Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TP-98282

Avionics application development, coalesce certifiability with business opportunity

E. Kesseler and E. van de Sluis

Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TP-98282

Avionics application development, coalesce certifiability with business opportunity

E. Kesseler and E. van de Sluis

This report is based on a presentation held on the DASIA '98,
Data System In Aerospace Conference, Athens, Greece, 25-28 May 1998.

Division:	Informatics
Issued:	June 1998
Classification of title:	Unclassified

Abbreviations

AOCS	Attitude and Orbit Control System
FAR	Federal Aviation Requirements
I/O	Input/Output
IMC	Instrument Meteorological Conditions
JAR	Joint Aviation Requirments
MC/DC	Modified Condition/Decision Coverage
SAX	Astronomical X-ray Satellite
SA/RT	Structured Analysis/Real Time
SD	Structured Design
VMC	Visual Meteorological Conditions
VOR	VHF Omnibearing Range

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 discussed 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. The safety of the aircraft depends on these functions, rendering them safety critical. To protect the interests of the general public an independent national governmental authority certifies the avionics application as fit-for-use i.e. airworthy. In this paper the experiences with the software development methods to meet these requirements are presented.

The development of aircraft is a commercial venture. In order to meet the business opportunity a short time-to-market is essential. To comply, the various parts of an avionics suite need to be developed concurrently by, in our case, several companies, while satisfying the safety and certifiability requirements. Based on metrics for several key process areas of Capability Maturity Model, the influence of concurrent engineering on the software development is discussed.

2 Air transport safety requirements

For safety critical software in airborne equipment the DO-178B 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 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 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, catastrophic failure$< 1x10^{-9}$

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 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 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 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. 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, whose formal audits add a stage in the approval process. The organisational independence improves the position of the assessors. Our customer had ample experience with DO-178B certification. The customer requested only minor modifications to the process documents. The certification audit resulted in only 2 very minor comments. We have experience with safety critical software development. This first use of DO-178B implies that DO-178B can be adhered to without prior knowledge of the specific certification process.

The project team is set up consisting of 2 separate groups, a development group and a verification group. The verification group is headed by a team member with sufficient authority to report, at his own discretion, to the company management outside the project hierarchy.

The customer prescribes the use of the C programming language. This is considered a potential risk for the successful development of the safety critical application. The C language contains numerous constructs that are unspecified, undefined or left to be defined by the compiler supplier (Hatton 1995). This risk is 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 is 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 Flight Display Subsystem is designed to operate in both Visual Meteorological Conditions (VMC) and Instrument Meteorological Conditions (IMC). Under the latter conditions the displays are needed by the pilot to fly, consequently the correct functioning of the displays is safety critical. A number of equipment items needs to be duplicated to achieve the required failure probability.

In case of an equipment item failure or a discrepancy between two sensors, a re-configuration control unit permits the pilot 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. Consequently the software becomes safety critical.

During normal operation the avionics application processes about 100 different flight parameters, originating from 10 different sensors. Two processors are used. The delay times within the entire embedded application should be guaranteed to be less then 30 msec with a cycle time of 20 msec for the main processor. Due to the many changes expected during the operational life of the embedded software 50% spare processor time shall be allowed for. The I/O processor has a cycle time of 360 micro seconds. Each parameter is classified as

- critical: loss or undetected error could lead to a catastrophic failure. Examples of critical parameters are the attitude parameters pitch, roll, and heading,
- essential: loss or undetected error could lead to a hazardous failure. An example of an essential parameter is VOR (VHF Omnibearing Range) for aircraft position determination,
- 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,
- no effect: loss or undetected error does not lead to a failure condition affecting the aircraft or the crew workload. An example is ground maintenance.

Depending on the criticality of the flight parameter, the software validates it in up to four complementary 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 the 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 embedded 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 mission critical 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) life cycle model,
- software analysis using Structured Analysis with Real Time 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 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 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) 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's proprietary C-coding standard, with project specific enhancements and enforced by a static analysis tool,
- use of proven of-the-shelf tools to support the software development process, wherever available.

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 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 of this approach is that the source code can be tested without availability of the target computer.

6 Business opportunity versus certifiability

The business opportunity of the avionics product is characterised by

- short time-to-market,
- concurrent engineering by several companies, distributed over Europe,
- continuously changing market demands leading to requirement volatility.

6.1 Concurrent engineering necessity

The commercially defined short time-to-market, required a concurrent definition of the system requirements with the software analysis process. Based on the success of this approach, two certifiable releases were defined, the first for the launching customer and the second satisfying all requirements for every customer configuration. For each customer the deliverable functions can be tailored to the airframe using configuration files. This obviates the need to certify each tailored delivery.

The market opportunities resulted in concurrent updates of the customer's system requirements during the entire design, coding and integration processes. To aid the integration of the avionics application in the customer developed displays and subsequently in the existing aircraft, a first pre-release of the software with very limited functionality was defined. The development of this pre-release complied with the documented software development processes. The only deviations were the informal nature of the customer reviews and the delay of the certification audits. (Note that this pre-release was not meant to fly. Even for pre-releases used in test aircraft during flight tests, formal reviews and a significant number of the certification audits have to be passed successfully.) The first pre-release served its purpose well. A lot of feed-back was obtained, resulting in many changes to and clarifications of the customer's system requirements.

Due to the success in eliminating system level problems by the concurrent engineering of the first software pre-release, the customer requested to continue the concurrent engineering. The following sections will analyze various aspects of the software development process using metrics.

6.2 Requirement volatility

Figure 1 depicts the number of implemented requirements for each pre-release. Figure 1 is cumulative, i.e. the number of partially implemented requirements are added to the number of fully implemented requirements. Superimposed are the number of requirement changes.

The steady rise in the number of (partially) implemented requirements (fig 1) shows that from a functional point of view this concurrent engineering has been very successful. At least 1 additional

pre-release is expected before the first certifiable release. Up to date the software contains almost all functions for the first certifiable release.

Currently on average there is 1½ change for every requirement (fig 1). A minority, but still significant, number of these changes relate to valuable feed-back from the user (pilot). Most remaining changes are caused by

- ambiguities in the plain English of the specifications,
- adding product features,
- the concurrent engineering of the displays (by the customer) and the avionics application,
- the integration of the displays with the avionics application and the aircraft. This integration has been expedited considerably by the pre-releases.

Based on this experience the exclusive use of plain English is insufficient to specify the system requirements and the interfaces between system components. However a more formal specification should be intuitively clear for both the application experts and the realisation team. Even the Teamwork analysis models caused communication problems with the application experts.

The official change procedure links every change with its commercial impact i.e. costs and schedule. The commercial dispositionning time turned out incompatible with the time-to-market. A separate process is used to swiftly document and decide on every technical change, irrespective of its commercial impact. Subsequently the commercial effort can be assessed for groups of changes. A short dispositionning time is a prerequisite for informal co-development with its inherent large number of changes.

The reduction of the number of implemented requirements between the first and second pre-release and the 5th and 6th pre-release is caused by new issues of the requirements document containing major new requirements.

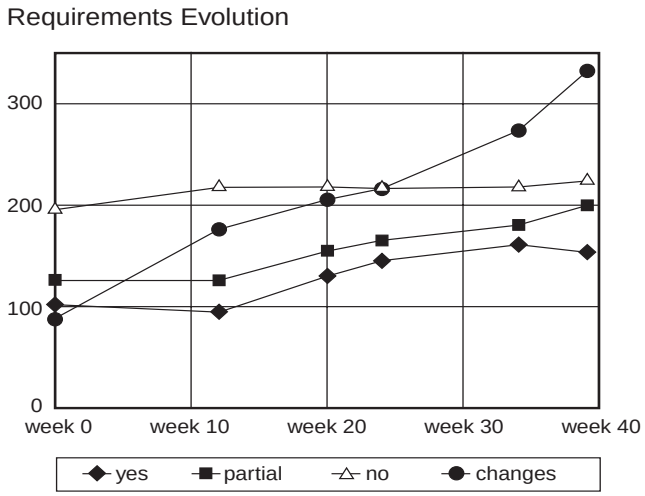

Fig. 1 Requirement implementation status (cumulative) and number of changed requirements

6.3 Pre-release evolution

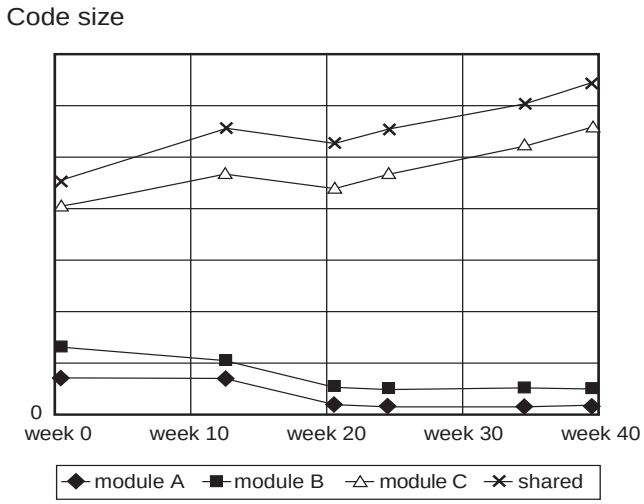

Fig. 2 Configuration item code size per pre-release, cumulative

The embedded application consists of 4 configuration items, 3 programme modules and 1 shared library module. By far the most requirements are implemented in Module C. The evolution of the configuration items (fig 2) shows the non-comment lines of target code. All auxiliary code (e.g. for testing, customised checkers etc) is excluded.

Between the first and the second delivery many functions from module B were moved to the shared library (fig 2) because of modified requirements combined with a consolidation of the design of both modules. Between the 2nd and 3rd delivery the same occurred for modules A and B/C. Also between the 2nd and 3rd delivery, the execution time was considerably reduced by optimising the software architecture. From the 3rd delivery onwards mainly additional requirements are implemented in module C with some minor module consolidation.

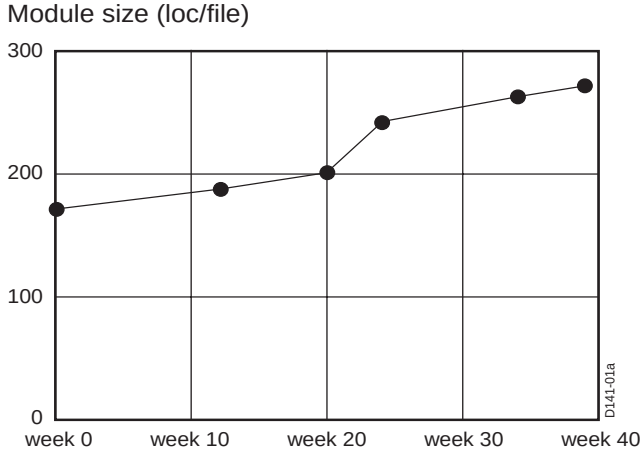

Fig. 3 Module size per pre-release

A module is defined by the architecture and usually contains several C-functions. The evolution of the module size (fig 3) shows that the first pre-release already contained nearly all modules, however with limited functionality. During the subsequent releases more requirements are implemented, resulting in larger modules. This illustrates that the 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 the proposed changes. Unfortunately the updating of the Teamwork models is too labour intensive to keep up with the implementation of the requirement changes. Due to the commercial pressure to implement requirements for the next pre-release (the co-development) both Teamwork models became rapidly outdated. The size of the software made a substantial increase of the team size infeasible. The result is that during certain periods the documentation was lagging behind the implementation.

6.4 Concurrent engineering experience

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 Conclusions

For air transport the safety requirements stated in DO-178B software are sufficiently clear to allow a first-time developer to define, without external support, a compliant software development process. Compliance can be achieved with the traditional waterfall software development model.

Producing aircraft is a commercial venture i.e. concurrent engineering of the various aircraft subsystems is needed in order to achieve the commercially 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 concurrent engineering i.e. which allows when a change occurs to concurrently update analysis, design, code, integration and verification (including traceability information). Currently commercially available tools do not provide this capability.

For simple verification tasks customised tools can be developed cost-effectively.

The exclusive use of plain English to specify system requirements and interfaces leads to ambiguities and hence requirements changes. A more formal method should be intuitively clear for both the application experts and the realisation team.

A swift dispositionning mechanism is needed for the many technical changes inevitable during the co-development. The commercial dispositionning was incompatible with this co-development pre-requisite.

Co-development accommodates the customer requests, improves avionics/air frame integration and provides market opportunities.

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-2167A. 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 Graw-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 SP-377) Noordwijk