The Increasing Role of Models in Airbourne Software Development

Continued from Page 8

type checking. Together these stages are often referred to as the 'compiler front-end'. Simulink Code Inspector combines a C compiler front-end with an equivalent process that starts from an executable model. If the source code is structurally equivalent to the model, the two intermediate representations will match exactly and the tool can give a pass/fail indication to replace the manual review. Importantly this check is independent of whatever process was used to write out the source code.

In Conclusion

Models can be used as a design aid to supplement a written specification, such as in the Airbus A380 example. As more of the design is developed as a model, it makes sense for the model itself to become a design document. DO-178C and DO-331 provide a set of guidelines to fully integrate these models into the design process. This improves the development and review of designs, enhances communication between teams or across the supply chain and opens up additional opportunities for automation. This helps engineering teams manage the ever increasing complexity whilst maintaining the highest levels of design assurance.

Mark Walker is a Principal Engineer at MathWorks UK, specialising in code generation and model verification. He can be contacted at: <mark.walker@mathworks.co.uk>

The Application of SafeScrum to IEC 61508 Certifiable Software

by Tor Stålhane, Geir Hanssen and Thor Myklebust

In the previous issue of the newsletter, we presented the agile development method Scrum which, in our opinion, could be used to develop safety-critical software. We also identified some areas where problems will arise. In this second part of the article, we present how we could enhance Scrum to make it possible to use in software development and still be compliant with IEC 61508-3: 2010.

We start this article with a preliminary description of the proposed SafeScrum approach. After this, we list what we expect to be the effects if the industry adopts this approach. Finally we wrap it all up by presenting some ideas on how to move on, to test and improve our ideas for a modernized approach to developing and assessing safety-critical software systems.

Separation of concerns

The IEC 61508 steps needed for developing the environment description and the System Safety Requirements Specification (SSRS) phases 1-4 (concept, overall scope definitions, hazard and risk analysis and overall safety requirements) are kept outside Scrum. The initial requirements of the system that is to be developed are the key input to the second part of the model, which is the Scrum process. The requirements are documented as SSRS project backlog items.

The annexes indicated in Figure 1 are part of IEC 61508. They describe recommended practices to be used during software development. For example, Annex A, Table A.2 describes recommended practices for software architecture design.

The separation of concerns is the main reason why we think SafeScrum is a sound concept. It introduces agility into software development, where theoretical analysis [9] shows that it is useful, and where practical experience [5, 6, 7] shows that it works and leaves the rest of the process intact.

Proposing a New Approach; SafeScrum

Our proposed variant of Scrum, SafeScrum, is motivated by the need to make it possible to use methods that are flexible with respect to planning, documentation and

Continued on Page 10

The Application of SafeScrum to IEC 61508 Certifiable Software

Continued from Page 9

specification while still being acceptable to IEC 61508-3: 2010, as well as making Scrum a practically useful approach for developing safety-critical systems. An overview of the SafeScrum development process is shown in Figure 2.

The rest of this section explains the components and concepts of this combined approach.

All risk and safety analyses at the system level are done outside the SafeScrum process, including the analysis needed to specify the target level of safety integrity (SIL). Software is considered during the initial risk analysis and all later analysis – on a per iteration basis. Just as for testing, safety analysis also improves when it is done iteratively and for small increments – see [1]. There are two types of hazards: (1) hazards that are a natural outcome of the interactions between the design and the outside world, and (2) hazards that come about specifically because of how the item is designed. Hazards of the first category are identified before software development starts, which is why we have separation of concerns – see Figure 1. For the latter category, the authors state that 'We discover most hazards as a system evolves. Hazard mitigations are properly restated as we learn.'

Due to the focus on safety requirements, we propose to use two project backlogs, one *functional project backlog*, which is typical for Scrum projects, and one *safety project backlog*, which is used to handle the safety requirements. Adding a second backlog is an extension of the original Scrum process and is needed to separate the frequently changed functional requirements from the more stable safety requirements. With two backlogs we can 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 are affected by which functional requirements. This can be done by cross-references in the two backlogs and can also be supported with an explanation of how the requirements are related if this is needed to fully understand a requirement.

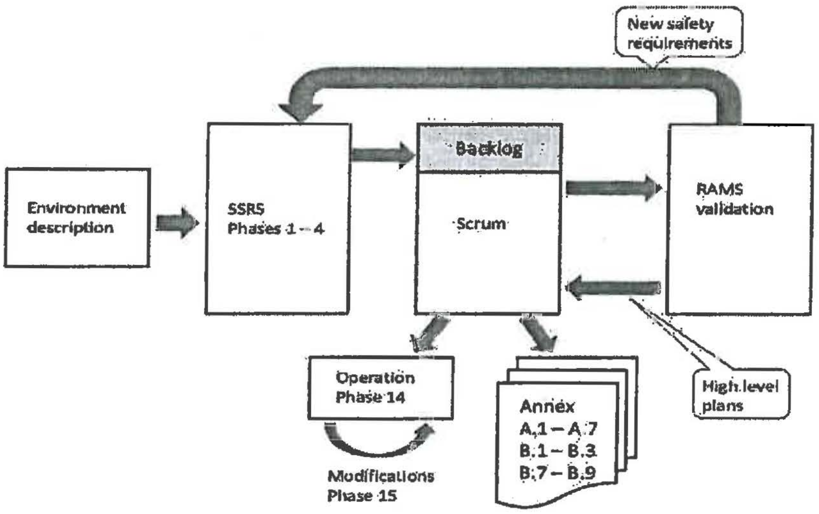

Figure 1: Separation of Concerns

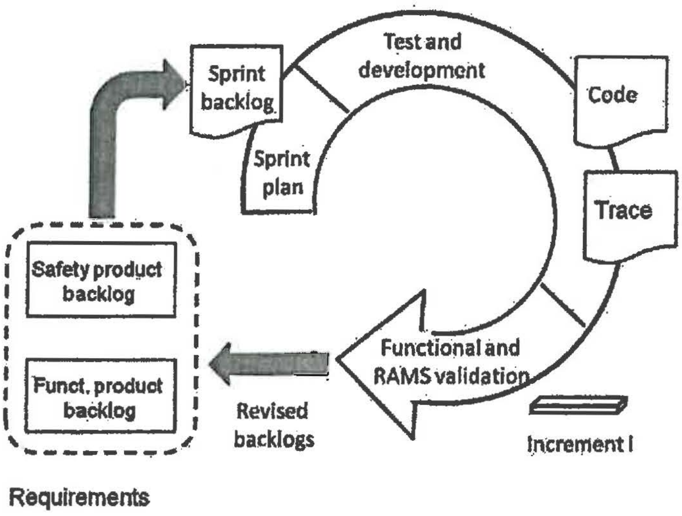

Figure 2: The SafeScrum Model

Continued on Page 11

The Application of SafeScrum to IEC 61508 Certifiable Software

Continued from Page 10

The core of the Scrum process is the repeated *iterations* – sprints in the Scrum terminology. Each iteration consists of planning, development, testing, and verification. For the development of safety-critical systems, we also need traceability between program code and backlog items, both for functional requirements and for safety requirements. The documentation and maintenance of the tracing information is introduced as a separate activity in each sprint. This activity generates the trace documentation – see Figure 2. In order to be performed in an efficient manner, traceability requires the use of a supporting tool. There are several process-support tools that can manage this type of traceability in addition to several other process support functions.

An iteration in Scrum starts with the selection of the top prioritized items from the project backlog. In the case of SafeScrum, items in the functional project backlog may refer to items in the safety project backlog, thus creating requirement interdependencies. The staffing of the *development team* and the duration of the sprint (30 days is common), together with the estimates for each item, decides which items to select for development. The selected items constitute the *sprint backlog*, which ideally should not be changed during the sprint. In the development phase of the sprint, the developers produce code to address the items selected from the sprint backlog.

An important practice in many Scrum projects is *test-driven development*, where the test of the code – usually some kind of unit-test [2] – is defined *before* the code is developed. Initially, this test is simple, but as the code grows, the test is extended to continuously cover the new code. The benefits of test-driven development are that the developer needs to consider the design of the code *before* implementation, it enables regression testing, and it provides low-level documentation of the code which is valuable for later refactoring or extensions of the code [8].

A sprint should always produce an *increment*, which is a piece of the final system. During development this should be executable code, but it may also be user interface mock-ups, database designs, etc., typically in the earliest part of a development project. The sprint ends by demonstrating and validating the outcome to assess whether it satisfies 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 go back into the backlog. To make Scrum conform to IEC 61508, the final validation in each iteration is done both as a validation of the functional requirements and of reliability, availability, maintainability and safety – RAMS – to address safety issues. If appropriate, the person responsible for V&V may take part in the validation of each sprint. He should also take part in the retrospective after each sprint to help the team to keep on focusing on safety considerations. If we discover confusions or deviation from the relevant standards, the assessor should be involved as quickly as possible. Using 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. Between the iterations, it is the duty of the customer or product owner to use the most recent experience to re-prioritize the product backlogs.

In addition to re-planning, applying the RAMS validation process to each increment will also give risk and hazard analyses a gradually evolving scope. This will improve the quality of these analyses. Even if the increments cannot be installed at the customer's site, they can still be tested and run as part of a system simulation. In addition, safety analysis performed on small increments could potentially be more

Continued on Page 12

The Application of SafeScrum to IEC 61508 Certifiable Software

Continued from Page 11

focused and thus give better results.

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.

Potential effects

There are three important goals that we can achieve through introducing Scrum as a process for the development of safety-critical software: (1) achieving the same safety level but reducing the cost, (2) developing software with a higher safety level without increasing the cost, or (3) achieving a shorter time to market. In addition, we will get several side effects such as continuous improvement of the development process, and more product innovations.

The main cost drivers when developing safety-critical software in the traditional way are (1) the extra work that needs to be done in order to ensure that the adopted process is compliant with the required standards, (2) the documents needed to show that we have really done this – hereafter referred to as Proof of Compliance – PoC – and (3) the long tail of error correction in a period when the system development is finished, while we still have a lot of work to do to remove detected errors before the system can be handed over to the customer.

In order to identify the potential effects of introducing Scrum, we need to consider how most of the work is done in a Scrum project. Design, planning and other important activities are done as group processes and worked out and documented on a whiteboard. Using a camera or a smart phone, the relevant whiteboards can be copied and stored for later to be printed out for the assessor. The development of a large amount of formal documents is not a common practice in Scrum. In previous research projects we have seen that some of the groups at Avinor – the Norwegian air traffic authority – use Scrum when developing safety-critical systems. They routinely take snapshots of all whiteboard discussions in the project and keep them as documentation in their development support tool, Jira.

The extra work needed to be compliant with the required standards is handled by introducing the SafeScrum process. In addition, developers find it easier to make necessary changes in an agile setting. The main reason for this is that the unit tests written during development function as a safety net. Our main contribution is, however, the documents needed for PoC and a more efficient process. For documentation, the most important points are:

- What we can reuse from previous projects – e.g., most of a project safety plan will be reused without changes, from project to project;
- What the assessors will accept as PoC for a given activity – e.g., the design process and development of the test plan can be done on a whiteboard and snapshots of this work plus a list of participants will, according to some assessors, be accepted as PoC of the processes;
- Several tools generate, or can be scripted to generate, information that can be used as PoC – e.g., for both unit testing and integration testing.

By cooperating closely and frequently with the assessor, the project will obtain a better understanding of which documents and information the assessors will need in the final certification process. This will reduce the amount of documents that are produced for the assessor only, and will thus reduce the total cost of document production. This solution is in accordance with McDermid and Rae's concept of GoalTopia [3], a paradigm that we heartily support. Replacing 'you shall work this way' with 'you shall use the methods and tools needed to achieve

Continued on Page 13

The Application of SafeScrum to IEC 61508 Certifiable Software

Continued from Page 12

this goal' is, for instance, the dominating concept for government regulations for operations on the Norwegian continental shelf.

It is a common industrial observation that when things change, resilience is more important than adherence to rules and regulations. As Morgan [4] so nicely express it: 'When it comes to thinking, rules are probably the last thing we need for our survival. Rules make us lazy in the way we think. They encourage us to accept the status quo. They stop us thinking outside the rules.' Thus, focusing on goals instead of rules and regulations, are the most efficient way to move forward.

The Scrum process contains a retrospective at the end of each sprint. Thus, we will get a process where inefficiencies and problems are addressed after each sprint, while making sure that the process still conforms to the relevant standards – e.g., IEC 61508-3. In addition, the use of test driven development will lead to more efficient testing and thus to a more efficient verification and validation process.

Another important aspect of Scrum is the ability to work with requirements that are frequently changing, based on the acknowledgement that software development is closer to design than to production. Some of this is outside SafeScrum but Scrum's focus on simple and thus highly maintainable solutions, and the way requirements are handled in Scrum, will make the job easier than when using other development paradigms. Even if the customer's requirements are not changing, the real world is, and if we cannot adapt to these changes we will develop a product that either has out-dated functionality or is using out-dated technology. Changes to requirements implies changes to the plans, and Scrum handles this much more efficiently than traditional, plan-driven development

In addition, Scrum opens up for product owner participation, thus enabling us to correct misunderstandings early in the development process, when they are still inexpensive to correct. This will lead to a shorter time to market, mostly due to less error correction work needed in the aforementioned project tail. This is a fairly common problem, also observed by one of this article's authors. According to one of our industrial partners, reducing the project tail is one of the most important anticipated effects of using Scrum.

All this may sound as just so much promise-ware. However, a quick search on the internet shows that agile development processes are used quite a lot already, also in development of safety-critical software. Some examples are NASA Mission Control Centre [5], Medtronic, a company developing medical equipment [6] and Kugler Maag Cie that develops software for the automotive domain [7].

How to get there?

So far, our ideas on how to adapt and apply Scrum in the development of safety-critical systems, and the benefits we think could follow are based on expert judgements. To move on and to see if these ideas are realistic we need to pilot and adapt the SafeScrum approach in close collaboration with industry *and* assessors. We started this project in August 2013. The project will be run in collaboration with two Norwegian actors: Autronica Fire and Security, and ABB, which will both apply SafeScrum in upcoming development projects. We are also open to other partnerships in relation with this work.

We see several challenges in the work ahead. Firstly, in order to gain industrial experience we need to apply SafeScrum in real projects meaning that we have to introduce uncertainty and an overhead by altering development processes that are already working and well established. Secondly, we need to ensure the independence of the assessor although part of the idea is to have the assessor work closer and more frequently with the development organization. Thirdly, accepting instability

Continued on Page 14

The Application of SafeScrum to IEC 61508 Certifiable Software

Continued from Page 13

and flexibility in requirements for safety-critical systems may create a tension in an industry that seeks control by extensive planning, and conformance to plans. However, to test and improve our ideas we need to put them into use.

References

[1] Morsicato, R. and Shoemaker, B., *Tutorial: Agile Methods in Regulated and Safety-Critical Environments*, ShoeBar Associates.

[2] Koskella, L., *Test Driven*. 2008, Greenwich, UK: Manning.

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

[4] Morgan, M., *Creating Workforce Innovation—Turning Individual Creativity into Organizational Innovation*. 1993: Business & Professional Pub.

[5] C. Webster, N. Shi and I. S. Smith: Delivering Software into NASA's Mission Control Centre Using Agile Development Techniques. IEEE, 2012

[6] van Schooenderwoert, N.: Safety-Critical Applications Built via Agile Discipline. Lean-Agile partners, 2008

[7] Müller, M: Functional Safety, Automotive SPICE® and Agile Methodology at KUGLER MAAG CIE GmbH. Presented at Automotive SPIN, 8th Automotive Software Workshop, 2011.

[8] Erdogmus, H., Morisio, M.: On the Effectiveness of the Test-First Approach to Programming, IEEE Transactions on Software Engineering, vol. 31, no. 3, March 2005.

[9] Huo, M., Verner, J., Zhu, L. and Babar, A.: Software Quality and Agile Methods. Proceedings of COMPSAC'04.

Prof. Tor Stålhane is at the Norwegian University of Science and Technology, Dr. Geir K. Hanssen is a Senior Research Scientist at SINTEF ICT, and Thor Myklebust is an Assessor at SINTEF ICT. The contact for comments on this article is <stalhane@idi.ntnu.no>

Challenges for Assuring Software

By John Clegg

Introduction

Software is becoming more and more prevalent in civil as well as military aviation. Highly reliable programmable hardware (which includes software) is often important for safety. Additionally, programmable hardware is getting more 'authority' and we are becoming more dependent upon it. With increasing system complexity, the full implications of software failures may not be understood and people can still die, even if the software works as designed.

For UK military aviation, it is a requirement that an aircraft is deemed sufficiently safe in its operating role. The key parts of the safety assessment are the identification of the hazards, the determination of their severity and probability, and the identification of the failure modes that contribute to the hazards. Generally, hardware items have random probabilities of failure which can be assessed or estimated and then combined to determine the probability of the hazard occurring. However, software is different in that software failures are not random, but built into software, and will occur whenever the appropriate trigger conditions are met.

This article considers some of the issues with assuring software; these can be broadly categorised as related to 'software failure', 'complexity', 'software modifications' and 'software discipline', and are discussed below. Some of the ways of addressing these issues will be addressed in a subsequent article.

Software Failure

Software failures are due to faults being introduced during the development cycle that are still present in the target code. As stated in Defence Standard 00-55 Issue 2 (Part 2) [1]: *'Software is not subject to failure modes caused by component*

Continued on Page 15