**Nationaal Lucht- en Ruimtevaartlaboratorium**

National Aerospace Laboratory NLR


NLR-TP-99241

# Development and verification of the FCDM safety critical avionics module

R. Udo and E. van de Sluis

**Nationaal Lucht- en Ruimtevaartlaboratorium**
National Aerospace Laboratory NLR


NLR-TP-99241

# Development and verification of the FCDM safety critical avionics module

R. Udo and E. van de Sluis

This report is based on a presentation held on the DASIA 1999, Lisbon on 17 - 21 May 1999.



Division: Information and Communication Technology
Issued: December 1999
Classification of title:

**Summary**

In the late 1990s the NLR and Fokker ELMO were involved in the development of a safety critical embedded system, the Flight Control Display Module (FCDM). The FCDM is part of the Flight Control and Display System (FCDS). The FCDS is developed by Sextant Avionique as part of their Avionique Nouvelle concept. The FCDS finds its first application in the EuroCopter EC-135 helicopter. Within FCDS, FCDM acts as a data concentrator and transmits flight control and navigation data to display units.

This paper describes the development and verification of the FCDM. The inputs for the development process are described: system requirements and a pre-defined hardware architecture. The development process resulted in a software architecture that is also briefly described. Partitioning has been applied to segregate less critical components, which resulted into three different levels for the software: A, B and E.
A brief description of all DO-178B software life cycle processes is given.

## Contents

(18 pages in total)

# 1 Introduction

The last thirty years show an exponential growth in the use of software within airborne systems [Storey]. These systems are typical examples of embedded systems. They are used to monitor the state of or even to control an aircraft and to display flight control and navigation data to the pilots. The nature of these systems is safety critical if the impact of a failure leads to conditions which prevents an aircraft from a continued safe flight and/or landing.

In the late 1990s the NLR and Fokker ELMO were involved in the development of a safety critical embedded system, the Flight Control Display Module (FCDM). The FCDM is part of the Flight Control and Display System (FCDS). The FCDS is developed by Sextant Avionique as part of their Avionique Nouvelle concept. The FCDS finds its first application in the EuroCopter EC-135 helicopter. Within FCDS, FCDM acts as a data concentrator and transmits flight control and navigation data to display units.

The guidelines for the production of software for civil airborne systems and equipment are documented in [DO-178B]. An independent third party (the certification authority) checks on compliance with these guidelines by performing a number of audits. According to DO-178B the system safety assessment process results into a classification of all software components into five possible software levels. Since the FCDM handles flight critical control and navigation data the FCDM software is classified as level A (the most severe level).

This paper describes the development and verification of the FCDM. The inputs for the development process are described: system requirements and a pre-defined hardware architecture. The development process resulted in a software architecture that is also briefly described. Partitioning has been applied to segregate less critical components, which resulted into three different levels for the software: A, B and E.
A brief description of all DO-178B software life cycle processes is given.

# 2 FCDM system requirements

Apart from FCDM, FCDS is made up of display units, Instrument Control Panels (ICP) and a Reconfiguration Control Unit (RCU).
Flight control data is displayed on the Primary Flight Display unit (PFD) and navigation data is displayed on the Navigation Display unit (ND). The ICP enables the pilot(s) to change flight settings and to select flight equipment and display modes.

Two operation modes are defined for the FCDM:

- Single or Double Pilot Visual Flight Rules (SPVFR, DPVFR) with one FCDM.
- Single or Double Pilot Instrument Flight Rules (SPIFR, DPIFR) with two FCDMs. In this case a cross-link for data validation between the two FCDMs is used to achieve the IFR safety requirements. The FCDM module itself, however, has no redundancy.

In IFR safety critical flight equipment is duplicated and manual reconfigurable by the pilots by using the Reconfiguration Control Unit (RCU). This reconfiguration is typically performed in case of discrepancy of pre-defined parameters between the equipment.

The following figure shows the FCDMs in a DPIFR configuration with two sets of display units (PFD/ND), two ICPs and the RCU. As an example of connected input equipment the Attitude Heading Reference System (AHRS) is shown.

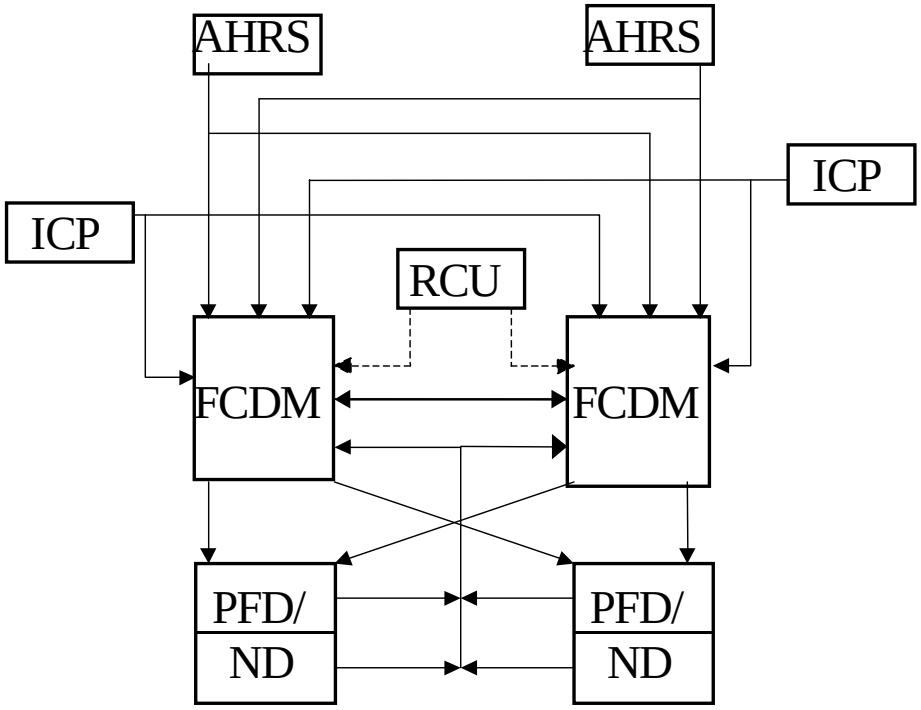


*Fig. 1 FCDM in Dual Pilot Instrument Flight Rules (DPIFR) configuration*

All indicated busses are ARINC 429 busses with the exception of the discrete busses from RCU to the FCDMs. ARINC 429 is a standard communication bus in civil avionics; discrete busses transmit only logical 1/0 states.

The FCDM implements the following functionality:

- Aircraft configuration management by means of a downloadable FCDS configuration file.
- Parameter selection received from the helicopter equipment (ARINC 429 sensors).

- Safety validation by checking refresh rates of all equipment and calculating discrepancies for duplicated equipment.
- Transmission of processed data to PFD and ND and to other helicopter equipment.
- Management of two ICPs.
- Management of manual equipment reconfigurations.
- Management of cyclic in-flight checks.
- Management of function tests and storage of test results.
- Failure management.

## 3 Hardware architecture

The FCDM module consists of two Central Processor Units (CPUs). One CPU is dedicated for all ARINC 429 input and output handling (I/O processor), the other CPU is dedicated to discrete I/O handling, data processing, data validation, flight control, etc. (Main processor), see figure 2.
The IO hardware components are:

- Three ARINC 429 controller chips. Each controller contains 8 input channels with control logic to check the consistency (parity and length) of received ARINC words and 3 output channels.
- Intel i386EX CPU @ 66 MHz. with an embedded timer, used as Real Time Clock (RTC).
- Memory: 256 Kbytes static RAM (for run-time data, e.g. set-up data, stack, etc.) and 256 Kbytes PROM (for IO firmware).

The Main hardware components are:

- Intel i386EX CPU @ 66 MHz. with embedded:
  - Watch Dog Timer (WDT).
  - Timer used as Cycle Timer.
- Memory: 256 Kbytes static RAM (for run-time data, e.g. set-up data, stack, etc.), 512 Kbytes PROM (for the boot firmware and the application software) and 16 Kbytes Non Volatile RAM (NVRAM, for non-volatile data, e.g. error log data, ICP data, FCDS configuration file, etc.).
- One serial I/O controller with an EIA 485 serial port for maintenance purposes and an EIA 232 serial port for a debug and status monitor.
- Discrete I/O interfaces with 48 inputs and 32 outputs.

The I/O and Main CPUs are linked through shared memory (16 Kbytes dual ported RAM) to exchange set-up information, I/O control and ARINC 429 flight data.

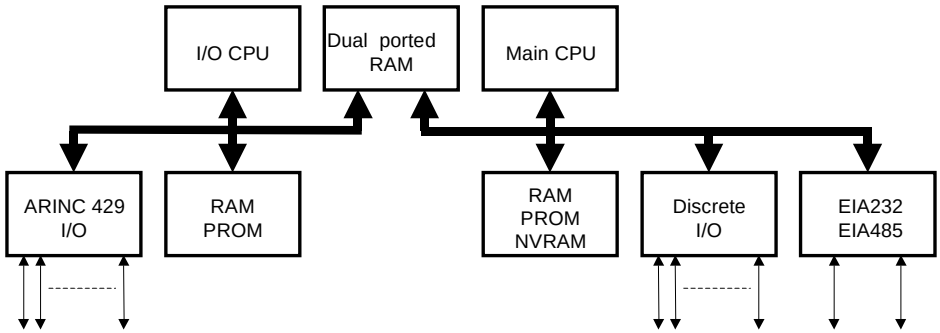


*Fig. 2- FCDM dual CPU hardware architecture*

## 4 Software architecture

Analysis of the system requirements and the already available hardware resulted in the decision to develop three software components:

- A boot strap loader.
- Firmware for the I/O processor.
- Software for the Main processor.

Figure 3 depicts this software architecture. It shows the downloadable FCDM/Main software (by the boot strap loader), the downloadable FCDS configuration file and the ARINC 429 and discrete inputs/outputs.

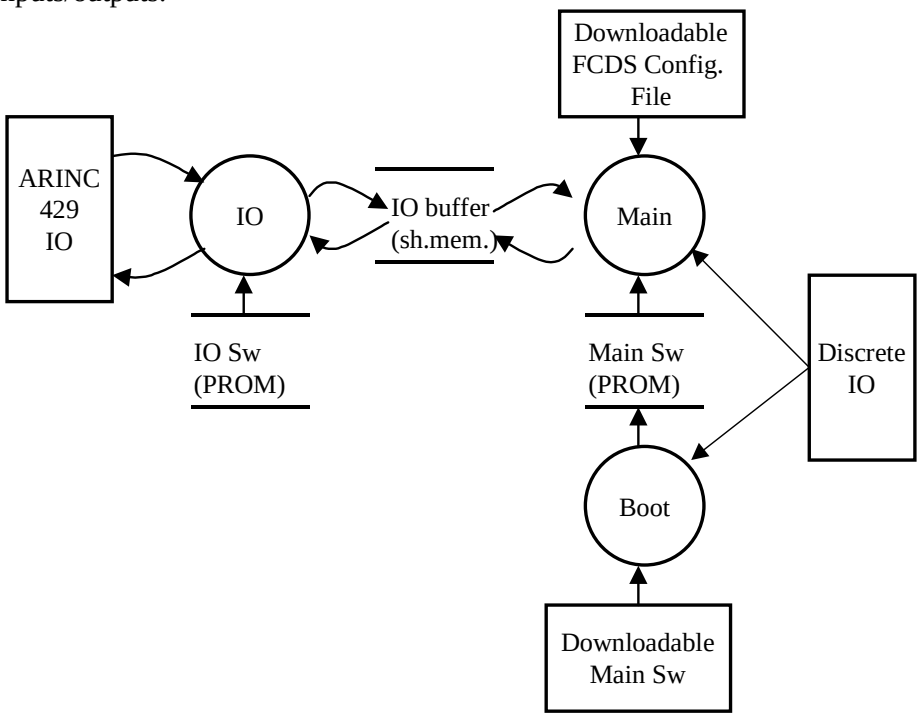


*Fig. 3 FCDM software architecture*

### 4.1 Boot strap loader

The boot strap loader is used to download data and/or application software (i.e. a hardware production test/acceptance procedure or the Main application software) and to start application software.
The hardware production test/acceptance procedure is used to perform a comprehensive functional test of the FCDM hardware. It is loaded into the FCDM on ground on a dedicated workbench. This item is outside the scope of this paper.

#### 4.1.1 Safety consideration

The decision whether to start the load function or to start the application program may have a catastrophic impact on the helicopter safety and is therefore classified to be DO-178B level A. The down load function itself, however, is level E.

The boot strap loader is defined to be firmware since its functionality will not change in time; it is located in the FCDM/Main PROM.

### 4.2 FCDM/IO firmware

The FCDM/IO firmware receives, selects and validates all ARINC 429 data and transmits ARINC 429 data to a/o the display units through the normal and safety busses.
In the FCDM/IO set-up mode set-up data is exchanged from FCDM/Main to FCDM/IO through the shared memory. This data is used to initialise the ARINC 429 controllers and the I/O processing. The load of set-up data is completed with a verification of the checksum of all set-up data.

In the operational mode FCDM/IO receives, selects and validates (in terms of data coherency: number of bits and parity) ARINC 429 data. The ARINC 429 controllers perform this task. All data is time-stamped and stored on its specific location in the shared memory, either in a FIFO (for data of which not a single sample may be missed) or in a *most recent sample buffer* (for data of which always the most recent sample has to be available).
In operational mode FCDM/IO transmits ARINC 429 data as well. ARINC 429 output data is stored in FIFOs in shared memory. There is one FIFO for every ARINC 429 output channel.

The set-up mechanism allows a flexible configuration of all ARINC 429 inputs and outputs. Since the basic IO functionality will not change in time, the FCDM/IO is defined to be firmware.

### 4.2.1 Safety consideration

Since the FCDM handles safety critical control and navigation data all the FCDM/IO firmware is considered to be level A.

To guarantee the integrity of the FCDM/IO hardware and firmware FCDM/IO performs an exhaustive Power-on Built-in Test (a/o checksum verification on the firmware and memory tests). This test is performed only in case of a *cold* start.
The implementation of a Cyclic Built In Test (CBIT, a/o memory tests) increases the integrity of FCDM/IO even further.

## 4.3 FCDM/Main software

The FCDM/Main software implements the major part of the system requirements.

The FCDM/Main software uses tables for all input/output processing, reconfigurations, calculations and monitoring activities. At start-up these tables are optimised (adding redundant data from other tables) and actualised with data from the FCDS configuration file to speedup the real-time processing as much as possible.
A new FCDS configuration file can be downloaded in maintenance mode through the EIA 485 serial link.

In operational mode FCDM/Main processes all ARINC 429 data. All ARINC 429 data is validated in terms of refresh rate. To meet the system integrity safety critical parameters are checked for value discrepancies. The integrity of the FCDMs is increased by reading safety critical data from the cross-link and to check them for value discrepancies.

The ICPs only send incremental data to the FCDMs. FCDM /Main maintains for both ICPs all modes and states of the flight settings. These settings are stored in the NVRAM, hence after an in-flight re-start FCDM continues with the most recent values for the settings.

Reconfiguration is determined by reading the RCU input discretes and the RCU information from the opposite FCDM through the cross-link. Comparing these sets of information makes the final decision. In case of reconfiguration of equipment, the equipment is not taken into account for any processing but the remaining equipment takes over.

### 4.3.1 Safety considerations

To guarantee the integrity of the FCDM/Main hardware, software and FCDS system consistency FCDM/Main performs a Power-On Self-Test (POST). This POST is executed at start-up in case of a *cold* start or fatal error.

The implementation of a Cyclic Built In Test (CBIT, a/o checksum verification of all set-up tables, memory tests) increases the integrity of FCDM/Main further.

All operational functions processing flight critical data are considered to be level A. CBIT and POST functions are level B, while maintenance functions are considered to be level E.

### 4.3.2 Partitioning

To ensure that more critical software components are not affected by less critical components the components have to be segregated from each other. One architectural concept, as proposed by DO-178B to comply with this, is software partitioning.
The FCDM/Main software uses the partitioning to:

- Provide isolation between functionally independent software components.
- Isolate faults.

Isolation between software components is achieved by using:

- Spatial partitioning.
- Temporal partitioning.

FCDM/Main implements the spatial partitioning by using the i386EX paging mechanism. Temporal partitioning is used to execute the CBIT as a separate task in background.

# 5 Software development process

## 5.1 Requirements process

For the requirements analysis Structured Analysis with Real Time extensions (SA/RT) [Hatley & Phirbhai] supported by the Cadre TeamWork CASE tool was used. The SA/RT process and control specifications contain the high level requirements in plain English.
All system requirements are identified with a unique number. This number is reflected within the requirements analysis (and the SA/RT model) to accommodate requirements traceability from system requirement to high level requirement.

## 5.2 Design process

Software design is the process through which high level requirements are translated into low-level requirements and the software architecture. The software architecture basically consists of a structure of interrelated modules. The modules contain coherent sets of functions and definitions of data structures.

For the design the Structured Design method (SD) [Yourdon] was planned to be used. Since the system requirements evolved during the implementation of the project the SD model became unmanageable. It was decided then to stay with the preliminary design and to insert structured pseudo code into the source files. The pseudo code was extracted from the source files at the end of the project and forms the main constituent of the design document.
A simple call tree presents the software architecture in the design document.

High level requirements are allocated to specific functions within modules. Function names are used as identifier for the low-level requirement identification. This accommodates the requirement traceability from high level requirements to low-level requirements.

### 5.3 Coding process

The objective of the coding process is to produce source code based upon the software architecture and the low-level requirements.
For this process the customer required the use of the C programming language. Due to the criticality of the application an ANSI C compiler is used and the NLR proprietary C coding standard was enhanced according to the guidelines of [Hatton]. Using a Commercial Of The Shelf (COTS) static analysis tool enforced the (restricted) use of the C programming language features.

Since the source code was enhanced with detailed design directives (file headers, function headers and pseudo code) traceability between low-level requirements and source code was easily satisfied.

### 5.4 Integration process

The objective of this process is to produce the application executables by using the target computer, the source code from the coding process and the software architecture from the design process.

In a first level of the implementation the core of the FCDM software (the table driven input/output processing, reconfigurations and calculation activities) was developed and tested in the UNIX host environment. After the target system became available the FCDM software was adapted and tuned for the target environment.

To test and debug the application in its target environment a COTS in-circuit emulator in conjunction with a cross-debugger and a COTS ARINC-429 simulation tool was used.

The ARINC simulation tool simulates the helicopter environment by producing a number of relevant ARINC 429 data busses (i.e. simulated helicopter equipment) and displays in real-time a number of selected ARINC 429 busses from the FCDM.
The emulator enables the developer with step by step operations, memory and register checking, breakpoint setting, timing analysis, etc.
The target computer used for the integration process was a so-called development board. To be able to clip-on the emulator this board was not provided with the protective coating, as the production boards are. Moreover, the watch dog capability was disabled to allow debugging with the emulator. Apart from that the development board and the production boards are functionally identical.

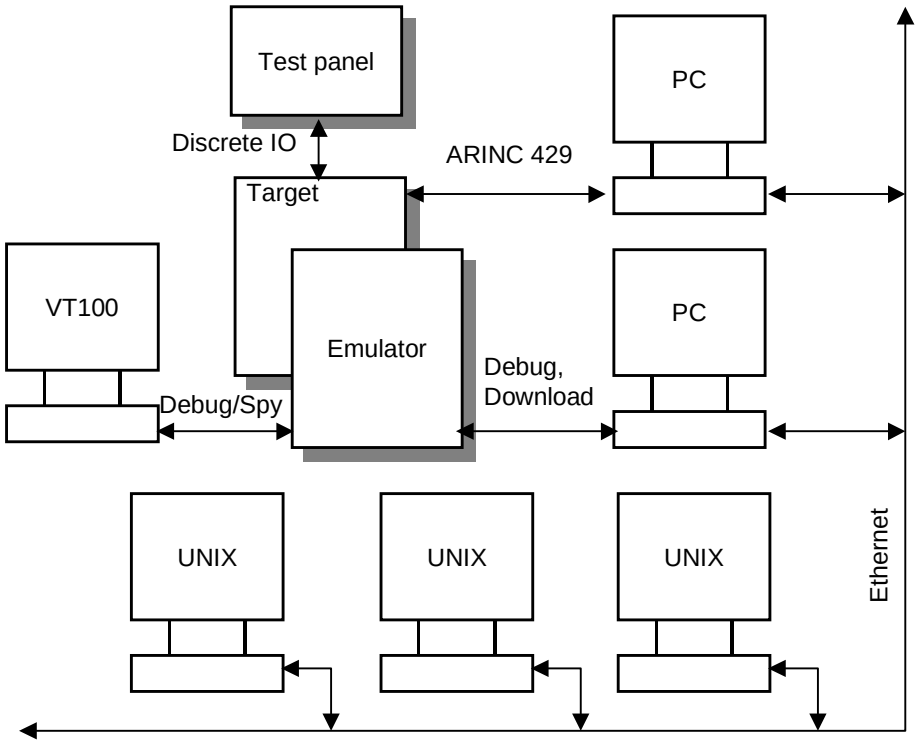


*Fig. 4 FCDM development/integration environment*

Figure 4 shows the development/integration environment for the FCDM.

The test panel is used to simulate all discrete inputs/outputs. The UNIX systems are used for the development and configuration management. The PCs are used for testing and debugging the application with the in-circuit emulator in conjunction with the cross-debugger and the ARINC simulation tool. The VT100 terminal is used for in-line debug and spy purposes.

# 6 Software verification process

## 6.1 Verification

Verification is defined in [Ref. DO-178B] as 'the evaluation of the results of a process to ensure correctness and consistency with respect to the inputs and standards to that process'. So, verification is not simply testing, but typically a combination of reviews, analyses and test. The general objectives of the software verification process are to verify that:

- The system requirements have been developed into software high-level requirements that satisfy those system requirements (verification of requirement process).
- The high-level requirements have been developed into software architecture and low-level requirements that satisfy the high-level requirements (verification of design process).
- The software architecture and low-level requirements have been developed into source code that satisfies the low-level requirements and software architecture (verification of coding and integration process).
- The executable object code satisfies the low-level software requirements (testing).
- The means used to satisfy the verification process objectives are technically correct and complete for the software level (verification of the verification process itself).

For level A software, [DO-178B] requires that the above objectives are satisfied with independence, which in practice means that other persons than the developers perform the verification activities.
For lower software levels less emphasis is on requirements coverage and structural code coverage and their verification.

With testing as the only exception, all the above verification objectives have been achieved by analysing and reviewing the outputs of the process. Note that the verification process itself is subject to verification.

## 6.2 Testing

Testing is defined in [Ref. DO-178B] as `the process of exercising a system or system component to verify that it satisfies specified requirements and to detect errors'. Three types (or levels) of testing can be distinguished:

- Validation or hardware/software integration testing.
- Integration testing.
- Unit or low-level testing.

### 6.2.1 Validation or hardware/software integration testing

Validation or hardware/software integration testing is used to verify that the object code is compliant with and robust to the high-level requirements. Validation testing is performed by using the ARINC-429 simulation tool. With this kind of test environment test automation is not possible, which is a major concern when regression testing has to take place. All validation tests are performed on an FCDM production board.

### 6.2.2 Integration testing

Integration testing is used to verify the interrelationships between software requirements and components and to verify the implementation of the software requirements and software components within the software architecture. Integration testing is performed by making emulator scripts and executing them on an FCDM (development) board. The output of these scripts is then compared with expected output results. This approach makes it possible to automatically re-run the tests whenever necessary.

### 6.2.3 Unit or low-level testing

Unit testing is used to verify that the software units are compliant with and robust to the low-level requirements. This testing is performed by using a COTS test tool that can also provide detailed structural code coverage information of the tests that have been performed. For this purpose the tool instruments the source code, this then has to be re-compiled and run again. For level A software this approach is only allowed when it is guaranteed that the compiler produces object code that is directly traceable to the source code. All unit tests have been performed on an FCDM (development) board using an emulator.

The successful implementation of the partitioning mechanism implied that no unit testing was applied to level E software units.

## 7 Other life cycle processes

Note: A detailed description of the other DO-178B life cycle processes is outside the scope of this article.

### 7.1 Software planning process

The DO-178B document is not a development standard. Basically it consists of lists of objectives to fulfil in each software life cycle process. The main objective of the planning process is to define how compliance to the objectives is achieved. For this the actual software development standard was defined and methods and tools were chosen.

For example consider the verification process: DO-178B requires the generation of verification results but does not prescribe where or how to document the results.
For FCDM the [DoD-2167A] standard was used as a guideline for the documentation.

### 7.2 Software quality assurance process

The software quality process assesses the software life cycle processes and their outputs to obtain assurance that the objectives are satisfied, that deficiencies are detected, evaluated, tracked and resolved, and that the software product and software life cycle data conform to certification requirements.
One of deficiencies found by the FCDM quality assurance manager was that tool qualification of the unit test tool was insufficient. An audit of the supplier was deemed necessary and subsequently performed.

### 7.3 Software configuration management process

The main activities of this process include configuration identification, change control, baseline establishment and archiving of the software product, including the software life cycle data.
To manage the produced source code the NLR proprietary software repository tool is used. During the coding process this tool enables the developers to develop several modules in parallel and to update their development version with the latest committed version.

### 7.4 Certification liaison process

The objective of the certification process is to establish communication and mutual understanding between the producer and the certification authority. For this it is important to communicate the Plan for Software Aspect of Certification (PSAC) with the certification authority as soon as possible. Changes required by the authority at a later stage (during development) may be very difficult and expensive to implement.

## 8 Concluding remarks

The development of the safety critical avionics FCDM module as described above led to a successful certification of the FCDS.

As usual there appeared to be a conflict between the proposed development life cycle and the time to market as well as evolving system requirements. The major impact within the development phase was that the Yourdon structured design method for the detailed design appeared to be too heavy to maintain; the Yourdon structured design mehtod was replaced by the far more pragmatic use of pseudo code and a simple call tree for each software component.

The major impact for the verification was that the three test processes (validation, integration and unit testing) had to be performed in parallel.

An early separation of software levels for the diverse software functions in the FCDM module is very advantageous. Not only to ensure that more critical components are not affected by less critical components, which is a safety requirement, but also in terms of reduction in the effort of the software verification process.

The use of a coding standard is obvious for a safety critical application. However, since C isn't Ada additional guidelines are necessary to make it restrictive and by that safe. A (COTS) tool to enforce the coding standard and the restrictive use of C relinquishes a lot of tedious code walkthroughs.

The use of tables to implement the input/output processing, reconfigurations, calculations and monitoring activities resulted in highly compact source code. This approach resulted in easily maintainable code and corresponding test cases and procedures.
The tables were tested both manually (by inspection) and at the integration/validation level.

DO-178B is known to be a very mature and adequate guideline. The practical usage of DO-178B in the FCDM project confirms this. It provides the flexibility required in commercial applications development, while not compromising the safety requirements.

## 9 Abbreviations

| | |
|---|---|
| AHRS | Attitude Heading Reference System |
| CBIT | Cyclic Built In Test |
| COTS | Commercial Of The Shelf |
| CPU | Central Processor Units |
| DPIFR | Double Pilot Instrument Flight Rules |
| DPVFR | Dual Visual Flight Rules |
| FCDM | Flight Control Display Module |
| FCDS | Flight Control and Display System |
| ICP | Instrument Control Panels |
| ND | Navigation Display unit |
| NVRAM | Non Volatile RAM |
| PFD | Primary Flight Display unit |
| POST | Power-On Self-Test |

| | |
|---|---|
| PROM | Programmable Random Access Memory |
| PSAC | Plan for Software Aspect of Certification |
| RAM | Random Access Memory |
| RCU | Reconfiguration Control Unit |
| RTC | Real Time Clock |
| SA/RT | Structured Analysis with Real Time extensions |
| SD | Structured Design method |
| SPIFR | Single Pilot Instrument Flight Rules |
| SPVFR | Single Pilot Visual Flight Rules |
| WDT | Watch Dog Timer |

## 10 References

[Storey] Safety-critical Computer Systems
N. Storey
Addison-Wesley, 1996

[DO-178B] Software considerations in airborne systems and equipment certification
RTCA Inc. 1992

[DoD-2167A] Department of Defence (DoD)
Defence System Software Development, 1988

[Hatley & Phirbhai]
Strategies for real-time system specification
Hatley, D.J., Phirbhai, A.
Dorset House Publishing, 1988

[Yourdon] Modern Structured Analysis
Yourdon, E
Prentice-Hall Inc., 1989

[Hatton] Safer C
Developing Software for High-integrity and Safety-critical Systems
Hatton, L.
Mc Graw-Hill, 1995