Using a Goal-Based Approach to Improve the IEC 61508-3 Software Safety Standard

Thor Myklebust

Information and Communication Technology SINTEF Trondheim, Norway[1]

Tor Stålhane

Norwegian University of Science and Technology Trondheim, Norway

Børge Haugset

Information and Communication Technology SINTEF Trondheim, Norway

Geir Kjetil Hanssen

Information and Communication Technology SINTEF Trondheim, Norway

Abstract *In this paper we argue that the methods and techniques specified in the annexes in IEC 61508-3 are just sound software engineering principles. Problems when developing safety critical software are not caused by lack of adherence to the standard per se but by ignorance of sound engineering principles related to the specified techniques. Further we argue that IEC 61508-3 should be more flexible regarding the safety lifecycle requirements by mentioning the use of modern software development practices together with the V-model.*

1 Introduction

In our opinion, the most important things needed when making safety critical software is general, sound engineering competence, combined with competence in software development and the application domain, good communication within the development team and mindfulness of safety. It is our opinion that standards, such as IEC 61508-3 (IEC 61508-3:2010), are useful since they list more or less the best techniques and methods available at the time when the standards were written. This is their strong point but also their weak point. The weakness becomes apparent when we want to apply new tools, techniques or methods. Software engineering research has gradually acknowledged the importance of context as a factor in determining success of methods, and that it is impossible to determine which method is best - it all depends on context (Dybå 2013).

Different project development contexts may give rise to entirely different best practices. At that point in time, the standard moves from being helpful – what are the recommended current "best" practices – to becoming a hindrance. For new methods not listed in the standard, the assessor has to be consulted. This complicates the process and is highly dependent on the assessor subjective opinion. There are two ways to get past this problem:

- We can work to change the standard and thus include the new techniques or methods into the standard. This is our current approach, as one of the authors takes part in the maintenance work of IEC 61508-3 that started in autumn 2014, but it is just a stop-gap measure. We will face the same problem again when something new comes along.
- Move towards a goal-oriented standard (IMO.org). This will allow us to use whatever tools, techniques and methods that are appropriate as long as we meet the standard's designated goals

Appealing as the second alternative is, it creates one new problem. A typical statement in a goal-oriented standard could be "The methods for testing and the volume of tests must be sufficient for the defined SIL". The question is who shall decide what is sufficient – the regulators, developers, the customer or the assessor?

[1] thor.myklebust@sintef.no

2 Background

Below we present information related to the main topics of this paper, the software safety standard IEC 61508-3, information related to goal-based standards and agile development of software.

2.1 IEC 61508-3

The lifecycle requirements are presented in chapter 7 in IEC 61508-3:2010. The objectives are to structure the development of the software into defined phases and activities. The requirements are based on the waterfall process methodology although "*any software lifecycle model may be used provided all the objectives and requirements of this clause are met*". The current standard has not succeeded in presenting the requirements as model independent since the requirements are presented according to the waterfall model, including the V-model. This makes it difficult for the manufacturers to use other models although we have indicated that with a little flexibility there are no large obstacles for using agile development models like Scrum for safety critical software (Stålhane et.al. 2012).

Chapter 7.1.2.7 in the standard requires that appropriate techniques and measures shall be used. Techniques or measures are highly recommended for the relevant safety integrity level. If a technique or measure is not used, then the rationale behind not using it should be detailed with having in mind that the large number of factors that affect software systematic capability. It is not possible to give an algorithm for combining the techniques and measures that will be correct for any given application. This should be performed during the safety planning and agreed with the assessor. As a result the standard is in practice a prescriptive standard.

2.2 Goal based standards

In the last two decades there has been an increasing tendency towards a goal-based approach to regulation and standards (requirements for the manufacturers, what they have to do – that includes alternative ways of achieving compliance) compared to the earlier prescriptive regulations and standards (requirements that have to be met if a user wishes to claim compliance with the standard). The reasons behind a goal-based approach are rapid technology changes, new development processes and the legal viewpoint. Too restrictive standards may be viewed as a barrier to trade.

There has been some research on goal-based standards together with the change of some of the safety standards towards a goal-based approach. Already issued goal-based standards are e.g. IMO (International Maritime Organization) standards (IMO.org) and the defence standard 00-56 (DS 00-56). In the research papers (6) – (15), several aspects related to the use of Goal-based standards have been discussed, including: level of argument and evidence, goal-based safety cases and challenges related to COTS equipment (Commercial Off The Shelf). Based on this survey there seems to be little research on the use of new software development methods and goal-based standards. Further, the “Techniques & Measures” in Annex A and B in IEC 61508-3 have not been studied by the research community related to goal-based standards.

2.3 Agile development

Agile software development is a way of organizing the development process, emphasizing direct and frequent communication, frequent deliveries of working software increments, short iterations, active customer engagement throughout the whole development life cycle and change responsiveness rather than change avoidance. This can be seen as a contrast to waterfall-like models, which emphasize thorough and detailed planning, and design upfront and consecutive plan conformance. Several agile methods are in use whereof *extreme programming (XP)* (Beck 2004) and *Scrum* (Schwaber 2001) are the most commonly used for development of non safety critical systems. Figure 2 explains the basic concepts of an agile development model.

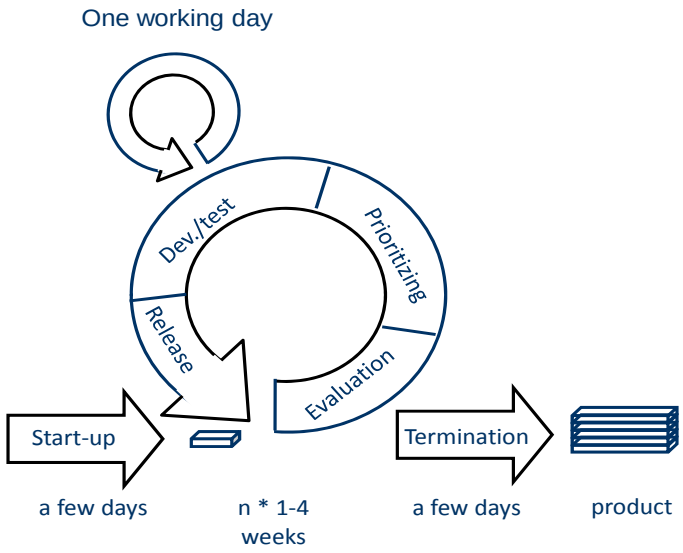

Figure 1: The basic agile software development model

The main constructs of this model are (based on Scrum):

- Initial planning is short and results in a prioritized list of requirements for the system called *the product backlog*. Developers also develop *estimates* per item.
- Development is organized as a series for *sprints* (iterations) that lasts a few weeks. Each sprint should have the same length throughout the project.
- Each sprint starts with a *sprint planning meeting* where the top items from the prioritized product backlog is moved over to the *sprint backlog* – adding up to the amount of resources available for the period. These requirements will be implemented in the following sprint.
- Each working day starts with a *scrum***,** which is a short meeting where each member of the development team explains what she/he did the previous work day, any impediments or problems that need to be solved, and planned work for the work day.
- Each sprint *releases* an *increment* which is a running or demonstrable part of the final system.
- The increment is *demonstrated* for the customer(s) and other stakeholders, who decide which backlog items that have been resolved and which that need further work. Based on the results from the demonstration the next sprint is planned. The product backlog is revised by the customer and is potentially changed / reprioritized. This initiates the sprint-planning meeting for the next sprint.
- When all product backlog items are resolved and / or all available resources are spent the final product are released. Final tests can be run to ensure completeness.

The adaptation of Scrum to development of safety-critical systems, which we call “SafeScrum”, is motivated by the need to make it possible to use methods that are flexible with respect to planning, documentation and specification while still being acceptable to IEC 61508, as well as making Scrum a practically useful approach for developing safety-critical systems. The rest of this section explains the components and concepts of this combined approach.

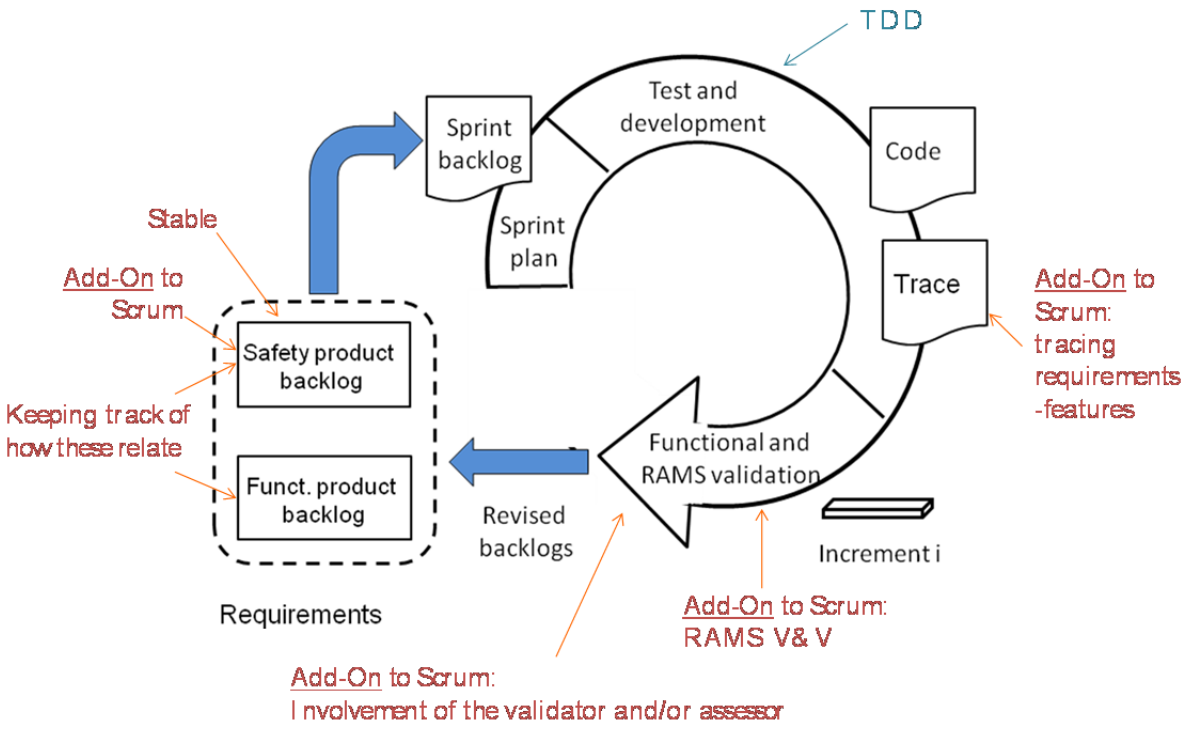

Figure 2: The SafeScrum model. TDD: Test Driven Development

The safety requirements and other requirements are documented as *product backlogs.* A product backlog lists all functional and safety related system requirements, prioritized by importance. The safety requirements are quite stable (relevant regulations and safety standards are normally stable during the project), while the functional requirements can change considerably over time. Development with a high probability of changes to requirements will favor an agile approach.

All risk and safety analyses on the system level are done outside the SafeScrum process, including the analysis needed to decide the SIL level. Software is considered during the initial risk analysis and all later analysis. Just as for testing, safety analysis also improves when it is done iteratively and for small increments – see (Morsicano).

Due to the focus on safety requirements, we use two related product backlogs, one *functional product backlog*, which is typical for Scrum projects, and one *safety product backlog*, to handle safety requirements. We will keep track of how each item in the functional product backlog relates to the items in the safety product backlog, i.e. which safety requirements that are affected by which functional requirements. These two backlogs do not necessarily need to be separated by more than different tags within the same requirement tracking tool.

The core of the Scrum process is the repeated *iterations*. Each iteration is a mini waterfall project or a mini V-model, and consists of planning, development, testing, and verification. For the development of safety critical systems, traceability between system/code and backlog items, both functional requirements and safety requirements, is needed. The documentation and maintenance of trace information is introduced as a separate activity in each sprint – see fig. 3. In order to be performed in an efficient manner, traceability requires, in practice, the use of a supporting tool.

The iteration starts with the selection of the top prioritized items from the product backlog. In the case of SafeScrum, items in the functional product backlog may refer to items in the safety product backlog. The staffing of the *development team* and the duration of the sprint (14 days is common), together with the estimates of each item decides which items that can be selected for development. The selected items constitute the *sprint backlog*, which ideally should not be changed during the sprint. The development phase of the sprint is based on developers selecting items from the sprint backlog, and producing code to address the items.

A sprint should always produce an *increment*, which is a piece of the final system, for example executable code. The sprint ends by demonstrating and validating the outcome to assess whether it meets the items in the sprint backlog. Some items may be found to be completed and can be checked out while others may need further refinement in a later sprint and goes back into the backlog. To make Scrum conform to IEC 61508, we propose that the final validation in each iteration is done both as a validation of the functional requirements and as a RAMS validation, to address specific safety issues. If appropriate, the independent safety validator may take part in this validation for each sprint. If we discover deviations from the relevant standards or confusions, the assessor should be involved as quickly as possible. Running such an iterative and incremental approach means that the development project can be continuously *re-planned* based on the most recent experience with the growing product.

As the final step, when all the sprints are completed, a final RAMS validation will be done. Given that most of the developed system has been incrementally validated during the sprints, we expect the final RAMS validation to be less extensive than when using other development paradigms. This will also help us to reduce the time and cost needed for certification.

The key benefits of this combination of a safety-oriented approach and a process model for agile software development are that the process enables:

- Continuous communication between customers, the development team and the test team(s).
- Re-planning based on the most recent understanding of the requirements and the system under development.
- Mapping of functional and safety requirements.
- Code-requirements traceability.
- Coordination of work and responsibilities between the three key roles; the development team, the customer and the assessor.
- Test-driven development of safety critical systems.

All of these points will help us to get a more visible process and thus better control over the development process.

3 Development of a hypothesis

Approach

The four authors have together participated in several development projects where the IEC 61508 standard has been used, both as developers and assessor. Our current analysis of IEC 61508-3 is based on a thorough analysis of the standard, performed as follows:

- All standard requirements in annexes A and B of IEC 61508-3, pertaining to software that were categorized as HR (highly recommended) for SIL 3 were inserted into a table.
- Since we are discussing common software development practices, all requirements that were only related to the development of safety critical systems are removed as indicated in the column marked with “Safety”.
- For each of the remaining requirements, we inserted our interpretation of the intent of this requirement – see example in table 1 below. Thereafter, we organized them into the traditional categories of software development activities – analysis and design, reuse, coding, validation and verification (V&V) and finally maintenance.
- The list of intents was checked to see if there were intents that were not part of sound engineering practices. Our conclusion is that there was none.

Table 1: Example of requirements walk-through

A9 - Software Verification (IEC 61508-3)			
Requirement	SIL 3	Software development interpretation	Safety
Formal proof	No	-	Yes
Animation of specification and design	No	-	Yes
Static analysis Boundary value analysis • Control and data flow analysis • Design review • Formal inspection	Yes	*Rigour may range from language subset enforcement to mathematical formal analysis.* *Formal inspection is seldom used*	No

A9 - Software Verification (IEC 61508-3)			
Requirement	SIL 3	Software development interpretation	Safety
Dynamic analysis and testing • Boundary value analysis • Structure based testing	Yes	*White box testing. The requirement is supported by the use of tools like Junit (http://junit.org/)*	No
Forward and backward traceability between the software design specification and the software verification	Yes	-	Yes
Offline numerical analysis	Yes	-	Yes

Results

As stated in the start of this section, all IEC 61508–3 requirements are mapped onto standard software development methods and techniques – analysis and design, reuse, coding, validation and verification (V&V) and finally maintenance:

Analysis and design

- Design standard. Will be enforced by a computer-aided design tool.
- Semi-formal methods – e.g. UML (Unified Modelling Language). Sequence diagrams and finite state machines are part of UML.
- Structured method, structured programming. A modular approach, encapsulation, one entry-point – one exit-point and a fully defined interface are all part of structured programming
- Design review. Most projects do a design review. The level of details and effort will, however, vary.

Reuse

- Use of trusted software modules and components. Common practice

Coding

- Strongly typed programming language. Most programming languages are now strongly typed
- Language subset. Most companies use just a language subset.
- Coding standards. Most projects have one. Scrum requires one. Coding standards are available both in books, reports and can be bought.

Validation (testing) and verification – V&V

- Static analysis. Rigor may range from language subset enforcement to mathematical formal analysis. The analysis shall include boundary value analysis, control and data flow analysis and dynamic analysis and testing. This can be done using tools such as QA-C and QA-CPP (www.programmingresearch.com/).
- White box testing and structure based testing. The requirement is supported by the use of tools like Junit.
- Data recording and analysis. Fault records, test logs and software baseline info. Used in all or most testing activities.
- Functional and black box testing. Always used
    - Equivalence class and partitioning testing are important part of black box testing but often not done formally.
- Interface testing. Is part of black box testing
- Test management and automatic tools, requires that coverage targets are defined and met. Coverage targets is an important part of white box testing.

Maintenance

- Impact analysis. Is mostly used by the manufacturers.
- Re-verify changed and affected modules. Is always used.
- Revalidate complete system. Is always used.
- Software configuration management. Common practice.

- Data recording and analysis. Just common sense / diligence. Fault records, test logs and software baseline info. Used in all or most testing activities. This also includes checks for:
    - Completeness of modification
    - Correctness of modification

Conclusion

We have shown that all IEC 61508 – part 3 requirements for software development in annexes A and B are just sound and established software development practice. We do not, however, know which methods and tools that will be available in the (near) future. In addition, it is important to be aware that the standard requirement e.g. boundary value analysis, says nothing on how much, to what level and how extensive it should be.

Thus, it would have been more sensible and practical to replace a large part of annexes A and B with a statement to the effect that the developers should use well tested methods for software development and add a requirement on the developers' knowledge and experience. A good example of how this can be done is tables B2 and B3 in EN 50128 (EN 50128:2011). In these tables the standard defines the responsibilities and key competencies of all roles involved. Some of the key competencies for a developer (implementor) is, for instance, competence in the implementation language and support tools and understanding the relevant parts of EN 50128.

Hypotheses

Based on the discussions above, we offer the following hypotheses:

H1: When the development of safety critical software fails, either by not delivering a product or delivering a product that did not obtain a certificate, this is due to lack of adherence to the IEC 61508 standard and not due to lack of adherence to sound software engineering practices.

H2: Strict adherence to a standard may prevent developers from choosing methods and techniques that are optimal for the situation at hand.

To test the hypotheses, we need documents from failed projects that can be used to gain understanding of how, why and in what respect the project failed. Such documents are for instance post mortem reports (Birk 2002) and other types of project assessment reports. In addition, we should interview relevant project personnel in cases where the project was terminated less than a year ago. Given access to this information, the two hypotheses can be tested as follows:

H1: We will reject H1 if data in more than 90% of the projects reviewed indicate that the project's problems stem from lack of sound engineering practices.

H2: We will reject H2 if data from more than 90% of the projects reviewed indicate that the project has chosen methods that are not optimal for the project due to strict adherence to the standard.

Threats to validity

The hypotheses will be tested based on our interpretation of available documents and our interpretation of interviews, where the available information will be based on project participants' sometimes unreliable memory.

4 Software lifecycle models

The second edition of IEC 61508-3 (IEC 61508-3:2010) has moved towards a more goal-based approach than earlier editions of this standard (IEC 61508-3:1998). The main change was that a new chapter 7.1.2.2 was included, see Table 2 below. According to IEC 61508-3 (IEC 61508-3:2010), any lifecycle may be used provided that all the objectives and requirements are met. Although this is stated in the standard, it presents the requirements as if the v-model is the only model to be used. In the Table 2 below we have presented the current requirements together with suggested future requirements to make the standard Goal-based.

Table 2: Goal-based lifecycle requirements

Cl.	Requirements in IEC 61508-3	Goal-based requirements	Comments
7 7.1	**Software safety lifecycle requirements** General	No change in the Title of the chapter and subchapter	-
7.1.2.1	"A safety lifecycle for the development of software shall be selected and specified during safety planning in accordance with clause 6 of IEC 61508-1.	No change	Our previous paper (Stålhane et.al. 2012) present as an example of an agile safety lifecycle. A company introducing a new method not mentioned in IEC 61508-3 should include their adaptation in their quality/safety system. In addition they should justify the use of the new method as we also show (IEC 61508-3:2010)
7.1.2.2	"Any software lifecycle model may be used provided all the objectives and requirements of this clause are met."	No change	-
7.1.2.3	"Each phase of the software safety lifecycle shall be divided into elementary activities with the scope, inputs and outputs specified for each phase. NOTE See Figures 3, 4 and Table 1."	No change in the requirement, but add two notes: "NOTE 1: A software lifecycle typically includes a requirements phase, development phase, test phase, integration phase, installation phase and a modification phase". Note 2: A software lifecycle in a typical Sprint includes an evaluation phase, prioritizing phase, development and test phase and release phase.	None of the references mentioned in the column "Requirements subclause" in Table 1 is to ch.7.1 for lifecycles. The evaluation of Table 1 is therefore considered outside the scope of this paper.
7.1.2.4	"Provided that the software safety lifecycle satisfies the requirements of table 1, it is acceptable to tailor the V-model (see figure 6) to take account of the safety integrity and the complexity of the project.	Provided that the software safety lifecycle satisfies the requirements in chapter 7, it is acceptable to tailor the model chosen (e.g. V-model or Scrum) to take account of the safety integrity and the complexity of the project.	Two notes mentioned in 7.1.2.4 are not included in column 2
7.1.2.5	"Any customization of the software safety lifecycle shall be justified on the basis of functional safety."	No change	See (Stålhane et.al. 2012) for an example of a sufficient justification
7.1.2.6	**"**Quality and safety assurance procedures shall be integrated into safety lifecycle activities."	No change	-

Cl.	Requirements in IEC 61508-3	Goal-based requirements	Comments
7.1.2.7	"For each lifecycle phase, appropriate techniques and measures shall be used. Annexes A and B provide a guide to the selection of techniques and measures, and references to IEC 61508-6 and IEC 61508-7. IEC 61508-6 and IEC 61508-7 give recommendations on specific techniques to achieve the properties required for systematic safety integrity. Selecting techniques from these recommendations does not by itself guarantee that the required safety integrity will be achieved.	No change	One note mentioned in 7.1.2.7 is not included in column 2
7.1.2.8	"The results of the activities in the software safety lifecycle shall be documented (see clause 5).	No change	One note mentioned in 7.1.2.8 is not included in column 2
7.1.2.9	"If at any phase of the software safety lifecycle, a modification is required pertaining to an earlier lifecycle phase, then an impact analysis shall determine (1) which software modules are impacted, and (2) which earlier safety lifecycle activities shall be repeated.	No change, but add one note: Note 2: If the earlier lifecycle is one of the phases in the sprint, the requirements included should be included in a later sprint For further information on Impact analysis and Agile methods, see (23) – (25).	One note mentioned in 7.1.2.9 is not included in column 2

6 Conclusion

Our conclusion is that IEC 61508-3's handling of software development in its annexes is just a codification of current sound engineering practices for software development. The reason for this is obvious: The standards have been created based on what the committee participants considered important practices and artefacts. In addition, they were created without leaning too much on actual research into each aspect. Even though IEC 61508 has been updated once, it has still been focused on the waterfall process. It could also in principle include items not actually needed as well as a lack of other important aspects or 'other ways of solving the equation'. It described the world as these groups of people saw it at the current time, given their background and knowledge. We believe time is ripe for change from a method-centric to goal-based assessment focus. We believe this could work because these practices together make the stakeholders and developers look more closely into what they are making. They scrutinize all matters, double-check the code and test it thoroughly. However, that does not mean that that particular set of practices is the only way of developing software.

We have analysed the lifecycle requirements in IEC 61508-3 ch.7 and suggested how the requirements could be modified to ensure that the next edition of IEC 61508-3 becomes a goal-based standard. Only ch. 7.1.2.4 has to be changed.

References

[1] IEC 61508-3:2010. Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements. Ed.2

[2] T. Dybå, "Contextualizing empirical evidence", IEEE Software, Vol.30, no 1, pp81-83, Jan.2013

[3] www.imo.org/OurWork/Safety/SafetyTopics/Pages/Goal-BasedStandards.aspx

[4] T. Stålhane, T. Myklebust and G. Hanssen. The application of Safe Scrum to IEC 61508 certifiable software. PSAM11/ESREL 2012. Helsinki June 2012.

[5] Issue 3 of UK Defence Standard (DS) 00-56, published at the end of 2004

[6] McDermid, J. and Rae, A.: Goal-Based Safety Standards: Promises and Pitfalls. In proceedings of Twentieth Safety-Critical Systems Symposium. 2012. Bristol, UK: Springer.

[7] Stensrud E., Skramstad T., Li Jigyue and Xie Jing. Towars Goal-based Software Safety Certification Based on Prescriptive Standards. 2011 First International Workshop on Software Certification.

[8] Becht H. Moving Towars Goal-Based Safety Management. ASSC2011

[9] McDermid J. and Rae A. Goal-Based Safety Standards: Promises and Pitfalls. Achieving Systems Safety 2012, pp 257-270

[10] Coglianese C, Nash J, Olmestad T (2002) Performance-based regulation: prospects and limitations in health, safety and environmental protection. KSG Working paper series No RWP02-050. http/ssrn.com/abstract=392400. Accessed 2014-08-05.

[11] Kelly T, McDermid J and Weaver R. Goal-based safety standards: Oppurtunities and challenges. In Proc 23rd Int system safety eng. Conf, system safety society. San Diego 2005.

[12] Hoppe H. Goal-based standards – A new approach to the international regulation of ship construction

[13] Penny, J., Eaton, A., Bishop, P., Bloomfield, R., "The Practicalities of Goal-Based Safety Regulation", Proc. Ninth Safety-critical Systems Symposium (SSS 01), Bristol, UK, 6-8 Feb, pp. 35-48, New York: Springer, ISBN: 1-85233-411-8, 2001

[14] Melkild E. M. B. Goal and evidence based dependability assessment. NTNU Thesis 2013

[15] Menon C, Hawkins R, and McDermid J. Defence Standard 00-56 Issue 4: Towards Evidence-Based Safety Standards. SSS-2009.

[16] K. Beck and C. Andres: Extreme programming explained: embrace change, 2nd Edition. 2004, Boston: Addison-Wesley Professional

[17] Schwaber, K., Beedle, M.: Agile Software Development with Scrum. 2001, New Jersey: Prentice Hall

[18] R. Morsicano and B. Shoemaker: Tutorial: Agile Methods in Regulated and Safety-Critical Environments. ShoeBar Associates

[19] IEC 61508-3:1998. Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements. Ed.1

[20] Stålhane, T., Hanssen, G., and Myklebust, T.: The Application of SafeScrum to IEC 61508 Certifiable Software – part 1 and 2. Safety Sytems. The SCSC Club Newsletter, vol 23, no 1 and 2.

[21] EN 50128:2011 Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems. Edition 2.

[22] Birk, A., Dingsøyr, T., and Stålhane, T., *Postmortem: Never Leave a Project without It.* IEEE Software, 2002. 19(3): p. 43 - 45.

[23] T. Myklebust, T. Stålhane, G. K. Hanssen and B. Haugset. Change Impact Analysis as required by safety standards, what to do? PSAM 12 Hawaii 2014

[24] T. Stålhane, G. K. Hanssen, T. Myklebust and B. Haugset. Agile Change Impact Analysis of Safety Critical Software. SafeComp, SASSUR 2014

[25] T. Stålhane, V. Katta and T. Myklebust. Change Impact Analysis in Agile Development. EHPG Røros 2014