Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TP-97669

Safety and commercial realities in an avionics application

E. Kesseler and E. van de Sluis

DOCUMENT CONTROL SHEET

	ORIGINATOR'S REF.		SECURITY CLASS.
	TP 97669 U		Unclassified

ORIGINATOR
National Aerospace Laboratory NLR, Amsterdam, The Netherlands

TITLE
Safety and commercial realities in an avionics application.

PRESENTED AT
the Second World Congress on Safety of Transportation, Delft, The Netherlands, 18-10 February, 1998

AUTHORS	DATE	pp	ref
E. Kesseler and E. van de Sluis	Jan. 1998	20	10

DESCRIPTORS

Aerospace safety	Program verification (computers)
Avionics	Quality control
Certification	Real time operation
Computer programming	Software engineering
Maintainability	Software reliability

ABSTRACT

To fly aircraft under all (adverse) conditions, pilots must rely fully on the data presented to them, and on the reliable and timely forwarding of their commands to relevant aircraft subsystems. The avionics application, Flight Control Display Module (FCDM), 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. The experiences with the software development methods to meet these requirements are presented.

For air transport the safety requirements are stated in DO-178b: software considerations in airborne systems and equipment certification. The main part of the FCDM software is subject to the most severe classification of DA-178B. Compliance to DA-178B is assessed by an independent, government authorised, third party. This third party issues a certificate releasing the product for operational use. The influence of these safety requirements and the independent DO-178B compliance assessment on the software development and verification methods are described. The black box has a successful application in air transport. The extension of the black box approach in FCDM is discussed.

The development of aircraft is a commercial venture. In order to meet the market demands, permanent changes occur during the sofware development proces. Many different versions of and/or extensions to the product for the various customers are required. The reliability, maintainability, safety and certifiability of the product may not be compromised. The impact of all these customisations on the development and verification methods is assessed.

General standards for safety critical software are emerging. Some differences and similarities with DO-178B are highlighted. These standards provide opportunities for general purpose products. Their possible impact on airworthy equipment is assessed using the FCDM case.

Abstract

To fly aircraft under all (adverse) conditions, pilots must rely fully on the data presented to them, and on the reliable and timely forwarding of their commands to the relevant aircraft subsystems. The avionics application, Flight Control Display Module (FCDM), 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. The experiences with the software development methods to meet these requirements are presented.

For air transport the safety requirements are stated in DO-178B: software considerations in airborne systems and equipment certification. The main part of the FCDM software is subject to the most severe classification of DO-178B. Compliance to DO-178B is assessed by an independent, government authorised, third party. This third party issues a certificate releasing the product for operational use. The influence of these safety requirements and the independent DO-178B compliance assessment on the software development and verification methods are described. The black box has a successful application in air transport. The extension of the black box approach in FCDM is discussed.

The development of aircraft is a commercial venture. In order to meet the market demands, permanent changes occur during the software development process. Many different versions of and/or extensions to the product for the various customers are required. The reliability, maintainability, safety and certifiability of the product may not be compromised. The impact of all these customisations on the development and verification methods is assessed.

General standards for safety critical software are emerging. Some differences and similarities with DO-178B are highlighted. These standards provide opportunities for general purpose products. Their possible impact on airworthy equipment is assessed using the FCDM case.

Contents

1 figure

(20 pages in total)

1 Introduction

To fly aircraft under all (adverse) conditions, pilots must rely fully on the data presented to them, and on the reliable and timely forwarding of their commands to the relevant aircraft subsystems. An avionics application which is currently being developed by NLR, the Flight Control Display Module (FCDM), 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 experience with the software development methods to meet these requirements in a commercial environment are presented. The final section highlights some differences and similarities between the emerging general standards for safety critical software.

2 Air transport software safety requirements

For air transport, apart from the normal customer-supplier relation relating to the functional requirements, the safety requirements are stated in DO-178B: software considerations in airborne systems and equipment certification [ref 1]. 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 (aircraft) acceptance by certifying the aircraft. It is only then that the software is airworthy and can be considered ready for use in the aircraft concerned. DO-178B provides a world wide "level playing field" for the competing industries as well as a world wide protection of the air traveler, 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.

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 text. The failure probability in flight hours (i.e. actual operating hours) according to the Federal Aviation Requirements/Joint Aviation Requirements FAR/JAR-25 [ref 2] 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 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 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 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 application which is classified as level A.

DO-178B 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

  The software planning process 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 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 and 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 process may be started. In case the inputs of a process are incomplete, e.g. the previous process has not been completed, transition can be allowed when the transition criteria are satisfied. Special attention needs to be paid to the verification of process inputs which become available after the subsequent process is started.

- integral processes

The integral processes 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.

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 recognises two types of tool
- software development tools, which can introduce errors
- software verification tools, which can fail to detect errors.

The FCDM project has only developed software verification tools. Every tool needs to be verified against the Tool Operational Requirements (TOR), the contents of which is prescribed in DO-178B. 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 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 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 of the project hierarchy.

To ensure a strict traceability from requirements to design, to code and to integration a review was planned after completion of each 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.

The customer required use of the C programming language was considered a potential risk for the successful application development. The C language contains numerous constructs that are unspecified, undefined or left to be defined by the compiler supplier [ref 3] 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 sold at least 5 years to hundreds of customers. This illustrates the rigour of the applied verification processes.

4 Overview of the avionics application

The flight 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 instruments are necessary for the pilot to be able to fly, consequently the correct functioning of the instruments 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 display subsystem consists of the following equipment:

- 2 Flight Control Display Modules,
- 4 Smart Multifunction Displays,
- 2 Instrument Control Panels,
- 1 Reconfiguration Control Unit.

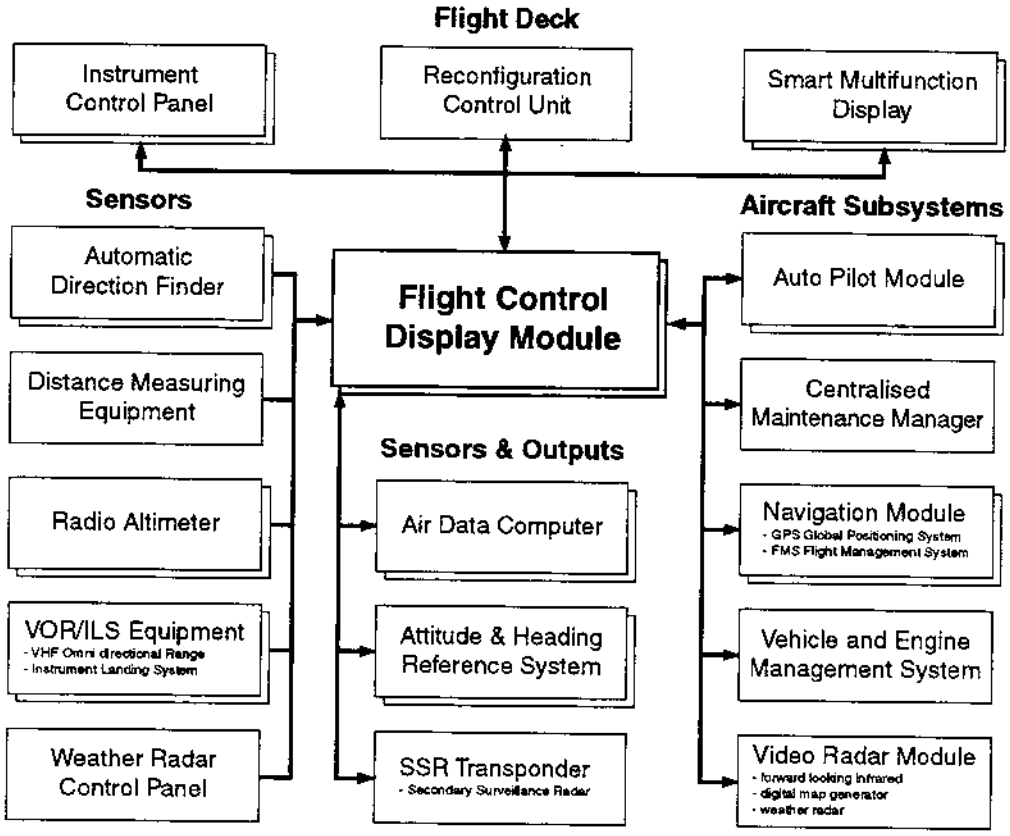

Fig. 1 Overview flight control display module environment

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

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 configurations. When a sensor is reconfigured, it is logically switched-off. This illustrates how software and a multiplied hardware device reduce the failure rate to to the required level. Consequently the software becomes safety critical.

During normal operation FCDM processes about 100 different flight parameters, coming from 10 different sensors. 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 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 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 data, validation is performed 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 parameters produced by the two independent redundant sensors,
- module discrepancy test: a comparison between two parameter values produced by the same sensor; one value directly read by FCDM from the sensor, and one obtained from the redundant FCDM via a cross-talk bus.

FCDM itself does not have a black box capability. However, since FCDM is a spider in the web of the avionics subsystems, it is made responsible for monitoring the health of these subsystems. Any discrepancy between multiplied equipment and abnormal behaviour is logged into

non-volatile FCDM memory and also send to the on-board maintenance device. The logged errors can be downloaded from FCDM during on-ground maintenance. This log allows an early warning system to prevent possible future malfunctions leading to accidents.

5 Experience gained with safety critical software development methods

The definition of the FCDM 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. The Attitude and Orbit Control System (AOCS) software for the Italian- Dutch SAX (Astronomical X-ray Satellite) [ref 4] has been developed using the following software development method

- customer supplied specifications provided in plain English
- use of ESA PSS-05 life cycle model [ref 5]
- software analysis using Structured Analysis with Hatley and Pirbhai Real Time extensions (SA/RT) 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 entire 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 1,000 lines of code. This can be categorised as an extremely low value, refer also to [ref 6]. 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.

For FCDM the customer prescribed the use of the DOD-STD-2167A [ref 7] 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
- software analysis using Structured Analysis/Real Time 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

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

- automated test tool to aid the construction and cost effective repetition of module tests and as many of the integration tests as practical
- a mandatory 100% code coverage for level A software. 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 operator in an expression, its independent effect on the outcome of the expression is demonstrated
- execution of all module tests and the integration tests on the target system with a hardware emulator with instrumented code. Subsequently repetition of all these tests with non-instrumented code to check whether the same results are obtained as with the instrumented code. An emulator considerably expedites the analysis of unexpected results.

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.

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 FCDM 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 FCDM 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 ensured that the relevant procedures are strictly enforced.

From a functional point of view this concentrated development effort resulting in 4 pre-releases of the software, has been very successful. Up to date the software contains nearly all functions while at the same time around 150 changes to the system requirements have been accommodated. Valuable feed back from the user (pilot) has been obtained. Also the development of the displays and especially its integration with FCDM and the aircraft have been expedited considerably.

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, as 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 the invalidating some already completed flight trials of the aircraft.

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. The Teamwork tool is too labour intensive to keep the analysis and design up to date.

Independent personnel is required for the verification of the coding and integration processes. This requirement combined with the outdated status of the analysis and design, means that 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 delay of the certification. At least co-development saves re-certification effort as well as the generation of much documentation describing pre-releases.

7 Comparison of standards for safety critical systems

This section will highlight some differences between several standards for safety critical systems, based on the avionics application experience.

DO-178B has been specifically constructed for airborne systems and pre-dates the other standards. Currently other standards for safety critical systems are available, ISO/DIS 15026 [ref 8] and IEC 1508 [ref 9]. Like DO-178B also ISO/DIS 15026 recognises an "integrity assurance authority" besides the customer and supplier. This authority issues a certificate of compliance. The number of software integrity levels and their criteria are to be negotiated with the integrity assurance authority. As an example the IEC 300-3-9 [ref 10] is included in the standard. JAR/FAR-25 requires a catastrophic failure to occur less then once during 10-9 flight hours. IEC 300-3-9 classifies a system exhibiting a failure with catastrophic consequence and an incredible frequency (defined at < 10-6 per year, which at a commercial utilisation rate of 1000 flight hours per year for the aircraft involved, equals 10-9 per flight hour) only as intermediate. Like DO-178B ISO/DIS 15026 does not recommend a life cycle. An example life cycle with methods to achieve confidence are included. This example life cycle defines 8 phases based on the waterfall model and consequently does not accommodate the commercial necessities of co-development and short time to market.

IEC 1508 classifies the failure frequency in 4 levels, the most severe allowing between 10-6 and 10-4 dangerous failures per year. This is more frequent then DO-178B combined with JAR/FAR-25. IEC 1508 declares itself unapplicable for lower failure rates. IEC 1508 defines a safety life cycle with 16 phases which include modifications after delivery, retrofit and decommissioning. The inputs and outputs per phase are specified. Highly recommended techniques are prescribed. Not using these requires a mandatory justification. As a minimum ISO 9000 quality assurance is required. Although the need for iteration is mentioned, the software life cycles and the recommended techniques do not seem to accommodate this commercial necessity. DO-178B is based on independent execution of the verification by the supplier combined with the organisationally independent certification authority. IEC 1508 prescribes the independence of the assessors per safety level for a number of activities. The options are independent person, independent department and independent organisation. For systems with the highest integrity level IEC 1508 states a structured method supported by a tool as "highly recommended". C is classified as "positively not recommended" and IEC 1508 is impartial for C with subset and coding standard.

The emergence of these standards means market opportunities for tools which support the development of safety critical systems. This may benefit the development of safety safety critical systems in various ways:

- tools may be qualified once by the vendor, reducing considerable project qualification effort,
- tool checks may become more comprehensive reducing (re-)verification effort,
- tool operations may improve allowing to support the commercially necessary co-development,
- commercial tools should significantly lower the effort needed to produce safety critical systems and hence their price, making those systems affordable for many more applications.

8 Conclusion

For software development in air transport the safety requirements are stated in DO-178B. This document is sufficiently clear to allow a first-time developer to define, without external support, a compliant development process. This document is used by both software producers and by the independent certification authority which ensures compliance. Developing software according to the traditional waterfall model allows compliance to be achieved.

Producing aircraft is a commercial venture which means that the various aircraft subsystems need to be co-developed in order to achieve the commercially determined time to market. The spiral model is more appropriate then the waterfall model. An integrated tool set is needed which supports the co-development i.e. which allows when a change occurs to concurrently update analysis, design, code, integration and verification (including traceability information). Currently 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. Emerging general purpose safety standards suggest that safety critical systems will also have to be build for other application areas. This could make the required tools commercially attractive for tool vendors, which in turn, could reduce the costs associated with safety critical system development.

The 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. By performing these iterations before certification at least the re-certification costs of the intermediate versions have been avoided.

References

[ref 1] DO-178B, Software Considerations in Airborne Systems and Equipment Certification (December 1992)

[ref 2] Federal Aviation Requirements/Joint Aviation Requirements FAR/JAR-25

[ref 3] Safer C, L. Hatton (1995). Mc Graw-Hill

[ref 4] Development procedures of the on-board attitude control software for the SAX satellite, G.J. Dekker, NLR Technical Publication TP 96573 L (1996)

[ref 5] ESA Software Engineering Standards, ESA-PSS-05 (1991)

[ref 6] Software faults: the avoidable and the unavoidable: Lessons from real systems, L. Hatton, Proceedings of the ESA 1996 product assurance symposium and software product assurance workshop (ESA SP-377), Noordwijk, March 1996, page 271-275

[ref 7] DOD-STD-2176A Department of Defense (DoD) Defense System Software Development (February 1988)

[ref 8] ISO/DIS 15026 Information technology - System and software integrity levels (1996)

[ref 9] IEC 1508 Functional safety:safety related systems, 7 parts, (June 1995)

[ref 10] IEC 300-3-9 Dependability management - Part 3: Application guide - Section 9: Risk analysis of technological systems (1995-12)