Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR TP 98037

Reliability, maintainability and safety applied to a real world avionics application

E. Kesseler and E. van de Sluis

DOCUMENT CONTROL SHEET

	ORIGINATOR'S REF. NLR-TP-98037		SECURITY CLASS. Unclassified

ORIGINATOR
National Aerospace Laboratory NLR, Amsterdam, The Netherlands

TITLE
Reliability, maintainability and safety applied to a real world avionics application

PRESENTED AT
the European Safety and Reliability Conference 1998, Trondheim, Norway, 17-19 June 1998

AUTHORS	DATE	pp	ref
E. Kesseler and E. van de Sluis	Jan. 1998	17	8

DESCRIPTORS

Airborne equipment	Pilot performance
Avionics	Program verification (computers)
Certification	Quality control
Computer programming	Reliability analysis
Failure analysis	Software engineering
Flight safety	Software reliability
Maintainability	Workloads (psychophysiology)

ABSTRACT

NLR currently co-develops a civil avionics application that connects aircraft subsystems with the aircraft flight deck by means of modern digital data buses. It combines, controls, processes and forwards the data between the subsystems and flight deck. High reliability of these functions is required to ensure the safety of the aircraft. For this kind of applications certification is required by an independent, government authorised, third party. Guidelines for this certification are documented in the
DO-178B standard: software considerations in airborne systems and equipment certification.
This paper describes practical experiences with DO-178B. The software development methodology used to guarantee the reliability, maintainability, safety and cerifiability of the product in a commercial environment is discussed. Requirements volatility and short time-to-market characterise the commercial reality.

Abstract

NLR currently co-develops a civil avionics application that connects aircraft subsystems with the aircraft flight deck by means of modern digital data buses. It combines, controls, processes and forwards the data between the subsystems and flight deck. High reliability of these functions is required to ensure the safety of the aircraft. For this kind of applications certification is required by an independent, government authorised, third party. Guidelines for this certification are documented in the
DO-178B standard: software considerations in airborne systems and equipment certification.
This paper describes practical experiences with DO-178B. The software development methodology used to guarantee the reliability, maintainability, safety and certifiability of the product in a commercial environment is discussed. Requirement volatility and short time-to-market characterise the commercial reality.

Contents

3 figures

(17 pages in total)

1 Introduction

To fly aircraft under all (adverse) conditions, pilots must fully rely on the data presented to them, and on the correct and timely forwarding of their commands to the relevant aircraft subsystems. The avionics application connects these subsystems with the aircraft flight deck by means of modern digital data buses. It combines, controls, processes and forwards the data between the subsystems and the flight deck. High reliability of these functions is required to ensure the safety of the aircraft. In this paper the experiences with the software development methods to meet these requirements in a commercial environment are presented.

2 Air transport safety requirements

For safety critical software in airborne equipment the (DO-178B 1992) standard has been developed. The aim of this document is to provide guidance to both the software developers and the certification authorities. Usually acceptance of software is based on an agreement between the developer and the customer. In civil avionics an independent third party, the certification authority, performs the ultimate system acceptance by certifying the entire aircraft. It is only then that the constituent software is airworthy and can be considered ready for use in the aircraft concerned. (DO-178B 1992) provides a world wide "level playing field" for the competing industries as well as a world wide protection of the air traveller, which are important due to the international character of the industry. The certification authority is a national governmental institution which in our case delegated some of its technical activities to a specialised company.

2.1 Safety classification

Based on the impact of the system failure the software failure can contribute to, the software is classified into 5 levels. The following is a verbatim copy of the (DO-178B 1992) text. The failure probability in flight hours (i.e. actual operating hours) according to the Federal Aviation Requirements /Joint Aviation Requirements(FAR/JAR-25) has been added.

Level A: Catastrophic failure
Failure conditions which would prevent continued safe flight and landing
(FAR/JAR-25) extremely improbable, < 1x10-9

Level B: Hazardous/Severe-Major failure
Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be:
- a large reduction in safety margins or functional capabilities
- physical distress or higher workload such that the flight crew could not be relied on to perform their tasks accurately or completely
- adverse effect on occupants including serious or potentially fatal injuries to a small number of those occupants

(FAR/JAR-25) extremely remote, 1x10-9 < hazardous failure < 1x10-7

Level C: Major failure
Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example,
- a significant reduction in safety margins or functional capabilities
- a significant increase in crew workload or in conditions impairing crew efficiency or

- discomfort to occupants, possibly including injuries

(FAR/JAR-25) remote, 1x10-7 < major failure < 1x10-5

Level D: Minor failure

Failure conditions which would not significantly reduce aircraft safety and which would involve crew actions that are well within their capabilities. Minor failure conditions may include for example,

- a slight reduction in safety margins or functional capabilities
- a slight increase in crew workload, such as, routine flight plan changes, or some inconvenience to occupants

(FAR/JAR-25) probable, minor failure > 1x10-5

Level E: No Effect

Failure conditions which do not affect the operational capability of the aircraft or increase crew workload.

The following text will only consider the part of the avionics application which is classified as level A.

2.2 Software life cycle

(DO-178B 1992) on purpose refrains from making a statement about an appropriate software life cycle. The life cycle is described rather abstract as a number of processes that are categorised as follows

- software planning process which entails the production of the following documents
    - plan for software aspects of certification. The main purpose of this document is to define the compliance of the software development process to (DO-178B 1992) for the certification authorities. This document contains many references to the project documentation generated as part of the life cycle model used,
    - software development plan, which defines the chosen software life cycle and the software development environment, including all tools used,
    - software verification plan, which defines the means by which the verification objectives will be met,
    - software configuration management plan
    - software quality assurance plan.
- software development processes consisting of
    - software requirement process,
    - software design process,
    - software coding process,
    - integration process.

Each software development process has to be traceable, verifiable and consistent. Transition criteria need to be defined by the developer to determine whether the next software development process may be started.

- integral processes which are divided into
  - software verification process,
  - software configuration management process,
  - software quality assurance process,
  - certification liaison process.

  The integral processes are a result of the criticality of the software. Consequently the integral processes are performed concurrently with the software development processes throughout the entire software life cycle.

2.3 Verification

Verification is defined as "the evaluation of the results of a process to ensure correctness and consistency with respect to the inputs and standards to that process". Verification can be accomplished by review, analysis, test or any combination of these 3 activities. Review provides a qualitative assessment of correctness.

Analysis is a detailed examination of a software component. It is a repeatable process that can be supported by tools. (DO-178B 1992) recognises two types of tool

- software development tools, which can introduce errors,
- software verification tools, which can fail to detect errors.

The avionics project has only developed software verification tools. Every tool needs to be verified against the Tool Operational Requirements, the contents of which is prescribed in (DO-178B 1992). Software development tools need to be tested using normal and abnormal conditions. Software verification tools need only be tested using normal conditions. For software tools the same documentation and configuration control procedures apply as for the airborne software. Every software tool needs approval of the certification authority.

Testing is "the process of exercising a system or system components to verify that it satisfies specified requirements and to detect errors". By definition the actual testing of deliverable software forms only part of the verification of the coding and integration processes.

3 Experience gained with safety critical software development

Usually the software development process is agreed between the customer and the supplier. For certifiable software a third party is involved, adding a stage in the approval process. The organisational independence improves the position of the assessors. In the our case the customer had ample experience with (DO-178B 1992) certification and decided, after approving the process documentation, to postpone the review with the certification authorities until the completion of the coding process. Only minor modifications were needed in the process documents, implying that (DO-178B 1992) can be adhered to without prior knowledge of certification.
The project team was set up consisting of 2 separate groups, a development group and a verification group. The verification group was headed by a team member with sufficient authority to report, at his own discretion, to the company management outside the project hierarchy.
To ensure strict traceability from requirements to design, to code and to integration a review was development process. Experience with previous mission critical software development suggested variability of detailed system requirements, so analysis is used wherever possible. Part of the analysis can be strictly defined and subsequently implemented in a customised tool. Tool support reduces the costs for repeated analysis. The software verification tools performed according to expectations to reduce the impact (both in time and costs) of the many late requirements changes (fig 2).

The customer required use of the C programming language was considered a potential risk for the successful development of the avionics application. The C language contains numerous constructs that are unspecified, undefined or left to be defined by the compiler supplier (Hatton 1995). This risk was reduced by choosing an ANSI-C compliant compiler complemented by a project coding standard defining, amongst others, a safe subset of C. Compliance to this project coding standard can be checked automatically by customising a commercial tool. During verification of this tool the version management by the tool supplier turned out to be inadequate. The tool was already marketed world wide since 1986 to hundreds of customers. This illustrates the rigour of the applied verification processes.

4 Overview of the avionics application

The aircraft display subsystem is designed to operate in both Visual Meteorological Conditions (VMC) and Instrument Meteorological Conditions (IMC). Under visual meteorological conditions the displays aid the pilot during flight, under instrument meteorological conditions the displays are necessary for the pilot to be able to fly, consequently the correct functioning of the displays is safety critical. The latter conditions imply that a number of equipment items needs to be duplicated to achieve the required failure probability.

When configured for instrument meteorological conditions the displays consist of the following equipment

- 2 avionics control display application systems, containing the avionics application,
- 4 smart multifunction displays,
- 2 instrument control panels,
- 1 reconfiguration control unit.

The avionics application is the interface between the on-board sensors and the displays (fig. 1). The sensors and some aircraft subsystems send flight parameters via digital buses to the avionics application, which validates the parameters and sends them to the displays. A number of flight parameters is also computed within the avionics application itself.

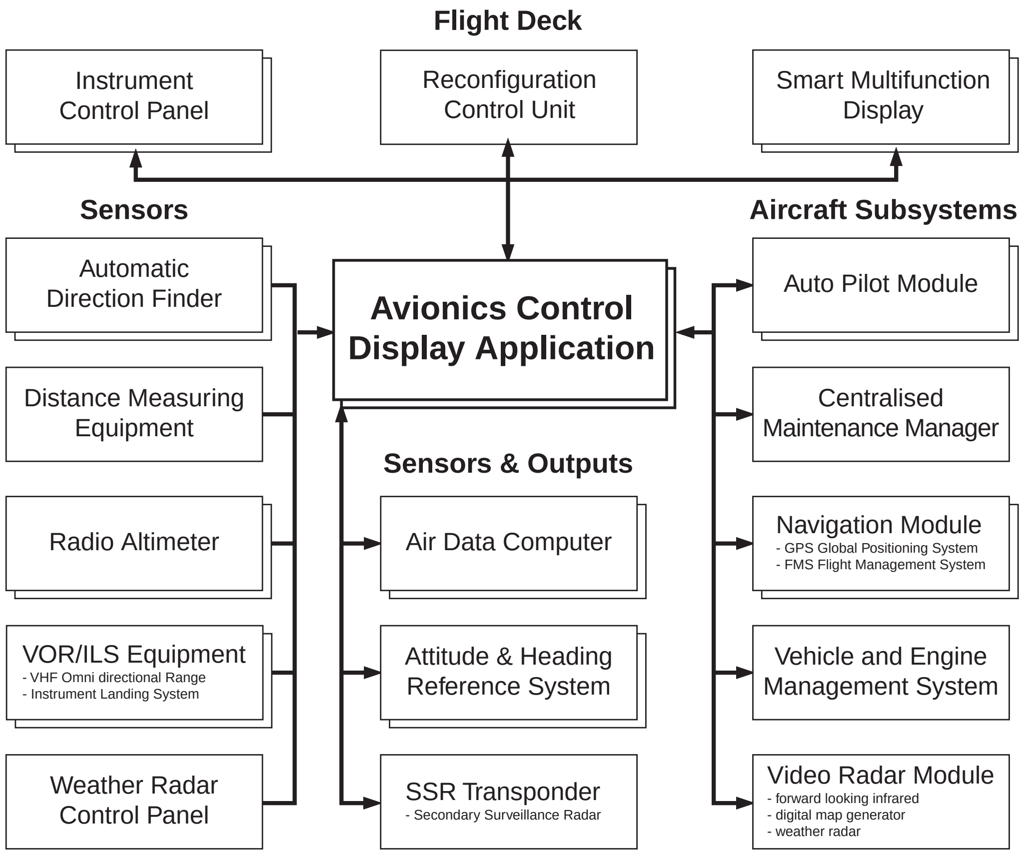

Fig.1 Overview avionics control display application environment

During normal operation the avionics application processes about 100 different flight parameters, In case of failure of an equipment item or a discrepancy between two sensors, the reconfiguration control unit permits the crew to choose between different display configurations. When a sensor is reconfigured, it is logically switched-off. This illustrates how software and a duplicated hardware device reduce the failure rate to the required level. Consequently the software becomes safety critical.

During normal operation the avionics application processes about 100 different flight parameters, originating from 10 different sensors. The delay times within the avionics application should be guaranteed to be less then 30 msec.

Each parameter is classified as

- critical: loss or undetected error could lead to a catastrophic failure condition. Examples of critical parameters are the attitude parameters pitch, roll, and heading. The software that handles these parameters is classified as (DO-178B 1992) level A,
- essential: loss or undetected error could lead to a major failure condition. An example of an essential parameter is the VOR (VHF Omnibearing Range for aircraft position determination). The software that handles these parameters is classified as level B,
- non-essential: loss or undetected error could lead to a minor failure condition. Examples of these parameters are the long term navigation parameters, like the flight plan. The software that handles these parameters is classified as level D.

Depending on the criticality of the flight parameter, validation is performed by the avionics application in four different ways

- coherency test: a check on correct length and parity of the data,
- reception test: a check on the timely arrival of the data,
- sensor discrepancy test: a comparison between two data values produced by the two independent redundant sensors,
- module discrepancy test: a comparison between the two parameter values produced by the same sensor; one value directly read by the system from the sensor, and one obtained from the redundant system via a cross-talk bus.

5 Experience gained with safety critical software development methods

The definition of the avionics application software development method has been guided by previous experience with mission critical software. In spacecraft the software on which success of a mission depends is classified as mission critical.

5.1 Previous experience

The Attitude and Orbit Control System (AOCS) software for the Italian-Dutch Astronomical X-ray Satellite (SAX) (Dekker 1996) has been developed using the following software development method

- customer supplied specifications provided in plain English,
- use of the (ESA PSS-05, 1991), life cycle model,
- requirement analysis using Structured Analysis with Hatley and Pirbhai Real Time system extensions (SA/RT) (Hatley & Pirbhai 1988) supported by the Teamwork tool. The process-specifications are written in plain English, including a copy of the relevant requirement number(s),
- software design using Yourdon Structured Design (SD) supported by the Teamwork tool. The module-specifications are written in pseudo code and include a copy of the relevant requirement number(s),
- coding in the customer prescribed C-language. A proprietary C-coding standard was used, enhanced for this specific project. The coding standard provides a uniform coding style, improving readability, maintainability and modifiability. Based on experience in many project recommended constructs are provided an unwanted constructs are prohibited. Project enhancements include no recursion (to prevent unpredictable maximum execution times) and no dynamic memory allocation (to prevent unpredictable maximum memory size).
  The entire structured design module- specification was included as comment in the code,
- module testing and integration testing with a self imposed 100% code coverage requirement.

After validation and delivery the resulting system contained 1 error in 20,000 lines of non-comment source-code. This error was found during the SAX satellite integration tests plus the entire operational life of the satellite. The resulting error density is 0.05 error per 1000 lines of code. This can be categorised as an extremely low value, refer also to (Hatton 1996). This error density was achieved even though the first delivery consisted of 16,000 lines of code and subsequently about 8,000 lines of code were added/modified resulting in a total size of 20,000 lines of code.

5.2 Method used

For the avionics application the customer prescribed the use of the (DOD-STD-2167A, 1988) life cycle model and the use of the C-language. Based on the successful SAX AOCS development the following elements of the SAX AOCS software development method are retained

- customer supplied specifications provided in plain English,
- requirements analysis using Structured Analysis/Real Time systems supported by the Teamwork tool,
- software design using structured design supported by the Teamwork tool,
- use of NLR proprietary C-coding standard, with project specific enhancements, including the ones described for the SAX AOCS project. The C-coding standard is enforced by a static source code analysis tool.

Based on the SAX-AOCS experience of a very substantial amount of changes during and after the implementation phase, even more emphasis is placed on tools to support the development activities. Added to the software development method are

- a mandatory 100% code coverage for software classified at (DO-178B 1992) level A. This code coverage consists of statement coverage (every statement executed) plus decision coverage (every decision executed for pass and fail) plus the modified condition/ decision coverage (mc/dc). Mc/dc requires that for every condition in a decision, its effect on the outcome of the decision is demonstrated,
- the use of an automated test tool to aid the construction and cost effective repetition of the functional tests and code coverage tests. Only for code coverage tests the source code has to be instrumented by the test tool,
- execution of module tests and integration tests on the target system. The test tool is used to generate test harnesses in the C programming language, which can be (cross-)compiled and run on the host computer or the target computer. The advantage is that the source code can be tested without having the target computer available.

5.3 Formal methods

Formal methods with comprehensive automated environments could provide benefits in safety critical environments. However such automatic verification systems, including sufficient vendor support, are not yet available for real world applications. Formal methods can provide important evidence for certification. However certification must consider multiple sources of evidence and ultimately rests on informed engineering judgement and experience (Kopetz, 1997). (DO-178B 1992) does not consider formal proofs an alternative to the recommended methods, but allows its use for satisfying some of the documents objectives.

6 Commercial realities versus safety critical application development

Due to the commercially defined short time to market, the customer definition of the system requirements was performed concurrently with the software requirements process. The resulting analysis was subjected to a number of informal technical assessments, but no formal verification was performed.

6.1 Co-development necessity

The commercial nature of the aircraft development even resulted in concurrent updates of the system requirements during the design, coding and integration processes. Consequently the planned deployment of separate development and integration teams turned out to be infeasible.
To aid the integration of the avionics application in the customer developed displays and subsequently in the existing aircraft, a first version of the software with very limited functionality was delivered. This version was produced based on a successive completion of the documented software development processes. However none of the formal reviews with the customer or the certification authority had been performed. The first version served its purpose well. A lot of feed-back was obtained, resulting in many changes to and clarifications of the system requirements.
Due to the success in eliminating system level problems by the informal co-development of the first version of the avionics application and the displays, the customer requested to continue the informal co-development and allocate all project resources to it. The personnel resources of both teams were combined, however the 2 separate team managers with their complementary responsibilities remained. All activities were executed for only one of the teams. The respective team leader ensure that the relevant procedures remain strictly enforced.

6.2 Requirement volatility

The steady rise in the number of implemented requirements (fig 2) shows that from a functional point of view this concentrated development effort has been very successful. At least 1 additional pre-release is expected. Up to date the software contains nearly all functions for the nominal behaviour. At the same time the number of requirements changes, almost all of which have been accommodated, equals the number of requirements (fig 2). These changes relate in part to valuable feed-back from the user (pilot). Most remaining changes are caused by the co-development of the displays and especially the integration of the displays with the avionics application and the aircraft. This integration has been expedited considerably by the pre-releases.

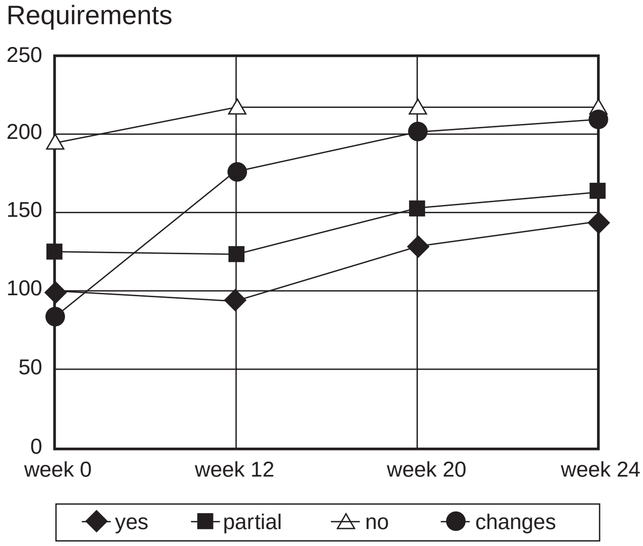

Fig. 2 Requirement implementation status (cumulative) and number of changed requirements

6.3 Incremental deliveries

The avionics application consists of 4 configuration items, 3 programme modules and 1 shared library module. By far the most requirements are implemented in Module C. Between the first and the second delivery many functions from module B were moved to the shared library (fig. 3) because of modified requirements combined with a consolidation of the design of both modules. Between the second and third delivery the same occurred for modules A and B/C. Between the third and the last delivery mainly additional requirements are implemented in module C with some additional module consolidation. Between the second and third delivery, the execution time was considerably reduced by optimising the architecture design.

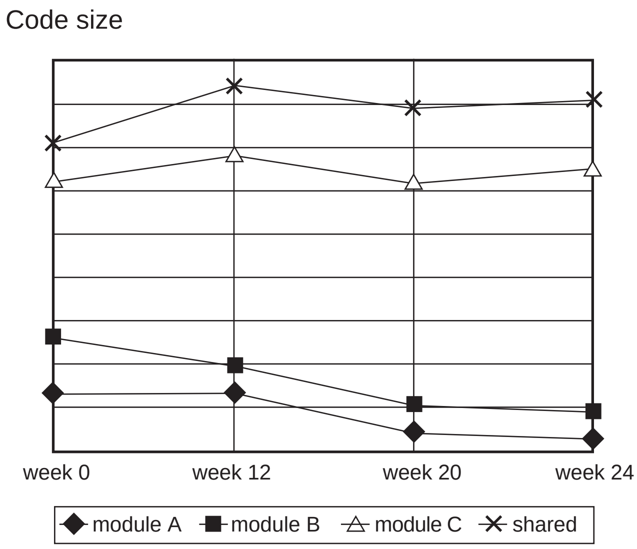

Fig. 3 Module code size per pre-release, cumulative

6.4 Co-development experience

This informal co-development has only been possible because the documented software requirement process and software design process had been completed before the coding of the first software version started. The available Teamwork models also aided in assessing the consequences of proposed changes. The drawback of the informal co-development is that a very considerable amount of documentation work remains. Based on the software size it was impossible to enlarge the team. Also all verification and the exhaustive mc/dc testing still needs to be performed. It is inevitable that the verification will result in a new version of the software, which will be submitted to the certification authorities. The reverse side of the early and successful delivery of the co-development versions is the risk of invalidating some already completed flight trials of the aircraft.

Safety critical avionics applications require independent personnel to verify the coding and integration processes. Consequently another person needs intimate knowledge of each module as well as an up-to-date detailed design to verify the implementation. The commercial pressure to implement requirements in the next pre-release, combined with the labour intensive Teamwork tool to update the analysis and design models results in both models to become rapidly outdated. Consequently the verification can not keep up with the co-development.

After the last pre-release delivery costly re-work needs to be done, which also delays the certification schedule. It is unclear how much of the schedule time gained during the co-development is lost due to the resulting delayed verification and certification. At least co-development saves re-certification effort as well as the generation, formal release and formal review of much documentation describing pre-releases. An important lesson learned from the informal co-development is to try to keep the verification process up with the actual implementation to comply with the commercial time to market.

The many system requirement changes require a cheap and easily repeatable verification process. This can only be achieved by using strictly defined development methods which allow strictly defined analysis. The well defined analysis should be executed by automated tools. These tools should be sufficiently user-friendly and efficient to allow the analysis, design and testing to be updated concurrently with the code modifications resulting in a spiral development model. As a complete integrated suite of development tools is not commercially available, the best option is to use as much available tools as possible. For some simple unsupported (verification) tasks proprietary tools can be produced cost-effectively. Only the tool for checking compliance to the coding standard was sufficiently user-friendly to be used during the co-development.

7 Conclusion

For air transport the safety requirements stated in (DO-178B 1992) software are sufficiently clear to allow a first-time developer to define, without external support, a compliant development process. Compliance can be achieved with the traditional waterfall software development model. Producing aircraft is a commercial venture i.e. the various aircraft subsystems need to be co-developed in order to achieve the commercial determined time to market. In this environment the spiral model is more appropriate then the waterfall model.
An integrated tool set is needed to support the co-development i.e. which allows when a change occurs to concurrently update analysis, design, code, integration and verification (including traceability information). Currently commercial available tools do not provide this capability.
To minimise the effort of the recurring verification, analysis is the preferred method, supported by tools wherever available. For simple verification tasks customised tools can be developed cost-effectively.
The commercial need to deploy all human resources to development has significantly reduced the development time as well as allowed co-development of the avionics application with several other aircraft subsystems.
It can not be assessed whether this time reduction is offset by the resulting delay in updating the documentation and certification. Performing these iterations before certification at least avoids the re-certification costs of the intermediate versions.

References

Dekker, G.J. 1996. *Development procedures of the on-board attitude control software for the SAX satellite*, NLR technical publication TP 96573 L

DO-178B. 1992. *Software Considerations in Airborne Systems and Equipment Certification,*
DOD-STD-a. 1988. Department of Defense (DoD) Defense System Software Development

ESA-PSS-05. 1991. *ESA Software Engineering Standards*

Federal Aviation Requirements/Joint Aviation Requirements FAR/JAR-25

Hatley, D.J., Pirbhai, A. 1988. *Strategies for real-time system specification.*
Dorset House Publishing

Hatton, L. 1995. Safer C. Mc Gaw-Hill

Hatton, L. 1996. *Software faults: the avoidable and the unavoidable: Lessons from real systems*, Proceedings of the ESA 1996 product assurance symposium and software product assurance workshop (ESA S-377) Noordwijk

Kopetz, H. 1997. *Real time systems, design principles for distributed embedded systems.*
Kluwer scientific publishing (p 250).