

Electronic Control Modules Testing & Validation Practice

Submitted by



**687, 16th Main, 4th T Block
Jayanagar, Bangalore – 560 041
91-80-22445466 / 22240404**

TABLE OF CONTENTS

1	Introduction.....	3
2	ECU Software Testing	3
2.1	White Box Testing	3
2.1.1	Scope	3
2.1.2	Unit Testing Design Techniques	3
2.1.3	Unit Testing Practice	3
2.2	Black Box Testing	4
2.2.1	Scope	4
2.2.2	Black Box testing Techniques.....	4
2.3	Unit testing.....	5
2.4	Integration Tesing	5
2.5	Functional and System testing	6
3	DE Skills	7
3.1	Team members	7
3.2	Expertise	8
4	DE Offerings.....	9
5	Case Studies	10

1 Introduction

As software explodes in complexity and size, comprehensive ECU tests are necessary more than ever before. Since we do know the key to delivering bug-free software is to test early and test often. Only a tough zero-error policy can help avoid vehicle recall campaigns. So for many manufacturers and suppliers, ECU testing has become a key phase in the development process. This document provides DE's Testing and Validation practice for Electronic Control Units (ECUs) used for Automotive vehicles.

2 ECU Software Testing

2.1 White Box Testing

2.1.1 Scope

DE is capable of performing most aspects of white box testing as outlined in this document.

While white box testing is applicable at the unit and integration. So while it normally tests paths within a unit and it can also test paths between units during integration. Though this method of test design can uncover an overwhelming number of test cases, it might not detect unimplemented parts of the specification or missing requirements. But you can be sure that all paths through the test object are executed.

Since the number of test and effort involved is not proportional to the test outcome, DE practices to limit White box testing for an Unit and extends in some cases to Integration level.

2.1.2 Unit Testing Design Techniques

- Data Flow testing
- Control Flow testing
- Static test for MiSRA compliance using QAC and Code inspection

2.1.3 Unit Testing Practice

- Path testing using Flow graph notation, Cyclomatic Complexity, Myers interval.
 - Cyclomatic complexity gives a quantitative measure of the logical complexity and number of independent paths.
 - Cyclomatic complexity provides upper bound for number of tests required to guarantee coverage of all program statements.
- Control Flow and Data Flow testing to cover Relational expressions, Boolean expression, Compound condition, Loop testing, Boundary checking and Data integrity test in a Unit

2.2 Black Box Testing

2.2.1 Scope

DE is capable of performing most aspects of Black box testing as outlined in this document.

Black Box testing is done by the tester independent of designer. Tester shall test the compliance to requirements and from User perspective. Tester can design the test as soon as the requirements are available.

Since the development team is having good knowledge of the system internal logic, it's becoming an industry practice to have third party testing. DE has the set of specialized and skillful resources to cater black box testing needs for the Automotive industry.

We practice black box testing techniques to identify good enough test cases to verify and validate.

2.2.2 Black Box testing Techniques

- Smoke testing
- Testing Customer Requirements
- Equivalence partitioning
- Boundary Value analysis
- Decision Table testing
- Scenario testing
- Risk Testing
- Combination testing
- Test Early and Often
- Regression testing
- Random testing
- Domain testing
- Stress testing
- Crash Testing

2.3 Unit testing

Unit testing is to verify the implementation against design. It implies that the test engineer needs to know the complete functionality of a particular module under test.

Unit testing isolate each part of the program and show that individual parts are correct. Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly (i.e. regression testing).

Unit test design produces test cases also need to cover paths through the unit with attention paid to loop conditions and Boundary condition. There are different ways of measuring code coverage, the main ones being used are

- i. *Statement Coverage* (Has each line of the source code been executed and tested?)
- ii. *Condition Coverage* - Has each evaluation point (such as a true/false decision) been executed and tested?
- iii. *Path Coverage* - Has every possible route through a given part of the code been executed and tested?

Unit testing helps eliminate uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.

2.4 Integration Testing

Integration testing is used to test when individual software modules / units are combined and as a group (assemblages).

Assemblages are said to be integrated when:

- a. They have been compiled, linked, and loaded together.
- b. They have successfully passed the integration tests at the interface between them.

Tester needs to get the High level Logic and data flow of the assemblage under test from the developer to test the interaction between the Units in the assemblage.

Integration testing identifies problems that occur when units are combined and the purpose of **Integration testing** is to verify functional, performance and reliability requirements placed on **major design items**. These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using Black box testing, success and error cases being simulated via appropriate parameter and data inputs.

All test cases are constructed to test that all components within assemblages interact correctly. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the **Integration testing** of further assemblages.

Integration testing is done in following three common strategies based on the nature of the program.

- The top-down approach to integration testing requires the highest-level modules be test and integrated first. This allows high-level logic and data flow to be tested early in the process and it tends to minimize the need for drivers. However, the need for stubs complicates test management and low-level utilities are tested relatively late in the development cycle. Another disadvantage of top-down integration testing is its poor support for early release of limited functionality.
- The bottom-up approach requires the lowest-level units be tested and integrated first. These units are frequently referred to as utility modules. By using this approach, utility modules are tested early in the development process and the need for stubs is minimized. The downside, however, is that the need for drivers complicates test management and high-level logic and data flow are tested late. Like the top-down approach, the bottom-up approach also provides poor support for early release of limited functionality.
- The third approach, sometimes referred to as the umbrella approach, requires testing along functional data and control-flow paths. First, the inputs for functions are integrated in the bottom-up pattern discussed above. The outputs for each function are then integrated in the top-down manner. The primary advantage of this approach is the degree of support for early release of limited functionality. It also helps minimize the need for stubs and drivers. The potential weaknesses of this approach are significant, however, in that it can be less systematic than the other two approaches, leading to the need for more regression testing.

2.5 Functional and System testing

Function testing is a technique used to test the functionality specified in the requirements specification. System testing is a technique used to test the program performance when integrated with the complete system. This test evaluates the program performance when integrated with the entire system with program specific requirements.

These Black Box testing techniques attempts to find errors in the external behavior by

- Identify Missing functionality
- Identify interface errors in the system
- Identify Performance issues
- System Initialization and termination issues
- Verify compliance to Customer requirements

DE can do functional and system testing with network simulation using CANoe. DE has the expertise to do automated testing using piAutoSim.

3 DE Skills

3.1 Team members

- DE has a primary focus of retaining the best talent available team members
- DE has dedicated team to performing recruitment and Human Resources (HR) functions.
- Suitable candidates with the necessary mindset are hired through a rigorous process of written exams, interviews and reference checking.
- Since white box testing requires a deeper understanding of the software development process, software engineers with a development background are placed on white-box testing assignments for the duration of a project.
- Since Black Box testing requires tester with no coding knowledge, good capability to understand the requirements, Domain knowledge, ability to identify good enough test case and skill to do random test and Risk test, Test engineers with such capabilities are recruited.

Roles	Responsibility	Deliverables
Project Manager	<ul style="list-style-type: none"> ○ Evaluate any client-specific process / method for testing, test data collection for testing. ○ Formulation of Entire Project Plan ○ All commercial communication with the client ○ Quality assurance at project level ○ Risk Management 	<ul style="list-style-type: none"> ○ Project Plan ○ Risk Document ○ Commercial
Project / Technical Lead	<ul style="list-style-type: none"> ○ Scheduling and tracking of activities as per project plan ○ Managing the team assigned to the project ○ Establish Test strategy and incorporating client-specific process / method for testing and test report generation. ○ Interfacing with Customer on technical aspects ○ Verify Test Environment ○ Interfacing with support services ○ Quality assurance at programmer level 	<ul style="list-style-type: none"> ○ Test Strategy ○ Test Plan ○ Status Report ○ Module Validation Report ○ Validation summary report ○ Any specific deliverable as requested by the client.
Project team member	<ul style="list-style-type: none"> ○ Requirements Analysis ○ Test Environment setup ○ Analyzing the Design ○ Writing Test Procedure ○ Execute Unit Test and generate Unit Test report 	<ul style="list-style-type: none"> ○ Internal to DE
Configuration Control	<ul style="list-style-type: none"> ○ Configuration control of the hardware and software environment ○ Assurance of proper working conditions ○ Maintenance and secrecy control 	<ul style="list-style-type: none"> ○ Internal to DE

Testing & Validation Practice

	<ul style="list-style-type: none"> for software o Assist in test environment setup 	
Quality Assurance	<ul style="list-style-type: none"> o Assisting Project Leader in quality assurance o Periodic audit of project to ensure compliance o Ensuring good data collection o Analyzing data and providing feedback o Suggesting process improvements 	<ul style="list-style-type: none"> o Internal to DE

3.2 Expertise

DE has the expertise to use the following toolsets that help in White box and Black Box testing of ECUs.

Testing & Validation Tools: LabVIEW, Quality Assurance C (QAC), Rational Test RealTime (RTRT), PiAutoSim

Network Interface Tools: Vector CANalyzer, CANoe and Dearborn Group Gryphon / Hercules

Activity	Result	Tools Used	Purpose
Code Review / Static Analysis	Analyzed Code	Quality Assurance C (QAC) with MISRA	Calculate Cyclometric Complexity(CC) and Mayer's Interval (CC) Conformance with MISRA and ISO Guidelines
Unit Testing using test scripts	Tested Code for Integration and test reports	Rational Test RealTime (RTRT)	To write the test scripts to cover 100% paths and performance profiling
Functional Testing	Tested code binary and test reports	IAR Embedded Workbench	Code compilation and generating binary
		Trace32 debugger	To flash the code binary and debugging
		CANalyzer, CANoe, DG Gryphon	To test network interfaces and simulate the system Network
Validation Testing	Validated code	ClearQuest	To log the defects for all baselines.
Configuration Management	-	Multisite Clearcase	

4 DE Offerings

DE is currently focused on supplier based ECU testing to support following needs of ECU development.

- Support OEM Supplier on White box testing, in turn optimizes the development time.
- Support on Regression test. In turn supports development team to focus on next incremental build.
- Third party independent evaluation of the ECU software against requirements.

DE offers following supplier based ECU testing services:

- Unit Testing (White Box Testing)
 - To verify Implementation against design
 - To verify code compliance with MISRA and ISO Standards
 - To test the code for path coverage and Performance profiling
- Functional and System Testing (Black Box Testing)
 - To test the ECUs for all features as specified in requirements
 - Simulate inputs using test boxes and Network tools
 - Observe the output on Network / visual inspection

Note:

Developer and tester have to exchange information at both Unit and Integration testing with proper documentation to accomplish the goals of White Box testing.

Tester and Subsystem has to exchange information for functional and system testing with proper documentation to accomplish the goals of Black Box testing

In general no test by itself is complete, to maintain the quality of the program, since we know we can't cover all the possible tests with available time and Cost, DE practice to identify Good enough test cases.

At DE we use Virification and Validation practice throughout the software development lifecycle to meet quality objectives.

5 Case Studies

Testing of DHMI Cockpit Integrated Display for Visteon

- Functional testing
 - DE was involved in requirements analysis
 - DE was involved in developing the test procedures
 - DE was involved in setting up test environment using CANoe configuration and CAPL scripts provided by Ford
 - DE was involved in Black box testing
 - DE was involved in submitting the Validation summary report
- Unit testing
 - DE was involved in Design analysis
 - DE was involved in development of Unit test procedure
 - DE was involved in executing test cases using Trace 32 debugger on the target
 - DE was involved in providing the Validation summary report

Testing of P356 Instrument cluster for Visteon

- Unit testing
 - DE was involved in generating the State Flow diagrams
 - DE was involved in analyzing the design
 - DE was involved in generating the Test Procedure
 - DE was involved in RTRT scrip generation and test
 - DE was involved in providing the Validation summary report
- Functional Testing
 - DE was involved in reviewing the Test procedure against requirements
 - DE was involved in correcting the Test procedures
 - DE was involved in executing the Test procedure
 - DE was involved in submitting the Validation summary report

Unit testing of Diesel System for Yuchai and NDM

- DE was involved in static analysis of Code and Simulink based models
- DE was involved in Analysis of Design
- DE was involved in development of Unit test procedure
- DE was involved in RTRT scrip generation and test
- DE was involved in providing the Validation summary report

Unit Testing of Engine / Transmission Control Software Modules for Delphi

- DE was involved in Unit testing of following modules
 - Fuel Injection
 - Throttle Position Control
 - Ignition
 - Torque Drivability
 - Torque Controllability
 - Spark Control
 - Idle Air Control
 - Idle Speed Control
 - Exhaust Gas Recirculation
 - Adaptive Cruise Control
- DE was involved in TDP development and setting up the test environment
- DE was involved in Analysis of Design
- DE was involved in development of Unit test procedure
- DE was involved in RTRT scrip generation and test execution
- DE was involved in submitting the Validation summary report

Unit testing of GMX 365 Body Control Module for Visteon

- DE was involved in development of TDP on this project. Project test environment constituted of microcontroller HC12 DP128, IAR Compiler and Trace 32 simulator.
- DE was involved in Analysis of Design
- DE was involved in development of Unit test procedure
- DE was involved in RTRT scrip generation and test execution
- DE was involved in submitting the Validation summary report

Unit testing of Huron-Wheel Chair for Sunrise Medicals

- DE was involved in Analysis of Design
- DE was involved in development of Unit test procedure
- DE was involved in executing test cases using Trace 32 debugger on the target
- DE was involved in providing the Validation summary report