Towards a continuous certification of safety-critical avionics software

Claude Baron, Vincent Louis

Towards a continuous certification of safety-critical avionics software

Abstract - Many industries use safety-critical systems and software, the failure of which may result in the loss of human lives. This article investigates the development and certification of safety-critical software, with a focus on the avionics industry. It highlights the problems encountered in companies to demonstrate compliance with the certification requirements and indicates current industrial practices. It demonstrates the interest and importance of closely and continuously integrating certification requirements in the software development process. It underlines a very recent trend in industry that consists in taking inspiration from agile principles in order to ensure that certification requirements applicable to software development are met as early as possible. It presents some successful industrial experiments and concludes on practical lessons that could be transferred to other projects.

Keywords – Agile development, development process, certification standards, avionics, safety-critical software engineering

I. INTRODUCTION

In avionics many systems are described as "safety-critical". The criticality[1] of a system is based on the consequences of its failure, and the risk of this failure leading to loss of equipment or human lives. When an aircraft function is deemed safety-critical, an authority such as the European Aviation Safety Agency (EASA) for civilian aircraft or the DGA[2] for French military and government state aircraft, generally requires that the methodology used for the system development process has been proven and recognized as acceptable. This authority carries out one or more audits, either directly or through another body, to ensure that the industrial development process complies with the objectives specified in the recognized and applicable standards, following a certification process. This process consists in fulfilling the regulatory requirements recommended for each type of aircraft and submitting the proof to a certification authority. Systems and equipment, including embedded software, must be approved in order to be accepted for certification. The certification authority's approval depends on the success of the product lifecycle demonstration or test.

The certification process is essential in regulated safety-critical fields. In avionics, it is mandatory. Its effectiveness has been demonstrated by a constant reduction of fatal accidents through the years despite the growth in air traffic. The several independent assessments are of utmost importance to ensure to the system environment and its future users that the level of trust is in line with the intended use.

Standards defining technical activities and processes have been written to help detecting errors as early in the development process as possible. However, the implementation of such standards guidelines by industry is often seen as extremely time-consuming and costly. Therefore, certification is often seen as a constraint, resulting in further activities that are deemed to be superfluous and which generate cost overruns, such as drafting the software specifications, carrying out reviews (of requirements, procedures and test results), and managing the configuration of all engineering elements. In reality, manufacturers very rarely carry out these activities during the prototyping stage. They are often pushed back and conducted shortly before certification audits. Certification does indeed have a cost, because it requires additional activities on top of the standard development process, but

[1] The criticality (C) of functions provided by a system is determined with respect to the failure conditions in relation to those functions. It is calculated by taking the occurrence (O) of the failure conditions, their severity (S) and their detectability (D): C = O * S * D [DOD MIL-HDBK-338B 1998].

[2] DGA (Direction Générale de l'Armement) means General Direction of Armed Forces.

"doing the certification" at the last minute, after prototyping, carries a much greater risk of additional costs due to retrofitting, or reverse engineering. As many certification audits fail because the objectives are not met, additional audits must be regularly carried out, which leads to postponing certification. A recent illustration of this behavior could be still found during the development of critical avionics software for a military UAV, where, despite 10 successive audits, the supplier was still unable to demonstrate compliance with certification requirements; this resulted in significant cost overruns.

Having observed the current state of these practices, the U.S. Congress has encouraged the American certification authority, the Federal Aviation Administration, to reduce the number of constraints in order to lower costs. This, however, may come at the expense of safety. An alternative option would be to try and facilitate certification, by carrying out certification activities throughout the development stages. This would contribute to minimize the effort and the financial impact. Furthermore, the capacity to continuously provide proof of compliance would not only make the process more efficient.

This article discusses the challenges faced by the industry during the development of safety-critical software, with a focus on the avionics industry. It is based on the analysis of a large body of data and experience, hitherto unpublished, resulting from multiple audits led by the DGA, as certification authority with more than 30 years of proven expertise in this field. This field analysis is complemented by an extensive review of the literature on agile software development for safety-critical software and related avionics regulations. Thank to this, the article highlights the problems companies often encountered when demonstrating compliance with the certification requirements. It also addresses the current state of industrial practices, the upcoming challenges and the associated necessary improvements; it highlights some successful projects.

Overall, the paper is a reasoned opinion paper. It provides arguments in favor of a better integration of certification requirements into the development process. It demonstrates the interest and importance of closely and continuously integrating certification requirements in the software development process and underlines a very recent trend in industry that consists in taking inspiration from agile principles in order to ensure that software development certification requirements are met as early as possible. It also notes practical lessons that could be transferred to other projects.

Section II sets out the background to this study, introduces the matter of software certification in the aeronautical industry. Section III explains how certification objectives and requirements affect the software development process. Section IV provides a feedback and an analysis of current postures and practices in industry with respect to certification activities. Section V gives an overview on the industrial trend to implement continuous development. Section VI proposes to extend this trend with continuous certification, by introducing agility into the development process of safety-critical software; it also discusses some first industrial experiments that initiate this trend. The article concludes by highlighting the need to help practices evolve.

II. SOFTWARE CERTIFICATION PROCESS AND STANDARDS

After a reminder on safety assessment, this section explains how to ensure, thanks to certification, that industrial systems comply with current regulations and aeronautical standards.

A. Safety assessment

Safety uses systems theory and systems engineering to prevent foreseeable accidents and minimize the consequences of unforeseeable accidents. It takes into account the loss of human life (or injuries), the destruction of assets, mission failures and environmental damages [Leveson 2003]. Safety is a planned, disciplined and systematic strategy for identifying, analyzing, evaluating, eliminating and controlling hazards throughout the system's life cycle in order to prevent or reduce the number of accidents.

Safety standards are guidelines edited by regulation authorities to determine if the product will perform reliably in its operational context. They

recommend a number of stages, deliverable documents and output criteria focusing on planning, analysis and design, implementation, verification and validation, configuration management and quality assurance for the development of a safety-critical system [Rempel 2014]. Furthermore, they generally outline expectations for the creation and use of traceability in a project. Safety-related activities begin in the very first stages of the project's concept development, and continue throughout the design, production, testing and deployment and decommissioning stages.

Manufacturers use various strategies to ensure a high level of safety. However, analysis techniques rely solely on the skills and expertise of the safety engineers. The most common conventional strategies to ensure safety are failure mode effects and safety-criticality analysis (FMECA) [Leveson 2004] and fault tree analysis (FTA) [Wessiani 2018]. They are now being challenged by the introduction of new technologies and the growing complexity of the systems we want to build. Exhaustive testing of a complex system with a lot of integrated critical software is all but impossible as the time taken to gain a credible estimate of its failure rate is excessive except for systems with the lower levels of safety integrity requirements. To gain confidence in the safety of a software-based system both the product (the system) and the process of its development need to be assessed. The use of models and automation for certain parts of the safety analysis reduces costs and improves the quality of the analyses [Braun 2009].

B. Software certification in aeronautics

After defining the regulatory objectives and requirements, regulation authorities often suggest acceptable means of compliance for each regulatory requirement. These are recognised techniques that enable safety objectives to be met. Then industry stakeholders (manufacturers and authorities) produced guidelines (standards) to meet the requirements and develop systems and software in line with regulations. Systems and components, whether separately or interconnected, must be designed so that the occurrence of catastrophic failures that reduce flight or landing safety is "extremely improbable" and is not due to a single failure; this is known as a fail-safe design concept [Gario 2018].

The authority must validate that the methods chosen by manufacturers to fulfil regulatory objectives comply with the fundamental aspects required for certification. Correct application of an engineering process is the only way to ensure that the product fulfils safety objectives. Audits are a way of verifying the technical content produced by the processes implemented. Although the audit cannot be exhaustive, as it focuses on a sample of engineering data, this random method is deemed satisfactory. During this type of exercise, applicants (certification candidates) need to demonstrate their ability to design software, overcome problems and size the resources in order to meet all regulatory objectives. These objectives are determined by the software's criticality level, which is based on a system analysis that identifies how the software may contribute to failure condition[3] scenarios.

TABLE I. classifies failure conditions according to the severity of their consequences, on a scale of one to five, with one being 'No Safety Effect' and five being 'Catastrophic'. If the failure condition causes fatalities or incapacities to the crew or multiple fatalities to the passengers, or as "normally causing hull loss" to the aircraft, it is considered catastrophic.

TABLE I. RELATIONSHIP BETWEEN SEVERITY OF EFFECTS AND CLASSIFICATION OF FAILURE CONDITIONS

Severity of the Effects						
	Effect on Aeroplane	No effect on operational capabilities or safety	Slight reduction in functional capabilities or safety margins	Significant reduction in functional capabilities or safety margins	Large reduction in functional capabilities or safety margins	Normally with hull loss
	Effect on Occupants excluding Flight Crew	Inconvenience	Physical discomfort	Physical distress, possibly including injuries	Serious or fatal injury to a small number of passengers or cabin crew	Multiple fatalities
	Effect on Flight Crew	No effect on flight crew	Slight increase in workload	Physical discomfort or a significant increase in workload	Physical distress or excessive workload impairs ability to perform tasks	Fatalities or incapacitation
Classification of Failure Conditions		No Safety Effect	Minor	Major	Hazardous	Catastrophic

[3] A condition having an effect on the aircraft and/or its occupants, either direct or consequential, which is caused or contributed to by one or more failures or errors, considering flight phase and relevant adverse operational or environmental conditions or external events (AMC 25.1309 from [RTCA DO-178C 2012].

TABLE II. establishes a relationship between the severity of a failure condition and the probability of its occurrence. If the failure condition is considered catastrophic, then its probability of occurring (acceptable quantitative probability) should be less than 10^{-9} per flight hour and its acceptable qualitative probability should be "extremely improbable". At the aircraft level, which should be capable of withstanding 100 catastrophic failure conditions, and for which 10% of crashes are due to technical failures, it is deemed economically and socially acceptable to lose one plane for every one million flight hours (probability $<10^{-6}$).

TABLE II. RELATIONSHIP BETWEEN PROBABILITY AND SEVERITY OF FAILURE CONDITION

Classification of Failure Conditions	No Safety Effect	Minor	Major	Hazardous	Catastrophic
Allowable Qualitative Probability	No Probability Requirement	<-Probable->	<--Remote-->	Extremely <----------> Remote	Extremely Improbable
Allowable Quantitative Probability: Average Probability per Flight Hour on the Order of:	No Probability Requirement	<----------> $<10^{-3}$ Note 1	<----------> $<10^{-5}$	<----------> $<10^{-7}$	$<10^{-9}$
Note 1: A numerical probability range is provided here as a reference. The applicant is not required to perform a quantitative analysis, nor substantiate by such an analysis, that this numerical criteria has been met for Minor Failure Conditions. Current transport category aeroplane products are regarded as meeting this standard simply by using current commonly-accepted industry practice.					

For software, the aim is to reduce the risk of introducing errors during the development phase. If a latent error is triggered, the deterministic behavior of software will systematically result in a failure. Consequently, the solution is to place constraints on the software engineering process. These constraints are objectives to be complied with as outlined in the DO-178C standard.

C. The DO-178C Standard

Among the guidelines used in avionics, a key standard is the DO-178C (Software Considerations in Airborne Systems and Equipment Certification) [RTCA DO-178C 2012]. It sets out the safety conditions applicable to safety-critical avionics software in commercial aviation and general aviation. In their overview of safety-critical software certification in civil aviation, [Kornecki 2008] highlights that "DO-178 guidelines serve industry well and promote rigor and scrutiny required by highly critical systems".

It is based on four main principles:

- Software is so complex that it is practically impossible to guarantee that it is error-free. Consequently, if the final product cannot be guaranteed, the manner in which it is produced must be as reliable as possible.
- Even if the development process is reliable, errors can occur. Several verification activities should be performed at each step in order to eliminate all potential residual error.
- DO-178C is a document that focuses on controlling three processes to reach technical goals: the development process, the verification process and the configuration management process. No methods or techniques are specified. Only the objectives are specified. The manufacturer decides which method to use to fulfil the objectives.
- Most safety measures are assumed to have been taken at the system level, and quality assurance of the software development should ensure they were correctly implemented.

Software specifications and the way they are produced also play a major role in safety. DO-178C requires that for certain aspects of the development, there must be two separate teams, one which performs the task and one which verifies the task. The applicant must therefore provide proof of this independence by keeping track of all people that perform tasks.

Safety analyses assign a safety-criticality level to each software solution, which reflects the severity of failure conditions that they contribute to. This safety-criticality level is called the Development Assurance Level (DAL) and by applying the DO-178C standard, it indicates the software engineering activities to be carried out to certify the software [ARP4754A 2011].

Software programs are classified into five safety-critical levels, which determine the level of development assurance or DAL (from E to A). The closer this level is to A, the higher the number of DO-178C objectives will be. TABLE III. shows the number of objectives to be met according to DO-178C in terms of development assurance level. If the failure condition is considered to be catastrophic, then the development assurance level of the contributing software will be classified DAL A, and the DO-178C will require it to satisfy 71 objectives.

TABLE III. DO-178C OBJECTIVES VERSUS THE DEVELOPMENT ASSURANCE LEVEL

Failure Condition Severity	No Safety Effect	Minor	Major	Hazardous	Catastrophic
Development Assurance Level (DAL)	E	D	C	B	A
Number of DO-178C Objectives	0	26	62	69	71

Certification actions must be carried out to reach these objectives. These actions must be implemented in the processes required to meet the objectives. For example, one of the development process objectives is to develop high level software requirements to produce the expected output data (software specification and traceability, for instance).

An important point to remember is that aeronautical standards do not impose the means of compliance but rather provide a description of the objectives to be achieved through implementing of a process.

III. CERTIFICATION CONSTRAINTS ON THE DEVELOPMENT PROCESS OF SAFETY-CRITICAL SOFTWARE

This section first reviews the constraints imposed by certification objectives on the conventional software development process in avionics then highlights the importance of the verification process.

A. Certification objectives and requirements

Software engineering processes enable the development of software that corresponds to the customer's needs, is reliable, maintainable and efficient. Fig. 1 shows the typical stages in software development: user requirements are transformed into software requirements, which are then used as a guide to draw up a software architecture, before moving on to detailed design and coding. This is followed by tests for each function, conducted by unit, followed by integration tests, software tests and acceptance tests by the customer. Each test plan is prepared in the downward part of the V-cycle (on the left on the figure).

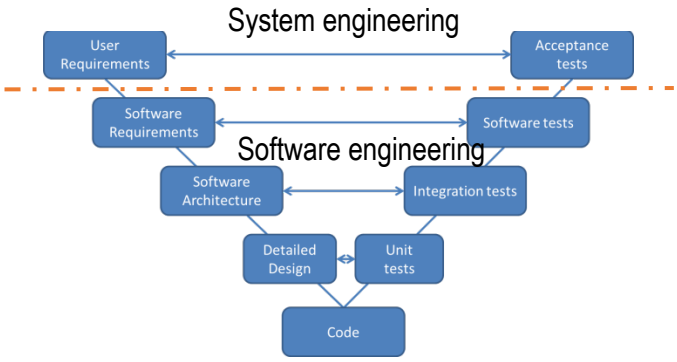


Fig. 1. Standard software life cycle

Aircraft certification using the standard DO-178C does not impose any life cycle requirements but defines separate processes which could be combined to describe the life cycle of a given project:

- Planning process: development, verification and configuration management plans;
- Development process: specifications, design, coding and integration;
- Integral processes: verification, configuration management, quality assurance and coordination for certification.

For each process the following are identified: assurance objectives (for example, defining the architecture and elements enabling coding), the means to satisfy them, entry data (for example, specifications, development plan, design rules), activities (for example, defining the architecture, derived requirements), products (for example, the design description) and transition requirements.

DO-178C also specifies the objectives that must be met to obtain certification. As an example, here are a number of objectives that are mentioned in the standard:

- The software's functions must be systematically specified in a general specifications document based on the system requirements.
- An architectural design and a detailed design will be required for the most safety-critical software.
- Each specification or design element must be developed, precise, coherent, traceable and verifiable.
- The source code will be developed from these elements before being used to generate the executable object code.
- All requirements must be tested. The tests should be based on the requirements to cover the nominal

behavior and robustness test cases and not on the code (requirement based testing [Skokovic 2010]).

- The structural coverage[4] of the source code, obtained by executing these requirement-based tests, must be measured. The structural coverage criteria are modified based on the software's safety-criticality (Statement Coverage, Decision Coverage, Modified Condition/Decision Coverage)[5].
- The source code should be developed in compliance with coding standards.
- Traceability should be established between data items.
- Configuration management must be used to handle engineering data. In some cases, the production of data and its verification must be performed independently.
- Finally, quality assurance activities must be conducted and logged.

Depending on the safety-criticality level, the cyclomatic complexity [Ebert 2016], which represents the number of decisions obtained by studying the algorithm control graph, the structure nesting depth and the number of parameters, will be restricted by increasingly constraining limits. TABLE IV. specifies the constraints to be respected depending on the software category (criticality level). For example, for target category C software, the cyclomatic complexity of algorithms must not be more than 15.

TABLE IV. SOFTWARE QUALITY METRICS IN THE AEROSPACE SECTOR

Quality Charateristics	Metric	Target Category A	Target Category B	Target Category C	Target Category D
Reliability Evidence	Structural code coverage	Coverage => 100% Modified Condition / Decision Coverage =>100%	Decision Coverage => 100%	Coverage => 100% for on board software Statement Coverage => 100% for ground	
Reliability Evidence	Requirement coverage	100%	100%	100%	100%
Maintainability Modularity	Cyclomatic complexity	<10	<12	<15	<=20
Maintainability Modularity	Nesting level	<5	<5	<5	<7
Maintainability Modularity	Number of statements (per functions)	<100	<100	<100	<200
Maintainability Stability	Requirement stability	2%	5%	10%	15%

Measures of the source code can be completed using the concept of "technical debt" [Osetskyi 2018] which evaluates the cost of correcting anomalies to comply with coding standards [NT DGATA 2016]. A supervisory strategy that measures the ratio of technical debt against the cost of the new code [Letouzey 2012] will facilitate the gradual resorption of previous anomalies.

B. *Safety-critical software verification process*

Verification is the most important chapter in DO-178C, in terms of volume (13 pages of descriptions compared to an average of five in other chapters) and in terms of the resulting workload (for the A380, there are four lines of test for every line of embedded code). It is a cross-functional process, as it applies to all the other processes. It recommends a combination of reviews (inspections of a product by an independent body - qualitative analysis), analyses (detailed examination of a product that may be done using a tool - quantitative analysis) and tests (execution of software to compare the results obtained with the results expected - functional tests, functional and structural coverage analyses) to detect and report errors introduced during development.

It is important to note that the standard DO-178C does not distinguish between "validation" and "verification" activities. Both are indiscriminately named "verification". The DO-254 standard [RTCA DO-254 2006] is clearer on this matter. It refers to "validation" as the activity that involves ensuring the requirement under consideration is compliant with and supports the upper level requirement ("Are we building the right product?"). As for verification, this entails making sure the result obtained when executing the implementation meets expectations ("Are we correctly building the product?"). Fig. 2 shows the difference between "validation" and "verification" activities.

[4] Code coverage is a measure used to describe the degree to which the source code of a program is executed when a particular test suite is run. It shows the percentage of the source code that has been tested or not tested. It is expressed as a percentage of the code executed in comparison to the full code.

[5] There are several levels of coverage: Function coverage, statement coverage, condition coverage (Boolean-type logic operators) and decision coverage (logical operators composed of conditions and logical connectors). In addition, modified condition/decision coverage (MC/DC) requires that each condition of the decision be evaluated (at least once) when it affects the final value of the decision.

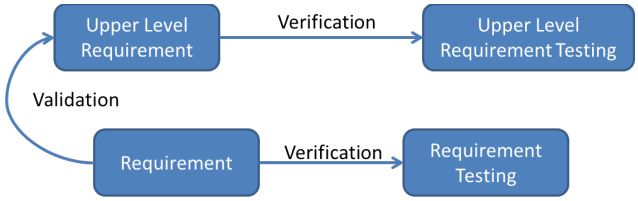


Fig. 2. Difference between validation and verification [RTCA DO-254 2006]

The constraints associated with each engineering level will have an impact on how the company teams are organized. At the project level, the constraints will be adapted to conduct the required reviews, with or without an independent party. Engineering activities such as instrumentation to measure the structural coverage of the code, verification of the algorithm precision, calculation of the worst-case execution time, creation of robustness and equivalence class testing as well as production of documents will also have to be conducted in an increasingly strict manner.

The independence requirement shows the impact on organizations in developing DAL A or DAL B level software. Two types of independence are considered:

- Independent appraisals or analyses: the person who conducts the appraisal must be different to the person who produced the data;
- Independence between those who carry out the activities: for example, between the person performing the coding and the person who selects the requirement-based test cases.

Verification independence requires tasks to be clearly distributed to ensure that the activities or appraisals were correctly conducted by an independent third party. At least three people are needed in a team in order to develop DAL A level software according to verification independence requirements: the developer and/or reviewer of data produced by the auditor, the auditor and/or reviewer of data produced by the developer, and the quality assurance manager.

IV. Analysis of Software Certification Issues for Manufacturers

This section results from the analysis of the DGA (as technical authority) internal surveys, based on more than a hundred industrial certification audits carried out each year, that provided a most interesting practical experience feedback.

One of the DGA's responsibilities is to monitor the compliance of all governmental aircraft systems with regulatory requirements. The DGA also appraises systems used in other various fields, such as space, naval, medical, missiles and drones. Therefore, it has a global view on the industrial practices in the development of safety-critical systems. In this market, certain longstanding manufacturers have been applying the same rigorous processes for decades while developing their safety-critical software, whereas others have just recently implemented their first structured methodology. The DGA has also observed new arrivals (start-ups) in defense markets companies that have never had to demonstrate the reliability of their software.

In the avionics industry, whereas the effectiveness of guidelines is recognized (there has been a continuous drop in the number of air accidents as it appears in the Fig. 2), in practice, implementing certification guidelines is seen by manufacturers to be a costly and time-consuming exercise, requiring actions that are considered to be superfluous in the prototyping phase.

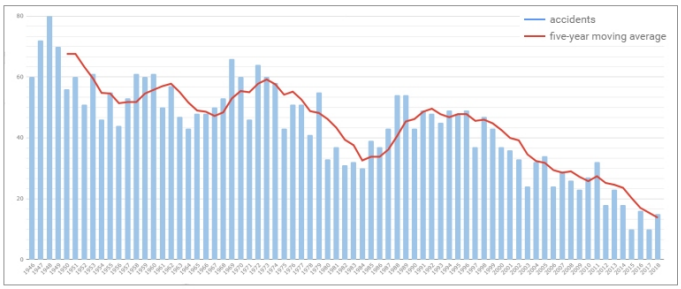


Fig. 3. Fatal accidents in general aviation from 1946 to date (from Aviation Safety Network releases 2018 airliner accident statistics)

Thus, standards guidelines are not always implemented. For example, in the military field, manufacturers rarely apply rigorous, auditable processes when developing safety-critical software. An argument often heard is that the system is made for war, so safety issues are secondary.

However, it is a necessity and a guarantee to third parties that each system can be used with an acceptable risk level and the absence of certification actions can cause disparities that lead to grave consequences. For instance, it would be extremely harmful if a drone crashed in an unsecured zone or a missile strayed from its trajectory, striking an unwanted target.

Based on feedback from the DGA in the field, when addressing certification requirements, they are sometimes dealt with late in the development process by the various stakeholders involved. Certain objectives in these guidelines are often fulfilled only at the end of the development phase, "because it has to be done", to prove the software is compliant.

But this is risky (in terms of compliance) and is likely to be costlier, and more expensive than the expenses incurred for certification. Certification indeed has a cost, because it requires additional actions to be included in the standard development process. But, as shown in Fig. 4, the cost of purely certification-related activities accounts for a mere 3% of the project budget.

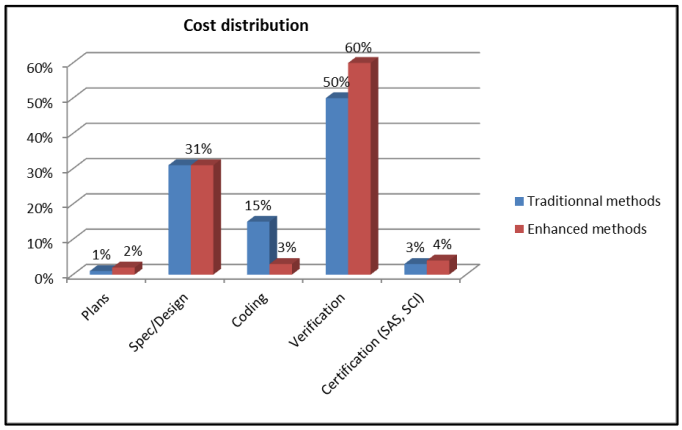


Fig. 4. Cost distribution in software development (DGA Techniques Aéronautiques internal report)

Addressing certification late in the project incurs a higher risk of additional costs. For example, a partially fulfilled structural coverage objective will require costly reverse engineering or additional analyses.

Furthermore, many manufacturing companies poorly estimate the cost of moving from one assurance level to the next. The DAL A is often seen as the "holy grail" of certification that is excessively expensive to attain. In reality, the biggest cost and scheduling differences are between level D and the level above (30% more investment required to go from D to C, 50% to go from D to B and 55% to go from D to A). The DGA's experience and the HighRely study [Hilderman 2009] show that the biggest financial step is between DAL D and DAL C. The cost and scheduling differences to apply the standard DO-178B according to HighRely are shown in Table V.

TABLE V. INCREASE IN COST VERSUS DEVELOPMENT ASSURANCE LEVEL

Level E	Level D	Level C	Level B	Level A
Baseline	E + 5%	D + 30%	C + 15%	B + 5%

The detailed design and tests that are required for DAL C-level software require additional actions whose purpose is to ensure there is no unintended functional behavior. These activities incur additional development costs in comparison to a DAL D-level software. Fig. 5 provides extrapolations of the metrics.

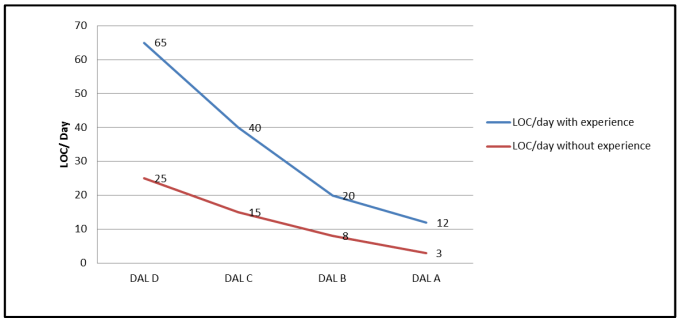


Fig. 5. Number of lines of code (LOC) developed per day versus the development assurance level [Hilderman 2009]

The objectives to be attained to acquire DAL A address error categories specific to dependability (source-code-to-object-code traceability, MC/DC structural coverage). They have no major impact on the development cost.

In conclusion, to improve the software development process while meeting the objectives of certification standards, a better integration of these objectives in the development process is needed in order to help companies facing the different certification issues.

V. SOFTWARE DEVELOPMENT PRACTICES: A CONTINUOUS AND INTEGRATED PROCESS

Software, as well as the teams and deployment infrastructure, are growing increasingly complex. To develop, test and deliver software quickly and consistently, developers and organizations have created strategies to manage and automate these processes. The use of continuous integration mechanisms, and more recently additional practices such as continuous delivery and continuous

deployment, are becoming more widespread in industry [Düllmann 2018].

Continuous integration focuses on integrating the work of individual developers into a primary repository several times a day to quickly detect integration problems and speed up collaborative development. Continuous delivery involves reducing friction in the deployment or publication process, by automating the steps necessary to deploy a version so that the code can be safely published at any time. Continuous deployment goes even further by automatically deploying every time the code is changed.

1) **Continuous integration**

Integration covers all the activities to be carried out once the development is complete, to obtain a functioning "ready-to-use" product. Verifying the consistency between several software components and correcting possible anomalies is part of the integration process. Continuous integration means integrating a component as soon as possible to ensure it is consistent with the other components and that any modifications made do not cause regression, and then generate an operational executable program. It is an essential step for automating all the repetitive tasks in a software development process, enabling certain activities/data required for certification (such as execution of tests and metrics of code coverage) to be executed and produced.

The concept of continuous integration emerged as an objective of project organization in [Royce 1998]. Continuous integration was then made popular with 'extreme programming', a practice that involves developers of the same application reintegrating the code they are working on as frequently as possible, and launching a process with each integration that automatically verifies the application's functioning, so that anomalies are detected on input [Pillou 2018]. [Fowler 2006] describes the elements of this practice: the use of a baseline repository to manage versions of the source code, the automation of the build process, automated unit and function tests and the daily execution of the whole system (build and test). This speeds up the compiling, deployment and coding test phases, thus resulting in productivity gains.

Continuous integration, through the systematic execution of all software tests at each build, renders quality assurance possible thanks to code quality metrics, improved dependency management, early detection of integration errors (due to an omitted inclusion for example, or possible regression of previously implemented functions) and ensure the software complies with standards (naming conventions, programming issues…) applicable to the project. It also allows for faster response times to changes, and the standardization of the application's sources and life cycle.

Continuous integration thus targets two objectives: reducing to a minimum the duration and effort necessary for each integration episode, and the ability to provide a working product at any time. In practice, these objectives require integration to be a procedure that can be reproduced and automated insofar as possible. It includes the execution of a battery of unit tests and function tests for every publication in the source repository. Even if just one of these tests fails, the team's priority is to restore the stability of the product. The procedure is executed quickly and regularly [Duvall 2007].

2) **Continuous delivery**

Continuous delivery is an extension of continuous integration. According to [Fowler 2013], the purpose is to build an application that can be approved for production as a trusted system at any time. This way of working is very popular in the DevOps movement, whose motto is: "You build it, you run it."

It focuses on the automation of the software delivery process, so that the teams can easily deploy their code for production, while being assured of its reliability at all times. By ensuring that the codebase is constantly in a deployable state, publishing the software requires no complex coordination or advance-stage testing [Humble 2011].

Continuous delivery means that the time between an idea and its availability to users is as short as possible. It is a beneficial practice because it automates the steps between the verification of the code in the repository and the decision to release functional, tested builds on the production infrastructure. The steps that guarantee the code's quality and exactness are automated, but the final decision on what must be released remains in the hands of the organizers to guarantee maximum flexibility.

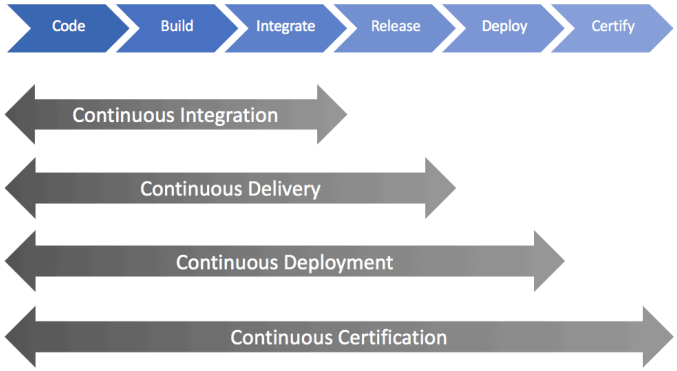
As with continuous integration, continuous delivery requires the implementation of rigorous processes that capitalize on the use of tools and organization that is tailored to be efficient.

3) **Continuous deployment**

Continuous deployment goes even further. It is an extension of continuous delivery, which involves delivering every change made to the software to the end user. In this type of operation, there is no human involvement to decide on when to deploy during production; an automatic deployment system deploys all changes, except those which fail a test. This practice accelerates the feedback loop and enables developers to better focus on the software development, because there is no “delivery date” to look ahead to.

However, this entirely automated deployment cycle can be a source of anxiety for teams that are concerned about abandoning the control of their system as to what is released. The trade-off that automated deployment offers is sometimes deemed too dangerous for the rewards it provides.

Fig. 6 outlines software development activities and the various processes associated with them.



Continuous development processes

Our opinion is that a natural extension of this ‘continuous’ dynamic therefore consists in continuously conducting the activities required by certification throughout the safety-critical software development; this corresponds to continuous certification [Louis 2019]. The goal of continuous certification is to apply agile principles to the development of safety-critical software. At each iteration of software development, aimed at providing an operational increment of the final product, the required certification objectives must be met to achieve a "Certification Ready" status on that intermediate deliverable, thus avoiding that certification requirements are only met at the end of the process.

VI. TOWARDS A CONTINUOUS CERTIFICATION

Avionics industry stakeholders strongly believe that the certification standards in their field require linear development (such as Waterfall or V-staging). However, while there are references to it in the standards, they do not impose any life cycle; they set the objectives described in the processes to be established. Agile, whose efficiency is recognized for software development, is not incompatible with aeronautical standards. An agile development of avionics safety-critical software thus is possible. It would be even more efficient if answering the DO-178C requirements was fully integrated in the development process.

This section first reminds the limits of traditional software development and the key concepts and general pattern of agile. It then highlights the benefits of integrating agility to have a continuous development process for software subject to certification, while underlining the barriers. To conclude, it mentions and discusses a few examples of successful experiments in industry, thereby proving that these barriers can be surpassed and initiating a trend to follow.

A. Limits of traditional development strategies

Among the several sources of project failure, we commonly agree on a high level of compartmentalization and a lack of communication between teams, as well as a costly and burdensome documentation to produce. Quality assurance often comes last, solely solicited to acquire a stamp to validate a project, meaning it became an adjustment criterion. Client needs are often poorly accounted for, and the solution lacks value. Lastly, the deliverable often is not available on the scheduled date and development cycles are too long [Standish Group 2015].

To overcome these issues, silos need to be broken down [Xue 2017], information must be gathered from the teams and the client, working methods should leave a margin for initiative, and quality must be assured throughout the development process. To do so, a V-model cycle is not suitable. As we can see in

the descending section of Fig. 6 the stages follow on in a cascading manner, in a linear sequence that, to analyze it simply, has three stages: everything is thought of, everything is planned, and everything is done, exactly as planned. Then everything is tested and delivered, once and for all.

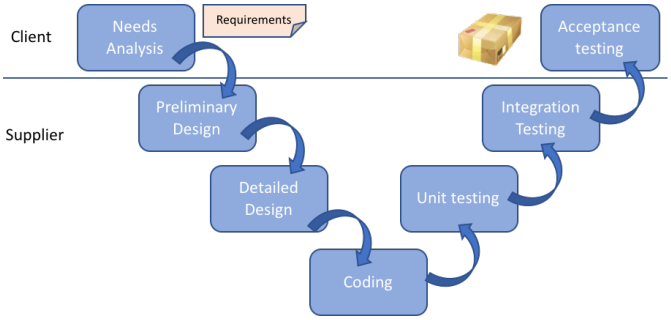


Fig. 6. Simplified diagram of a V-model cycle (adapted from [Ninni 2019])

This has several drawbacks. As clients are mainly involved at the beginning and the end of the cycle, it demands an in-depth initial analysis and design stage to ensure all needs and possible issues are anticipated. If clients have forgotten a constraint or wishes to add or modify something, they must wait until the product delivery, then launch a new project that will take their new needs into account (this rule is often bypassed but results in significant delays and additional costs). Even if the V-model cycle authorizes backtracking, a late discovery of a problem (for instance during integration) could threaten the entire project. Lastly, if the project is delayed, it is generally the final tasks to be carried out, such as reviews for certification or product testing, which suffer. The time allocated to these activities is therefore reduced, which negatively affects the product's quality.

The main issue in our context is to avoid situations that result in the certification activities being carried out too late, under the pretext that they do not add any value. Once the software is delivered to the client and the acceptance tests have been done, there is not much point in holding a requirement review. However, this review is much more beneficial if it is conducted throughout development, when the data is produced. This both limits the efforts and maximizes the impact of a detected anomaly (completeness, testability). This risk needs to be avoided because economic pressure tends to render these activities less beneficial if they are carried out too late.

For these reasons, agile frameworks are extremely useful. Projects are organized based on iterative short loops rather than a long linear sequence of stages. The aim is to deliver intermediary versions of operational solutions to the client so they can measure the project's progress and validate the direction taken. They promote communication in and among the teams, as well as with the various stakeholders, the client and the certification authorities. Agile frameworks take a pragmatic approach: the main thing is that everything works and that everyone involved is satisfied, including the certification authorities.

B. *Agile software development*

Agile corresponds to a philosophy that guides all actions and processes in an organized structure designed for the clients [Diaz 2017]. The primary objective is to maximize business value as early as possible in short, high-quality, industrialized increments, thereby reducing short-sightedness. It also allows for better, faster adaptation to changes, enabling developers to continuously improve the solution and capitalize on the collective intelligence of a company. The aim is to create a more natural way of working. Agile offers freedom and a certain degree of autonomy, both for project organization and engineering, but do requires great rigor, a precise and demanding framework, and daily monitoring.

Agile puts forward a certain number of values:

1. Valuing individuals and interactions over processes and tools. The principle consists in setting small, clearly defined objectives that can be easily achieved and that it is possible to commit to. This means the commitment can be respected, teams can be proud and satisfied of their work, and continue.

2. Focusing on operational software rather than comprehensive documentation. Agile means producing complete, high-quality segments of the application, and the expected solution behavior is constantly verified. Working software is preferred to the completeness of the functions. This does not mean that documentation is of no importance: the solution evolves regularly, so the necessary documentation must be maintained and sufficiently comprehensive, so it can be capitalized on.

3. Collaboration within teams and with customers (the entire team is responsible for each task) rather than contract negotiation; creating valuable software and delivering it as early as possible. There is a shared vision of the software and the project (sharing the same objectives, the same language, the same budget constraints, deadline and organization), headed by a customer representative, who is part of the development team.
4. Accepting and responding to change rather than following a strict plan.

Twelve principles stem from these values [Manifesto 2001]. Among them, satisfy the customer, accept that requirements may need to be changed, deliver frequently, motivate teams, face-to-face conversation and simplicity are some of these principles.

As a result, there is a real possibility the solution can be brought to market faster, with higher productivity and quality, lower costs, greater satisfaction for stakeholders, greater commitment and work satisfaction from employees - all of which are solid reasons for adopting an agile approach.

C. *General pattern of agile software development*

Several frameworks support agile. Some examples among the most well-known are: Scrum [Scrum 2018], eXtrem Programming [Xu 2009], Safe [Leffingwell 2016], Lean Management [Salma 2018], Kanban [Ahmad 2013] or DevOps [Goudeau 2016] [Verona 2016]. Even if each does have its own specificities, they are based on similar values and mechanisms [Saleh 2019] [Alqudah 2017].

The agile life cycle (see Fig. 8) is characterized by short iterations lasting a few weeks. The project is divided into functionalities to be developed. They are described using the customer's choice of vocabulary and represent the need from the user's point of view. For each functionality, there is an estimate of the volume of work needed to develop, test and validate it, as well as a relatively simple test similar to a validation test. A detailed description of the technical options to be implemented is added to the functions. The list is drawn up, and the customer evaluates the priorities.

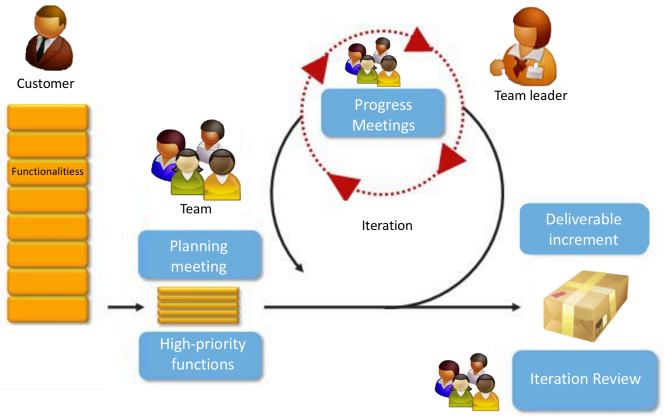


Fig. 7. Agile process

Several ceremonies structure the execution of an iteration. The method is supervised by a team leader to ensure it works. At each iteration, he organizes a planning meeting during which the most high-priority functionalities for the customer are selected from the list. They will be developed, tested and delivered to the client after the iteration. During the iteration, short progress meetings are organized each day at which all team members indicate the tasks carried out the day before, the tasks planned for the current day and the problems encountered. The aim of this meeting is not to resolve the problems but simply to identify and mention them so that the iteration objectives can be met. Following this meeting, the team leader updates what was done and evaluates the team's pace of work. At the end of an iteration, a demonstration of the latest developments is provided for the customer. It is also an opportunity to debrief on how the team operates and find areas to improve on.

D. *Introducing agility in industrial practices in avionics*

The amount of software used in safety-critical systems has been increasing at a rapid rate in aeronautics since the last decades. At the same time, software technology is changing, projects are pressed to develop software faster and more cheaply, and the software is being used in more critical ways [Rierson 2013]. Agile methods had a huge impact on how software is developed. In many cases, this has led to significant benefits, such as quality and speed of software deliveries to customers. However, safety-critical systems have widely been dismissed from benefiting from agile methods. Indeed, agile practices, according to the way they are popularized,

advertise minimal documentation, refactoring of code, upfront planning and iterative release of project, that in a first sight seems to contradict safety requirement standards of safety critical systems [Mwadulo 2016]. Products that include safety critical aspects are therefore faced with a situation in which the development of safety-critical parts can significantly limit the potential speed-up through agile methods, for the full product, but also in the non-safety critical parts. For such products, [Kasauli 2018] demonstrates that the ability to develop safety-critical software in an agile way will generate a competitive advantage.

However, very few companies in the avionics industry use agile to develop safety-critical software. They usually are conservative and want to use the traditional methods because they have been thoroughly tested over time and they are familiar with [Mwadulo 2016]. Many are afraid of having to convince the authority to introduce a new method. Others do not want to modify their engineering workflow for fear of having to prove that existing projects are not affected by these modifications. This is a legitimate strategy but deprives the development teams of numerous technological advances that facilitate the production and certification audit processes.

Furthermore, as noted by [Lemoussu 2018], guidelines often are poorly interpreted. A survey performed in [Kornecki 2008] also assesses that « the relative vagueness of these guidelines causes significant differences in interpretation by industry and should be eliminated». One of the reasons for the high development costs of avionic systems complying with standards may be a lack of sufficient understanding of how to employ these standards efficiently [Youn 2014]. For example, manufacturers misinterpretation of the DO-178C guidelines often results in self-imposing a V-model. In their defense, as we can see in Fig. 9, DO-178C appears to suggest a linear development of systems. This is due to the fact that ARP4754 and DO-178 standards were written at a time when development in avionics was based on a V or a waterfall model. However, as long as a process can be demonstrated to meet the needs of the relevant standard, the development team is free to use whatever processes they want to use [Douglass 2020].

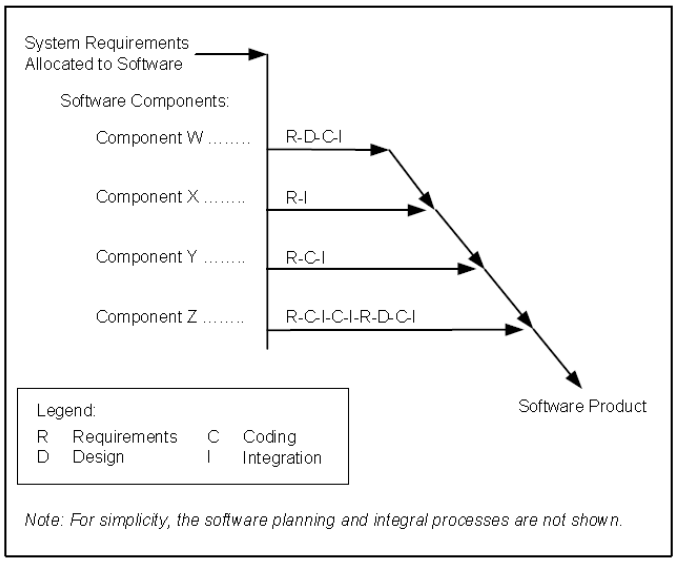


Fig. 8. Development cycles illustrated in [RTCA DO-178C 2012]

Lastly, certification-related regulations are often referred to at the end of the development phase or are conducted at the end of the development process activities, just to prove that the software complies with the standard. Demonstrating compliance is not a continuous activity, whereas integrating it within the development process would have several advantages, in particular providing greater safety assurance and reducing costs.

E. Some emerging initiatives, initiating a trend

A few but successful initiatives that have been launched in the avionics and automotive industries, show a recent, if marginal, change in industry methods, to make the development of safety-critical software more agile. This trend does however seem promising.

An example to date is Thales Avionics' development of the ADIRU calculator [Chenu 2013]. The teams in charge of development succeeded in setting up a continuous delivery process with its customer (Airbus). The teams demonstrated the feasibility of this innovative concept, which involves regularly delivering a solution (here, a combination of hardware and software) with a limited but operational functional scope. The number of errors observed by the customer was 99% less when compared to a similar project. The cost of product integration has been decreased from 30% to nearly 5% of the project budget. To obtain these results, the teams, working in agile mode, understood that the

certification objectives were non-negotiable and put in place methods and tools to automate as many activities as possible (traceability, test execution, statistical analysis). Several times per day, the latest software version is fully tested. This practice grows and maintains an assembled and operational software product. Such practices have revealed encouraging results. The cycle-time has been reduced from one year to 20 days. Therefore, integration is no more a late big batch of work. This activity is now performed early and very often within each iteration. In a 3-year timeframe, 9 versions of the product have been delivered on schedule to the customer. Finally, the software has successfully passed its first flight-tests.

Airbus Helicopter more recently experimented the Scrum framework to develop a new avionics system (military application embedded software). [Marsden 2018] shows how apparent contradictions between agile practices and avionics software certification objectives have been resolved in a number of Airbus projects. It is demonstrated that significant improvements in quality, schedule and cost have been achieved. Moreover, several use cases prove that, when carefully deployed, agile techniques are not only compatible with DO-178C, but through greater visibility and openness actually simplify it.

Nexter, a longstanding defense industry company, has adapted its practices to the IEC61508 standard. Supported by Serma, an engineering company, it has also set up a process to assess the compliance of its contractors' software development with the IEC61508 guidelines. It has been deployed to all new safety-critical software developments such as for the military project named EBMR (Engin Blindé Multi-Rôles) in 2019. From the authority point of view, this is an extremely positive initiative, that guarantees that safety is monitored in future developments of defense systems. As a newcomer to the certification world, Nexter quite easily succeeded in performing this transition because building a new process requires less effort than adapting which is outdated.

Tesla implemented a continuous process to develop safety-critical software (data integrity and confidentiality, service availability, safety functions) that are embedded in their vehicles [Vöst 2016]; it is shown on Fig. 10. It features agile principles of continuous integration and continuous deployment. From a commit, application software is automatically integrated at ECU level, then this ECU is deployed on test benches before the legal acceptance and the deployment over the air. End user firmware update is allowed at the end of each iteration every month. However, the safety analyses and approval from U.S. authorities (National Highway Traffic Safety Administration, NHTSA) remain sequential. [Vöst 2016] doesn't state if NHTSA analyses each deployed software release.

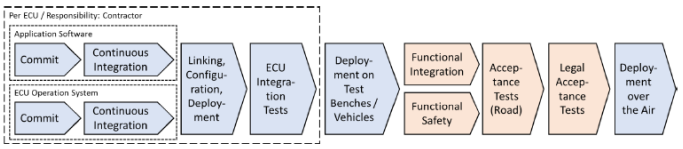


Fig. 9. Continuous development process followed by Tesla [Vöst 2016]

Sogilis very recently succeeded in developing a DO-178C/ED12C – DAL A level "Autopilot" software for drones (Pulsar Flight System project) by applying certain agile principles, in particular test-driven development [Mrabti 2018]. They went even further by formalizing the expression of test cases through their formal expression, facilitating the automation of function verification while respecting the crucial principle of requirement-based testing.

The methodology implements a number of standard tickets with JIRA. The tickets represent a set of development activities to be carried out corresponding to the writing of requirements, test cases or source code. In addition to the specification of activities, they designate one or more activity and review managers to ensure independence at this level if necessary.

Different types of tickets are therefore defined, in relation to the operations they require:

- Type 1: Improvement of a process or a system (Evolution of certification plans, development standards or system requirements)
- Type 2: Creation or review criteria (test case) of software requirements
- Type 3: Definition of software architecture
- Type 4: Definition of the expected behavior of the various components and test cases (unit tests) to validate their proper functioning
- Type 5: Writing the source code of the components and the corresponding unit tests
- Type 6: Code integration and test execution (integration tests, functional tests and user tests)

- Type 7: Management of the different possible configurations of the software.
- Type 8: Specific management of open problem report
- Type 9: Delivery

Each ticket is created and written according to the progress of the project to define the tasks to be carried out and the people responsible for them, including actors and reviewers. The edition of a ticket can lead to the creation of other tickets in connection with the first and the associated activities will then be carried out by the assigned personnel.

Each ticket is also characterized by a state, image of the progress of the activity linked to the ticket. All of these states make it possible to define the lifecycle of a ticket or workflow, which is the model for the evolution of the state of tickets during their development (see Fig. 11). Thus, the workflow is defined as a chain of states linked to each other by transitions (see Fig. 11). Transitions represent the conditions necessary for a ticket to transit from one state to another. The crossing of a transition is therefore directly linked to the actions carried out in response to the activities described in the ticket.

The principle of the workflow is first of all to guarantee a chain of states that all the tickets, whatever their type, will have to follow so that the related activities are carried out in the right way, implying respecting requirements related to the development process and the organization of the team. This workflow must be simple: states and transitions are carefully determined, and their number is minimized; this keeps the work organization intuitive and easy to follow.

1- New ticket
2- Planning
3- Team takes possession
4- Activity started
5- Rejection
6- Pause
7- End of activity
8- Activity verified
9- Review rejected
10- Verification rejected
11- Ticket reopened
12- Rejection
13- Planning
14- Retrospective done

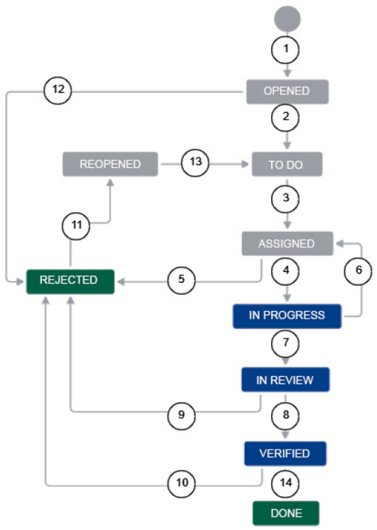


Fig. 10. Ticket management workflow

Setting up a process like this one does come at an initial cost that must be shared as much as possible by several projects and development teams. Processes for continuous development are complex due to the numerous bricks used to automate the tasks. Moreover, changes in working habits that are brought by continuous integration process must be accompanied by training in best practices. Lastly, certain obstacles must be overcome so that it can be applied in a certification context. For instance, a lack of documentation will not be tolerated: the processes must be stable and certain objectives required by standard DO-178C must be partially satisfied (i.e. completeness).

However, the value added for the end user, the team's satisfaction with the work done and the image portrayed to the certification authority is invaluable. When the solution is evaluated, the maturity can be seen and means that its release can be authorized with confidence in the result.

In synthesis, these experiments show there exists a current industrial trend in safety-critical development which consists in making the certification process even more integral to the development process. They demonstrated that safety assessments could be continuously performed by the authorities.

F. Benefits of introducing agility

Agile aims at ensuring that all the customer's requirements are met. In terms of software development for certification, the authority may be considered as the most important customer that must be satisfied to respond to airborne safety systems as a societal challenge. Early integration of its needs reduces risks and development costs. It boosts the level of trust by guaranteeing the activities are conducted at the ideal time, rather than right before the auditor's visit just to satisfy them. It facilitates the sampling stage during certification audits as it allows for immediate traceability between all data and systematic saving of proof that the activities required for certification have been conducted (reviews, test result, structural coverage rate, performance metrics)

This approach requires automated processes to be put in place using the right tools, to continuously carry out the tasks required to satisfy the objectives in DO-178C, or any other safety-critical software

development standard ([IEC 61508 2010], [ISO 26262 2018]). The aim is to reach a "ready for certification" status before the end of each iteration for every piece of data or activity produced (requirement, test, review). Being able to automate the whole process means that if necessary, the application can be reworked without difficulty, by immediately measuring the activities that it is necessary and sufficient to repeat. Automating all tests saves a significant amount of time in this stage. The teams will not be afraid of making changes to the software and the customer's and the authority's needs will be satisfied at less expense.

G. Issues or potential barriers

The standard DO-178C outlines the objectives to be met during development phases by implementing a number of activities. These activities are grouped according to the type of process. The methods and tools to organize and deploy the processes and activities are not specified in DO-178C. The issue there is defining a workflow that makes it possible to plan activities and processes in agile iterations, in compliance with DO-178C, as experimented [Mrabti 2018]. Beyond the specific case of avionics software, an actual research debate is regarding whether safety-critical systems are better developed with traditional waterfall processes (iterative development) or agile processes (incremental development) that are purportedly faster and promise to lead to better products [Tordrup 2018].

In an earlier study, [Weyrauch 2004] considered the use of Agile for safety critical software development, identifying not only the issue of *whether* agile methods can be used but *how* they can be used in the safety-critical world, addressing a panel of myths, worries, solutions and experiences. [Douglass 2012] concluded that some key agile practices can assist in the development of safety-critical systems, such as incremental development (evolutionary development with frequent requirements-based verification), test-driven development (development and application of unit tests as the code is developed), continuous integration (continuously building software and verifying the various components work together properly), dynamic planning (updating plans based on continuously measured "ground truth") and risk management (identifying and prioritizing project risks and reducing them through risk strategies). [Tordrup 2018] highlighted that, however incremental development seems better suited than iterative development, four problems seems to remain, about documentation, requirements, lifecycle and testing.

[Coe 2013] attempted examining the agile-management principles and the basis of DO-178C and identified four main sources of possible conflict, outlined in TABLE VI. However, this study seems to convey a poor interpretation of the agile principles and also of those promoted by the DO-178C standard.

TABLE VI. AGILE-MANAGEMENT PRINCIPLES VERSUS KEY RECOMMENDATIONS IN DO-178C (ADAPTED FROM [COE 2013])

Agile-management principles	**Key recommendations of DO-178C**
Individuals and their interactions	Processes and tools
Working software	Comprehensive documentation
Evolving needs in collaboration with the customer	Rigorous specification of requirements
Adapting to change	Following a plan

Let us discuss these potential sources of problems.

Individuals and their interactions vs. Processes and tools

Rather than siloing teams, Agile team complete everything including design, development, and testing. This human interaction can benefit to the production a certifiable software solution, even more so when the team must present their work to an auditor. The social dimension is unavoidable in order to reach a justified level of trust. So that the auditing is not just a situation to put up with, right from the beginning of the project the capacity to be audited should be taken into consideration. A team that is proud of the work done, comfortable with their engineering process and which is able to quickly present all the required proof, naturally inspires trust.

However, unsiloing teams means that there isn't a separate team that handles compliance tasks and there's no person whose sole job is validation. For regulated industries, verification, validation, traceability, and other activities that produce compliance documents typically falls on quality assurance members. According to [Krüger 2019], a simple solution would consist in that some members of the team will be dedicated this role. [Gardner 2020] presents the evidence from the literature of the benefits of agile methods to develop safety-critical software with regard of the independence of roles.

[Van Schooenderwoert 2018] sharing their feedback on using Agile to develop safety critical complex systems for medical devices, also noted that Agile is not only compatible with a critical and complex environment, but is also extremely effective in providing an ability to test the product frequently, with some adaptations, including the definition of roles as well as the selection of a risk management method compatible with the standard that will have to be integrated to Agile.

Working software vs. Comprehensive documentation

The DO-178C requires three documents to be delivered: the Plan for Software Aspects of Certification, the Software Accomplishment Summary and the Software Configuration Index. The format of other engineering data is left open. In the past, industry practice was to work on the documents in Word or in databases such as Doors, which are not necessarily suited to consistently satisfying the objectives of DO-178C. Other digital formats such as HTML or Markdown appear to be more efficient in covering traceability and version management of documentary data.

There is often a confusion about documentation and traceability. The Agile tenet doesn't mean documentation should be eliminated. It recommends minimum documentation, but this does not mean no documentation at all. It's driving towards the elimination of wasteful documentation. No large-scale software engineering project can exist without formalizing the essential technical information. The certification process does not ask for more than that essential information, which offers a guarantee that the behavior implemented in the source code corresponds to a detailed technical specification apt for testing. The Agile methodology aims to produce valuable reports, of which a traceability matrix could be included if it's for a regulated industry. For safety-critical software, with regard to detailed specifications, there should be no absent or unintended behavior in the source code. This necessitates quite a low level of granularity to ensure there is no possible interpretation of the requirement by the person in charge of coding it. The agile principles of iterative functional increments are perfectly tailored to this requirement. In conclusion, maintaining concise and well-organized records that ensures traceability is possible with Agile. Moreover, a product development solution (such as Helix ALM), automating the process, which allows both instantly generating documentation and streamline operations, can simplify this [Krüger 2019].

Evolving needs vs. Rigorous specification

For a given functional scope, the rigorous activities required by DO-178C can be carried out in agile mode so that a certification-ready solution can be delivered at each iteration. Adding a feature has an impact on the previously implemented functionality, so the iteration in question must also take into account the modifications required for previous artefacts (specifications, tests, source code). Otherwise, they have to be programmed during later iterations to quickly ensure the whole solution is consistent. The configuration management system should be able to log the development history. However, overall performance metrics and the completeness of the desired behavior can only be established at the end of the software development.

[Vuori 2011] underlines that organizations can change their processes to a more agile way without risking the safety of products. [Hanssen 2018] provides an overview of agile software development and how it can be linked to safety and relevant safety standards. It proposes guidelines and additions to make Scrum, for instance, both practically useful and compliant with the additional requirements found in safety standards.

Adapting to change vs. Following a plan

Finally, the plans must be as stable as possible. This does not exclude the possibility of modifying them as part of a controlled continuous improvement process.

There is, however, another issue to deal with: contracting this way of working, where the solution delivered can differ greatly from the functionality initially stated. In a conventional contract, if the needs change, the supplier is fully responsible for that risk. Certain precautions can be taken, but the flexibility of agile methods will be limited. Some initiatives have emerged, such as separating each iteration into an individual flat-rate product, meaning the client can stop the contract at any point. The principle of requirement trade-offs involves producing an unplanned feature in exchange for the removal of

another less important, non-priority feature of equal cost. More traditional methods such as contract riders allow a certain number of modifiable requirements (<10%). Certain manufacturers bill customers after each iteration. Swiftly taking changing needs during development into account in customer contracts will become an increasingly large challenge, in particular when dealing with cyber-security issues, as the cycle of threats move quicker than current development cycles.

In addition to this above analysis, there is a point to which attention must be paid when willing to introduce agility in the development of safety-critical software: each agile framework has its own specificities, and the method chosen for a project should depend on the context and constraints. For example, Scrum should not be chosen - at least without adapting it - to develop software subject to certification. Scrum recommends that the entire team be responsible for all tasks; this is not directly compatible with the independence required in standards that mean the developing team must be different to the testing team, and that task managers be identified and traced. In this example, a lean management method would be preferable, with each member of the agile team being responsible for a task, or else an adapted Scrum method to divide up tasks and record the identity of the person who carried out the activity. In synthesis, the point of using Agile is not to strictly adhere to every bit of it but to improve the product development process [Krüger 2019].

VII. CONCLUSION

This article discussed the issue of safety-critical software certification in avionics from the point of view of an authority and assessed the state of current industry practices. It highlighted the difficulties encountered in companies and offered some leads for possible improvement of practices through agile methods. It identified the challenges ahead in the near future and the fundamental changes and transformations that are occurring in the field of safety-critical software engineering, underlining the need for companies to be ready to adapt their practices before long.

Certain agile principles, and how they are interpreted, are not entirely compatible with the certification process. Agile indeed improves development. However, in safety-critical environments, modifications are necessary to ensure compliance is still met. Because compliance and safety are cornerstones of regulated industries, it's important to identify how critical information will remain part of an Agile process.

Some industrial initiatives have recently experimented introducing Agile into the development process. The results have shown that these new practices help boost the level of trust and reduce development costs. They indicate that a widespread adoption of agile practices in the avionics industry is feasible and that the authority's expectations are compatible with agile development processes. Delivering certifiable or certification-ready software more frequently to an end user ensures one requirement is covered and gives a clear indication on the progress made in certification to authorities. Thanks to the convergence of the various stakeholders' interests (development team, operators quality control, certification authorities), the development process runs more smoothly, efficiently and collaboratively, to satisfy the societal guarantees that are needed at the end.

Furthermore, agile methods place a focus on human values, a crucial aspect in the development of safety software, which relies on trust. These initiatives deserve to be encouraged, by positioning them and adapting them so that they meet certification requirements.

The avionics industry is leading the way in strict adherence to standards requirements. This is an aspect that avionics engineers have long grasped. We are also seeing new players enter the fields of drones and autonomous cars, implementing innovative industrial techniques and technologies such as agile methods, artificial intelligence, model-based safety assessment. New technology appears at a fast pace, which is out of step with the authorities' capacity to write them into regulations. The challenge for authorities is to keep up with these new developments by giving them a framework and guiding them to maintain a controlled level of acceptable risk.

REFERENCES

[1] [DOD MIL-HDBK-338B 1998] Department of Defense. “Military Handbook - Electronic Reliability Design Handbook”, 1998.

[2] [Leveson 2003] Leveson, Nancy. “White Paper on Approaches to Safety Engineering”, April 2003. http:// sunnyday.mit.edu/caib/concepts.pdf

[3] [Rempel 2014] Rempel, Patrick; Mäder, Patrick; Kuschke, Tobias; Cleland-Huang, Jane. “Mind the Gap: Assessing the Conformance of Software Traceability to Relevant Guidelines”, International Conference on Software Engineering (ICSE), New York, USA, ACM: 943–954, 2014.

[4] [Leveson 2004] Leveson Nancy, “A New Accident Model for Engineering Safer Systems”, Safety Science, Vol. 42, No. 4, April 2004, pp. 237-270

[5] [Wessiani 2018] Wessiani N.A., Yoshio F., “Failure mode effect analysis and fault tree analysis as a combined methodology in risk management”, IOP Conf. Series, April 2018.

[6] [Braun 2009] Braun P., Phillips J., Schatz B., Wagner S., “Model-Based Safety-Cases for Software-Intensive Systems”, Electronic Notes in Theoretical Computer Science 238(4):71-77, 2009.

[7] [Gario 2018] Gario, A., Andrews, A., Hagerman, S. “Fail-safe testing of safety-critical systems: a case study and efficiency analysis”, Software Qual J 26, 3–48, 2018.

[8] [RTCA DO-178C 2012] RTCA SC-205, EUROCAE WG-12, DO-178C/ED12C, “Software Considerations in Airborne Systems and Equipment Certification”, January 2012.

[9] [Kornecki 2008] Kornecki A., Zalewski J., "Software certification for safety-critical systems: A status report", International Multiconference on Computer Science and Information Technology, Wisia, pp. 665-672, 2008.

[10] [ARP4754A 2011] Society of Automotive Engineers, Aerospace Recommended Practice “Guidelines For Development Of Civil Aircraft and Systems”, November 2011.

[11] [Skokovic 2010] Skokovic P., Rakic-Skokovic, M., “Requirements-based testing process in practice”, International Journal of Industrial Engineering and Management, 2010.

[12] [Ebert 2016] C. Ebert, J. Cain, G. Antoniol, S. Counsell and P. Laplante, "Cyclomatic Complexity," in *IEEE Software*, vol. 33, no. 6, pp. 27-29, Nov.-Dec. 2016, doi: 10.1109/MS.2016.147.

[13] [Osetskyi 2018] Osetskyi Victor, “What Technical Debt Is and How to Calculate It”, Agile Zone, Opinion, July 2018. https://dzone.com/articles/what-technical-debt-it-and-how-to-calculate-it

[14] [NT DGATA 2016] DGA Techniques aéronautiques, Note Technique 16-DGATA-P1301261003001-1P-C “Référentiel d’exigences d'ingénierie des logiciels et composants électroniques complexes pour la prise en compte de la sûreté de fonctionnement”, 2016.

[15] [Letouzey 2012] J.-L. Letouzey, "The SQALE method for evaluating technical debt," in Proceedings of the Third International Workshop on Managing Technical Debt, pp. 31-36, 2012.

[16] [RTCA DO-254 2006] RTCA and EUROCAE, RTCA DO-254/EUROCAE ED-80 “Design assurance guidance for airborne electronic hardware”, 2006.

[17] [Hilderman 2009] Hilderman Vince, “DO-178B Costs Versus Benefits”, HighRely White Paper, 2009. http://www.highrely.com/whitepapers.php

[18] [Düllmann 2018] T. F. Düllmann, C. Paule and A. v. Hoorn, "Exploiting DevOps Practices for Dependable and Secure Continuous Delivery Pipelines," *2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE)*, Gothenburg, Sweden, 2018, pp. 27-30.

[19] [Royce 1998] Royce W, “Software project management: a unified framework”, September 1998.

[20] [Pillou 2018] Pillou J.F., “Concept de l’Intégration Continue”, issu de CommentCaMarche, 2018. https://www.commentcamarche.net/

[21] [Fowler 2006] Fowler M., Continuous Integration, https://martinfowler.com/articles/continuousIntegration.html, 2006.

[22] [Duvall 2007] Duvall Paul, Matyas Steve, Glover Andrew. “Continuous Integration: Improving Software Quality and Reducing Risk”, 2007.

[23] [Fowler 2013] Fowler M., Continuous Delivery, 2013. https://martinfowler.com/bliki/ContinuousDelivery.html

[24] [Humble 2011] Humble J, Farley D. “Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation”, 2011.

[25] [Louis 2019] Louis V., Baron C., " Vers une certification continue des logiciels critiques en aéronautique ", Techniques de l’Ingénieur, 27 p, November 2019.

[26] [Standish Group 2015] Standish Group International, Inc. “Report in Computer World”, 2015.

[27] [Xue 2017] Xue R., Baron C., Esteban P., “Optimizing product development in industry by alignment of the ISO/IEC 15288 Systems Engineering Standard and the PMBoK Guide”, International Journal of Product Development, vol. 22, issue 1, pp. 65-80, 2017.

[28] [Ninni 2019] Ninni L., Blog Launizo consulting, 2019. https://www.launizo.com/blog/methodes-et-outils-de-productivite-en-entreprise-1/post/les-methodes-agiles-3

[29] [Diaz 2017] Diaz Vargas D., Baron C., Esteban P., Gutierrez C., " Is there any Agility in Systems Engineering?", INSIGHT journal, INCOSE, December 2017.

[30] [Manifesto 2001] Cunningham W, Beck K., Fowler M, Thomas D, “Manifesto for Agile Software Development”, August 2001.

[31] [Scrum 2018] Scrum.org, “What is Scrum?”, consulted 02/12/2018. https://www.scrum.org/resources/what-is-scrum?

[32] [Xu 2009] B. Xu, "Towards High Quality Software Development with Extreme Programming Methodology: Practices from Real Software Projects," 2009 International Conference on Management and Service Science, Wuhan, 2009, pp. 1-4, doi: 10.1109/ICMSS.2009.5302042.

[33] [Leffingwell 2016] Leffingwell D., “SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Enterprises “, 2018

[34] [Salma 2018] A. Salma, C. Anas and E. H. Mohammed, "How can Top management succeed in a lean manufacturing implementation in the small and medium sized enterprises?," 2018 International Colloquium on Logistics and Supply Chain Management (LOGISTIQUA), Tangier, 2018, pp. 176-181, doi: 10.1109/LOGISTIQUA.2018.8428287.

[35] [Ahmad 2013] M. O. Ahmad, J. Markkula and M. Oivo, "Kanban in software development: A systematic literature review," *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, Santander, 2013, pp. 9-16, doi: 10.1109/SEAA.2013.28.

[36] [Goudeau 2016] Goudeau Stéphane, Metias Samuel, Découvrir DevOps, l’essentiel pour tous les métiers, Dunod, Mars 2016.

[37] [Verona 2016] Verona Joakim, “Practical DevOps”, Packt Publishing, February 2016.

[38] [Saleh 2019] S. M. Saleh, S. M. Huq and M. A. Rahman, "Comparative Study within Scrum, Kanban, XP Focused on Their Practices," *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, Cox'sBazar, Bangladesh, 2019, pp. 1-6, doi: 10.1109/ECACE.2019.8679334.

[39] [Alqudah 2017] M. Alqudah and R. Razali, "A comparison of scrum and Kanban for identifying their selection factors," *2017 6th International Conference on Electrical Engineering and Informatics (ICEEI)*, Langkawi, 2017, pp. 1-6, doi: 10.1109/ICEEI.2017.8312434.

[40] [Rierson 2013] Rierson L., “Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance”, January 2013.

[41] [Mwadulo 2016] Walowe Mwadulo M., “Suitability of Agile Methods for Safety-Critical Systems Development: A Survey of Literature”, International Journal of Computer Applications Technology and Research Volume 5– Issue 7, 465 - 471, 2016.

[42] [Kasauli 2018] Kasauli R., Knauss E., Kanagwa B., Nilsson A., Calikli G., “Safety-Critical Systems and Agile Development: A Mapping Study”, 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), August 2018.

[43] [Wolff 2012] S. Wolff, "Scrum goes formal: Agile methods for safety-critical systems," International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches, Zurich, pp. 23-29, 2012.

[44] [Lemoussu 2018] Lemoussu S., Chaudemar J.-C., Vingerhoeds R.A., “Systems Engineering and Project Management Process Modeling in the Aeronautics Context: The SMEs Study Case”, International Journal of Mechanical and Mechatronics Engineering, vol. 12 (n° 2). pp. 88-96, 2018.

[45] [Kornecki 2008] Kornecki A., Zalewski J., "Software certification for safety-critical systems: A status report”, International Multiconference on Computer Science and Information Technology, Wisia, pp. 665-672, 2008.

[46] [Youn 2014] Youn W., Yi B-J, “Software and hardware certification of safety-critical avionic systems: A comparison study”, Computer Standards & Interfaces, Volume 36, Issue 6, pp. 889-898, 2014.

[47] [Douglass 2020] Douglass B., “Agile analysis practices for safety-critical software development”, pp 1-14, February 2013. Consulted 9th June 2020, https://www.ibm.com/developerworks/rational/library/agile-analysis-practices-safety-critical-development/

[48] [Chenu 2013] Chenu E., "Integration Continue” , Séminaire Ingénierie des Systèmes Complexes à Logiciels Prépondérants, ISCLP, 2013.

[49] [Marsden 2018] Marsden J., Windisch A, Villermin J., Aventini C., Mayo R., Grossi J., Fabre L., ”ED-12C/DO-178C vs. Agile Manifesto – A Solution to Agile Development of Certifiable Avionics Systems”, Conférence Embedded Real Time Software And Systems (ERTS^2), Toulouse, France, Février 2018.

[50] [Vöst 2016] Vöst S., Wagner S., “Towards Continuous Integration and Continuous Delivery in the Automotive Industry”, 2016.

[51] [Mrabti 2018] Mrabti A., Gautherot D., Brossard V., Moy Y., Pothon F., “Safe and Secure Autopilot Software for Drones”, Conférence Embedded Real Time Software And Systems (ERTS^2), Toulouse, France, Février 2018.

[52] [IEC 61508 2010] International Electrotechnical Commission, “Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems”, https://www.iec.ch/functionalsafety/standards/

[53] [ISO 26262 2018] ISO 26262, ISO TC22/SC3/WG16, “Road vehicles - Functional safety”, First edition 2011, retrieved 2018, https://www.iso.org/fr/search.html?q=26262

[54] [Tordrup 2018] Tordrup L., Nielsen P., “A Conceptual Model of Agile Software Development in a Safety-Critical Context: A Systematic Literature Review”, Information and Software Technology, 2018.

[55] [Weyrauch 2004] Weyrauch K., Poppendieck M., Morsicato R., Van Schooenderwoert N., Pyritz B., “Agile Methods for Safety-Critical Software Development, Extreme Programming and Agile Methods - XP/Agile Universe”, Lecture Notes in Computer Science, Vol. 3134, 2004.

[56] [Douglass 2012] Powel Douglass B., Ekas L., “Adopting agile methods for safety-critical systems development”, IBM Software White paper, October 2012.

[57] [Coe 2013] Coe David J., Kulick Jeffrey H., “A Model-Based Agile Process for DO-178C Certification”, World Congress in Computer Science, Computer Engineering, and Applied Computing, Las Vegas, USA, 2013.

[58] [Krüger 2019] Krüger G., “Agile for Software Development: Safety Critical-Environments”, November 27, 2019, consulted June 3rd 2020, https://www.perforce.com/blog/alm/agile-software-development-safety-critical-environments

[59] [Gardner 2020] Gardner P., “Agile methods and safety critical software – Are they compatible?”, Adacore, consulted June 4th 2020, https://fr.slideshare.net/AdaCore/agile-methods-and-safety-critical-software-peter-gardner

[60] [Van Schooenderwoert 2018] Van Schooenderwoert N., Shoemaker B., “Agile Methods for Safety-Critical Systems: A Primer Using Medical Device Examples”, June 2018.

[61] [Vuori 2011] Vuori M., Agile Development of safety-critical software, Tampere University of Technology report 14, Tampere, 2011.

[62] [Hanssen 2018] Hanssen G., Stålhane T., Myklebust T., “SafeScrum® – Agile Development of Safety-Critical Software”, 2018.