LCA AVIONICS AND WEAPON SYSTEM MISSION COMPUTER SOFTWARE DEVELOPMENT: A CASE STUDY

Srikant Visweswariah
Deputy Project Director (Avionics System)
ADA, Bangalore
(Currently Chief Technology Officer at Bitsoft Systems)
&
B C Banerji
Group Director (Mission Software)
ADA, Bangalore

Abstract: The development of software ver 3.0 for the Light Combat Aircraft (LCA) Mission Computer was a large and complex project in which Ada was used as both design and code language. The development process and the experience gained are covered in this paper.

1. INTRODUCTION

The Avionics and Weapon System (AWS) in any modern day fighter aircraft enables the pilot to perform various mission functions and thereby meet the stipulated operational role of the aircraft.

The AWS must meet the following functional requirements in order to complete a Mission:

a) Receive inputs from
- Sensors
    - Radar, INS, Airdata system, LDP, FLIR, Radio Altimeter etc
- Communication Systems
    - Data link, voice link
- Radio Navigation System
    - TACAN
- Identification System
    - IFF (Responder and Interrogator)
- Missiles
    - Identification signal, seeker head position, locked-on to target signal etc.
- Electronics Counter Measures System
    - Radar Warning receiver
    - Self Protection Jammer
    - Offensive Jammer
- Pilot Controls
    - Hands on stick and throttle (HOTAS) controls
    - Other controls

b) Compute required parameters for navigation and fire control
    - Navigation algorithms: Guidance to steer point etc
    - Fire Control algorithms: weapon aiming, missile launch etc.

c) Output computed results to
    - Displays:
        - Basic flight, steering and navigation parameters
        - Weapon aiming, missile launch symbologies
        - Sensor outputs
        - Failures and warnings
    - Audio System: AWS status and warnings
    - Weapons

d) Control weapon launch/firing
    - Weapon selection and preparation
    - Launch/fire sequencing
    - Jettison

e) Control/co-ordinate/manage sensors optimally
    - Mode commands
    - Slaving commands

2. LCA AVIONICS AND WEAPON SYSTEM FUNCTIONS

The major functions performed by the LCA Avionics and Weapon System are:

- Operational (or mission) functions including fire control functions
- Mission preparation and data retrieval functions
- System-crew dialogue functions
- Aircraft communication functions
- Integrated maintenance functions

The operational functions include Navigation Guidance functions as well as Air-to Air, Air-to-Ground and Air-to-Sea weapon functions. Mission preparation involves the preparation of a data cartridge on the ground and transferring this information onto the aircraft. Data retrieval consists of the avionics system recording or storing information on the data cartridge, which could then be read on the ground. System-crew dialogue functions include cockpit controls/inputs management, display synthesis and management, and warnings management. Aircraft communication functions include both voice and data communication. Integrated maintenance functions include both in-flight maintenance and on-ground supplementary maintenance.

In the LCA, a number of utility system management systems (USMS) are integrated with the AWS. These systems are for fuel/oxygen management, engine health monitoring, environmental control system management etc.

The above mentioned functions are achieved through integrated functioning of the Mission Computer, sensors, controls, displays and the weapon systems - all software driven and in real time mode.

3. AWS SOFTWARE DEVELOPMENT PROCESS

As indicated in fig.1, an analysis of the Air Staff Requirements (ASR), the LCA mission profiles and the system requirements in each profile led to the definition of AWS requirements. An analysis of these requirements then led to AWS design. The first step in the design was the generation of AWS Broad specifications as indicated in fig.1. The next step was the generation of the AWS functional architecture and the Global specification of each Operational or mission function. Each Global specification describes completely a specific operational function from the Pilot's point of view. It includes operation of Pilot's controls, symbologies, warnings etc.

4. AWS FUNCTIONAL ARCHITECTURE

The definition of the AWS functional architecture involved:

- Classification of the AWS functions into various categories or groups
- Definition of the hardware units and software functions for each functional element in each group

The LCA AWS functions have been broadly classified into

- Central processing functions performed by the Mission Computer
- Functions performed by various external equipments like sensors (Radar, LOP, FLIR, RWR, etc), Weapon release/stores management units, communication units, etc.
- Cockpit man-machine dialogue equipment functions performed by Display Processor and various control panels and Display Surfaces in the Cockpit.

The above categorisation is based on a distributed processing concept with sensor-oriented functions being performed by various sensor computers and mission oriented functions being performed by a Central Computer (Mission Computer). Cockpit Man-machine dialogue equipment functions form another very important category of LCA AWS functions. The LCA AWS consists of a large number of units, like the Mission Computer, sensors, cockpit dialogue units etc, connected together by MIL-STD-1553B serial multiplex data buses.

5. MISSION COMPUTER (MC) FUNCTIONS

The MC is the manager of the LCA AWS and performs the following functions:

- Operational/Mission functions like Navigation and Guidance management, Fire control computations and management, etc
- Cockpit man-machine dialogue management functions
- Functions which manage/link external equipments like sensors, armament stores, communication, and radio navigation equipment etc.
- AWS Initialisation functions
- Miscellaneous functions like Air data Computations etc.
- AWS Maintenance management functions
- 1553B data Bus Controller functions

The MC hardware consists of a dual 80386-based computer with dual port RAM for inter-processor communication.

6. <u>MC SOFTWARE DEVELOPMENT OVERVIEW</u>

6.1 <u>DEVELOPMENT RESPONSIBILITY</u>

The MC software has been jointly developed by ADA, HAL, ASIEO and two private sector companies, M/s Processware System Pvt Ltd., and M/s Accord Software and Systems with ADA taking total responsibility and leadership. An independent V&V team from ADA was also involved during every step of the development process.

6.2 <u>DEVELOPMENT STEPS</u>

The steps involved in the MC software development were the following:

- Definition of MC software functional architecture
- Definition of MC Software Requirement Specifications (SRS)
- Design, code and unit testing
- Integration testing

The functional architecture was defined using Structured Analysis and Design Technique (SADT). SRS was defined using Ward and Mellor DFD methodology. Ada language was used as both design and code language. The MC software development and documentation broadly followed the MIL-STD-2167A standard. The MC software has been categorised as a computer software configurable item (CSCI) that executes on the MC hardware configurable item (HWCI).

6.3 <u>DEVELOPMENT ENVIRONMENT</u>

- Hardware
    - VAX 3195, VAX 3196, VAX 3400, MICROVAX II with VMS
    - About 20/25 VAX terminals
    - MC targets (Qty 4)

- Software
  - VAX Ada Native compiler running on VAX/VMS
  - DDCI Ada Cross compiler running on VAX/VMS producing code for MC target
  - VAX debugger
  - DDCI Ada cross debugger running on VAX and Debug monitor running on MC target
  - Code Management System (CMS) for configuration management
  - C Cross compiler/debugger for bus controller development
- MC Static Test Rig
  - The test rig consists of a large number of Pcs (one PC per hardware unit in the AWS) connected to the MC through 1553B data buses. This rig simulates the Inputs/Outputs of all the units connected to the MC on 1553B data buses. Provision is there to display all the 1553B data bus traffic to/from each unit as well as modify the data bus inputs to MC.

7. MC SOFTWARE FUNCTIONAL ARCHITECTURE

The first step in the generation of MC software requirements specifications (SRS) was the definition of the MC software functional architecture using SADT methodology. This consisted of

- Decomposition of MC functions hierarchically into various functional modules
- Definition of tasks performed by each module
- Definition of information flow between these modules

7.1 FACTORS IN FUNCTIONAL ARCHITECTURE DEFINITION

The driving factors have been modularity, flexibility and growth potential. As the MC performs a large number of Operational/Mission functions, the following requirements were specified and taken into account while defining the functional architecture:

- There is one specific operational function for each specific weapon
- There are specific operational functions for Navigation functions
- Each operational function is independent of other operational functions. It may be specified, implemented and tested independently of other operational functions
- During the life of the AWS, new operational functions would get added as new weapons are added
- According to pre-defined selection rules, operational functions may be selected to run concurrently (superposed) with each other or exclusively.

- No operational function is tied to specific cockpit controls or external equipment/sensors though it receives input from them
- No operational function is tied to specific display surfaces or external equipment/sensors though it sends outputs to them
- Separate development teams would work on various operational functions.
- Each operational function performs its tasks in the following manner
    - It receives
        - Inputs from the Pilot: From Cockpit Controls
        - Inputs from external equipments: sensors etc.
    - Then mission computations are performed
    - After computation, it produces
        - Outputs to the Pilot: Flight/Steering information, Guidance/Release cues, warnings etc, for display
        - Outputs to external equipments: Stores/weapons, sensor control information etc.
    - If the operational function does not receive the required inputs in order to enable it to produce its outputs, it generates warnings.

7.2 <u>CATEGORIES OF MC FUNCTIONAL MODULES</u>

Based on the above driving factors, the following categories of MC functional modules were defined:

- **OPERATIONAL MACROFUNCTION MODULES**:
    - These modules perform core mission tasks. There is one such module for each weapon and the various macrofunction module are independent of each other.

- **ENVIRONMENT MODULES**
    - These modules provide the environment for the operational macrofunction modules to perform. These modules are further categorised as
        - **System Logic Modules: (Cockpit Input Synthesis and Output Synthesis Modules**
            - These modules manage the cockpit man-machine dialogue including warnings, etc and activate other modules. These modules exchange information with operational macro function modules
        - **Modules Managing/Linking external equipment**
            - These modules manage various sensors and link other external equipment to the MC. These modules exchange information with operational macrofunction modules. These modules are further categorised as
            - Sensors management modules
            - Armament/stores management modules
            - Communication linking modules
- **AWS INITIALISATION MODULE**

- **BUS EXCHANGE MANAGEMENT MODULE**
- **GROUND SUPPLEMENTARY MAINTENANCE MODULES**

Under each category of functional modules, specific modules were defined and these were termed as terminal functional modules. These terminal functional modules were not decomposed further in the functional architecture.

8. MC SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

The MC SRS represents the essential model of the MC software and was defined in terms of the SRS for each of the terminal functional modules defined in the MC functional architecture. The SRS was generated using Mentor Graphics CASE tool and Ward and Mellor Data Flow Diagram (DFD) with real time extensions methodology. Essentially the MC functional architecture in SADT form was first hierarchically mapped onto context diagram/levelled data flow diagrams etc up to the terminal functional module level. Then the SRS for each terminal functional module was generated by different teams using the Mentor Graphics CASE tool. Each SRS team was defined the context of their terminal functional module (called a process in Ward/Mellor methodology) in terms of higher-level data flow diagrams etc. After the SRS of each terminal functional module was completed these were then integrated/balanced with the higher level DFD's/context diagram on the CASE tool to generate

- MC context diagram
- Complete set of levelled DFDs
- Complete set of state-transition diagrams
- Complete MC Data Dictionary
- Complete set of process specifications including role of each process
- Functional descriptions/explanatory notes wherever required
- Design constraints (including real time constraints)
- Data exchanges between various functional modules

The above list constituted the MC SRS. In ver 3.0 of the MC SRS, 26 terminal functional modules were considered and there were approximately 150 DFD's and the data dictionary size was about 10,000 lines (includes Data/Control elements and Data/Control flows). The SRS team consisted of about 25-30 systems engineers.

9. MC SOFTWARE DESIGN, CODE AND TEST (DCT)

The MC software DCT was done by a team of about 35-40 engineers. There were separate teams for Design, Design V&V/testing, QA, configuration management, hardware related software etc.

9.1 DESIGN APPROACH

Ada was used as the design language, as well as the coding language except for Bus Exchange Management module, which was implemented in C. The whole design process was driven by a set of DESIGN DIRECTIVES in various aspects of the design. These directives were issued to the design team members during the course of development by the design team leader in consultation with the project team. These directives addressed issues common to all team members as well as specific to each member. These directives served to achieve uniformity and cohesiveness in design. Some of the issues covered by the directives are; guidelines for preliminary design from SRS, design consideration for handling overflows in BUS interface procedures, design considerations for bus outputs etc. To date 69 directives have been issued.

Following MIL-STD-2167A standard, the MC CSCI was decomposed into a number of first level computer software components (CSC's). Each CSC corresponded to a specific functional module defined in the functional architecture. Additional CSC's were defined to cover implementation requirements.

9.1.1 PRELIMINARY DESIGN

A uniform approach was adopted for preliminary design of a functional module/first level CSC from the SRS. This involved the definition of a Ada package specification for each CSC, as well as Ada data/types packages specifying data exchanges between that CSC and other first level CSC's. Each first level CSC package specification contained procedures, which were called by the user of that package - namely, the MC scheduler in our case. When the procedures defined in first level CSC package specifications were called by the MC scheduler, they achieved the functionality defined in the SRS of that functional module/CSC. The package specification design was based on the concept of data abstraction and information hiding for which Ada language was eminently suitable. Data exchanges between various first level CSC's were strictly on the basis of formal parameters defined in the procedures in the package specifications. Sufficient care was taken to ensure that data element names etc. used in the SRS were retained to a large extent in the design for better traceability. Prior to preliminary design review, all package specifications were compiled and checked for absence of circular dependency. The design of each CSC was handled by one or two members of the design team.

9.1.2 DETAILED DESIGN

The Ada package bodies of the above-mentioned first level CSC package specifications constituted the detailed design. In many cases, the package bodies themselves contained one lower level package specification and their bodies. These were, of course, not visible to the MC scheduler. In detailed design, each first level CSC was further decomposed into lower level CSC's and each last level CSC was decomposed into several Computer Software Units (CSU's). A CSU was defined as the lowest level procedure/function which achieved a certain small but specific functionality. In addition, to Ada being used as a design language, the main design features of each CSC were recorded in English and formed a part of the design document.

9.1.3 <u>DESIGN EVALUATION</u>

During the design/code process, Design/code walkthrough's were conducted for

- Traceability checks from SRS to Design/code
- Correctness
- Completeness
- Compliance to design directives/coding standards
- Understandability

9.2 <u>TEST APPROACH</u>

The testing of the MC software was done at three levels:

a) <u>CSU level testing (done by the design team).</u> Each CSU was subject to white box testing. Each execution path in the CSU was identified and tested.

b) <u>CSC level testing (done by the Design V&V/testing team).</u> Each CSC was subject to black box testing and tested for functionality. The design V&V team independently generated test plans and test software for each functional module (first level CSC) on the basis of the SRS. Each design V&V team member handled the testing of 1-3 functional modules. The CSC level tests were first conducted on the HOST Computer (VAX) and then on the MC target. An Automated Test Software (ATS) has been developed to automate CSC level testing on both the Host and the Target. Using the ATS, multiple runs of the test CSC is possible; with independent sets of test data (input and expected output date) for each run. Testing of a few CSC's on the target took about 15-20 hours.

c) <u>CSCI level testing (done by the total software development team)</u>
MC CSCI level testing is black box testing on the target and was done using the MC test Rig and 1553B data bus monitors. After completion of CSC level testing on the target, the CSC's were integrated to form the CSCI and ported onto the target. CSCI testing was done in two stages:
    - In the first stage, the MC CSCI software was down loaded into the target from the VAX 3195 through an RS422 link with the Debug monitor running on the target. The software was then run and tested under debugger control.
    - In the second stage, the MC CSCI software was loaded onto the target EEPROMS as would be the case under actual operating conditions. Testing was then carried with the MC disconnected from the development environment.

Each cycle of MC CSC1 testing carried out by the development team as well as by the independent V&V team took about 3.5 to 4 months.

In the final MC CSCI testing 60 errors were detected in about 100,000 lines of Ada source code and an analysis of the type of errors detected is as follows:

Type of Errors	**Percentage of total errors**
System Design	32
SRS	33
Design/Code/CSC Test	20
ICD (1553B Interface Control Document)	10
Others	5

A significantly higher percentage of errors in System design/SRS was due to the inherently manual nature of system design and SRS even though a CASE tool had been used.

10. MC SOFTWARE CONFIGURATION MANAGEMENT

Software Configuration Management is absolutely essential for large software projects. To take care of configuration management, a Configuration Control Group (CCG), comprising of the MC Project Manager, Design Team Leader, Design V&V Team leader, Internal QA and Configuration Control Manager was formed.

10.1 DEVELOPMENT CONFIGURATION ITEMS

The following were the Development Configuration items used in the MC CSCI development:

1. Change notice for Software Requirements Specification
2. Software Design (Preliminary and Detail Design)
3. Requests for ICD (Interface Control Documents) Changes
4. CSU, CSC Test Description
5. CSCI Test Plan
6. CSCI Test Description
7. CSC Software Problem Report
8. CSC Software Problem Response Report
9. CSCI Software Problem Report
10. CSCI Software Problem Response Report
11. Source Code - File Header and Modification History
12. CSC Code Walk Through Reports
13. CSC Code Walk Through Response Report
14. Request for Change in Design
15. Change Notice to Design

16. CSC Test Completion Report
17. Test Rig Problem Report
18. Ada Program Library on Host used by Design team
19. Ada Program Library on Host used by Design V&V Team
20. Ada Program Library on Target used by Design and Design V&V Team
21. CSC wise Ada test Programs

10.2 FLOW OF CONFIGURATION CONTROL AND CORRECTIVE ACTION PROCESS DURING DCT

The Configuration Control during DCT started from the time the SRS was handed over to the Design and Design V&V Teams. The two teams operated in parallel, interacting whenever changes were required at any stage of the software development.

The Design Team studied the SRS and proceeded with the Preliminary Design, followed by the Detailed Design. In parallel, the Design V&V team also studied the SRS and prepared the CSC Test Plans and Test Descriptions, followed by CSCI Test Plan and CSCI Test Descriptions.

- Whenever any problems was noticed in the SRS for any CSC by either of the teams, Change Notice for the SRS for that CSC was raised in the appropriate format, by the team member in consent with the respective CSC member of the other team and the CCG.
- The Change Notice was then reviewed by the CCG. The CCG then approved/disapproved the change based on the technical merit, potential side effects, overall impact on other system functions and other related issues.
- If the change was approved, the Change Notice to the SRS was released, with copies of the same being distributed to all the affected members of Design team, Design V&V team, Design Leader, Design V&V Leader, Internal QA, Project Manager and Independent V&V.
- Once the change notice was released, the changes were affected in the design, test plans and test software.
- Informal reviews of the Design were conducted by the Design V&V team at various stages of the design.
- Formal Code Walk Through reviews were conducted by the Design V&V team at the completion of the base version of the Preliminary Design and Detail Design, and Code Walk Through Reports were generated by the Design V&V team.
- Based on the review of the Code Walk Through report, the Design Team prepared the Code Walk Through Response Report.
- If the designer wanted to change the design based on his own analysis or due to change in the design directives, he generated a Request for Change in Design.
- Based on the reviews by CCG, the designer then generated Change notice to Design.

- Any changes the designer made to the code after the base version, were reflected in the modification history of that file.
- Any changes needed in the 1553B data bus Interface Control Document were generated by either of the team with the consent of the CCG. The change was requested in the format for Request for ICD change.
- The request for ICD Change, after the approval from the CCG, was handed to the avionics group maintaining the Interface Control Document, which issued the change in the form of ICD Revision.
- Based on ICD revision history, the Design, Test Plans and Test software were updated.
- The design team tested the CSU during the CSC coding phase. The design team corrected any errors and no formal reports were generated.
- Once CSC coding was over and all the CSU's were tested, the code was then handed over to the Design V&V team for the CSC testing.
- The Design V&V team generated the CSC test programs and the databases based on the Test Plans.
- On receipt of the Code from the Design Team, the Design V&V team after testing the CSC generated Software Problem Report (SPR).
- The Design team then responded by generating the Software Problem Response Report (SPRR).
- After the SPR and SPRR were reviewed and released by the CCG, suitable changes notices were initiated for change in SRS, Design, ICD, Test Plan and Description, Test software and databases.
- Once the CSC testing was completed, the Design V&V team generated Test Completion Report for that CSC.
- All the Changes needed in the CSC Test Plan Document were consolidated and new revision for the Test Plan Document was released. The new release listed all the change notices taken care in that release.
- The CSCI test plan and descriptions were generated based on the SRS and ICD, supplemented by the Global Specifications, etc. Newer revisions of the CSCI Test plan and description documents were released to incorporate the Change notices.
- Any problems noticed during the CSCI testing were reported in the form of MC Software Problem Report.
- A response for the same was generated in the form of Software Problem Response Report. After the approval of the solution from the CCG, the appropriate change notices were generated as explained previously.
- The CSCI testing was performed using the Test Rig and problems found in the Rig were documented in the form of Test Rig Problem Report.

11. <u>MC SOFTWARE DEVELOPMENT EFFORT AND STATUS</u>

The total software development effort upto ver 3.0 level was about 150 man-years. About 40-45 engineers worked on the project during peak effort periods.

The embedded software developed in Ada language came to about 100,000 lines of source code and the total software developed including test code came to approx 4,50,000 lines. More than 2000 hours of software testing on the target has been done. The testing by Independent V&V team and CRE (type certification authority) has been completed and the MC has been delivered for Integrated AWS testing at the Dynamic Avionics Integration Rig.

12. EXPERIENCE GAINED FROM THE MC SOFTWARE DEVELOPMENT PROJECT

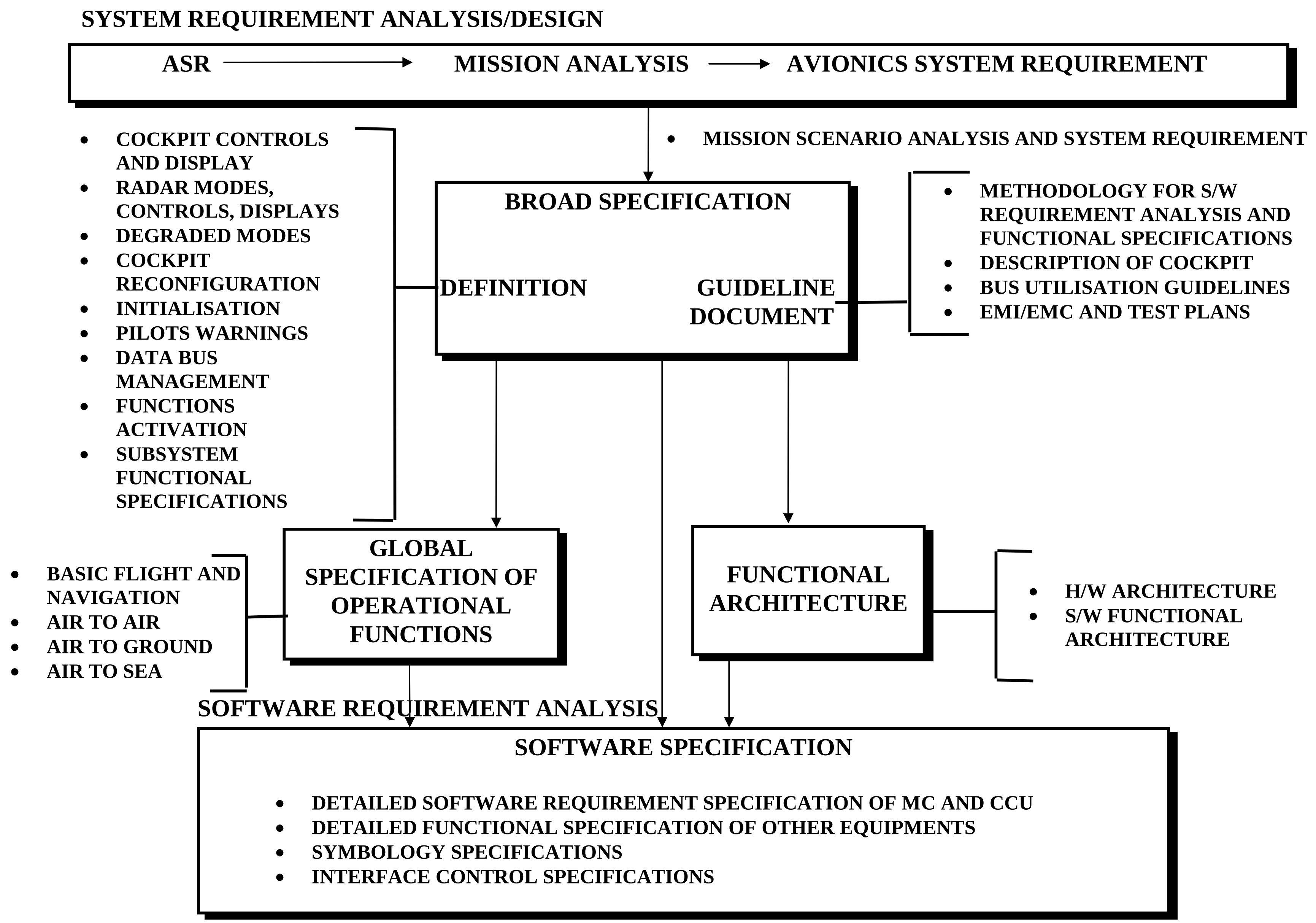
For the success of large and complex software development projects, the following are considered essential:

- Good Project Management and leadership
- Teamwork with engineers mutually supporting each other and communicating with each other.
- A comprehensive and effective software development process control with no short cuts.
- Testing engineers/teams to be independent from Design engineers/teams.
- Automating the testing process as far as possible
- Effective Internal QA different from design/test teams with adequate experience and peer acceptance
- Effective Configuration Management
- Effective error correction mechanisms/processes
- Documented Design directives/Test directives for communication when large teams with different backgrounds are working together and personnel attrition rates are high.
- Good design
  - Amenable to modification and growth
  - Testable
  - Maintainable

Ada language has been found to be extremely suitable for large embedded projects for the following reasons:

- Preliminary designs can be compiled. This minimises integration problems later on.
- Readability of design/code
- Strong type checking
- In-built exception handling
- Packaging concept: For Data abstraction, information hiding and, programming in the large.

AVIONICS AND WEAPON SYSTEM
SOFTWARE DEVELOPMENT PROCESS (UPTO SOFTWARE SPECIFICATION LEVEL)



SYSTEM REQUIREMENT ANALYSIS/DESIGN
ASR
MISSION ANALYSIS
AVIONICS SYSTEM REQUIREMENT
• MISSION SCENARIO ANALYSIS AND SYSTEM REQUIREMENT
• COCKPIT CONTROLS AND DISPLAY
• RADAR MODES, CONTROLS, DISPLAYS
• DEGRADED MODES
• COCKPIT RECONFIGURATION
• INITIALISATION
• PILOTS WARNINGS
• DATA BUS MANAGEMENT
• FUNCTIONS ACTIVATION
• SUBSYSTEM FUNCTIONAL SPECIFICATIONS
BROAD SPECIFICATION
DEFINITION
GUIDELINE DOCUMENT
• METHODOLOGY FOR S/W REQUIREMENT ANALYSIS AND FUNCTIONAL SPECIFICATIONS
• DESCRIPTION OF COCKPIT
• BUS UTILISATION GUIDELINES
• EMI/EMC AND TEST PLANS
• BASIC FLIGHT AND NAVIGATION
• AIR TO AIR
• AIR TO GROUND
• AIR TO SEA
GLOBAL SPECIFICATION OF OPERATIONAL FUNCTIONS
FUNCTIONAL ARCHITECTURE
• H/W ARCHITECTURE
• S/W FUNCTIONAL ARCHITECTURE
SOFTWARE REQUIREMENT ANALYSIS
SOFTWARE SPECIFICATION
• DETAILED SOFTWARE REQUIREMENT SPECIFICATION OF MC AND CCU
• DETAILED FUNCTIONAL SPECIFICATION OF OTHER EQUIPMENTS
• SYMBOLOGY SPECIFICATIONS
• INTERFACE CONTROL SPECIFICATIONS

Fig. - 1