Master's Thesis

Software Configuration Management in Scrum Projects

Andreas Bergström DT06

Department of Computer Science
Faculty of Engineering LTH
Lund University, 2008

ISSN 1650-2884
LU-CS-EX: 2008-33

Software Configuration Management in Scrum Projects

Andreas Bergström

Department of Computer Science
Lund Institute of Technology
Sweden

LUND UNIVERSITY

June 2008

This thesis is submitted to the Department of Software Engineering and Computer Science at the Lund Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering.

Contact Information

Author:
Andreas Bergström
E-mail: bergstrom.andy@gmail.com
Homepage: www.andreasbergstrom.net

External advisor:
Magnus Hägg
TietoEnator
Homepage: http://www.tietoenator.com

University advisor:
Lars Bendix
Lund Institute of Technology
Homepage: http://www.cs.lth.se

Mailing address:
Faculty of Engineering LTH, Lund
LTHs kansli
PO Box 118
SE–221 00 Lund
Sweden

Internet : www.lth.se
Phone : +46 46 222 72 00 (Informations Desk at Faculty of Engineering LTH)

Abstract

Software Configuration Management (SCM) is an important part of traditional software development. It keep things under control and helps software development teams create top-quality products without chaos and confusion. The Configuration Manager is the key person that makes sure all of this happens.

Scrum is becoming a very popular project management method for agile software development. Scrum values self-organizing- and cross-functional teams, and because of this, there should be no roles within the team, i.e. no designated Configuration Manager. Obvious questions that arise are; Who will handle the Configuration Management? Is it considered at all?

Since there are very little information written about SCM in Scrum projects, this thesis aims to investigate if- and how SCM is commonly handled in Scrum projects.

From interviews with Scrum practitioners I learned that it is rare for SCM to be considered explicitly. The main reason seems to be lack of knowledge. Because of this I have written checklists to help unexperienced teams with their SCM-functionality.

The Configuration Manager will still be an important asset, but his or her duties will probably shift to more educational and consulting when SCM improvements are needed and when problems arise.

Keywords

Configuration Management, Scrum, Agile, Software Development, Roles & Responsibility

Preface

After discussing several alternative for my thesis with my supervisor at TietoEnator, the idea about Software Configuration in Scrum projects came up. He was currently working as a Configuration Manager and I had previously taken some courses that covered basic SCM, which I found very interesting. I looked into it some more, and there are indeed lots of book and articles written about Scrum, SCM, and even Agile SCM. However, almost never is SCM mentioned explicitly in the Scrum literature or vice versa, leaving Scrum people with the expression that SCM is not needed and SCM people with the impression that Scrum is not sound from an SCM perspective.

I therefore intend to enlighten these topics further, and I hope to be able to make a small contribution to the development industry.

Acknowledgments

First and foremost I would like to thank all the knowledgeable and experienced people I interviewed. They took time from their busy schedule to answer my questions, shared their experience and gave valuable feed-back to my ideas.

My supervisor Magnus Hägg for introducing me to the subject and for all his help during the project, giving me valuable feed-back and setting me up with the right people.

Last but not least my examinator Lars Bendix for all his help and assistance during the whole time. His research and knowledge about SCM and agile methods has been a great inspiration and a big help for me.

Andreas Bergström

Contents

1 Introduction

Software Configuration Management (SCM) is an important part of Software Development. A properly implemented SCM system keep things under control and helps software development teams create top-quality software without chaos and confusion. The Configuration Manager is the key person(s) that makes sure all of this happen.

Scrum is becoming a popular project management method for agile software development. Scrum is more like a framework and says nothing about how to develop software (unlike eXtreme Programming for example), instead it's about how to *manage* the development process. Furthermore, there should be no specific roles within the Scrum Team, i.e. no designated Configuration Manager.

Because of this, there is an obvious risk that no one will consider SCM activities and that the project will be unmanageable after some time, especially if the practitioners are unexperienced in the area and they don't know the benefits. From interviews with Scrum practitioners I learned that it's rare for SCM to be considered explicitly, mostly because of lack of knowledge. Although a lot of literature has been written about Scrum, SCM is almost never mentioned explicitly.

1.1 Research Questions

The questions I intend to answer with this thesis are the following:

- Is Scrum complete in a SCM perspective?
- How can SCM practices provide service and support to Scrum projects?
  - In what way can the Product Owner benefit from SCM?
  - In what way can the Scrum Master benefit from SCM?
  - In what way can the Scrum Team benefit from SCM?
- How can/should the Configuration Manager act when working with a Scrum Team?

1.2 Aims and Objectives

As I see it, there is a clear need to investigate just how well SCM is handled in different Scrum projects, since there are so few guidelines and material covering this topic.

When using Scrum (and other agile project management methods), there is often no designated Configuration Manager within the development team. This is because the team should aim to be cross-functional and thus, it should be up to the team members it-selves to handle the daily SCM-tasks together. If the team has limited knowledge about SCM it can quickly lead to chaos.

This report aims help Scrum practitioners to implement SCM-functionality and explain which SCM tasks are necessary and why some, probably, could by left out, i.e. to make Scrum complete in a SCM-perspective.

One stakeholder of the project, the Scrum practitioners, will benefit of having a 'manual' or some guidelines how to improve their SCM-functionality, which could perhaps also work as template when writing a brief SCM-plan. These guidelines will look different for each of the three roles in Scrum; the Product Owner, the Scrum Master, and the Scrum Team.

The other stakeholder, the Configuration Manager, will need to know how their duties will differ when working with a Scrum Team, and which things are important to think about.

1.3 Method

In the beginning of the project I concentrated on studying the main topics: Software Configuration Management and Scrum. By doing this I got a much greater theoretical ground to stand on.

In order to find out more about how SCM is handled in different Scrum projects I interviewed experienced people from different companies. I talked to Scrum Masters, Product Owners and SCM-experts. I did not manage to organize a meeting with a Scrum Team.

After theory studies and the preliminary interviews I had an analysis phase where I collected materials and ideas from many different sources, and from that I made some first conclusions.

These conclusions were later discussed with a few of the previously interviewed persons in order to get feedback. Based on the new input I continued working further with the analysis.

During the project I also held presentations at the annual IKEA IT CM-conference (Helsingborg) and at The Scandinavian SCM day (Lund), where I also got some valuable feedback from experienced Configuration Managers.

1.4 Limitations

One of the most difficult parts was to find people to interview, and as I mentioned I did not get a chance to talk to a Scrum Team. However, several of the Scrum Masters and Product Owner had a wide experience and had been closely involved with the Scrum Teams. I therefore think I got the same information and feed-back from them as I would have got from a Scrum Team.

Unfortunately I had no chance to test my results in a real development project, which would have been valuable. The chapter about developer oriented aspects of SCM is only theoretical and have not been reviewed by Scrum people. However, most of the techniques are commonly accepted 'best practices' so they are likely to be useful. The checklists however have been reviewed by experienced Product Owners and Scrum Masters and I received very positive feed-back.

1.5 Disposition

Chapter 2: I will in this chapter give a brief Theoretical Background of the two major topics discussed throughout the report; Software Configuration Management and Scrum.

Chapter 3: After the theoretical overview, I will do an analysis about how SCM and Scrum can work together in an agile fashion. I will also talk about how the CM-role is likely to be affected.

Chapter 4: Based on the Analysis, a result will be presented in Chapter 4.

Chapter 5: At last; Conclusions and Further work.

2 Theoretical Background

Since the target groups of this thesis are both Scrum practitioners and Configuration Managers, I will in this chapter give a brief introduction of the two major topics discussed throughout the report, i.e. Scrum and SCM.

As I understand, there might be limited knowledge about SCM in development teams, and it will therefore be useful for some readers to get a short overview about what SCM is all about, and about its benefits. We will talk about the traditional aspects of SCM, which are more aimed towards management, and also some developer oriented aspects.

It is perhaps equally important for the Configuration Managers, who is about to start working with a Scrum Team, to understand not only *how* Scrum works, but also the values behind Scrum that must be respected when implementing the SCM-functionality.

The analysis in Chapter 3 about how SCM and Scrum can work together, will partly be based on the basic theory presented in this chapter.

2.1 Software Configuration Management Benefits

Configuration Management is the art of identifying, organizing, and controlling modifications to the software being built by a programming team. The goal is to maximize productivity by minimizing mistakes [1]. Practicing configuration management in a software project has many benefits, including increased development productivity, better control over the project, better project management, reduction in errors and bugs, faster problem identification and bug fixes, and improved customer goodwill [2].

Software projects are becoming more complex in size, sophistication, and technologies used. In addition, users of software systems have matured, and the bugs and defects in a system are detected and published faster than ever. So if the companies wants to keep their reputations intact and prevent its market share from dropping, they have to provide the fixes and patches quickly.

To survive in this competitive world, organizations need some sort of mechanism to keep things under control or total chaos and confusion will result, which could lead to product or project failures and put the company out of business. A properly implemented Software Configuration Management system is such a mechanism that can help software development teams create top-quality software without chaos and confusion [2].

A properly designed and implemented SCM system has several benefits, including [2]:

- Improved software development productivity
- Easier handling of software complexity
- Improved security
- Higher software reuse
- Lower software maintenance costs
- Better quality assurance
- Reduction of defects and bugs
- Faster problem identification and bug fixes
- Assurance that the correct system is built

There are both the traditional aspects of SCM, which are more aimed towards management, but also developer oriented aspects. In the following two chapters we will go further into this.

2.2 Traditional Software Configuration Management

In order to follow the analysis in Chapter 3, and the results in Chapter 4, it will be necessary to get a brief brief overview of the concepts of Traditional SCM.

SCM traditionally consists of the four activities: Configuration Identification, Configuration Control, Configuration Status Accounting and Configuration Audit [2].

If the reader is already familiar with these concepts, this section can be skipped.

2.2.1 Configuration Identification

Configuration Identification is the activity where a system is divided into uniquely identifiable components, called Configuration Items (CI), for the purpose of software configuration management.

Examples of CI includes: project plan, specifications, design documents, source code, test plans and test data, executables, make files, tools and the SCM Plan.

A checklist for the selection of configuration items can include:

- Is the item critical/high risk and/or a safety item?
- Is the item to be used in several places?
- Will the item be reused or designated for reuse?
- Would the item's failure or malfunction have an impact on the system?
- Will the item be maintained by diverse groups at multiple locations?
- Is the item highly complex or does it have strict performance requirement?
- Is the item likely to be subject to modification or upgrading during its service life?

Each Configuration Item must be uniquely identified. The identification method could include naming conventions and version numbers and letters.

When using a waterfall method this will be the first major SCM function, providing the basis for the other SCM activities. Since you have to establish the structure and elements of the complete software system and define interrelationships between the CIs, it can be a very heavy activity, especially for large projects.

2.2.2 Configuration Control

Software requirements can change often and quickly, and the development team must be able to respond to these changes in order to satisfy the customer. However, if changes are implemented in an uncontrolled manner, it will sooner or later lead to chaos. Configuration Control is about handling changes in a controlled way through formal change control procedures, including: evaluation, coordination, approval or disapproval and implementation of changes to configuration items.

A change request (CR) is a document containing a call for an adjustment of a system. CRs typically originate from one of five sources [3]:

1. Problem reports that identify bugs that must be fixed, which forms the most common source
2. System enhancement requests from users
3. Events in the development of other systems
4. Changes in underlying structure and or standards (e.g. in software development this could be a new operating system)
5. Demands from senior management

A change request is first evaluated by a Change Control Board (CCB), that approves or disapproves the request. The concerns the CCB discusses in order to make this decision include the following [2]:

- *Operational impact.* What will the effect of this change be on the final product?
- *Customer approval.* Will the change require customer approval? Is it a major change?
- *Development effort.* What is the impact of the change on interfaces and internal software elements of the final system?
- *Interface impact.* Will the change affect the established interfaces of the system?
- *Time schedule.* At what point is this change incorporated? What is the time for incorporation with minimal impact on cost and schedule?
- *Cost impact.* What is the estimated cost of implementing the change‘
- *Resources impact.* What resources (e.g., infrastructure, skill, people) will be required to implement the change?

- *Schedule impact.* How will the processing and incorporation of the change affect the current schedule?
- *Quality impact.* How will the change affect the quality and reliability of the final product?
- *Feasibility.* Considering all of these factors, can this change be made economically? What is the risk of implementing the change? What is the risk of not implementing or deferring the implementation?

If a request is approved, the proposed change is assigned to a developer for implementation. When completed, the implementation needs to be verified through testing before the CCB can finally close the Change Request.

2.2.3 Configuration Status Accounting

The aim of Configuration Status Accounting (CSA) is to keep managers, users, developers, and other project stakeholders informed about the various configuration stages and their evolution. This implies three basic tasks: data capture, data recording, and report generation [2].

CSA involves the storage and maintenance of:

- Information about the product's configuration (such as part numbers or changes install in a given unit)
- Information about the product's operational and maintenance documentation
- Information about the SCM process (such as the status of change requests)

A good status accounting system should be able to answer question such as the following, and many more:

- What is the status of an item?
- What items were affected by a particular change request (CR)?
- Which version of an item implements a certain CR?
- What CRs are assigned to whom?

- How many high-priority CRs are currently not implemented?
- What is different about a new version of a system?

Status accounting reports include change logs, progress reports, CI status reports and transaction logs.

The information provided by the status accounting function is useful in determining the performance characteristics of the project, such as number of change requests, approval rate, number of problem reports, average time for a change resolution, average implementation time, and cost of implementing a change. This information will help when evaluating the performance of the project an when comparing different projects. Also, these details will help fine-tune the estimation and costing procedures of the organization.

2.2.4 Configuration Audit

The objective of the Configuration Verification and Audits is to verify that the software system matches the configuration item description and that the system is complete [2]. The process can be divided into three parts; Functional-, Physical-, and In-process Configuration Audit.

Functional Configuration Audit

A functional configuration audit aims to ensure that the software product has been built according to specified requirements. This process often involves testing of various kinds.

Physical Configuration Audit

A physical configuration audit determine whether all the items identified as a part of CI are present in product baseline.

In-process Configuration Audit

An in-process audit ensures that the defined SCM activities are bring properly applied and controlled.

Usually a representative from management, the Quality Assurance (QA) department, or the customer performs such audits. The auditor should have competent knowledge both of SCM activities and of the project.

2.3 Developer Oriented Software Configuration Management

In the last section we talked about the Traditional SCM activities. They are mostly about keeping control over the project; ensure that it progresses according to schedule and that its delivery contains all the right parts. Thus aimed more towards management.

There are however also developer oriented aspects of SCM, including: Version Control, Workspace-, Build-, Change-, and Release Management [4].

With the background of these concepts, further analysis about how they can work together with Scrum will be made in Chapter 3.

2.3.1 Version Control

A Version Control System offers many advantages to both teams and individuals. It allows multiple developers to work on the same code base in a controlled manner, and it keeps a record of the changes made over time. If you see some strange code, it's easy to find out who made the change, when and why. Besides this, you can easily go back an hour, a day, or a month and fix the mistake. The system also allows you to support multiple releases (branches) of your software at the same time as you continue with the main line of development.

2.3.2 Workspace Management

Workspace Management is about setting up the developer's private workspace, also known as 'Sandbox', in an appropriate way. The workspace should be optimal for the developer's productivity and also allow several developers to work in parallel without disturbing each other.

2.3.3 Build Management

Build Management handles the activity of putting together modules in order to build a running system. The build can include steps like retrieving the correct components from various locations in the repository, compilation, running scripts that generate source files from meta-data, etc. Multiple variants of the system can be described in a single system model and the Build Management tool will derive different configurations, effectively building a tailored system for each platform or product variant. Incremental builds, that only compile and link what has changed, can be used during development for fast turn-around times. A full build, rebuilding the entire system from scratch, is normally used during system integration and release.

2.3.4 Change Management

Change Management is a general term encompassing controlling and tracking change. It includes tools and processes which supports the tracking of changes from the origin to the approval of the actually implemented source code. This is commonly handled through a formal Change Control Board which must allow the change before it can be implemented. In some cases it may important to keep traceability between a change request and its actual implementation, and also to allow each piece of code to be associated to an explicit change request.

2.3.5 Release Management

Release Management deals with both the formal aspects of the company releasing to the customer and the more informal aspects of the release to the project. In the case of a customer release, it's necessary to preform a configuration audit before the actual release. When releasing changes to the project, we must decide how to integrate changes from developers, when this should be done, and the quality requirements of the change.

2.4 Scrum

In the previous sections, we got an overview about what SCM is. Here we will continue the theoretical background and talk about Scrum; a project management method for agile software development [5].

As with the previous sections, it will be necessary to know some basic concepts in order to follow the analysis in Chapter 3 and the results in Chapter 4.

2.4.1 Overview

The illustration in Figure 2.1 shows the basic Scrum Process, which we will return to in section 2.4.3. But first some basic terminology:

The *Product Backlog* is a prioritized list of all product requirements, including new features, functions, technologies, enhancements, and bug fixes. The content can come from anywhere; users, customers, sales, marketing, customer service, and engineering can all submit items to the backlog. However, only the so called *Product Owner* (explained below) is allowed to prioritize the list, and will as a result decide what will be implemented. The first Product Backlog may be a list of requirements from a vision document, and then emerges as the customer's understanding of their needs increase.

The development teams take on as much top-priority Product Backlog items (Change Requests) as they think they can manage during the following iteration, called a *Sprint* in Scrum. During each Sprint, the teams maintains a list of tasks to perform, called the *Sprint Backlog*. Each Sprint should result in a fully functional, and potentially shippable version of the product.

2.4.2 Scrum roles

Scrum has three important roles; the Product Owner, the Scrum Master, and the Scrum Team.

Product Owner

The Product Owner is the one representing the customer. As already mentioned, he or she is responsible for creating and prioritizing the Product Backlog, and at the end of the Sprint, the Product Owner is also responsible for reviewing the system.

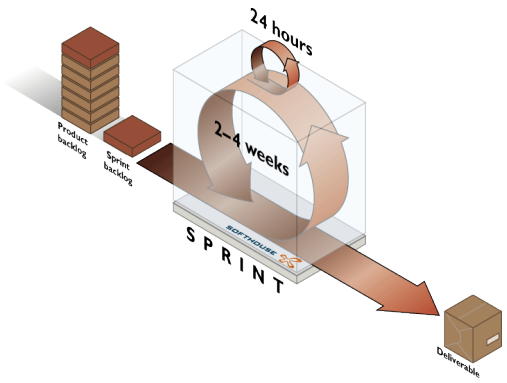

Figure 2.1: The Scrum Process (image from Softhouse)

For commercial development, the Product Owner may be the previous product manager. For in-house development, the Product Owner could be the project manager or the user department manager.

Scrum Master

The Scrum Master is responsible for ensuring that Scrum values, practices, and rules are enforced. The Scrum Master is the driving force behind all the Scrum practices; he or she sets them up and makes them happen. The Scrum Master presents management and the team to each other.

The Team Leader, Project Leader, or Project Manager often assume the Scrum Master role.

Scrum Teams

Scrum Teams are small (5-9 people), self-organizing and cross-functional, thus performs all design, development, tests etc. together. The team has full authority to do whatever is necessary to achieve the Sprint Goal. It is only constrained by organizational standards and conventions. Team members can interview others, bring in consultants, read books, browse the web, or whatever they need (within budgetary constrains) to achieve the goal.

There are no roles or titles within the team and the members doesn't have job descriptions other than doing the best work possible.

Since the team size are relatively small, multiple teams often develop product increments in parallel, all teams working from the same list of wanted features. This set-up is known as a *Scrum of Scrums*. How this influence and effect the SCM functionality is however outside of my scope.

2.4.3 The Process

Sprint Planning Meeting

At the Sprint Planning Meeting, customers, users, management, the Product Owner and the Scrum Team determine the next Sprint goal and functionality. The Sprint Planning Meeting actually consists of two separate meetings. First, the teams meets with the Product Owner, management and users to figure out what functionality to build during the next Sprint. At the second meeting, the team works by itself to figure out how it is going to build this functionality during the Sprint.

Daily Scrum Meetings

Each Scrum Team meets daily for a 15-minute status meeting called the *Daily Scrum*. The Scrum Master is responsible for successfully conducting the Daily Scrum, by keeping it short and making sure the team members speak briefly. Other stakeholders can also attend the Daily Scrum, but they are there as guests and are not allowed to interfere in any way.

Every person from the team shall, one at a time, briefly answer three questions:

- What have you done since the last Daily Scrum?
- What will you do between now and the next Daily Scrum?
- What got in your way of doing your work?

The first two questions gives the attendants a brief progress report, and lets the team synchronize their work. If a team member identifies something that is stopping him or her from working effectively, it is the Scrum Master's top priority to remove that impediment. Such impediments can be that a server is down or that the team member was asked by management to do something else.

Sprint Review Meeting

The Sprint Review Meeting is a four-hour informal meeting, where the team presents what it has been able to build during the Sprint.

Management comes to the Sprint Review to see what the team has been able to do with the resources that it has been given. Customers come to see if they like what the team has built and the Product Owner comes to the Sprint Review to see how much functionality has been built.

Since this is suppose to be an informal meeting, no one should prepare extensively for it. That is why Power Point presentation and similar are forbidden. The Sprint Review is a working meeting where everyone should get an understanding for the product increment, as this is the knowledge they will need for the next Sprint Planning Meeting.

Sprint Retrospective Meeting

After each Sprint Review, but before the next Sprint Planning Meeting, a Sprint Retrospective is held at which all team members reflect about the past Sprint. The purpose of the retrospective is to make continuous process improvement.

2.4.4 Advantages

Short iterations (Sprints)

By having these short iterations and daily meetings, the team will always know exactly where they are all the time. For example: every day you will know which features remains to be completed, if the team members did what they were suppose to, and by looking at the so called Burndown chart (visualizes estimated remaining work) you will see if the team is progressing as the marketing (project management) has hoped.

No waste of time

Because you only work on a few, top-priority, features at a time, Scrum makes sure that the team is not spending time and money developing stuff that no one will use. As a result, the development speed is likely to increase.

Cross-functionality

Scrum wants to avoid the barrier between different rolls and make everyone work together towards the same goal. Everyone is responsible for the whole product and by working together the team will join their skills to create better software with higher quality.

According to two of the principles behind the Agile manifesto [6]:

- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The best architectures, requirements, and designs emerge from self-organizing teams.

2.4.5 Important values in Scrum

When using Scrum, self-organizing and cross-functional teams are empowered to find their own way through complex situations. This freedom, along with the creativity that results from it, is one of the core benefits of Scrum. The development team will take more responsibility for the product if they are not told what to do all the time.

The Scrum Master is a coach or a leader - not a boss. Therefore, there will in many cases be a necessary shift from controlling to facilitating. This is probably the hardest thing in many organizations - to change the way they think.

It is also important for the customer participants to be committed, knowledgeable, collaborative, representative, and empowered. Otherwise the developed product typically do not transition into use successfully, even though they may satisfy the customer.

2.4.6 Why Scrum sometimes fail

The organization is not ready to adopt Scrum

Maybe the most common way for Scrum to fail is that organizations are not ready to adapt Agile development. The Product Owner continues to work controlling instead of facilitating, doesn't empower the team, and won't let them come up with their own way of working. This fear of loosing control is seen when requirements are over-specified and the team has lots of constraints. It is therefor essential to have an experienced Scrum Master who can help the Product Owner write requirements more abstract and to make sure that management knows what is needed in order for Scrum to work in their organization.

In conclusion, for Scrum to work, a change of mind-set is necessary.

The Product Owner can't prioritize

Another thing that is guarantied to wreck a Scrum project is if the Product Owner is unable to prioritize the work. One major benefit is that the development team always should work on what gives the most business-value. This can of course only be done if the Product Owner know what he or she is doing, make good decisions, and is able to effectively communicate and collaborate with the team.

The teams are not ready to work cross-functional

Traditionally there are fixed roles within the team; designer, programmer, tester, CM etc. In Scrum, everyone in the team should work cross-functionally.

The benefits of this is that everyone feels much more responsible for quality and the overall success of the project.

The team are not willing to improve their engineering skills

Much like some organizations and managers can have problems adapting to a new method, so can developers. A common problem is that developers doesn't work in an Agile fashion when it comes to method and technique. They have to use the power given to them and react to upcoming problems - otherwise the project is doomed.

Already existing problems become obvious

The problems that will arise the first time you try Scrum will probably not have anything to do with the method itself. Instead, it causes problems that are already there to become obvious.

- Every day you might find out that the people in the team didn't do what they were suppose to.
- Only one team member knows how to build the system.
- By looking at the Burndown chart, you will early see that your project is not on track.
- At the end of every Sprint you might find out that the team are not progressing as marketing (project management) has hoped.

On the bright side - you will find this out while you still have time to fix it.

3 Analysis

In the previous chapter we got an overview of the basic theoretical background of SCM and Scrum. That will work as an instrument when analyzing how the two major topics; Software Configuration Management and Scrum, can work together. The analysis however are not only based on theory. I have also taken into consideration real life experience from the people that I have interviewed; both Scrum Masters, Product Owners, and Configuration Managers.

First, some general differences when using SCM in Scrum projects, and also how the Configuration Manager is likely to be affected by them. Then I will discuss how the Traditional SCM-activities can be handled in Scrum. Last I will also talk some about how the developer oriented aspects of SCM can be used in a Scrum project. Most of these techniques are commonly known as 'best practices' (for Agile development at least) and does not necessarily have to be done in a different way when using Scrum.

It's important to stress that the material in this chapter are not about how things *should* be done, rather it's a discussion about how they *can* be done.

Based on the Analysis in this chapter a 'result' will be presented in Chapter 4, which attempt to help unexperienced teams to implement an appropriate SCM-functionality.

3.1 Why should we use SCM in Scrum?

It may seem that there is no (or little) room for SCM in agile- or Scrum projects, because many regard SCM as a process-heavy thing that gets in the way of the 'real work' of software development. This is of course not true. Even in agile development there is need of having a successful SCM-functionality, and it will bring much of the same benefits mentioned in section 2.1. It will however, have to be handled differently in order to meet the agile principles.

SCM will be different in Scrum

Much of the same benefits in SCM that are found in traditional development will also be found in agile development, for instance when it comes to keeping track of and coordinating the project. However, to meet the Scrum (and agile) values and principles, there are a few important things to think about [7]:

- Serve its practitioners and not vice-versa!
- Track and coordinate development rather than trying to control developers
- Respond to change rather than trying to prevent it
- Strive to be transparent and 'frictionless', automating as much as possible
- Frequent and fast coordination and automation
- Eliminate waste - add nothing but value
- Simplicity - the art of maximizing the amount of work not done - is essential (Agile manifesto)
- Working software is the primary measure of progress (Agile manifesto)
- Lean Documentation and Traceability
- Continuous and Visible Feedback on Quality, Stability, and Integrity

In short: add nothing but value and strive to be transparent!

SCM is a 'whole team' responsibility

One of the most profound differences is that instead of being the responsibility of one or more Configuration Managers, SCM will be a 'whole team' responsibility. Scrum wants to avoid the barrier between different roles and make everyone work together towards the same goal of successfully meeting the project's business and technical objectives. Everyone should be responsible for the whole system and by working together the team will join their skills and create better software with higher quality. This should of course also apply to SCM!

SCM should be part of every team member's day-to-day tasks and activities. Everyone is responsible for regularly integrating, building, and testing in their workspace before committing their changes. If the build breaks, the whole team takes ownership of getting it working again.

Automate as much as possible

In Scrum you are working in very short iterations or Sprints. Because of this, you will in most cases test- and build your system several times a day, make a release every two weeks and also deploy to production very often. Therefore, automating these events will be a key factor for a successful SCM functionality.

Despite the fact that many activities will have to be automated, it's important that SCM tools and practices/processes doesn't get in the way of development. Achieve this by minimizing your artifacts, adding nothing but value, eliminate waste and center on the people who add value.

Educate the Scrum practitioners

Since there are no traditional Configuration Managers, there is an obvious risk that some essential SCM tasks are forgotten or ignored due to lack of knowledge and experience. The Product Owner, the Scrum Master and the Scrum Team will need more education about the benefits of having a working SCM-functionality and it is vital that traditional Configuration Managers, or at least SCM experienced people, are near to assist the practitioners when problems arise. In Chapter 4, we will also look at some SCM related questions to ask yourself, as a help when getting your SCM-functionality started.

If SCM is done badly by unexperienced members, it can slow down the development, frustrate the developers and limit customer options. If SCM is done 'right' however, it can give you: reproducibility, integrity, consistency, coordination and a better quality.

3.2 The SCM-expert's role in Scrum

In Scrum there are no defined rolls within the team. In addition, the team should perform all the work required to meet the Sprint goal, which also includes SCM-tasks. So where does that leave the traditional Configuration Manager?

In traditional software development, the Configuration Manager (CM) is a central part of the development process. The CM is the one who carries out all the work. It is for example his or her job to build the system, make sure that all artifacts are under control, that all the right parts are included in a release and that the overall SCM-functionality works smoothly for everyone.

In a Scrum project, the daily tasks performed by a traditional CM is likely to be divided into separate parts; one for the Product Owner and one for the Scrum Team (and perhaps also one for the Scrum Master).

Although, some might argue that regular developers cannot do the work performed by a CM; they don't have time, they don't have the knowledge and (perhaps foremost) they don't have any interest in SCM concerns. In some shops it seems like the development team and the Configuration Managers are two species that doesn't get along very well. Developers see CMs as overly formal, rigid and bureaucratic. People that just gets in the way of the actual work - writing code. CMs see developers as undisciplined or ignorant of SCM concerns, constantly compromising product quality or integrity in the name of schedule or development speed.

As mentioned in section 2.4.4, one of the core values and advantages in Scrum is having a self-organizing and cross-functional team. Scrum wants to avoid these barriers between roles and make everyone work together towards the same goal. Similar disputes- and misunderstandings can of course arise between more roles than just CMs; testers, designers, security-experts etc. can all have a preferred way of working and consider the other's way as totally unreasonable.

As a result of Scrum's cross-functional ideology, the work performed by CMs will probably change quite radically.

One possible scenario for CMs is to be a permanent member of the team. They will be the expert on SCM tasks, but will also contribute on design, writing test, source-code, documentation - everything that is needed to meet the Sprint goal. They will also educate others, so for example everyone in the team knows how to build the system. Just like former testers will educate and help former CMs when needed.

One other possible scenario is for the CMs to be a SCM-coach or in-house consultant/service provider. They will assist and help the people involved in development with SCM-related

problems, both the Scrum Teams and the Product Owners.

The combination of these two would also be a possibility (and in many cases perhaps the best approach). In the beginning of the project, the CMs work closely to the team, helping them to set up the appropriate infrastructure, probably including source code control, a build machine, continuous integration etc. They stay with the team and educate them until they know enough to handle the daily SCM routines them selfs. After that, the CM can be available for another team, and/or be of assistant when problems arise.

In conclusion, the former Configuration Manager's knowledge will still be of great value and an important asset also when working with Scrum. However, their work description will probably not be the same.

3.3 Traditional SCM in Scrum

In the section 2.2 we talked about the Traditional SCM-activities; Configuration Identification, Configuration Control, Configuration Status Accounting, and Configuration Audit. In this section we will return to these activities, but this time, in a discussion about if or how they can be handled in a Scrum project. The analysis are based both on theory and on interviews.

3.3.1 Configuration Identification

Much like in traditional development, the Configuration Identification will be the first major SCM function even in a Scrum project. It will not be as heavy though, since the team only plan for one short Sprint (2-4 weeks) at a time. On the other hand, the activity will be done repeatedly during the project (in the beginning of every Sprint), compared to only ones in for example a waterfall method. In addition, Scrum strive to minimize their artifacts, adding only what will bring business value. Therefore, it might not be necessary to write artifacts like a project plan, design documents, test plans and a SCM plan. Because of these reasons, the Configuration Identification is likely to be a much smaller, and simpler, process.

I believe that applying a Version Control System forces the selection of configuration items; all artifacts that are important for the project become configuration items and goes into the shared repository. This should be done as early as possible so the information about them can be shared among the team.

3.3.2 Configuration Control

In Scrum, there is no Change Control Board (CCB) like in a traditional SCM environment. Instead, change requests from the Product Backlog are discussed, prioritized, and decided by the Product Owner in the Sprint Planning Meeting. The team then takes the top priority items and puts them in the Sprint Backlog. The amount of items is based on the length of the Sprint and the complexity of the changes/features. In my opinion, this functionality can be described as a light-weight CCB. Much of the same activities including; evaluation, coordination, approval or disapproval are indeed present, only far less formal.

Two other important aspects of Configuration Control are tracking and traceability. This could be for instance the possibility to trace the changes made to a file back to the specific change request or user-story. Likewise it can be the possibility to trace the files (source code, test cases, documentations) that were changed/created when implementing a certain change request or user-story. This will further be discussed in section 3.4.4.

3.3.3 Configuration Status Accounting

According to one of the principles behind the Agile manifesto [6]:

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

Also in Scrum, much of the Status Accounting are done with face-to-face conversations. Every day a short Daily Scrum meeting is held where all stakeholders can come and listen to the progress; which user-stories the team are working on at the moment, how much is done, and how much is left until the end of the Sprint. They can also hear if anything is interrupting their work. The task-board is fully visible at all times, and a Burndown chart is used to illustrate the Sprint progress. In addition they can take a look at the Product Backlog to see what will be implemented in the future. At the end of every Sprint (about every 2-4 weeks) the Product Owner (and other stakeholders) can come and listen to the result and get a demo of how the system works so far. Since the items in the Sprint Backlog are limited, it's easy to keep a list of what new features and changes were made in the different Sprints, and to write the corresponding release note.

In some projects however, it can be impossible to get together in a daily face-to-face meeting, for example when different teams are speared geographically. Then, the use of a tool, specially designed for Scrum, can help coordinating and keeping everything in one place. All information should then be visible both to the Product Owners and the teams; the Product Backlog, the Sprint Backlog and the (automatically generated) Burndown chart.

Most of the time, this status transparency seems to be enough for the stakeholders. However, the Scrum Master has to operate within the culture of the organization, and in some projects the management or the customer may require additional information. For example, they may want to be able to generate statistics about speed, the amount of code written, how much time it usually take to fix a bug or implement a feature, and how much maintenance it has generated. It can also be that management are unable to come to the Daily Scrum meetings, and instead requires a weekly progress summary. It's important though to be aware of the fact that all these reports comes with a price, and because of this, make sure not to produce status reports that no one will ever read.

3.3.4 Configuration Audit

In traditional development, the Configuration Audit activity is a formal verification that all the required items are present and that they all meet the requirements. The core objective of the audit are also present in the Scrum method, however it is as expected far less formal.

The Configuration Verification and Audits are done at the Sprint Review Meeting. The Sprint Backlog contains all the required items for that delivery and since the team has only been working for 2-4 weeks, the list is usually quite short. The team demonstrate each story in the Sprint Backlog to the Product Owner and he or she can than see that (or if) the items satisfies the requirements. Thus, both the physical- and functional configuration audits are done in an informal and face-to-face manner. In some occasions however, it's possible that the Product Owner requires more than just a demo. This could for example include test results and/or written documentations.

From my interviews I've heard that the in-process configuration audit is not a common task in the Sprint Review (unless the Product Owner requires it). Instead, this can be discussed in the Sprint Retrospective meeting.

In conclusion, I think it's fair to state that the four Traditional SCM-activities are in fact present in the Scrum method but are, not surprisingly, handled far less formal. All four activities are instead done in a face-to-face manner, which hopefully will lead to enhanced corporation between the roles and increased understanding of each others concerns.

3.4 Developer oriented aspects of SCM

The traditional SCM activities are mainly directed to the managers - how to keep track and organize the project, thus aimed toward the Product Owner in Scrum. There are however also developer oriented aspects of SCM, as mentioned in section 2.3. These activities will be the responsibility of the Scrum Team to handle, and includes: Version Control, Workspace Management, Build Management, Change Management and Release management.

In the Theoretical Background chapter, we only got a short overview about what the concepts are about. Here we will instead talk about just how the Scrum Team can use them.

Since the Scrum method doesn't say anything about how to develop software, none of these techniques will - or can - be specific only to a Scrum Team. The techniques will however be of use for them, if they choose to embrace the benefits of a successful SCM-functionality.

3.4.1 Version Control

The main purpose of Version Control is to manage different versions of configuration object that are created during the software engineering process [8]. A Version Control System (VCS) keeps track of all changes to every file and supports parallel concurrent development by enabling easy creation of variants (branching) and the later re-integration of the variants (merging). Every type of object that evolves in the software development environment should be Version Controlled and kept in the repository. This does not only include source code, tests and documents, but also binary objects such as word-processor documents, executable programs and bitmap graphics objects.

Besides keeping track of changes to a file and enable parallel development, a Version Control System can roll back to a previous version of a given file in case of a problem or for debugging. It can compare two version of a file, highlighting differences to see what changes are made, and it will maintain an instant audit trail on every file: versions, modified date, modifier, and any additional amount of meta data your system provides for and that you choose to implement [2].

Thus, Version Control gives the ability to trace the history of all CIs in a system and to re-create any previous version of a file. When things are not working, the files can be rolled back to previous versions and the changes made after the successfully working version can be inspected to find out what is causing the error or malfunction.

One of the keys to agile development is knowing what is happening. The SCM repository needs to be a communication tool. If you can't put your code change in the repository when

you have done your unit-testing, only you know its state, and others have to work from an older state of the corresponding files. This complicates everyone's work, forcing additional parallel changes and merge/test cycles. Make sure your process will allow developers to put code into the repository when it is ready, and not when someone else is [9].

3.4.2 Workspace Management

As mentioned in the previous section, the Version Control System typically have a repository where all versions of the system are kept. Since this repository is shared by all developers, the individual members can't work directly within it. Instead, they place a copy of the system in their private workspace, make the desired modifications, and then add the changes back to the repository. As a result, several developers can work in parallel without disturbing each other.

The more attention you pay to setting up the workspace, the better you will be in the long run. Effective private workspaces give you the following advantages [10]:

- **Productivity:** By providing a way for developers to control how and when they start working with new code you allow them to focus on the task at hand and minimize interruptions in thought. This helps the individual. Likewise by providing build and test processes that developers run pre-commit, you catch integration issues before they make it into the repository, and improve the productivity of the team.
- **Progress:** By enabling frequent integration and testing, you give developers more confidence in the code base. This will enable them to feel more comfortable about integrating frequently, and moving forward in a regular fashion.
- **Accountability:** Developing build and integration processes at the beginning of the process allows you to get the application to a point where you can demonstrate it to stakeholders earlier.

The integration between the workspace and the Version Control System will also enable you to use Continuous Integration (CI). CI allows all developers to benefit from changes as soon as possible, and it allows you to integrate changes from the rest of the team for early detection of incompatible changes. This will reduce complex integration problems that are common in traditional projects that integrate less often. According to Farah [9], Continuous Integration is one of the biggest gains of an Agile development process, which when properly administered leads to greater product quality. For more information about Continuous Integration, I can strongly recommend Martin Fowler's article [11].

Setting up workspaces for your team may require a bit up-front work, but the increased rate of progress will make it well worth the effort.

3.4.3 Build Management

There are mainly three types of different builds [12]. The Private Developer Build, which provides a consistent way for the developer to build the software in the confines of their private workspace. The Team Integration Build to synchronize the team and give feedback on code quality/integrity. Last the Formal Release Build which creates the deployable package.

In section 3.4.2 we talked about the benefits of having a properly set up private workspace. In Scrum, as well as most agile methods, we develop in short iterations and will therefor have to integrate the work done by the developers much more often than when using a waterfall method. Because of this, it will cost effective and probably also necessary to automate the integration build process.

Benefits of having an automated build process

Ideally, you will rebuild every time someone makes a commit to the shared repository. Here are some of the benefits [13]:

- **Minimize risk:** find out about incompatibilities when they occur. This makes it easier to manage these incompatibilities and easier to fix.
- **Improved quality:** The product is always in a usable state, so less time is spent dealing with quality issues later in the development cycle. This also allows testing to start very early in the development cycle, making it easier to deal with usability issues that testers may raise. When a bug is fixed, testers are able to verify that fix quickly.
- **Speed up bug fixing:** it is always easier and quicker to fix bugs when they appear rather than later, when more code may have to be examined to find the bugs.

There are no hard and fast rules on when and how the builds should be done, however, here are some suggestions to help you on your way [13]:

Build all, or not at all: "The Daily Build" should build the entire project, compiling every line of code in the project and produce every executable, dll or library. In other words, the build process should be producing all the deliverables that can be produced at the time. This avoids issues involving compatibility between dll's etc.

If the build fails, fix it and then build again: the longer issues that cause a build to fail are left, the harder they become to resolve. Fix the problems (be they erroneous code or design issues) as soon as possible, don't leave them on some TODO list waiting for someone to remember to do it.

Label successful builds in your Version Control System: if your VCS supports the concept of labeled versions, then make use of this feature. This make it easier to revert to a "known good build" when things go horribly wrong or when some weird impossible to find bug appears.

Archive the binaries for each build: This will help you find those weird an impossible to find bugs. You can simply test the applications to find out when the bug began appearing, and along with labeled versions in your VCS, will give you a starting point for code changes that may be the cause of the problem.

Use robots, not people. Humans are not very good at repetitive tasks, often performing steps out of order, or missing steps altogether. An automated build process overcomes this by being able run the build process in a repeatable manner. And it doesn't get bored easily!

A way to automate the build process

There are many approaches to get a successful Build Management functionality, and I will here talk about one possible way. For more information, I can recommend the book "Ship It!" [14].

Develop in a private workspace

The first thing to have is a proper workspace where you can play around without disturbing other developers. In addition to the developer machine and the repository, you have a build machine. The build machine is an unattended server that simply gets all of the latest source code from the repository, builds, and tests it, over and over again. The result of this build is the product release. Most of the time, this release will just be thrown away after each build, but every so often this is will be what you ship to your customers and end users. It's built the same whether it's a regular daily build or it's the final release build.

Script your build

A build converts source code into a runnable program. Depending on the computer language and environment, this may mean you're compiling source code and bounding images and other resources together as needed. Remember we're not talking about compiling/building within your IDE. We're talking about a build on your own machine that parallels the "official" build on the build machine.

You can build your product in a variety of ways. You could have a list of steps that looks something like this:

1. Compile your code.
2. Copy the compiled code to your installer program.
3. Move the latest copy of your third party libraries (e.g. database drivers and parsers).
4. Drop in your non-code files such as HTML, graphics and documentation.
5. Copy over your help files to the installer.
6. Build the installer.

You should be able to run the build script just by typing a one-line command (for instance: `ant buildinstaller or make all`). You have a problem if you do anything by hand in your build or packaging process. This will be a wise investment of your time early in the project.

If you're using your manual build system properly, you will be able to build your entire product:

- With one command
- From you Version Control System
- On any team member's workstation
- With no external environment requirements (such as specific network drivers)

Build Automatically

An unattended build is an automatic one. However, before you can implement an automatic build, you must already have a manual build system in place that you can run with a single command. You can't automate a process that doesn't exist.

Ideally, you will rebuild every time the code changes. That way you'll know immediately if any change broke your build. Add a light weight set of smoke tests to this system, and you also get a basic level of functional insurance as well. You'll move beyond 'Does it compile?' and also ask 'Does it run?'.

With a well selected test suite, basic functionality is retested and bugs are not allowed to be reintroduced. The immediate feedback on every code change catches problems quickly so that issues can be fixed quickly. If you're using an automatic build system, you are far ahead of most software teams. But there are still a few questions you need to ask yourself:

- Do you have test in the system? After all, no one cares if it compiles if it doesn't run.
- Is anyone paying attention to the system? Are the notifications turned on?
- Does your build get fixed quickly or stay broken for days?
- Does your build finish in a reasonable time, or does it take too long to complete?

3.4.4 Change Management

As mentioned in section 2.2.2, traditional Change Management is handled through a formal Change Control Board that must allow the change before it can be implemented. This is not the case in Agile development though. In Scrum, changes (as well as new functionality) are approved and prioritized by the Product Owner at the Sprint Planning Meeting.

Every day, you have a Daily Scrum meeting in which each team member must answer the following three questions:

- What have you done since the last Daily Scrum?
- What will you do between now and the next Daily Scrum?
- What got in the way of doing your work?

Issues that arise which may impact a task are raised and added to ether the Sprint Backlog or the Product Backlog, depending on the magnitude of the issue. Change Management that successfully meets the need of an agile project must thus focus on close collaboration via informal face-to-face communication, and on keeping processes and tools lean and simple [15].

The first step toward agile Change Management is a change in mind-set! Those who think of change control as preventing changes to an agreed upon 'baseline' of project/product scope must change their mind-set to embrace change as a natural and expected part of development. The focus needs to shift from preventing change, to managing expectations and tracking changes. Instead of saying "No!" to a change request, the "agile" answer should

simply inform the customer of the associated risk and impact upon the currently agreed upon cost, schedule, scope and quality, and then let the customer make an informed decision. This is what the fourth 'value' of the Agile Manifesto [6] implies when it espouses "Responding to change over following a plan".

A way to make Change Management easier

Agile Change Management must help us do these two things [15]:

- Be more responsive to change requests
- More quickly and easily implement those changes

Be more responsive to change

The Scrum method enable projects in several ways to be more responsive to changes. By authorizing and empowering the Scrum Team to correct problems with the code's behavior and structure, the waiting time for a formal change proposal to be approved is eliminated. By having a Daily Scrum meeting, developers can raise larger problems to the backlog and make everyone in the team aware of it. Having short Sprints minimizes the time between specifying a change request and the implementation of it. The Team and the Product Owner will meet at the Sprint Planning where change-control decisions can be made and communicated quickly and communicated face-to-face with minimal waiting and documentation. The team will inform the Product Owner of impact and risk, and puts him or her in control of the scope and priorities.

Quickly implement changes

Agile methods enable projects to more quickly and easily implement changes by [15]:

- Working in short but complete cycles on the smallest possible 'quanta' of work with tight feedback loops (e.g., short iterations, test-driven development, continuous integration, etc).
- Mandating simple design, and emergent design methods (like refactoring) to make the code as simple and easy as possible to change
- Minimizing the number and size of non-code artifacts that must be produced or updated in order to implement a change
- Working only on the features scheduled for the current iteration

Perhaps the single most influential practices that agile methods provide for Change Management are the use of incremental and iterative development to quickly deliver very small but very frequent executable milestones and obtain customer feedback on the results. In addition, close customer collaboration to promote daily interaction and face-to-face communication between customers and the development team.

Traceability

Traceability between artifacts in a software development project can provide several benefits including: validate that the right system (i.e. what the customer paid for) was built, identify and isolate a set of changes that were made (e.g. for debugging), maintaining an audit trail about who did what, when, where, why and how, and for various status reports. However, Traceability can also be a very heavy- and time-consuming process for several reasons:

- It can impose unnecessary restrictions upon the speed and productivity of development
- It can make it harder to make necessary changes and encourages rigidity rather than flexibility
- It can encourage 'analysis paralysis' and heavy 'up front' planning and design
- It goes against the values in the 'Agile Manifesto' because traceability emphasizes comprehensive documentation over working software and contract negotiation over customer collaboration

A lean approach to Traceability should instead aim to work in accordance with Agile values, Lean principles and the basic tenets of sound SCM, without introducing any friction to the development process. Several good articles describe ways to achieve this:

To easily trace changes without overhead and heavy-handed process, the use of change-packages is the number one practice according to Joe Farah. It makes developer's work easy and it makes it easier for them to trace back through history looking for specific changes [16].

Along the same line, Brad Appleton et. al. describes in one of their articles [17] how to achieve Lean Traceability by using techniques like Task-Level Commits, Task-Based Development and Test-Driven Development, combined with a basic integration between a Version Control tool and a Change Tracking System. Promising approaches such as Event-Based traceability can also help automate this.

Another recommended article by Brad Appleton et. al. about traceability is "The Trouble with Tracing: Traceability Dissected" [18], where they discuss more about how traceability adds value to the business and how you can make it easier.

3.4.5 Release Management

In Scrum, you develop in short iterations and every other week you have to produce a potentially shippable release. Because of this, it will be very important to have a good release process set up. A successful Release Management functionality will have benefits like: high quality releases, be able to make quick and accurate release builds, a repeatable process for deploying releases, and as a result getting cost-effective releases.

In the last section we just briefly mentioned the Release Build as one of the three builds (the other two were the Private Developer Build and the Team Integration Build). Typical steps of a Release Build could include [12]:

- Apply a label to the code line
- Get a copy of the code line (workspace or instance) to your build machine
- Build the application
- Possibly test the application and report on those tests
- Package the targets of the build by either checking them into the Version Control Tool and label as appropriate
- Zip up the release for distribution, or possibly even run installer software against the release to package it for distribution
- Email a report showing the results of the build

While the integration and release builds are similar in many ways, the primary difference is that the release build is intended to package and store the build results for installation and distribution purposes. The integration build is intended to test the quality of the code line and offer feedback to the development team while the recipient of the Release Build is typically the developer's customer.

To be even more specific about the differences between an Integration Build and a Release Build, the following points are worth noting [12]:

- Integration Builds typically operate on the latest (tips) on the branch, not always the case with a Release Build
- Integration Builds run exhaustive tests on the build, not always the case on a Release Build
- Release Build packages the release for installation and distribution, mostly not the case with Integration Builds
- Release Builds include automated SCM reports like BOM (bill-of-materials) and CR (change-request) reports - rarely the case with Integration Builds
- Release Builds occur only when the development team intends to package a release of software for distribution (Test or production environments), and thus occur far less often than the typical Integration Build.

Benefits of having a good release process

Some of the benefits that will come when having a good release process can include that:

- You can quickly make a release, and meet release deadlines
- You know exactly what went into a particular release, and by then you can also make sure that all the right versions of pieces shipped or was published
- Your release will have a better quality

According to Robert Cowham [19], the production of Release Notes is a key indicator for process health. These are typically a list of changes: bugs fixed and features added. They are very useful to everyone to understand what is in the release, how it differs from previous releases, and whether a particular problem will be fixed by upgrading or not.

Some ways to make the release easier

The Scrum method alone will make your team better at making releases. Every two weeks you will have to produce a potentially shippable release, and because of this continuous practice you will improve you skills and constantly be forced to address problems and fine tune the release. Compare this to a waterfall method where you perhaps only produce a release ones a year.

Using SCM patterns can also make the release easier. These can include using a Version Control System, Task-Level Commits [20], Test-Driven Development [21], Continuous Integration [11] etc.

Automation is a basic requirement for frequent releases. Automation of unit testing in agile methods leads to much greater confidence that the system works and that bugs have not been introduced. Automated builds catch integration problems early making them much easier to fix. Automation of the production of release notes makes the project much more controlled and less subject to surprise. Automation of installs and upgrades are also key requirements for systems these days.

4 Results

The biggest Software Configuration Management-related problem in Scrum project seems to be the overall lack of knowledge about what it is, how to use it, and its benefits. All of the people that I interviewed agreed on this.

In Chapter 2 we got an overview about what Configuration Management is, and about its benefits. In Chapter 3 we talked about both how the traditional- and the developer oriented activities could successfully be handled in a Scrum project. Based on the analysis in Chapter 3, I have written SCM-related questions for the Scrum practitioners to answer. By answering these, the goal is to find out what actions you have to take to get a better SCM-functionality in your project. The questions are divided according to role; Product Owner, Scrum Master, or Scrum Team. I have also formatted the questions in this chapter into three checklists, that can be found in the corresponding appendices.

This will thus be the result of my thesis. An aim to convince and help these unexperienced Scrum practitioners to decide on an appropriate SCM-functionality - for their specific project! This is an important thing to remember; the checklists are not in any way complete for every project. Furthermore, they are not intended to be a step-by-step way to know what you need to do - but rather a step-by-step way to know what you need to think about. The idea is that they should be a good start and it shall inspire the Scrum practitioners to start think about, and discuss, SCM in a practical way.

4.1 The Product Owner

As we saw earlier, the traditional SCM activities are mainly directed to the managers, and thus, aimed toward the Product Owner in Scrum. The SCM related questions that a Product Owner should consider will because of this be about how to keep track of the project, make sure you get everything you need, and if you need to put any organizational constraints on the team.

4.1.1 Configuration Identification & Control

Do you need to put any of the following constraints on the Scrum Team:

Version Control Tool?

The project might be part of a bigger project within the organization, and it may therefore be necessary to use the same Version Control System (e.g. Clear Case), for integration reasons.

Repository structure?

For the same reason as the previous question it might be necessary to have a certain repository structure.

Identification Conventions?

The organization might have a preferred way of describing the identification (numbering) criteria for the software and hardware structure, and for each document or document set.

Naming Conventions?

There might also be an organizational standard about which file naming convention to be used in projects and how file configuration integrity should be maintained.

Labels?

Describe the requirements for labeling media and application software.

Branch Strategy?

Does the team have to use any specific Branch Strategy?

4.1.2 Configuration Status Accounting

Is the Product Backlog and the Sprint Backlog visible for everyone?

The Product Backlog and the Sprint Backlog should be visible for everyone (all stakeholders) at all times. If teams are spread geographically it could be wise to use a tool to coordinate and keeping everything in one place.

Is the Burndown chart updated on a daily basis?

The Burndown chart should be updated every day in order to illustrate an accurate progress.

Does other stakeholders than the Scrum Team attend the Daily Scrum Meeting?

The Daily Scrum Meeting is an excellent way to get a brief verbal update about the progress of the project. It's good if other stakeholders (e.g. management, customers etc.) attend the meeting as well. This in order to avoid unnecessary progress reports.

Is the Status Accounting enough for the organization?

In some organizations, the transparency Scrum provides (highly visible Product Backlog, Sprint Backlog, Burndown Chart, Daily Scrum Meetings etc.) are not enough to meet all stakeholder's interests. This might be the ability to product certain statistics, especially when dealing with critical systems. It can also be that managers doesn't have the time or possibility to attend Daily Scrum Meetings or Sprint Reviews, and would instead value (for example) a short written summary after each Sprint.

4.1.3 Configuration Audit

Do you need to have Traceability?

Does the benefits of having a certain degree of Traceability overweight the cost?

Often organizations seems to need Traceability between code and requirements because 'it is good to have'. But when asked, no one ever uses it. Be aware of the cost and be sure to consider the Return On Investment (ROI) before deciding that the team needs to keep a certain degree of traceability between their artifacts.

One reason for having more traceability can be in critical complex systems where it's essential to find bugs very quickly.

Different degrees of traceability includes:

- Requirement -> User-story
- User-stories -> Sprint (Common Release note)
- User-story -> Tests
- Files -> User-story
- Lines of code -> File

Do you need to have Documentation?

Ask yourself:

- Does the benefits of having a certain degree of documentation overweight the cost?
- Who will read it?
- Will it be difficult to understand the system without documentation?
- Are there any other better ways to transfer the information found in a traditional documentation document, e.g. by having a meeting?

Different types of documentations include: Architecture, Design, Technical, User manual, Test, Marketing, Maintenance, Deployment and Configuration.

Does the team know how the system should be delivered?

For example if the delivered increment after each Sprint should be able to install by a update/patch or if it should be a full product (monolith deploy).

Does the team know your 'Done'-criteria?

That a user-story or a new functionality is 'done' can mean many things. Does it include tests, documentation etc.? Make sure you and the Scrum Team speak the same language.

Do you get everything else you need on the Sprint Review?

You might need to get test results and/or printed documentation etc.?

4.2 The Scrum Master

The Scrum Master's main responsibility is to ensure that the practitioners can work in the most efficient way possible. This will also include efficient use of SCM. The questions will thus focus on helping, inspire, and coach the team as well as the Product Owner to do the best work possible.

Is the team cross-functional?

As mentioned in section 2.4.4, one of the core values in the Scrum method is having a cross-functional team. This should of course also apply on SCM, and thus, there should not be a designated Configuration Manager.

Does the team have basic knowledge about SCM?

If the Scrum Team has very little knowledge about SCM, it might be good to take on less new functionality in the first Sprint and instated time-box education and/or workshops. An experienced Configuration Manager can for example help the team implement basic functionality and make sure all team members learns how to use it. Convince the Product Owner that this will pay of later.

Does everyone know how to handle the daily SCM-tasks?

All team members should for example know how the Version Control System works and how to build the system. Since there will not be a designated Configuration Manager to handle the builds, the team members them selfs must learn how to do this.

Does the team have all necessary hardware and software for their development infrastructure?

For example in section 3.4.3, we saw some of the benefits of having an automated build process, and that you might need a separate build machine.

Do we follow up the SCM activities on the retrospective meeting?

Discuss your new SCM activities with the Scrum Team. Are they helpful or not for your project, and what can you do even better the next Sprint?

Have you done everything else to help the team with potential SCM-problems?

If you see a potential SCM-problem that the team has not though of - raise the question for discussion and talk about what you can do about it.

4.3 The Scrum Team

The Scrum Team will preform all the daily SCM activities. This could include the content in section 2.3, i.e. handling the Version Control System and performing Builds and Releases. They must also make sure the customer gets the requested 'SCM-products'. The first thing to do when implementing your SCM-functionality would be to find out exactly what you need to have in your specific project. The questions in this section aims to help with that.

The Sprint Planning Meeting can be a good opportunity to discuss new SCM functionality. Bigger implementations should be time-boxed in the Sprint Backlog - just like any other new functionality. The implemented SCM functionality can/should later be discussed at the Retrospective Meeting; What went good? What do we need to change or remove? What can we do even better during the next Sprint?

4.3.1 Version Control

Do you need to roll back your server or your program to recover from a bug in a short amount of time?

This is something that all project will encounter sooner or later and a Version Control System makes this much easier.

Can the whole team work simultaneously?

It is of course vital for high effectiveness that no one have to wait for another developer before they continue- or start with a new task.

Do you immediately get access to other's new features or bug fixes as soon as they are done?

When using a Version Control tool you will get instant access to your team member's changes, as soon as they check it in to the shared repository.

Can you create multiple branches at need?

A good Version Control tool should allow the developers to create branches in order to develop in different ways independently of the other (e.g. one branch for each supported platform; Windows, Unix, Macintosh etc).

Does everyone in the team know how to use the tool effectively?

For this to work, it's essential that all developers know what they are doing. They should have a good process about check-in and check-out procedures, know how branching works etc.

4.3.2 Workspace Management

Is your development machine optimal for your productivity?

For example, you should be allowed to use the IDE of your choice. If not, is there a good reason that everyone in the team uses the same IDE?

Can you develop without disturbing other developers?

Can you work with your code uninterrupted from other team members, and can others work with their code uninterrupted from you?

Do you have a good integration/synchronization with the Version Control System?

You should be able to check out code from the repository, play with it until it works satisfactory, and then check it back in.

Is it rare that you spend a lot of time integrating your system?

If you often have to spend days integrating the system, you should consider implementing Continuous Integration in your project. You will then be able to integrate changes from the rest of your team as soon as they are ready for early detection of incompatible changes.

Is it rare that you have a lot of bugs after integration?

Likewise, if after integration you spend days correcting bugs, you might want to consider the benefits of a CI system.

Is it rare that outstanding bugs remains undetected in the system for a long time?

By integrating, and running test several times a day, it is a much higher probability that you find bugs as soon as they are introduced. In addition, if they are discovered quickly, they will also be much easier to solve.

4.3.3 Build Management

Can you build your private system in your sandbox easily before checking in?

Every team member should be able to do a private build before checking in their code into the repository. If not, lots of bugs will infect the repository and spread to the other team members. You should be able to run a build script with a single command, if not, it's a risk that most of the team won't bother.

Can the system be built on any team member's workstation?

The build script should also stay under Version Control and work on all workstations.

Are the build process automatically executed when someone checks in a new task?

Ideally, an Integration Build should run every time a new functionality are checked in to the repository to provide quick feed-back to the development team.

Does everyone in the team know how to build the system?

The Scrum Team should be cross-functional, and this should of course also include builds. Besides, it could be very dangerous if only one member know how to build-, or release the system.

Does the team receive information when a build fails?

It has no value to build after each commit if no one notice that a build failed.

Does your build get fixed quickly?

It has no value to build after each commit if builds stays broken for days.

4.3.4 Change Management

Can you respond quickly to changes?

If an important change request arise, can you have it implemented at the end of the following Sprint? Or in the case of a more critical change, terminate the Sprint and start working on the change right away?

Can you implement the change quickly and easily?

Does it often cause a lot of work to implement a change, e.g.: a lot of artifacts need to be updated etc.?

Have you considered a practical degree of traceability between artifacts?

Most project at least keep traceability between Sprints and User Stories, i.e. release-notes. As mentioned in section 3.4.4, more detailed traceability could cause a lot of work, but it doesn't have to, and it can be very beneficial for some projects.

4.3.5 Release Management

Can you quickly do a release?

Since you have very short iterations in Scrum (e.g. 2 week Sprints), it should not take long to do a release at the end of the Sprint. The release process should be automated.

Do you know exactly what went into a particular release?

It could be very bad if you don't keep this under control. However, in Scrum you just focus on a few top-priority features each Sprint, so the number of artifacts to keep track of is much smaller than in, for example, a two year waterfall project.

Can you release multiple variants simultaneously?

Your build-, and release script should be able to easily create multiple variants of a release.

Can your end-users easily install your software?

Does the release contain a user manual, or a small instruction of some sort? Or will it be obvious for all users how to install your system?

Does several people in the team know how to release the system?

It could be very dangerous if only one person in the team knows how to release the system.

4.3.6 Naming Convention

Do you have a naming convention defined within the team?

You should define a naming convention and make sure everyone in the team uses it. The artifacts will be easier to keep track of and understand.

Does your naming convention follow the general organizational standard (if any exist)?

Sometimes the company have a general naming convention. Talk to the Scrum Master if you have any doubt about it.

Does the Product Owner have a preference about it?

Your project might just be one part of a bigger project. In that case the, Product Owner could inform you about which naming convention is preferred.

4.3.7 Branch Strategy

Does your team have a Branch strategy defined?

Discuss with your team, and decide on a good branch strategy for your specific project.

Does the Product Owner have a preference about it?

Perhaps there is some organizational standard, or a preferred way to handle branches. Ask the Scrum Master to find this out.

5 Conclusions and Future work

I found that the traditional Software Configuration Management activities are in fact included in the Scrum method, although quite different and much less formal. In general, Scrum (as other agile methods) tends to value face-to-face information transfer instead of formal reports, and this is very much reflected in the Scrum approach to the four traditional SCM activities.

To answer the first of my questions: "Is Scrum complete in a SCM perspective?" the answer would be no. The Scrum method does not mention SCM explicitly, but it also doesn't contradict SCM practices. It should ultimately be up to the empowered and self-organizing team to figure out the optimal (and Lean) way. However, in some organizations, the common informal way of dealing with SCM in a Scrum project might not be enough to meet their standards.

From my interviews I learned that often the developer oriented aspects of SCM, such as Version Control, Workspace Management, Build Management, Change Management and Release Management, are not being considered much. The main reason for this is lack of knowledge.

This leads us to my second question: "How can SCM practices provide service and support to Scrum projects?". To get unexperienced teams started with the SCM-functionality I have written checklists for the three different roles in Scrum; the Product Owner, the Scrum Master, and the Scrum Team. It was necessary to split them up because the different roles will benefit from SCM in different ways. It is important to stress that these checklists are not intended to be a step-by-step way to know what you need to do - but rather a step-by-step way to know what you need to think about. I have also written recommendations about how to implement the selected SCM-functionality in an Agile- and Lean fashion that should fit well within the Scrum method.

The Configuration Manager will still be an important asset to all the three Scrum roles, but his or her duties will probably shift to be more educational and consulting when SCM improvements are needed and when problems arise.

Further work

The checklists may not complete for every project. It's a good start, but will probably have to be tailored for the specific project.

As mentioned in the introduction chapter, I had no chance to test my result in a real development project, which of course would have been valuable. However, I have showed it to experienced Scrum practitioners and received very positive feed-back.

This thesis was also limited to one Scrum Team. When several teams are working together in a Scrum of Scrums, it will probably add complexity to the SCM-implementation. This scenario would have been interesting to investigate further - perhaps in another master's thesis.

Bibliography

[1] W. Babich, *Software Configuration Management - Coordination for Team Productivity*, Addison-Wesley, 1986.

[2] A. Leon, *Software Configuration Management Handbook*, Artech House, second edition, 2005.

[3] Wikipedia, Change Request, http://en.wikipedia.org/wiki/Change_request.

[4] L. Bendix and T. Ekman, Software Configuration Management in agile development, 2007.

[5] K. Schwaber and M. Beedle, *Agile software development with Scrum*, Prentice Hall, 2002.

[6] Manifesto for Agile Software Development, http://agilemanifesto.org.

[7] B. Appleton, *Agile Configuration Management Environments*, Chicago Software Process Improvement Network (C-SPIN), 2004.

[8] J. Koskela, Software Configuration Management in agile methods, 2003.

[9] J. Farah, *The next generation of Agile CM*, CM Journal, 2007.

[10] S. Berczuk, B. Appleton, and R. Cowham, *Private Workspaces - Where Development Process Meets CM Process*, CM Journal, 2006.

[11] M. Fowler, Continuous Integration, 2006, http://martinfowler.com.

[12] S. Berczuk, B. Appleton, and S. Konieczka, *Build Management for an Agile Team*, CM Journal, 2003.

[13] V. Parrett, Build Process Automation, 2000, http://www.finalbuilder.com/articles.aspx.

[14] J. Richardson and W. G. Jr., *Ship It! A Practical Guide to Successful Software Projects*, The Pragmatic Programmers LLC, 2006.

[15] B. Appleton, S. Konieczka, and S. Berczuk, *Agile Change Management - from first principles to best practices*, CM Journal, 2003.

[16] J. Farah, CM: THE NEXT GENERATION of Top 10 Best Practices, 2007.

[17] B. Appleton, R. Cowham, and S. Berczuk, *Lean Traceability: A smattering of strategies and solutions*, CM Journal, 2007.

[18] B. Appleton, R. Cowham, and S. Berczuk, *The Trouble with Tracing: Traceability Dissected*, CM Journal, 2005.

[19] R. Cowham, B. Appleton, and S. Berczuk, *Release Management: Making it Lean and Agile*, CM Journal, 2004.

[20] A. Hastings, *SCM Patterns: Building on 'Task-Level Commit'*, CM Journal, 2004.

[21] T. Freese, *Toward Software Configuration Management for Test-Driven Development*, CM Journal, 2003.

Appendix A: SCM-checklist for the Product Owner

Configuration Identification & Control

Do you need to put any of the following constraints on the Scrum Team:

- ☐ Version Control Tool?
- ☐ Repository structure?
- ☐ Identification Conventions?
- ☐ Naming Conventions?
- ☐ Labels?
- ☐ Branch Strategy?

Configuration Status Accounting

- ☐ Is the Product Backlog and the Sprint Backlog visible for everyone?
- ☐ Is the Burndown chart updated on a daily basis?
- ☐ Does other stakeholders than the Scrum Team attend the Daily Scrum Meeting?
- ☐ Is the Status Accounting enough for the organization?

Configuration Audit

- ☐ Do you need to have Traceability?
  - ☐ Requirement -> User-story?
  - ☐ User-stories -> Sprint? (Common Release note)
  - ☐ User-story -> Tests?
  - ☐ Files -> User-story?
  - ☐ Lines of code -> File?
- ☐ Do you need to have Documentation?
  - ☐ Architecture?
  - ☐ Design?
  - ☐ Technical?
  - ☐ User manual?
  - ☐ Test?
  - ☐ Marketing?
  - ☐ Maintenance?
  - ☐ Deployment?
  - ☐ Configuration?
- ☐ Does the team know how the system should be delivered?
- ☐ Does the team know your 'Done'-criteria?
- ☐ Do you get everything else you need on the Sprint Review?

If you have any checked boxes - be sure to inform the team what you need and why it is important for the delivery.

Appendix B: SCM-checklist for the Scrum Master

- [ ] Is the team cross-functional?
- [ ] Does the team have basic knowledge about SCM?
- [ ] Does everyone know how to handle the daily SCM-tasks?
- [ ] Does the team have all necessary hardware and software for their development infrastructure?
- [ ] Do we follow up the SCM activities on the retrospective meeting?
- [ ] Have you done everything else to help the team with potential SCM-problems?

Appendix C: SCM-checklist for the Scrum Team

Version Control

- ☐ Do you need to roll back your server or your program to recover from a bug in a short amount of time?
- ☐ Can the whole team work simultaneously?
- ☐ Do you immediately get access to other's new features or bug fixes as soon as they're done?
- ☐ Can you create multiple branches at need?
- ☐ Does everyone in the team know how to use the tool effectively?

Workspace Management

- ☐ Is your development machine optimal for your productivity?
- ☐ Can you develop without disturbing other developers?
- ☐ Do you have a good integration/synchronization with the Version Control System?
- ☐ Is it rare that you spend a lot of time integrating your system?
- ☐ Is it rare that you have a lot of bugs after integration?
- ☐ Is it rare that outstanding bugs remains undetected in the system for a long time?

Build Management

- ☐ Can you build your private system in your sandbox easily before checking in?
- ☐ Can the system be built on any team member's workstation?
- ☐ Are the build process automatically executed when someone checks in a new task?
- ☐ Does everyone in the team know how to build the system?
- ☐ Does the team receive information when a build fails?
- ☐ Does your build get fixed quickly?

Change Management

- [ ] Can you respond quickly to changes?
- [ ] Can you implement the change quickly and easily?
- [ ] Have you considered a practical degree of traceability between artifacts?

Release Management

- [ ] Can you quickly do a release?
- [ ] Do you know exactly what went into a particular release?
- [ ] Can you release multiple variants simultaneously?
- [ ] Can your end-users easily install your software?
- [ ] Does several people in the team know how to release the system?

Naming Convention

- [ ] Do you have a naming convention defined within the team?
- [ ] Does your naming convention follow the general organizational standard (if any exist)?
- [ ] Does the Product Owner have a preference about it?

Branch Strategy

- [ ] Does your team have a Branch strategy defined?
- [ ] Does the Product Owner have a preference about it?