objectives beforehand and using them to plan testing and development of a certain User Story – and after sprint using the objectives alongside Definition of Done to determine if the User Story is done or not.

In practice, use of test objectives, CoS or DoD varies a lot. Often in sprint planning the only question asked from test engineer is "how long will it take for you to test this?" without even determining what actually should be tested for the User Story to be done. In some situations, test plans are even being done for features that are already released some time ago. Reasons for the lack of test planning vary: lack of time being the most probable one. Still, the improper planning has generated situations where test engineers do not know actually what to test, in what scale the testing should be done or what is the required output of the current test task.

Another issue strongly related to test planning is the deprivations in the flow of information. Tightening up test planning in prior to development process is obviously not an answer – it is not agile and in most cases the possible effects of a new implementation become known just *during* the development process. At this point, communication between test and development engineers plays a vital role: the developer responsible of a specific implementation is the most likely to know about the regression risks, dependencies and other possible effects and thus the most logical direction of action when new defects are found.

It is clear that no mere test execution tool can help with these issues and in the light of this knowledge we decided to add some additional research to the study. I would conduct the comparison of the tools as originally planned, but after that there should be some consideration of the problems mentioned above. The original goals of the study are presented in Chapter 4 as well as the requirement specification for the tool. Chapter 5 presents the study of the tools and Chapter 6 consists of the conclusions of the original study and the further development ideas regarding the issues stated on this chapter.

4 QUALITY MANAGEMENT METHODS

4.1 Theoretical targets

4.1.1 Goal

The first time I came across test management -related problems was when I was working in my old team at Nokia as a summer trainee. The program was quite large, which reflected directly to the size of the test asset and the amount of produced test data. A significant amount of time was consumed in reporting and producing Excel-spreadsheets and Power Point -shows for different interest groups.

One goal of this thesis is to find ways to avoid situations stated above. If a test engineer has a mass of duties that require significant amount of time, all that time is eventually taken away from the most important task – the quality assurance of the program. I am not stating here that reporting the test results is a straight forward reason for the quality issues in software projects, but simply taking into consideration the amount of time used in different side activities and their effect on the actual testing work.

In Chapter 3 I introduced some of the required outputs of the test engineers in our organization and stated shortly some of the actual tools used when doing testing. The methods mentioned are quite different in nature and in practice. The HP Quality Center is the main test environment used in Nokia – it provides detailed test cases, which can be formed to test sets and linked forward to specific requirements, so that the requirement status can be viewed on exactitude of a test case. Quality Center (or QC)-cases are extremely detailed and they are very fit for example detailed functional testing or when test cases are created by one person and run by another, who does not necessarily have equal understanding of the target program.

The QC is very widely used; it runs on multiple servers and might have several hundreds of users simultaneously. This has lead to severe performance issues which reflect very often to our testing tasks. However, QC is our default database for the test work and transferring all the testing activities away from the QC-database is not possible at this moment. Instead, it probably would be wise to track down methods that allow the test engineer to do the testing "outside" the QC-environment and afterwards import the collected data to QC or other management-level solution – providing the necessary information and avoiding the time-consuming downtime at the same time. One way of doing this could be exploratory testing and reporting based on Excel-spreadsheets – a modification of SBTM presented in Chapter 2.

As already stated Exploratory Testing differs a lot from the scripted, test case-based testing done with QC. The main idea is that test engineer is responsible for planning, running and reporting the current testing task. It is reasonable to have some predefined information about where to concentrate the tests, but the test set must be able to yield when needed – for example if it looks like that some area needs to be tested more thoroughly than the other. The disadvantage of this kind of approach is that since every test engineer has his own way of doing things, it is very likely that there will very soon be several different Excel-spreadsheets, wiki pages or other places where the information is stored. Thus, compiling the needed reports and requirement coverage from these can actually be a lot more time-consuming than from the QC – the variation of the collected data may even affect the very transparency and reliability of the testing work.

The goal of this thesis is in a way to seek the medium of these working methods. Since we are agile and lot of the testing is done within the scrum teams, the method has to be agile, flexible and as transparent as possible – since the problems faced on one team could easily reproduce in other and without valid communication method it can generate a lot of double effort and even compromise the comprehensiveness of program-wide tasks. The tool should provide a way to use less time on reporting and more time on testing, and at the same time be easier to use than the current methods.

Flow of information

One of the major questions in test management and test environment tool is the fluent flow of information and whom the information may concern. In this thesis, the affected

parties in the lifecycle of a User Story are Requirement, Product, Team and the Task. The Figure 5 shows the information needed by each interest group and the ideal flow of information.

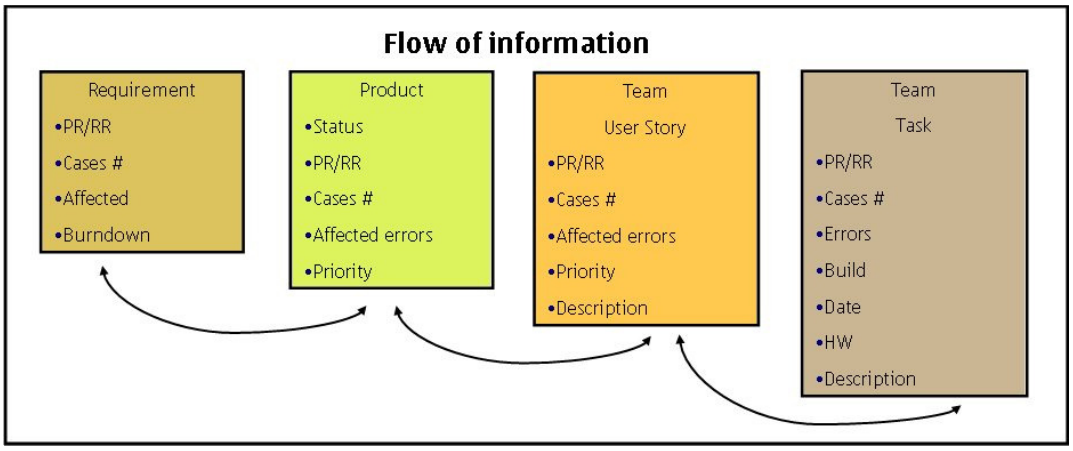


Figure 5. The Flow of Information.

The requirement tree displayed in Figure 5 is a four-tiered structure. It describes the flow of information between the tiers and all the information that each tier needs from the other. For example, the User Story in Product Backlog is extracted from a requirement. After this it is allocated for a team, placed to the sprint backlog and divided to tasks. The testing of a User Story generates information such as Run and Pass Rates, number of cases ran, the affected errors, the build which it was tested on etc. and the additional description from the test engineer. Some of the information should be available in Sprint review and yet again in Quality Gate reviews – the most important being the run and pass rates and affected errors.

Still referring to the Figure 5, there are some clear requirements towards the flow of information provided by the testing tool. It should provide only the needed information for each tier and such work as an information gatherer from all the lower tiers. This means that on the requirement tier there should be information gathered about the pass and run rates as well as the related errors of the *whole requirement*, or in other words all the errors existing in the User Stories related to that requirement.

One important aspect of the flow of information is the Description or Comment-information. Basically, it should provide a way for the test engineers, developers, product owners and scrum master to share information about the current and upcoming tasks.

4.1.2 Qualifications for the Quality Control Tool

Briefly, the goal of this thesis is to find ways for the scrum teams to communicate more efficiently about quality and to give the test engineers a better and more flexible framework for their work. To be able to actually test the available tools and methods, a test bench was formed. The test bench contains some requirements and user stories linked to those requirements. Some of the user stories are divided into two sprints and again into exploratory test-styled, general test cases. The test bench is designed to be only suggestive of our real test environment – the requirements, user stories and test cases seen in later screenshots are not from real backlogs.

For the comparison of the different tools, the most important non-functional and functional requirements were also listed. The requirements gave a sensible idea what to look for and what especially should be considered when testing and evaluating a specific tool. The key areas of the requirements are functionality, accessibility, usability, flexibility and scalability, informativity and price.

Functionality

For the test engineer, the tool should work as a way to keep track of the exploratory testing done. Using the tool, results of testing done during the sprint should be usable for example in Sprint Review without having to form different reports or charts from them.

For the Scrum team, the tool should work as a way to store, update and follow the test objectives. It also gives developers, test engineers and product owners a channel to update information about the User Story – for example what particular areas are reasonable to test. After Sprint, the tool should provide enough information whether the test objectives are met or not.

For management, the tool should provide information about the progress of a Requirement in User Story level. In addition, it would be good if some metrics about the software maturity could be easily generated from the tool.

Additional functional features could be for example compatibility with other products. This could mean interfaces for Quality Center or importing and exporting data to Excel or a Wiki page. At some point, it would also be reasonable if all the related errors from the current error database could be straightly linked to the tool.

Accessibility

The tool should provide access for test engineers, developers and managers – and from internal and external networks as well. A web based tool could probably be the most flexible in this sense, although a lot of common files and information is provided in web drives as well. When using a web based tool, usage of different browsers and perhaps even operating systems should be taken into account.

Usability and Reliability

Along accessibility, usability plays a key role within these requirements. It is important that the tool is easy and reliable to use. The tool should not suffer from long downtimes caused by technical support and maintaining it should be fast – by choice it should be done by the testing team itself.

Flexibility and Scalability

Objects in the tool should be easily revised, as test objectives may change during sprints. The tool should also be able to handle large amounts of data and scale to fit different sized programs.

Informativity

The user should be able to see the situation of a test objective or User Story without having to compile complex reports or charts. Information should be available at least on run and pass rates (or other form of the maturity of the software), affected errors and the description or team comments.

4.2 Quality Control tools that meet the specifications

4.2.1 Available tools

Since there is a significant amount of different sized software projects already working agile, it seems logical to think that there must already be some ready-made tools for the agile quality control. However, after spending few days searching the Internet I was left surprisingly empty-handed. It seemed like manual test engineers are a very rare sight in agile and even rarer is to have a particular tool to fit them in the agile team management. Agile testing tools such as FitNesse or Breeze (LoadStorm 2009) exist, but they are mainly meant for automated testing.

After giving it a thought, the lack of variation seemed reasonable. The open source and freeware tools are mainly meant for small projects with few developers – not for a multisite, international corporation with thousands of developers. The larger solutions, like the Rally Software's project management tools (Rally Software 2009) do look appealing in demos and overviews, but their pricing is an obstacle especially in the current economical situation.

The next step in the study was to turn to Nokia's own agile coaches and ask for their recommendations. The result was a few good wiki pages, interesting discussions and one reference to a tool – which later came out to be meant mainly for test automation as well. It turned out that agile coach's focus is on the agile means, not the methods to implement them in teams.

At this point, it was time to extend the scope of the search. Different project management tools, database solutions and more traditional testing tools became next in line of potential tools. Even if a product is not especially *made* for Agile, it might be possible to implement it to *fit* in it with reasonable effort.

4.2.2 Prioritization and limitations

The most important issue in question was without a doubt the timetable. It was very strict and limited the actual thesis work to about five months (from December to the end of April) – clearly insufficient time to study the subject and compare the tools in the widest possible scale. To tackle this, we limited the actual scope of this thesis to the Scrum team with a simple reference to each User Story's requirement and narrowed the amount of the methods to be studied to four. The relations of the reconfigured scope can be seen in Figure 6.

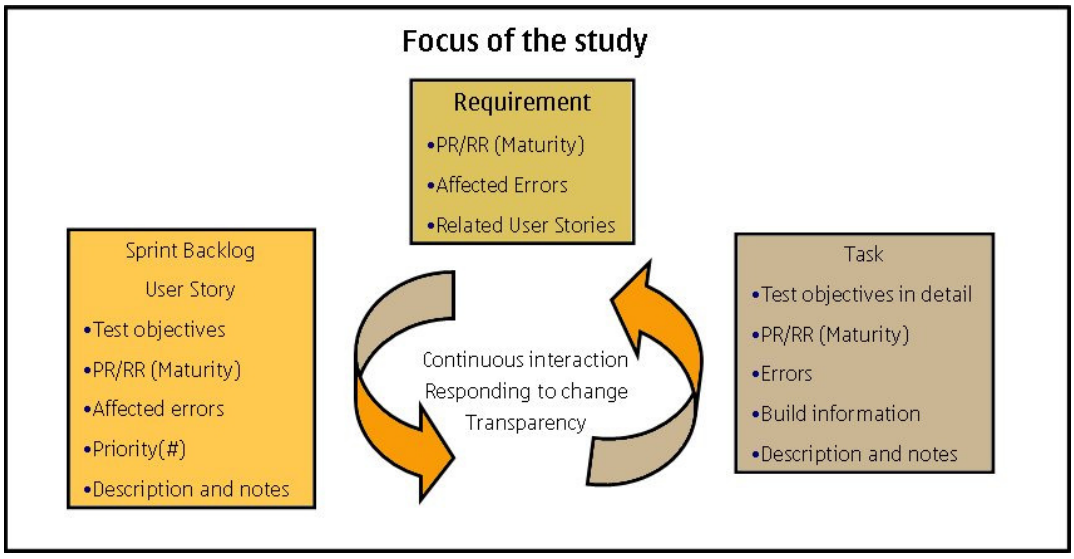


Figure 6. Focus of the Study.

In addition to reconfiguring the scope of the study, there were some issues about what features the tool should support and what are not so important at this point. For example, all the currently existing data for test automation or from old user stories is not supported – since for example test automation has its own test management tools and if user stories are returned to the backlog and planned for another sprint, the test objectives can be reconfigured.

An additional, greater risk for the study involved the actual tools studied. What would happen if a proper tool is not found or the existing tools do not provide the needed functionality? Since the goal of the study was simply to evaluate and chart the tools available, this was a risk that just had to be taken. It is clear that our program will not be left without a test management solution – the question of what that the tool actually will be, may be left without an answer in this study. In that case, the outcome of the study

should cover further development ideas and opinions about what should we focus on next.

4.2.3 Quality Control Tools in This Study

As mentioned in Chapter 4.2.1, the study became more of a question of implementation. I decided to choose the tools that emphasize the other requirements instead of the tools being strictly designed for agile teams. Usability, flexibility and transparency became the most important criteria and soon there were even options to select from. Many project management tools, test management tools and database solutions provide hierarchical structures to work with – very suitable for linking the requirements to the user stories. Many of these solutions were also freeware or even open source.

I ended up choosing two completely external tools – one database solution and one test management software. I addition, the Quality Center would work as a reference tool: it is a good example of a good tool in a wrong environment and gives a lot of hints about how things should not be done in a small development team. Alongside Quality Center, Excel-spreadsheets are already quite widely used in test planning and storing of test data. Being reconfigurable and well known, Excel-spreadsheets might work inside the Scrum team quite well but it is still a bit unclear if they can provide enough ways to display data comprehensively without an extensive need to rework it.

The first 3rd party solution, Dabble DB, is a very interesting database solution tool, which resembles Excel quite a lot. Web-based user interface sounds very tempting compared to the network drives used with Excel and the demo video on the program's web page shows a lot of interesting features definitely worth trying. The other 3rd party tool, Test Run, is a new and a fresh look of a traditional test management tool. While it still is in development phase, the developer's idea of it being more than a tool designed just for a test engineer makes it a very promising candidate.

5 QUALITY CONTROL TOOLS IN STUDY

During the testing of the tools, I marked down the main features, advantages and disadvantages I found during the testing process. The list of the more detailed findings relatively to the main requirements set in Chapter 4.1.2 can be found from Appendix 1. In this chapter I present the tools studied from the test engineer's point of view and introduce some of the best features as well as the most significant weaknesses of each one.

5.1 HP Quality Center

HP Quality Center is a web based test management tool used widely at Nokia. It supports test cases with very detailed test steps and planning the testing through requirements as well as test plans. Especially when used in very large test assets like extensive language variant, functional or regression testing, Quality Center provides an effective way to plan and monitor the test asset and software quality. The test cases can be linked to a certain test set and requirement, so that all the runs done to those cases show as passes or fails in requirement-view.

Administration of the test asset is easy, but utilizing the results takes time. Quality Center does not generate reports of run rates or percentages, so if any charts or other condensed information is needed it must first be exported for example to Excel and then treated further. Quality Center is neither a particularly fast tool to learn nor simple to use. The test cases and the complex hierarchy are too detailed and the whole infrastructure of the program is too complex for the Scrum team. As shown in Figure 7, the Quality Center's way to display test cases is highly detailed and does not support any kind of compiled way to display the test case data.

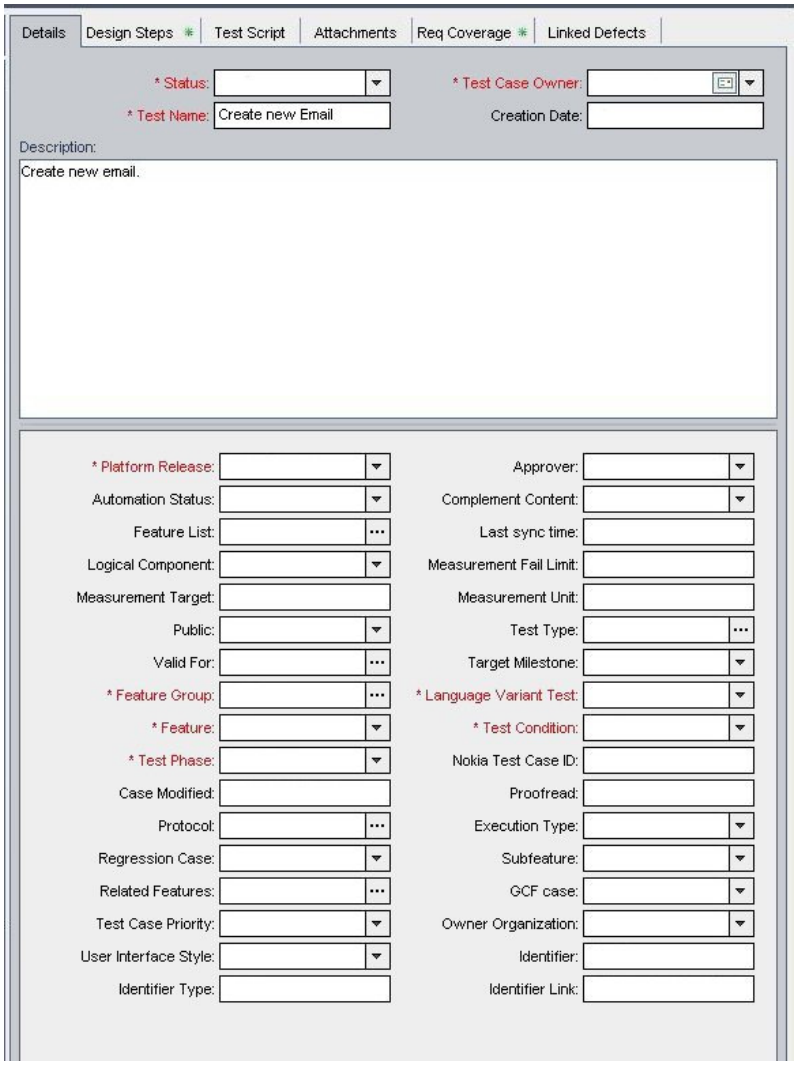


Figure 7: Test Case in Quality Center. (Quality Center 2009)

Due to this complexity it is not likely that there will be many developers who want to use Quality Center, even if it helps out test engineers. It is also reasonable to question the necessity of keeping a small ET-asset in a tool this complex. Even if the test cases could be used as user stories, the constant performance issues and downtimes related to Quality Center are such an impediment that using it for ET-purposes is not really advisable.

To summarize, Quality Center is a powerful tool for large test assets or teams. It contains a significant amount of features for planning and managing test work but offers very few ways to collect or represent data without using external tools. Extensive function set also leads to complexity, which is by far Quality Center's greatest vice when considering possible implementation to agile development work.

5.2 Excel – spreadsheets

Microsoft Excel is probably the most widely used spreadsheet application available. It features a range of functionality from calculations to macro programming – and has applications in numerous parts of a software project. Spreadsheet is a very versatile tool. Basically, it can be configured to meet almost any functional requirement set for a test management tool. For this comparison, I used a configuration that had Requirements and User Stories on one spreadsheet and Scrum team sprints on the other – between them was a link that provided a way to follow maturisation of a requirement through its user stories. This hierarchy is quite similar to the requirement tree presented in Chapter 4.1.1.

Since spreadsheets provide such a wide range of functionality, the initial configuration might take a lot of time. The basics can be achieved very easily, but advanced maintenance and for example utilizing the macros can require a lot of effort and a skilled user to configure them. However, an investment made as working hours at this point could pay off multiple times the investment – compared for example to a separately purchased 3rd party tool. Maintenance of spreadsheets is also usually much smaller of an issue than with complex web based tools administrated by other parties.

Spreadsheet is a good platform for Scrum work. Once it is properly configured, a shared spreadsheet supports multiple users and updating is easy. Since basically everyone has used spreadsheets before, a new spreadsheet should be easy to introduce and the efficient usage is up to the team. The team must commit to the updating and cleaning up of old data – otherwise the spreadsheet can just end up to be a great pile of old, useless information.

Information stored in spreadsheets is very reusable. Excel itself provides a number of ways to sort, link and compile information to graphs – which then can be used again in documents or wiki pages. With macros, spreadsheets can be linked to 3rd party programs, for example an error database or other test management tool like QC.

Overall, spreadsheets are an excellent tool for managing a small or medium sized test asset. Reconfigurable spreadsheets provide all the required functionality of our program. Being adaptable, they also are good choice for the agile team; in our case it could scale up from being a tool for a single team to gather efforts from all the scrum teams together in one file. Possible problems in informativity or usability caused by a vast amount of data can be dealt with proper administration, but it can take a lot of time. Spreadsheets are also time consuming to configure, but once configured they are very usable and reliable.

5.3 Dabble DB – database solution

The Dabble DB is a web-based database application. From sales to project management, Dabble markets itself to be restructurable and redefinable content management application (Dabble DB 2009).

In this study a free public Dabble-database account was created. The basic approach is quite similar to Excel, even data importing from spreadsheets is possible. The three categories mark the three tiers: Requirements, User Story and the Sprint level. Sprint level gets the User Stories assigned to that sprint from the User Story-category. In Sprint-category, User Stories are divided into cases representing ET goals – their combined status is show in the main view as shown in Figure 8. User Stories are then linked to the Requirement-category which shows all the requirements and links to User Stories related to them.

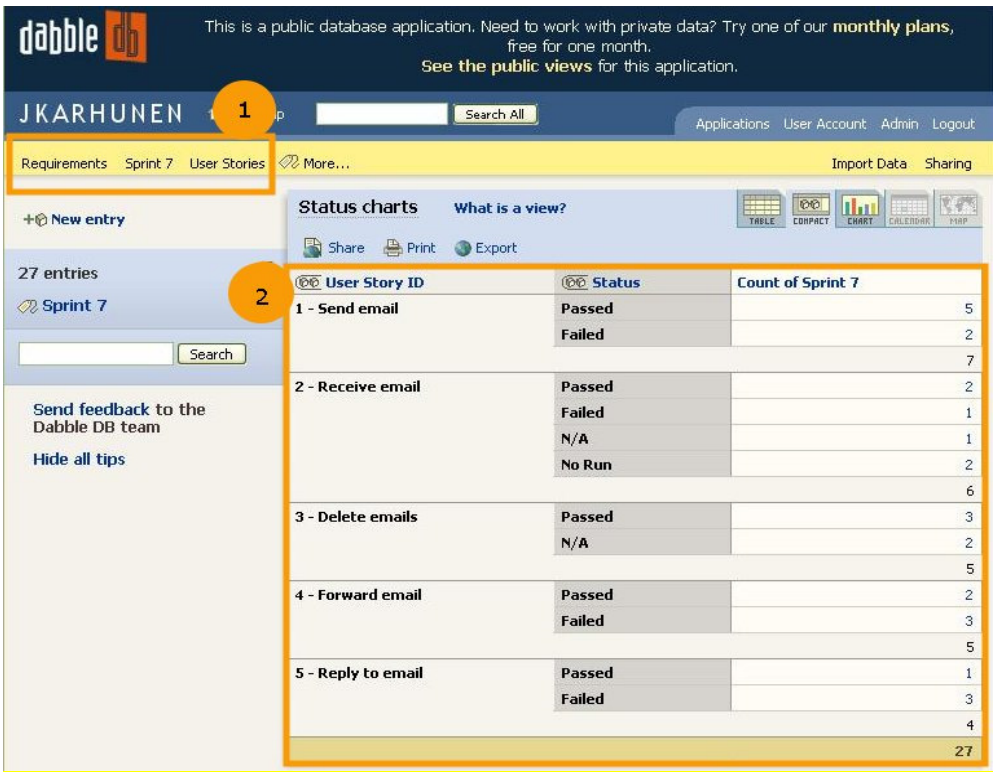


Figure 8. User Stories assigned to a single Sprint. 1: Available categories. 2: User Story name and status by test cases. (Dabble DB 2009)

On the first look, Dabble DB feels extremely convenient. The web-interface is lightweight and usable with multiple browsers, application supports multiple users and the administration of different user levels is very easy. The most promising features are the views. In every category there is a possibility to define multiple different views – ways to represent the data in the category. For example, in Sprint category there are two views at the moment: one that shows very detailed information about every User Story, names and pass/fail information about every test case like in Figure 9, which shows the more detailed view of the same shown in Figure 8. In this view, test cases can be modified and for example info about related errors can be added to test case information.

Figure 9. Detailed view of a single Sprint. 1: Preconfigured views. (Dabble DB 2009)

Figure 9 shows also another good feature of Dabble DB – the preconfigured views. Dabble can automatically make a table view more compact and display the data of a table in a chart. While the possibility to general graphical views is a good feature, the charts created by Dabble are quite insufficient. User cannot modify the charts and the basic charts do not actually tell that much about the maturity of the software or the status of a user story.

Dabble DB provides a convenient, powerful framework for handling numerical data. The categories can be easily used to create the same kind of hierarchy as in Excel-spreadsheets. Despite the detail, the web interface is lightweight and the basic functionality is easily achieved – after all it resembles Excel a lot. Although, this similarity can be a bit deceiving: while Dabble DB is strong in numerical information, it does not support compiling, comparing and gathering written data is not as widely as Excel. A good example is COUNTIF-formula which exists in Excel, but is not supported in Dabble DB. Due to this small difference, Dabble cannot count pass and run rates for test cases from their entry counts as efficient as in the Excel-spreadsheet.

Another significant lack in Dabble DB is a fast and easy way of commenting user stories or test cases. Like shown in Figure 10, the interface for modifying the information is very complex and has almost as much available entries as the Quality Center's test cases. It's likely that adjusting to this kind of interface would take a bit time and this feature alone is enough to make Dabble somewhat too complex for a scrum team.

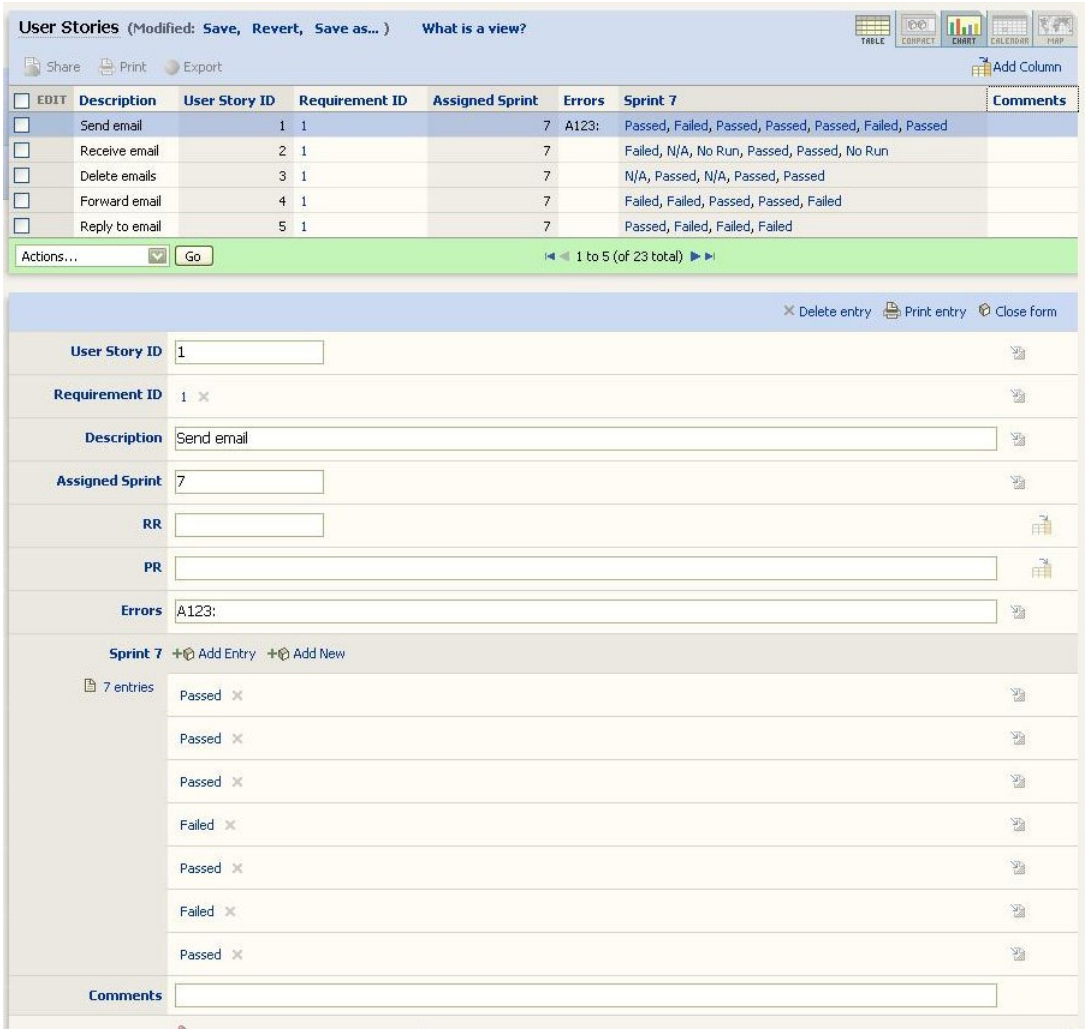


Figure 10. Modifying entry. (Dabble DB 2009)

To sum up, I would not recommend Dabble DB to be used as a tool in a scrum team. It does provide a good set of features, easy administration and easily adaptable basic functionality. However, the limitations on informativity and usability especially when being compared to Excel-spreadsheets are too significant for this to be a notable tool for a scrum team.

5.4 Test Run

Test Run is test management software developed by Byrne Reese. It is said to be developed not only for test engineers, but also for the project, program and product managers to understand the complete development lifecycle (Test Run 2008).

Test Run supports test cases, test plans and a mild hierarchy between them and the third tier, products. On this study, the product is treated as Requirement, test cases mark user stories and test plan collects the test cases as test objectives for a sprint. Figure 11 shows the overview of the active test plans. The overview shows the status of each test plan and allows user to import, export or edit test plan data.

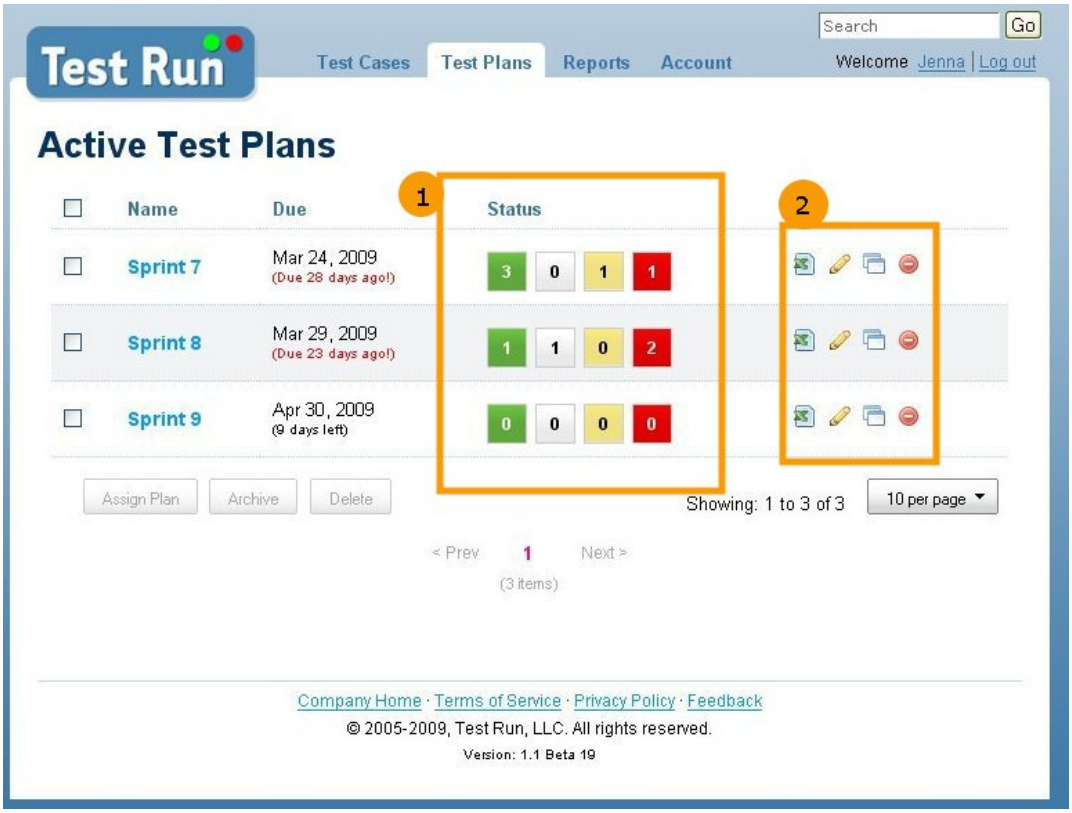


Figure 11. Test plans. 1: Status of a test plan (sprint). 2: Options for managing test plans. (Test Run 2009)

Like Dabble DB, Test Run has a simple, pleasant web-interface. It is lightweight to use and with data imports, it can use old test case data from for example spreadsheets. Also similar to Dabble DB, Test Run provides graphs and reports about the situation of test sets – unfortunately like in Dabble DB, the Test Run's promising graphical features are somewhat crippled by the lack of configurability.

For a Scrum team, Test Run offers a functional interface for distributing notes related to a single User Story. Figure 12 shows a user story with some added comments as well as how the basic test case description-field is used to show the ET-test cases.

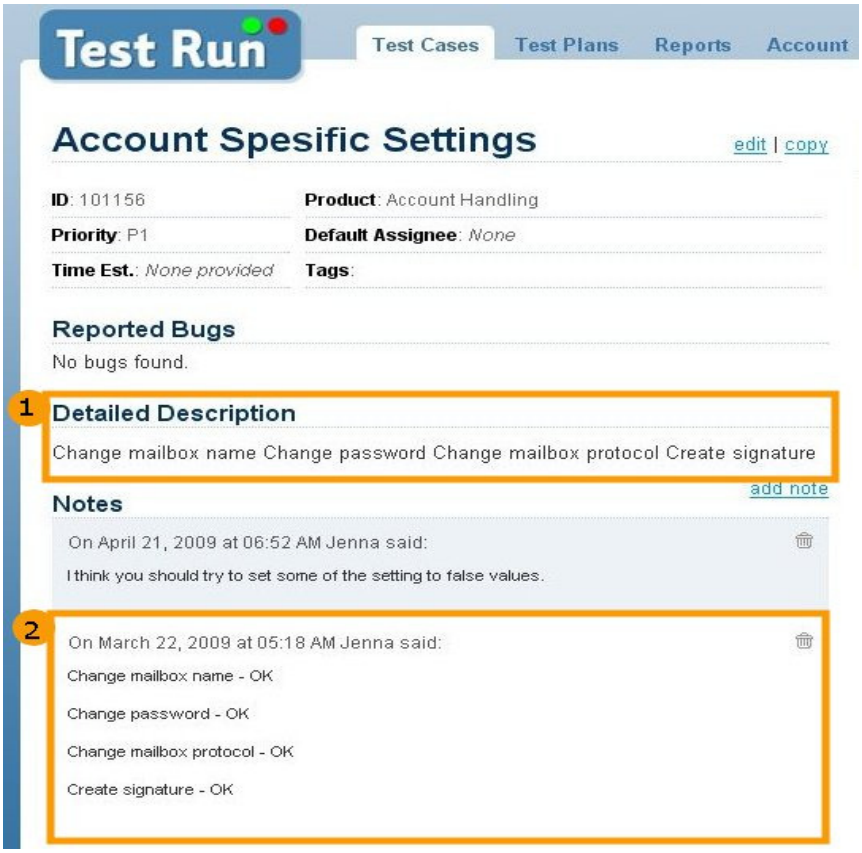


Figure 12. A User Story. 1: Detailed description shows the planned ET-cases. 2: By adding notes during the test run, test engineer can define all the done cases or areas covered by ET. (Test Run 2009)

The notes in Test Run are a quite efficient way to provide additional information for test planning – in addition to text, notes can include external files and they can even be private. Another interesting feature regarding Scrum team use is the handling of bugs. On Test Run, user can make references to a bug in an external error-database to a certain user story during the test run and manage them by marking them resolved. When combined the simple way for adding comments, this tool could very well work as a part of a scrum team communication.

The simple hierarchy of the Test Run works well, but because the tool is still under development, the Product-level does not have very much functionality available. If the user were for example able to better configure *what* the product is, follow the maturity level of a product and such, this tool would be a lead choice for a team approaching the test management for the product-type point of view.

When considering the requirements set for the test tool, Test Run is very noteworthy tool. It provides all the required basic functionality and is very easy, light and pleasant tool to use. Unfortunately, the beta version is not as extent as I hoped it to be – simplicity of some features like reports and product management and stray problems with performance do not really make it reliable enough for our needs.

6 CONCLUSIONS

6.1 Comparing the tools

Functionality

One of the outcomes of this study is that there probably is no tool that can provide all the required functionality without at least some reconfiguration of the tool itself. The compared tools were all at least somewhat configurable and each provided some very unique, good features for Scrum team work – for example Quality Center's good hierarchy structure, Dabble DB's web interface and Test Run's very powerful notes.

While every tool provided a variety of usable features, there were also a lot of lacks and deprivations especially in commercial tools – like Dabble DB's shorthanded calculation functions. Considering these deprivations and the almost unavoidable need for reconfiguration, it is quite clear that the most usable tool function-wise is the Excel spreadsheet. While it might require a lot of fine tuning, it is basically the only tool that can meet all the functional requirements without significant flaws.

Accessibility

Considering accessibility, Dabble DB and Test Run both provided an excellent, lightweight web-interface usable with multiple browsers. Sharing of database resources is easy and administration of user access could be done inside the development team without a need for requesting access rights from an external source – a good feature if all the users are inside the specific team. Possible security issue if, for example, subcontractors need to gain access to the tool as well. Spreadsheets, when stored on a common network drive, do not have this problem; administration can still be made inside the team but security issues have already been handled in a different level.

Usability and Reliability

Reliability and usability are the key issues with Quality Center at the moment. Constant downtimes make working with the test assets really frustrating and there are even questions about the reliability of the environment: the changes made to the test asset status in one view are not always updated properly to another view. One major issue more is the complexity of the environment – as already mentioned, it is not very easily adapted by the developers and the threshold of using the environment to do extended Scrum-related work is very, very large.

The 3rd party web tools on the other hand are very easy to use. Both Dabble DB and Test Run have very simple, easy to use user interfaces and it will not take a lot of time to learn the basic features. Reliability could, on the other hand raise some issues since the 3rd party applications are vulnerable to server downtimes and network problems in a same way as the Quality Center.

One of the best features of Excel-spreadsheets is the fact that they are familiar to almost everyone. A properly configured spreadsheet is easy to use and taking backups is very effortless. With spreadsheets, the largest reliability issue lies within the users – my personal experience of common Excel-spreadsheets is that they tend to accumulate into big heaps of old, useless data if the maintenance and common responsibility is forgotten.

Flexibility and Scalability

Agile way of working requires flexibility – ability to adapt the tools used while working on the constant changes of the working environment. Small and middle-sized projects like the one I am working with looses a lot of their flexibility, if the tools used are not designed or even adaptable for quick and effortless changes. The Quality Center is a good example – while changes made to test assets update to all the instances using the data, the amount of variables needed is so enormous that updating it simply takes way too much time.

The 3rd party tools in the study solve the flexibility problems quite well. A simple, comprehensive user interface with few variables is especially well implemented in Test Run. Changing the variables is easy, adding notes is effortless – for small or middle-sized software Test Run is the most flexible choice. It is also very likely that if the project expands and the amount of data will increase a lot, the Test Run's user interface and its clarity will suffer a lot less than when using Dabble DB or Excel-spreadsheets. With Excel, the problem of extensive amounts of data could be solved by adding extra spreadsheets to gather information from other spreadsheets; with Dabble DB this is not so easy because of the much weaker functions for handling written data.

Informativity

One of the original issues related to this study were the time and effort used to reporting instead of testing. Using Agile methods does not mean giving up reporting or documentation as a whole, but I believe that many projects could yield some of their reports, documents or charts by using tools or working environments that automatically collect, compile and present the data needed by for example external managers.

Clear, informative user interfaces can in some situations tell more about the project's maturity status than any pie chart or figure – of the tools described in this study Test Run clearly had the most informative, readable user interface. Dabble DB had also a very interesting possibility to configure views for different user groups – very usable for the management as they may not be as interested about a single User Story as they are about the overall maturity of the software.

Yet again, configurability sets some limitations to the commercial tools. While the features used for reporting and sorting out the information are useful, the lack of configurability cripples them a lot. Excel-spreadsheets may not look so pretty, but if configured correctly they can be an extremely efficient way to compile and present data. For example, information about sprints and finished user stories can be in a whole different file than the compiled, common information about the status of a requirement and the user stories linked to it.

Closure

I must admit, that I was somewhat surprised about the results of this study. I was quite confident that some 3rd party tool would definitely provide usable framework for Scrum team work without extensive configuration. While Dabble DB and Test Run provided a lot of very good, preconfigured features, it was interesting to see how much more our program actually would need from the tool in a form of configurability. In this sense, the study gave me a much better insight about our working environment than any introduction lesson.

Introducing either one of the 3rd party tools would be very easy and even the price would not most likely be an issue since they are quite inexpensive. Still, one big question remains: what is the point of using an external tool when we already have a program that almost everyone is familiar with, that is completely configurable to meet our criteria and for what we do not have to pay anything extra? Even if the Excel-spreadsheets are not the most simple, ready-to-go-solution, the 3rd party solutions do not actually provide anything new functionality that could not be achieved with Excel. However, studying the external tools gave me some good ideas about what would be the most efficient way to compile an Excel-spreadsheet and what is actually necessary for providing the information about software maturity.

6.2 Further development

In Chapter 3.3 I presented some problems and issues that have emerged from our project during the study: inadequate test planning and lack of communication in the scrum team. In the beginning of the study, I rarely got any information about what I should actually test in the sprint planning. The developers did tell about their implementations, but the information was mostly based on face-to-face communication and in some occasions I even forgot some issues under test before I got the build to test them with.

After completing the first stage of the study, I did another version of the Excel-spreadsheet, focusing now to the test planning and using the spreadsheet to gather information about any issues or ideas that developers or product owners might have about how the product should be tested. Discussion about testing and test objectives

definitely took a lot more time from sprint planning, but the spreadsheet definitely helped me to plan my own testing beforehand more efficiently. At this point, the tool served only the test engineer and the product owner: the product owner could tell at the sprint planning what kind of output he wanted from the testing or what at least should be tested. In sprint review, I could then show the results to the product owner and use them to state the reasons why I think that the feature has been tested thoroughly enough or why it should be tested more, fixed or otherwise reviewed. In that sense, I felt that the spreadsheet got the scrum team a bit closer to the idea that the team acts as a test manager and promoted better the fast feedback emphasized by session-based test management.

In general, I do not believe that a forced process can solve issues that are caused by the motivation or attitudes of the people using them. Still, while the developers in my team continue to participate somewhat inconsistently to test planning, I tend to believe that some forced communication might help them to see the importance of the shared knowledge.

From the positive experiences I had on my own tasks with the new spreadsheet, I had a short discussion with a developer from our project about what adjustments the spreadsheet would need in order to improve communication within the Scrum team and still be appealing enough for developers to fill out. As a result of this conversation, I designed a new, much simpler look for the spreadsheet. Screenshots from the new spreadsheet in Appendix 2 show the overview and the user story views with example information.

The new Excel-spreadsheet only has one view that a developer needs to use in order to comment all the user stories in one sprint and at the same time see what user stories are already tested, what features were planned to be tested already in sprint planning and what related problems have emerged in each user story. The detailed information about a user story can be reviewed on the story's own spreadsheet if needed.

The new spreadsheet seeks to resolve issues related to communication and an agile way to plan the testing. It should help to gather developers' and product owner's opinions about what they think should be tested from a new feature. In addition, the spreadsheet

tells if the story is done or not from the test engineer's point of view. The spreadsheet still has a simple linkage to the requirements based on the original targets of the study but the focus has shifted very strongly towards the communication and building up common test targets for a development team. I believe it would bring our testing work closer to session-based test management, but in a sense that the team together manages the testing effort and sets the goals that help to improve the overall quality of the product.

The spreadsheet has not yet been tested with a scrum team, but it has received some very encouraging feedback from development. Based on that feedback I believe that the spreadsheet is a usable solution that can be introduced to the teams very easily. At the moment, the quality issues in our program are under serious discussion and the attitudes toward team communication are slowly changing to support the use of a tool like the presented Excel-spreadsheet. I am hoping to be able to test the spreadsheet in actual sprint work at the latest during summer and convince the developers in my team that when the outlines of testing are set together and the communication is on a firm basis, the quality will improve and testing will become more efficiently focused, specific and successful.

If the spreadsheet proves to be a workable solution, there are a number of ideas for further development. Strengthening the linkage to the requirements, adding links to the error database or other external sources and so on are all interesting options for future studies of the spreadsheet itself. However, I would like to see the future development concentrate especially in our approach to testing and quality. If the spreadsheet is adapted well and communication inside the team actually improves, it could be easier to start planning testing and development more in the way it is described in Chapter 3.2: by actually using for example the Test Objectives not only as a criterion to pass a sprint but as a guideline to how the whole implementation should be done, how it should work and what should be done in order to ensure both the functionality and the quality of the product. In a way, I believe that only when taking the perspective of testing into account already at the planning stage and using that perspective as one of the guidelines for development, the quality of the product can actually be called built-in.

7 FINAL WORDS

The goal of this thesis was to find ways to improve the flow of information inside the scrum team and offer the team a framework for more efficient test management and planning as well as to offer more transparency to the testing effort. Using a test bench configured according to the requirements set for the tool, I performed a practical study comparing four different environments. During practical study, new problems regarding test management and planning emerged and the focus of the thesis was reconfigured so that the final result would commit to these problems as well as the original goal.

Output of this study is an Excel-spreadsheet designed to improve the communication inside the Scrum team, especially between developers and test engineers. The framework provided by the spreadsheet is simple and lightweight and implementing it for the use of the scrum team should not overburden the team with excess documentation. The tool has not yet been tested with a scrum team, but the draft versions that I have used in my own tasks as a test engineer have proven to be a dramatic enhancement compared to previous methods. Based on the outcomes of this study, the successful refocusing of working methods and the final outcome, the spreadsheet, I could say that the objects set for this thesis were met.

I am very satisfied with the outcome of this thesis. The Excel-spreadsheet helps me with my daily work and can provide my scrum team documentation about how I have tested the features that our developers have implemented. At the same time, I feel that I have been able to bring up some issues that I would not normally have the courage to bring up. In a way, this thesis gave me a good orientation to my work and has helped me to be an active member in a Scrum team. I was forced to rethink the ways of working and question the current policies.

I do believe though, that same kind of results could have been achieved with a different kind of approach to the study. If I would redo the thesis, I would forget the comparison of different tools and focus completely on the Excel-spreadsheet. The economical situation, a tight schedule which forced me to reduce some details and the newly emerged problems that felt much more interesting to study, made the comparative study feel a bit gratuitous from time to time. The study gave some ideas about what our team actually needs from the tool, but I can not help to think that the time used on the

comparison of different tools would have been more profitable if used on the development of the Excel-spreadsheet. The practical work done for this thesis could however provide a good starting point for future thesis writers.

As a whole, doing this thesis has been a good experience. The process and the people behind it have taught me a lot about software quality and agile software development and this thesis reflects those lessons and my own opinions about them very well. I feel that I have been able to combine the previous experiences I have had as a test engineer to the questions raised in my current position and build my own perspective to software quality.

REFERENCES

Written References:

Hazzan, Orit & Dubinsky, Yael 2008. Agile Software Engineering. Undergraduate Topics in Computer Science, Springer

Mark, Scott 2007. Test-Driven Development: An Agile Practice to Ensure Quality is built from the Beginning. Ioannis G.Stamelos & Panagiotis Sfetsos: Agile Software Quality Assurance, 2007. Information Science Reference.

Schwaber, Ken & Beedle, Mike 2002. Agile Software Development with Scrum. Prentice Hall.

On-Line References:

Bach, James 2003. Exploratory Testing Explained. [Online] [referred to 10.2.2009] http://www.satisfice.com/articles/et-article.pdf

Bach, Jonathan 2000. Session-Based Test Management [Online] [referred to 10.2.2009] http://www.satisfice.com/articles/sbtm.pdf

Battue Inc 2006. Scrum Process [Online] [referred to 20.04.2009] http://www.1dot0.com/dev_process.html:

Beck, Kent & Beedle, Mike & Bennekum, Arie van & Cockburn, Alistair 2001. Manifesto for Agile Software Development [Online] [referred to 7.4.2009] http://www.agilemanifesto.org/

FitNesse 2009. FitNesse Acceptance testing framework. [Online] [referred to 18.05.2009] http://fitnesse.org/

Itkonen, Juha & Mäntylä, Mika V. & Lassenius, Casper 2007. Defect Detection Efficiency: Test Case Based vs. Exploratory Testing. Helsinki University of Technology, Software Business and Engineering Institute, TKK, Finland. [Online] [referred to 18.04.2009] http://www.soberit.hut.fi/jitkonen/Publications/Itkonen_M%C3%A4ntyl%C3%A4_Lassenius_2007_ESEM.pdf

Kalman, Sam 2007. SBT Lite – Components of Session-Based Test Management. [Online] [referred to 18.04.2009] http://www.igda.org/qa/docs/SBTLite_whitepaper.pdf

LoadStorm 2009. Agile Testing Tool. [Online] [referred to 18.05.2009] http://loadstorm.com/agile-testing-tool

Panchal, Dhaval, 2008. What is Definition of Done? [Online] [referred to 27.04.2009] http://www.scrumalliance.org/articles/105-what-is-definition-of-done-dod

Pettichord, Bret 2002. Agile Testing – What is It? Can it Work? [Online] [referred to 20.1.2009] http://www.io.com/~wazmo/papers/agile_testing_20021015.pdf

Pettichord, Bret 2004. Agile Testing Challenges. Pacific Northwest Software Quality Conference 2004. [Online] [referred to 20.1.2009] http://www.io.com/~wazmo/papers/agile_testing_challenges.pdf

Rally Software 2009. Agile testing tool. [Online] [referred to 18.05.2009] http://www.rallydev.com/

Scrum Alliance Inc 2009. Scrum Alliance. [Online] [referred to 20.01.2009] http://www.scrumalliance.org/

Unpublished References:

Kontio, Pertti 2008. Introduction to Agile Development at Nokia,Internal Release. Refered to 20.1.2009

Oivo, Markku 2008. Scrum & Agile Development, Training. VTT Technical Research Centre of Finland 9.12.2008. Tampere

APPENDICES

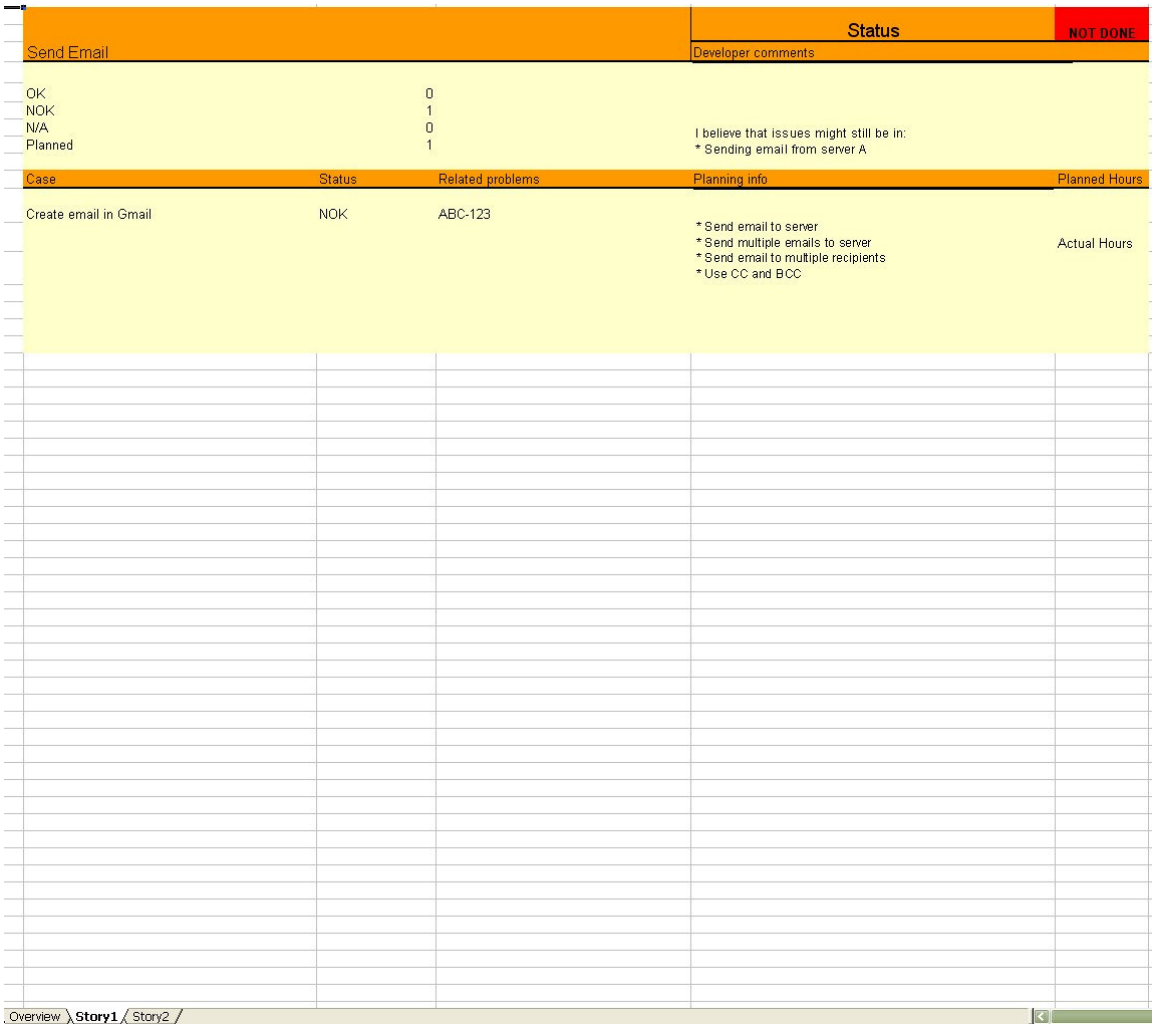
Appendix 1: Comparison of the Quality Control tools.

	Quality Center	Excel	Dabble DB	Test Run
Functionality	+ Support for test cases, detailed test steps, test plans, test sets and requirements + Links to Requirements work well - Too detailed for agile work - Too complex for Scrum team use	+ Adaptable + Can be configured to meet all the functional Requirements by building a hierarchical sheets that are linked to each other + Familiar and widely used + Provides a good platform for Scrum team use + data is very reusable - Needs quite a lot of spesifications and fine tuning in order to get the optimal setup	+ Database structure enables building similar hierarchy than in Excel + Web-based interface supports Scrum use + Data is reusable + Basic functions resemble Excel - More complex calculations are not possible - Support for notes is quite bad - Updating takes time and interface is not that good	+ Supports test cases, test plans and hierarchy structure that could be used to mark the requirements + Some really nice and usable features, like notes, handling of bugs etc. - Still under development, some functions do not work as widely as possible
Accessibility	+/- Due to company wide use, requires a complex process of access rights and administration - Only usable with IE	+ Usable not only in Windows, Open Office and Macs support this too + Can be used from network drive, linked to wiki page etc - If the sheet gets complex(for example, if macros are needed), support for other platforms might suffer	+ Web interface is very lightweight and useable by multiple browsers + Sharing of the database access is easy + Different views support the access levels	+ Web interface is very lightweight and useable by multiple browsers + Sharing of the database access is easy
Usability and Reliability	+ Comprehensive - Slow to learn, mostly due complexity - Constant downtimes are big impediment	+ Working, ready made sheet is easy to use + Reliable, when used right +/- Maintenance might require effort but can be done inside the team - Reliability of the information might be questionable if updates are trailing	+ Simple and easy user interface + Very easy and effortless to learn - both in basic and advanced features - Updating cells is not that straightforward - Reliability might be an issue, since it's an 3rd party application and server downtimes etc. might occur	+ Clear and easy interface + Effortless to learn - 3rd party issues might be a concern
Flexibility and Scalability	+ Changes update easily to the whole system + Large test assets suit well to this tool - Small programs/changes do not actualy require a tool this massive	+ Easily adapted to the small and middle-sized projects + Interest groups/requirement tiers can be added or removed easily +/- Can handle larger amounts of data as well, but clarity might suffer a lot - Sheet with large amounts of data might expand too much for convinient updating	+ Easily adapted to the small and middle-sized projects +/- Can handle larger amounts of data as well, but clarity might suffer a lot - Large amounts of data might make the tool very slow - Modifying the hierarchy is not as easy as in excel due weaker linking	+ Very easy to use in small/middle size projects, but could be adapted to larger scale assets better than database-type solutions - Is not very extensinble in hierarchy
Informativity	+ Information about requirement coverage is clear - In need of detailed results, export to excel is needed	+ A well composed sheets can at the same time provide both very detailed and general information - Extracting and modifying the information can take time	+ Different views are very interesting way to compile information from the database to charts + Views also provide a way to show every interest group the information they need	+ The interface itself is clear and informative +/- Reports look nice, but lack of configurability really cripples a nice feature
Other	Suitable for larger test assets like extencive language variant-testing. Does not have many - if any - considerable implementations in Agile.	Adaptable tool that suits well for the agile teams purposes.	Interesting tool, that unfortunately suffers a bit from it's similarity Excel. If mathematical operations were more extent,this would be a great tool.	Very nice tool for testing. Because it's still in development, some features are bit shorthanded. Good for simple test management and usable in agile, but lacks the formability required by this project.

Appendix 2: The Final Scrum Excel

The overview of the Scrum Excels shows all the user stories in the Sprint as well as the info from test planning and the developer comments.

The User Story view of the Scrum Excel shows all the detailed information related to a user story including: planned cases, information about their status, planning info and the comments from development.



Send Email			Status	NOT DONE
			Developer comments	
OK		0		
NOK		1		
N/A		0		
Planned		1	I believe that issues might still be in: * Sending email from server A	
Case	Status	Related problems	Planning info	Planned Hours
Create email in Gmail	NOK	ABC-123	* Send email to server * Send multiple emails to server * Send email to multiple recipients * Use CC and BCC	Actual Hours

Overview | Story1 | Story2