Master's Thesis

Implementing Traceability In Agile Software Development

Marcus Jakobsson D03

Department of Computer Science
Faculty of Engineering LTH
Lund University, 2009

ISSN 1650-2884
LU-CS-EX: 2009-02

Implementing Traceability In Agile Software Development

Marcus Jacobsson
Faculty of Engineering, LTH
Department of Computer Science
Sweden

Date: 2009-02-02

LUND
UNIVERSITY

SOFTHOUSE

Department of Computer Science
Lund University, Faculty of Engineering, LTH
SE-221 00 Lund, Sweden
www.lth.se

Contact information

Author:
© Marcus Jacobsson
E-mail: traceability@hanabi.se

External advisor
Christian Pendleton
Softhouse
E-mail: christian.pendleton@softhouse.se

University advisor
Lars Bendix
Department of Computer Science
E-mail: bendix@cs.lth.se

Abstract

How traceability is handled in agile methods vary from organization to organization, and project to project. When working with traditional development traceability is an important part of the process. However for some reason this is not the case when working with agile methods. For example in Scrum there is no real definition of how configuration management is supposed to be done. Can we add traceability to Scrum and still call it Scrum?

The initial problem for this thesis was to look into how traceability was supposed to be performed in agile methods, do we need traceability? Another problem that came up during the research is the question if traceability needs to be heavy, does it have to add administrative overhead for the team? Does the team need to write documents that risk not being up to date? Is it better not to have tracing information then having incorrect information?

The studies have resulted in a list of practices that can be used to add traceability to the agile methods. These practices can be used by themselves, however the best result will come when two or more are combined. There will be some examples of combining them as well as a larger example on how that will result in a way to trace requirements, tests, code and changes from the beginning to the end.

One of the focuses will be on the support of tools and how they can help reduce the administrative overhead of adding traceability to the project. If tracing is supposed to succeed in agile methods such as Scrum and XP, it needs to add some value to the project. Adding something that will only result in a cost in the end needs to be avoided.

Keywords

Traceability, Configuration management, Agile, requirements, searchable data, software interoperability, costs, benefits, documentation, scrum master, configuration managers

Acknowledgements

I would like to take this opportunity to thank everyone that has helped me with my master thesis. I would like start with thanking everyone that would let me interview them as well as all the people that were actively involved in the discussions at the CMCM[1] in Lund 2008-11-13. I also received some help from people that reviewed my working document and gave me feedback, this was very valuable and therefore a big thanks to all of you that sent me feedback.

In the end I would like to give an extra big thank you to Christian Pendleton and Lars Bendix for all the help they have given me and all the inspiring discussion. A big thank you also goes out to Tobias Olsson for helping me correct the spelling and grammar.

[1] http://www.cs.lth.se/SNESCM/Common/CMCM/

Contents

1 INTRODUCTION 1

2 THEORETICAL BACKGROUND 3

3 ANALYSIS 6

3.1 METHODS AND LIMITATIONS 6
3.2 THE ATTITUDE TOWARDS TRACEABILITY 7
3.2.1 Lost creativity 8
3.2.2 Scrum is complete 8
3.2.3 No one reads it 8
3.2.4 Heavy routines 9
3.2.5 Documentation means documents 9
3.2.6 Administrative overhead 10
3.2.7 Lack of motivation 10
3.3 LACK OF KNOWLEDGE 10
3.3.1 Traceability 10
3.3.2 The whole picture 11
3.3.3 Documentation in agile methods 11
3.3.4 It is expensive or at least it is not free 12
3.3.5 When and what do we need to trace 12
3.3.6 Routines 12

4 TRACING PRACTICES 13

4.1 PRACTICES 13
4.1.1 Stakeholder to requirements 14
4.1.2 Initial problem description to requirements 15
4.1.3 Requirements to story/sprint log 15
4.1.4 Requirements to requirements 16
4.1.5 Requirements to code 17
4.1.6 Requirements to version 19
4.1.7 Requirements to test 20
4.1.8 User evaluation to version 21
4.1.9 Platform information to release 21
4.1.10 Implementation documentation 21
4.1.11 Code and Table tracing 22
4.1.12 Developer to Code 24
4.1.13 What others are working on right now 26
4.2 PRACTICES COMBINED 27
4.2.1 Simple combinations 27
4.2.2 Complete solution example 29

5 GENERAL CONSIDERATIONS ON TRACEABILITY 33

5.1 IN WHICH CASES TO ADD TRACEABILITY 33
5.1.1 Project size 33
5.1.2 Small projects 33
5.1.3 Short time to market 34
5.1.4 Complex project 34
5.1.5 Maintenance 34
5.1.6 Long lived projects 35
5.1.7 Large projects 35
5.1.8 External or internal demands 35
5.1.9 High demands on security and stability 36
5.2 HOW TO ADD TRACEABILITY 37
5.2.1 Processes 38

5.2.2 Saving information *38*
5.2.3 Costs and benefits *39*
5.2.4 Tools *40*
5.2.5 Searchable format *41*
5.2.6 Documentation without documents *42*
5.2.7 Information overflow *42*
5.2.8 Correct and up to date information *42*
5.2.9 Understanding *43*
5.2.10 Discipline *43*

6 CONCLUSION **44**

7 BIBLIOGRAPHY **45**

APPENDIX A: QUESTIONS **48**

1 Introduction

This report focuses on how traceability can be added to the agile methods and what the potential costs and benefits there will be. This report was written as a part of the master thesis in computer science at the department of computer science at the Faculty of Engineering at Lund University. This report is aimed towards people with previous experience in Configuration management and agile methods.

Background

One of the problems is that traceability is an important part in traditional software development but it is not a standard practice for the agile methods such as Scrum and extreme programming (XP). Can agile methods be used in larger project where it was necessary to have traceability? Is the traditional development methods better suited? There was also focus on how traceability was supposed to be added to the methods like Scrum.

Problems

The initial problem was how to add traceability in agile methods such as Scrum and XP. Some organizations, customers and regulations require that tracing information is saved during development. Tracing information about events such as code changes, test results, implemented requirements and so on. The problem is how this is supposed to be done when using agile development. Some questions that came up were:

- Can traceability be agile?
- Can it be added without too much administrative overhead?
- Do tracing need to be expensive?
- Can the project gain anything from traceability?
- Is it possible to use agile methods when developing critical systems where there are requirements on traceability?

Today there are conflict between the configuration managers and the team. The configuration managers want traceability but cannot always present it in a way so that the team will understand why. This results in that the team does not want traceability and conflict is created. There are several reasons for this conflict which will be present as part of the analysis. The problem is however not only limited to these two groups. There is also a problem with the customers, scrum masters and managers being confused about traceability.

In traditional development the hierarchy is stricter then in the agile methods. This result in that the project manager or configuration manger can tell the team that they have to save tracing information and the team have to do it. In the agile methods the hierarchy is not realty there. According to the Scrum principles if the team considers something to be in the way it should be removed, however are tracing only suppose to help the team? Can they really decide if the information is useless for the project?

Another problem is that traditional traceability is often heavy and will create a lot of overhead. In traditional software development the processes is longer (time from the initial problem description to the final product) and the entire processes involves several teams. This is in contrast to the short and flexible ways in the agile methods. So does the team need traceability or is it only unnecessary overhead for the project?

Another problem that was looked at is who would benefit the most from tracing information. Perhaps the tracing information gathered by the team would help someone outside the team in another stage of the software's lifecycle.

There is a problem when tracing information is gathered and never used as well as never updated. Can this affect the final result of the product?

Goals and objectives

The goal of the master thesis was to look into if traceability can be added to agile methods and if so, how could it be done. In order to get a broader view of the problem, some people from the software development industry was interviewed. The following things were done in order to come to final conclusion:

1. Initial research, find the real problem with tracing in agile methods
2. Formulate a first problem description as well as a preliminary solution
3. Receive feedback and improve correct the preliminary solution

If tracing is suppose to be added in agile methods it should not add too much administrative overhead for the team. It would also have to be clear on how the team would benefit from the information gathered. Some of the initial questions that were created in the early stages of the research and that this report will try to answer are:

1. How is traceability handled in Scrum and XP?
2. When should traceability be implemented?
3. What is the cost for implementing this in an agile environment? Is the cost larger than the gain?
4. Does the intended lifetime for the software affect the demands for a more strict requirements plan?
5. Can it increase productivity or at least decrease the time it take to identify and fix bugs?
6. Can traceability reduce redundant maintenance?
7. How can tools help minimize the time/cost of tracing?

Road map

The second chapter contains a *short theoretical background* as well as who is the intended target group for the report. The third chapter, *analysis*, starts with information about the method used as well as some limitations to this study. Then there is some information about the real problem that was discovered during the research. In the forth chapter there are some tracing practices that can be used in order to get traceability working in agile methods, the benefits and costs of these practices are described as well some information about combining them. The fifth chapter *General considerations on*

traceability describes some of the considerations the people involved need to think of when adding traceability to agile methods. The last chapter*, Conclusion* contains the final conclusion on traceability in agile methods. This is a short summery of some of the key points with traceability in agile methods.

2 Theoretical background

The intended target group (not limited to) for this report is people working with (or have some experience with) configuration management as well as scrum masters in Scrum and coaches in XP. This chapter therefore focuses more on my view on Scrum, XP, configuration management and traceability. For those who are part of the target group or needs to refresh their memory there will be some links and recommended reading on the four subjects.

When in the later chapters agile methods are discussed, the main focus will be on Scrum and XP. To get more information about agile development there is a document called the agile manifesto [1].

Traceability

My short definition of traceability is:

> *Traceability is practices and routines used to trace requirements and changes throughout development. The tracing information should be useful for everyone involved in the project and not just the developers. In order to have traceability it needs to be possible to trace in both direction.*

Traceability is supposed to help keep all the information organized and easy to find. One part of traceability is to give all the teams involved in developing a product a way to see how different artifacts are linked together. Besides being able to see how artifacts are linked it is important to be able to trace information and track down decisions that were made. Traceability is considered to be more important and more accepted in traditional development where there is a longer development cycle and where it could be hard to remember the decisions that were made in the beginning of the project (perhaps some of the people involved in the beginning would leave during the project). Even if traceability could be considered important in traditional development it is not always used.

For more information about traceability there are *The ten commandments of traceability [2] as* well as some information about *Lean traceability [3]*.

Scrum

Scrum is a more project oriented approach to agile methods. One thing that is good with Scrum and also the reason for selecting it as one of the two methods to focus on is that it does not focus so much on the development but more on everything around it. This gives a broader view on development.

For more information about Scrum there is a book called *Agile Software Development with Scrum* [4] written by Ken Schwaber and Mike Beedle. This book will describe the advantages of Scrum and how to use it. For further information about Scrum and how to distribute Scrum there is a book called *The Enterprise and Scrum* [5], this also written by Ken Schwaber and will describe how Scrum can be used in larger organization where there could be a need to distribute the workload.

XP

While Scrum focuses on everything around the developing methods XP does the opposite and focuses on the team and developing practices. Therefore XP is a perfect partner to Scrum. XP focuses more on some configuration management practices such as version control, workspace management and build control. Because XP is a good partner to Scrum most of the practices that could be implemented to adding traceability to Scrum can be added to XP. Another reason is that some of the practices in XP could help implementing some tracing practices in Scrum, for example the version-control tool that is introduced from collective code ownership.

For more information about XP there is a book called *Extreme programming explained* written by Kent Beck [6] that will describe the basic practices in XP. There are also several pages on the web that will describe XP. One of them is *Extreme Programming: A Gentle Introduction [7]*.

Configuration management

It is hard to define configuration management. The key parts of configuration management that will be focused on in this thesis are traceability and version-control. These are only two of the components that form configuration management. One of the many definitions on configuration management is

> *A management process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design and operational information throughout its life* [8].

Configuration management focuses both on the teams and development as well as the organization as a whole. Some configuration management roles are focused more on the team and how version-control tools and branching patterns are used. Another role is version-control and document handling in order to support the managers.

For more information on the subject of configuration management there is a book called *Software configuration management patterns* written by Steve Berczuk and Brad Appleton [9]

Previous work on CM and agile software development

For more information about how configuration management can be used in Scrum there is a previous master thesis on the subject, the report is titled *Software Configuration Management in Scrum Projects* written by Andreas Bergström [10]. There is also a report on *Software Configuration Management Practices for eXtreme Programming Teams* by Lars Bendix [11]. The previous work on the subject was a broader view of configuration management in agile methods. This thesis is focused on traceability and how it should be implemented.

3 Analysis

The following chapters will describe what the problems with tracing in agile methods are today. This analysis is based on the interviews done during the initial part of the project, as well as the CMCM meeting in Lund. Two main issues were discovered to be a possible cause for not having traceability (or in some case not knowing that they had it). The first issue was the attitude towards traceability and the second issue was the lack of knowledge about it.

During the research phase the main source of information were interviews. The result from these interviews is presented in chapter 3.2 and 3.3. The problems that were found as well as the solutions that are presented in chapter 4 and 5 are more or less impossible to connect to a specific source. The information presented is my interpretation of the interviews as well as some studies from some books as well as blogs. Therefore this is my view of what the problem really is and it can differ from some of the people that were interviewed.

3.1 Methods and Limitations

The initial problem was formulated by Christian Pendleton at Softhouse Consulting in Malmö. In order to get a broader view of the problem most of the people interviewed were managers of some sort and only a few were actually part of a team or had experience as developers. This could result in a more management oriented result with focus on requirements handling in the agile methods.

Methods

The primary method used for this project was interviewing people at different companies and with different roles in order to get a broader view of what the potential problems with traceability were. There were also some literature studies as well as some searching around the internet in order to find previous work and comments on the subject.

After the initial phase with the interviews a paper known as *first result* was created. This document was then sent out in order for me to get feedback on the preliminary results. This document contained the practices in chapter four as well as the comments in chapter five. However it was quite hard to get people to read and send feedback on the first version. It take quite some time to read and review a paper and most people had a lot to do before the Christmas and new year holidays.

Towards the end of the research there was as Configuration Management Coffee Meetings in Lund (CMCM [12]) where the preliminary result was presented followed by a discussion about traceability in agile methods. The CMCM is a meeting where people from the industry come together to discuss configuration management subjects. At the CMCM meeting the preliminary studies were presented and then there were some discussions about traceability in agile software development. During the meeting some interesting thoughts and ideas on traceability were brought up and discussed.

Limitation

In the beginning of the project it was easy to find people to talk to. However towards the end when the working document was sent out in order to get feedback there were some problems getting feedback from people. This caused a problem as there was hope to get more feedback and for the document to trigger a discussion with the people that had received it.

In the beginning of the project there were some thoughts on reading up and writing something about traceability in Lean and distributed Scrum. However due to some problems finding people with experience with Lean this part was cut quite early. The reason for not doing some extra research about Distributed Scrum was due to the time limit of the research phase. But in some discussions there will be some information on how tracing could help distribute Scrum.

Sources

During the research on traceability several people with different roles were interviewed. The sources interviewed worked on different companies and on different projects, this resulted in a broader view of the problems and potential solution. Among the interviewed were configuration managers, developers, testers, product owners and scrum masters, scrum coach. What was interesting was how they all had a different opinion on traceability and the need for tracing in Scrum and XP. One of the most important sources found on the net was Brad Appleton's blog [13] where there were a lot of opinions on traceability in agile methods.

In the beginning several people at Softhouse were interviewed. They all had different roles and different experience. The roles these people had were scrum masters, scrum coach, product owner, developers, configuration managers as well as a tester.

Outside of Softhouse a scrum master at Securitas Direct AB was interviewed as well as a configuration manager at Cybercom group.

In order to get some feedback on why traceability would be useful in traditional development a project manager (which had a more lean approach) was also interviewed.

3.2 The attitude towards traceability

The first problem with the attitude towards tracing is the current state of mind with the persons that is using agile methods. The problem is not only with the team (the developers) but also with the configuration mangers, customers, product owners and the project leaders. Therefore it is important to not only blame the developers or the configuration managers but find everyone in the project chain that could cause a problem.

When *tracing* or *traceability* was brought up during interviews some of the people went into a defensive mode and stated that it is not needed in agile software development. Some commented upon the traceability saying it cannot be agile and it will not bring any benefits to the team.

Some statements were that tracing should only be implemented in agile development if it can be done without cost. There should not be any extra work involved to save or document the work and relationships. One person interviewed considers that tracing should be implemented with automated tools and the developers should not have to write a number or comment in the code in order to link it to another artifact or version, or having to open a document.

3.2.1 Lost creativity

Some developers (and in some cases even the configuration managers) feel that there is a loss of creativity when you have to add traceability and documentations. This is partly because some configuration managers create heavy routines (that in some cases is taken directly from the traditional development methods). The problems with heavy routines will be discussed in a later chapter. One of the comments on this was that the team members could not develop as freely as they were used to. For example in some cases the design of the code base needed to have a specific design or perhaps they were forced to work on only one task at the time.

3.2.2 Scrum is complete

Scrum is a religion

For some, Scrum is more of a religion then a developing method and they believe that if they run Scrum by the book everything will solve itself. In one of the interviews it was mentioned that there are those who read one or two chapters from a Scrum book and get the impression that if they run Scrum they have the knowledge in the team and will not need to document their work. Some believe that Scrum is complete and it does not need to be changed or improved.

Scrum does not need improvement

Some consider Scrum as something that does not need to be improved while other changes it and call it something else [14]. However many still adopt Scrum into something of their own. Some of the parts of Scrum are needed in order to call it Scrum, however that do not mean that we cannot add something new. Scrum is not something that should be considered complete and it needs to be changed according to the situation.

3.2.3 No one reads it

This could be a big problem for an organization. If the persons that produce the documentation and connect the artifacts (tracing) feel that no one reads it or that there is no point of it all. There is a risk that the quality of the documentation will go down or that important information is left out. If someone reads the document it is bound to be discovered, once it is discovered it could be too late to correct it. In another interview it was mentioned that he sometimes uses Scrum as an alias in order to question if the documentation is useful and if they need it at all. It was also mentioned that in some cases documents are produced and never read.

This is also something that I experienced from the course Software Development for Large Systems [15]. Some of the students felt that they were producing documents that no one read. The problem in this case was that they were right. The people that were suppose to review and comment on the documents never read the entire documents. They read some parts and then commented on that. This resulted in a lack of interest in doing a good job.

Tracing is target for similar critique. “Why do we need this?” and “Who will ever use this information?” are common questions when implementing routines for increasing the traceability. The problem here is that some of the information the team provides is supposed to help themselves. However not all tracing information is supposed to help the developers, some information aims to support maintenance of the product and other information is useful for the testers, managers and so on. So one of the problems found here is that perhaps the team does not see the entire development cycle.

3.2.4 Heavy routines

In the interview with one of the configuration managers he mentioned that a possible problem could be that some configuration managers have created heavy routines that affect the developer’s daily work and productivity. Then when they implement Scrum they do not change their methods and routines in order to make them agile and this causes the Scrum experiment to sometimes result in failure. Heavy routines could sometimes lead to a less agile version of Scrum and therefore some of the benefits from the agile methods would be lost.

3.2.5 Documentation means documents

A common misbelieve is that documentation of your work means that you have to create documents that (in some cases) no one reads. Managers are sometimes convinced that in order for them to have an insight into the project the team needs to create documents where they document their work. It is also believed that if you have to document then it is a lot of work and needs to be done in a specific way.

The developers feel that if they have to create documents then they will lose time they could spend on writing code and that they will not benefit from it. One of the problems with documentation in documents is that when someone needs to find the information they do not know where to look and ask around instead.

3.2.6 Administrative overhead

Tracing as well as documentation is sometimes considered to be unnecessary administrative overhead for the team, the team will have to spend a lot of time writing documentation. The team feels that they will have to spend time writing documents and documentation, time they could spend writing new functionality.

In some extreme cases the developers only see the developing part and do not care about testing their code. Some feel that it is not something that they should do and all they should do is write code, even if this goes against the agile intentions. This narrow-minded way of thinking could be a problem for the organization as whole. Not only will it in this case result in more bugs but it will also result in a lack of will to learn about the organization as well as a will to help anyone outside of the developing team.

3.2.7 Lack of motivation

The team might get some lack of motivation to implement tracing if they do not know the reason for this or feel that they can benefit from it. Tracing can be enforced if the customer wants it and is prepared to pay for it, but what if he does not know why he wants it or maybe even really know what he wants? If the customer do not know why he wants it or what he wants then it will be harder to motivate the team. It can also be that the customer is used to getting it and do not know how much it will cost him. Another problem in agile methods could be that someone from outside the team tries to enforce tracing, this could for example be an outside configuration manager, a manager or a customer. Forcing the team into saving tracing data could result in reduced quality and also a risk that the data is not being kept up to date.

3.3 Lack of knowledge

Today there is a lack of knowledge about traceability and how useful it can be when used in the right way. The lack of knowledge is not only among the developers, but as stated before a problem with almost everyone involved in the process. This chapter will explain some of the problems that were discovered considering the problem with the lack of knowledge about traceability. Some of the problems with lack of knowledge are also related to the lack of motivation. Therefore there could be some overlapping between this chapter and chapter 3.2. Some of the problems with the attitude towards tracing are based in the lack of knowledge.

3.3.1 Traceability

A potential problem is the lack of knowledge of why, how and what we want to trace. Not everyone knows what we can trace and why we would trace it.

Do you feel the need to trace in a project, and if so what kind of tracing?

After asking this question some got in and started discussing tracing requirements to lines of code and back. The discussion often ends in why would we need this? After leading them into the world of tracing, the second most popular artifact of tracing are

requirements to test and test to requirements. Not everyone thinks of this connection as a tracing at first, but more of something that is just there in order to validate the product. This shows that not all of the people that work as for example product owners and scrum masters know of traceability.

The problem is not just the lack of knowledge on what and how you can trace, but there is also a problem that people do not always know what the costs and benefits of implementing tracing are and therefore it is very important to give a clear and workable definition of traceability. My definition of traceability can be found under chapter 2 – Traceability.

3.3.2 The whole picture

Sometimes the team might not see the entire picture of the products development. What they see and experience is only the development work and automation of tests. There is a lack of knowledge and experience (and perhaps a will to understand) and they might not know how documenting from their part could help the testers do their work a bit better and in the end give the customer a product with higher quality. One comment was that people only see their own part and tend to sub-optimize.

The team might not see how tracing could help them, or why they should document the connection/information between two artifacts. In agile development, using only one team it is often easier to ask the person next to you why, how and when he did a specific change instead of looking through the documentation or the repository logs (if we have a version control tool). This could also be caused by the organization. Perhaps the team members can't really get an insight of how other units work. Information such as why a specific design was used or how a part of the system was indented to work.

3.3.3 Documentation in agile methods

A common misconception is that in agile methods you should not document. The knowledge should be in the team and everyone should know everything. But what happens if there are several teams working on one product, as is the case of distributed Scrum or outsourcing. What happens if one team member is lost?

One day the team will have to stop developing and turn it over to the customer or a maintenance group. If there is no tracing or documentation, how will they know what the team has implemented, how it was implemented and why it was implemented in that way? What is the relationship between artifacts? How are the requirements tested?

3.3.4 It is expensive or at least it is not free

It is expensive and the team will have to do a lot of extra work and the customer will have to pay more for the product, this is one of the comments received from the interviews. In the interview with a developer at Softhouse he mentioned that if you should implement tracing in Scrum it must be without adding any extra work on the team. So the team should not have to open a document or have to add a number to the code.
In the interview with a Scrum master he said that if the customer wants traceability and he wants to pay for it then we can implement it. Tracing can be considered something that is expensive and it will result in less stories being implemented. In most cases this could be true. However most people do not think about what will happen after the development is finished. Will there be a maintenance period, will the testers benefit from the information and so on.

3.3.5 When and what do we need to trace

One of the questions teams and manager could ask them self is when do we need to trace. Not everyone knows when or what to we need to trace. The answer to this question is important in order to create tracing/documentation routines that are not too light and not too heavy. "Do we need to trace requirements to line of code?", "Do we need to trace test to requirements?"

3.3.6 Routines

A problem when trying Scrum for the first time is that the configuration managers do not adapt the routines to be more agile. A problem is that companies sometimes try Scrum on smaller projects where there is no real need for heavy routines. The problem is to find a suitable level and how to make our routines more agile.

Using heavy routines could reduce the benefits from the agile methods. Perhaps some routines are not needed when working in agile methods. Heavy routines could reduce the efficiency of the project and some developers may feel that some routines reduce the flexibility. Routines such as only having to develop one story at a time and having to register when they go over to fix bug (a routine that could be necessary for the practice requirements to code in chapter 4.1.5). Such routines can be considered to reduce the creativity and freedom when developing.

4 Tracing practices

In the previous chapter some of the problems with adding traceability into the agile software development were presented. This chapter will describe the tracing practices that can be used in order to add traceability in the agile methods. The practices work well by themselves but in order to gain the most from them they should be combined with each other. To each practice there is a list of benefits and costs/drawbacks for adding it. While some of the practices are cheap or free others are expensive and will probably not have positive return on investment. In the end of this chapter there is a complete example that shows how tracing could be added in order to be able to trace throughout the entire development cycle.

4.1 Practices

These practices could be used to add traceability to the agile methods. There are several things that could be useful to trace. It is therefore important to know what, why and how to trace. Some of the practices in this chapter could be considered cheap or even free (ex. stakeholder to requirements) while others will be expensive and have a low or negative return on investment (ROI).

In order to have traceability it needs to be possible to trace in both direction [16]. If nothing else is stated it is assumed that if enough tracing information is saved it will be possible to trace in both directions.

Acknowledgements for the Practices
The practices in this chapter are based on the interviews and discussion made during the research on the subject. During the first weeks at Softhouse there were several discussions with Christian Pendleton and during these discussions the ideas of tracing practices were founded. The first practices were the ones in chapter 4.1.1-4.1.3 and were created after some interviews with Christian Pendleton. After the interview with Johan Natt och Dag at Riada[2] the practice in chapter 4.1.4 was formed.

Another person that was a great influence to these practices was Thomas Lundström. From the interviews with him the practices in chapter 4.1.10, 4.1.13 and 4.1.14 were created. The two practices that involve databases and the code base are standard practices used by developers today. The final practice that was based on a single interview is chapter 4.1.5 that is based on both the interview with Arne Åhlander and the interview with Jonas Andersson.

The other chapters are based on my previous experience with software development at LTH and with the game Gangsterhood[3], as well as a mix of interviews.

[2] www.riada.se
[3] www.gangsterhood.net

4.1.1 Stakeholder to requirements

Stakeholder to requirements is something that is not that uncommon when working with Scrum. In Scrum everyone is allowed to submit a requirement to the product backlog. By simple logging who entered the requirement (you can bind it both to role and person) you will have a way to find more information and domain knowledge about the requirement. This is something that can be useful during the entire development cycle. This information is very easy and cost efficient to implement and could in the end return a lot to the team. It is important to be able to trace from requirements to stakeholder as it is from the stakeholder to requirements.

The stakeholder information can also be useful if you want to have a feedback loop so that the stakeholders feel that something is happening and that they do make a difference. Otherwise you can sometime get a feeling that no one listens to your request and that nothing is happening.

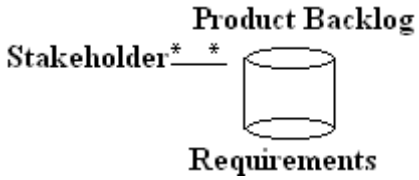

Figure 1 – Stakeholder to backlog/requirements

Benefits

- Possible to go back and see where a requirement came from. Could be valuable if we need more information.
- Possible to create a feedback loop and inform the stakeholder when the requirement is implemented.

Cost/drawbacks

- Cost of changing (if necessary) the database so it can store where the requirement came from (ex name, role and date and preferably store this information automatically upon adding a requirement).
- Cost of keeping the stakeholder information up to date. One comment was to perhaps link it to a customer handling system.

4.1.2 Initial problem description to requirements

If there is an initial problem description from the customer and he wants to validate that the initial problem description was properly converted to requirements, we need to keep track of this connection.

During the development of a new product, requirements are constantly added to the product backlog. Before a new sprint the product owner prioritizes the requirements and the team estimates how many they can handle during the sprint, the selected requirements are then moved to the sprint backlog. Therefore it is possible that this is something that is more useful in Lean development then in agile methods like Scrum. In Scrum the initial problem description could in the end be a small part of the requirements in the requirements database. It is equally important to be able to trace from requirements to initial description as it is in the other direction.

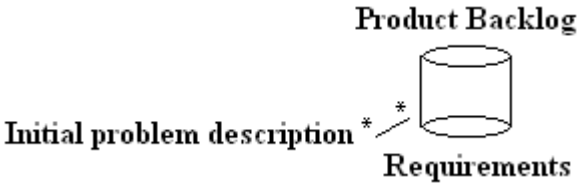

Figure 2 – initial problem description to backlog/requirements

Benefits

- Possible to validate that the initial problem is solved.
- It is possible to validate that all the tasks from the initial problem description is translated to requirements.

Cost/drawbacks

- Cost of changing (if necessary) the database so it can store where the requirement came from.
- Needs to link the requirements to the problem description.

4.1.3 Requirements to story/sprint log

This is something that can be implemented without any real extra work. All that is need is to add the requirements id to the story card and sprint log when it is created. By doing this it is possible to go back and find the original requirement. It also helps if we want to find more information that is connected to a specific requirement.

In some projects there will be changes or more information might be gathered and documented on the story card. If you tag your story card with a requirements ID as in figure 3 then this information could be tracked down later if you for example need to change or validate a requirement.

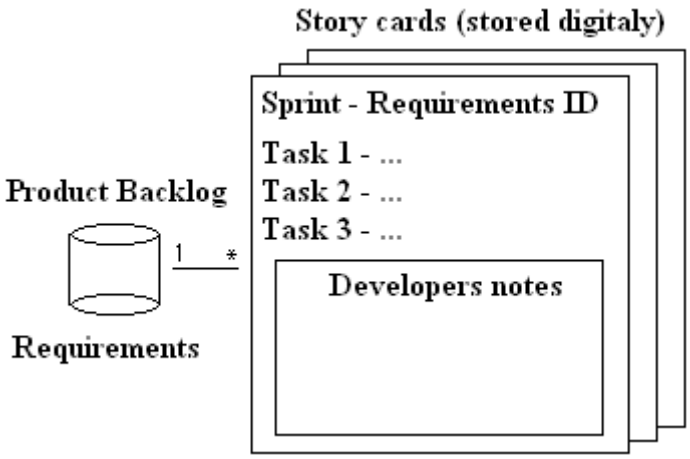

Figure 3 – requirements to story cards

Benefits

- Possible to link requirements to story.
- Possible to go back to the requirements database to see more information that is stored together with a requirement.
- Possible to update the requirements during development and testing.

Cost/drawbacks

- In order to link requirements to story card (automatically) you have to store the story card digitally.

4.1.4 Requirements to requirements

Requirements to requirements helps us find dependencies between different requirements. How detailed you want this information is up to you and the team. If we want a very detailed connection and are able to see exactly how different requirements affect each other, then this will be very expensive. A deeper analysis requires a lot of extra work and there is a high probability that you will miss something. If you strive to have a detailed connection and fail then the information might be useless. In figure four a simple example of two requirements that are related are shown. In this case it could be good to know that there is a requirement that states what the default background color should be (A35) when implementing a new requirement (B95).

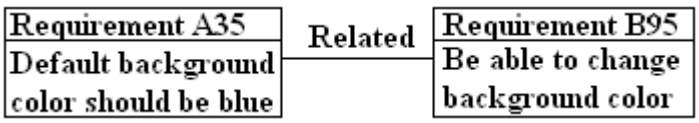

Figure 4 – Requirement to requirement

The other way to use this traceability is to put it on a higher level. The information can then be collected while you analyze the requirements and try to break them down. If you put it on a higher level the cost of implementing it will be less expensive and could potentially be more relative to the gain you get from it. A less detailed connection is easier to maintain and therefore have a lower risk of getting out of date. Information

gained from breaking down the requirements should be saved and linked to the requirement. This information could then be reused if the requirements needs to be reevaluated during development. When modifying an existing system it could also be good to analyze the existing requirements in order to find dependencies that could require a large architectural change to the project.

If you plan on having several Scrum teams working on the same project this could be an advantage, if you for example have big requirements that you want to split up into several smaller pieces and have several scrum groups working on those. At the end of the sprint you then want to merge everything together and have a complete product. In order to see that every piece is done and how they affect each other you may want to trace requirements to requirements.

It is also very important to know that incorrect information could be dangerous. It is therefore important to keep the information saved together with the requirements up to date and correct.

Benefits

- Will give the developers an insight on what could possibly be affected when a requirement is changed.
- If used together with requirements to test it will give the developers (and testers) a way to see what test they need to run again.
- Could help make better time estimations as well as better project/integration plan.
- Enables the possibility to split requirements up and to work on them in parallel.
- Better impact analysis when requirements change.

Cost/drawbacks

- A detailed connection could be expensive to implement and maintain.
- Will require maintenance.
- Incorrect information could be dangerous.

4.1.5 Requirements to code

This can be implemented to some extent using tools. If all we require is a way to see what changes were made when implementing a specific requirements or see too what requirement this line of code was created. However this requires discipline and the developers need to work on one task at a time or the tool needs to allow the developers to switch between tasks and log when they do this.

One set of tools that can be used in order to automate this process is XPlanner[4] and CVS[5]. However you will need to develop a plugin. This was done in the Coaching course [17] at LTH by Jonas and Jonas [18]. The tool was developed as a proof of concept and helped the developers to log what and when something was implanted. By using a tool like this and loging requirements to code the process will be automated and will not require too

[4] sourceforge.net/projects/xplanner/
[5] www.nongnu.org/cvs/

much extra work. However it requires discipline from the developers to only work on one task and to inform the system when they begin, end and change task.

If you want to implement tracing to code completely you will have to do this manually. The strictest version is that every line of code needs to be connected to at least one requirement and sometimes a line of code is needed for several requirements. This requires that developers manually log every line. This is extremely expensive and will rarely give you any benefits. Requirements to line of code are illustrated in figure 5.

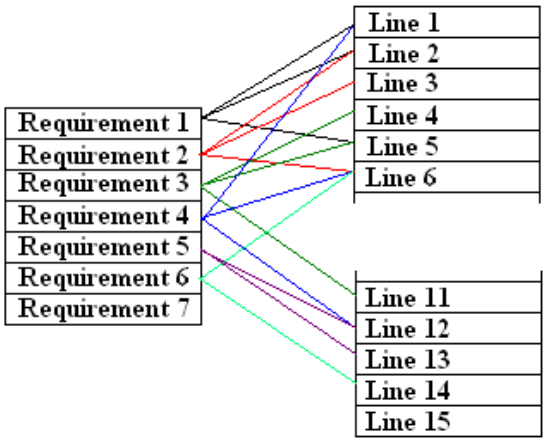

Figure 5 – Requirements to code.

Connecting requirements to every line of code used will also be expensive and difficult when using frameworks. A comment received on this is that the use of framework together with this tracing practice will be expensive and there will be a high risk of errors. Connecting requirements to code can be useful if we want to analyze the code afterwards. However if you only want tracing for code reviews then requirements to line of code is not the best way to go. If the team develop with test first or task driven development and thereby work on one task at a time and the team tag their commit with an ID there will be enough information to do a code review.

Benefits

- Possible for the maintenance group to get a hint on where to look when trying to fix a bug (if they know what requirement that is broken).
- The testers will know what they need to test.
- The developers can see what was changed when implementing a specific requirement.
- Everyone can find when, how and by whom a requirement was implemented.

Cost/drawbacks

- Very expensive to have requirements to line of code and back.
- The documentation of requirements to line of code will need to be maintained and have a high risk of being inaccurate (if not maintained correctly).
- The simplest form of requirements to code will require tool support or it will be expensive.
- Developers need to work on one task at a time or mark when they change tasks.
- Because requirements are often broken down to tasks you could end up having to log requirements to tasks before implementing this.

4.1.6 Requirements to version

By saving what requirements we have implemented in a specific system revision we can see what the difference between two specific versions are. If we have more than one customer that uses the same software it can be important to know what we have changed. It can also be important to know when a specific bug was detected and fixed.

A scenario could be that the customer A finds a bug in his version (3.2.1), the bug was then fixed in 4.0.0. The customer then wants to know if he can upgrade to 4.0.0 but unfortunately there was a big change so versions 4.0.0 do not work for him. This would mean that we need to implement the same fix in 3.2.1 and release a 3.2.2 version.

This can also be important to save in order to show the customer what we have implemented and what the customer is paying for. The tracing itself might not help increase the quality but it can help increase customer satisfaction.

One configuration manager said in the interview that he considers it important to have control. If the customer ask you what you have changed it is important to be able to answer. By logging requirements to version it will help you give some control.

Logging requirements to code can be done quite easily if you have a source control tool (example CVS, SVN[6], Clear case[7]) and storing the requirements ID when you commit a change. Then by tagging your release you can with some repository mining find what requirements were implemented between versions. Another solution could be to save the repository version with the requirement but this will require some extra administrative work.

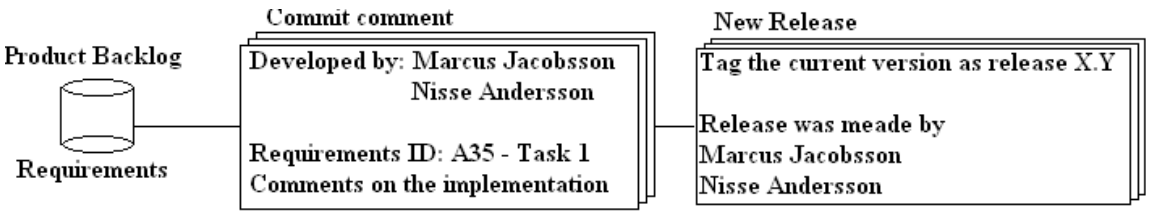

Figure 6 – Requirements to version (using version control)

[6] subversion.tigris.org/
[7] www-01.ibm.com/software/awdtools/clearcase/

Benefits

- Needed in order to tell what changes there are between versions.
- Makes it easier to customize a new version to a customer.

Cost/drawbacks

- You need to save when a specific requirements was completed and tested.

4.1.7 Requirements to test

By tracing requirements to test and back we have a way to validate that a requirement is implemented and tested. This will also help the testers to see what they need to change when a requirement is changed/removed and what requirements they need to test. It is also useful as validation that the system is implemented correctly and gives the customer some form of certification that we have tested the system.

Tracing to test could also help developers to know what tests and in a way what code they need to change once a requirement is changed if we have automated test.

Tracing requirements to test is often done in projects. A comment was that some requirements and testing tools can make connections between tests and requirements.

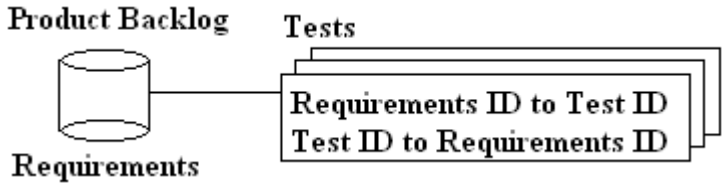

Figure 7 – Requirements to test

Benefits

- Makes it possible to validate that all requirements are tested.
- Makes it possible for the developers to see what test they need to focus on when a requirement is changed.
- Gives the tester a way to find the test that they need to change when a requirement is changed.

Cost/drawbacks

- Requires the testers/developers to log a connection between tests and requirements.

4.1.8 User evaluation to version

When a system is being evaluated by the end users (for example alpha and beta testing) there will be a lot of information submitted. This information needs to be stored in a way so that it is easy to access and to see who submitted what and to what version. Some information about the data stored and system information could also be useful in order to find out why the user experienced some problems. This information needs to be connected to a specific version.

Benefits

- Possible for the developers to access the feedback during development and to be able to run the system that was tested.

Cost/drawbacks

- Needs to be saved in an easy to access format (for example a database)

4.1.9 Platform information to release

Some companies need to be able to return back to an older version in case something goes wrong. Therefore it can be useful to save enough information about the platform in order to go back and recreate the exact environment that the software ran on (and if the system ran on a test platform, save the test data used). By linking platform information to a specific release it will be possible to recreate the system once more.

Some information that could be useful to save is operating system version, hardware specification, database data (or other data that is used to test the system) and so on. When trying to recreate the platform it can be useful to save the tests and the result they produce when that specific version was built.

Benefits

- Possible to recreate the entire system.

Cost/drawbacks

- Could be hard to save some of the information and even if all information is saved it can still be hard to recreate the system.

4.1.10 Implementation documentation

Traceability between two systems can be best described if it is broken down into two parts. The first part is when the systems are split into several smaller systems (for example the case when distributed scrum is used). The second part is when information is recorded from one project and used in another (Decision to final product).

Split the systems into smaller pieces
If the systems are part of a bigger project then the information cannot be kept in the team. It is important for the other teams to know for example when and how a specific feature or fix was implemented in order for them to do their job. Sometimes it can also be important to know what the others are doing, or who is working on a specific task right now. In some cases, for example when developing a client server, one team is working on the server and another team is working on the client, the client team may want to know when someone from the server team is working on something that will affect their solution.

Decision to final product
When a company is finished with developing a product for a customer they will move onto a new project. In order to not make the same mistake twice or having to invent the wheel all over again it could be good to save some documentation on why and how we solved a problem. By also documenting how the final system turned out, this could help the team make better decisions on another project and not do the same mistakes again. If for example the team made a specific design solution and in the end it turns out that it works like a charm, then perhaps the team can use this knowledge in future projects. Perhaps the team could document their work in a form of diary, like one of the people I interviewed.

Benefits
- Possible to keep information even after a project is over.
- Could reduce future development time by keeping a diary with decisions.

Cost/drawbacks
- The team needs do document their work (in some form).

4.1.11 Code and Table tracing

Dependency tracing
Tracing relationships between parts of the code could help the team get a better overview of the code base as well as make the impact analysis easier. In object oriented languages (OO) a notation know as Unified Modeling Language (UML) is used. It is also possible to do dependency tracing in non OO languages, but the following examples and explanations will be on how UML could be used for tracing. The reason for this is due to my previous experience with UML. The process of creating the UML diagrams could be painful and time consuming if done manually. When done manually the team will have to know exactly how everything is related and every time the code changes the UML diagram needs to be updated. So in order to keep them up to date and not spending too much time on them we need tools. There are several tools that autogenerate UML diagrams (however some have a tendency to break the code if used incorrectly). In visual studio there are tools that can autogenerate UML diagrams and if the software is written in Java and the IDE is Eclipse[8] then there is Omondo[9]. UML diagrams are more often

[8] www.eclipse.org
[9] www.eclipsedownload.com

used in traditional software development during the design phase. UML is normally used to design the software before the code is written.

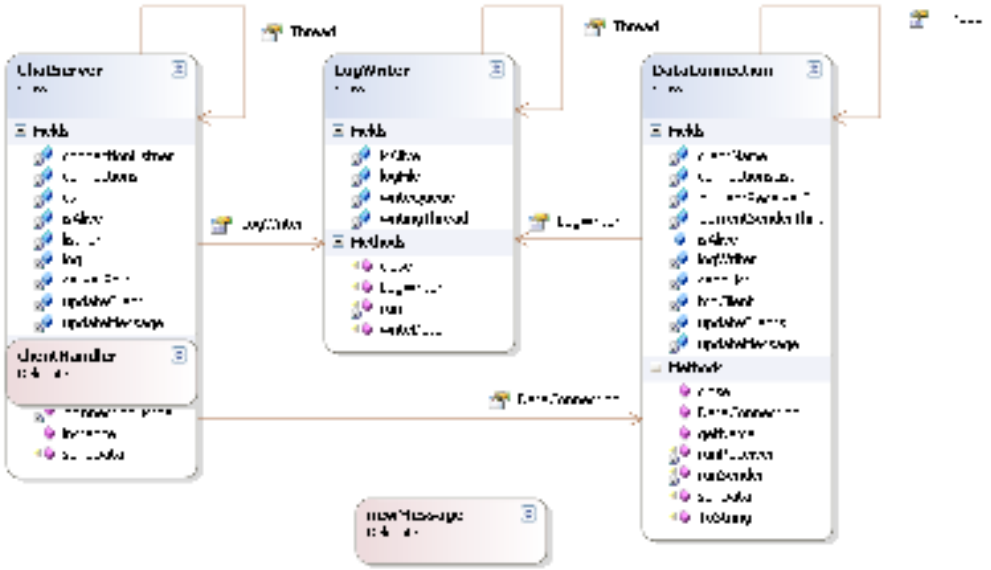

Figure 8 – Example of a UML diagram generated in Visual Studio[10]

Table to Table

Tracing relationship between tables in a relational database could be considered to be extremely important. A big database design can be complex and in order to keep the database clean and avoid redundant data as well as circular dependencies a good database design is a must. A complex database design can also be hard to understand if you only have the tables. However if you get a graphical representation together with the relationships it is much easier to use the database.

In the database world then there is something called Entity-relationship model (E/R [19]) as well as Star schema [20]. E/R modulation and star schema is often used to design the database and can in some databases such as Microsoft SQL Server be autogenerated. In the same way as with UML diagrams, it is important to be able to autogenerate the diagrams in order to make sure they are kept up to date. At the same time if we use tools (the right tools) the team needs to spend less time working on the diagrams and can therefore spend more time developing the software.

[10] msdn.microsoft.com/sv-se/vstudio/default(en-us).aspx

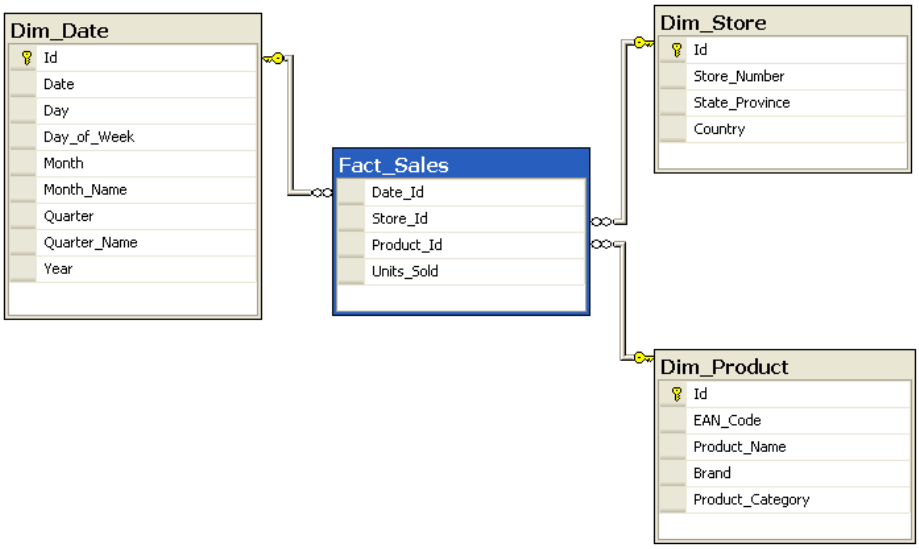

Figure 9 – An example of a star schema [22]

Benefits

- The developers can get a better overview of the code base and the database.
- Easier to spot if there is a flaw in the design.
- Could be used in order to illustrate the software and database design for someone outside of the team.
- Will help when turning over the project to another team.
- Could be used in order to minimize the time it takes to find a bug.
- Could reduce the time it takes to implement a new feature in the code or database.
- Could be used as a poster for the burn down chart.

Cost/drawbacks

- The team needs to spend some time during each sprint in order to update the diagram (less time if it can be automated).
- Could potentially be a cost for the developers (but a gain for the project).

4.1.12 Developer to Code

During one of the interviews it came up that it could be good to know who implemented a specific feature. One reason for this is to reduce the time needed to ask around. It can also be because the team needs to find out if there is a team member that needs some more instructions or education (for example on writing tests).

Reduce the need to ask around

In order to not having to interrupt the other team members and disturb their work flow, the team could use the version control tool to find out who have implemented something specific. However in order to do this, the team needs to be able to track from a specific requirement to a specific commit (or set of commits) and thereby be able to find the person responsible.

In XP where one of the practices is pair programming and the team members are moving around it is important to write the name of the people that have worked on a specific commit. In the course Software Development in Teams – Project [22] at LTH the XP teams will learn to develop product using the XP methods. In the course the pair programming is taken to the extreme and it is not uncommon to change partner 4-5 times a day. Sometimes the switching results in that user A is logged into the computer (and his or her account is used to connect to CVS server). However user A is not working on that computer and instead it is user B and C that is working on that specific story. So when team commits a change CVS will log it as if user A is working on that part of the code. Therefore it could be wise to record who is actually working on something in the commit comments.

Weak link
Sometimes there could be one or more people in the team that always writes bad tests (if, for example, XP and Test first is used). Or perhaps there is a problem with indentation (could cause merge conflicts). In these cases it could be good if the team is able to find the person (or persons) responsible and instruct them (or if necessary educate them). There was a comment on that this could lead to a blame game and requires a more mature team. The problem is that it can be a higher risk to have people that do not know how to write tests or good code. If there is a problem in the team it is important to eliminate it early and not let it grow.

Using the tracing information just to find out who implemented a bug is rarely positive. It is therefore important to, if it is necessary to find the weak link, to use the information in a constructive way that will help the team. Otherwise the team could turn against each other and then refuse to write their names in the commit comments or in other ways prevent others from finding out what they have done.

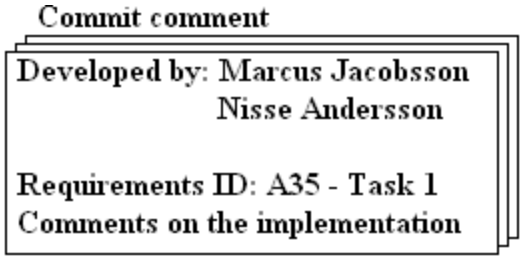

Figure 10 – Using commit comments to register who have worked on a commit

Benefits

- Spend less time to ask around.
- When not asking around you do not disturb people to the same degree and the team can work more efficiently.
- Easier to find and therefore solve the problem with people not doing their job correctly.
- People sometimes forget what tasks/requirements they worked on (if a long time goes by). But if you start a discussion with that person he or she will most likely remember why he or she did.

Cost/drawbacks

- The team needs to use a version-control tool.
- It is recommended that the team spend a few extra seconds on each commit and write the name of the person that have worked on the tasks/requirement.
- Possible for the team to refuse saving tracing information if it is used against them.
- Need create a standard for how the commit comments should be formatted, like in the example in figure 10.

4.1.13 What others are working on right now

This is something that is more important when working with distributed Scrum. If there is one team and everyone is sitting in the same room then it is easier to ask what the other team mates are doing right now. This is also recommended as it is important that the team member interact with each other. However sometimes the team members are located in different rooms/floors/parts of the country or in some cases different parts of the world. It could therefore be necessary to log what others are doing right now. Perhaps there will be a merge conflict or the code will interact with each other and therefore they need to discuss the solution with each other.

In Scrum as well as in XP there is a standup meeting every day where everyone presents what they are going to do during the day (daily scrum meeting in Scrum and stand up meeting in XP). This will partly solve the problem.

If for example a small web application is developed where the team members can mark that they are working on a specific story/task it will be easier to find out what others are doing in this exact moment. Another example received from a review was to add it to the developing tools and use task-based development.

There are several ways to solve this, the two main solution would be a push or a pull. In the push solution the information is pushed out to the other developers (perhaps with a notify box or field in the development tool. The other solution would be a pull, perhaps by visiting a web page where the team members can see what the others are doing now.

Benefits

- All team members will always know what the others are doing.
- Useful when working with distributed environments.

Cost/drawbacks

- Requires to setup/develop a system so the team can log what they are doing (perhaps be able to store the information with the stories for this sprint).
- Requires some discipline so that the team members update what they are working on (perhaps a system where the team members have to check out a stories).

4.2 Practices combined

There is a say "*The whole is greater than the sum of the parts*". This is also true when it comes to tracing. Every individual technique described in chapter 4.1 has some advantages and disadvantages. However when the techniques are combined, the team (and, in most cases the entire project) will gain more. In this chapter some ways of combining the techniques are described. However in the end there are too many ways of combining and adapting the practices to your projects that it will be impossible to describe them all in this chapter. However it will still give a hint on the advantages of tracing. Adding practices that will not help anyone will only create unnecessary overhead for the team. It is therefore important avoid adding unnecessary practices (and if they are added be ready to remove them).

4.2.1 Simple combinations

By combining one or more of the practices explained in chapter 4.1 it is possible to get something that is more useful. In some smaller project it is not always useful to have a full scaled version of the traceability practices, perhaps the team would benefit more from a small downscaled version. Using only one practice will give the team some advantage, but using more than one could lead to a bigger advantage. The reason for combining two or more practices is in order to get a hold of the information that we need. In this chapter there are some examples of how some practices could be combined.

Requirements tracing

A simple way of adding requirements tracing to the agile methods is by combining the practices explained in chapter 4.1.1 – stakeholder to requirements and 4.1.3 – requirements to story/sprint log in combination with adding the requirements id to the commit comments. This will give the team a way to follow the requirements from the time they were created to the commit into the repository as well as from a commit back to the creation of a requirement. All that is left is to add a tag to the repository when a release is made and it will be possible to have a small but complete way to trace the flow of a specific requirement during development. Another advantage is that we can go back and inform the stakeholder when their requirements were implemented.

Adding only a few small and easy practices and adding them in a way that creates a framework like in figure 11 will make it easy to add new practices later on. The framework in figure 11 is the same that is used in chapter 4.3 where a larger design is discussed.

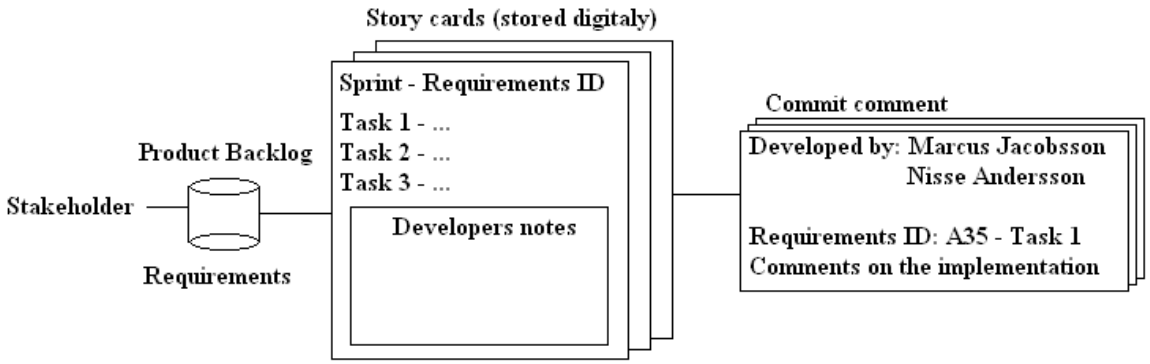

Figure 11 – A simple combination of the requirements practices

Test tracing

Adding a way to trace tests is useful in order to validate the product. Test tracing is more than just marking a test with a requirements id or adding a test id to a requirement (if both are used then there is a bi-directional tracing connection). When adding traceability on tests it is also important to give the testers a way to see what changes have been made and perhaps also by whom. The testers need to know when and what something specific was implemented. If they also know who implemented something they can (if needed) find the person that wrote the code and comments and ask for more information.

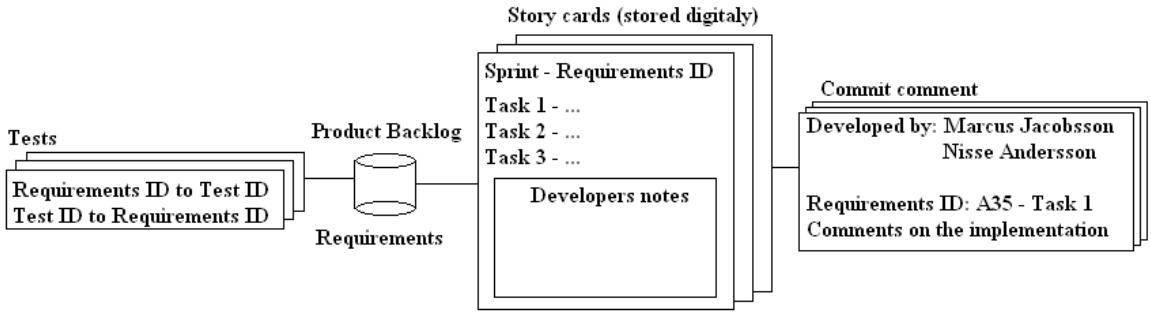

Figure 12 – A way to add test tracing

In figure 12 a small example of test tracing is presented. The testers will now have a way to find out more information about the implementation by having access to the story cards and the comments. They can also see who implemented a specific change and when it was implemented.

4.2.2 Complete solution example

This example will look how it is possible to combine several different techniques and describe some of the advantages and disadvantages. In the E/R diagram below a possible solution is described. Once all the information is connected it will be easier to access the information. After a while there will be a lot of information in the system and therefore a good search algorithm is necessary. The information could be presented in a way so the configuration manager just needs to connect the Google or Microsoft[11] search engine to the system, for example the Microsoft Search Server[12].

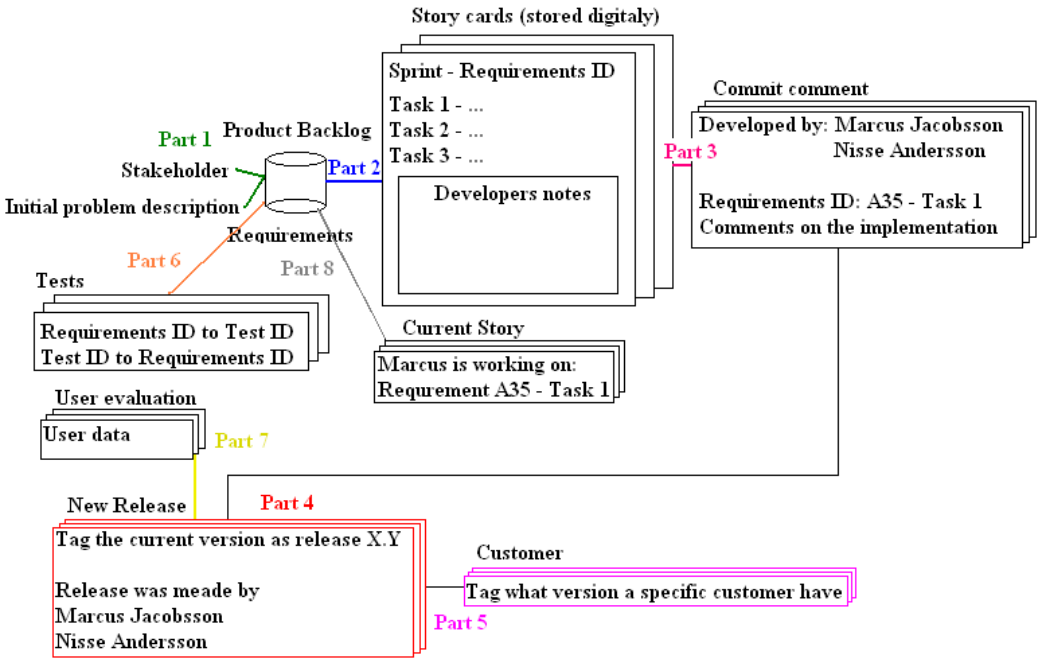

Figure 13 – An example of how the tracing system could be designed.

Description

Part 1 – Product Backlog (green parts)

The first step is to save the initial problem description and how it is connected to the requirements in the requirements database (product backlog). It is not necessary to save the initial problem description in the product backlog but it could be good to have it stored in a way so it is easy to access later on in the project. It is necessary to modify (if it is not already supported) the requirements database so that it is possible to store where the requirements came from. By doing this the team will always know where they can go for more information.

[11] www.microsoft.com
[12] www.microsoft.com/Enterprisesearch/

Part 2 - Story cards (blue parts)
From the product backlog the sprint log is created. If possible, use the product backlog as the sprint backlog and just mark the stories that the team are suppose to complete. From the requirements, story cards are created and every story card is broken down into tasks. The story cards (preferably stored as digital artifacts) will have the requirements ID stored on it. By doing this it will be possible to go back to the product backlog and find more information.

Part 3 – Commit comments (pink parts)
Because the stories are broken down into tasks it is not always possible to add the requirements ID to the commit comment. However if the team add the task ID and requirements ID it will be possible to trace when a specific task was implemented. If this is done then it will be possible to add a feedback loop back to the person who added requirements and they will know that their input affected the product. This will require that all the task notes are stored in a digital format (as well as the task comments), giving the team or teams a way to track all the way from requirements to code. This could help the tester know what to test.

Part 4 – Release (red parts)
If a tag is added to the repository when a release is made it will be possible to see what differs between releases. However it will also be possible to create fixes to older release and branch out (for example if a customer do not want the latest version and only wants a small fix to his or her version)

Part 5 – Customer (purple parts)
If there is a log on what version a specific customer has (as well as a version number in every version of the software) it will be possible for a support team to quicker help the customer. Perhaps when a customer call customer support they enter their customer ID or version number then the support personnel could start up that version of the software before the customer have reached the support.

Part 6 – Test to requirements (orange parts)
By storing what tests and to what version (and vice versa) the tester (if there is an external test group) will know what new test they need to create when a new requirement is added. They will also be able to access all developer notes and therefore they will know how a specific requirement is supposed to work and what a correct/incorrect output is.

When a requirement is changed or removed it will be easy to remove and change the affected test cases. In some cases there are tests that tests more than one requirement. In these cases the test cannot be removed. Tracing requirements to test will also give the team as well as the customer a way to validate that all the requirements work and that the team delivers what they are supposed to.

Part 7 – Feedback from user testing (yellow parts)
By linking the feedback the team receives from user testing to a version it will be possible for the team to go back and run a specific version of the system at the same time as they read the user evaluation. This can give the developers and testers a greater understanding of the feedback. If the user data is also stored and linked to the version then it will be possible to run a system that is close to (if not the same) as the one the customer is running right now.

Part 8 – What others are working on (gray parts)
By storing what the teams and team members are working on (perhaps the team member need to check in and out stories when they work). By doing this everyone will know what the others are doing.

Discussion
Linking tools together as well as combining practices will help reduce the workload when adding traceability and by creating routines the team will get a rhythm for adding tracing data. The information gathered from the system in figure 13 could be used to help everyone involved in the project. One example is that if the maintenance team get a bug report and find out that it requirement A35 that contains a flaw, they can get a hint on where in the codebase they should start to look (by using requirements to code). By linking requirements to test they can also find out what test they need to fix. But because tracing information is supposed to be saved in both directions it can (in some cases) be possible to find what requirements and tests that are affected by changing some lines of code. However this will only give the developers a hint on what they need to change.

Benefits

- Easy to access information if we connect a search engine. (let the team Google the information).
- Possible to retrieve statistics. By writing a few lines of code the team and project managers no longer need to write any documentation where they present statistics.
- Statistics could be read in real-time.
- More transparency, it will be easier for persons outside of the team to follow the progress.
- Less risk of losing information.
- Could result in less time searching for bugs in large systems, the team could easily find where to start looking for bugs and what tests to fix if they know what requirement that is broken.
- Possible to see what requirements are tested, and therefore validated.
- If the team needs more information they will know where to go for more information.
- The customer will know exactly what is implemented in his version.
- The team will be able to find out what versions the customer is using.

Cost/drawbacks

- Modifying the requirements database so we can store from what source the requirements came from.
- Discipline when writing a commit comment.
- All information needs to be stored digitally (if you have paper notes you could scan and upload them to the server but they would not be searchable).
- The team needs to learn how to use the systems for handling the tracing information.
- It will take some time and money to implement the system.

5 General considerations on traceability

This chapter contains some general discussions and opinions on how and when to add traceability. There will not be any silver bullets that can be used to add traceability, but instead give you some thoughts on the subject. In the end it is up to the configuration manager to create a plan and the team to follow that plan. When there is a discussion about the project size it will refer to the both number of man hours as well as the size of the code base. This is a combination of the man power as well as the number of lines of code involved.

5.1 In which cases to add traceability

Not all projects needs to have traceability but some form of tracing is always useful. In this chapter there will be some discussion and some thoughts on when it could be a good idea to add and not to add traceability as a practice in the agile methods. Some projects will require a full scaled tracing practices suite and some will require only the simplest of practices.

5.1.1 Project size

In the beginning of the project it could be wise to ask the customer how long they think the software will be used. If they estimate that the software will live for about one year after the release then maybe we do not need all the tracing and documentation. If there is a short development period then it will be easier for the team to "keep the knowledge in the head" and the risks for losing team members are low. However if the team or customer notices that the project size or the life span for the project is about to change then it is better to add traceability sooner rather than later. However adding tracing information after the development is complete or halfway through is difficult and time consuming. Therefore it is not recommended to add full tracing at this point. If the team members, testers and everyone else that were involved document what they know about relations up to this point there will be some information that can be useful later on. It is important to make sure that the one using this information is aware that it most likely is incomplete and at some point could be inaccurate. Perhaps instead of tracing information they could write a summary of the project up to this point where they document the important parts. Therefore if the customer is uncertain about how long lived the software will be, the team should have some degree of traceability.

5.1.2 Small projects

If it is a small project it could be difficult to reach the point where the project would gain anything from tracing. If the project will not gain anything, tracing information should be classified as waste and should not be there. However that does not mean that the team will not have any tracing information. Some information is relatively cheap and the team will get it after implementing for example version control. Another part of Scrum and XP that could be called tracing (however not documented) is the daily Scrum and stand up meeting in XP. This will give the team a way of knowing what have been implemented since the last meeting and what will be implemented until the next one.

5.1.3 Short time to market

In some cases this could be similar to 5*.1.2 small projects*. Tracing and documentation should be limited, but not excluded. If the information could help improve productivity then it should be implemented. The problem is that most tracing practices will only have a positive ROI on longer projects where there, for example, is extensive testing and a maintenance period.

5.1.4 Complex project

When having a very complex product and where it can be hard for the team/teams to "keep it in the team", the team will need to start documenting their work. This is also an environment where some tracing techniques will of help. Therefore it is important that the CM has created a good plan for the team and that the teams are aware of why they need traceability and why they need to document their work.

One example of combining practices could be stakeholder to requirements, requirements to requirements and requirements to test. By adding this you get a way to see what potential risk there could be to implement a specific requirement. This could help developers to know what test they need to focus on as well as help the team with its estimation.

5.1.5 Maintenance

Maintenance

At the end of the projects development cycle the project will often be turned over to another team, in some cases to a maintenance department or outsourcing it to another company that will maintain the software for (in some cases) a long period of time.

In order for the maintenance team to do their job and for them to be a bit more efficient the development team should save tracing information. In the end this is one of the primary beneficiaries of tracing as they will receive information that the developers have spent time producing. This could possibly be the reason that traceability is something that is considered a waste of time, the ones that produce the information are perhaps not the ones that will gain the most from it. If in the end it will take less time finding and fixing a bug then that would mean that the developing company (or the customer if they have taken over the maintenance of the product) will spend less money on the product.

The maintenance team would benefit from information such as what code is involved in order to implement specific requirements, so if the team gets a bug report and can connect it to a specific requirement, they will get a hint on where to look. They will also gain from knowing why and how a specific requirement was implemented. This will reduce the time it takes to understand the solution.

Another set of information that could benefit the maintenance team is to know what test they will have to run in order to test the code. Perhaps they will have to create new tests and modify older ones.

Long maintenance time
If there will be a long maintenance time after a completed project it is important to have documentation and tracing information. In general the development team will turn the project over to a maintenance team and the maintenance team will have to maintain the product. In Scrum and agile in general this is not going by the book. In Scrum it is supposed to be the same team that maintain and develop the product. Everything should be added to the backlog and prioritized against each other [4]. Most of the people interviewed described another solution at their companies. They have one team for development and another team for maintenance.

In order for the maintenance team to do a good job the development team will have to get the knowledge that is "in the team" out on paper (or into a database). Some information that is needed is the tracing information. For example if the maintenance team will be able to find out what requirement that is broken and can trace that down to code it will reduce the time it takes to find the bug. If the maintenance team can also find out what tests that test this feature they will know where to look for faulty tests.

5.1.6 Long lived projects

In the case where the project will span over a long period of time (years) it is always a good idea for the team to document their work and for the CM to create a traceability plan. When having a long development time without documentation and tracing the information will get lost. People have a tendency to forget and team members will come and go. In the end it will be impossible to keep the information in the team. Long lived project tends to be large, complex or both. This can result in that the team will spend more time searching for where to do the changes and what requirements that they have tested. It can also be wise to save information on why a specific design was chosen and how it turned out.

5.1.7 Large projects

Large projects have a tendency to get complex and more often than not, tend to live for a longer period of time. Therefore most of what is written in 5.*1.4 complex projects* and *5.1.6 long lived projects* is relevant to large projects. If there is a risk that the large project will be long lived then 5.*1.5 Maintenance* is also relevant to large projects.

5.1.8 External or internal demands

A project can have traceability requirements from both internal (the company demands that all projects should be possible to trace) and external demands (the customer have some requirements on traceability).

External demands
When the requirements on traceability come from an external source, the team should make sure that the customer knows about the potential cost and benefits from tracing. It could also help motivate the team if they know for example why the customer wants some specific tracing information (unless the customer tells the team, "*because*"). However, if the customer is prepared to pay the price then the team has to do it.

If the customer does not know why he or she wants some specific tracing information and the team knows that it will be an overall cost, then perhaps they should persuade the customer into not asking for it (in the case where the customer do not want to pay for it). When big companies outsources projects (or sub projects) to other companies they will demand that the team saves tracing information. In cases like this there is not much to do about it.

Internal demands
Internal demands in this section will be demands that come from the organization that develop the product. Another view is that if demands come from outside the team it could be considered external demands.

When the organization grows or the company has high demands on for example stability and security they often have a group of configuration managers (examples are ABB and Sony Ericsson) that will demand that tracing information is saved. Perhaps this is not the optimal solution, but someone needs to see the entire development cycle in order to decide what information is needed for the other teams. But the teams are more likely to know what they need in order to do their job, for example the developing team is perhaps more likely to know what they need rather than a configuration manager group without direct contact or developing experience. But the developing team might not know what the testers and a maintenance group needs.

In one of my discussions with a configuration manager he told me that he does not really care which developing method the team uses, as long as they follow the CM plan and save the tracing information needed. In that case they have demands on being able to retrieve a specific version and the entire test suite as well as the test result from that version.

Internal demands can also be from the team itself. One of the comments was that some developers try using a version-control tool and after that state that they never want to work on a project without a version-control tool.

5.1.9 High demands on security and stability

Extreme demands
There is a big group of projects that will have extreme high requirements on stability and security. When a medical company develops new products they need to follow laws and regulation on the subject. For example if they need to deliver the medical products for the Swedish market they need to follow LVFS 2003:11 [23]. The same goes if they want to

release to the American market they need to follow the rules and regulation instituted by the American government.

It is not only medical equipment that has high demands on stability and security. Some other examples are train development, the defense industry, airplanes and nuclear power plants (will the Danish people ever forget Barsebäck?).

If the team is developing a system for environments where there are high demands on security and stability then it will be hard not to document without creating official documents. However that doesn't mean that they should not use automated tools. Perhaps it will be easier for the team to find the information they need if the information is stored together with requirements and if they can link it to a series of commits.

High demands
There are several types of project that demands security and stability, but not at the extent as you would when developing a new airplane. In these cases it could be better to reduce the amount of documents produced and setup a system that will enable the teams to test more often and have review meetings where the team (and perhaps external sources) could review and test the code. In order to do this in an easy way the team will need to at least be able to show the lines of code that differs between versions, perhaps more ideal would be to link it to requirements.

Insurance
If the software would fail, and the team have tracing and documentations of their work they could always use this in order to show that they have control over the project, for example if they have requirements->code and use code review it will show that it was not easy to find and that they did what they could in order to prevent this. Also if used in combination with requirements->test the team will have more evidence to support that this requirement was tested and the code was reviewed so it was something that was difficult/impossible to anticipate.

5.2 How to add traceability

This chapter will bring up some of the considerations one should take before and during the introduction of traceability as an explicit practice in the agile method at a company. Because there is no silver bullet when it comes to traceability the plan for tracing should be adapted at each company, in the best case the plan will be adapted to each project. There are several things that are important to consider when creating a plan, because in the end traceability needs to add some value to the project. The chapter will not give any explicit solutions on how to practically do it but will give some thoughts on what to think about and what to consider.

5.2.1 Processes

The first part is that the configuration manager needs to set up the routines for tracing when using agile software development. When using Scrum it would be good if the configuration manger consulted the team and listen to what they have to say (however it is the configuration managers right to ignore request if there is no basis for the request). The processes need to be adapted to the project (at least to some extent). In complex projects where it is demanded that certain documents or information is to be delivered together with the project, the CM plan needs to describe processes for collecting that information.

In smaller projects, or projects where there is no demands on specific documents the CM plan could be a bit more focused on saving the information in a way that is less formal (for example as comments on a web page, saving information using commit comments and so on.

The customer also needs to approve the plan, for example if the customer wants some specific tracing information he or she needs to know what it costs and be prepared to pay for it. However if the customer does not ask for it then the tracing information should be there in order to increase productivity or increase quality.

Tracing practices in Scrum and XP

In both Scrum and XP there are some practices that could be called traceability. But traceability without documentation, such as the stand up meeting and daily Scrum, this meeting will give the team a way to find out what was done since the last meeting and what will be done until the next one.

Another part in Scrum and XP that will give some form of traceability is the story cards and sprint log. It is possible to go back and find out what requirements that the team has implemented during the sprint/iteration.

5.2.2 Saving information

The tracing information should be saved during the sprint. It is important to document the code, tests and design while it is fresh otherwise it will begin to fade. If the information is saved in, for example, the repository during the sprint, it is still possible to write reports based on this information after the sprint. This could help the team be a bit more productive during the sprint at the same time as the information is stored and kept up to date. However this would mean that you would have to modify your “done” criteria (in Scrum) so the sprint is not over until certain documents are produced.

5.2.3 Costs and benefits

Traceability has both benefits and costs. When it comes to traceability, "*There is no such thing as a free lunch*" [24]. In the end documentation and tracing is not free, however the cost will vary depending on the degree needed. Some tracing practices, as described earlier, will help the developers, other will help the maintenance and some will help management and configuration managers. Some practices will only be a cost for the project but laws and regulation will enforce it upon the project.

At the same time as the routines are created there need to be cost/benefit estimations. The ones responsible should ask themselves *"What are the costs for implementing this tracing practice in this project?"* and "*What can we possibly gain from it?*". It is also important to inform the customer to know of the costs and benefits so that he or she can decide if it is something he or she wants to pay for.

If the customer wants to introduce tracing and it only leads to a cost for the team, then perhaps it should be placed as a user story for the team to estimate. This could work if it is something that only needs to be done during a few sprints. A tip received from a review was that perhaps it could be placed as a general story and that tracing costs should be added when estimating the stories. By doing this the customer will know that in order to get this he or she will get less functionality (or longer development time).

Parallel work

Tracing can help split the requirements into smaller requirements (that can then be split into tasks). What the team (or management) needs to do is split the requirements into smaller requirements and document the relationships. The team (or teams) could then be able to see the relations between the new requirements. When splitting a requirement it could also be good for the team to know what the others (teams as well as team members) are working on right now as well to be able to see what they have changed and why. If a team member can find out who wrote a specific set of code then he will not have to ask around and can go to that person directly (and thereby save time). However in order to work in parallel the team will also need to have a good development plan as well as support of tools.

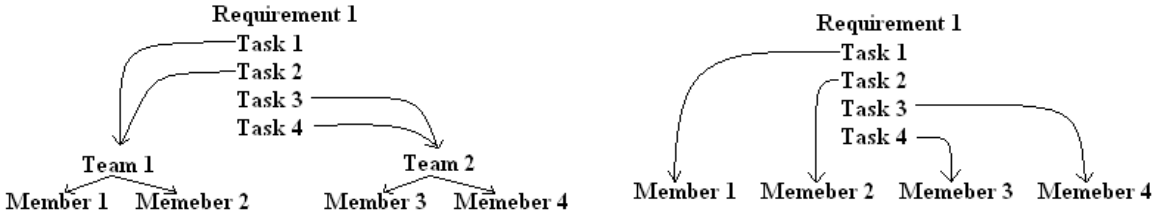

Figure 14 – Parallel work (on the left with two scrum teams and on the right with one scrum team)

Administrative work

Tracing will require some extra administrative work even if it is done right. In the end the configuration manager will have to create a plan on how the tracing information is supposed to be gathered and stored. The team will have to document their work (even if it is only a short commit comment). The more tracing practices the configuration manager adds to the team the less time they will have to write code (many small streams form a river). It is therefore important that the CM plan does not contain any (for the specific project) unnecessary practices. In the end if there is no external demands on tracing then the information gathered is supposed to help the teams, maintenance and test teams. Not add more work because it could "be good to have".

Less time coding
As there will be more administrative overhead and the team will have to (to some degree) document their work this could end up in less time coding. However it could also lead to less time searching for information. If the team is spending less time searching then perhaps they will find more time writing code and developing new features (or write more tests). However developing new features is not the only aspect of software development, so even if the team will find themselves spending less time writing code, perhaps others feel that their work is a tad easier. So in the end even if the team will spend less time coding, perhaps they will be a bit more efficient when they search for information and we could save money down the line.

Less time searching for bugs
In the end tracing information (used right) will help the teams find bugs faster, at least in large and complex projects (as well as projects that run over a long period of time). As explained in earlier chapters.

5.2.4 Tools

Can we have tracing in agile software development without the support of tools? In the article about Trustworthy Transparency Over Tiresome Traceability [25] Brad Appleton writes that he is "*in favor of using a good tracking tool* " and that he thinks that "*index cards simply do not cut it as a serious means of tracking storing,, storing, searching, slicing & dicing development/change requests*".

By using tools some practices could (in theory) be free. One example to explain this is version control tools. Version control tools are implemented in order to keep track of changes and help developers. It will give some advantages when saving information for traceability. This will not give complete traceability but could be used as a base for tracing. Tools will help the team gather information and store them in an informal way. The information stored could in the end be helpful for the teams that work on project. The information could also be used in order to gather statistics for the managers and customers. However when the information is supposed to be presented in a formal way, then perhaps the team will need to use the information gathered and write an official document.

Using tools can help create tracing that is not too heavy. Some routines can in fact be almost free once the tool is in use. One of these things can be how or what role that added

a requirement. Another feature that can be cheap once the system is in place (and if you have the right tool for the job) is requirements to changed code.

Benefits

- Higher probability that the information is stored and is up to date.
- Less time documenting and more time coding.
- Not to heavy routines.

Cost/drawbacks

- Initial cost of setting up the system.
- Cost of developing/buying the system.
- Education.

5.2.5 Searchable format

Something that could be easy to implement and have a great return on investment is to store all information in a searchable format. Sometimes the information is documented but it is close to impossible to find it when it is needed. So instead of the people involved with the product have to ask around for the information. Using a good search engine in combination with prewritten queries would lead to a reduction in the time it takes to find the information. If the time it takes to look for information is reduces the teams need to spend less time searching and less time asking around. If the team members do not work at the same location or someone is sick or absent it will be hard to ask around for the information. In these situations a solution where the teams can search for the information would be ideal. If the information is stored in a digital format and is connect to a search engine (for example Google[13]) it is easier to find the information you are looking for (less time looking, more time developing)

Benefits

- All information is easy to access.
- It could be possible to auto generate reports.

Cost/drawbacks

- Initial cost of setting up the system.
- It requires discipline from the team so the information get stored (for example commit comments, information stored on story cards).

[13] www.google.se

5.2.6 Documentation without documents

In order to make tracing agile the information needs to be stored in a way that does not create too much overhead (the ideal is no overhead or at least less overhead then the team gain from it). One step in order to achieve this is to try to create routines and use a set of tools so the information is saved without having to write documents.

For example if the team uses a version control tool then it is possible to use the commit comments to save information, then create a server script that collects and presents the information on for example a web site. If you save the requirement ID when you commit you can connect the comments to a requirement and you will have some tracing information without any document.

One of the people interviewed had a diary where he documented his work. This is another way to save information and documenting your work. An unofficial document like a diary will give the team a way to document in their own way. This is more a way for the person that writes the diary to be able to back track changes (for example save some information on why you made a specific design implementation). The information could also be helpful if a team member plan on leaving and need to get his replacement up to date. The team members can then go through the notes and see what the new member needs to be informed of.

5.2.7 Information overflow

Too much information will only make it difficult to find the important information when looking for it. Therefore it is important to limit the information saved and presented in order to minimize the time it takes to enter the information as well as the time it takes to find some specific information. If it takes too long time to search for information the team members as well as for example testers will go ask around instead. This results in that the information gathered being close to useless and it have resulted in an overall cost for the project.

5.2.8 Correct and up to date information

In the same way as tracing information can help teams involved it is important to know that incorrect or out of date information is dangerous (more than the lack of information). If the teams fail to keep the information up to date or if someone changed the requirements when they are implemented but do not change the original requirements then the project could be in deep trouble. In one interview it was mentioned that the requirements sometimes change but they cannot change the requirement in the requirements database so they do not. Then the testers test the function as stated in the requirement in the requirements database and it fails. Then the testers need to find out why and where it was changed.

5.2.9 Understanding

If the team do not understand or know why they are tracing and it is not clear that the documents or information gathered is ever used, they will most likely create something with less quality or fail to keep the information up to date (if it's a non automated procedure).

It is not only the team that needs to have understanding about tracing. The customer only needs to know that it could result in a cost and the configuration manager needs to understand that the plans needs to be a bit more flexible in order for the team to get more advantages from the agile developing methods. In the end perhaps all it comes down to is education in order to increase the knowledge and understanding of tracing.

5.2.10 Discipline

The Team

The main focus with traceability will be on the team. In the end it is the team that will do most of the documentation (also known as saving tracing information). In order for their work to be documented and that the documentation is up to date and correct the team needs to have discipline and follow the routines. The team needs to know that if they give incorrect information or forget to update some information it could hurt themselves, the testers and in the end the maintenance team.

Configuration manager

It is not only the team that will need to have discipline. The same also goes for the configuration manager to not add useless practices to the plan. He or she should also make sure that the team does what it is suppose to do.

6 Conclusion

Tracing done right will result in higher product quality as well as less time working on the project (time spend from gathering requirements until the product is faced out). There is however no silver bullet when it comes to tracing and it should be adapted to each project and organization. Most of the job with tracing will be done by the team and one of the claims that are relevant to this is *Write once, read many* [26]. The information saved during development could save the time for many team members, testers, managers and maintenance teams.

There are several practices that can be used separately as well as together with each other, using one of these could result in an overall gain but it is often better to try to combine them. Most of the practices described in the report will work well together and could be used without adding too much overhead. However some practices will be expensive and in the end result in an overall cost for the project.

In both Scrum and XP there are some practices that is in fact a form of traceability. Practices such as the daily scrum meeting (what others are doing). The tracing is in most cases informal and not documented. So by adding a little extra work at a few points in the agile methods would be a first step in introducing traceability.

Traceability should not be implemented in every project and especially not fully fledged traceability. In order to get the most out of the agile methods, traceability needs to be adapted. There are several considerations that should be taken into account, not only when adding tracing but when adding something new to the agile methods in general.

There is no such thing as a free lunch; this is also true when it comes to traceability. Adding traceability to the agile methods will in most cases result in more overhead for the team. But even if it results in overhead for the team other parts of the development cycle will gain from the information gathered.

As stated before other departments can gain more from tracing than the development team. Therefore it is important to not only take the development team into consideration when deciding on traceability, departments such as testing and maintenance will gain a lot from tracing.

In order to make heavy traceability more agile tools and automation will be needed. Tools and automation will result in an initial cost from implementing, developing or modifying the tools but will reduce the work load for the team and therefore reduce the time the team will have to spend saving tracing information.

If the information is stored digitally and in a searchable format (Google the information) then the team would spend less time searching for the information. It is also important to keep unnecessary information out of the system, too much information will result in information overflow and that will result in longer time to search for information and in the end the team will ask around instead of searching in the database.

7 Bibliography

[1] Agile manifesto
http://agilemanifesto.org/
Visited 2009-01-09

[2] The Trouble with Tracing: Traceability Dissected
http://www.cmcrossroads.com/content/view/6685/135/
Visited 2009-01-09

[3] Lean Traceability: a smattering of strategies and solutions
http://www.cmcrossroads.com/content/view/9089/264/
Visited 2009-01-09

[4] Agile Software Development with Scrum
Prentice Hall
Ken Schwaber, Mike Beedle
ISBN: 0-13-067634-9
Published 2002

[5] The enterprise and Scrum
Microsoft press
Ken Schwaber
ISBN: 9780735623378
Published 2007

[6] Extreme programming explained
Addison-Wesley Professional.
Kent Beck
ISBN-10: 0-201-61641-6

[7] Extreme Programming: A Gentle Introduction
http://www.extremeprogramming.org/
Visited 2009-01-09

[8] CONFIGURATION MANAGEMENT GUIDANCE (MIL-HDBK-61A)
http://assist.daps.dla.mil/quicksearch/basic_profile.cfm?ident_number=202239
Visited 2009-01-12

[9] Software Configuration Management patterns
Steve Berczuk; Brad Appleton
ISBN: 0201741172

[10] SCM in Scrum Projects
Lund University, Faculty of engineering, Department of computer science,
Andreas Bergström

[11] Software Configuration Management Practices for eXtreme Programming Teams
Lund Institute of Technology, Department of Computer Science, Sweden
Ulf Asklund, Lars Bendix, Torbjörn Ekman

[12] Scandinavian Network of Excellence in Software Configuration Management
http://www.cs.lth.se/SNESCM/Common/CMCM/
Visited 2009-01-09

[13] Brad Appleton's ACME Blog
http://bradapp.blogspot.com/
Visited 2009-01-09

[14] Smashing tablets
http://groups.yahoo.com/group/scrumdevelopment/message/12666
Visited 2008-11-04

[15] ETS032 - Software Development for Large Systems
http://www.cs.lth.se/ETS032/
Visited 2009-01-09

[16] Software Requirements Styles and Techniques
Soren Lauesen
ISBN-10: 0-201-74570-4

[17] Coaching av programvaruteam
http://www.cs.lth.se/EDA270/EDA270-2008/
Visited 2009-01-09

[18] Spårning av krav i agila projekt 2007-02-20
D04, Faculty of Engineering, LTH
Jonas Andersson (d04jad@student.lth.se)
Jonas Andersson (d04jan@student.lth.se)

[19] Entity-relationship model
http://en.wikipedia.org/wiki/Entity-relationship
Visited 2009-01-09

[20] Star schema
http://en.wikipedia.org/wiki/Star_schema
Visited 2009-01-09

[21] Star-schema-example.png
http://en.wikipedia.org/wiki/Image:Star-schema-example.png
Visited 2009-01-09

[22] Software Development in Teams – Project
http://www.cs.lth.se/EDA260
Visited 2009-01-09

[23] Läkemedelsverkets författningssamling
http://www.lakemedelsverket.se/upload/lvfs/LVFS_2003-11.pdf
Visited 2009-01-09

[24] There's no such thing as a free lunch
http://www.phrases.org.uk/meanings/tanstaafl.html
Visited 2009-01-12

[25] Trustworthy Transparency over Tiresome Traceability
http://bradapp.blogspot.com/2006/07/trustworthy-transparency-over-tiresome.html
Visited 2008-10-31

[26] Programmer as Reader
IEEE Software 1987
Adele Goldberg

Appendix A: Questions

1. Can you give a short description over how the requirements process works in your projects?
2. What type of tracing do your company seek/have
3. How do you think that tracing in agile and traditional project should work.
4. What artifacts do you or your team feel could be good to be able to trace between?
    a. What type of information do you feel is important to save
    b. What information does you or your team fell is important to save?
    c. Is there any type of tracing information that you don't feel any use for?
5. Who do you think should do the CM work in Scrum? What I want to know I primarily who should to the extra work that tracing brings. Who should design the routines and plans?
6. How can (if possible) tracing affect the quality of the product?
    a. Is there any specific tracing/connection/documentation that would help improved the quality more than any other?
7. How do you see that tracing (on both a long term and short term) could reduce the cost for development, testing and maintenance?
8. Is it possible to create tracing routines that do not demand so much extra administrative work, and therefore better suited for agile methods?
    a. Do you think that tracing is something that we want to/should have when using an agile method such as XP, Scrum and Lean?
9. What tracing methods do YOU think is interesting and why?
10. What is the value of every type of tracing?
11. Who can be interested in the different types of tracing (CM, team, customer)?
12. Why do you think that there is such a big resistance towards tracing?
13. Do you know of any tools that could make tracing easier and thereby reduce the workload on the developers?
14. How do you think tracing between requirements->requirements, requirements->version and so on could affect the possibility to make better estimation when we need to make changes?
15. If you should add tracing of requirements in Scrum, what level should you place it on how do you think it should be implemented?
16. How do you think that the cost would be affected
    a. Do tracing needs to be heavy?
    b. Can tools make it easier?
    c. Can the developers gain from tracing?
    d. Can it reduce the cost of turning it over to the maintenance group?

17. Do you have any experience with distributed Scrum or environments where they have several Scrum teams that work on the same project?
    a. If so, do you think that tracing requirements->requirements can help find functionality that can affect each other?
18. What can the cost be for implementing tracing in an agile method?
19. Can tracing increase productivity or the time it takes to find and fix bugs, why and how?
20. Can it reduce maintenance related costs, why and how?
21. What type of questions would you as a (PO, developer, and tester) like to have answers for where traceability could be helpful, and how valuable is it?