

Development of High Reliability Control Software for HF120 Turbofan Engine

Makoto TEZUKA*

Shohei SUGIMOTO*

Keisuke KAWAI*

Hideaki JINNO*

Yuta AKAI*

ABSTRACT

When developing application software for an aircraft engine electronic control system, the development process is required to comply with DO-178B, one of the guidelines issued by the Radio Technical Commission for Aeronautics. All activities to be conducted during software development should be defined by each development organization, and Honda developed the software based on the process defined by Honda.

The application software was developed separately during the design phase consisting of requirement definition, software design and software implementation. The verification phase consisted of review, testing and analysis. In addition, Honda established the development system for the project tailored for the joint development between Japan, USA and Mexico with international time differences. Documents and design data could be shared and managed while maintaining the integrity within the development system for this project.

As a result, Honda established a highly reliable software development process for the HF120 turbofan engine control system in compliance with DO-178B. Jointly with General Electric Company, Honda obtained Type Certificate issued by the Federal Aviation Administration for the HF120 turbofan engine, including the engine control system software.

1. Introduction

At turbofan engine for an aircraft generally has a long service life, and durability is necessary for its control system components. The control system components control the engine appropriately based on the engine condition, even if a sensor fails. Therefore, a Full Authority Digital Engine Control System (FADEC) was applied as a control system for the HF120 turbofan engine (HF120). Honda co-developed the control system with General Electric Company (GE). The HF120 control system has engine control characteristics in its software so that change requests from an airframe manufacturer or the Federal Aviation Administration (FAA) can be accommodated during the service periods.

The Fuel Pump Metering Unit (FPMU), the Engine Control Unit (ECU) hardware and the Operating System Software (OS), the Application Software (AS) installed in the ECU are components for the overall control system. Honda was responsible for developing the AS, validating overall system integration, and confirming engine compliance.

The control system software development is used to satisfy DO-178B⁽¹⁾, the de-facto standard for software development process to assure airworthiness. DO-178B specifies certain activities to be performed during software development. Each development organization or team should define how to perform their tests, and the software development data generated by its process is used as objective evidence to demonstrate conformance of the overall software development using DO-178B.

This paper describes how Honda constructed its software development process to enhance reliability of the control system software.

2. Control System Overview

2.1. Aircraft Engine Control

Turbofan engines in modern jet aircraft need high precision thrust control. Engine controls also need to ensure thrust response capability for engine accelerations and decelerations with no compressor surge, stall and lean blowout.

The engine control system determines the necessary fuel

* Aircraft Engine R&D Center

flow rate as a function of thrust demand from aircraft and engine conditions, ambient conditions, aircraft operating conditions and other factors. Fuel flow rate is regulated by changing the fuel flow passage area determined by the position of the Fuel Metering Valve (FMV). Since turbofan aircraft engines are operated in high altitude with low air density, sufficient fuel control accuracy is needed for both low flow rate regions at high altitude and high flow rate regions closer to the ground.

2.2. Control System Architecture

Figure 1 shows the HF120 Control System Architecture. The system consists mainly of ECU, engine sensors, FPMU and the airframe interface.

Electronic components including the ECU and various sensors are designed to provide high reliability. Each channel of the ECU compares the health level of mutual electronic system via Cross Channel Data Link (CCDL) to control the engine. Even when the ECU has a malfunction in one of the two channels, usage of the faulty channel is immediately suspended and switched to the remaining healthy channel to continue proper engine control.

The FPMU is a key component to ensure the control function. The main functions of the FPMU are boosting of fuel pressure and supply of fuel to the engine combustor. The FPMU has an FMV to control fuel flow, a Bleed-Off Valve (BOV) Actuator to avoid surge and stall, and a Shut-Off Valve (SOV) to cut-off fuel in emergencies, all build in. Furthermore, the FADEC is operated by a dedicated power source of the Permanent Magnetic Alternator (PMA) built-in FPMU, and needs no external power source except for starting.

Airframe information for thrust control such as thrust lever position and cockpit switch signal is transmitted to the ECU through the airframe interface. Signals indicating engine running conditions are sent to the airframe through the airframe interface.

3. Overview of Developing Highly Reliable Software

3.1. Control System Requirements

The control system for an aircraft engine is required to meet the Federal Aviation Regulations of the United States, where airworthiness standards for engine control are described. Safe operation, fundamental control response of engine thrust, electromagnetic interference protection and reliability are specified.

Aircraft engine specifications are issued by an airframe manufacturer, and aircraft engine interface control documents are issued by the engine manufacturer. Based on these documentations, the engine requirements are prepared by the engine manufacturer. These engine requirements are flowed down to the control system requirements, then further down to software requirements and design detail specifications.

Development of all components including software is accomplished based on these requirements.

3.2. Software Development Method

Software development complied with the V model is shown in Fig. 2. Design flow is indicated on left side while verification flow on the right.

The result of an individual work project indicated by the V model is managed by a configuration management

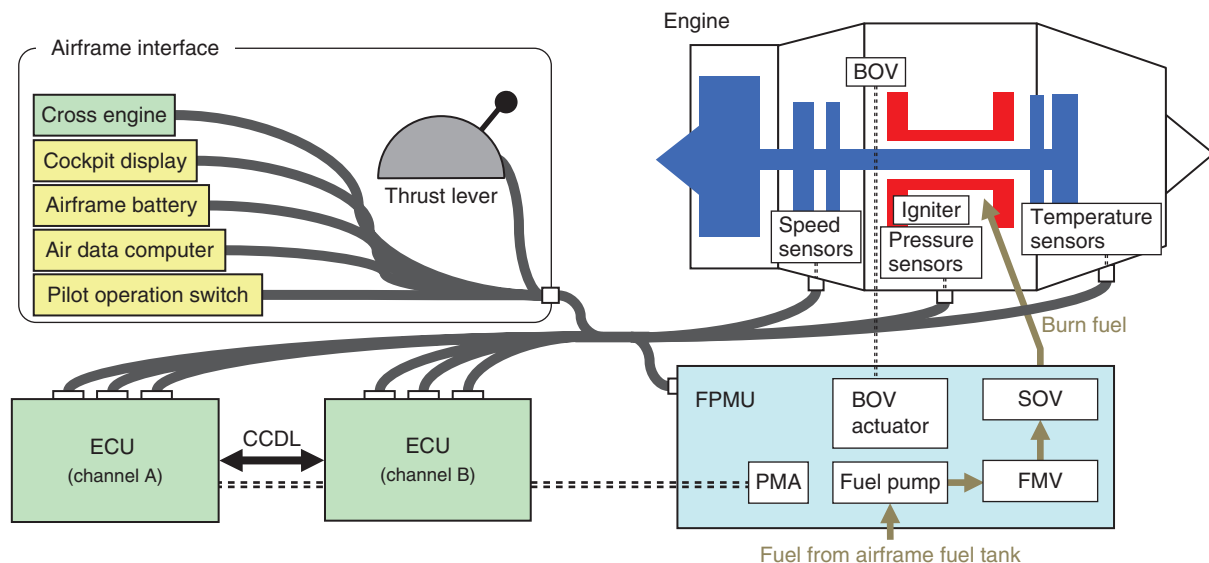


Fig. 1 FADEC system

database. The configuration management is composed of software release control, version control and change control for software elements. This is intended to record work traceability with respect to all work.

3.2.1. Software design technique

When receiving the defined engine level requirements, FADEC development starts by defining the control system requirements. Functional and non-functional requirements anticipated in FADEC need to be clearly stated and various types of interface specifications are set forth so as to define hardware and software requirements.

In the HF120 development, requests from customers were defined as part of the control system requirement. Then, a software requirement was flowed-down from the control system requirements.

Verification tasks in compliance with DO-178B are required to show 100% coverage of the software requirements by testing. Therefore, the software requirements were prescribed accurately following the method which is commonly accepted by the software engineering, so that all requirements are verifiable.

Afterwards, design and implementation were conducted based on the software requirements. In the HF120 development, Matlab/Simulink and Unified Modeling Language were used to design the software behavior, and those models were implemented in C language. Any model described by Simulink was implemented by using automatic code generation tool.

3.2.2. Software verification method

Verification activities occupy the majority of software development, and the verification method affects the reliability of control system software. The verification activities generally consist of review, testing and analysis. Main verification activities include code review, unit test, software integration test, system integration test and engine validation test as shown in Fig. 2. These activities help ensure the integrity of the verification process.

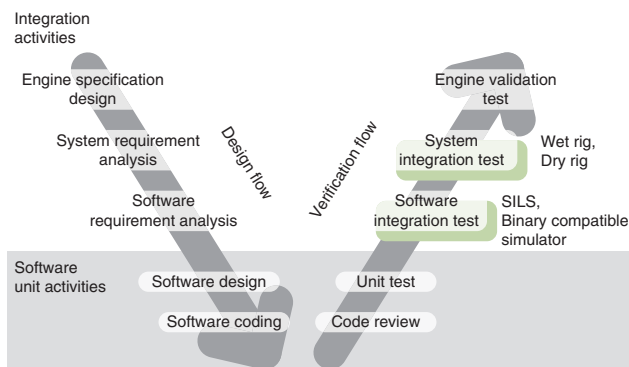


Fig. 2 Software development flow

4. Software Verification Method and Environment

Four types of environment for integration test were provided during the HF120 development, and each test had its optimum testing environment selected to conduct the verification.

Integration test environments can be classified into software integration test environment and system integration test environment. Modeling scope for each integration test environment is shown in Fig. 3. Software In the Loop Simulation (SILS) and binary compatible simulation can be classified as software integration test while dry rig and wet rig can be classified as system integration test environment.

SILS is a test environment modeling all elements of the FADEC system, making closed-loop simulation with the engine plant model possible. On the other hand, binary compatible simulator which incorporates no engine plant model is structured as an open-loop testing environment modeling only ECU hardware and OS.

The dry rig is a testing environment combining the ECU and a simulator, which can electrically drive the ECU on the basis of engine plant model computing result, and can simulate the ECU by closed loop. The wet rig is an environment where actual fuel can be flown through the FPMU, and can simulate a combined ECU and FPMU by closed loop.

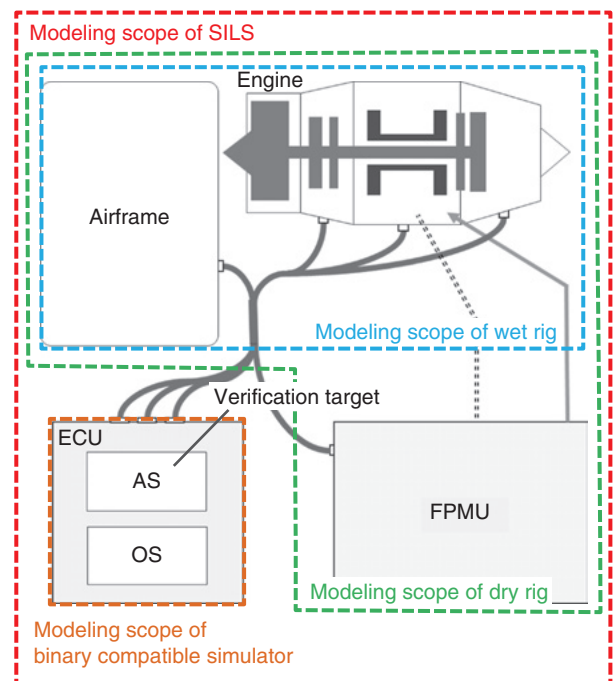


Fig. 3 Modeling scope

4.1. Software Integration Test

4.1.1. SILS test

In order to evaluate software accurately, specific environment was designed to simulate engine and software behaviors in nominal conditions by excluding the characteristics unique to hardware as well as noise characteristics of simulator equipment and the ECU. The noise injection test also needs a simulated environment that can precisely inject noise characteristics into the software. A verification method using SILS was introduced to meet these requirements for the simulation environment.

SILS was used for the verification related to thrust control and noise immunity during the verification process. For the thrust control related verification, SILS was constructed in parallel with the completion of software coding, and overall engine operating conditions including altitude and cabin bleed were simulated by SILS to confirm that every control response meets the design requirements. For the next step, the same conditions were tested in the dry rig including an actual ECU to evaluate if the difference between the test and the control response obtained from SILS was within the tolerance. It was demonstrated through these two steps that the software requirements are satisfied in nominal system conditions and, furthermore, the same software satisfies the requirements on the actual ECU.

For noise protection verification, SILS characteristics were utilized, making it possible for any reasonably conceivable noise pattern to be precisely injected into the software. Also, with test repeatability being available, software noise immunity properties were verified with accuracy and efficiency.

4.1.2. Binary compatible simulator test

Some software requirements describe details dependent on hardware characteristics, for example, access timing for communications and storage. Confirmation of such requirements by testing necessitates an environment to accurately simulate hardware behaviors. A binary Compatible Simulator was prepared to meet the following objectives in AS verification:

- (1) Ability to generate ECU internal faults at any given point in time.
- (2) Ability to send arbitrary patterns of communication data to ECU at any given point in time.
- (3) Ability to virtually confirm the internal condition of the storage device at any given point in time.
- (4) Ability to run an AS executable file to be loaded in an actual ECU with binary level compatibility, leading to an AS processing result in complete agreement with actual ECU.

This test environment was established on the CPU instruction set simulator by integrating actual AS and the other simulated component models. OS, ECU hardware, parts of the engine and engine diagnostic tools were subjected to modeling.

Under this environment, an ECU internal fault can be generated at arbitrary times, and test coverage can be obtained in response to the requirement for accommodation of the fault. The CCDL fault injection test is explained as a representative example. ECU hardware carries two sets of redundant boards inside of a single chassis to share control data between these boards by using CCDL communication. This environment modeled such dual redundant hardware and CCDL, simulating disconnection and data corruption for CCDL. Because of this, system robustness can be confirmed by encompassing fault occurrence patterns.

Other than the aforementioned advantages during the verification, usefulness of this environment in parallel development of AS and OS has also been recognized. In the HF120 development, the OS and AS interface was defined in advance. Then, AS development was executed by using an OS modeled by OS and AS interface specifications, and a simulative integration test was performed by the modeled OS prior to integrating AS and the formal release OS. Any interface inconsistency during the formal release OS acceptance test could smoothly be addressed by executing this advance integration test. Specifically, any difference in behavior between the formal release OS and the modeled OS can be tracked down by comparing the simulation results of these OSs, providing immediate determination of whether the difference comes from the unmet requirements or an interface definition issue.

4.2. System Integration Test

4.2.1. Dry rig test

The dry rig test uses the actual ECU hardware where OS and AS are loaded. In this test environment, software behavior can be verified on the hardware. For this reason, open and short-circuit test on the dry rig make the verification of the complex propagation of the faults in the ECU hardware and software possible.

Figure 4(a) shows the dry rig configuration. The simulator inputs an electrical signal to the ECU on the basis of engine model computing results. Output signal from the ECU becomes input to the engine model.

In order to increase reliability of the system integration test, approx. 90% of the test cases were automated in the dry rig test. Not only automated operation of the simulator but also version information such as ECU hardware and OS, data storage for statistical analysis and test report preparation using tables and charts were automated in the HF120 development. Establishment of such an automated technique helped achieve high software reliability from the following perspectives:

- (1) Ability to perform the test repeatedly which is hard to do manually. This makes evaluation possible if the uncertainty of the hardware characteristics is within the acceptable range, and assures the robustness of the ECU given the test result.

- (2) Ability to control multiple ECU inputs at short intervals, which is hard to set up manually. This makes full verification of complex and very rare timing possible.
- (3) Ability to perform testing that assures repeatability. This makes the direct comparison of test results between different AS versions possible, and then identifies the functional area affected by software change in the development phase. The impact analysis from the test was also applied when there was a change in OS or ECU to diagnose and maintain the functionality of an integrated system.
- (4) Achieved high quality of the test by standardization and automation. Specifically, the standardization was applied to the method to determine input conditions that can meet requirement-based test coverage, setting acceptable tolerances for pass or fail parameters, and furthermore, report format and how to display signals in charts. These standardized processes were then incorporated into the libraries so that many testers can import any standardized process.

4.2.2. Wet rig test

Verifications of FPMU-related FMV control and SOV control are conducted on SILS environment modeling FPMU or the dry rig. A wet rig was constructed to verify the combined actual FPMU and software as a final verification prior to full engine testing.

As shown in Fig. 4(b), the wet rig consists of an actual FPMU, actual ECU and fuel equipment supplying actual fuel to the FPMU, throttle valve simulating a fuel nozzle, test chamber, motor to drive the FPMU, and engine model simulator. In addition, the fuel equipment and test chamber simulate an aircraft environment so that low-to high-temperature control of the fuel can be made, and together with the engine model simulator, it can simulate various types of FPMU operating conditions.

The wet rig was used to conduct verification of the

FPMU-related control functions prior to engine testing, providing accurate and efficient verification in a given operating condition.

4.3. Analyses

4.3.1. Analysis of requirement-based test coverage

In order to secure AS reliability, a test should demonstrate 100% coverage of the requirements. The coverage analysis was performed in two steps to show this completeness.

First, if the test was prepared to satisfy all requirements, then all requirements being testable were confirmed. Test completeness with respect to individual requirements was confirmed by focusing on the following three perspectives:

- (1) Case Scenario – Find out if multiple likely cases have been tested. Engine start conditions, for example, takes into account of the case where initial condition is an inflight restart at high altitude.
- (2) Boundary – When assuming two conditions, determine if static behavior at each condition (as well as dynamic behavior during the transition to another condition by crossing the boundary) have been tested. Parameters indicating flight condition, in particular, altitude, Mach number and ambient temperature were rigorously examined.
- (3) Stress – In addition to normal conditions, failure condition must also be tested. By doing so, the equivalent behavior (as in normal condition) could be obtained or proper processing to address whether the failure condition would be activated was confirmed.

If any insufficient coverage was detected during the analysis process, appropriate test details were added. As an example, assume that an additional testing was deemed necessary for an ECU hardware internal fault after making a coverage analysis for the requirements on a dry rig test. In this instance, certain test items in the binary compatible simulator environment were added to secure the full coverage.

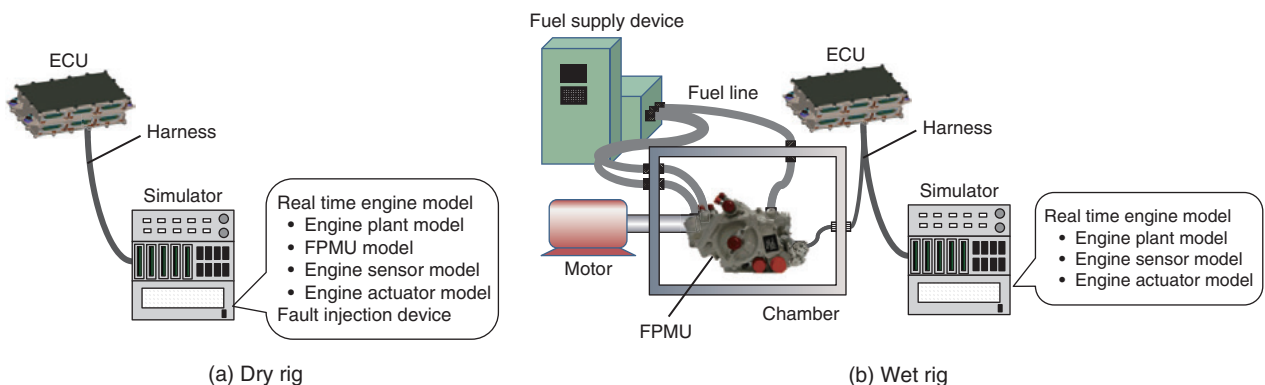


Fig. 4 Dry rig and wet rig

4.3.2. Execution time analysis

When a real-time control system fails to complete a task within the intended period of time, it may have an impact on its control target. System on Chip (SoC) used in recent embedded systems has high-speed processing functions such as multi-core, cache and various types of co-processors built-in as hardware. Use of these functions greatly improves calculation performance. In light of this, the following analysis was performed in order to derive the maximum execution time a task can take on an actual ECU.

First, an environment that can simulate SoC internal condition and its instruction set was prepared and used for the analysis of AS executable files. A software execution path that could cause maximum execution time was statically calculated by testing all branch points. Memory image causing the task to trace the execution path was also obtained by the analysis. This memory image was injected into an actual ECU to acquire actual task execution time and software execution path. After confirming the consistency between this software execution path and what was derived by the analysis, measured execution time was identified as the maximum execution time.

Implementation of engine maintenance functions was initiated in the HF120 during a stage where CPU throughput margin was small. In order to control the CPU throughput margin during development, this analysis was continuously conducted. This analysis also calculated CPU occupancy rate for individual software function by post-processing. When a heavily-loaded function is implemented beyond necessity, such information is visualized for our team to

be able to share. In order to reduce CPU loading, the team discussed various countermeasures and dedicated design efforts. As a result, even at the occurrence of unexpected random events, this software completes the individual tasks within the intended period of time.

5. Software Configuration Management

A life cycle in the HF120 software development is triggered by the system requirement change representing a customer or other request.

This system change request is broken down into requirement change and software design change so as to manage progress at each stage.

Software release is recognized as an aggregate of system change requests so that any change incorporated into the release, software design constituting the release, and various types of deliverables during validation can be identified.

GE and Honda have established a Configuration Control Board and made acceptance or rejection decisions for the execution of the system level requirement changes by setting certain criteria.

The software configuration management scheme is shown in Fig. 5, and the description of individual elements supporting the scheme is provided below.

5.1. Requirements Configuration Management

All requirement-related documents used in AS development were prepared using a document management tool. Introduction of a document management tool was

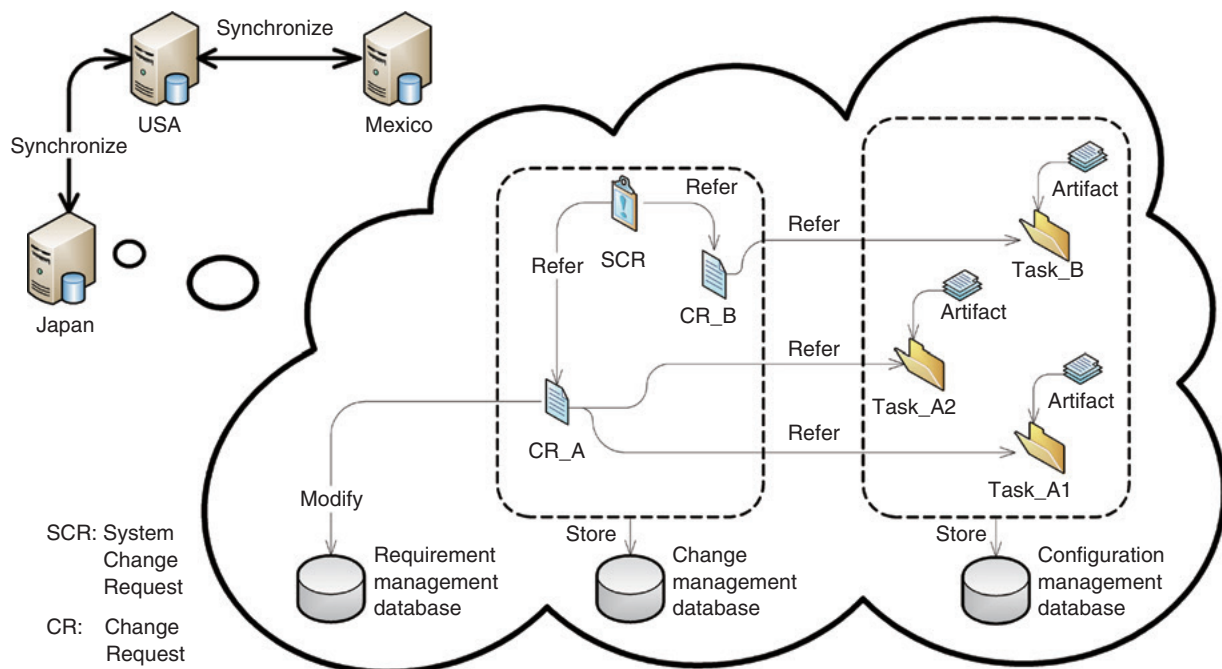


Fig. 5 Software development workflow

deemed important in order to cope with daily changes and keep the link between the affected documents updated. When a definition of a certain requirement is changed, this tool helps visualize the change impact across the related requirements. Other than identifying the statement of the requirement and the reason for its existence, how to verify the requirement was also defined within the database of the requirement specification. Because of this, a designer and verifier can reach agreement on verifiability at the preparation phase of requirement specification. Moreover, the document management tool is also used by the ECU supplier so that design activities interface specifications are objectively traced.

5.2. Software Life Cycle Data Configuration Management

Software is designed and updated using software design change requests, the design artifacts are registered with the revision history database. Change traceability information is prepared to show which software module was changed, how it was changed, and which change request prompted this change. This process shows where certain release is designated, for what purpose, and how it compares to the previous release. Configuration management information also manages system-level change descriptions, software change detail descriptions, and changes in software implementation level in coordination so that the desirable level of information can be extracted. Therefore, all software development members can utilize the configuration management information to better understand the related software's design history.

5.3. Sharing Configuration Management Database

The configuration management information indicates the latest status of development targets. Such information should be promptly shared between development organizations. For AS development, this has been achieved by scheduled database synchronization between companies in Japan, the U.S. and Mexico. As an example, code review and unit test of the software developed in Japan have been performed in Mexico. The synchronized configuration management database helped ensure the integrity of the software correction cycle performed in between Japan and Mexico.

This distributed database system is structured to recover the data from that at the other site even in the case of database deficiency at another site. Moreover, each site prepares a disaster recovery plan including off-site backup. Availability of respective sites and complementary backup between sites provide assurance of overall reliability.

6. Conclusion

This paper provided a general description of key processes to enhance the control system software reliability that constitutes the core of the HF120 Control System.

Honda flowed down the complete requirements during AS development, and achieved 100% coverage of established requirements by verification activities, resulting in the development of highly reliable software.

The software integration test focusing on the software and the system integration (including the hardware) were conducted in accordance with the intended use. These tests were complemented by analysis, verification, and coverage and reliability could be enhanced.

During the joint development across three countries, Honda established the software development system to rigorously manage software configuration and maintain high reliability of the control system software.

Honda added our own original ideas to the requirement creation, verification process and configuration management through the HF120 control system software development, resulting in a highly reliable software development process.

Reference

- (1) DO-178B, Software Considerations in Airborne Systems and Equipment Certification, RTCA, Incorporated. (1992)

■ Author ■



Makoto TEZUKA



Shohei SUGIMOTO



Keisuke KAWAI



Hideaki JINNO



Yuta AKAI