The Challenges to the Safety Process When Using Agile Development Models

Master Thesis

Hanne-Gro Jamissen

June 29, 2012
Halden, Norway

Østfold University College

Abstract

Safety related systems are traditionally developed using traditional models like the V-model. Agile development models are now increasingly used, and the experiences with these models makes it tempting to also use Agile models when developing safety related systems. To do this, Agile development models need include a safety process that also are as agile as possible. However, introducing safety activities into an agile environment reintroduces limitations from traditional development. The challenge is to reduce these limitations, so that the benefits of Agile development can be utilized also when developing safety related systems, and still maintain the expected level of safety.

The current thesis identifies and investigates some of the challenges to the safety process from IEC 61508 when using an Agile development process. The thesis starts by giving an overview of traditional development, Agile development and the safety process, before examining literature concerning Agile models used for developing safety related systems. To simplify the discussion, the safety process is grouped into three areas: safety analysis, safety assessment, and safety validation. Agile development is also divided into three groups: the incremental part, the iterative part, and main Agile practices not covered by the two first discussions.

The discussion starts with examining all the incremental implications to each of the safety process groups. Then the iterative implications are discussed, and finally the implications of the remaining practices are considered. The discussion is summarized by giving a suggestion for a more agile safety process, based on the Scrum model.

Keywords: Incremental safety process, iterative safety process, Agile safety process, hazard identification, hazard analysis, safety analysis, functional safety assessment, safety validation, software development, IEC 61508

Figure 1: 50 most used relevant words in the current document (created using www.wordle.net)

Acknowledgements

I want to give a special thank to my supervisor, Monica Kristiansen at Østfold University College, that helped me structuring my thesis when I needed it most.

Table of Contents

Chapter 1

Introduction

Iterative and incremental development (IID) has become more and more popular in software engineering in the last two decades[1]. In the current decade the focus is on the more light-weight IID models known as Agile. Agile development models target problems with handling changing requirements and involve users in the developing process. Agile models focus less on documentation and plans than traditional development models.

Safety related software development is more formalized through standards [39]. The development of these standards started when traditional development models (waterfall, V-model) were considered best for large, complex systems. Though the standards do not explicitly require traditional development models, the standards indicate that these models should be used. IID and Agile models are not mentioned to any extent. Development of safety related systems requires documentation that the system does not represent an unacceptable danger to humans or environments. The systems are often subjected to an independent certification before they are allowed to be put into operational use.

1.1 Problem statement

Agile software development models are now commonly used. It is said that these models produce software of higher quality at a lower cost and improve the satisfaction of both employers and stakeholders [14]. The increased use of Agile models results in an increasing number of developers who master Agile models. The decreasing use of traditional models evidently results in a decreased

[1] West and Grant [45] refer to a 2009 survey indicating that 35% use Agile and 21% use other IID models, while 13% use Watefall-like models. 31% do not use any formalized development model

number of developers who masters traditional models. This means that companies developing safety related software also want to try Agile models. However, safety was not an issue when Agile development models was defined, and applying these models uncritically cannot be recommended.

The users of Agile development models claim that these models give better quality at a lower cost [14]. Since better quality does not equal better safety, this claim is not directly relevant when developing safety related systems. Seen from a strict safety perspective a central question is if Agile models give the same level of safety as traditional development models. Seen from an economical perspective, a central question is whether Agile development reduces the cost, and still maintains the level of safety.

There is ongoing research considering whether Agile models can be used for developing safety related systems [42] [30]. The focus in these articles is the development model itself. When safety process activities are mentioned, it seems to be assumed that the safety activities can be performed as for the traditional development models, or it is suggested that this should be further investigated. When searching for literature concerning how to perform extra safety activities within Agile environments, like initial hazard analysis and functional safety assessment, the results are rather sparse. However, some recent articles also mention aspects of the safety process [16] [40]. Two central questions arise:

- How may Agile development affect the safety process?
- How may the safety process affect Agile development?

The focus is to maintain the level of safety offered by traditional development, but at the same time maintain most of the agility that is the success of Agile development. If the necessary changes to Agile models compromise agility, Agile benefits are lost and the justification for using Agile models are not present. In addition, adapting the safety process to an Agile development model should not make the project more expensive than using a traditional safety process and a traditional development.

1.2 Scope

The scope of this thesis is to identify the areas of Agile development that cause changes to the safety process and the requirements of the safety process that impose changes to Agile development. The focus is to maintain the level of safety that is given by the safety process in traditional development. This should be the motivation for any proposed change in either the safety process or the Agile

development models. When the safety rationale is covered, the economical aspect is investigated. The economical effects is either extra costs added to the safety process by the underlying Agile development model, or loss of agility that decreases the expected cost reduction promised by Agile development models.

The main purpose of the current thesis is to identify areas that need to be changed. It is also suggested specific changes to the safety process and Agile development models. The focus is to build up Agile models [9] by adding safety considerations, and not to tailor down the safety process by introducing elements of Agile. However, the suggested solution is not claimed to be the best solution, or the complete solution. This is meant to be input for discussion when a concrete project is planned.

As the thesis focus on implications on specific safety activities, and not the software development model itself, there will be certain areas that are **not** covered:

- **Recommendation of development model**
  The current thesis focus on the safety process, not the development model itself. There will be no recommendations on how to use Agile development models or Agile development models for developing safety related software unless it is required by the safety process. There will neither be discussed whether Agile models are better than traditional methods for developing safety related software. When the current thesis refers to Agile software development models, it is assumed that the software development models are modified to comply with the relevant safety standard.
- **Use of formal methods**
  It will be no investigation of whether Agile development models are suited to be used with formal methods. In Sommerville [39] formal development are stated as a variant of the waterfall model and thus outside the scope for the current thesis. Use of automated safety assessment [10] requires formal methods, and is therefore not considered.
- **Hardware development**
  A lot of safety related systems contain both hardware and software development. The current thesis will only focus on the safety process related to the software part of the system development as Agile development models investigated are software development models. The coordination between the hardware and software development may be more important and more difficult when Agile development models are used, but this problem is not investigated.

- **Tools**
  Software development uses a large range of tools for developing, coding, testing, change management and so on. This thesis will not discuss the use of any specific tools. However, it is understood that tools are used, and must be used, to be able to perform the software development and the safety assessment efficiently and effectively. Note that tools used in development of safety related systems should have been subjected to a safety assessment of the same level of criticality as the system that is using the tools, especially when the tools are actively generating the whole or parts of the final system. Examples are generating code from UML, compilers, use of class libraries, etc. This is not further discussed.
- **Programming Languages**
  Agile models are often used with programming languages like Java, that are not among the languages recommended developing safety related systems [20]. The focus of the current thesis is the process, not techniques or programming languages, and use of programming languages is not discussed.

1.3 Methods

The method used in the current thesis is a literature search. Several research areas are investigated:

- The strengths and weaknesses in Agile software development models.
- Scaling of Agile development models as safety related systems often are large.
- Research concerning use of Agile development models for developing safety related systems.
- Research concerning the safety process within an Agile environment.
- Safety standards and their requirements to the safety process.

The information obtained in the literature search is sorted and discussed in respect to the requirements to the safety process.

1.4 Document outline

Some essential definitions and clarifications are given in Chapter 2. The complete list of definitions and abbreviations is given in Appendix A. The background chapter, Chapter 3, starts by giving a short introduction to traditional development models, which still are recommended for developing safety related software. In addition, basic elements of several Agile models, including a historical

view of some early incremental and iterative models, are shown. The chapter also contains an introduction to areas concerning development of safety critical systems. Chapter 4 gives an overview of the main articles and books that are the basis for the discussions in the succeeding chapters.

The discussion is separated into 5 chapters. Chapter 5 gives an introduction to the discussion. Chapter 6 discusses the effects of the incremental part of Agile on the safety process, Chapter 7 discusses the effects of the iterative part of Agile, Chapter 8 discusses the effects of Agile practices not discussed in the previous chapters, and Chapter 9 summarizes the discussion by suggesting a more safe Agile development model. This model is based on Scrum. Finally, Chapter 10 concludes and gives suggestions for further work.

Chapter 2

Essential definitions

This chapter contains the most essential definitions used in the current thesis. A complete list of definitions and abbreviations can be found in Appendix A.

2.1 Software development

Sommerville [39] defines the software process as *the related set of activities and processes that are involved in developing and evolving a software system.* He also states that a model is the description of a process. The terms software development process and software development models are often used as synonyms, even though there are slight differences.

2.1.1 Monolithic development

In monolithic development, all functions of a system are developed in parallel and then integrated at the end of the development. Monolithic development is the opposite of incremental development [19].

Monolithic development is also referred to as:

- BDUF - Big Design Up Front - focusing on the large amount of initial work needed.
- Big bang integration - focusing on integration and testing, and late detection of errors.

2.1.2 Sequential development

The waterfall model [39] is a sequential development model. It consists of separate and distinct phases of specification and development. The phases contains activities such as requirement specification, software design, implementation, testing. The result of each phase is approved before the next phase can start.

Sequential development models are often used as a common term for development models that are both sequential and monolithic and rely heavily on documentation for transferring information from one phase to the next. Sequential development is the opposite of iterative development.
Sequential development models are also referred to as:

- Plan driven development models [39] [9] - Processes where all of the process activities are planned in advance and progress is measured against this plan. Often associated with sequential models, but can also be used for other models.
- Traditional development models [9] - Used in contrast to modern models as Agile.
- Heavy-weight models - Focusing on the amount of ceremonies and documentation.
- Disciplined [9] - Focusing on compliance with an established process.

The current thesis uses the term *traditional development process*, meaning a process that is both sequential and monolithic.

2.1.3 Incremental development

Cockburn [13] defines incremental development as:

> *... a **staging and scheduling strategy** in which the various parts of the system are developed at different times or rates, and integrated as they are completed*
>
> It neither implies, requires nor precludes iterative development or waterfall development - both of those are rework strategies. The alternative to incremental development is to develop the entire system with ”big bang” integration.
>
> Incremental development helps you improve your process. Each time around the process, you get to change and improve your work habits.

In incremental development the activities are interleaved instead of separate, with rapid feedback across activities [39]. Incrementa development is the opposite of monolithic development.

2.1.3.1 Incremental delivery

In incremental delivery [33], an early version of the software with limited functionality is delivered to the customer, and then new versions with added functionality are delivered until the system is finished. The purpose is to give the customer value in form of a working system earlier than otherwise and to gain early feedback from the customers. The feedback assures that the final system is what the customer wants.

Incremental development does not require incremental delivery, but it is incremental delivery that gives the benefits of early feedback [33]. Incremental delivery cannot be obtained without an incremental development model.

2.1.4 Iterative development

Cockburn [13] defines iterative development as:

> *... a **rework scheduling strategy** in which time is set aside to revise and improve parts of the system.*
>
> It does not presuppose incremental development, but works very well with it. ... the difference is that an increment may actually ship, whereas an iteration is examined for modification.
>
> Iterative development helps you improve your product. Each time around the process you get to change and improve the product itself (and maybe some of your work habits).

Iterative development is the opposite of sequential development.

2.1.4.1 Incremental / Iterative confusion

Iterative and incremental are two essential terms in software development. They often seem to be synonyms, but they are not. As the previous definitions show, incremental is staging, iterative is rework. This difference is illustrated in Figure 2.1.

2.1.5 Agile development

Agile development models are a set of incremental and iterative development models that encourage agility, i.e. rapid and flexible response to changes. An Agile development model [26] has short

Figure 2.1: Incremental vs. iterative development [32]

increments, enabling frequent releases to the customer to gain feedback, and are low on ceremonies (documents and formal steps).

Agile development is also referred to as light-weight models, which are models that are light on ceremonies and documentation [26].

2.1.5.1 Handoff

Handoffs are defined as *a separation of knowledge, responsibility, action, and feedback* [14].

Sequential development models typically have handoffs between the development phases, where the output from one phase is delivered to a new team that will continue the development. Agile models try to avoid handoffs to make development more efficient.

2.2 Safety

Storey [41] defines safety as *a property of a system that it will not endanger human life or the environment*. Another definition is that safety is *freedom from unacceptable risk* [20].

2.2.1 Safety related system

Safety related system is one by which the safety of the equipment or plant is assured [41].

IEC61508 [20] defines a safety-related system as a designated system that both

- implements the required safety functions necessary to achieve or maintain a safe state for the EUC; and
- is intended to achieve, on its own or with other E/E/PE safety-related systems and other risk reduction measures, the necessary safety integrity for the required safety functions

A safety related system may also be referred to as:

- Safety critical system - Is often used as a synonym, but may sometimes be used for systems of high criticality [41].
- High integrity software - Includes safety related software, but is more broad-ranging as it also covers systems with high economic or security risks.[1]

2.2.2 Safety Integrity Level (SIL)

IEC 61508 [20] defines SIL to be a *discrete level (one out of a possible four), corresponding to a range of safety integrity values, where safety integrity level 4 has the highest level of safety integrity and safety integrity level 1 has the lowest*. The safety integrity is defined as the *probability of an E/E/PE safety-related system satisfactorily performing the specified safety functions under all the stated conditions within a stated period of time*

The determined SIL for a system influences the development process and the safety process. IEC 61508 recommends a set of activities to be carried out for each SIL. Both the development process and the safety process must be tailored to fit the IEC 61508 recommendations for the given SIL.

[1] http://hissa.nist.gov/pubs/nist204.pdf visited 8 March 2012

2.2.3 Safety process

There is no precise definition of a safety process. This paragraph reflects how the term is used in the current document. The safety process is the extra activities involved in the development of a safety related system that interacts with the development process. Safety activities integrated in the software development process are not considered a part of the safety process. The safety process runs prior to, in parallel with, or after the software development. In the current thesis the safety process covers the activities given by the IEC 61508 [20] overall safety life cycle, excluded the actual realization part.

Chapter 3

Background

This chapter describes models that can be used to describe a software development process. The models are selected from different families of software development models. The families of models are briefly described and summarized before individual models are described in detail. This chapter emphasizes the parts of the models and the differences between models that may be important regarding safety and traceability.

The second part of this chapter contains a description of the safety process, with emphasis on IEC 61508 [20]. Finally a brief description of traceability, an important basis for the safety argumentation is given.

3.1 Software development models

The first software development process models used were inspired by hardware engineering methods [8]. These models are referred to as traditional models, and their most typical features are that they are sequential and requirement driven. The Waterfall-model and the V-model are both examples of traditional software development models. They are associated with monolithic development, witch involves no delivery until the complete system is finished, a lot of documentation and rigid procedures to follow.

The next generation of software development models targeted the resistance of change in traditional models. These models are both iterative and incremental. Examples include the Spiral-model and the Unified Process. These models are still relying on written documentation. The Agile models are also iterative and incremental, but they have introduced a new way of organizing the software

development as stated in the Agile manifesto [1]. Scrum and Extreme Programming are two commonly used Agile models. There exists an Agile version of the Unified Process called the Agile Unified Process. The Agile processes are often called light-weight processes, as they have less written documentation and less formalism. These processes can be described as:

- **Iterative** in contrast to **sequential**
- **Incremental** in contrast to **monolithic**
- **Agile** or light-weight on documents and formalism in contrast to **heavy-weight**

3.1.1 Traditional software development models

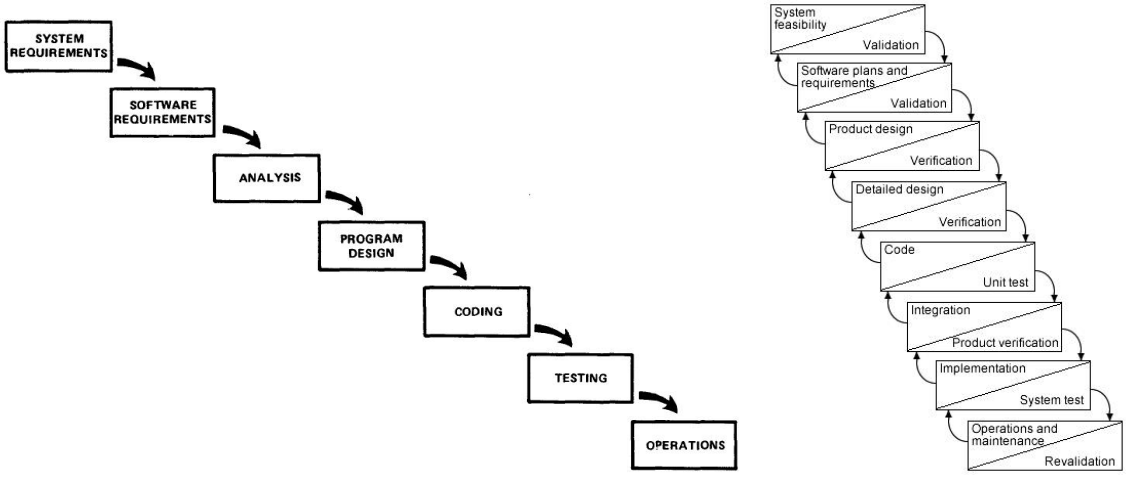
Traditional software development models are sequential and monolithic. They rely on extensive documentation to pass on the information from one development phase to the next. One of the earliest traditional models is the Waterfall model. An elaborated version of the Waterfall model is known as the V-model, which emphasizes test-activities.

3.1.1.1 Waterfall

The origin of the waterfall model is not very clear, Sommerville [39] indicates that this model was introduced by Royce [35] in 1970, but Royce uses a model close to the waterfall model as a concept for the steps necessary in implementing a computer program. He concluded that this implementation is risky and invites failures. Then he addresses the risks and ends up with a more iterative model. Boehm [8] mentions a software development model developed by hardware engineers and used as early as 1956. This model is quite similar to the waterfall model and the V-model.

The waterfall model [39] is also a plan-driven model. The project is planned and scheduled before the work can start. It consists of separate and distinct phases for specification and development. The phases contain activities such as requirement specification, software design, implementation, testing, and operation and maintenance. The result of each phase is approved before the next phase can start. However, in practice these phases overlap and give feedback to earlier phases in some extent.

Traditional software development is often associated with ”big bang” development, i.e. monolithic software development. This means that the complete software is developed simultaneously, and the complete software is integrated and tested at the same time. This results in late discovery of discrepancies. The system is not deliverable until all modules are integrated into the total system.



(a) Concept from Royce [35] (b) Update from Boehm [7]

Figure 3.1: Waterfall model

This is also called ”big design up front”, which means that the complete system is designed before implementation is started. This approach normally uses specialized staff, the architects do the architecture, the programmers do the coding and so on. Communication relies heavily on written documentation where the output from one phase serves as input to the next phase.

One of the strengths of the waterfall model is simplicity. The activities in the waterfall model fit directly into a time line, and documentation is produced in each phase. The process is visible and the progress can be monitored against the development plan [39].

One weakness of the waterfall models is that separate activities may lead to the need of specialists in each activity. Transferring from one phase to the next may also lead to transferring from one team to another. Another problem may be that non-documented assumptions and interpretations are not transferred to the next team.

Larman [26] says that the waterfall model *requires a low-change, low-novelty, and low-complexity problem*. He also points to some important problem areas of the waterfall model:

- ”Complete” Up-front Specifications with Sign-off
- Late Integration and Test
- ”Reliable” Up-front Schedules and Estimates
- ”Plan the Work, Work the Plan” Values

Sommerville [39] recommends the Waterfall model only for projects where the requirements are

well understood and not likely to change much during the development.

3.1.1.2 V-model

The V-model is based on the waterfall model, but the activities are shown in a V-form with emphasis on testing; each activity in the left side on the V has a corresponding test activity. On the right side in Figure 3.2, a specific test activity is defined on the basis of the output from the corresponding development activity and should be defined as early as possible, preferably before the implementation is made.

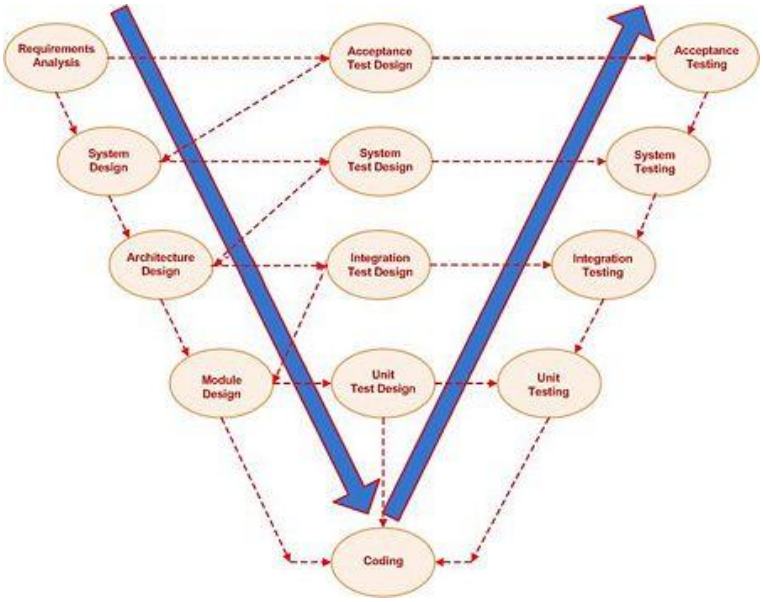


Figure 3.2: V-model (http://en.wikipedia.org/wiki/Image:V-model.JPG)

3.1.1.3 Change management

Change management [39] is the process of controlling changes to a system. The goal is to predict the change impact and control the costs. Change management is considered a part of configuration management along with version management, system building and release management.

Traditional development models are criticized for their inability to handle changes during the development. The description of these models do not indicate that feedback to previous development phases is given. In real life, however, changes are handled also for these types of models. Since

the models are open for interpretations, there are local guidelines, written or unwritten, on how to handle changes. Sommerville [39] describes an ideal change management routines for changes occurring after the software is released:

- A change request is received
  The change request can be an error description or a request for new functionality

- The change request is checked for validity
  The change request could be based on a misunderstanding, or the requested error correction or feature is previously reported and handled.

- Change impact analysis and cost estimates are performed.

- Change control board (CCB)
  The change control board evaluates the benefits and cost of the change, the consequences of not making the change and how many users this change would benefit.

- The software is modified
  If approved by the CCB, the software is modified and all related documents are updated.

- The software is tested.

- The change request is closed.

In the development phase, a simplified change management procedure is used [39]. Problems and changes should still be recorded, whereas changes with limited impact can be approved or rejected by the developer. Changes involving modules handled by different teams should be approved by an independent authority, e.g. the system architect.

In sequential development, the documents are approved at the end of each phase. However, changes and errors are detected throughout the development, and they often affect previous phases. Sommerville [39] warns that the rework needed to modify and approve documents can be expensive, and that this rework may not be done in earlier phases. Problems may be ignored, programmed around or left to be handled later. The result is badly structured systems and documentation that is not consistent with the code.

3.1.2 Agile software development models

Agile software development arised from the early incremental and iterative development models. One of the main justifications for introducing iterative and incremental models is better handling of changes.

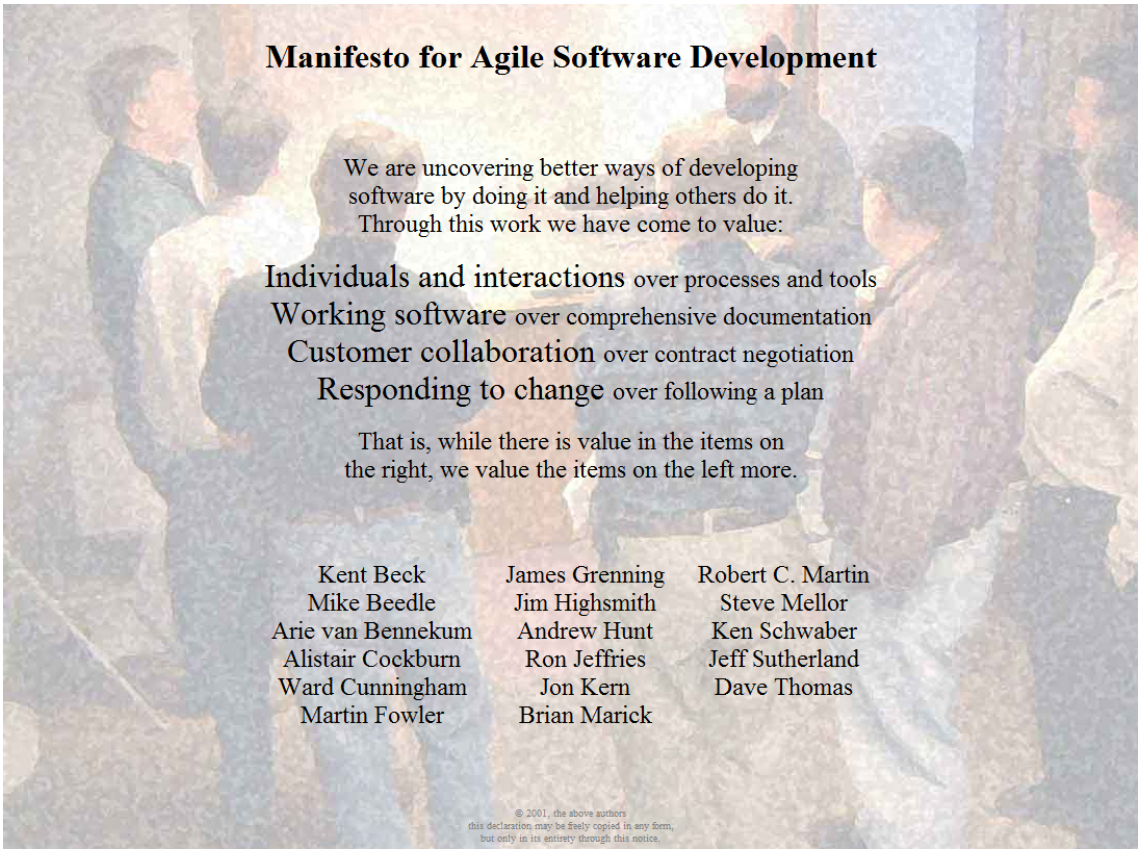


Figure 3.3: The Agile Manifesto [1]

The first iterative and incremental models that were defined, have larger increments and longer iterations than used by Agile models. These models focused more on iterations (improving and refining the software) than increments. Incremental development is used since new functionality is added as the project progress, but little use of incremental delivery and customer feedback, thus incremental delivery has been incorporated in later versions of the models. These early models (the Spiral model and RUP) still relies in written communication in contrast to Agile models. Some examples of early incremental and iterative development models are the Spiral model and the Unified

Principles behind the Agile Manifesto

We follow these principles:

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

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

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

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

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

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

Working software is the primary measure of progress.

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

Continuous attention to technical excellence and good design enhances agility.

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

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

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

Figure 3.4: The Agile Principles [2]

Model. However, these models were not Agile when they were first defined, though Agile instances of these models may have been defined later.

The Agile models refer to a group of software development models with a set of common values (see Figures 3.3 and 3.4), rather than a specific model. The name Agile has been used from 2001 when the Agile manifesto (Figure 3.3) was signed,. However, some of the models now considered as Agile already existed before this time. Two of the most popular Agile models are Scrum and Extreme Programming (XP). Derived from XP, the practice of Test Driven Development is often used separately, as a separate Agile model.

This section starts with a presentation of the Spiral model and RUP to show the history. The main presentations concern Scrum and XP.

3.1.2.1 Spiral model

The spiral model [7] is a risk driven process, where each iteration starts with an analysis of the project risks. Then the project develops the functions that are found to be most risky. Each cycle in the spiral is completed with a review of the product developed in this cycle and planning of the next cycle, see Figure 3.5. The project risks are then reanalyzed. On the basis of this new analysis, the project is stopped or continued. Project risks must not be confused with safety risks. Project risks are risks related to the project realization only, e.g. risks regarding the economy, technical issues and available staff and their skills.

The last cycles of the Spiral model consists of Waterfall-like development. The main difference is that this is not started until the most risky parts of the project is investigated. The knowledge gained from the prototype phases are used when developing the system.

3.1.2.2 Unified Process

The Unified Process (UP) was originally developed to support object oriented development using UML. The Unified Process, also called the Unified Software Development Process (USDP) [22], is a framework rather than a specific process model. This gives the users a great flexibility in defining how they should use UP. The resulting model can be close to traditional models, it can be Agile or it can be anything in-between. It is defined an Agile instance of the Unified Process, the Agile Unified Process (AUP). The commercial version of UP is called Rational Unified Process (RUP). RUP contains more detailed descriptions and guidelines (best practices) related to all the elements of UP.

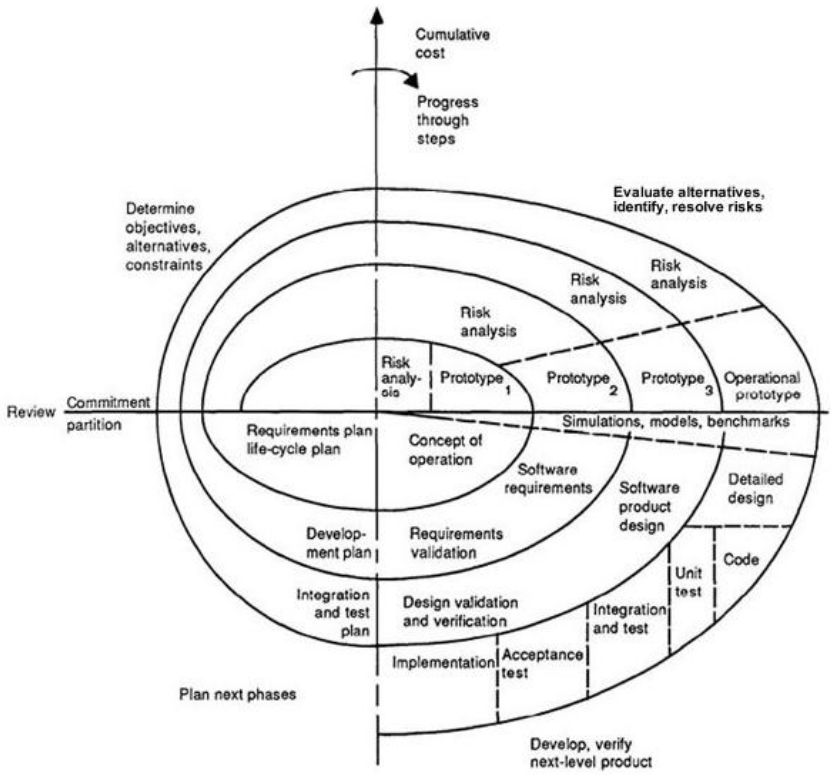


Figure 3.5: Spiral model [7]

The disciplines in Figure 3.6 resemble the phases of the waterfall- and V-model. The main difference is that the time line does not follow the disciplines. For traditional models, the disciplines (what to do) and the phases (when to do it) are the same. For UP disciplines and phases are separated. This enables the disciplines to be repeated for each phase; one phase consists of several iterations. Figure 3.6 shows that the weights of the disciplines differ between the phases; the requirements are heavy in the inception phase, while implementation is heavy in the construction phase. However, all disciplines exist in all phases. The corresponding figure from [22] shows only the five core disciplines (called workflows), Requirements, Analysis, Design, Implementation and Test.

The iterations in Figure 3.6 are combinations of iterations (re-do) and increments (add-on). An iteration allows fixing the wrongs from the previous iteration, and adding new functionality. Jacobson et al. [22] use iteration and increment as synonyms and do not follow the definitions given in Sections 2.1.3 and 2.1.4.

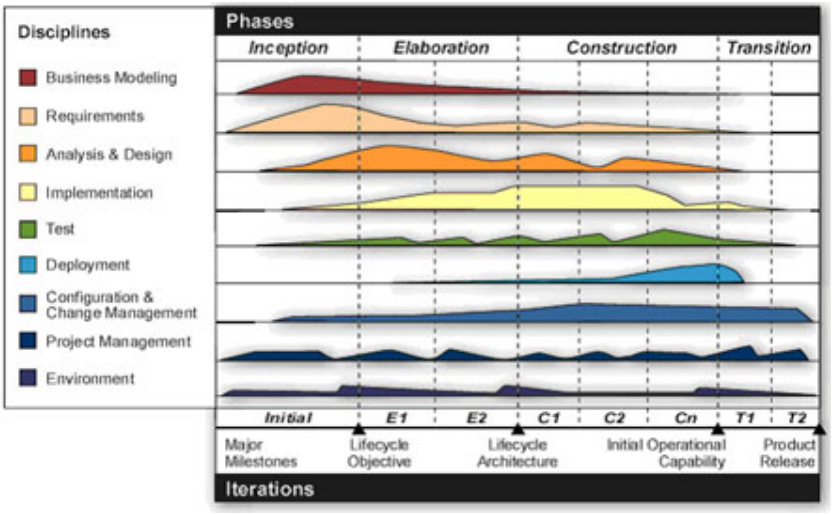


Figure 3.6: IBM's Rational Unified Process [22]

The key modeling elements of UP are then:

- Roles - who
- Activities - how
- Artifacts - what
- Workflows - when

A role is associated to individuals or groups during the project. One individual may have several roles and one role may be performed by several people. The role specifies the competence and responsibility the person executing this role should have. The activity of a specified role is a unit of work. The activity has a clear purpose, creating or updating an artifact. A *role* will perform an *activity* producing an *artifact*. A *workflow* are a sequence of activities that produces a valuable result and show interactions between roles. UP, like traditional models, relies heavily on written documentation (artifacts), but can be configured to be more light-weight.

3.1.2.3 Scrum

Both the Spiral model and the Unified Process are iterative and partly incremental, but they both have some reminiscence from traditional development models such as distinct development stages and written documentation. The Agile models are incremental and do not have any distinct development stages. In addition, face-to-face communication is preferred over written documentation.

Scrum is mainly a framework for project management and is often used together with a more

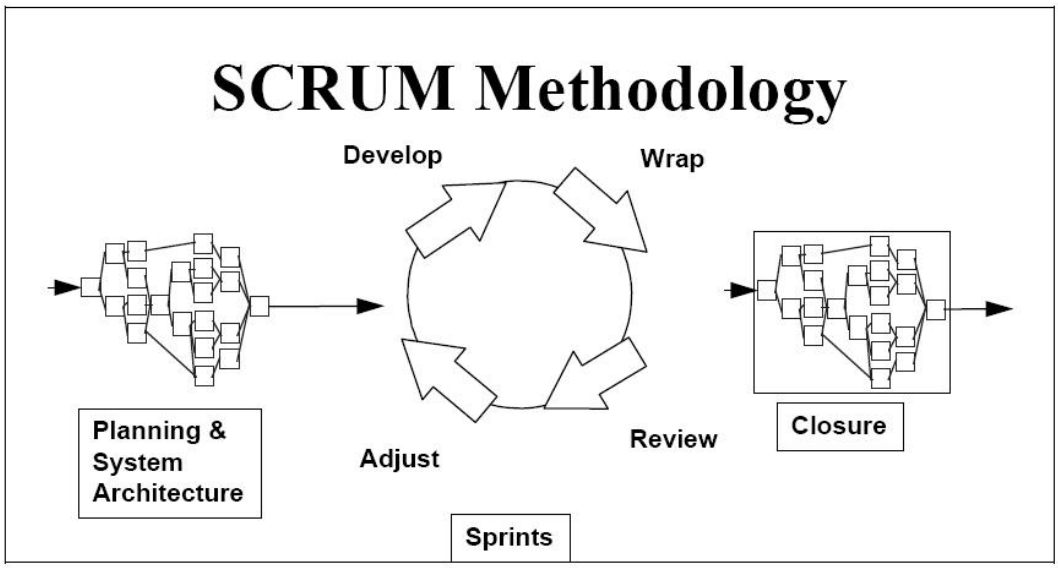


Figure 3.7: Scrum overview [36]

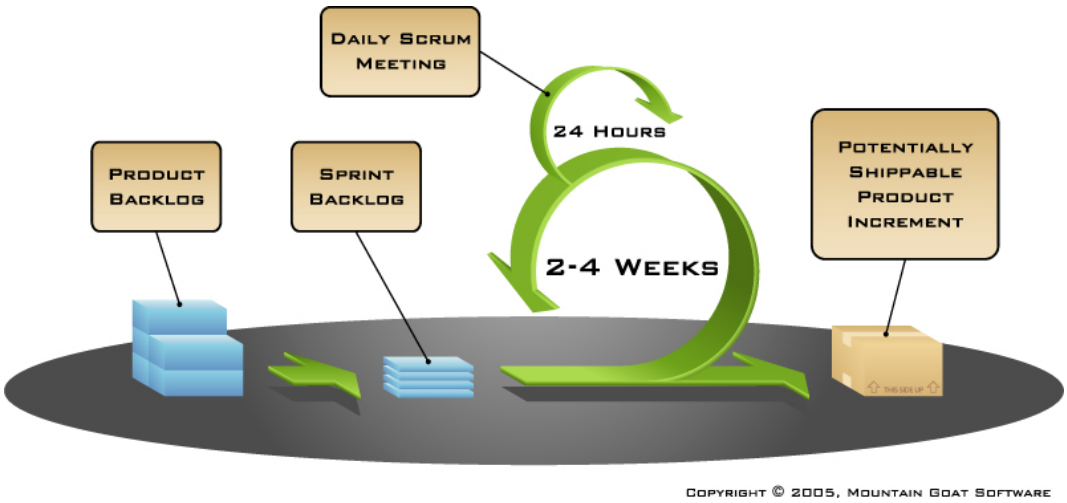


Figure 3.8: Scrum work flow [38]

specific software development model. The combinations of Scrum and Extreme Programming, or Scrum and Test Driven Development, are often used.

As shown in Figure 3.7, the early Scrum divided the development into three main phases; the Pre-game containing the planning and the system architecture, the Game containing the sprints

and the Post-game containing the closure. Whereas the Pre-game and the Post-game are following defined processes, the Game are highly adaptable. The 2011 Scrum-guide [37] has no references to the Pre-game and Post-game, this is no longer considered within the scope of Scrum. Figure 3.8 shows the work flow for the Game phase.

In the following a short summary of the 2011 Scrum Guide [37] is given. Scrum is based upon three pillars, transparency, inspection and adaption. It consists of the Scrum Team, the Scrum Events, the Scrum Artifacts and a common Definition of Done.

- **The Scrum Team:**
  - The Product Owner
    is responsible for the Product Backlog; for ordering the items, keeping the backlog visible and clear to all. The Product Owner is one person, not a committee.
  - The Development Team
    is responsible to deliver a potentially releasable increment (according to the Definition of Done). The team is crossfunctional, i.e. have all the skills needed to produce the increment among the team members. The team does not contain any specialized sub-teams. The team size should be in the range of 3 to 9 members.
  - The Scrum Master
    is responsible for ensuring that the Scrum is understood and used properly.

- **The Scrum Events:**
  - Sprint Planning Meeting
    plans the next Sprint in detail and establish the Sprint Backlog. It has two parts, first deciding what to deliver in the next Sprint, then deciding how to achieve this.
  - The Sprint
    produces a “Done”, usable and potentially deliverable product increment. Sprints have constant duration. The Sprint scope may be renegotiated with the Product Owner as more information is gained during the Sprint.
  - Daily Scrum
    is a 15 minute meeting very day to synchronize activities an plan for the next 24 hours.
  - Sprint Review
    is where the developed increment is inspected. It is identified what has been “Done” and what has not been “Done”, and the new increment is demonstrated.

  - Sprint Retrospective
    is where the Scrum team inspect itself to identify improvements to the way they work.

- **The Scrum Artifacts:**
  - Product Backlog
    contains an ordered list of everything that might be needed in the product. This is the single source of requirement of changes to the product. The Product Backlog is never complete.
  - Sprint Backlog
    contains the Product Backlog items to be developed for the current Sprint, and a plan for delivering the increment.
  - Increment
    is the sum of all backlog items that are developed during the sprint and the previous sprints. An increment is potentially deliverable.

- **The Definition of "Done"**
  contains all the steps necessary to transform the Sprint Backlog items to a deliverable increment.

3.1.2.4 Extreme Programming (XP)

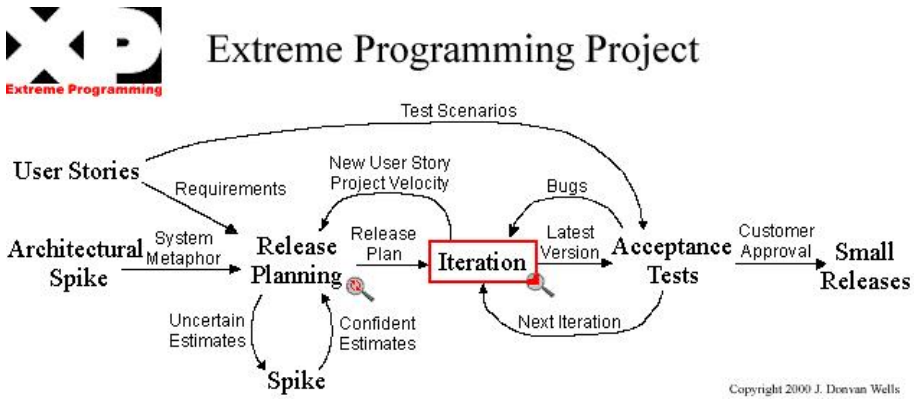


Figure 3.9: Extreme Programming flowchart[44]

Extreme programming is an Agile software development model often used to complement Scrum. Some of the XP practices fits nicely into Scrum, as the Planning game (Scrum Sprint

planning) and the Short iterations (Scrum Sprint). The XP User story is equivalent to the Scrum Product Backlog item. Other aspects of XP complements Scrum in areas where Scrum does not have detailed content, like use of Test Driven Development, Pair programming and Refactoring.

Some of the original twelve core XP practices [6] has been renamed to clarify the meaning. The current chapter uses the practices as described in [23] with the original name in brackets. These practices are visualized in Figure 3.10 and briefly explained below:

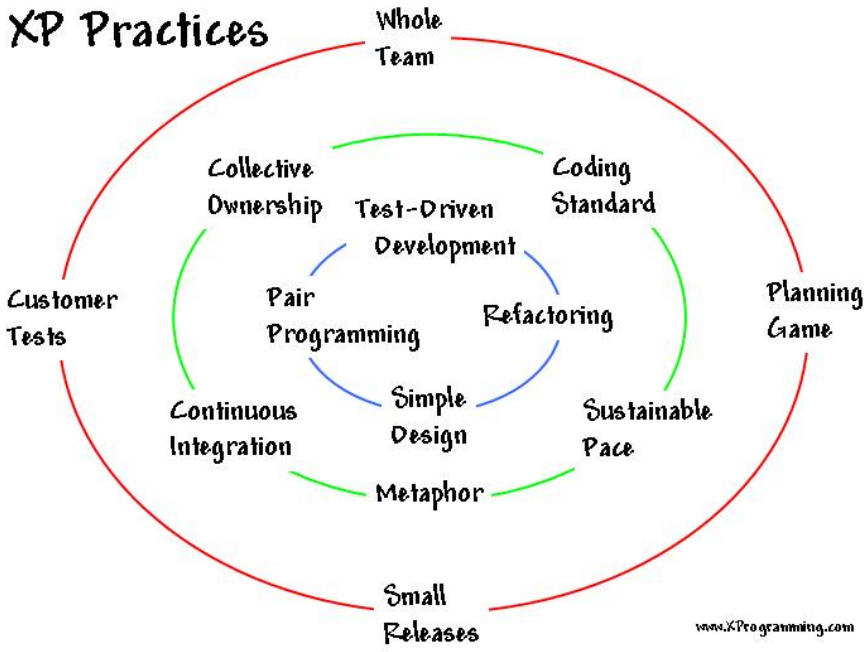


Figure 3.10: Extreme Programming practices [23]

- Whole Team (On site customer)
  The customer is an integrated member of the whole team and the team have all the necessary skills among them to perform the assigned tasks.
- Planning Game
  The Planning Game determines the scope of the next release or in detail for the next increment. This plan is revised regularly.
- Small releases
  A simple system is to be released quickly, subsequent releases will be done on short cycles

- Customer Tests (Testing)
  Customers write tests demonstrating that features are finished. Automation of the tests is important, enabling the tests to be rerun often.

- Test Driven Development (Testing)
  Short cycles of adding unit tests, make code work.

- Simple Design
  Choosing the simplest design that will pass the test.

- Design Improvement (Refactoring)
  Restructuring the code to remove complexity without changing the behavior. Focus on lowering the coupling and increasing the cohesion of the units. The Design Improvement practice requires extensive testing to verify that no unwanted effects exists.

- Pair Programming
  All code is written by two programmers at one machine

- Continuous Integration
  The system should be fully integrated at all times.

- Collective Code Ownership
  Everyone is allowed to change any code at any time.

- Coding Standards
  Making the code appears as written by one person.

- Metaphor
  A common vision of how the program should work.

- Sustainable Pace (40 hour week)
  To enable the team to work hard and effective in the long run, avoiding exhausting and unproductive extreme work load.

3.1.2.5 Change Management

No traditional change management process is needed in Agile. Continuous prioritizing of requirements and replanning are built into Agile models. Change impact analysis to identify necessary

retesting is discouraged, as the automated tests make it possible to test the entire system for each change [27].

Sommerville [39] mentions that traditional change management may not be compatible with Agile models. He especially mentions that approving every change conflicts with the Agile practice of refactoring, where any coder can improve any code without external approvement.

3.2 Safety process

The purpose of the safety process is to manage the risks throughout the project. This risk management process is performed continuously in parallel with the software development process and these two processes will interact. This content of the current section is fetched from the standard IEC-61508-1 [20].

Safety related software is often a part of a large system. The overall system safety life cycle is shown in Figure 3.11. The actual development, including the software development, is done in box 10 in Figure 3.11. Box 10 is further detailed in Figure 3.12. The term Safety process in the current thesis is used to cover the activities in the overall safety life cycle except box 10.3, 10.4, 10.5, 11, 12, 14, 15 and 16 In addition is includs the functional safety assessment.

All output from the safety process and relevant output from the development process are summarized in the safety case, for more details, see chapter 3.2.3.

3.2.1 Overall safety life cycle

The overall safety life cycle is a term from IEC 61508 [20], describing the activities necessary to develop a safety related systems. Figure 3.11 identifies three distinct groups of activities within the system development, the initial activities (box 1-5), the planning activities (box 6-8) and the final development activities (box 12 and 13). The functional safety assessment is not shown in the figure. Box 10 is further detailed in Figure 3.12.

Box 1 - 5 in Figure 3.11 are safety-related activities performed in the beginning of the project. The objective of the Concept (box 1) is to develop a sufficient understanding of the EUC and its environment to be able to perform the succeeding safety activities. The objective of the *Overall scope definition* (box 2) is to determine the boundaries of the EUC and the EUC control system.

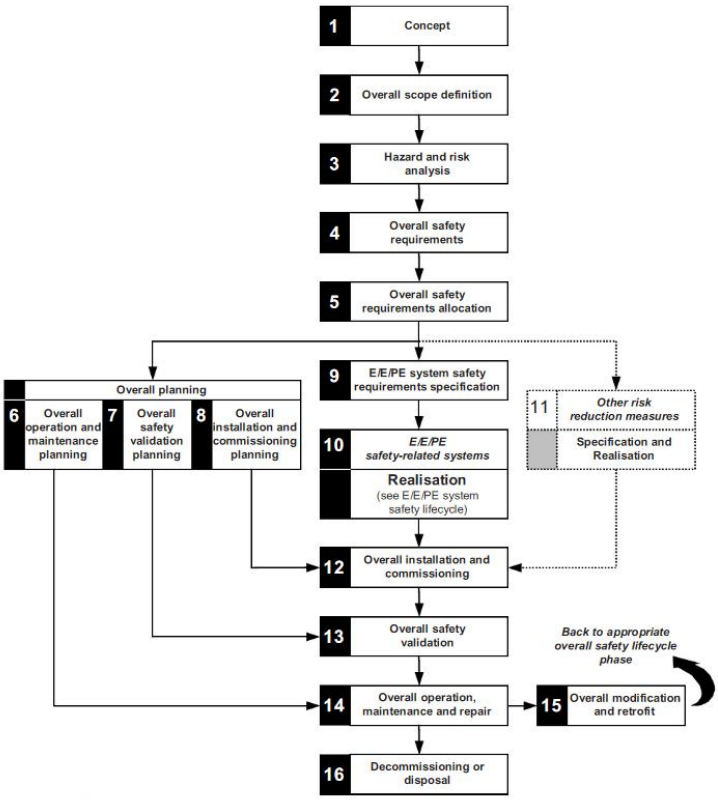


Figure 3.11: Overall Safety Lifecycle [20]

3.2.1.1 Hazard and risk analysis

The hazard and risk analysis (box 3) is the activity that determines the hazards, the event sequences that leads to these hazards, and the risks associated with these hazards. The hazard and risk analysis is the basis for determining the safety requirements (box 4, 5 and 9). Safety requirements consist of two parts:

- Safety functions requirements
- Safety integrity requirements

The safety functions requirements are the requirements to the safety functions. The safety functions requirements are a part of the systems functional requirements that are the basis for the system and the software development. The functional safety assessment is performed to assure that the safety functions requirements are met.

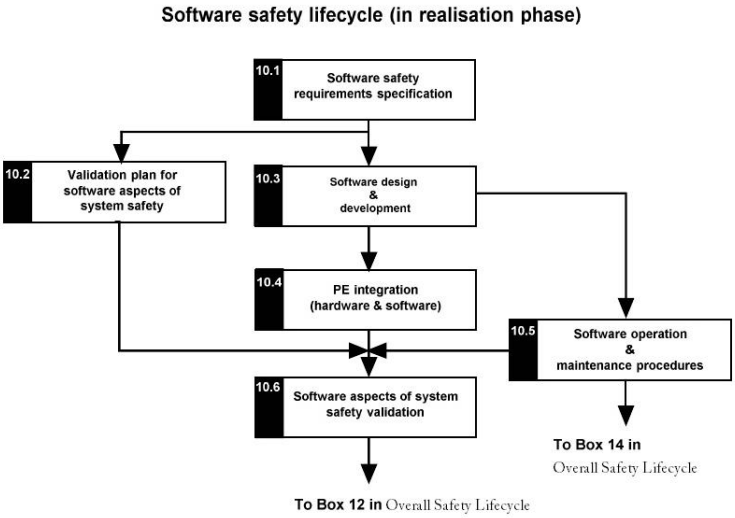


Figure 3.12: Software Safety Lifecycle (realisation phase) [20]

The safety integrity requirements are associated with each safety function and indicate the probability of failure for that function. Since software failures are always systematic failures and the probability of a given software error is hard to quantify, the Safety Integrity Level (SIL) is used. Each safety function is given a discrete level of four Safety Integrity Levels, where 1 is the lowest level of safety integrity and where 4 is the highest. The SIL is input to the software development process; the standards define a set of activities for SIL; if the appropriate activities are performed it is possible to have confidence in that the system satisfies the requirements for the given SIL. If the system consists of functions of different SILs, the system should be developed according to the highest SIL, unless there can be shown sufficient independence. The SIL also determines how to do the functional safety assessment.

The safety requirements are further decomposed into the software safety requirements, Figure 3.12, box 10.1.

3.2.1.2 Verification

The verification is not shown in Figure 3.11, but each phase in the safety process and the software development shall be subjected to verification before entering the next phase. This verification is either static or dynamic testing. Details on the verification within the software development

process is out of scope for the current thesis, but IEC-61508-1 [20] states that also the specific safety activities shall be verified. The software verification is a part of the software development process and must include verification of safety functions.

3.2.1.3 Safety validation

Validation is defined as *confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use are fulfilled* [20]. This definition differs from the definition of verification by adding the word *particular* and words *for a specific intended use*.

Safety validation is performed for the software (see Figure 3.12, box 10.6) and for the overall system (Figure 3.11, box 13). The purpose of validation is to determine whether the system performs according to the requirements for the intended use. If the result from validation differs from the expected outcome, it should be decided whether the validation can continue or if a change request have to be issued.

IEC 61508-4 defines that the requirement of independent person/department/organization also applies for persons performing safety validation.

3.2.2 Functional safety assessment

According to IEC 61508 [20] the functional safety assessment shall investigate and judge the systems functional safety and the how it complies to the standard.

Functional safety assessment may be performed for all activities in the overall safety lifecycle, and for all activities in the software lifecycle. When functional safety assessment is performed, the scope of the functional safety assessment and what techniques to use are determined when planning the functional safety assessment. The SIL is a major input when planning safety assessment. The judgment of the systems functional safety in each lifecycle phase can either be accepted, qualified accepted or rejected. If it is qualified accepted, modifications are recommended to obtain the judgment of accepted. IEC-61508-3 indicates that the assessor also should consider if the methods, languages and tools used in the development are understood and well suited.

When developing safety related software, the functional safety assessment is performed in parallel with the software development process. These processes interact; a software activity produces the input for the corresponding safety activity and the safety activity gives feedback to the software activity. IEC-61508-1 states that the functional safety assessment should be applied to all relevant phases in the lifecycle (see Figures 3.11 and 3.12), but this is not visualized in the figure to reduce

complexity in the diagrams. A functional safety assessment should be tailored according to the level of integrity required for a specific project.

3.2.3 Safety case

Kelly [24] says that a safety case *should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context.*

A safety case [41] documents the safety justification of a system. It is sometimes referred to as safety argument, safety justification or safety assessment report. A safety case is presented to the certifying authorities as evidence that all the potential hazards are identified and dealt with. A safety case also demonstrates that the development model used is appropriate and correctly used. A safety case is created early in the project and then expanded as the system is developed and assessed, and it is maintained throughout the lifecycle of the system.

IEC 61508 does not explicitly require a safety case. However, a safety case will cover the requirements in IEC 61508 of what the output from the functional safety assessment should be. A safety case is therefore not in conflict with the IEC 61508 standard.

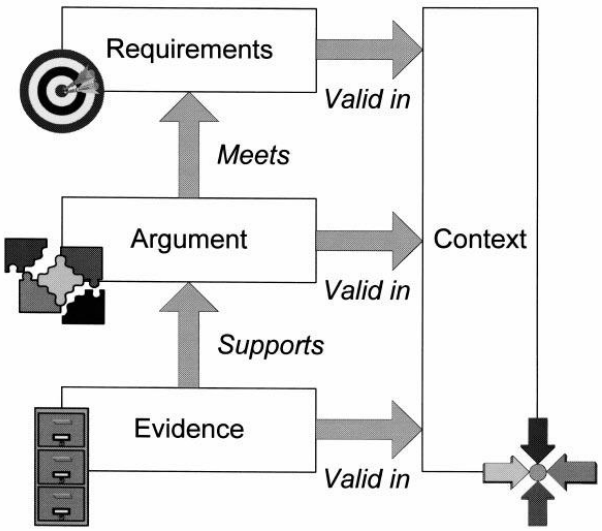


Figure 3.13: Dependencies between elements of the safety case [25]

A safety case consists of a large set of interdependent parts. A change in one part may cause consequential changes in other parts [25]. These dependencies are shown in Figure 3.13. According to this figure, the safety case consists of:

- Requirements: the safety objectives that must be addressed to assure safety
- Argument: showing how the evidence indicates compliance with the requirements
- Evidence: information from study, analysis and test of the system in question
- Context: identifying the basis (e.g. assumptions) of the argument presented

3.2.4 Change management in IEC-61508

IEC-61508-3 [20] says that a change in any phase should cause an impact analysis. The result of the impact analysis will tell which modules that are affected and which earlier phases that need to be repeated. Reverification is not an issue during the development phases as no verification is done yet.

Figure 3.11 shows the phase Overall operation, maintenance and repair where changes may occur. The process of requesting and authorizing these changes belongs to the Overall modification and retrofit phase. All requested changes shall be analyzed for impact and for new hazards, see Figure 3.14 [20]. If the modification request is approved for implementation, the development of the change starts from the appropriate phase within the overall safety life cycle.

IEC-61508-3 [20] says that the aim of impact analysis is *to determine the effect that a change or an enhancement to a software system will have to other software modules in that software system as well as to other systems*.

When the impact analysis is done, it is decided whether only the changed software, all affected software modules or the complete system need to be reverified. This decision depends on the criticality of the affected modules and the nature of the change. Safety related systems are often subjected to a safety certification prior to being put into operation. In these cases, all changes to the system cause the need for a recertification.

3.2.5 Certification

Certification involves an external, independent organization and is performed as the last element in the development process prior to deploying the system. Certification is not always mandatory, it depends on the country where the system operates and/or the industry the system is developed for. Some certifications, e.g. ISO-9000, certify the organizations. Safety certifications, e.g. IEC-61508, certify the product.

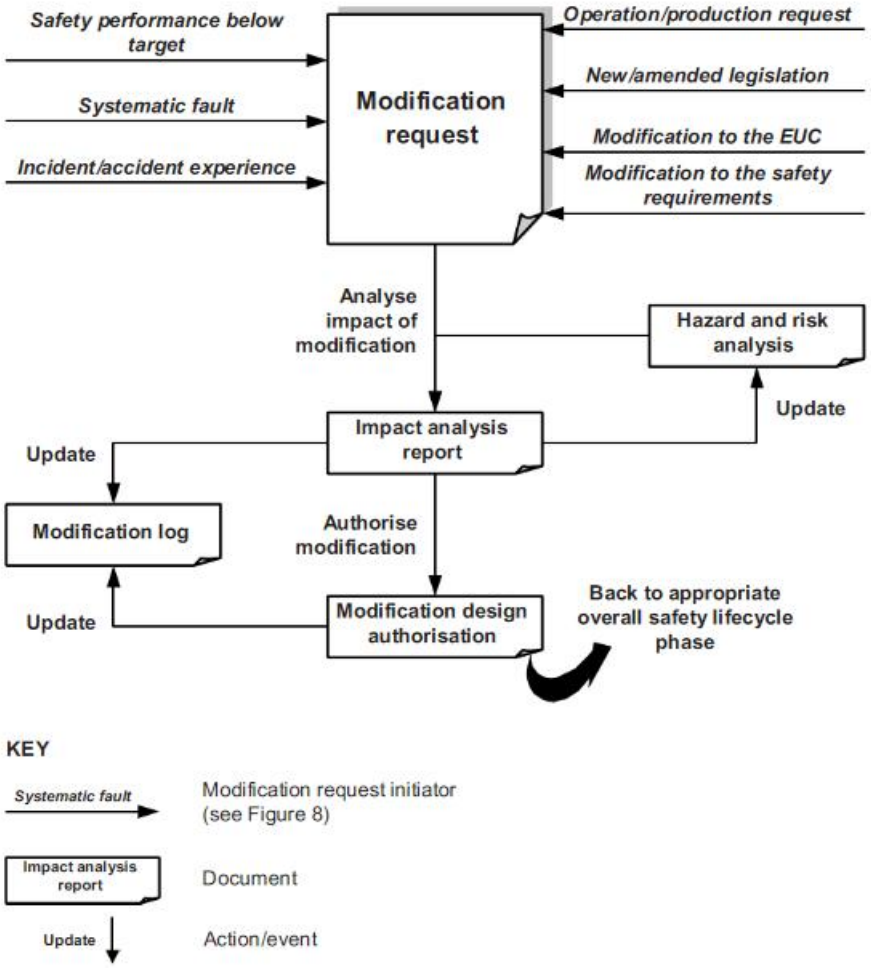


Figure 3.14: Example modification procedure model [20]

3.3 Traceability

In IEC 61508 [20], requirements traceability is required to ensure that all requirements are fulfilled and to ensure consistency between lifecycle activities. Both forward and backward traceability are required in all activities within the overall safety lifecycle and the software safety lifecycle. Forward traceability is used to check that a requirement is addressed in later lifecycle stages. Backward traceability is used to check that each implementation is justified by a requirement.

Wiegers [46] mentions the following benefits of requirements traceability:

- Certification
- Change impact analysis
- Maintenance
- Project tracking
- Reengineering
- Reuse
- Risk reduction
- Testing

Requirement traces are input to functional safety assessment and safety validation. A requirement trace is an important element of a safety argument. However, requirements traceability is only one possible representation of traceability. The following describes traceability in general.

Gotel et al. [17] give some essential terms regarding traceability. A trace artifact is a traceable unit of data (e.g. a requirement), that qualifies as either the source artifact or the target artifact. The size of the traceable unit defines the trace granularity. A trace link is a single association between two trace artifacts, from a source artifact to a target artifact. This implies a primary direction for tracing, but the trace links can be traversed in both directions. A trace can be a noun, describing the group of a source artifact, a target artifact and their associated trace link, or a chain of such groups. A trace can also be a verb, describing the act of following a trace link. Traceability is then the potential to follow a trace link from a source artifact to a target artifact.

The main goal of traceability is that the effort spent on establishing and maintaining traceability reduces the effort spent on other activities, and that the total job effort is smaller than doing the activity without traceability. To achieve this, the cost of managing trace data must be kept to a minimum. At the same time, the quality of the trace data must be sufficient to meet the project needs [21]. Mäder and Gotel [28] define the activities in a traceability lifecycle as:

1. Defining traceability
2. Creating and maintaining traceability
3. Using traceability

The traceability lifecycle is shown in Figure 3.15. A project change with traceability impact or a change in the defined traceability, initiates trace maintenance. If the current traceability does not satisfy the traceability needs, the defined traceability should be changed.

Traceability decay is defined as the gradual disintegration and break down of the traceability on a project [28]. Traceability maintenance is defined as those activities associated with updating pre-existing traces and creating new traces [28].

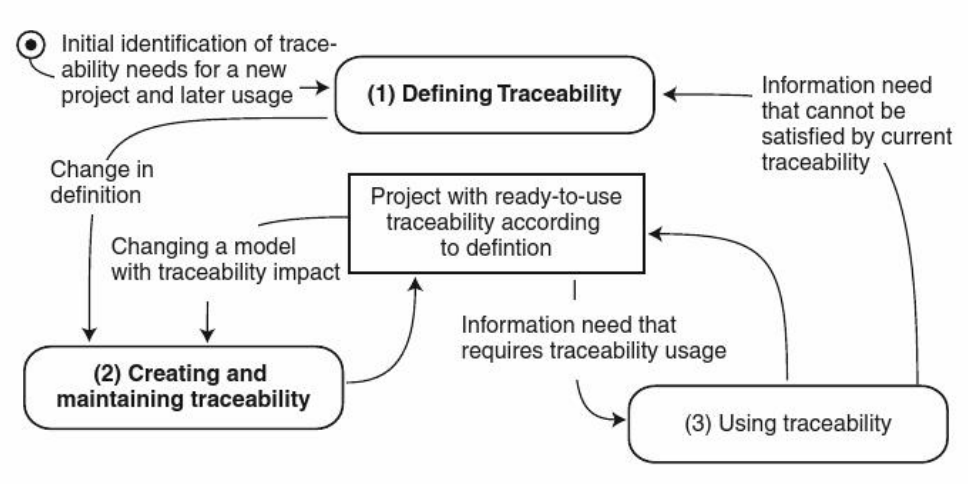


Figure 3.15: The traceability life cycle for a development project [28]

Chapter 4

Literature study

The purpose of this chapter is to give an overview of possible challenges to the safety process when developing software using Agile models. This chapter contains a selection of literature concerning the subject.

Articles concerning Agile development of safety related systems nearly solely focus on the software development process; how to incorporate the safety development activities within an Agile framework. However, these articles may give an overview of Agile models strong and weak points related to development of safety related software. Recently there are some articles that also mention aspects of the safety process in an Agile environment.

To be able to collect pieces of information for assembling an Agile safety process, literature covering separate characteristics related to developing safety related systems are investigated. The current chapter starts by an introduction on how to balance agility and discipline [9]. The next section investigates some general aspects of Agile. The focus is on aspects stated as problem areas or aspects stated as solutions to problems with traditional development [14] [43]. The third section investigates the use of Agile in large projects [15] [27]. Agile and safety is described in the fourth section, covering Agile and safety standards [42] [30], Agile and the safety process [16] [40] and Agile and traceability [12]. Finally, literature concerning building a safety case gradually is referred [25] [24].

4.1 Balancing Agility and Discipline

Boehm and Turner [9] suggest balancing agility and discipline, according to what fits the actual project best. They identify 5 dimensions that can be used when selecting a suitable development

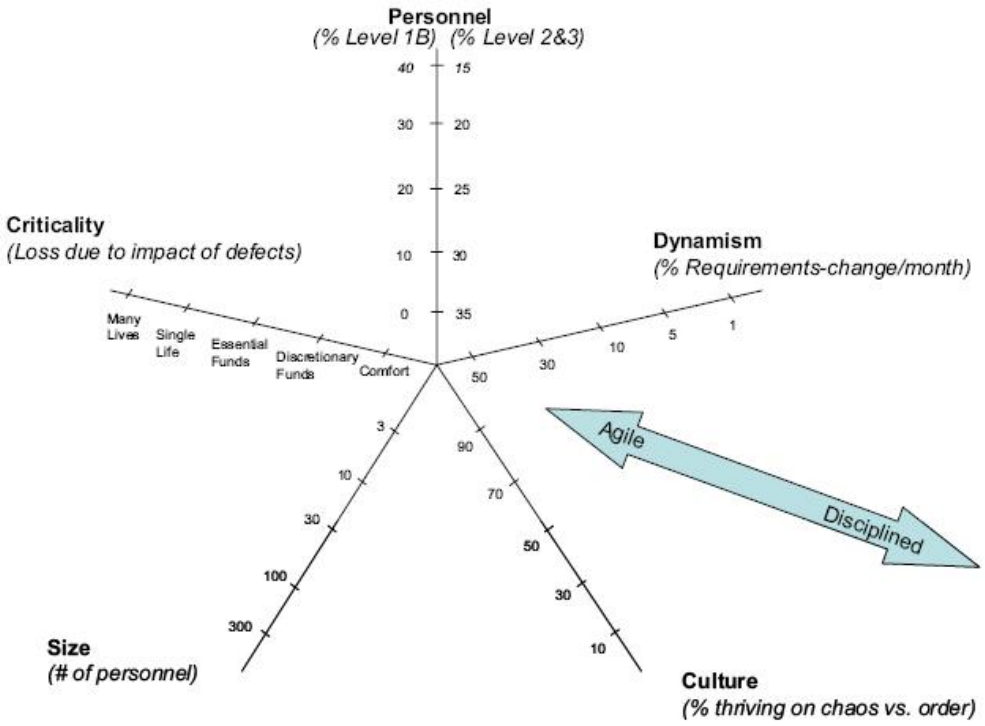


Figure 4.1: Dimensions affecting method selection [9]

method. These dimensions are size, criticality, personnel, dynamism, and culture, see Figure 4.1. The size-, criticality- and culture-dimensions are symmetrical; Agile is more likely to succeed with characteristics towards the center of the graph, whereas plan driven models will succeed with characteristics toward the periphery. The personnel- and dynamism-dimensions are asymmetrical. For the personnel-dimension, plan-driven development works for the whole scale of personnel skills, while Agile development works best towards the center. For the dynamism-dimension, Agile development works for both stable and dynamic requirements, while plan driven development is best for stable requirements.

To identify the best suited development model, the project is rated according to the dimensions in Figure 4.1. If close to center, Agile should be selected, and for projects in the periphery, plan driven should be selected. For projects with mixed profiles the architecture should be prepared to encapsulate the Agile parts, these should then be developed using Agile, the rest should use plan-driven.

Some project cannot be divided into subsystems with clear Agile or plan-driven homeground,

or some subsystems will still have areas where it does not not fit the homegrounds for the most suitable development model. Boehm and Turner also identify a set of possible project risks; risks concerning the project environments, risks when using Agile models, and risks when using Plan driven models. The risks are summarized below:

- Project environment risks
  - Technology uncertainties
  - Many diverse stakeholders to coordinate
  - Complex systems of systems
- Agile risks
  - Scalability and criticality
  - Use of simple design or YAGNI
  - Personnel turnover or churn
  - Not enough people skilled in Agile methods
- Plan driven risks
  - Rapid change
  - Need for rapid results
  - Emergent requirements
  - Not enough people skilled in plan driven methods

These risks are analyzed using example projects, including a large, safety critical system. They also suggest ways to reduce these risks. They conclude that neither Agile nor plan driven models provide a silver bullet, and that they each have home grounds where one dominates the other and that both agility and discipline is needed.

4.1.1 Points for further discussion

- The Boehm-Turner scale and the dimensions
- Encapsulation of Agile parts when mixed profile
- The home grounds of traditional and Agile models
- Use of Simple Design

4.2 Agile in general

This section focus on aspects of Agile models in general, with a focus on Agile solutions on traditional problems and some problems related to Agile models.

Redmill [33][34] reports early experiences using incremental deliveries. His intentions were to raise the awareness of potential problems regarding incremental delivery. He point to the advantage of early feedback, that he emphasizes as the most important success factor of incremental development. However, this generates a lot of changes that may cause project delay and increased cost. He stresses that the cost of validation will increase for each increment, and the validation duration for an increment may eventually exceed the duration of the increment. He promotes initial requirements, that will give better estimates and design.

These articles are from 1988 and 1992 and some of his experiences may be outdated, as the need for a separate development system. Some of Redmill's experiences may have been solved in Agile development, but may arise again for safety related system. For example that the time spent for revalidation is reduced in Agile by the use automated tests, but work spent on incremental safety assessment cannot be automated. Some of his experiences contradict Agile recommendations, he says that requirements should be developed initially. He points out is that it is incremental *delivery* is what promotes the valuable feedback, not incremental development itself. He concludes that incremental delivery increases cost and complexity of the project, compared to traditional development.

Cohn [14] mentions that cross functional teams are used to minimize the effects of handoffs. The more the whole team feels the shared responsibility, the fewer handoffs are needed within the team. When eliminating handoffs, problems related to waiting and knowledge transfer are eliminated. Cohn also says that written documentation discourage two-way communication. He also says that upfront work is anticipating, while incremental work is adapting.

Refactoring [14] is crucial to the success of Test Driven Development and XP. Refactoring is changing the structure of the code, but not the behavior. Refactoring is giving better design by fixing small problems before they become large. problems.

Turk et al. [43] has analyzed the twelve Agile principles in Figure 3.4. They identifies some underlying assumptions of these principles. Agile models are less likely to be applicable for situations where these assumptions do not hold. They argue that these assumptions would limit Agile models in respect to personnel and product. The assumptions are checked for limitation for distributed development environments, subcontracting, large teams, reusable artifacts, safety-critical software

and large, complex software.

Some assumptions that may be dangerous for safety related systems or large projects are:

- Visibility assumption - that project visibility can be obtained solely through the working code
- Continuous redesign assumption - that the continuous redesign makes the software simpler and that the system can maintain the structural and conceptual integrity when continuously refactored.
- Face-to-face assumption - that face-to-face communication is the most productive way of communication.
- Cost of change assumption - that the cost of change do not dramatically increase over time
- Documentation assumption - that developing extensive, consistent and complete documentation is unproductive

They conclude that it is important to know that these implicit assumptions exist, and that either an Agile development model is adapted to overcome these problems, or more traditional development models are used for areas where Agile is limited.

4.2.1 Points for further discussion

- Incremental delivery is the key of incremental development [33] [34]
- Increasing revalidation of each increment [33] [34]
- Initial requirements [33] [34]
- Handoffs [14]
- Cross-functional teams reduces handoffs and documentation [14]
- Refactoring [14]
- Written documentation discourages two-way communication [14]
- Visibility assumption [43]
- Continuous redesign assumption [43]
- Face-to-face assumption [43]
- Cost of change assumption [43]
- Documentation assumption [43]

4.3 Agile and large projects

When Agile models first emerged, they were intended for small project. The success of Agile models i small scale, have lead to using Agile models also in larger scale. Large scale Agile is relevant when developing for safety related systems because these systems often are large.

Elshamy and Elssamadisy [15] start out by defining a large project as a team of 50 to 100 people. They point to the reliance of heavy collaboration within an Agile team, and that dividing into sub-teams may lead to that problems with consistency and duplication is not discovered. They propose that a core team starts developing enough of the project that the primary business-areas are touched and a significant portion of the architecture is built. As the core team is a small team, Agile development can be fully utilized. When a homogeneous architecture emerges the problem is conquered and the project can be divided, and the full team may be divided into sub-teams.

Larman and Vodde [27] start their book by stating that large scale Scrum is regular Scrum. They show two frameworks for scaling Scrum; framework 1 for scaling up to ten teams, and framework 2 for further scaling, building on framework 1. Framework 1 has up to 10 teams, but only one Product Backlog and one Product Owner. Framework 2 has more than 10 teams, one Product Owner and one Product Backlog. However, different area-views (Area Backlog) of the Product Backlog are assigned to the Area Product Owner, each with their own dedicated teams, arranged as framework 1 type. Larman and Vodde also promote using feature-teams in large scale Scrum; this would reduce the need for inter-team interactions and coordination.

Larman and Vodde says that workshops reduce the waste of handoffs and increase collaborations within and between teams in multiteam development. Workshops should be used when needed, and just in time when needed. Workshops should only cover the immediate needs and goals. To establish the initial Product Backlog, an intensive Product Backlog refinement workshop could be held, containing a series of sub-workshops, as Vision workshop, What-to-analyze workshop, Detailed requirements workshop and a Parallel Effort- and Value-estimation workshop. During the detailed requirements workshop the requirement areas used for dividing the teams according to framework 2 could be identified. This initial Product Backlog refinement is considerably larger than the regular Product Backlog refinement for each increment, but besides the size it is the same activity. Design workshops should be used when needed, just prior to implementing the designed features.

Larman and Vodde also describe how to handle work that cannot be included in the Definition of Done, ie. cannot be done within the Sprint. They point out that the ideal is that all work necessary to produce a new increment is included in the Definition of Done. However, they acknowledge that this

is not always possible, and that there may exist a set of activities, called Undone work, consisting of the remaining work to be done before shipping the product. Undone Work is not unfinished work, but is a set of activities that are intended or planned to be done outside the Sprint. They describe how to handle Undone Work to minimize problems with a large delay prior to shipping. Undone Work may be handed off to a separate Undone Unit and be handled in parallel with the development. They do not consider this as a truly good solution because of the handoff to a separate team.

4.3.1 Points for further discussion

- Core team [15]
- Frameworks for Scaling Agile and Scrum [27]
- Use of workshops [27]
- Product Backlog refinement [27]
- Different views into the Product Backlog [27]
- Undone Work [27]
- Avoid handoffs [27]

4.4 Agile and safety

Research regarding Agile used for safety related systems focuses on the development process, and not so much on the safety process. This section first gives an overview of the state of the art on using Agile models for safety related systems. Then some research on Agile models for developing safety related systems are investigated. Different aspects are investigated. One is how Agile models conform to safety standards. Another is covering the safety process and Agile models. Traceability is important when developing safety related systems, and a part investigating traceability and Agile is included.

4.4.1 State of the art

This subsection tries to summarize to what extent Agile models are used for developing safety related software.

Cawley et al. [11] did a systematic literature review of Agile development for safety related embedded software. They found only a few publications showing that Agile methods have been actually used for developing safety related software. This may not mean that Agile models are not

used in this domain, it could simply be that the organizations using Agile within this domain do not publish their experiences. Their summary of the literature shows that Agile practices can mapped to the DO-178 standard.

Paige et al. [31] also point out the limited literature regarding use of Agile for safety related development and the lack of empirical data. However, they have found articles discussing the theme and indicating that Agile models may be used for safety related development. They also mentions some successful implementations of Agile within large, mission-critical systems.

This shows that there is some interest in trying Agile models for developing safety related software, and that there are literature supporting the use of Agile models in this area.

4.4.2 Agile and safety standard conformance

Thorsen [42] evaluates the suitability of the XP model for safety related development. She suggest a new version of the IEC 61508-3 software life cycle adding some of the XP practices.

The following XP practices are marked as not adopted in Thorsens proposed modified software safety lifecycle:

- The project team is traveling light (Cannot travel light when developing safety related software)
- Make frequent small releases (Integration of HW and SW makes only a few releases feasible)
- Small initial investment (the initial investment have to be considered thoroughly and have to be comprehensive so that later supplement are not too costly)
- Choose a system metaphor (too vague for a safety related system)
- No functionality is added early (To be able to have a stable system, some functionality has to be added early.)

The following XP practices are marked as partly adopted in Thorsens proposed modified software safety lifecycle:

- Move people around (Assumes the people have the right competence for the assigned task.)
- Fix XP when it breaks (only if changes are according to IEC 61508.)
- Create spike solutions to reduce risks (must be more formal.)
- Refactor whenever and wherever possible
- Integrate often (often software integration, but not software/hardware integration.)
- Leave optimization till last (Depends on which part of the code the optimization affect. Safety-critical code should only be changed for good reasons.)

Thorsen also mentions that technical safety considerations may be outside the customers expertise,

and that the customer-role must be expanded to include safety specialists.

The conclusions in [42] are that the IEC 61508 requirements cannot be met by XP, since the safety lifecycle anticipates the V-model. However, the safety lifecycle can be modified to facilitate the use of XP practices. XP has some aspects that may be beneficial for safety, but XP as it is has no focus on safety. However, most XP practices can be adopted fully or partly within the modified safety lifecycle proposed by Thorsen.

Paige et al. [30] has evaluated the suitability of XP in the area of high integrity systems engineering. They do not map this directly to any standard. The following XP practices are marked as partial applicable, indication that these practices have safety implications:

- Planning Game
- Small Releases
- Metaphor
- Refactoring
- Pair Programming
- Forty Hour Week
- On Site Customer

Paige et al. consider the practice Collective Ownership as not applicable for developing safety related systems, as high integrity software requires a high level of domain knowledge. They also mentions some areas where XP must be extended; among these are the safety process and design representation. Their conclusion is that XP and High integrity systems engineering are largely compatible, and it is worth further investigation. They suggest that a suitable safety process should be added to XP, but let this be an open question. They further conclude that there is substantial compatibility between Extreme programming and high integrity systems engineering and that it is worth while to test this in industrial case studies.

4.4.3 Safety process and Agile

Paige et al. [29] and [31] has customized XP to be suitable for developing high integrity systems. They have tailored XP to suit the actual team and project, but also including techniques from plan-driven development. Additions to XP is the risk management, safety analysis and the safety case development. XP practices are retained, except for the Metaphor and Collective Ownership practices.

The development process is viewed as a series of pipelined activities:

- Story engineering and Planning (from XP)
- TDD and integration (from XP)
- Verification & validation (from XP)
- Safety analysis
- Safety case development
- Evaluate feedback and adjust (from XP)

Prior to the development of each increment, there will be exploration spikes for a few days or a few weeks. This will give the developers understand the purpose of the system, defining an initial architecture and evolving it.

They have conducted a case study using the proposed process. The case is a system for providing pilots with altitude data during flight. They developed user stories and safety stories. This distinction was done to be able to document the output from the safety engineering. A safety story refers to the related user story.

Ge et al. [16] address two of the key challenges of using Agile when developing safety related systems:

- the up-front design in safety critical development and how to make this more Agile, but still sufficient.
- building the safety case incrementally and how to reuse safety arguments from previous releases

They discuss the differences between the Agile approach for up-front design and the safety approach. In Agile development some up-front design are done, but is normally incomplete and consists of an abstract architecture. For safety related systems the up-front design needs to be sufficient detailed to be used as an input to the hazard analysis and that the effort in this phase will increase with the increasing SIL. They also looked at what information is needed for the hazard analysis techniques, suggesting that the Fault Tree Analyses required less details than the Hazard and Operability analysis (HAZOP) and the Failure Modes and Effect Analysis (FMEA).

They mentions a modular approach for building safety arguments incrementally using the Goal Structuring Notation (GSN [25][24]). There are separate safety arguments for each module, and one argument regarding the interactions and one argument for the system. The module arguments are not linked together, as this is handled by the interaction argument. Each software increment should produce the module arguments, and the interactions arguments are then produced when the system is integrated.

They have tried out their approach on the same example used in [29]. They started out with

information of the hardware failures and effects that were analyzed using FTA on the system architecture. They found that the up-front design could be simple and still allow the safety analysis to be performed. However, the simple design must be detailed enough.

The software developed consisted of 5 modules, and the software was developed in 6 increments, the last increment did not produce any code, just the safety arguments for the interactions. They concludes that the safety arguments can be performed incrementally, but they experienced two problems:

- The duration of an increment must be longer tan the 2-4 weeks recommended by Agile models to allow the safety arguments to be produced.
- They lacked measurements to ensure that the safety arguments are sufficient strong.
  They added an argument to demonstrate that unintended interactions do not occur, but still found some arguments to be logically incomplete or logically weak.

In their safety process Paige et al. [31] outline the safety analysis and safety case development. The safety analysis includes hazard identification and hazard analysis and results in safety stories that is fed back to the Story engineering and planning activity. They observed that doing the safety analysis incrementally would invalidate previous work and that this effect was made even worse by their choice of doing the hazard identification and analysis after the development. They also experienced problems of how to argue process-related evidence in an Agile process.

Stålhane et al. [40] describes a Safe Scrum model. They have separated the development process (Safe Scrum) and the rest of the safety activities that are to be handled outside the Safe Scrum. The software safety requirements specification are done prior to the Safe Scrum development and safety assessment and safety validation are done after the Safe Scrum development. They indicate some feedback due to requirement change. However, a some safety validation is added to each increment of the Safe Scrum, included in the Sprint Review meeting. The Sprint review will demonstrate and validate both the functional requirements and the safety requirements. This is no formal assessment, the assessors are advised to, but not required to participate. They expect the final assessment and validation to be reduced due to this incremental validation.

The Product Backlog is divided into two [40], the Functional Product Backlog and a Safety Product Backlog. These backlogs are linked to show dependencies between the functional requirements and the safety requirements. To cope with extra work added to increments, Stålhane et al. [40] identifies documentation and maintaining traceability as extra activities to be included in the Sprint. To cope with this extra load, they suggest using tools supporting traceability and separate documentation teams for all documents not related to the code development.

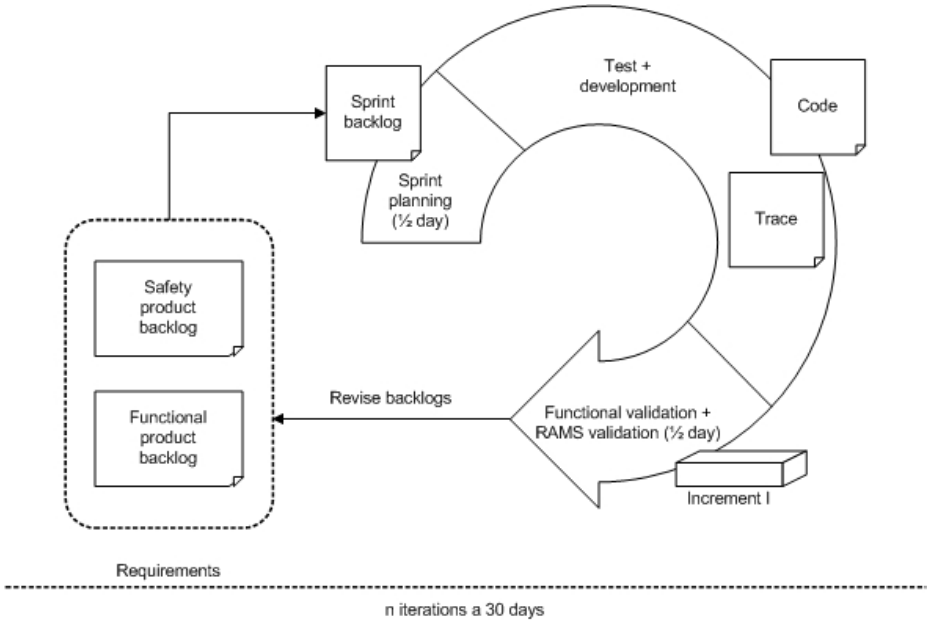


Figure 4.2: Stålhane et al. [40] The Safe Scrum model

The Safe Scrum (Figure 4.2) uses two Product Backlogs, one for the functional requirements and one for the safety requirements. The functional requirements will refer to their related safety requirements. When the item in the functional backlog is related to an item in the safety backlog, the Safe Scrum is used for developing. Requirements unrelated to safety is developed using normal Scrum.

Safe Scrum adds the maintaining of trace information, tracing backlog items to and from source code, to the development. At the end of each Sprint, a safety validation is included in the functional validation. They indicates that the assessor may participate in this validation, and also in the reflection-activities after each Sprint, to keep the developers focused at safety issues.

Stålhane et al. also proposes that all documentation that are not required for coding, is moved out of Scrum; a separate documentation team, working closely with the development team, is suggested.

4.4.4 Traceability and agile

Maintaining traceability in traditional development with relatively few changes is a challenge. Maintaining traceability incrementally is an even greater challenge. Mäder and Gotel [28] mention that although there is much research in the area of establishing traceability initially, there is less

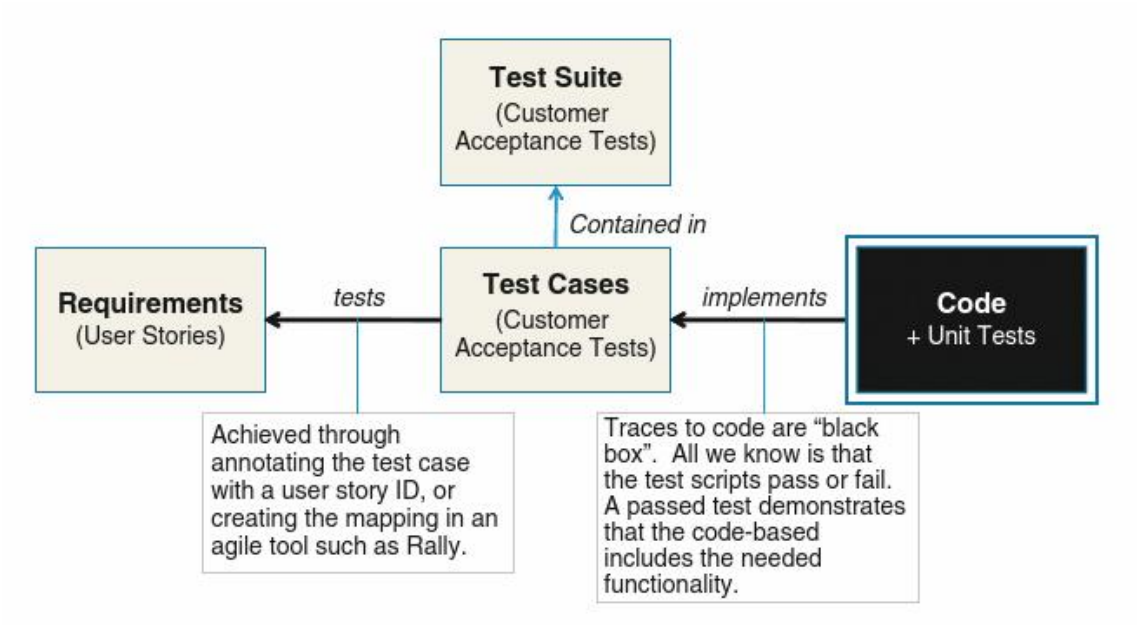


Figure 4.3: A traceability information model for a basic agile project [12]

focus on maintaining traceability. They refer to research concluding that traceability maintenance currently is both expensive and error prone.

Maintaining traceability requires adding new links, updating existing links and removing existing links that have become obsolete. Mäder and Gotel mention that the goal is to have complete and correct traceability, but this cannot be verified automatically. They suggest defining a traceability information model (TIM) and keeping the traceability compliant to the TIM.

Cleland-Huang [12] says that in many cases traces can be established as a byproduct of the normal activities within Agile development. The basic traceability information model for Agile development is shown in Figure 4.3. This model shows that a user story and its related test are satisfied by the code. However, it does not support impact analysis, i.e. identify the impacted sections of the code.

According to Cleland-Huang there are other traceability tasks that are of interest. To identify the source of a requirement, the trace from the stakeholders to the requirements they are responsible for should be established. This can easily be achieved by tagging each requirement with the contributing stakeholders' name. Tracking which version of the software a given requirement is implemented in can easily be done through the version control tool. All committed changes are tagged with the ID of the implemented requirements. This provides traceability into the code, the granularity is decided

by the check-in unit.

Tool support for traceability is needed to manage a large amount of trace information. Gotel and Mäder [18] mention that no trace tool solve the traceability problem; such tools are often expensive, but do not fulfill the expectations.

There are two main strategies [17] for updating trace information, continuous or on-demand. When updating traces continuous, a change in a trace artifact is followed by changes in the impacted trace links. When updating traces on-demand, the whole or a large portion of the traces is updated prior to the traceability use.

Ingram and Riddle [21] suggest strategies for balancing the cost of traceability maintenance with the quality of the traceability information. They defines the traceability quality as a function of three factors, the trace granularity, the recall and precision of the links, and the level of coverage. Better quality traceability increases the cost. They therefore suggest that the traceability quality can vary over the system. By using high quality traceability where it is needed, and lower quality traceability elsewhere, the cost can be reduced. Ingram and Riddle also mention that the frequency of trace updates also can be varied for different parts of the system.

Larman and Vodde [27] suggest defining requirements as executable tests. In this case the requirement specification and the acceptance test specification are the same document and no explicit trace is needed. They also suggest tagging every acceptance test and every unit test with the requirement ID. This will result in trace from requirement to code.

4.4.5 Points for further discussion

- Spike solutions should be more formal [42]
- Upfront vs. Simple design [42]
- Structuring the safety case to promote incremental safety case building [16]
- Incremental safety case increases the increment size [16]
- Experience of using a simple up-front design as a basis for the hazard analysis [16]
- Safety requirements in separate backlog [40]
- Dependencies between functional requirements and safety requirements [40]
- Informal incremental safety validation [40]
- One final safety assessment for the complete system [40]
- Separate documentation team [40]
- Agile trace as a byproduct of normal Agile activities [12]
- On-demand / continuous trace update [17]

- Vary traceability quality and update frequency [21]
- Requirements as executable tests [26]

4.5 Safety case

The safety case is the main input to the certification authorities. This section investigates literature concerning updating safety case and maintaining a safety case over time. Maintaining a safety case in an Agile environment is expected to be more challenging than in a traditional environment.

Kelly and McDermid [25] recognize the fact that a safety case need to be updated through out its life time. They identifies the following problems in safety case maintenance:

- Difficulty in recognising (importance of) challenges to the safety case
- Difficulty in identifying the indirect impact of change
- Lack of assurance/justification of the change process
- Insufficient information recorded to support the change process

They introduce a notation, the Goal Structuring Notation (GSN) to visualize the interdependence of the individual safety case parts. They show how the dependencies in the GSN can be followed to identify the impact of a change. The GSN is further developed into a standard and are supported by tools.[1]. The GSN consists of different symbols for goals (claims), solutions (evidence), strategy, context, justifications and assumptions. An undeveloped-symbol is also available. These symbols are connected by arrows, one meaning *supported by*, the other meaning *in context of*.

Kelly [24] investigates how to perform an incremental safety case development (although within the V-model environment of the software development). At first he refers the problems experienced with developing the safety case as a closing activity:

- Large amounts of redesign resulting from a belated realisation that a satisfactory safety argument cannot be constructed
- Less robust safety arguments being presented in the final safety case.
- Lost safety rationale.

Kelly shows how to use the GSN to build the safety case successively, by starting out identifying goals, subgoals, strategy and context. The solutions, justifications and assumptions are added incrementally.

[1] http://www.goalstructuringnotation.info/

4.5.1 Points for further discussion

- Difficulty identifying indirect impact of change [25]
- GSN [25]
- Consequences of late developed safety argument [24]

Chapter 5

Discussion

Discussing how the safety process and Agile development affect each other is a large topic. The discussion is divided into 3 separate chapters, the first (Chapter 6) covering incremental development and the safety process, the second (Chapter 7) covering the iterative development and the safety process, and the last (Chapter 8) covering the rest of the Agile practices and the safety process. A summary of the discussions, suggesting a more Agile safety process within a Scrum frame, is given in Chapter 9.

Before exploring these topics, the current chapter tries to define a safety related system with respect to the dimensions given by Boehm and Turner [9] and discusses some aspects of the safety process in a sequential environment. Then the aspect of handoffs are shortly discussed for traditional development and Agile development. Handoff is a weakness of traditional models that is minimized in Agile models. Finally, the safety process activities are grouped to simplify later discussions.

The sources for this discussion are the literature mentioned in Chapter 4. In addition, IEC 61508 [20] is a central source, as well as literature covering Agile and safety in general (see Chapter 3).

5.1 Dimensions of a safety related system

Figure 5.1 tries to sketch a possible area (the grey area on the figure) where a safety related system can be placed in Boehm and Turner's polar chart [9]. This polar chart has five dimensions, where two of the dimensions are related to the staff available (Personnel and Culture). The other three dimensions (Size, Criticality and Dynamism) are properties of the project. The Dynamism and Personnel dimensions are asymmetrical. Plan driven models require low dynamism, whereas Agile models handle projects with both high and low dynamism. Plan-driven models handle a wider range

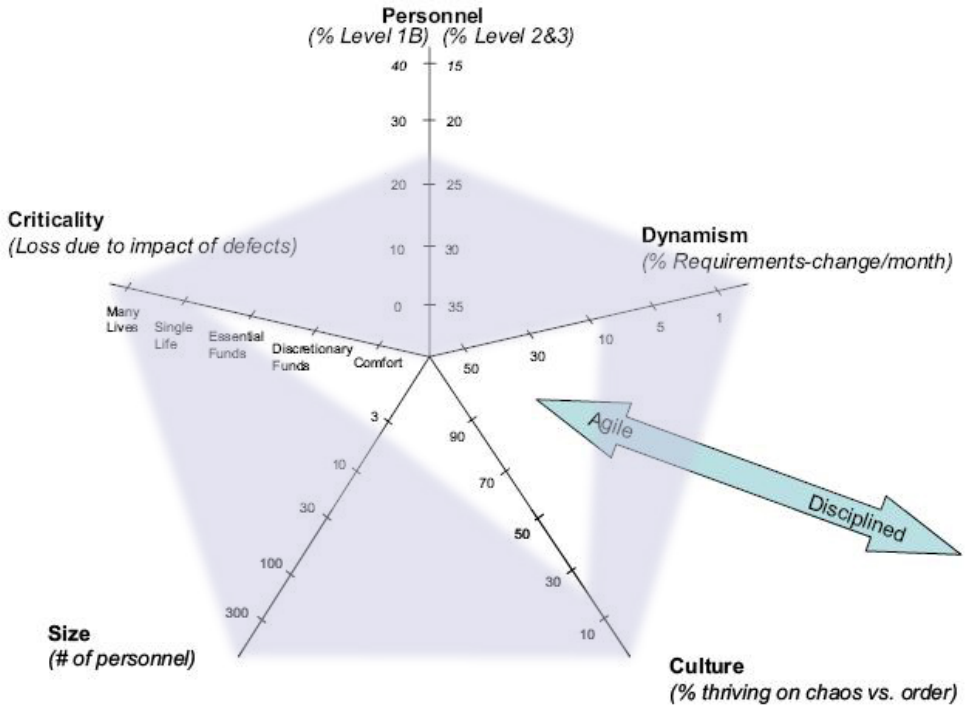


Figure 5.1: Illustrating a possible range for safety related systems on the Boehm/Turner scale

of personnel skills than Agile models, which work best with a high level of skill.

In the following a short introduction to the five dimensions for safety related systems are given:

- Size
  A safety related system has no standard size. Software for a control system is typically large, whereas software for a protection system is often small. In Figure 5.1, the size of the system is indicated form the largest systems to very small systems. However, a safety related system needs extra effort and additional staff to handle safety requirements and very small project sizes are therefore rare.

- Criticality
  A safety related system covers *one to many lives* on the criticality dimension. A high integrity system may also cover *essential funds*.

- Personnel
  This is an organizational dimension, and not a dimension of a safety related system. Developing a safety related system require more skills than developing an average system.

- Dynamism
  Safety related systems tend to have more stable requirements than business systems. A control system is highly interacted with the physical process that is controlled, and a large portion of the requirements arises from this. Agile models handle systems with both high and low dynamism, while plan driven models perform best for systems with low dynamism.

- Culture
  This is an organizational dimension, and not a dimension of a safety related system. Developing a safety related system expects and requires less chaos than an average system.

Dimensions describing a safety related system can then be summarized as high criticality, often low dynamism, and to some degree, large size. However, small projects and projects with some requirement change cannot be ruled out. Some safety related systems fit into the home grounds of traditional development, whereas others represent a mixed profile, containing characteristics that partly fit Agile development models and partly fit traditional development models. The discussions in the current thesis assumes larger projects with fairly stable requirements, unless otherwise stated. This are systems that Boehm and Turner [9] consider as best suited for plan driven models. However, these types of systems are well suited to identify implications between Agile models and the safety process.

5.2 The safety process in a traditional environment

This section recapitulates the scheduling of the activities in the traditional safety process. It also summarizes some of the strengths and weaknesses of the traditional safety process. This is not meant as a complete list, but a summary which can be compared to the strengths and weaknesses of the Agile safety process.

5.2.1 The scheduling of activities in the traditional safety process

The activities in the safety process are scheduled as initial, parallel or closing activities in the traditional safety process. The box numbers refer to the boxes in Figures 3.11 and 3.12 [20]:

Initial safety activities are all activities leading up to establishing SIL and SW safety requirements: Concept (box 1), Overall scope definition (box 2), Hazard and risk analysis (box 3), overall safety requirements (box 4), overall safety requirement allocation (box 5), system safety requirement specification (box 9), software safety requirement specification (box 10.1). The box numbers refer to Figures 3.11 and 3.12. These activities are largely initial in a sequential environment, but are supposed to be repeated when changes occur. Another activity that has a large initial part is planning of functional safety assessment. This is done prior to the actual assessment, but after the SIL is established.

Parallel safety activities can either interact with the development process or be done independent from the development process. Functional safety assessment is an ongoing activity, assessing the software in different phases. Establishing the safety case and arguing safety should be done in connection with functional safety assessment. Safety assessment interacts with the development process. The planning of safety validation is done in parallel to the development with little interactions with the development process.

Closing safety activities are the activities performed largely after the development are finished. The final safety activity is safety validation. These are the activities for software safety validation (box 10.6) and overall safety validation (box 13). Functional safety assessment can be performed as a closing activity, i.e. assessment of the last phase.

The safety process activity scheduling is summarized in Figure 5.2. This figure only shows the scheduling of the activities, not the weight of the activities. The numbers in Figure 5.2 refers to the boxes in Figures 3.11 and 3.12. Figure 5.2 also shows that planning of the safety activities are separated from performing the corresponding safety activity.

5.2.2 Strength of the traditional safety process

The main strength of the traditional safety process is that it is standardized and proven over many years. Initially the system is analyzed to identify all possible hazards prior to development. For each phase the whole system is analyzed (to the extent planned by the safety manager). At the end of the development, the complete system is analyzed at different levels of details and different views into the system. Reassessments are limited, and if some reassessments were incomplete in early phases, the assessment in the next phases covers the whole system (to the planned extent). The project is

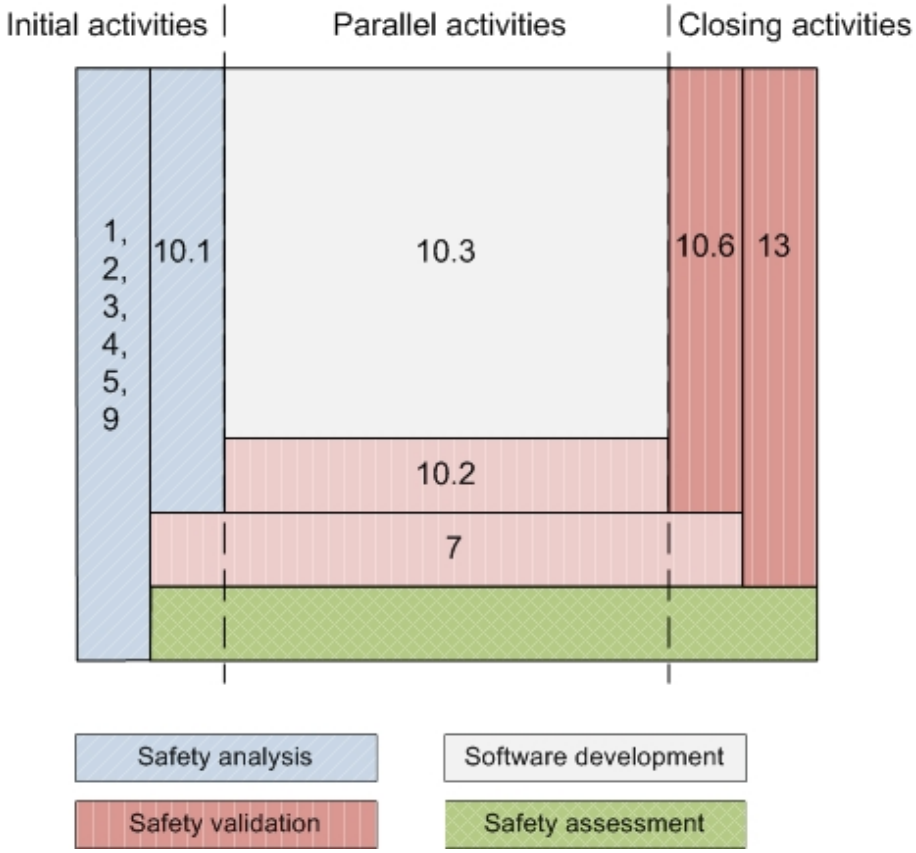


Figure 5.2: The safety process according to IEC 61508

closed by performing a safety validation on the complete system to assure that all safety issues are covered.

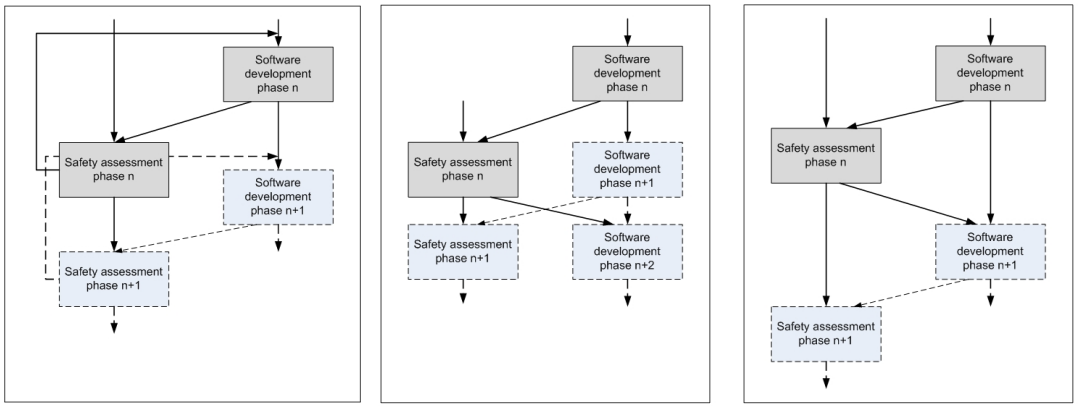
5.2.3 Weaknesses of the traditional safety process

The traditional safety process is based on traditional development models and will therefore inherit the weaknesses of these models. Among these are the limitations of handling changes, the dependencies created by handoffs and late discovery of integration problems. The safety process will actually create more handoffs, mainly handoffs to safety assessment activities and handoffs back to the development process.

IEC 61508 [20] states that functional safety assessment shall cover all phases within the overall safety lifecycle and within the software safety lifecycle. However, it states that functional safety

assessment can be performed for each phase or for a number of phases. The assessment does not need to be performed after each phase, and an assessment can then cover output from several phases. Functional safety assessment can then be performed once, late in the project. Safety discrepancies are then discovered late. Correcting or implementing safety functions when the software is close to finished, gives fewer alternatives for solution than it would have done earlier in the project. Late corrections are then more expensive and less safe than early corrections.

Performing functional safety assessment for each phase gives the possibility of discovering safety issues earlier. The ideal practice is shown in Figure 5.3(a). Here the development of the next phase is not started until all remarks from the assessment is handled in the current phase or earlier. However, this practice introduces a delay before the next phase can start for the affected functions or modules. It also adds an uncertainty of when the next phase can start and the staff cannot be used effectively.



(a) Back to phase n (b) To phase n+2 (c) To phase n+1

Figure 5.3: Safety assessment feedback in a traditional software development

Figure 5.3(b) shows that the development in the next phase continues in parallel with the safety assessment of the previous phase. This allows the development to run without delays, but the safety feedback is delayed. Even though the actual changes due to the feedback are handled properly by repeating the affected phases, the safety changes is more difficult to incorporate in the existing design or code, than it would have been if the safety issues was included from the beginning.

In 5.3(c) the safety assessment is done between the development phases. This will assure that the safety feedback is given as soon as it is detected, minimizing the negative effect of incorporation in existing design or code. However, the total duration of the project is increased and thus more

expensive. The delay between the development phases and between the assessment phases can result in the staff having periods of inactivity.

The safety validation is the last activity. This assures that the complete system is validated, but any discrepancies detected by the safety validation is then discovered late and are thus expensive to correct. At this late stage of the development, the best solutions concerning safety may not be available.

Traditional development models are best suited for stable environments, and have deficiencies in how they handle changes. [39] [9]. Since the safety process, especially safety assessment, will generate changes to the system, this underlying weakness of the development model will be amplified when developing a safety related system.

IEC 61508 describes two approaches for the safety process, one in the development phase (shown in Figures 3.11and 3.12) and one in the operational phase (shown in Figure 3.14). Hence the safety process of IEC 61508 handles changes more rigid in the operational phase requiring the change to be authorized (after impact analysis and hazard analysis) and logged.

5.3 Handoffs

From one phase to the next in a traditional development model there is a handoff, often from one team to another. This is one of the weaknesses of traditional development, as mentioned above. These handoffs creates dependencies, the next team cannot start until the first team is finished. Handoffs also lead to misunderstandings and delays. Cohn [14] refers to a definition of handoff as a separation of knowledge, responsibility, action, and feedback. He also mentions that the cross functional teams of Agile are one response to minimize handoffs. Others will be face-to-face communication and use of workshops.

When adding elements from traditional models into Agile it should not mean uncritically adding traditional limitations and weaknesses. When adding initial work, closing work, more teams, or documentation, handoffs are added. It is then important to see if handoffs are avoidable or if the effect can be reduced.

The pipelining suggested by Paige et al. [29] for developing safety related systems and functional safety assessment incrementally seems to reintroduce handoffs. The waterfall-limitation with handoffs from one phase to the next has now an additional restriction that all phases should fit in the same timebox-length. They suggest that one timebox contains development and safety activities from different increments. They have separated the planning of an increment from the software

development, that is again separated from the evaluation and adjustments, creating handoffs not present in Agile.

The safety process itself reintroduces some handoffs, possibly from the safety analysis to the software development and unavoidable from the software development to the safety assessment and then to the safety validation due to the requirements of independence for the staff performing these activities. An important issue is then to minimize the effect of these handoffs. One Agile way is to use face-to-face communication, possibly in a workshop setting, and not relying entirely on documents for transferring information. As the workload of development of a function differs from the work load of the assessing or validating the same function, these activities should not be restricted within the same timebox length. An assessment increment could then be of different length than a development increment, or the increments could have the same length, but different content.

5.4 Grouping the activities of the safety process

In this section the elements of the safety process are grouped into three groups to simplify the discussions. The elements in one group have similar properties, allowing them to be discussed together.

The purpose of this chapter is to establish some terms for the succeeding discussions. Using one term covering several elements with common characterizations simplifies the discussions. However, the terms described here may have other meanings in other contexts.

In an Agile environment, the safety activities may be scheduled differently than in an traditional safety process (as described in Section 5.2.1). This is among the issues discussed in the following chapters

Safety analysis The term safety analysis is used in the discussion in the following chapters to describe the activities that lead up to identifying the SIL and the safety requirements. These activities are basically initial in the traditional safety process. This term then includes all activities described as initial safety activities above. New or changed hazards cannot be identified or analyzed without checking that concept and scope are still valid, or updating concept and scope. If new or changed hazards are detected, this leads to new or changed safety requirements. These activities are closely related, and thus discussed as a whole. Safety analysis of a function is done prior to developing the function.

Safety assessment The term safety assessment in the discussion includes the activities Planning of functional safety assessment and Functional safety assessment from IEC 61508. In addition the preparation of the safety case is included in this term as the functional safety assessment is a major input to the safety case and the safety case presents the judgments from the functional safety assessment. The planning and performing the assessment are grouped as one activity since Agile models do not separate planning from the actual work. Safety assessment of a function is done after developing the function.

Safety validation The term safety validation in the discussion includes the activities Planning of safety validation, Software Safety validation and Overall safety validation from IEC 61508. The Agile models do not separate planning from the actual activity, so these activities are grouped together. Safety validation of a function is done after developing the function.

Due to the requirement of independence, safety assessment and safety validation are expected to be performed by separate teams. In the following chapters these teams are called the safety assessment team and the safety validation team. Safety analysis has no requirements of independence, and are then not required to be performed by a separate team. However, the term safety analysis team is sometimes used in the following chapters to describe a set of people having knowledge and responsibility regarding safety analysis and safety requirements.

The remaining activities from Figure 3.11, box 6, 8, 12, 14, 15 and 16 are not discussed in the current thesis. These are activities that are expected to be least influenced by Agile development. However, overall installation and commissioning would be affected if incremental delivery is used.

The Overall modification and retrofit phase in IEC 61508 [20] represents a form of incremental development for safety related systems (see Figure 3.14). However, this phase is mostly concerned with error corrections and minor additions to a stable system and not to the frequent additions in incremental development. The recommendations in IEC 61508 for this phase may therefore not be directly applicable for incremental development. The recommendations may however be used as the basis for the discussion of the safety process in an incremental environment as it is the only example of an incremental safety process within IEC 61508.

5.4.1 Summary

When discussing the safety process in an Agile environment, it is important to know when the safety activities are performed in a traditional process.

- Traditional safety analysis is largely performed initially, but is also performed when a change occur. Safety implications are detected early and consequently SIL is established early. However, in traditional safety process, there are expected few changes.
- Traditional safety assessment is planned as an initial activity, and performed mainly as an ongoing activity for each development phase as planned. However, worst case is that it is performed only as a closing activity.
- Traditional safety validation is planned in parallel to the development and performed as a closing activity.

The traditional safety process inherits and amplifies the weaknesses of traditional software development models. Among these are the late discovery of errors and safety discrepancies. This is expensive and may reduce safety. If performing functional safety assessment after each phase, valuable feedback is offered to the development process.

Handoffs result in dependencies, delays and information loss. When adding safety activities to Agile development, handoffs should only be added if it is not possible to avoid.
The discussions in the next chapters use the following terms:

- Safety analysis - covering all activities leading up to safety requirement establishment or change.
- Safety assessment - covering planning and performing functional safety assessment and preparing the safety case.
- Safety validation - covering planning and performing the safety validation.

Chapter 6

Discussion: The safety process within an incremental environment

In this chapter, the incremental development and incremental delivery aspects of Agile models are investigated. In incremental development the system is divided into small pieces of functionality that each can be implemented within a predefined timebox. The developed functions are integrated as soon as they are finished. This allows a working system with reduced functionality early, and the system is completed gradually.

Incremental development is one of the most fundamental changes from traditional development and has a radical impact on the safety process. In traditional development the work in one phase is directly dependent on the work done in the previous phase. This is not the case in incremental development. However, there is an indirect dependency, since the new increment must be integrated with the already developed work.

This chapter discusses the effects incremental development has on the safety process and the effects the safety process has on incremental development. The discussion groups the safety activities into three groups: safety analysis, safety assessment and safety validation. Each of these groups is discussed separately. Traceability is an important factor in developing safety related systems, and how to handle traceability maintenance to support incremental safety activities are discussed. In addition, the limitations of incremental delivery on safety related systems, and its impact on closing activities in the safety process, are discussed. Finally, the Agile practice of short increments is discussed.

6.1 Safety analysis

The safety analysis section is divided into five parts. The first part investigates the aspects of initial safety analysis. The second part mentions some methods that may make initial safety analysis more Agile. The third part investigates the aspects of safety analysis as an incremental activity, and the fourth part discusses the combination of initial and incremental safety assessment. Finally, the fifth part summarizes the safety analysis discussion.

The initial activities in the IEC 61508 safety process are: Concept, Overall scope definition, Hazard and risk analysis, overall safety requirements, overall safety requirement allocation, system safety requirement specification, and software safety requirement specification. The output from these initial phases is the safety functions requirements and their associated safety integrity requirements. They are input for determining the safety integrity level (SIL) for the system. The systems SIL will then give recommendations and constrains regarding the software development process and the safety process [20].

In Agile models, too much initial work is discouraged. Cohn [14] states that Scrum projects does not have initial work, neither analysis nor design. He says that up-front work is anticipating user needs, while Scrum is adapting to user needs and thus all analysis and design are done within the increment. Other advices are more open to initial work; up-front requirement and design works for stable environments, but causes expensive rework in dynamic environments [9]. Larman and Vodde [27] introduce some initial work to cope with the extra size and complexity for large projects. They also describe how this initial work can be done in an Agile context.

6.1.1 Safety analysis initially

One important argument for initial safety analysis is that SIL must be defined as early as possible. The SIL of the system decides how rigorously the system must be developed and what techniques to be used within the development. This implies that initial work must be done to a degree that the safety requirements can be detailed enough for deciding the SIL with high degree of confidence. If the SIL initially is defined too high, the development of the system will be more expensive than necessary. If the initial SIL is detected to be too low later in the development, the developed system must be discarded and redeveloped according to the rules of the increased SIL.

Paige et al. [29] say that the incremental nature of Agile development does not allow for complete initial safety analysis and that hazard management should therefore be incremental in order to be successfully applied to XP projects. However, it is not the incremental nature of Agile, but the

frequent requirement changes that do not allow for the complete initial safety analysis, as requirement changes and new requirements will invalidate the initial safety analysis, and possibly the SIL. Ge et al. [16] say that requirement changes are less frequent when developing safety related systems, but the consequence of a requirement change is substantial, and may invalidate the complete safety argumentation.

Redmill [33] encourages initial requirements, contradicting Agile recommendations. He implies that initial requirements give better estimates and better design. He emphasizes that the functionality, reliability, maintainability, and the expandability of the system will be better when the complete set of requirements are present early in the project.

There are also other arguments supporting an early safety analysis or giving benefits of doing an amount of initial work. The initial safety analysis is done at system level. Many safety related system consist of hardware development in addition to the software development. Traditionally, hardware- and software development are separated early, due to different approaches of development. To be able to separate the hardware and software requirements, requirement analysis, including safety analysis, must be done initially. He especially warns that timing in real-time system can suffer, a major problem for safety related systems.

By dividing the software into sufficient independent subsystems, different SILs can be allocated to the subsystems [20]. Developing a system with a high SIL is expensive and reducing the extent of such development is important. Smaller subsystems will also reduce the scope of later safety activities such as the safety reassessment. The partitioning also allows different development models to be used for the subsystems, according to what fits the subsystem best [9]. Software partitioning with sufficient independent subsystems cannot be done without initial requirement- and safety analysis.

Adding a larger initial work than anticipated in Agile models makes the model less Agile. The benefits of using Agile models are then reduced. However, the requirements in safety related systems are often relatively stable. The requirements mainly arise from physical processes and legislation, and less from the customers' demands and wishes. Low frequency of requirement change allows a larger part of the work to be done initially, even in an Agile environment. The rework due to changing requirement should be at an acceptable level. However, the Agile property of handling changing requirements is not fully utilized when developing safety related systems.

The main purpose of initial safety analysis is to define a credible SIL. To do this, a credible architecture must also have been defined. However, the initial SIL may be invalidated by later changes, which can lead to invalidating all the developed code. A credible initial architecture and SIL can only reduce the possibility of later invalidation, not eliminate it. In systems with frequent

requirement change it may be difficult to determine a credible SIL initially.

6.1.2 Improving agility of initial safety analysis

Initial work, like initial safety analysis, are not considered Agile. However, when scaling Agile there is an understanding that some initial work is needed to cope with the extra size and complexity [27]. This section describes two approaches for doing initial work more Agile and their possible impacts on the safety process.

Larman and Vodde [27] suggest using workshops involving both the customers and the developers to produce the initial Product Backlog and to create a common understanding. Involving the developers in early work minimizes the effects of handoffs. Larman and Vodde also propose that these requirement workshops is repeated frequently to refine the Product Backlog during incremental development, i.e. the initial work are not signed off as finished work, it is expected to be changed and dealt with throughout the development.

In safety related systems, the requirement workshops need to include the safety analyzers, the domain experts, the Product Owner and developers. The customer has less impact on the requirements of a safety related system, and should then be included in these workshops only if he is a domain expert or a safety expert. The requirement workshops should also include safety analysis. See the discussion in Chapter 7.2.1.

Elshamy and Elssamadisy [15] offers an alternative to workshops for establishing initial work and a common understanding before scaling up the development to all teams. They suggest to use a core team developing the first parts of the system, covering the main business areas. The purpose of this is to get an early overview of the architecture and still be Agile. When a stable architecture is in place, the project can be scaled up. The system is divided and distributed to subteams and will continue to grow. If this approach is used to establish the architecture that should be a basis for the safety analysis, there are some considerations. The core-team approach can only be used for safety related systems prior to safety analysis if the system developed by the core team is thrown away, and only the architecture are kept. This is because the development is done prior to establishing SIL and the development may not have been done according to the rules for the defined SIL. More important, this development is done prior to hazard identification and establishment of the safety requirements. This means that safety considerations are not included in the initial architecture. This is not necessarily solved by just including incremental safety analysts for the core team. Critical safety findings may invalidate all work done by the core team. However, by performing incremental safety analysis for the core team and throw away the developed code, the defined architecture may be

sufficient to serve as a basis for safety analysis. A combination of Workshops and spikes solutions, to clarify difficult areas, may be more efficient.

6.1.3 Safety analysis incrementally

There are strong indications that safety analysis should be done initial, but does this exclude incremental safety analysis? Stable requirements do not mean that there are no changes in the requirement. Actually, the safety process itself generates new requirements[31]. When new requirements occur, old requirements may be invalidated. As the development progress, more information is gained that could help detect hazards and requirements that did not be discovered initially.

The modification procedure from IEC 61508, shown in Figure 3.14, indicates that for each change, the impact should be analyzed and, if necessary, the hazard analysis should be performed. IEC 61508 also indicates that new hazards may be detected during this analysis. This is the ongoing part of the traditional safety analysis, and indicates that safety analysis should have an incremental component to maintain the level of safety obtained by traditional safety process.

Incremental safety analysis means that all the aspects of safety analysis shall be repeated, check that the concept and scope are still valid, check if there are new hazards and establish new safety requirements.

A central question is if incremental safety analysis generates extra rework compared to a more traditional safety analysis. If no changes, incremental safety analysis simply will state that the prior safety analysis is still valid for the actual requirements. If a requirement is reanalyzed due to Backlog refining, this means that less work was needed for the initial analyze of the requirement. If a requirement is changed or new requirements are defined, a complete analyze is required, as it is when using a traditional safety process. None of these can be considered extensively rework compared to traditional safety analysis.

New safety requirements can invalidate earlier work [29], but that is not specific to neither incremental development nor safety requirements. This is a problem in a dynamic environment with many requirement changes. However, doing safety analysis purely incrementally, without any initial part, will increase the possibility and consequences of this invalidation of earlier work, including SIL. It is actually a result of the Simple Design practice discussed in Section 8.2.6.

6.1.4 Balancing initial and incremental safety analysis

Performing a complete and detailed safety analysis initially and then repeat the analysis incrementally seems to double the safety analysis cost. However, these analyses can be considered as having different purposes. The main purpose of initial safety analysis is to determine a credible SIL of the project. The main purpose of incremental safety analysis is to analyze new, changed and refined requirements to determine the corresponding new, changed or refined safety requirements.

The category of the project should be taken into account. Low requirement dynamism (the Boehm/Turner scale, Figure 4.1) allows detailed level of initial requirements, and thus more comprehensive initial safety analysis. The degradation of the initial analysis results would be low and thus low effort spent on incremental safety analysis. Higher requirement dynamism would lead to a higher level of degradation of the initial analysis results. The initial analysis should then be less detailed, but still at a level allowing a credible SIL to be determined. The incremental safety analysis is more important, and requires a higher effort.

Ge et al. [16] shows an example where they have performed a hazard analysis on a small project, using a simple up-front software design and performing a FTA based on the hardware failure modes and effects. They conclude that a simple design can be sufficient, but it needs to be detailed enough for the preliminary hazard analysis. However, their example is a small and non-complex system and their conclusions may not be valid for larger and more complex systems.

The level of detail of the initial safety analysis will influence on the amount of incremental work. If the initial analysis has been detailed and the dynamism is low, there will be little requirement refining during the development. If the initial safety analysis has been less detailed, there will be more requirement refining and thus reanalysis during the development. Although projects with low requirement dynamism allows a detailed initial analysis, these projects will also benefit with a lower level of initial detail, leading to refining and reanalyzing the requirements regularly and thus incorporate the experience gained through development. This is the more Agile way.

6.1.5 Safety analysis summary

Safety analysis should be performed both initial and incremental. The main purposes of initial safety analysis are to minimize the risk of detecting an increasing SIL late in the project development, and to be able to divide the systeminto subsystems with sufficient independence that may determine different SILs. The main purpose of incremental safety analysis is to handle new and changed requirements. Changes may also affect the requirements that were defined initially but not yet

developed.

Initial safety analysis causes indirectly an extra cost due to less agility. When adding initial work to Agile development, it is not possible to gain full effect of the cost reductions promised by Agile models. In projects with frequently changing requirements, much of the initial work is invalidated which increases the amount of incremental safety analysis. It also increases the possibility of detecting an increased SIL, which can dramatically increase the cost. Frequently changing requirements add extra cost to safety analysis, regardless of the development process. Incremental safety analysis itself will not introduce extra costs, since incremental safety analysis is not direct rework of earlier analyses. Instead, incremental safety analysis adds more details to previous analyses or analyze changes or new requirements that occur.

6.2 Safety assessment

The safety assessment section is divided into four parts. In the first part, aspects of incremental safety assessment are discussed. Then, aspects of safety assessment as a closing activity are investigated. The third part gives some thoughts on how to reduce the effects of the increasing reassessments. Finally, the fourth part summaries the safety assessment discussion.

The safety assessment activity described in this subsection includes the planning of the functional safety assessment, the functional safety assessment and the preparation of the safety case. IEC 61508 indicates that safety assessment should be performed at least once, but preferably inline with the development to give early safety feedback. If safety assessment is done once, it should be done as a closing activity covering all development phases.

6.2.1 Incremental safety assessment

The ideal safety assessment is traditionally been done throughout the development process; when the development in one phase is finished, a safety assessment is performed to detect and handle safety issues as early as possible. To maintain this safety feedback in an Agile environment, safety assessment should continue to be an ongoing activity, i.e. safety assessment should be incremental. In incremental development, an increment may be considered as a phase, and thus a safety assessment can be done after each increment. The lack of documentation produced within the increments, may lead to extra documentation-increments at the end of the project to produce the necessary basis for safety assessment, resulting in safety assessment as a closing activity.

If functional safety assessment is done only as a closing activity, and not incrementally, safety

issues are detected too late in the process, similar to the late error detection of the big-bang integration in traditional software development. This means that the expected benefit of Agile development is lost by introducing a constraint from the traditional development; if assessment is not passed and this is detected late, a part of the system must be redeveloped. Kelly [24] points out that late safety case production gives large amounts of redesign, less robust safety arguments and lost safety rationale. Incremental safety assessment is then needed to be able to influence and supervise the safety while the system is developed.

There are two main problems with incrementally safety assessment is the reassessment of the previous increments. Redmill [33] indicates that revalidation will be an increasingly part of the increment work as the system grows, making less room for actual development. Agile development solved this problem by using automated regression testing, but the problem reoccurs regarding safety assessment.

One point from Kelly and McDermid [25] regarding updating safety cases is the *difficulty in identifying the indirect impact of change*. This problem escalates within an incremental environment where the indirect changes to the system given by the new increment needs to be identified, reassessed and included in the safety case update. Ge et al. [16] remark the lack of measurements to verify the quality of the safety arguments made for each increment.

One problem is the increasing cost of reassessment. Another problem is how to assure that the incremental safety assessment has covered all impacts of the assessed increment and that the achieved level of safety for the complete system is sufficient. To solve the latter problem, the functional safety assessment is required to be done for the complete, growing system for each added increment, maximizing the first problem.

The safety assessment also inspects the development process. It assures that the development process is according to IEC 61508 for the required SIL, and that the software is developed according to the described process. In Agile, the development process is not described in detail, and are expected to be adapted along the development. This means that assessing the software development process must be an incremental activity to have visibility into the development process and to assure that changes to the development process does not conflict with IEC 61508. If this should be inspected as a closing activity, changes to the development process not complying with IEC 61508 may invalidate large portions of the developed software. In addition, information on how the software development process has been executed throughout the project is not present at the end of the project unless there are take actions to document this.

Planning safety assessment are both initial and incremental. The scope and the techniques of

safety assessment is identified early to match the detected SIL. This plan should just be a guide to the scope of the assessment, helping the incremental assessment planning to decide whether a requirement should be subject to assessment.

6.2.2 Closing safety assessment

Using only a closing safety assessment match the minimum requirement of IEC 61508 [20]. However, the previous section concluded that performing safety assessment as a closing activity alone reduces agility, and retains weaknesses of the traditional safety process. However, an additional final safety assessment could be required. In a traditional development, safety assessment of the last development phase also act like a final safety assessment for the complete software. Combining a reduced incremental safety assessment with a closing assessment can reduce the reassessment costs and still assure that all impacts of the changes are assessed.

Closing activities can be done in several levels, e.g. software level, subsystem level, system level. Late integration of subsystems and system is not according to Agile models as this opens for late discovered discrepancies.

6.2.3 Reducing the increasing reassessment

Incremental safety assessment causes a large amount of reassessment to assure that the assessment covers all impacts of the new increment, while using only closing assessment maintains the uncertainties of the big-bang integration. Precautions that can limit this cost and reducing the uncertainties are thus welcome.

One possible solution is limiting the extent of the reassessment by having a architecture and design that is characterized with low coupling between the units and high cohesion within a unit. Partitioning the software into independent subsystems is the first step to obtain this. Low coupling and high cohesion is also a goal for units within each subsystem. When planning safety assessment, the reassessment problem must be considered.

Change impact analysis is the traditional approach for limiting the rework needed. Change impact analysis prior to safety assessment can be done after the implementation is finished, and thus, more information is available to detect impacts, than prior to development. However, to be able to significantly narrow down the area to be reassessed, the traceability must have high quality, perhaps better quality than required by requirement traceability. To obtain high quality traceability, more effort is required in producing the traceability matrices. It should be considered if the time

saved from using change impact analysis will justify the extra work for establishing more detailed traceability matrices.

Using a combination of incremental and closing assessment would make it possible to not do complete reassessments incrementally. The assurance that the complete software is assessed is guaranteed by closing assessment, and incremental assessment will give enough feedback to avoid large surprises late in the project.

Kelly and McDermid [25] introduces the Goal Structuring Notation, a graphical way of structuring a safety case. This GSN is showing the structure of the safety case, and has symbols showing what arguments that are not developed. There exist tool support for the GSN. A graphical representation of the safety case (GSN or others) makes it easier to get an overview of the arguments, their dependencies and what is missing or affected. Perhaps incremental safety case establishment could benefit from a "possibly broken argument" or "affected"-symbol, to be used where there is strong indications that the argument is affected, but the reargumentation is intentionally postponed.

Ge et al. [16] uses the GSN to suggest a safety case structure that promotes incremental safety case argumentation and also indicates what to be argued incrementally and what could be argued as a closing activity. The structure gives separate safety arguments for each module and additional arguments for their interdependencies. Given this structure, the module safety arguments could be developed incrementally with some of the major interdependencies, and the closing activity could be to inspect all the dependencies. However, safety issues regarding the dependencies between units could have large implications and costs when detected late.

Stålhane et al. [40] suggest that a short ($\frac{1}{2}$ day), informal incremental safety review should be performed for the requirements with associated safety requirements, as a part of the Scrum Sprint Review. The independent safety assessment is performed only as a closing activity, but they argue that it will be less extensive because of the incremental reviews. This informal reviews are not performed by the independent safety assessment team, but they can attend. Such reviews will detect some safety issues and give valuable feedback, but the limited extent and duration for the review will still leave a lot to be discovered by the final assessment. The incremental safety process from Stålhane et al. do not indicate that the incremental safety review covers functions indirectly impacted by the changes in the new increment. If this is left entirely to a closing assessment increases the possibility that safety problems are detected late in the project. The closing, formal assessment may emphasize different factors than the informal review. The final safety assessment also implies that the safety argumentation is only done as a closing activity, and this may result in less robust

safety arguments [24]. When using Scrum in general, every implemented requirement is to be reviewed in the Scrum Sprint Review. When developing safety related systems, this will of course also apply to the safety requirements. An informal safety review in the Scrum Sprint Review can be compared to the document reviews used in traditional development. Both in IEC 61508 [20] and in Scrum these reviews are a part of the verification activity, and is no assessment. This approach cannot replace an incremental safety assessment.

Doing safety assessment first incrementally and then one more time at the end of the project seems at least to double the work and thus cost. However, incremental assessment should focus on feedback and the needs for the safety argumentation to avoid late detection of major safety discrepancies, while the final assessment should focus on finishing the safety case and judge the achieved level of safety.

6.2.4 Safety assessment summary

The safety process is affected introducing incremental safety assessment. This gives early feedback that improves safety and safety argumentation. However, incremental safety assessment causes an increasingly amount of reassessments to be sure to cover all impacts of adding a new function to the already assessed software. This implies increased costs from assessment compared to the traditional safety process. Routines for how to reassess the impact of changes given by a new increment must be established. Adding a closing safety assessment reduces the agility of the development. Discrepancies detected by the closing safety assessment, can lead to major redesign late in the project, and may also give less robust safety arguments. Using closing safety assessment only, without incremental assessment, is therefore not recommended.

By combining incremental and closing safety assessment, incremental assessment gives valuable feedback early and reduces the possibility of late detected safety issues. Closing safety assessment assures that the complete system is assessed, and reduces the extent of incremental assessment.

6.3 Safety Validation

The safety validation section is divided into three parts. First closing safety validation is discussed, then incremental safety validation is discussed, and finally a suggestion that the two can be balanced is mentioned. Closing safety validation is discussed before incremental safety validation, because safety validation is a closing activity in the traditional safety process. Safety validation includes testing, but may also include other activities as analysis.

Software verification is assumed to test that the software meets the safety functions requirements. Safety verification should therefore be included in normal testing activities within the software development and is not a subject for the current section. It is, however, assumed that the software have been subjected to verification prior to safety validation.

6.3.1 Closing safety validation

Safety validation and safety assessment are both used for judging the achieved level of safety. In contrast to safety assessment, safety validation has traditionally been performed as a closing activity. One could then argue that performing safety validation as a closing activity would not introduce any safety risks compared to traditional development. However, using only closing safety validation reduces the agility of the development process, as it would reintroduce the weakness of late discovery of errors, from traditional software development.

The benefit of closing safety validation is that the complete and final system is validated and all impacts of previous changes are then covered. This indicates that closing safety validation must be performed, but possibly combined with some incremental safety validation.

In Scrum-terms work not performed within the increment is regarded as Undone Work [27]. Large amount of Undone Work is discouraged and represent risks , for example performance tests are often Undone Work, leading to late discovery of performance problems. This indicates that only performing safety validation as a closing activity is risky, and that some of the benefits of Agile development may be lost. It is suggested [27] that Undone work should be performed in parallel to the development if possible.

6.3.2 Incremental safety validation

Planning safety validation is done in parallel with the development also in traditional development, and this indicates that this activity should have an incremental part. Agile development does not separate the planning from the development. Planning safety validation as an activity separate from the incremental development is not possible, as there are no complete approved set of requirements to base the validation planning upon. However, given the output from the initial safety analysis, there exist a set of requirements that can be used to start planning safety validation. This set of requirement is not approved in the traditional way, and is a subject for further changes, indicating an incremental planning to catch up with these changes. Planning safety validation must then be done incrementally, closely connected to the development process, to be able to track the changes

and new details of requirements.

Another argument for incremental safety validation is that safety validation can give some feedback to the development team. As the safety related system cannot be delivered frequently to operational environment as expected in Agile development, such feedback must come from other sources to retain some of the agility of incremental delivery, see discussion in Section 6.5.

The safety validation should then be planned incrementally, for each increment the corresponding safety validation aspects should be investigated. The validation should also be performed on the incrementally "delivered" software, both to give needed feedback to the development team, and to test and improve the validation itself. It should also be investigated if there are validation tests that are suited for automation. These test can then be repeated for each increment without major cost increase.

6.3.3 Balancing incremental and closing safety validation

The planning of safety validation must be done incrementally. Parts of the actual validation can be done incrementally to provide feedback, but will cause extensive revalidations if all impacts shall be investigated for each increment. By combining incremental safety validation with a closing validation is it possible to avoid the extensive revalidation and still be able to give feedback, and closing validation is where the actual safety validation is done.

6.3.4 Safety validation summary

The planning of safety validation should be done incrementally to be able to incorporate changes from the initial requirement specification.

The traditional safety validation is done as a closing activity, and continue to do this also i Agile environments should result in the same level of safety. However, this reduces agility and maintain the weakness of the traditional safety process of detecting safety implications late in the project. Additional incremental safety validation should be introduced.

Validating incrementally causes an increasing amount of revalidation to cover all impacts of a change. Automation can reduce this cost. However, the cost may be justified by getting a better and more Agile safety validation, and that the validation provides valuable feedback to the development team, also on non-safety issues.

6.4 Traceability

Requirement traces are essential when developing safety related systems. Both forward and backward traces are required by IEC 61508 [20]. Actually, IEC 61508-7 states that the aim of traceability is to maintain consistency between lifecycle stages. In Agile, the stages are the increments, and as one increment is not based upon the work of the previous, no trace between the increments can be established. The interpretation of this aim within an Agile context can therefore be to verify that all requirements are developed and tested, and to construct evidences for the safety case.

The goals of traceability when developing safety related systems are then to establish safety arguments and to pass certification. IEC 61508 requires change impact analysis, and traceability is an essential part of change impact analysis. In addition, traceability gives the safety assessor a map into the documents and code to be assessed for a given increment.

The traceability information can be established on-demand or as a continuous activity [17]. The next two sections discuss these strategies in view of incremental development and how they influence the safety process. The role of traceability in change impact analysis in safety analysis and safety assessment within an incremental environment is also discussed.

6.4.1 On-demand update of traceability

The on-demand update of traceability information requires a trigger to start the update [17]. In traditional development this can typically be the end of a development stage. In incremental development, the trigger can be the end of an increment. However, as increments are considerably smaller than a traditional development stage, this update will be done often enough to be regarded as continuous trace update.

Trace update can be done prior to each delivery when using incremental delivery. Since incremental delivery is not possible for most safety related systems, this approach gives only one trace establishment, prior to the delivery. Trace establishment prior to the final delivery gives no overhead due to the traceability establishment. However, it will make incremental safety assessment impossible. Safety assessment and safety arguments rely heavily on available traces. It also gives an extra handoff at the end of the project. The final safety assessment and safety validation cannot be done until the traceability information is established. Establishing the traceability information as a closing activity also implies that the traceability activity benefits are restricted to fulfilling the certification requirements, and possible benefits during the development cannot be utilized.

6.4.2 Continuous update of traceability

The continuous update of traceability information in an incremental environment implies that the traces are updated within the corresponding increment. Constructing requirement trace links incrementally is not just establishing the traces for the current increment, but also updating traces which are influenced by the current increment and removing traces that have become obsolete. As the developed system increases, the job updating traces increases. The updated traces, and an overview of the changes from the last edition, are all inputs to the functional safety assessment. The requirements trace should then be established within the development increment (e.g. included in Scrum Definition of done) and be delivered at the same time as the software.

The quality of traceability information consists of the granularity, the precision and the level of coverage, and increasing the quality result in increasing the cost of maintaining the traceability [21]. To reduce traceability maintenance costs, Ingram and Riddle [21] suggest varying the quality of traceability over the system. Safety assessment planning should then also include deciding where the traceability must be fine-grained and precise to fulfill safety assessment needs. It must also be decided if there are areas where traceability quality can be lower without compromising safety. In addition, the plan should include which links that must be updated incrementally and which links that can be updated more seldom.

Capturing trace links are not always expensive, Cleland [12] points out that some traceability can easily be obtained in Agile projects, since requirements and acceptance tests are directly connected. With minor additions, the version control tool can provide some traceability into the code. Traceability can be achieved in Agile by using requirements as executable test [27]. The trace link from requirement to test is then implicit as they are the same document. The acceptance tests and unit tests should be tagged with the corresponding requirement ID, obtaining a trace between them, and consequently between requirements and code.

When applying continuous update of traceability and when traceability is used as a basis for safety assessment, the challenge is to assure that the traceability information is stable during a feature assessment and according to the code and document version that are assessed. The consequence is that version control also must be applied for traceability information. The assessment basis when assessing a particular feature is then the corresponding versions of the code, documents and traceability information.

6.4.3 Using traceability in change impact analysis

Using traceability in change impact analysis requires that the traces already are defined. This is normally the case for changes to already implemented requirements and changes caused by identified errors. However, when developing incrementally, impacts of the new increment cannot be directly identified by using traceability. The new increment implements new requirements that have no established traces. Best case is to follow the traces of a related requirement (if requirement relations are recorded). When the impact of the new requirements forming the new increment is to be estimated, the traceability assistance is at best imprecise.

Larman and Vodde [27] discourage using traceability for change impact analysis, when the goal of the traceability is to discover which tests that should be rerun. They point out that when tests are automated, all tests can be rerun for every change. This is the Agile answer to traceability for change impact analysis. However, when developing for safety, it is not only the test coverage that needs to be investigated. Safety assessment is not automated and cannot be rerun totally for every change without unacceptable cost. The safety validation may include some automated tests, but also includes manual tests and inspections.

When a new increment is assessed and validated, the new functions are implemented and the corresponding traces are established and can be used for change impact analysis. One question here is if the trace granularity is detailed enough to identify a small enough area for reassessment or revalidation. As the purpose of this change impact analysis is to reduce the amount of assessment rework, assessment rework cost reductions should exceed the cost of maintaining any extra trace information needed for change impact analysis.

6.5 Incremental delivery

Incremental delivery is where a system with reduced functionality, but included some of the customers most valued functions, are delivered early. The system is then frequently updated with new functions until the system is complete. Incremental delivery is stated as a major success factor for Agile development and is expressed in one of the Agile Principles.

Paige et al. [30] have some objections regarding incremental delivery for safety related systems. It is generally not possible to deliver only a part of a safety related system. The minimum size of a safety related system that can be certified and delivered are often very large, and the certification requirements prevent an incomplete system to be delivered. Paige et al. also remark that the feedback from customers on a safety related system are of limited value, based on observable behavior. They

also point out that a safety related system consists of very different components, hardware, software and mechanical, built by separate teams. These components must be integrated for each release.

Incremental delivery gives rapid feedback from the customer. This is one of the main advantages of using incremental development [34]. Safety related systems cannot fully utilize this Agile practice, but by using simulated environments to where the system may be "delivered", some of the incremental delivery benefits can be regained.

If a safety related system can be able to use some form of incremental deliveries, this will affect the safety process. The current chapter will discuss safety analysis, safety assessment and safety validation in respect to incremental delivery. In addition, some thoughts on how to get some of the benefits of incremental delivery even if this is not possible in the extent that Agile models expect.

6.5.1 Safety Analysis

Safety analysis is an activity that is performed initially and incrementally. Since this activity never is a closing activity, it is not affected by incremental delivery.

6.5.2 Safety assessment and validation

Functional safety assessment and developing a safety case are both activities with a large portion at the end of the development. Safety validation is the finishing activity within a traditional safety process. All these activities are clearly affected if the system is delivered incrementally. Safety assessment and safety validation must then be done largely as incremental activities. The closing activities must be minimized when using incremental delivery.

As discussed above, incremental safety assessment and safety validation cannot assure safety unless complete reassessment or revalidation is done for each increment. Unless the system is very small, this strategy will be too expensive, and frequently delivery is therefore not cost-effective.

6.5.3 Improving agility when incremental delivery not possible

Frequent incremental delivery cannot be recommended for safety related systems. Without the frequent delivery, agility is reduced substantially. To be able to use Agile development models at all for safety related systems, it is important to regain some of the missing feedback.

One possibility is using internal deliveries to independent teams. "Deliveries" to simulated environments can however be done without a complete safety assessment and validation, since the main goal is to gain feedback to the software development as early as possible. The safety assessment

and safety validation could possibly be regarded as "customers" when the system developed as far is delivered for assessment and validation. They provide feedback with focus on safety.

If incremental delivery to the operational environments is possible and desired, the cost and impact of closing activities may be reduced by not delivering as frequently as each development increments. This means planning the releases in advance, i.e. a more plan-driven approach. The Agile principle of frequent delivery is then not followed, but the larger deliveries to the customer can be combined with more frequent deliveries to non-operational, simulated environments.

6.6 Short increments

The Agile models promote short increments in the range of 14 - 30 days. Safety systems have demands that challenge the Agile increment length. The basis for safety assessment includes documentation and requirements trace that should be prepared within the increment. The safety functions requirements pose a challenge, since they are closely related to other requirements and cannot be developed separately. In addition, the complexity of a safety related system may not allow the requirements to be broken down to the detail that is expected in Agile development. In complex systems, the tight dependencies between subsystems may make it impossible to find small enough bundles of features that can be handled within one increment [43]. The extra work related to incremental safety validation and safety assessment must be considered.
Sources for increased workload includes:

- A requirement has related safety requirements that should not be implemented separately
- Extra development activities required by the safety standard
- Establish and update the requirements traceability information
- Establish and update additional documentation needed by a safety related system
- The complexity of a safety related system
- Dependencies to the hardware development
- Lack of certified tool support
- More extensive verification procedures
- Safety analysis
- Safety validation
- Safety assessment

The increment length can be increased or the increment scope can be reduced. Increasing the increment length makes the development process less Agile. Reducing the increment scope can reduce

the flexibility for adjusting the increment scope when it does not progress as planned. Reducing the increment scope can also result in separation of requirements that would benefit from being implemented together.

The main purpose of a short increment length is to deliver software frequently to get valuable feedback as early as possible. When the safety related system cannot benefit fully from these frequent deliveries, the increment length could be increased some without compromising the Agile aspect too much. However, some of the listed work can be removed from the development increment. Safety assessment and safety validation is required to be done by independent people[20]. If assessment and validation staff is included is the development team, the independence requirement would create handoffs within the development team. Because a handoff to the assessment or validation team is inevitable, it can be justified defining safety assessment and safety validation as Undone Work.

Stålhane et al. [41] proposes separate documentation team working in close relation with the Scrum team. However, they limit this separation to apply only to documents not related to code development. To avoid an extra handoff via the documentation team, all documents required by the safety assessment team to assess the code should be established by the development team. However, some types of documentation, as user documentation, are best suited to be written by an independent team.

6.6.1 Safety functions requirements

The safety functions requirements are requirements to the safety functions, and arise from the functional requirements of the system. The safety functions requirements are not independent requirements that can be developed separate from the related functional requirements.

6.6.2 Safety assessment

Ge et al. [16] indicates that producing safety case will require increments of longer duration than the advised 2-4 weeks in Agile. The software cannot be delivered without the safety argument, and thus the time necessary for producing the safety argument would define the increment duration. Stålhane et al. [40] has not included safety assessment directly in the increment, but instead they suggest a safety review in what seems to be an expanded Scrum Sprint Review by $\frac{1}{2}$ day. The total length of the Sprint + planning and review activities is then increased.

Larman and Vodde [27] uses the term Undone work for the work decided not to be done within

the increment. This is not unfinished work, but work that is not included in the Definition of done, and thus is expected to be done outside the increment before the product can be delivered. The Undone work will then increase for each increment added. Their main advice is that this should be avoided, but they also acknowledge that some work would take too long to include in an increment. This opens for the functional safety assessment to be regarded as Undone work, and could be incrementally handed off to an assessment team. Although handoffs should be avoided in Agile [27], the handoff to safety assessment cannot be avoided because of the required independence of the safety assessor.

There is strong indications that the functional safety assessment should be done outside the increment development to reduce the need for an expanded increment length. Having safety assessment done by a separate team is neither advised, nor unprecedented in Agile.

6.6.3 Safety validation

The safety validation team does not have the explicit requirement of independence as the assessor in IEC 61508, but the definition of independent person, -department, -organization in IEC 61508-4 mentions that this applies for both assessment and validation. This indicates that safety validation should be excluded from the development team as for safety assessment. The handoff to the validation team is then inevitable.

As safety validation cannot be included in the Definition of Done, it is regarded as Undone Work, and the safety validation team can be regarded as an Undone Work-team. The development team will then “deliver” software and documentation to both the safety assessment team and the safety validation team. One essential question is whether the delivery to the validation team should be assessed before validated. This would introduce an additional handoff. When the early validation cycles aim at planning and giving feedback, no actual validation is performed. When closing safety validation is started, the main portion of the software is already assessed to a large extent. There will always be feedback from validation to development, from assessment to development, from assessment to validation, introducing new changes causing new assessment and validation. Adding extra dependencies here will cause delays. Allowing validating non-assessed functions may cause some extra functions to be revalidated. The latter is then the most Agile solution, and the safety is not compromised as all software will be both assessed and validated before delivered.

6.6.4 Improving coping with complexity within an increment

Agile development models assume that the software easily can be participated into small independent increments. The complexity of safety related systems do not allow this in the degree expected by Agile models. In this case the possibilities are to increase the length of the increment to cover all dependencies, or to find a way to keep the increment length.

To be able to keep the increment length, it could possibly be introduced a form of lookahead planning [14]. The upcoming Sprint is planned in detail, but also the next two Sprints is taken into account to detect dependencies early. Cohn suggest lookahead planning to detect dependencies between teams, but it can also be used to discuss dependencies for succeeding Sprints. Highly dependent safety requirements can then be done by consecutive Sprints, and the team has an overview of the functionality of the next three Sprints instead of just one. This requires that the Product Backlog is ordered in a way that closely related safety requirements are grouped together.

6.7 Summary

To achieve the same level of safety as traditional safety development offers, Agile development must include some of the practices from the traditional development. This includes adding more initial and closing work. To reduce the effect of these additions, the safety process should include incremental activities.

By adding initial and closing activities to Agile development, agility is reduced and it is not possible to gain all Agile benefits. The expected cost reduction from using Agile models is therefore reduced.

When doing incremental safety activities, the problem of increasing rework arise. This considerable increases the cost compared to traditional safety activities. However, incremental safety activities compensate for the weaknesses in traditional safety processes. To be able to use incremental safety activities, the increasing rework must be reduced. One suggestion is to combine initial and incremental safety analysis, and incremental and closing safety assessment and validation. Tool support and automation of the trace maintenance are necessary to perform the continuous updates that arise in an incremental safety process.

The increment length when developing safety related systems, increases compared to standard Agile development. This is due to, among others, the added complexity of safety related software and the preparations of the basis for safety assessment. However, safety assessment and safety validation cannot be included in the work done each increment, due to the requirement of independence.

Safety analysis Safety analysis must be done initially to the detail where a credible SIL can be established. The initial work reduces agility, but can be justified for systems with few requirement changes. Incremental safety analysis must be done to handle changing requirements. The incremental safety analysis also keeps the focus on safety and assures that the safety requirements are understood by the developers.

Safety assessment Safety assessment can be done only as a closing activity, and still comply with the minimum requirement by IEC 61508. However, this reintroduces the big-bang integration problem of traditional development models and leads to late detected safety issues and less robust safety argumentation. This also reduces the expected effect of using Agile models, since early feedback is one of the key factors of Agile.

Doing safety assessment incrementally implies a lot of reassessments to cover all impacts of the new features implemented for each increment. To assure that the all impacts are covered, an increasingly part of the code must be reassessed. Additional work related to establishing the basis for the safety assessment, like extra documentation and traceability, is also subjected to rework. Maintaining very high quality traceability can reduce the extent of the reassessment each increment, but adds extra cost that possible not is justified by the reduced reassessment.

A combination of incremental and closing safety assessment can be used, since incremental delivery cannot be utilized by most safety related systems. The main purpose of incremental assessment is to give early feedback, also focusing on the safety argumentation needs. The main purpose of closing assessment is to assess the complete system and also cover all impacts of incremental development. The final assessment makes it possible to reduce the extent of incremental assessments.

Safety validation The planning part must be done incrementally to be able to detect changes and detailing of the requirements. Performing safety validation purely as a closing activity corresponds to the closing safety validation of the traditional safety process and can be expected to give the same level of safety. However, this maintains the weakness of late discovery of safety errors, and reduces agility. An incremental part of safety validation is suggested to give early feedback. Incremental validation gives an additional cost, but test automation can reduce this cost.

Chapter 7

Discussion: The safety process within an iterative environment

This chapter looks at the iterative development part of Agile models as defined in Section 2.1.4. The first part of this section analyzes the concept of iterative development and its effect on the safety process. The main focus is on functional safety analysis. The second part looks at how iterations are used in Agile models.

I have not found any literature that covers this theme properly. All articles mentioning iterative development together with safety analysis are actually talking about incremental development (as defined in Section 2.1.3).

7.1 Functional safety assessment of iterative work

The iterative workflow has the same limitation as the traditional workflow, which means that any phase is directly dependent on the work from the previous phase. When iterating, the work is done once, then improved until it is decided to be good enough.

To be able to influence the safety of the iterated software, safety assessment can be performed for each iteration. This situation is illustrated in Figure 7.1. As the figure shows, two iterations must be done before safety assessment results can influence the next iteration. There is also at one or more iteration at the end of the project, which only handles safety feedback.

One problem is to decide when one iteration is complete so that the next iteration can start. The software development is not finished until all iterations are done. Another problem is that it is impossible to know in advance how many iterations that are required to finish the development

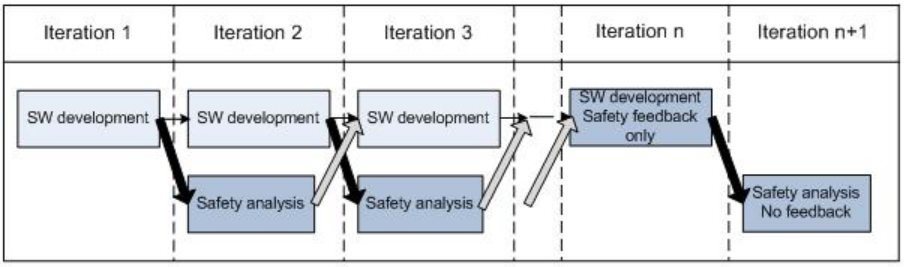


Figure 7.1: Safety assessment when iterating

with an acceptable level of quality. In early iterative development models, risk analysis is used to to identify the parts of a project that need extra examination. In the spiral model [7], prototypes are used to inspect possible problem areas, and the work is considered to be done when all risks are mitigated. The experiences from the prototypes are then used when developing the complete system at the end of the spiral. In this case the prototype (first iteration) has a defined end and can be assessed. In addition, the results can be used when the system is developed (second iteration).

Another question is how much effort should be invested in assessing work that is not finished. Should the assessment shown in Figure 7.1 use rigorously assessment techniques and produce a complete safety argument for each iteration, knowing that the software can be radically changed during the next iteration? Or should the iterations only produce advices to the development team, and the complete assessment and the safety case establishment be done in the last iteration? These questions will be discussed in the next section.

7.2 Iterative development used in Agile models

Agile development uses very short increments, i.e. the software is divided into very small pieces before the implementation are done within each increment. With such small parts there may be no need for iterating, since the solution often seems straightforward. One important goal with iterative development is learning. In Agile development this learning is done prior to the actual development, e.g. Scrum use workshops and XP use spike solutions. In addition, Agile models use Test Driven Development with very small iterations between testing and coding.

7.2.1 Scrum - workshops

Larman and Vodde [27] suggest using workshops to create a common understanding among the participants. These workshops can be used at various times in the development. In the beginning of a project a workshop can be used to refine the initial Product Backlog. Then the Product Backlog can be regularly refined using requirement workshops before the start of each increment. In addition, workshops can be used for modeling, architecture and design.

In traditional thinking, safety analysis should be done prior to the workshop and the result from the workshop should be assessed before it can be used in further development. This will dramatically reduce agility of the development process, and create dependencies and delays. The workshop will be delayed until safety analysis is done and the development will be delayed until safety assessment is performed. In traditional thinking it is also assumed that the output from the workshop is locked and unchangeable. In Agile thinking this is only a starting point, and it is likely to be changed. A workshop must not be mistaken for a phase in traditional development. The output from a workshop do not have the quality of a sequential phase, where the phase output can be checked against the output from the previous phase.

There is no need to separate the safety analysis from the workshop. Safety specialists must be workshop participants on the same level as the other participants. In this way the safety analysis will evolve along with the discussion, and the safety needs are communicated to and discussed with the developers. However, sometimes safety needs extra focus and may require a separate workshop. There are safety analysis techniques that will fit into the workshop frame, for example HAZOP and FMEA.

7.2.2 XP - Spike solutions

Thorsen [42] classifies Spike solutions as a practice that can be only partly adopted when using XP for developing safety related systems. She comments that Spike solutions should be more formal, but she does not explain why. A Spike solution is a simple program exploring possible solutions to a problem and ignoring all other concerns[1]. A spike solution is not a complete solution, and it is expected to be thrown away. The final solution is developed within the normal frames of an increment. The difference between a workshop and a spike solution is that the spike solution most often involves programming.

A Spike solution has a defined start and a defined end. A Spike solution can start with a safety

[1] http://www.extremeprogramming.org/rules/spike.html

analysis to explore the safety aspects of the spike and end with a safety assessment to see if the safety goal is achieved. However, making this too formal will introduce handoffs and dependencies between the safety analysis team, the spike solution team and the safety assessment team. A better way is to view this as a possibility of exploring safety along with the functional exploring.

The guidelines in XP regarding Spike solutions include the words "...ignoring all other concerns". Does this mean that it is ok to ignore safety within a Spike solution? The spike solutions are supposed to be as small as possible and introducing safety may shift the focus from what is expected to be learned. Some spike solutions might have none or only minor safety aspects. In these cases, safety functions can be ignored. Other functions need to be inspected more closely before any safety issues can be addressed. In this case two spikes might be needed, the first ignoring safety to gain knowledge and the next including safety functions. Also pure safety spikes should be encouraged, to focus on complex or serious safety issues.

Before starting a spike solution it has to be decided whether or not safety issues should be addressed within this spike. If it is decided that safety issues should be addressed or that it is a pure safety spike, the spike solution should start with a safety discussion. This discussion should involve the whole team in thinking safety for the function to be explored. This will assure that safety aspects are handled. If necessary, a safety assessor can be involved in the final steps of the Spike solution, discussing the proposed safety solutions. A question is, should there be any formality in documenting this safety-discussion. The safety analysis and safety assessment will be done again when the actual development is started. The spike solution is documented with the tests and code produced, and this should be sufficient for recording the lesson learned. Any results from an earlier discussion should not be considered as binding. The actual development should be seen and handled as a development from scratch.

7.2.3 Test Driven Development

Test Driven Development (TDD) has very short iterations. First a test is written, the unchanged code is tested and fails, then the code is changed and tested until it passes the test. These iterations can be as short as a few minutes. It is not practical to assess the test script before testing, this will disturb the flow of the development and delay the development seriously.

Because TDD is used within an incremental frame, only a small part of the software is developed. Even if there is given no safety feedback during the development, the safety analysis before starting, and the safety assessment of the finished increment should be sufficient.

7.3 Summary

Workshops and spike solutions do not produce any deliverable software. The deliverable software is produced within the normal increment of development with the normal environment that includes safety analysis and safety assessment. However, if safety is ignored in the workshop or in the spike solution, the lesson learned is perhaps not the lesson that should have been learned. The important issue is to influence the safety thinking, without overwhelming the workshops or spike solutions with expensive ceremonies that only discourage using these iterating practices. Workshops should also be encouraged for communicating safety to the development teams, and spike solutions should be used for exploring possible safety solutions.

The iterative part of Agile development mainly affects the safety analysis. The safety analysis should be done prior to, or as a part of a workshop, prototype or spike solution. Only the last iteration (i.e. the actual development) produces a finished code that can be assessed and validated. Safety assessment and safety validation are therefore not affected by the iterative part of Agile development.

Chapter 8

Discussion: The safety process and other Agile practices

In the previous two chapters incremental and iterative parts of Agile development models are discussed. These are Agile practices that concern the outer process of Agile models. In addition to these two parts, Agile development has a set of other practices that may influence the safety process. These Agile practices are discussed in the current chapter. The discussion is limited to the practices of Scrum and Extreme Programming. In addition, the communication and documentation of Agile models are discussed.

8.1 Scrum

This section discusses the main elements of Scrum [37], which are the Scrum Team, the Sprint Events and the Scrum Artifacts. The Scrum team consists of the Product Owner, the Development Team and the Scrum Master. The Sprint consists of five events, the Sprint Planning meeting, the Sprint development, the Daily Scrum, the Sprint Review and the Sprint Retrospective. The artifacts consist of the Product Backlog and the Sprint Backlog. The Definition of Done is an important part of Scrum and is also discussed. In addition to these elements of Scrum, Feature teams, a practice for scaling Scrum [27], is discussed.

8.1.1 The Product Owner

The Product Owner is the sole person responsible for managing the Product Backlog. [37]

Safety related systems have additional demands when prioritizing the Product Backlog, e.g. to implement safety functions as early as possible. The Product Owner must have a clear understanding of the safety requirements. He should be assisted by persons from the safety analysis team when grooming the Product Backlog and prioritizing the items. The Product Owner will participate in the safety analysis and the result from the analysis may influence the product Backlog prioritizing.

8.1.2 The Development Team

The development team is implementing the Product Backlog items selected for a Sprint [37].

The main task for the development team is to develop the software. This activity is box 10.3 in Figure 3.12, and not directly involved in the safety process, but there are some connections. The development team gets input from safety analysis through the safety requirements. The development team is also responsible to provide the basis for safety assessment and safety validation. Additionally, the safety analysis team should be available for discussions concerning the safety requirements at any time. Involving the development team in safety analysis is therefore desirable.

8.1.3 The Scrum Master

The Scrum Master shall ensure that the Scrum team understands Scrum and use Scrum correctly [37].

The Scrum Master must have an understanding of the restrictions and challenges that arise when developing a safety related system using Scrum. Safety related development requires a development model that in some cases contradicts the Scrum rules and practices. However, the Scrum Master belongs to the development process, not to the safety process.

8.1.4 Sprint Planning Meeting

The Sprint planning meeting selects the Product Backlog items to be developed for the next Sprint, and details them into Sprint Backlog items [37].

When planning a Sprint, safety analysis of the planned functions must be included. This is discussed in the section for incremental safety analysis (see Section 6.1.3). However, the safety analysis activity adds extra workload to the Sprint planning meeting. In large scale Scrum, the activity of grooming the backlog [27] refines, adds new, and removes items from the Backlog. Some of the safety activity should therefore be handled here, reducing the extra workload on the Sprint Planning meeting.

8.1.5 The Sprint development

This is where the Sprint Backlog is implemented. The result is added to the existing software to produce a potentially shippable code. [37]

The Sprint development is performed by the Sprint development team, see the discussion in Section 8.1.2.

8.1.6 Daily Scrum

The Daily Scrum is a 15 minute meeting, to identify what has been done the previous day, and planning what to do the current day [37].

The Scrum Guide [37] states that only the team members can participate in the Daily Scrum, since this is no status meeting. However, representatives from the safety assessment team should be allowed to occasionally observe these meetings if desirable. This will obtain visibility into the software development process. In traditional software development, the process is visible through the delivered documents. In Scrum (and other Agile models), delivered documents and code do not describe how the development process was followed. Observing the Daily Scrum is one possibility of assessing how the software development process is implemented.

8.1.7 Sprint Review

The Sprint Review meeting is held at the end of the Sprint to inspect the new increment. The Scrum Team and the stakeholders are discussing the new functions. [37]

The safety stakeholders are of course the safety analysis team that is responsible for the safety requirements. Representatives of the safety assessment team and the safety validation team should also attend the Sprint Review meetings. This will give a face-to-face introduction to their later work, but is not intended to replace any written documents that are the main basis for their job.

8.1.8 Sprint Retrospective

The Sprint Retrospective is a meeting where the team members inspect how they worked in the last Sprint and suggest improvements for the next Sprint [37].

Safety related systems have more requirements to the software development process than regular Scrum. The development process is customized according to a safety standard (e.g. IEC 61508 [20]) and must comply with the requirements at a given SIL. This means that changing the development process can affect safety, and changes should not be implemented without consulting the

standard. The safety analysis team should attend this meeting as experts of this standard, and discuss and approve the changes. Major changes to the development process should be assessed prior to implementation.

The Sprint Retrospective is also an opportunity for the safety assessment team to assess that the software development process is implemented as expected. A representative of the assessment team should attend the Sprint Retrospective.

The Sprint Retrospective can propose process improvements for the next Sprint. When developing a safety related systems, adaptions of the development process cannot be done without checking safety implications. By also including improvements in the Product Backlog [27], they can be subjected to a safety analysis before implemented. Activities required by the safety plan and the safety standard are often less agile and may be flagged as activities to change or remove. The safety analysis can then reject or adjust these improvements so that safety is not compromised.

8.1.9 Product Backlog

The Product Backlog in Agile development is equivalent to the requirement specification in traditional development. The Product Backlog is an ordered list of everything that is needed to make the product, and is the single source for any changes made to the software [37].

8.1.9.1 Product Backlog and requirement specification

The Product Backlog is organized quite differently than a traditional requirement specification. Dependencies between backlog items are considered as reduced flexibility. The Product Backlog focus on the features not implemented. It is, however, not clear what is done to the items that are completed. Probably they are marked “finished” or moved to an unofficial finished backlog and it is probably no effort of updating these items if they are affected by changes. Changes to a requirement will appear as a new backlog item with no reference to the original requirement. When developing safety related systems, it is expected that the requirement specification is complete and up to date during the project lifetime. An updated and complete requirement specification is one important part of the safety case that is presented to the certification authorities. The central question is if the Product Backlog can be used as a requirements specification.

The Product Backlog in Agile is a to-do list. An Product Backlog item can be a requirement, a change to an earlier implemented requirement, or simply describing an error correction, possibly not affecting the requirements. If the Product Backlog is changed to be a complete and updated

requirement specification, the Product Backlog will not be useful as a to-do list. Another solution may be to divide the Product Backlog into two parts. One having a part of finished items containing updated requirements, and the other are the to-do list containing new requirements, changes to requirements and error corrections. In fact this is two different uses and thus two separate documents. The Product Backlog items are prioritized according to which is to be implemented next, and are continuously re-prioritized. A requirement specification is organized by functionality and dependencies. The Product Backlog cannot contain the updated requirement specification. The Product Backlog should continue as a pure to-do list also when developing safety related system. It should be supplemented with a separate requirement specification. The Product Backlog Items should refer to the related requirements. The Agile document most suitable for containing the complete set of requirements are the acceptance test specification [27].

The activity of grooming the backlog [27] must then be extended to update changed requirements. A changed requirement will then appear in the Product Backlog as a change request, and the updated requirement specification will contain the complete requirement. Both will be available to the developers. The Product Backlog is still the Agile to-do list, but accompanied by a requirement specification available for assessment and certification.

8.1.9.2 Separate Safety Backlog

Stålhane et al. [40] suggest removing the safety requirements from the Product Backlog, creating a separate Safety Backlog. Functions in the Product Backlog can be linked to one or more items in the Safety Backlog, and the Safety Backlog items refer back to the actual Product Backlog items. Stålhane et al. point out that there exist dependencies between safety requirements and functional requirements. Their Safe Scrum is only invoked if a requirement in the normal Product Backlog refers to a requirement in the Safety Backlog.

Larman and Vodde [27] are on the other very clear that it should be only one Product Backlog, even for large systems. They suggest instead using different views into the Product Backlog. A safety view into the Product Backlog will offer a overview of the safety requirements not yet implemented.

A problem with the interconnected backlog items suggested Stålhane et al., is to decide when a Backlog item can be removed. If a safety requirement is linked to more than one functional requirement or a functional requirement is linked to more than one safety requirement, the criteria for removing a backlog item is not clear. If a safety requirement are connected to more than one functional requirement, and the safety requirement is removed when it is handled the first time, the

remaining links are pointing nowhere.

If the purpose of the separate Safety Backlog is to identify all safety requirements, the Product Backlog cannot provide this as it contains only non-implemented requirements and change requests. The requirement specification identifies all safety requirements, and can be used to produce a list of all safety requirements.

If the purpose of the separate backlog item is to be able to implement the safety requirement together with the associated functional requirements, this must be handled when prioritizing the Product Backlog. Safety requirements that will strongly benefit from be implemented together with the associated functional requirement, should be included in the same Backlog Item.

8.1.10 Sprint Backlog

The Sprint Backlog contains the Product Backlog items selected for the current Sprint. The Sprint Backlog is the plan for how to implement the Sprint [37].

Safety requirements from the Product Backlog are detailed into the Sprint Backlog in the Sprint planning meeting, where safety stakeholders are present for discussions.

8.1.11 Definition of Done

The Definition of Done is a shared understanding of what is needed to be done before each Sprint Backlog item and the new increment are finished [37].

Incremental safety analysis is done prior to establishing the Sprint Backlog, and thus outside the activity where the Definition of Done applies. Incremental safety assessment and incremental safety validation should be done by independent teams. Despite that these activities are required to define a new increment finished and deliverable, safety assessment and safety validation should not be included in the Definition of Done (Chapter6.6). However, the basis for incremental safety assessment and incremental safety validation must be included in the Definition of Done, including documentation and traceability for the new functions. If this is not included in the Definition of Done, i.e. is done at a later time or by a separate team, this would introduce an extra handoff and thus reduce agility.

The Agile ideal is that all work is included in the Definition of Done. In large scale development this could be difficult. Larman and Vodde [27] recommended to define a set of activities called Undone Work, containing all activities that must be carried out after the Sprint is finished, and before the system can be delivered. The Undone Work-list is a complement to the Definition of

Done. Together will they define what is needed to be done before the new increment is finished (Definition of Done) and deliverable (Undone Work). Undone work may be done later in separate Sprints by the development team or by a separate team in parallel with development.

Defining safety assessment and safety validation as Undone Work and perform the Undone Work by a separate team, allows separate teams for safety assessment and safety validation. This fits the independence requirement for the teams performing safety assessment and safety validation.

8.1.12 Feature teams

Feature teams are not a practice of standard Scrum and are not described in the Scrum Guide [37]. Feature teams are the preferred way of organizing teams when scaling Scrum, in contrast to component teams [27].

The organization of the development team may influence safety analysis. The safety analysis must be done at the complete feature prior to Sprint Backlog establishment. Using component teams, the Sprint planning meeting must contain all component teams involved in the features to assure that the safety issues are handled across the components.

Component teams are less agile. Such teams create more coordination and handoffs and the responsibility for a feature is divided between teams [27]. Handoffs introduce a risk of losing information and divided responsibility. It includes a risk that not all aspects, including the safety aspects, of the feature are handled. It is extra challenging to the safety analysis team to assure that the safety requirements are distributed correctly across the component teams. Hence, feature teams should be favored when developing safety related systems.

In Section 6.6.2 it is advised not to include safety assessment or safety validation in the software development team. Safety assessment and safety validation is done after the complete feature is developed. The organization of the software development team is therefore irrelevant for these safety activities.

8.2 Extreme Programming (XP)

This section refers to the core practices of Extreme Programming (see Section 3.1.2.4). The practices will be examined to decide whether they affect the safety process or not. How a practice influences the safety process, and how the safety process influences the practice is then discussed for each of the relevant practices.

8.2.1 Whole Team

This practice was earlier called On-site customer, meaning that the customer should be an integrated member of the team, owning the requirements. Safety related systems often involve a large number of stakeholders and customers. Such systems also require knowledge from a various number of subdomains. A single customer cannot be expected to have the necessary knowledge of the entire system. Having a large team of customers available full time for a longer period is unrealistic [30]. It can neither be expected that the customer has the required technical safety expertise [42].

The safety process requires *owners* of the safety requirements, playing the role of the customer regarding the safety aspect of the system. Whether this role is occupied by a real customer is not important; the main purpose is to participate in establish and prioritizing the safety requirements and follow up on the safety requirements in the Sprint Reviews. The safety stakeholders need to participate in the safety analysis to be able to establish the safety requirements. The safety stakeholders should also be available for the safety assessment team and the safety validation team.

8.2.2 The Planning Game

The Planning Game consists of Release Planning and Iteration Planning. The Planning Game must include safety analysis for the planned functions. This is discussed in the section for incremental safety analysis, see Section 6.1.3.

8.2.3 Small releases

This practice advocates frequent and small releases. This also implies very short increments, one to three weeks is suggested [44]. This practice is closely related to incremental delivery and is discussed in Sections 6.5 and 6.6. These sections conclude that small and frequent releases to end users are the discouraged for safety related systems. However, small releases may be delivered to simulated test environments.

Safety relates systems need larger releases than suggested in XP to maintain safety across the system and to handle the additional work from implementing the safety requirements. If too small releases are used, the safety activities are done on an insufficient basis and may fail to detect safety threats. If too large releases are used, the agility of the project is reduced.

8.2.4 Test Driven Development

Test driven development contains the programmer tests, also called unit tests. Test-driven development is an iterative part of XP. This practice is discussed in Section 7.2.3.

8.2.5 Customer tests

In addition to the programmer tests, there are customer tests, also called acceptance tests. Since the customers are responsible for the user stories, they are also responsible for testing the user stories. The customer tests are then created from the user stories. These tests are back-box tests that must be run and passed before a user story is considered finished. The customer tests should be automated. This makes it possible to run the tests often, as a regression test. [44]

The safety requirements must be included in the customer tests and be a part of the regression test. This also implies that the stakeholders originating the safety requirements, are playing the role of the Agile customer regarding the safety requirements, also if they are not actual customers. The safety process itself is not affected by the customer tests, but assumes that they exist and are run frequently. The safety validation is a separate activity, and is not covered by the customer tests practice.

8.2.6 Simple Design

The Simple Design is the practice of selecting the simplest design possible for the current task, even if it is thought that a more complicated design could be beneficial when implementing later tasks. This practice is also referred to as Do Simple Things, You Arent Gonna Need It (YAGNI), Once And Only Once or Simplify Vigorously[1]. Anticipating future tasks are considered counterproductive, as the future tasks may be changed or not implemented.

Boehm and Turner [9] single out this practice as a practice that do not scale very well and that large project with foreseeable changes will benefit from more up-front architecture and design. Paige et al. [30] mention that a simple design is desirable in safety related software and that this practice can be fully adopted to safety related development.

Turk et al. [43] state that Agile models assume that the cost of a change does not dramatically increase over time. This contradicts the accepted belief that errors detected late are more expensive to correct than errors detected early. Use of integrated development environment and automatic tests

[1] http://c2.com/cgi/wiki?ExtremeProgrammingCorePractices visited January 2012

may help to manage those costs to a degree. Turk et al. concludes that changes with wide impact will have increased cost the later they are detected even in Agile environments.

Thorsen [42] points out that there are fundamental differences between the up-front design expected by IEC 61508 and the Simple Design practice in XP. She admits that a simple design may prevent hazards to occur. She also mentions the prioritized list of resolving hazards:

1. Eliminate the hazard
2. Prevent or minimize the occurrence of the hazard
3. Control the hazard if it occurs
4. Minimize the damage

Early removal of hazards is cheap and effective. For later changes the cheapest and best options may not longer be available.

This shows that adding safety functions late in an XP project when using the Simple Design practice, the cost will increase and the safety will decrease. The Simple Design practice cannot be omitted, as this is an inherent practice of all Agile development models. The team does only know a small subset of the requirements. These are the requirements to be implemented in the current increment, and perhaps some other top priority requirements. This makes it difficult to anticipate and plan for later changes. The Simple Design practice simply states this fact and says that any anticipation in this case will be a waste of time and money, and that the best strategy is to select the simplest design available for the requirements to be implemented.

The Simple Design practice can be dangerous when developing safety related systems, and the practice cannot be omitted. It may only be possible to minimize the effects of this practice, but this would reduce agility. Some initial architecture and design would be beneficial, this is also needed for other reason, see Section 6.1.1. Since implementing safety functions late in development is not advisable, this is an indication that requirements with safety implications should be prioritized and performed as early as possible in the project.

In Section 6.6 it is mentioned that the complexity of a safety related system may cause larger increments. To cope with the added complexity and minimize the effects of the Simple Design practice, the lookahead planning [14] proposed in Section 6.6 could be applied. This would allow the developers an overview of a larger fragment of the system than offered within one increment.

Safety assessment will benefit from a resulting simple design. The impacts of a change are less when the code is well structured. However, the Simple Design practice itself will not result in a simple design, it only selects the simplest design available at a given point. The name of the practice is misleading and leads to misunderstandings, as Paige et al. [30] states this practice as

directly applicable to safety related systems. However, use of the Simple Design practice to evolve a truly simple design implies extensive use of refactoring (the Design improvement practice). This implication is discussed in Section 8.2.8. Without the refactoring, the Simple Design practice only produces unmaintainable code.

The safety process is impacted by this practice when prioritizing the requirements. It could be beneficial to prioritize requirements with high safety implications. When safety functions are implemented early, later developed functions must adapt to the safety functions.

8.2.7 Pair Programming

Pair Programming is a practice within the software development process, and has no consequences for the safety process.

8.2.8 Design Improvement

Refactoring is the art of changing the code structure without altering the code behavior [14]. The result of a refactoring should be better structured, readable and maintainable code. However, it is crucial that this restructuring also maintains safety. Changes due to refactoring should therefore be regarded as any other change; a refactoring must be analyzed for safety implications prior to coding and the result of a refactoring must be assessed and validated to assure that the change has not compromised the safety.

Turk et al. [43] point out that contradicting the common belief that software gets more complicated over time, the continuous redesign is assumed to make the code simpler and maintaining the structural integrity. They say that critical architectural aspects of safety related code can be difficult and expensive to change. For safety related systems it will pay to anticipate such changes early.

Refactoring is beneficial for safety since it simplifies complex code. Paige et al. [30] remark that this may be applicable for autogenerated code that is difficult to understand and assess. They also indicate that due to stable environments, refactoring should rarely be needed when developing safety related systems.

The result from a refactoring is then also wanted for safety related systems, but the safety process adds a large overhead to the refactoring activities. Extensive refactoring is then not practical when developing safety related systems. However, the overhead may be reduced by combining a set of different activities. It is possible to add initial design that acts as a guideline for the detailed design and coding. For trivial refactoring it is also possible to add automated static verification, as SPARK

[4]. Static verification combined with regression tests should reduce the need for reassessment and be a supplement to revalidation.

The recommendation is that main rule should be that every refactoring is analyzed for safety implications prior to coding. This means that every refactoring must be done as separate activities, and not implemented simultaneously with the feature where the need for refactoring was discovered. This limitation will essentially obstruct all small refactorings that are normally done along with the normal work with little extra cost [39]. It will also make the implementation of the new feature more difficult without the needed refactoring. In these cases it may be sufficient to only assess the refactoring without a prior safety analyze. Refactoring integrated with feature development is cheaper to assess, as the refactoring and the feature are assessed as a whole.

Large refactoring or refactoring with wide impact should be given a separate Product Backlog item and thus be subjected to safety analysis prior to development, and safety assessment and safety validation after development. Non-trivial refactoring, but with limited impacts can omit the safety analysis, and be subjected to an immediately following safety assessment to assure that possible safety implications are handled. If the safety implications are not sufficiently handled, the feedback to the development team is given in form of a new Product Backlog item. Trivial refactoring can be handled by using tool support and static verification. Evaluation of which types of refactoring that requires a certain safety handling is done when planning the safety process.

8.2.9 Continuous Integration

The Continuous Integration is a practice within the software development process, and has no consequences for the safety process. Safety assessment and validation will be based on official baselines of the software that do not change while assessing or validating.

8.2.10 Collective Code Ownership

Collective Code Ownership is a practice within the software development process, and has no consequences for the safety process.

8.2.11 Coding Standard

The Coding Standard is also input to safety assessment. The software should be assessed to be compliant with the coding standard. The Coding Standard should also be a subject for assessment early in the project.

8.2.12 Metaphor

A Metaphor is described as a common understanding of the system. Paige et al. [30] state that simple metaphors seem too vague to be of any value for safety related systems. For safety related systems, metaphors need to show different views into the system, including a safety model. Paige et al. indicates that these metaphors are not simple enough for XP use.

In Section 6.1.1, it is argued that safety analysis requires an amount of initial work. It is also suggested to use workshops for making the initial work more agile and to create a common understanding of the system and the safety aspects. These activities create a common understanding, but is not the simple metaphor of XP. The complexity of a safety related system and especially the safety aspects cannot be captured in a simple metaphor. The focus should therefore be on the "common understanding" part, and not on the "simple" part of this practice.

8.2.13 Sustainable Pace

Sustainable Pace (also called 40 hour week) is a practice within the software development process, and has no consequences for the safety process.

8.2.14 Cross functional teams

Both Scrum and XP use cross-functional teams. That means that all skills necessary to perform the job should be present among the members in the team. Within a team, the members will also over time obtain knowledge outside their initial expertise. The use of cross-functional teams reduces the need of documentation and handoffs [14].

The benefits from cross-functional team could lead to that people with safety assessment skills and safety validation skills are integrated in the development team. As discussed before, in Section 6.6 the required independence of these safety activities inhibits such integration.

8.3 Communication and documentation

One of the Agile principles says *The most efficient and effective method of conveying information to and within a development team is face-to-face communication.* The Agile manifesto states that it prefers *Working software over comprehensive documentation*. All Agile development models focus heavily on face-to-face communication, and reduce the focus on written documentation. Written

documentation in Agile projects that can be expected to be up to date at any point are the customer tests, the unit tests and the code.

The safety process, as described in IEC 61508 [20] requires written documentation. However, face-to-face communication is not ruled out, as it is stated that the assessment team shall have access to any person involved in the system development. This section discusses Agile communication, focusing on communication between the development team and the safety teams within an Agile context.

8.3.1 Face-to-face communication

Face-to-face communication is the preferred form of communication to and within Agile teams and are postulated as one of the Agile principles [2]. This section investigates whether face-to-face communication also is the best way of communicating between the safety teams and the development teams.

8.3.1.1 Safety Analysis

The safety analysis phase contains the activities concept, overall scope definition, hazard identification and establishing the safety requirement from IEC 61508. These activities cannot be separated from other top level design activities, and should not be done by the safety team alone. Involving representatives from the development teams in these activities allows face-to-face communications. However, safety assessment and certification require that the premises for and the results from the hazard and risk analysis, as well as the safety requirements, must be documented in writing.

Some techniques used for hazard identification, for example HAZOP, are inherently face-to-face. In Section 7.2.1 it is recommended using workshops for performing safety analysis and to anchor the safety requirements in the development teams. The safety functions requirements are written down and included in the Product Backlog as any functional requirements. When a safety requirement is about to be implemented, the person responsible for the safety requirements is available for clarifications and discussions.

Safety analysis will benefit from face-to-face communication with the development team. The scope, hazards and safety requirements must be written. The result from safety analysis is basis for the system development, as well as safety assessment and safety validation. These documents need to be accessed from different teams at any time over a longer period of time, and must be consistent. The safety analysis results and their premises must then be documented.

8.3.1.2 Safety assessment

Turk et al. [43] says that it is assumed that face-to-face communication is the most productive method of communicating. They challenge the valuation of tacit knowledge over externalized knowledge, and says it would lead to corporate memory loss, and reduce the possibility to learn from collective experience. They also mentions that face-to-face communication make it difficult to track information.

Using face-to-face communication as the main communication between the development team and the safety assessment team is not possible. The main communication prior to the assessment is from the development team to the assessment team. During assessment, there may be need for clarifications. After assessment, the main communication is from the assessment team to the development team. Face-to-face communication are bi-directional, not uni-directional. It is only the clarifications that would involve bidirectional communication and thus true face-to-face communication.

The level of independence required for those who should perform safety assessment is one reason for not including the safety assessors in the development team (Section 6.6.2). Using face-to-face communication as the main communication with the development team would then compromise this independence. It would also involve the development team in the assessments, at the expense of their development tasks. Face-to-face communication cannot provide the complete foundation that safety assessment needs. It is not consistent over time and cannot be presented to certification authorities. If the assessors obtain the assessment basis mainly from face-to-face communication, they would be forced to document the information, to be able to include it in the safety argumentation. This would just move the cost of documentation from one team to another, and will give less accurate documents retrieved from second hand information.

The assessment results reported to the development team are mainly changes to the system needed to pass the assessment. These changes must be precisely recorded and tracked, and appear in the Product Backlog or similar.

The main communication between the development team and the safety assessment team should then be written documentation. But this does not exclude using face-to-face communication. Clarifications during assessment will involve face-to-face communication between the assessor and the developer. However, prior to assessment, a presentation of the functions developed may accompany the written documents. The assessment findings could be presented to and clarified with the development team. This would allow some agility, i.e. reduce the information loss of the handoff between the assessment and the development teams.

8.3.2 Documentation

The previous section (8.3.1) states that written documentation is necessary when developing safety related software. Face-to-face communication is to be used as a supplement to regain some agility. However, the Agile community sees developing extensive, complete and consistent documentation and software models as counterproductive [43].

An Agile assumption is that project visibility can be achieved solely through the delivery of working code. This is a problem for systems that is delivered without human interface, e.g. embedded systems [43]. The visibility of the project is essential for safety assessment. If only the working code is to be assessed, a large part of the functions cannot be seen, since a complete run of all the systems function are not possible within an acceptable time frame, even if everything could be visible through the human interface. Therefore the assessment team needs to inspect more than the running software. The static source code, however, contains more than what is visible through the working code.

Keeping documents up to date within traditional development was difficult and expensive. In an incremental environment, the keeping a document updated will be even more difficult and more expensive. It is not just to document the new feature, but also to update other features that is affected. For documents the unexpected effects of a change cannot be found using an automated test, as for the code. Adding documentation to Agile models means losing agility and adding cost. It is then important, as recommended by the Agile community [3], to have a clear purpose for the documents.

A common misunderstanding is that incremental development includes a mini-waterfall in each increment [40], performing all the activities from the traditional process within the increment [16]. This is regarding Agile as an incremental-sequential model. The activities of analyze, design, implementation and test are clearly done also in Agile development, but the activities are not separated and performed sequential. The iterative practice of Test Driven Development is often used within an increment. The mini-waterfall misunderstanding may also give expectations that incremental development also easily can produce the same documents as sequential development, and the incrementally developed system can be assessed in the same way as a sequentially developed system.

The documents have no longer the primary function to support the software development process. The motivation for the developers to establish and update documents they may consider counterproductive, is probably low. If the quality of the extra documents is reduced compared to the documents from traditional development, the safety may be compromised. If uncritically expecting to develop and assess the same amount of documents as in a traditional safety process, the cost may increase without any significant gain in safety. Extra documents to satisfy safety assessment should

only be established if the cost of establishing and maintaining them are justified by the increased level of safety assessing these documents would provide.

The main purpose of documents when developing safety related systems using Agile models, is to be input to safety assessment and the later certification process. This could cause the developers, maybe unconsciously, selecting just what they think the assessors need to know, or selecting what they think is needed to pass safety assessment. This shift in document purpose, is a potential risk to safety. How to state the purpose and how to incorporate necessary documents in Agile development is a challenge.

Instead of integrating the documentation in the development increment, Larman and Vodde [27] recommends architectural document workshops after a number of increments, to document how it actually ended up to be. However, this would delay safety assessment and safety assessment feedback will come later in the project, with added cost correcting any safety errors. This occurs because the safety process depends on these documents, whereas for non-safety related systems these documents are meant for maintainability and future use only and no dependencies exist within the project.

However, there are documents in Agile development, the acceptance test reflecting the requirements, the unit tests, reflecting the module design and the source code. These are the primary documents of Agile development. Agile documents are mostly executable and unexpected change impacts are easily detected. The acceptance tests will reflect the updated requirement specification. The Agile documents should be the main subject for assessment. When assessing these documents, it is needed some documentation and traceability to guide into the source code. Such guides may not need the level of detail expected in traditional design documents, and are less impacted to changes. Using extra resources to assess documents that is just developed to be assessed, will not improve safety, but will add cost.

8.4 Summary

Some Agile practices neither affect the safety process or are affected by the safety process. They concern the development process only. Other practices are affected by the safety process to some degree. Representatives for the safety assessment team should attend the Daily Sprint and the Sprint Retrospective to be able to assess how the development process is carried out. Representatives for the safety assessment team and the safety validation team can attend the Sprint Review meeting to get an overview of the new functions that will supplement the written documentation.

Safety analysis is closely connected to some of the development process activities. The safety analysts are stakeholders responsible for the safety requirements. The safety analysts will then attend Backlog grooming meetings and Sprint Planning meetings to participate in safety analysis and to be available to the Product Owner and the development team for discussing the safety requirements. The safety analysts also attend the Sprint Review meeting to review the developed safety functions.

The Product Backlog cannot be used as a requirement specification when developing for safety. The Product Backlog is a to-do list, and should continue to be a to-do list also when developing for safety related systems. In addition to the to-do list, a complete and updated requirement specification must be available to be presented to the certifying authorities. Safety must be considered when prioritizing the Product Backlog. Dependencies between functional requirements and safety requirements imply that these requirements should be implemented together. Safety functions should also be implemented as early as possible.

The practice of Simple Design cannot be dismissed when developing safety related systems. It is a consequence of just knowing the requirements for the next increment. In that case the most sensible is at any time to choose to do the simplest design that fulfills the current requirement. This would mean that safety is only considered if it is a safety requirement that is developed. However, when safety functions are not considered early, they can be more difficult and expensive to implement. Late implemented safety functions may also result in a lower degree of safety.

Design Improvement is of course wanted for safety related systems. Agile models improves the design by refactoring. However, when adding the safety process to every refactoring, the cost of doing this will increase and necessary refactoring will be delayed. The need for frequent refactoring can be reduced by using some initial architecture and design, but it cannot be removed. Refactoring is needed to reduce the decay in maintainability of the code. It is necessary to find solutions that allow a reduced safety process for trivial and low-impact refactorings. If not, refactoring will increase cost if it is used, or reduce safety if not used.

Face-to-face communication is important within safety analysis. The safety analysis should also include development team members. The results and premises for the safety analysis must be documented. Face-to-face communication cannot be used as a basis for safety assessment or safety validation. The basis for safety assessment and safety validation must be documented, and will be included in the materials presented to the certifying authorities.

Adding documents to Agile development, also means adding document rework to each Sprint. Every added document presents an increasing cost. The traditional documents that are a basis for

assessment cannot be expected to exist or be of significant value to safety assessment. Establishing and maintaining these document for the sole purpose of assessing them, have a cost that cannot be justified by increased safety. It is suggested that the test documents and the code is the primary assessment objects, using secondary documents as a map into the code.

Chapter 9

Discussion: A more Agile safety process

This chapter summarizes the discussions in the three previous chapters by exemplifying the safety process within a Scrum environment. Each of the three safety process elements, safety analysis, safety assessment and safety validation, are related to the elements of Scrum in the development process that they interact. The safety analysis activities are integrated with the software development process. The safety assessment activity and safety validation activity are run as separate projects due to the requirement of independence. The Scrum framework is also suggested to manage the safety assessment process and the safety validation process, i.e. safety assessment and safety validation are organized as separate Scrum teams with separate Product Backlogs.

Scrum is selected as the reference Agile model in the current chapter because Scrum is a framework that focuses on the process, and not the techniques and methods used within the framework [9].

9.1 Safety process

Figure 9.1 summarizes Chapter 6 in respect to how the safety activities are staged within an incremental development. The activities are referred using the box-numbers from Figures 3.11 and 3.12 in IEC 61508. In addition colors are used to illustrate the different safety activities described in Chapter 6. Note that incremental delivery is not included in Figure 9.1. In case of incremental delivery, the closing activities must be performed for each delivery.

Figure 9.1 also shows the three main phases of Scrum, Pre-game, Game and Post-game [36], although these terms are not referred in recent Scrum documentation. The scope of current Scrum (as described in the Scrum guide [37]) is limited to the Game-activity. However, for large scale

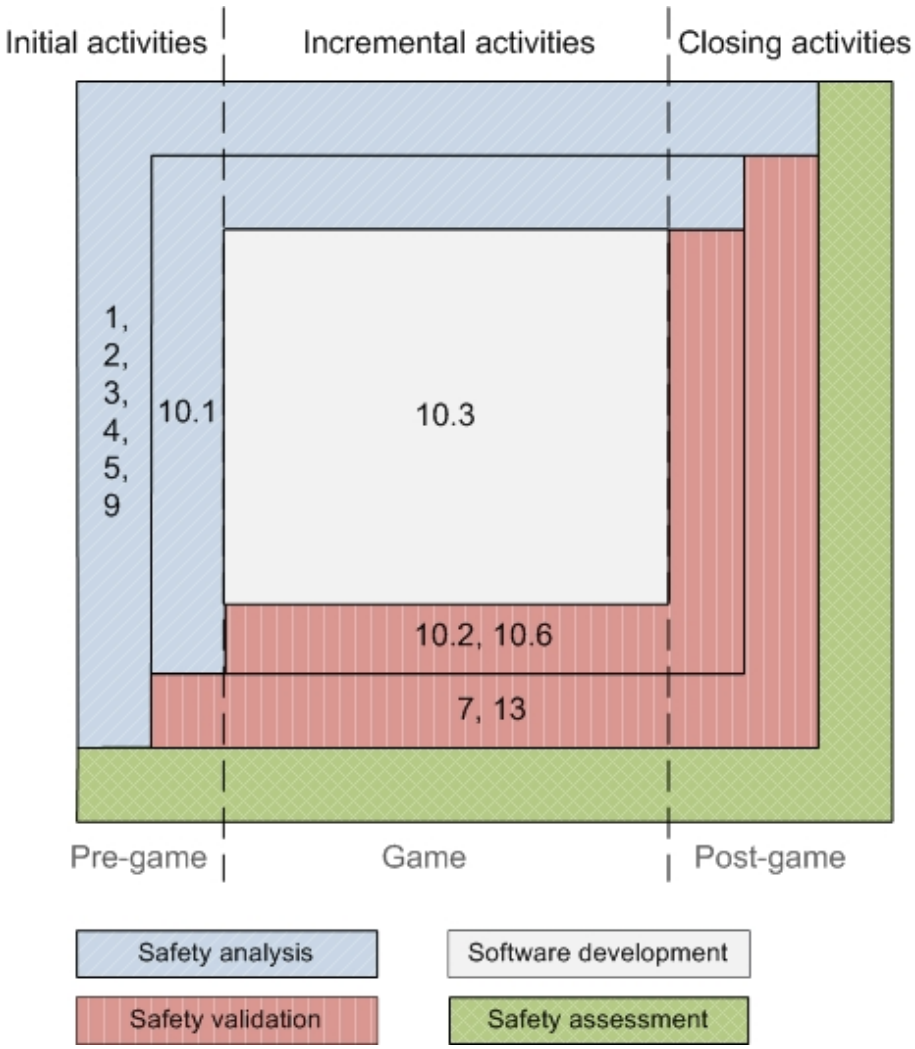


Figure 9.1: Incremental safety process related to IEC 51508

Scrum there are descriptions of how to handle initial work and closing work within Scrum and Agile settings [27].

Compared to Figure 5.2, the Figure 9.1 shows that safety analysis, though still initial, also are done incrementally. The planning of safety assessment and safety validation are not considered as separate activities and are expected to be done along with the incremental activities.

The tailoring of the development process and the safety assessment is not shown on Figure 9.1. These activities are directly dependent of the determined SIL. For each level of SIL, IEC 61508 recommends a set of activities to be used when developing the system and assessing the system to

achieve the requirements for the determined SIL.

9.2 Scrum safety analysis

As discussed in Chapter 6, the safety analysis should be done partly initial and partly incremental. The initial safety analysis is essential to define the SIL that is the basis for how the safety process and the safety activities within the development process are planned and carried out. The incremental safety analysis is done to analyze the changes that appear throughout the development, either new requirements or refining or change of existing requirements. The safety analysis is not a separate safety activity, but is closely interconnected with all development activities that are done prior to coding.

9.2.1 Initial safety analysis

The initial safety activities may be done strictly traditional, strictly Agile or using a combination of these two. The traditional approach ends up with written documents, which are signed off and assessed before starting Agile development. This introduces a handoff between the initial phase and the incremental phase, and communication is mainly done through written documents. Such handoffs represent uncertainties in the form of project delays and information loss and represent a considerable loss of agility.

The Agile models often use workshops. Workshops are not mentioned in the Scrum guide [37] as a part of official Scrum, but Larman and Vodde [27] recommend workshops as a means to cope with the complexity in large Scrum projects.

Combining the traditional way with the Agile way may be more effective. Not all activities fit into a workshop, but the workshops can be used at least at the end of the initial work to enable face-to-face communication in addition to written documents. This face-to-face communication reduces the effects of necessary handoffs.

9.2.2 Incremental safety analysis

Scrum has an activity prior to each increment called the Sprint planning meeting. This meeting is advised to last for about one day for a Sprint of 30 days. The meeting has two parts, *What to do* and *How to do it*. In addition, there is an ongoing activity of grooming the Product Backlog, refining and prioritizing the backlog items. In standard Scrum (one Scrum team only) this work is done in

the Sprint Review meeting [37], but for large scale Scrum a separate grooming activity is suggested [27]. One possibility is to divide the safety analysis between the Backlog grooming and the Sprint planning meeting.

The Product Backlog grooming handles the arrival of new requirements and refinements of existing requirements. The backlog items are prioritized and top priority requirements are refined to a level that is appropriate for development. The safety aspect of this job is to attend to the safety requirements and their links to the related functional requirements. When a requirement is refined, the safety impact of this refinement must be analyzed. If any safety impact, a revised hazard identification and analysis must be performed. New requirements are subjected to their first complete safety analysis.

The Sprint planning meeting first decides which of the top items of the Product Backlog that will fit into a sprint. This discussion must include the safety aspect and assure that requirements are not split into several sprints if the safety would benefit from implementing them together. The second part of the Sprint planning meeting decomposes the selected Product Backlog items into Sprint Backlog items, a form of detailed design of the functions to be implemented. The safety aspect of this activity is to participate in the discussions, assuring that the safety requirements are well understood, and that the Sprint Backlog items reflect the intentions of the original safety requirements. This enables face-to-face communication between the safety analysis team and the development team.

A central question is whether the safety analysis of new and refined requirements can be included in the Sprint planning meeting instead of the separate Product Backlog grooming. This can involve the development team more directly in the safety analysis and give them a better basis for further development. However, as the safety aspect is an important input to the Backlog prioritizing, it must have been completed prior to the Sprint planning, since the prioritized Product Backlog is the input to this meeting. Another aspect is that the Sprint planning meeting must be prolonged significantly if safety analysis is included.

9.3 Scrum safety assessment

As discussed in Chapter 6, the safety assessment should be done partly incremental and partly as a closing activity. However, if using incremental delivery, the weight must be on incremental safety assessment.

There are several options for doing incremental safety assessment. It can be included in the

Sprint itself, the Sprint Review or in a separate assessment-sprint. The closing safety assessment can be done traditionally, or as assessment-sprints.

9.3.1 Safety assessment included in the Sprint

Safety assessment is an activity that is necessary to perform before a new increment is deliverable. This indicates that the safety assessment should be included in Definition of Done, and thus be included in the Sprint. This is not advised, due to the independence requirement of the assessors and the extra length it adds to the Sprint (see Chapter 6.6.2). Safety assessment within the Sprint will also prevent optimal utilization of the resources, creating dependencies and handoffs within the Scrum team. For large project with many teams, the assessors has to be switched between teams to respond to the teams changing needs for assessment.

9.3.2 Safety assessment as a part of Sprint Review

The Sprint Review meeting has a limited duration, 4 hours for a 4 week Sprint is suggested [37]. Safety assessment is a time-consuming and systematic activity and does not fit within the frame of the Sprint Review meeting. The Sprint Review is a meeting using face-to-face communication, and are not suited to perform a formal safety assessment.

The purpose of the Sprint Review meeting is to let the Scrum Team and the stakeholders discuss the results of the Sprint [37]. This review also includes the safety functions. The safety stakeholders are in this case representatives from the safety analysis team. They should be involved in the discussion, deciding whether the safety requirements are implemented as expected. However, they cannot judge the achieved level of safety. This review is no assessment, just a normal verification activity.

The Sprint Review meeting is a step in the software development. The required independence of the safety assessment team will not allow them to be involved in discussions during software development. However, they should attend this meeting as observers to get an overview of the implemented functions. This meeting is a source of information in additional to the written documentation. This also applies for the safety validation team, they should also attend the Sprint Review.

9.3.3 Separate safety assessment sprints

Work not included in the Definition of Done is regarded as Undone Work, and the best way to do this is to use a separate Undone Work Team [27]. The discussion in the previous chapters suggests defining the safety assessment as Undone Work. This implies that a separate safety assessment team can be introduced. This option of using a separate assessment team is selected for the Safer Scrum model in Section 9.5 below.

9.4 Scrum safety validation

Planning the safety validation must be an incremental activity, to be able to handle changing requirements. The validation planning produces the safety validation specification. Performing the safety validation is traditionally a closing activity, some but incremental validation will offer valuable feedback to the development team. The previous discussion suggests that safety validation should be performed by a separate team due to required independence.

9.5 A safer Scrum

The Safer Scrum suggested in the current thesis consists of three parts, the Software development part, the Safety assessment part and the Safety validation part, see Figure 9.2. Functions not passing the assessment or the validation will cause new entries in the Product Backlog.

The Safety assessment team and the Safety validation team are considered as Undone units [26] when seen from the software development team. However, both the Safety assessment and the Safety validation should be run as separate Scrums to benefit from the management abilities of Scrum. This means that the Undone Work lists are considered as an Assessment Backlog and a Verification Backlog.

Figure 9.2 are simplified. It do only show the main direction of information and do not show the feedback from assessment and validation to the software development. The results and premises of an initial safety analysis can also be subjected to a safety assessment, this is not explicitly shown on the figure.

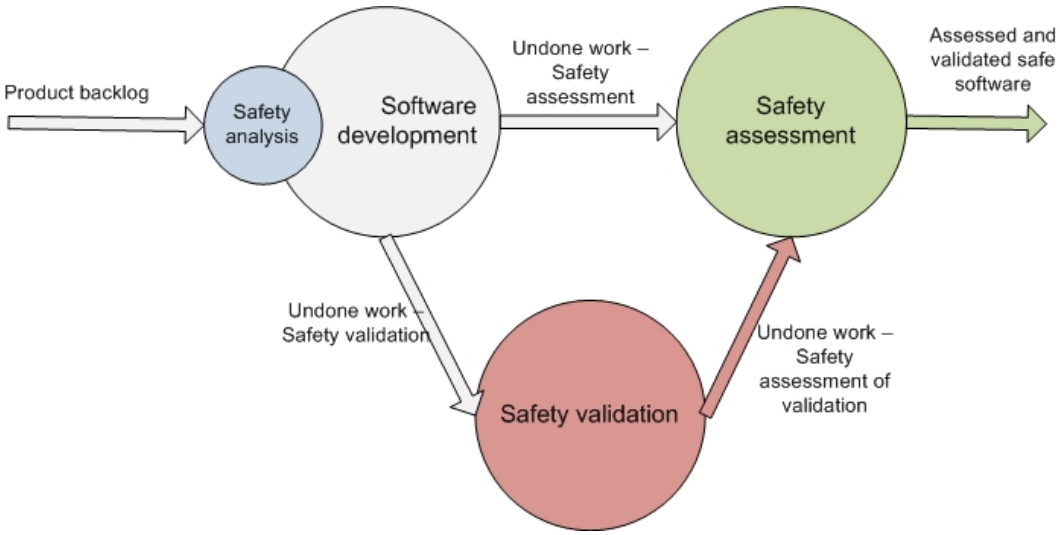


Figure 9.2: The main parts of a Safer Scrum

9.5.1 Safer Scrum development

The software development process and its interactions with the safety process are shown in Figure 9.3. This figure shows that the safety analysis interacts with the development process in the grooming of the Product Backlog, the Sprint planning meeting and the Sprint Review meeting. In addition the safety analysis itself produces new elements in the Product Backlog, i.e. the safety requirements. The safety assessment team should be observers in the Daily Scrum, the Sprint Review and the Sprint Retrospective. The safety validation team should participate as observers in the Sprint Review. The feedback from the safety assessment and safety validation goes through the Product Backlog. The output from the software development process is input to both safety assessment and safety validation.

The elements of the safety process are directly connected to the software development process. The safety analysis is done in connection to the grooming of the backlog. If a backlog item is split, the safety analysis must also split the corresponding safety requirements and decide whether this new level of detail makes more hazards or safety requirements visible. A new or changed requirement must be analyzed, and a removed requirement may have related safety requirement that can or cannot be removed.

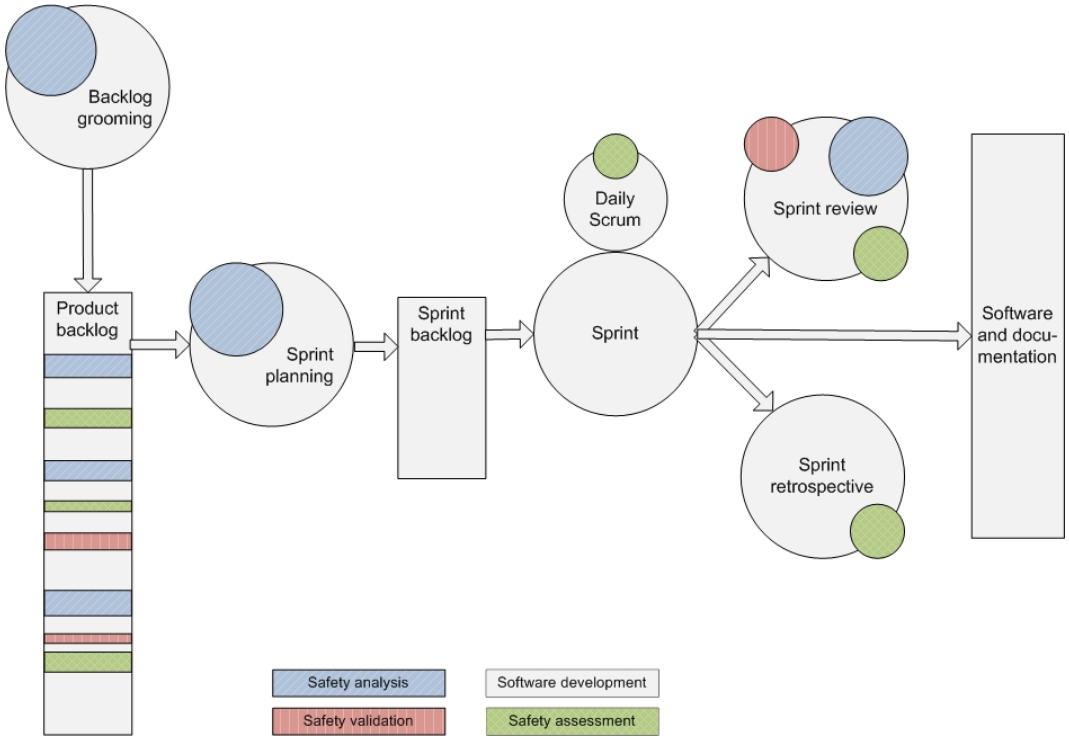


Figure 9.3: The development part of a Safer Scrum

Product Owner should be assisted by safety analysts to address safety considerations while prioritizing the Product Backlog.

Scrum Master The Scrum Master must have an understanding of the restrictions and challenges that arise when developing a safety related system using Scrum.

Development team It is preferable that the development team includes persons with knowledge of developing safety related software and that are familiar with the safety techniques that may be used.

Product Backlog The Product Backlog contains safety requirements from the safety analysis, and feedback from the safety assessment and safety validation. Safety functions should be prioritized as early as possible. This is because when implementing safety functions late, the best options may not longer be available. Safety functions and closely related requirements that will benefit from being developed together should be prioritized as a group.

Grooming the Product Backlog Prior to prioritizing requirements, the safety implications must have been determined through safety analysis.

Sprint Planning The safety analysts has three main tasks in the Sprint Planning. The first is to help the Product Owner and the development team in selecting the safety requirements related to the selected functions. The second task is to communicate the safety requirements and their implications to the development team face-to-face. Finally, the safety analysis team must also be available for discussing the partitioning of the safety requirements into Sprint Backlog items.

Definition of done for the software development team do not include Safety validation and Safety assessment, but contains the necessary basis for assessment and validation.

The Sprint develops the basis for safety assessment and validation according to Definition of Done.

Daily Sprint A safety assessment member should attend the Daily Sprint as often as possible. This is necessary to be able to assess the software development process.

Sprint Review Representatives from the safety analysis team, safety assessment team and the safety validation team participates in the Sprint Review. The safety analysis members are there as the stakeholder with special interest in the implemented safety requirements. The safety analysts are reviewing the safety requirement together with the development team, in the same way the other requirement stakeholders are involved in reviewing their requirements. The safety assessment members and the safety validation members are present to gain information on what is done in this increment, allowing face-to-face communication in addition to the written communication given by the deliverables to this teams.

Sprint Retrospective A safety assessment member should attend the Sprint Retrospective. This is necessary to be able to assess the software development process.

Output from the safety analysis is the safety functions requirements in the Product Backlog. These backlog items are not independent items, but are related to one or more functional requirements and can normally not be developed separately from the associated functional requirements.

The output from the development process is the source code and test documentation, and also the basis for assessment and validation.

9.5.2 Scrum safety assessment

The safety assessment should at least partly be done in parallel to the development to provide valuable feedback as early as possible. When the software development is incremental, this means that the safety assessment will be incremental, following the output from the development process. The current section suggests using Scrum as a framework also for the safety assessment. This approach will both work for incremental assessment and for closing assessment. Scrum safety assessment is not limited to assess software. Anything that can be assessed, can be included in the Assessment Backlog, e.g. system design, hardware and user documentation. One assessment team can assess input from several development teams. Figure 9.4 shows the safety assessment run as a separate Scrum.

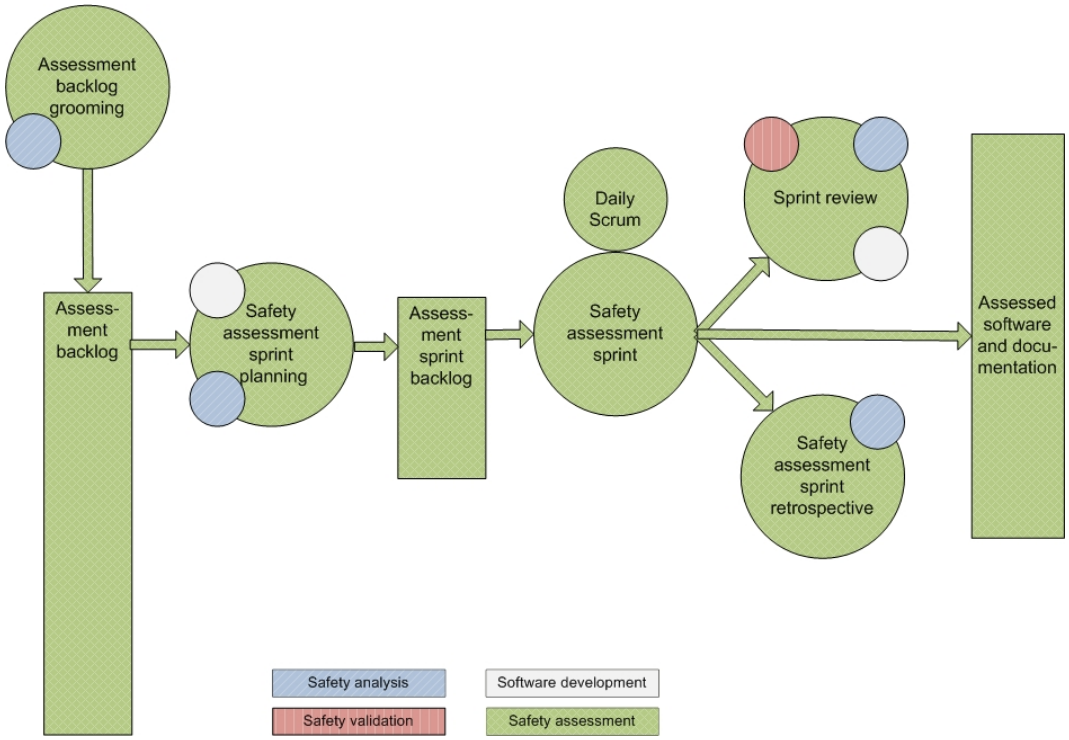


Figure 9.4: The assessment part of a Safer Scrum

The safety assessment Scrum works essentially like a normal Scrum, see Section 3.1.2.3. The

Scrum framework is now just filled with safety assessment instead of software development. However, some elements of the safety assessment Scrum need additional comments:

Assessment Owner The Assessment Owner must be a different person than the Product Owner due to the requirement of independence.

Assessment Backlog The Assessment Backlog will be any input that may be assessed. The Assessment Backlog is a pure to-do list. However, it refers to the updated requirement specification, the code and additional documentation that is always available to the safety assessment team. The assessment team also need to know in what edition of the code and documents this requirement is first implemented. The Assessment Backlog can also contain work from other sources than the software development, e.g. from hardware development.

Assessment Backlog Grooming Representatives for the safety analysis team should be available to answer questions related to the safety requirements. The Assessment Backlog items should also be prioritized. This opens for assessing the items in another order than they were developed. The entries in the Assessment Backlog should be prioritized differently for closing assessment than for incremental assessment. Incremental assessment should be performed in the order that the code is finished to be able to give feedback as early as possible. For closing assessment, the focus is on the complete system, and the Assessment Backlog items should then be ordered to focus on safety.

In case that only one safety assessment team is needed, there is no need for a separate Assessment Backlog Grooming. The prioritizing of the Assessment Backlog Items is then done as the last activity in the Assessment Sprint Review.

Assessment Sprint Planning Representatives for the safety analysis team should be available to answer questions related to the safety requirements. Representatives from the development team should be available to help determining the impact of the change in the incremental part of the assessment. This meeting will decide if a backlog item should be a subject of the safety assessment or not. The guidelines for how this is decided is given by the safety assessment plan.

Assessment Definition of Done The Assessment Definition of Done may be different for incremental assessment and closing assessment. This is decided when planning the safety assessment. The difference will reflect the purposes and the weight of these two assessments. The main purpose of incremental assessment is to give early feedback, and the main purpose of closing assessment

is to judge the achieved level of safety. The weight for each of the two types of assessment will depend on factors such as SIL and project size.

Assessment Sprint Review In this meeting the results of the assessment is presented. Representatives from the safety analysis team are present. They are stakeholders responsible for the safety requirements. They should check that the requirements are understood correctly. Representatives from the development teams and the validation team can be present to get an overview of the assessment output.

Output from the safety assessment team will differ in the incremental part and the closing part. In incremental safety assessment the main output is feedback to the development process. This will be entered in the Product Backlog and will be analyzed and prioritized according to normal rules. It will also be produces an incremental safety case, containing all information from the assessments performed. The output from closing safety assessment will still be some feedback to be handled by the development team. When no more feedback is given, the finished safety case can be presented.

9.5.3 Scrum safety validation

The planning of the safety validation must be done incrementally, to be able to incorporate new and changed requirements in the test plan. Parts of the safety validation should be considered done incrementally, to provide feedback on safety issues as early as possible. Figure 9.5 shows the safety validation run as a separate Scrum.

Safety validation Scrum works essentially like a normal Scrum. The Scrum framework is now just filled with safety validation instead of software development. Safety validation Scrum has the same comments as for Safety assessment Scrum, see Secrion 9.3, with the following exceptions:

Validation Backlog elements need to refer to the updated requirement specification only. The safety assessment team should be able to attend the Validation Scrum Review get an overview of the validation output. The safety assessment team should also be allowed to attend the Validation Daily Scrum and the Validation Scrum Retrospective to be able to assess the safety validation process.

Output from the safety validation team will differ in the incremental part and the closing part. In incremental safety validation the main output is the incrementally growing safety validation specification. There will also be feedback to the development process. This will be entered in the Product

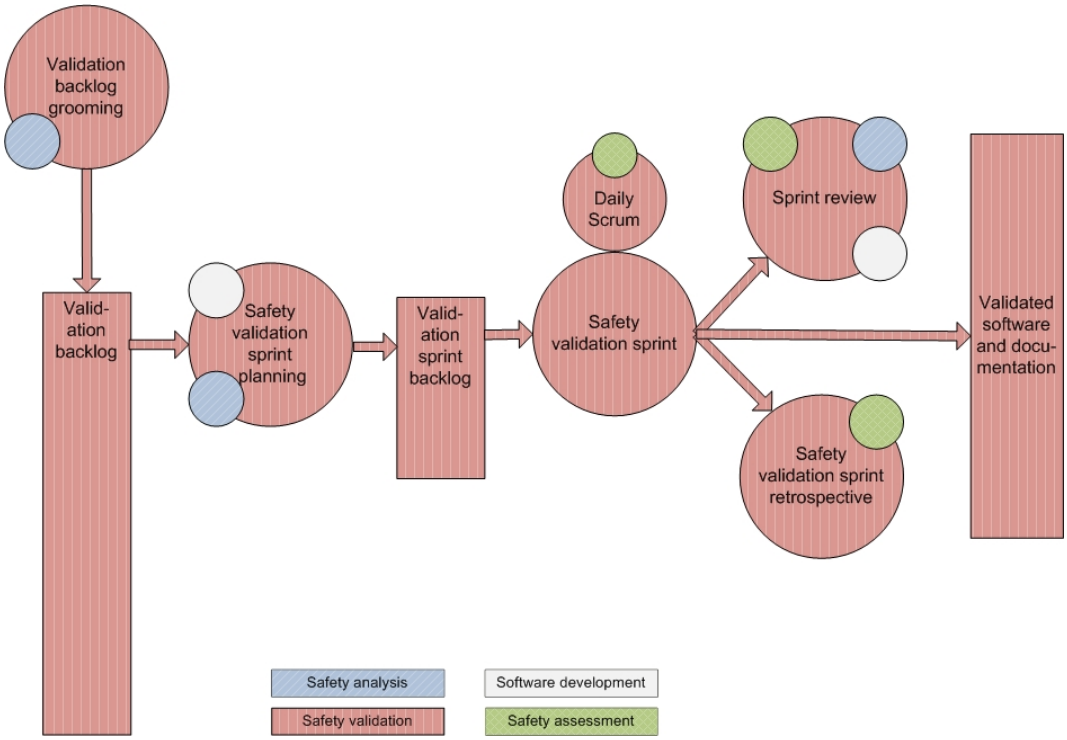


Figure 9.5: The validation part of a Safer Scrum

Backlog and will be analyzed and prioritized according to normal rules. It may also be produced an incremental safety validation log, containing all information from the validations performed. The output from closing safety validation is still some feedback from the validation to be handled by the development team. When no more feedback is given, the finished safety validation log can be presented.

9.5.4 Scaling the Safer Scrum

When defining this Safer Scrum, it is expected that the model should be scaled. The suggestion for scaling the development process is using the frameworks suggested by Larman and Vodde [27] and using feature teams. There are some special considerations concerning the safety process. Safety analysis must be done for all levels of Backlog refinement and Sprint Planning. It must be assured that the work is partitioned and distributed to teams in a way that promotes safety.

The safety assessment process and the safety validation process should be scaled in the same way as the development process. However, the number of assessment teams and validation teams

do not need to match the number of development teams. One assessment team and one validation team can handle input from multiple development teams. Parts of the output from one development team can also be routed into multiple assessment/validation teams if this would benefit the safety or the project economics. The assessment teams and the validation teams can also receive input from hardware development, assessing and validating at software level, hardware level and system level.

9.6 Summary

The current chapter shows the Scrum development process and its connections to the incremental safety process. The safety analysis is included in the development process, mainly in Backlog grooming, Sprint Planning meeting and Sprint Review. The safety assessment and the safety validation is suggested to be run as separate Scrum projects with their own Backlog. These processes have some points of interactions. These interactions are mainly handoffs, and it is shown where information also can be presented face-to-face.

Chapter 10

Conclusions and further work

10.1 Conclusion

Safety related systems are expensive to develop, and have high quality requirements. Agile development promises higher quality at a lower cost, and it is therefore tempting to use Agile development also for safety related systems. The development of a safety related system consists of two main parts, the safety process which includes extra lifecycle activities with focus on safety requirements and their fulfillments, and the development process which includes methods and techniques that are known to promote safety when implementing the safety requirements.

The problem discussed in the current thesis is the safety process within an Agile environment. Two central questions are how Agile models are affected by the safety process, and how the safety process is affected by Agile models. The focus is to maintain the level of safety offered by traditional development, but at the same time maintain most of the agility that is the success of Agile development. Using Agile models must not reduce safety compared to traditionally developed systems. In addition, adapting the safety process to an Agile development model should not make the project more expensive than using a traditional safety process and traditional development.

The method used to answer the questions, is a literature review. The literature reviewed concerns aspects of Agile that may be of relevance for the safety process, such as Agile for large systems, Agile and safety, and Agile and traceability. In addition to this, background literature describing Agile models and the safety process, including the standard IEC 61508, are essential to the discussion.

The terms safety analysis, safety assessment and safety validation are used to simplify the discussion and cover each a group of safety activities that are performed together. Safety analysis covers activities 1, 2, 3, 4, 5, and 9 in Figure 3.11 and 10.1 in Figure 3.12. Safety assessment covers

functional safety assessment, planning of this activity, and development of a safety case. Safety validation covers activities 7 and 13 in Figure 3.11 and 10.6 in 3.12. Some of the main conclusions from the discussions are given below.

Using Agile for developing safety related systems works best for systems with stable requirements and infrequent releases. The reason for this is that some elements from traditional development must be added to Agile models to handle the needs from the safety process. A safety related project with frequently changing requirements, will experience some risks regarding late discovery of safety requirements. The worst case scenario is that the new safety requirement increases the SIL of the system, and thus invalidates all previous work. In any case, late implementation of safety requirements often has wide impacts, and the best solution regarding safety may not longer be available.

A safety related project with frequent releases may have problem assuring safety for each release. Assessment and validation must be done only incrementally and assuring that all impacts are handled for each increment can involve extensive rework and thus considerable cost. If certification is required before delivery, this adds extra cost and delay for each release.

To be able to detect safety requirements early, and establish a credible SIL prior to development start, some initial work is needed. The main purpose of this initial safety analysis is to define the safety requirements to a level where a credible SIL can be defined, and give an overview of the system and the safety requirements. However, if requirements change frequently, the initial work will soon be invalidated.

To be able to assure safety without extensive rework, incremental safety assessment and safety validation could be complemented with a closing assessment and validation. The incremental part can be reduced without insisting to cover all impacts, if the closing activity is be designed to catch the impacts not covered by the incremental activity. However, if frequent releases are required, closing activities represents a delay on every release. Since new requirements arise and existing requirements need to be detailed or changed during development, additional incremental safety analysis is also required for systems with stable requirements. These changes must be analyzed to check for safety implications, as required by IEC 61508.

The main purpose of incremental safety assessment is to give feedback to the development team as early as possible and reduce the risk of discovering unhandled safety issues late in the project. The traditional safety process also includes safety assessment in early phases with feedback to the development process. Safety validation is traditionally done only as a closing activity, and doing it as a closing activity also in an Agile environment will not reduce safety. However, it reduces agility

since feedback from the validation comes too late in the project.

For most safety related systems, incremental delivery cannot be used. This means that the feedback from customers and users that is an important factor in the success of Agile models, is lost. Alternative sources for such feedback must therefore be defined. Incremental safety assessment is one source of feedback, focusing on safety functions. Doing some safety validation incrementally may be another source of "customer" feedback. This may justify some extra cost spent on incremental safety validation.

Agile development models are designed for developing systems with high frequency of requirement change. However, safety related systems with high frequency of requirement change are not well suited for Agile development models, because each new requirement emerging, presents a risk that the system is detected to be of higher criticality than assumed when the project started. A system with higher criticality has more strict requirements on how it should be developed. When a requirement that increases the criticality is detected, it will invalidate the already developed system. If it is detected late, the worst case scenario is that the system or project is terminated.

The main change to the safety process to make it more agile is to make it more incremental. This applies to both safety analysis, safety assessment and to some degree safety validation. This does not mean that safety assessment and safety validation should be included in the development Sprint. These safety activities should still be handled separately due to the requirement of independence. Another change to the safety process is that the content of documentation and traceability necessary for safety related systems should be revised to fit Agile development. An example is that the requirement specification could be contained in the Agile acceptance test specification, giving an implicit traceability link from requirement to test.

The main challenge for an incremental safety process is that incremental work generates a lot of rework to cover all change impacts. An incremental safety process can add more cost than a traditional safety process to assure the same level of safety if the rework problem is not addressed. However, the incremental safety process also offers benefits as safety discrepancies detected and handled earlier. This also gives a more robust safety argumentation.

Agile models need to include some aspects of traditional development to be suitable for developing safety related systems. This includes adding some initial activities and some closing activities. The safety analysis needs an initial activity to be able to define the system's SIL. Safety assessment needs a closing activity to reduce the extent of incremental reassessment. In addition, to be able to perform a safety assessment and present the results to certifying authorities, some documentation and traceability must be added to Agile models. However, this additions should be as small as

possible, since maintaining traceability and documents also implies rework.

Safety functions should be prioritized in the Product Backlog to be implemented as early as possible. This assures that the best options for implementing the safety functions are available, and that the safety functions will be considered for later implemented functions.

The main challenge of adding traditional aspects to Agile development is that agility is lost. This is an indirect cost, since the cost reduction and the quality improvements promised by Agile cannot be fulfilled. One of the main reasons is that the weaknesses of traditional models are brought back, e.g. if the emphasis is on the closing safety activities, the uncertainty of the big-bang integration re-emerges.

The focus when adding traditional activities to Agile should therefore be to avoid or reduce dependencies and information loss from handoffs, and to assure that the development team receives feedback, including safety feedback, throughout the development.

10.2 Further work

The current thesis focuses on the process, not methods and techniques. The process is what to do, and when and where to do it. Methods and techniques are how to do it. However, the current thesis touches areas of methods and techniques that are important to be able to use a more Agile safety process.

10.2.1 How to reduce rework

One issue in the current thesis is that incremental work creates rework. Agile models have reduced their rework concerning testing by introducing automated testing. This means that the complete system can be tested frequently to detect unexpected impacts. However, the additions from the safety process cannot be as easily automated. To be able to do the safety activities incrementally and still maintain agility, the effort used on such rework must be reduced. Developing a safety related system requires a higher assurance that all necessary rework is done than a system with no safety implications. The areas where substantial rework is identified are:

- Updating documentation incrementally
- Traceability maintenance incrementally
- Safety assessment and safety case incrementally
- Safety validation incrementally

The activities are related, for example higher quality traceability gives more precise identification of the extent of the reassessment needed, but the cost of maintaining higher quality traceability should not exceed the reduced cost of the assessment rework. There are research on automated trace capturing [5]. A question would then be if automated trace capturing gives acceptable traceability quality to reduce safety assessment extent and to assure that all impacts are covered.

10.2.2 How to perform Agile safety assessment

It is discussed that safety assessment should be performed incrementally to give feedback, and can be complemented with a closing safety activity to cover all impacts of incremental development. This is the process activities.

The current thesis asked the question if safety assessment in an Agile environment can expect to have access to the same documents as expected in a traditional safety process. It is presumed that this is not necessarily the case. It is also pointed out that the purpose of most of these documents has shifted, from being primary documents in traditional development, to being secondary documents in Agile development. Primary documents meaning documents supporting the development process, and secondary documents meaning documents developed only to serve a need outside the development process, e.g. safety assessment.

It should be investigated how this shift in document purpose will influence their content and in what degree this influence their usability in a safety assessment. It should also be investigated whether the primary Agile documents, the source code and test documents, can be used as the primary sources for safety assessment, and whether there is a need for additional supporting documents to perform an effective safety assessment.

10.2.3 How to perform safe refactoring

Refactoring is an activity that either can be overloaded by the safety process, or not being used even if needed, to avoid this overload. Stable requirements and initial architecture may reduce the need of refactoring when developing safety related systems, but there are still situations where refactoring is needed.

The current thesis mentions use of static analysis and a simplified safety process for some categories of refactoring, and a full safety process for large or complex refactoring. How to define a practical safety process so refactoring can be used also for safety related systems, should be investigated.

References

[1] Agile Alliance. Agile Manifesto. http://www.agilemanifesto.org/, 2001. Visited June 2012.

[2] Agile Alliance. Agile Principles. http://www.agilemanifesto.org/principles.html, 2001. Visited June 2012.

[3] Scott W. Ambler. Best Practices for Agile/Lean Documentation. http://www.agilemodeling.com/essays/agileDocumentationBestPractices.htm, 2009. Visited June 2012.

[4] P. Amey and R. Chapman. Static verification and extreme programming. *ACM SIGAda Ada Letters*, 24(1):4–9, 2004.

[5] H.U. Asuncion and R.N. Taylor. Automated techniques for capturing custom traceability links across heterogeneous artifacts. *Software and Systems Traceability*, pages 129–146, 2012.

[6] K. Beck. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[7] B. Boehm. A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(4):22–42, 1986.

[8] B. Boehm. A view of 20th and 21st century software engineering. *International Conference on Software Engineering*, pages 12–29, 2006.

[9] B.W. Boehm and R. Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Professional, 2003.

[10] S.A. Bohner, R.S. Arnold, and Arnold. *Software change impact analysis*. IEEE Computer Society Press Los Alamitos, Calif, 1996.

[11] O. Cawley, X. Wang, and I. Richardson. Lean/agile software development methodologies in regulated environments–state of the art. *Lean Enterprise Software and Systems*, pages 31–36, 2010.

[12] Jane Cleland-Huang. Traceability in agile projects. In Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman, editors, *Software and Systems Traceability*, pages 265–275. Springer London, 2012. 10.1007/978-1-4471-2239-5_12.

[13] Alistair Cockburn. Incremental Versus Iterative Development. http://alistair.cockburn.us/Incremental+versus+iterative+development, 2007. Visited June 2012.

[14] M. Cohn. *Succeeding with agile: software development using Scrum*. Addison-Wesley Professional, 2009.

[15] A. Elshamy and A. Elssamadisy. Applying Agile to Large Projects: New Agile Software Development Practices for Large Projects. *LECTURE NOTES IN COMPUTER SCIENCE*, 4536:46, 2007.

[16] X. Ge, R.F. Paige, and J.A. McDermid. An iterative approach for development of safety-critical software and safety arguments. In *2010 Agile Conference*, pages 35–43. IEEE, 2010.

[17] Orlena Gotel, Jane Cleland-Huang, Jane Huffman Hayes, Andrea Zisman, Alexander Egyed, Paul Grünbacher, Alex Dekhtyar, Giuliano Antoniol, Jonathan Maletic, and Patrick Mäder. Traceability fundamentals. In Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman, editors, *Software and Systems Traceability*, pages 3–22. Springer London, 2012. 10.1007/978-1-4471-2239-5_1.

[18] Orlena Gotel and Patrick Mäder. Acquiring tool support for traceability. In Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman, editors, *Software and Systems Traceability*, pages 43–68. Springer London, 2012. 10.1007/978-1-4471-2239-5_3.

[19] DR Graham. Incremental development and delivery for large software systems. *Software Prototyping and Evolutionary Development, IEE Colloquium on*, 1992.

[20] IEC. 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, ed 2.0. *International Electrotechnical Commission*, 2010.

[21] Claire Ingram and Steve Riddle. Cost-benefits of traceability. In Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman, editors, *Software and Systems Traceability*, pages 23–42. Springer London, 2012. 10.1007/978-1-4471-2239-5_2.

[22] I. Jacobson, G. Booch, and J. Rumbaugh. *The unified software development process*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.

[23] Ronald E Jeffries. What is XP. http://www.xprogramming.com/xpmag/whatisxp.htm, 2001. Visited January 2012.

[24] TP Kelly. A systematic approach to safety case management. 2004.

[25] TP Kelly and JA McDermid. A systematic approach to safety case maintenance. *Reliability Engineering and System Safety*, 71:271–284, 2001.

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

[27] C. Larman and B. Vodde. *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*. Addison-Wesley Professional, 2010.

[28] Patrick Mäder and Orlena Gotel. Ready-to-use traceability on evolving projects. In Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman, editors, *Software and Systems Traceability*, pages 173–194. Springer London, 2012. 10.1007/978-1-4471-2239-5_8.

[29] R.F. Paige, R. Charalambous, X. Ge, and P.J. Brooke. Towards Agile Engineering of High-Integrity Systems. In *Proceedings of the 27th international conference on Computer Safety, Reliability, and Security*, pages 30–43. Springer, 2008.

[30] R.F. Paige, H. Chivers, J.A. McDermid, and Z.R. Stephenson. High-integrity extreme programming. *Proceedings of the 2005 ACM symposium on Applied computing*, pages 1518–1523, 2005.

[31] R.F. Paige, A. Galloway, R. Charalambous, and X. Ge. High-integrity agile processes for the development of safety critical software. *International Journal of Critical Computer-Based Systems*, 2(2):181–216, 2011.

[32] Jeff Patton. Don't know what I want, but I know how to get it. http://www.agileproductdesign.com/blog/dont_know_what_i_want.html, 2008. Visited June 2012.

[33] F. Redmill. Computer System Development: Problems Experienced in the Use of Incremental Delivery. *SAFECOMP '89*, 1989.

[34] F. Redmill. Incremental delivery-not all plain sailing [software development]. *Software Prototyping and Evolutionary Development, IEE Colloquium on*, page 6, 1992.

[35] W.W. Royce. Managing the Development of Large Software Systems. *Proceedings of IEEE WESCON*, 26(1):9, 1970.

[36] K. Schwaber. SCRUM Development Process. *OOPSLA'95 Workshop on Business Object Design and Implementation*, 1995.

[37] K. Schwaber and J Sutherland. SCRUM Guide. http://www.scrum.org/storage/scrumguides/Scrum%20Guide.pdf, 2011. Visited September 2011.

[38] Mountaingoat Software. An Overview of Scrum for Agile Software Development. http://www.mountaingoatsoftware.com/scrum/overview, 2005. Visited June 2012.

[39] I. Sommerville. *Software Engineering 9*. Pearson, 2011.

[40] T. Stålhane, T. Myklebust, and G. Hanssen. The application of Scrum IEC-61508 certifiable software. Unpublished, 2011.

[41] N.R. Storey. *Safety Critical Computer Systems*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1996.

[42] Liv Ryssdal Thorsen. Extreme Programming in safety-related systems. Master's thesis, Norwegian University of Science and Technology, 2002.

[43] D. Turk, R. France, and B. Rumpe. Assumptions Underlying Agile Software-Development Processes. *Journal of Database Management*, 16(4):62–87, 2005.

[44] Don Wells. Extreme programming: A gentle introduction. http://www.extremeprogramming.org/, 2000. Visited March 2012.

[45] D. West, T. Grant, M. Gerush, and D. D'Silva. Agile development: Mainstream adoption has changed agility. *Forrester Research*, 2010.

[46] K.E. Wiegers. *Software Requirements*. Microsoft Press Redmond, Wash, 2003.

List of Figures

Appendix A

Glossary of Terms

A.1 Definitions

"Big bang" development Developing the complete system in parallel leading to the integration of the total system at the same time. This late integration leads to late discovery of inconsistencies and incompleteness in the system.

Cohesion The dependence between software elements within a module. High cohesion is desirable.

Coupling The dependence between software elements in different modules. Low coupling is desirable.

Definition of Done An agreement between the Product Owner and the Development Team of what "Done" means [27].

Incremental delivery Deliver consecutive versions with limited, but increasing functionality until the system is complete, with the purpose of giving the customer value in form of a working system, and providing valuable feedback to the developers [33].

Incremental development a **staging and scheduling strategy** in which the various parts of the system are developed at different times or rates, and integrated as they are completed [13].

Impact analysis Identifying the potential consequences of a change, or estimating what to be modified to accomplish a change [10]

Iterative development a **rework scheduling strategy** in which time is set aside to revise and improve parts of the system [13].

Handoff A separation of knowledge, responsibility, action, and feedback [14]. Handoffs leads to dependencies, delays, and misunderstandings.

Functional safety assessment investigation, based on evidence, to judge the functional safety achieved by one or more E/E/PE safety-related systems and/or other risk reduction measures [20]

Plan driven development Traditional software development, focusing on plans and architecture

Product Backlog Scrum terminology: The Product Backlog is an ordered list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product [37].

Product Backlog Item Scrum terminology. May also be referred to as a User story (XP terminology).

Refactoring is changing the structure of the code, but not the behavior [14].

Safety is a property of a system that it will not endanger human life or the environment [41]. - freedom from unacceptable risk, [20]

Safety assessment see Functional safety assessment

Safety case A means of documenting the safety justification of a system. [41]
A safety case should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context [24]

Safety lifecycle necessary activities involved in the implementation of safety-related systems, occurring during a period of time that starts at the concept phase of a project and finishes when all of the E/E/PE safety-related systems and other risk reduction measures are no longer available for use [20]

Safety validation see Validation

Safety-related system is one by which the safety of the equipment or plant is assured [41] p.2

Sequential development ... Often used in opposition to iterative development.

Software development process Set of activities

Software development model Theoretical description of a process

Software development methodology Set of methods

Software lifecycle activities occurring during a period of time that starts when software is conceived and ends when the software is permanently decommissioned [20]

Systematic failure failure, related in a deterministic way to a certain cause, which can only be eliminated by a modification of the design or of the manufacturing process, operational procedures, documentation or other relevant factors [20]

Traceability Traceability is the potential to follow a trace link from a source artifact to a target artifact [17]

Traceability decay the gradual disintegration and break down of the traceability on a project [28]

Trace Artifact A traceable unit of data (e.g. a requirement), that qualifies as either the source artifact or the target artifact [17]

Trace Granularity The size of the traceable unit [17]

Trace Link A single association between two trace artifacts, from a source artifact to a target artifact [17]

Trace Maintenance The activities associated with updating pre-existing traces as changes are made to the traced artifacts and the traceability evolves, creating new traces where needed to keep the traceability relevant and up to date [28]

Testing is the process used to verify or validate a system or its components [41]

Undone Work is the work not included in the Definition of Done, but that must be done before the product are shippable. [27]

Validation - the process of determining that a system is appropriate for its purpose [41] - confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use are fulfilled [20]

Verification - the process of determining that a system, or module, meets its specification[41] - confirmation by examination and provision of objective evidence that the requirements have been fulfilled [20]

A.2 Abbreviations

AUP	Agile Unified Process - Instance of UP/RUP
BDUF	Big Design Up Front
E/E/PE	Electrical/electronic/programmable electronic. Based on electrical (E) and/or electronic (E) and/or programmable electronic (PE) technology. From IEC 61508 [20]
EUC	Equipment Under Control. From IEC 61508 [20]
FMEA	Failure Mode and Effect Analysis
HAZOP	HAZard and OPerability study
IID	Iterative and Incremental Development
RE	Requirement Engineering
RT	Requirement Traceability
RUP	Rational Unified Process
TDD	Test Driven Development - Agile development model, derived from the XP practice of Test Driven Development, or it refers to the XP practice itself.
UML	Unified Modeling Language
UP	Unified Process
USDP	Unified Software Development Process
XP	eXtreme Programming - Agile development model
YAGNI	You Ain't Gonna Need It - Agile term justifying the practice of never do any work until it is required