Parameters for Efficient Software Certification

Roland Wolfig, e0327070@student.tuwien.ac.at

Vienna University of Technology, Real-Time Systems Group

1 Abstract

Software certification is a common approach in the aerospace domain to ensure safety, quality and dependability. Because of the effort which is several times higher than for general software, an efficient approach is very important.

This paper is based on a survey research which was supported by 30 experts of software development and certification in the aerospace domain. It provides data about the most important and critical parts of a software and certification process, how this parts can be influenced and the amount of effort which may be saved by an efficient process.

A rather new approach which is also considered is modular certification according to DO-297, which is a prerequisite for certification of integrated architectures. The data which has been collected also deals with a change of the system architecture from a federated to an integrated one. Based on the topicality of this approach, assumptions about wins or losses are rather hard to make.

2 Introduction

Software quality is a widely discussed topic, especially in the aerospace domain where a fault in the software may lead to catastrophic behavior. Therefore several approaches have been developed to ensure software quality aerospace software.

The most common approach is certification according to DO-178B, which defines guidelines for different phases of a software development project. The main idea of such process definitions is to have a controlled way for the planning, the design, the implementation and the verification of software where all steps are reasoned by at least two persons, the executing one and the reviewing one. This effort provides confidence in understanding the system behavior under several conditions.

DO-178B [1] is a means of compliance with airworthiness requirements [3], [4] which is acceptable for the aviation authorities and therefore is treated like a standard. Every process which has been chosen to provide evidence has to ensure compliance with the objectives of DO-178B.

A rather new guideline for the certification of integrated modular avionics is DO-297 [6], which considers the certification of integrated systems and architectures. This approach, called modular certification, allows the independent development and certification of aircraft functions and their corresponding integrated architecture which hosts these safety-relevant applications.

A major problem for such a certified software project is the effort which is several times higher than for a common software project. Based on this and the fact that an authority has to check the system and the used processes too, the costs for such a project are increasing with every software level. Especially for Level A certification, which is the highest criticality class in DO-178B, the effort is up to eight times higher than for the same software development without certification [9].

According to the high amount of certification costs and effort, it is interesting to know which are the most critical parts in the development and certification process and which effort may be saved with an efficient process. This would allow to improve the planning, reduce design faults in the beginning and minimize the risks of such a project.

Therefore this paper wants to point out the critical phases of such development processes, help to find key criteria for saving effort and name efficient ways for dealing with these steps.

3 Software Certification - DO-178B

RTCA DO-178B [1] is the certification standard which is used for commercial aerospace software. It defines guidelines for the processes which have to be used for the development of safety-relevant software.

Based on the objectives defined in the standard a process has to be established which ensures that the software is sufficiently safe and understood. Sufficiency is defined according to software levels, which are defined by the circumstances and the failure conditions of the software [5]. The software levels reach from Level A for catastrophic to Level D for minor failure conditions.

For a detailed description of the failure conditions and the corresponding process objectives, please refer to Table 1.

Failure Conditions	**Software Level**	**Process Objectives**
Catastrophic	A	66
Hazardous	B	65
Major	C	57
Minor	D	28

Table 1: Software Levels and Objectives

According to these software levels and their objectives, the development process has to be defined. Therefore, the following stages of the project have to be considered in advance:

– Planning Process

- Development Process
  - Requirements Process
  - Design Process
  - Implementation Process
- Verification Process
- Configuration Management Process
- Quality Assurance Process
- Customer and Certification Liaison

Especially the requirements and the design phase are of increased importance and should not be switched into the implementation phase too early. The requirements and the design should clarify the most likely problems and provide a solution which is clear and easy to understand. Every finding in the beginning reduces the effort for verification and rework, which may, in the worst case, be at the customer.

Figure 1 shows how the costs for a detected fault in the system increase, if it is found later in the process or even in the field [17], [16]. Another major aspect

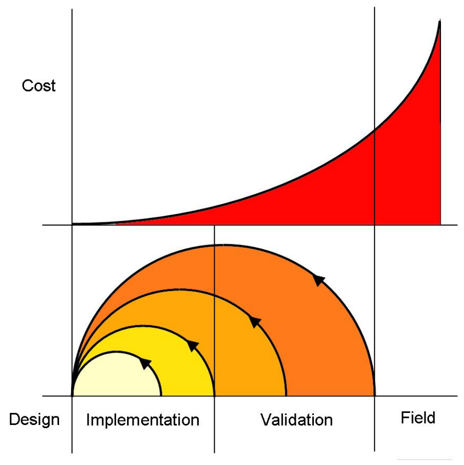

Fig. 1. Costs for Recognized Faults

for increasing quality is the feedback at the end of the project. It is not directly part of the standard but helps to improve planning estimations and to customize the process to fully comply with the structure of the project staff.

3.1 Reuse of Previously Developed Software

The reuse of already developed software and the corresponding certification package is a solution which is granted by the authority. According to the advisory circular AC20-148 [2], [10], it is possible to certify software as a reusable software component.

This approach provides the advantage that the certification effort for projects are minimized, after the completion of the first project and the corresponding approval. If this approval has been successful, the authority provides a letter of certification credit to the applicant which either reduces or eliminates the certification effort for the following projects, using the same piece of software.

The disadvantage of this approach is that the initial effort for the certification of reusable software is higher than for the common certification of an application. Furthermore, it is possible to formally or informally reuse parts of the already developed software life cycle data for follow-up projects including the same source code. Therefore a certification according to AC20-148 is rarely done.

The additional effort for the certification of a reusable software component is based on additional coordination, extended reviews and analysis at the developer, integration effort with the corresponding application, RSC change and post certification issues, document retention and increased effort in tracking compliance.

Certification for a reusable software component makes only sense, if the software does never change and is used very often, like control algorithms for example. It has to be kept in mind that the reduction provided by this approach starts the second time the software and the certification is used.

COTS Components: Commercial-Off-The-Shelf (COTS) software [7] is a special case of previously developed software, as it may get developed independently from the corresponding application. Operating systems, for example, may be considered COTS, providing certification credit like service history and a development life cycle.

As already mentioned, the COTS software needs to have the same software level as the application it supports. Furthermore, service history provides additional credit but does not preclude the presence of unintended functionality and therefore does not satisfy coverage objectives. Based on these constraints, a software life cycle data is needed for COTS components too.

4 Modular Certification - DO-297

Modular certification according to DO-297 [6], [13] is a rather new approach based on the need of certification for integrated modular avionics and the corresponding system architectures [15], [11].

The standard breaks down the whole system into the following levels to map the modular approach:

Module Acceptance: A module is a component or a collection of components which may be software, hardware or a combination of both which provides resources to the application and/or the system platform.

Application Acceptance: An application is based on modules and performs a function.

System-level Acceptance: The system-level consists of one or several platforms which provide a computing environment, managing resources for at least one application. Furthermore, it establishes support services and platform-related capabilities like health monitoring and fault management.

Aircraft-level Acceptance: The aircraft-level considers the integration of the system into the aircraft and its systems.

The standard defines an architectural approach which enables the certification of small, reusable modules and applications. The needed functionality is established by connecting the single parts of the distributed application with a communication system.

Using such an architectural approach forces the reuse of legacy systems and provides the possibility of modular platforms [14]. Therefore the certification activities have to consider the certification of modules and especially their integration into the platform. An interesting approach considered for the future is to use formal methods to verify the integration.

The certification of single modules in this approach is fairly similar to the certification effort needed for reusable software components. Therefore the reduction of certification effort is firstly given at the second use too. Furthermore, the communication system which connects the modules needs to be fully approved.

5 Parameters for Efficient Software Certification

5.1 Efforts and Savings

Based on the results of the survey and valuable data from several certified software projects [12] the efforts for the software development have to be divided according to Figure 2 on the following page including validation steps for each of the phases.

The key phases, where most of the efforts may get saved, are the requirements and the verification phase.

The efforts which are saved in the requirements and design phase may be achieved by increased attention, which means that the implementation is not started too early, and tool support, which provides a structured environment for creation and validation of the system design.

In the verification phase most of the effort may be saved by using an automated test suite. According to the costs of such a suite an appropriate solution has to be considered which is suitable for both current and upcoming projects.

5.2 Requirements, Design and Traceability

The requirements and design phases at the beginning are the most important parts in the software life cycle process. The requirements define what the output

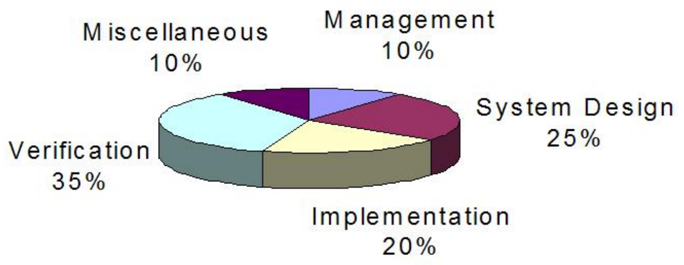

Fig. 2. Efforts of Software Development Process

have to be and therefore need to be clear and easy to understand. The design is derived from the requirements and describes how they should be implemented. Every fault or obscurity in this phase has much impact later on.

Requirements are the building blocks of the system. Therefore, the quality of the system depends on the quality of every single requirement. The design consists of detailed requirements, which have to be traceable to the high-level requirements mostly, and design components which describe complex algorithms and data structures to support the understanding. Another major point is the traceability from the requirements to the design and further on to the test cases, down to the source code. This ensures that nothing is missing and everything has a reason for existence. To ensure a constant quality level for the requirements and guarantee traceability through the process, some basic points have to be recognized.

Tool Support: A database-centric requirements management tool provides a lot of advantages to the development and certification process. Firstly, several process steps are already included in the tool and therefore the formal handling is simplified. Furthermore, the waterfall-based top-down life cycle process may be split up which allows moving forward from requirements to implementation and verification without the need of showing consideration for other parts of the system. Furthermore, such a tool checks that all relevant traceability information is available. Additionally, some of these tools provide the possibility of creating an evidence media which contains all necessary life cycle and traceability information in an easy-to-review form. According to this efficient way to deal with the process and based on the experts judgment, the effort for these steps may be optimized between 2% and up to 20% with respect to the process used before.

Standardized Requirements Definitions: There should be standards for requirement definitions which provides guidelines for the development to support the easy understanding of them. Furthermore, each requirement has to be self contained because this supports the verification of each requirement.

Design Components: If the requirement describes complex functionality, the developer should add definitions, figures and additional information which support the understanding. This encourages the demand for self-containing requirements and helps to comprehend the whole system.

Testability: The requirement has to be checked for testability. This has to be done by the requirement developer and especially by the reviewer. The easiest way to handle this is to write functional test cases in parallel to the requirements to find testability problems at an early stage of the requirements process. If this is not possible, the developer should at least give advice, regarding what to test, to the verification staff for efficient verification.

I conclude that, based on the experts judgment and own experience and research, considering these points in the requirements and design phase may reduce the effort and the costs for the overall project of up to 30% because a clear system design reduces the effort for implementation and verification.

5.3 Verification and Validation

The verification phase is commonly the phase of the software life cycle where most effort is needed. Based on estimations, the effort for this part of the software life cycle is up to 50% of the overall project effort.

Irrespective of the used tool, the verification effort may be minimized if the requirements, the design and the implementation are kept simple and clear (see also 5.2 on page 5). In this case, every function may be tested by its own and additional high-level tests, which verify the required functionality, complete the test of the whole system. Additionally, some robustness test cases are needed, where boundary values are used as input.

Another very important part is the validation phase. These reviews ensure the quality of every generated artifact and provides the transitions to the next phases. As already mentioned, especially higher effort for requirements and design review pays off later on.

According to these facts, there are several points to consider which help to get these steps done efficiently.

Test Suite: A major concern regarding the verification process is the use of a test suite. The advantage of such a tool is the possibility to automatically verify test cases and structural coverage. Considering the tool qualification package which has to be provided to the authority is already available which is most common for suitable tools, the verification effort may be optimized between 5% and up to 20% based on the process used before. These numbers are outcome of the research survey.

Verification Tools: The use of verification tools is an interesting aspect of DO-178B. It provides the possibility of getting complex algorithms, like schedulers, easily certified. The verification tools have to verify the results of these algorithms, to prove their safe and deterministic behavior. Furthermore, a tool qualification package is needed for the verification tool which provides confidence regarding the tool. The verification tool and its tool qualification package are mostly less expensive, if the verification for correctness have to be done several times, than to verify the algorithm itself.

Review Criteria and Checklists: The criteria for reviews should be clear, suitable, applicable, detailed and have to be stated on the corresponding checklist. Furthermore, sometimes it is useful to have detailed checklists, which name every single review item and every corresponding review criteria, including additional information on what and how to review the items. This increases the quality by increased efficiency.

Informal Reviews: Informal reviews of the life cycle artifacts, disclose major problems and increase the quality in an early stage. The overhead of a formal review process for immature artifacts is minimized and the necessary evidence is provided in a formal, second stage. This approach provides the same quality, like doing every review in a formal manner, but more efficiently.

According to these points, I conclude that the reduction of effort and therefore costs are up to 15% of the overall project, if the verification and validation is done in an efficient way.

5.4 Optimized Processes and Continuous Process Improvement

A well-set up process and its continuous improvement are key tasks for the development of high-quality software. There are several points to consider which help to optimize the process and therefore increase the quality and efficiency [8].

Quality Culture: Establishing a quality culture is a major issue considering safety-critical software. It takes some time to train, introduce and establish the processes and customize the work flow but finally it is an opportunity for the whole software development. This cultural approach increases the quality and efficiency considerably, if everybody who is involved feels responsible for producing high quality artifacts. The surveys' outcome is that, once established, a quality culture decreases the effort of the whole software development process by 5% to 20%.

Continuous Process Improvement: The defined and used process should not be written in stone. It has to be customized if a better approach is found or new tools are used. After the end of the project, a feedback loop has to be installed to verify the changes for their positive or negative influence. According to experts' judgment, if continuous process improvement has strong management support, possible improvements may be up to 20% in three years, depending on the maturity of the current process.

Training: Training ensures that every person, involved in the process, works in the correct manner to support the process and therefore forwards the project with the needed quality. Ensuring constant high quality, every person involved should get customized training according to their special needs. Based on my research, appropriate training increases the quality and therefore the efficiency by 2% to 10%, depending on the experience of the staff.

Concluding these points, I am of the opinion that optimized processes does not only save effort up to 15% in a single project, but also increase the quality in a sustainable manner on the long term.

6 Conclusion

There are several possibilities for efficient software certification by modifying and optimizing the process, considering several criteria described. These criteria name the phases of the process, where most of the effort is needed and especially where most of the effort may be gained.

The key aspect is that the additional effort which is spend on the requirements and design phase in the beginning pays off for the software development process and the whole product lifetime. Furthermore, it is possible to save a lot of effort in the verification phase using appropriate tools.

Another possibility to reduce the certification effort is an architectural approach according to DO-297. This modular approach deals with integrated architectures, enables a simplified development process for distributed systems and therefore allows a significant reduction of the certification effort, for single applications. Furthermore, all parameters for efficient software development and certification are also suitable for this modular approach, because the development process of a single component is very similar to the process defined in DO-178B.

All parameters which are presented in this paper are based on research and assume the saved effort comparing a new process to a well-established one. To achieve these goals may take some time but it will increase efficiency of development and the quality of the product.

Acknowledgment: I want to thank everybody who contributed to the survey with time, effort and knowledge, especially the DO-178C working group, where most of the input came from.

References

[1] RTCA: DO-178B: Software Considerations in Airborne Systems and Equipment Certification. Radio Technical Commission for Aeronautics, Inc. (RTCA), Washington, DC, 1992.

[2] Federal Aviation Administration: AC20-148: Reusable Software Components, 2004

[3] Federal Aviation Administration: FAR23: Airworthiness Standards: Normal, Utility, Acrobatic, and Commuter Category Airplanes, 1965 - 2006

[4] Federal Aviation Administration: FAR25: Airworthiness Standards: Transport Category Airplanes, 1965 - 2004

[5] SAE - Systems Integration Requirements Task Group: ARP 4754 - Certification Considerations for Highly-Integrated or Complex Aircraft Systems, 1996

[6] RTCA: DO-297: Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations . Radio Technical Commission for Aeronautics, Inc. (RTCA), Washington, DC, 2005.

[7] Federal Aviation Administration: DOT/FAA/AR-01/26: Commercial Off-The-Shelf (COTS) Avionics Software Study, 2001

[8] Frazer, K.: Return on Software Process Improvement, 1997.

[9] Hilderman, V.: Certifying an RTOS to DO178B: Tips & Tales, 2001.

[10] Leanna Rierson: Software Reuse in Safety-Critical Systems. Masters Thesis - Rochester Institute of Technology, 2000.

[11] Ali Bahrami: Complex Integrated Avionic Systems and System Safety. Europe/U.S. International Aviation Safety Conference, 2005.

[12] Groessinger, P.: Software certification for a time-triggered operating system, 2005.

[13] Rushby, J.: Modular Certification. Technical report, Computer Science Laboratory SRI International, 2001.

[14] Kopetz, H., Obermaisser, R., Peti, P., Suri, N.: From a federated to an integrated Architecture for dependable real-time Embedded Systems, 2004.

[15] Gruber, M.: DECOS Project Proposal. Austrian Research Center, Vienna, Austria, 2004.

[16] Poledna, S.: TTP-Tools - The tool set of the Time-Triggered Architecture. The Monterey Workshop Series, 2004.

[17] Boehm, B. Software Engineering Economics. Prentice Hall, 1981