Cumulative Coverage of the Simulink-based MIL Unit Testing for Application Layer of Automotive

Dmytro Humennyi[a], Valerii Kozlovskyi[b], Tatyana Nimchenko[b], and Yanina Shestak[c]

[a] Embedded division, N-iX corporation, Lviv, Ukraine
[b] National Aviation University, Kyiv, Ukraine
[c] Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

Abstract

The publication addresses the topic of assessing the completeness of the test coverage of software code to ensure compliance with the criteria of the test process, which are given in ISO 26262 in Part 6. And are used for the automotive industry as an application level. The common case of cumulative coating collection, which is observed at the component levels, is taken into account. Attention is paid to the case when the component consists of modules that have been implemented by MATLAB Simulink. The publication contains some approaches to estimating the completeness of model coverage by collecting cumulative coverage.

Keywords

Automotive, coverage estimation, model-based test, application level, functional safety.

1. Introduction

The modern car is a computer on wheels. The number of innovations in software has long exceeded the number of innovations in all industries related to automotive engineering, including mechanics, design, aerodynamics, optics, materials science, etc.

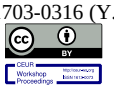
Software that is developed and runs on on-board vehicle microcontroller systems is the core value of a modern automotive company. Due to the great competitiveness in the automotive industry, the amount of software being written for this industry is scaling in a progression that is rather geometric. At a time when the market of information technology industry professionals is unable to meet such industry demands. Accordingly, software is often written by engineers without the appropriate qualifications and education.

All this leads to a large amount of low-quality software code that has both redundancy problems and critical implementation errors. A good, classic solution is to follow guidelines for code styling and verification methods. The MISRA document series is thus a good example of stylistic guidelines.

The ISO 26262, ISO 29119, ISO 9001 group of standards are those that provide comprehensive guidelines for quality assurance for both Functional Safety and Quality Assurance portions of the code. Being able to write code that could comply with MISRA C, ISO 26262, ISO 29119, ISO 9001 is costly. Because it requires significant human resources with low competence and time - and all that under competitive conditions.

2. Related Work

The cost of writing code was estimated by Yoon, Sang-Ho [1] and Drozdenko, Benjamin [2]. Important for the analysis are the works of Bispo, João, Luís Reis and João MP Cardoso [3] and Parkhomey I., Klaponin Y., Tkach M. [4], devoted to the estimation of the costs of the introduced tests

Emerging Technology Trends on the Smart Industry and the Internet of Things, January 19-20, 2022, Kyiv, Ukraine
EMAIL: apollo.d.g@gmail.com (D. Humennyi); vvkzeos@gmail.com (V. Kozlovskyi); fiona54@ukr.net (T. Nimchenko); lucenko.y@ukr.net (Y. Shestak)
ORCID: 0000-0001-6736-0543 (D. Humennyi); 0000-0002-8301-5501 (V. Kozlovskyi); 0000-0001-8196-5493 (T. Nimchenko); 0000-0002-1703-0316 (Y. Shestak)


for code coverage test. Problems and solutions describing alternative development methods, namely Model Based Design, belong to Pelliccione, Patrizio [5]. A complete review of testing methods for the automotive industry is done in [6].

3. Methods

But a good solution is to abandon the classical and certainly effective method of development, namely - writing program code, and adopt an alternative method - Model-Based-Design and its "right-hand side" of the V-scheme of development—Model-Based-Test.

According to ISO 26262, the part of the software component (this also applies to the hardware implementation), the failure of which threatens the life and health of both the vehicle user and the environment, are classified on a scale of ASIL levels ranging from A to D, where D describes cases of the highest threat.

Along with other criteria indicating the safety of software implementation, there are recommendations for evaluating the completeness of test coverage by test scenarios, followed by an evaluation of each of these scenarios according to the expected behavior.

It is important to note those levels of software implementation that require coverage assessment. Thus, according to ISO 26262 - 6 (9.4.5) - MCDC coverage is recommended for ASIL A, B, C and strongly recommended for ASIL D.

From a Model-Based-Design perspective, the "Unit" abstraction must be implemented within a single file and must occupy a subsystem allocated to it.

The Simulink environment with Simulink Coverage, Simulink Report and Simulink Test allows you to get an estimate of the test coverage for a single module.

For the most part, according to recommendations from Mathworks, the Test Harness is based on the Simulink Test tool. Test Harness includes such basic blocks as Test Assestments, Test Sequences, and Unit Under Test, which perform the roles of test sequence verification, test sequence generator, and the object under test, respectively. As shown in Fig. 1.

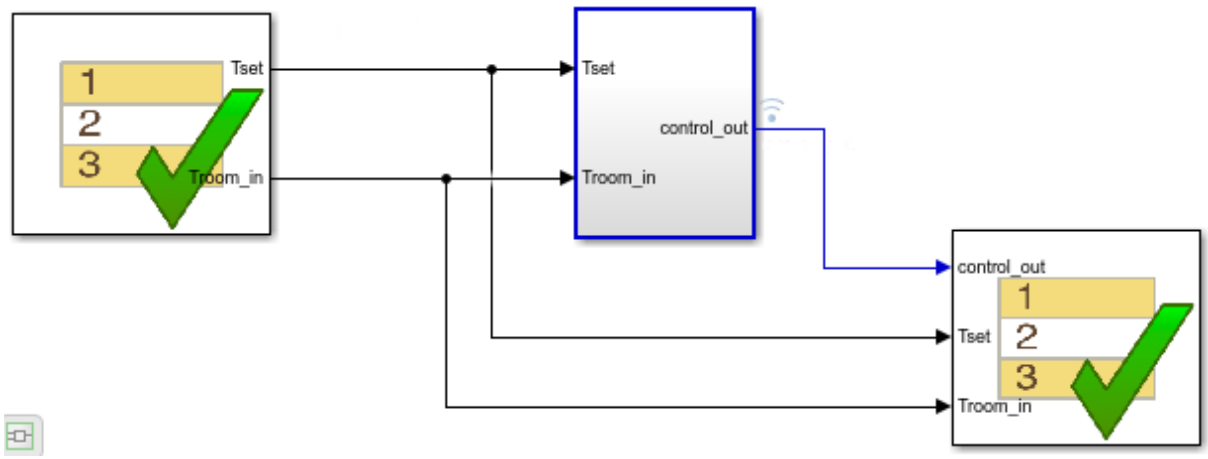


Figure 1: Typical Test Harness composition in Simulink. Test Sequences block on the left, Unit Under Test in the center, Test Assessments on the right

Thus, one module is implemented in a subsystem, and both compliance and the percentage of test coverage can be evaluated for it by embedded methods.

If we consider the developed software from a higher degree of abstraction, then, by integrating with each other, software modules form software components.

Modules within components have gained additional entities - interfaces. These interfaces have properties, in particular—data type. Coverage of two or more modules by tests implies that intermediate interfaces are also covered by tests, thus reducing the risk of errors at the interface level.

An important task is to develop methods for evaluating the completeness of component coverage after the first level of integration.

Mathworks provides several tools for assessing coverage at the integration level. One of them is based on the Design Verifier Toolset and consists in looking for Dead-Logic in the implementation. Design Verifier does not actually evaluate coverage. It points to parts of the implementation that are not involved in any of the code or model scenarios. Thus, this tool does not comply with recommendation ISO 26262-6 (9.4.5).

4. Results

Another way, supported by Mathworks, is to use Simulink Coverage Tools, available with MATLAB, certified for ISO 26262 and ISO 29119 and gives access to coverage metrics for each of the component subsystems.

For example, Mathworks recommends using the following code:

```
cvdo = cvsim (mdl)
cvsave (<path_to_file>, cvdo)
cvhtml (<path_to_file>, cvdo)
```

with the .CVT file labeling the section describing the coverage. The section looks as follows:

```
CV_Database {
modelcov {
 id                      1
   name                  "TestHarness_NAME"
   refModelcovIds        [2 1]
   topModelcovId         [1]
  }
    data {
     decision "100 M96x1[1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0"
     " 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 63 0]"
     condition "100 M244x1[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0"
     " 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0"
     " 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0"
     " 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0"
     " 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]"
   }
```

The Data section describes the dimensionality of the module under the test and is important for collecting cumulative coverage with other blocks.

The application of this method becomes possible by modifying the proposed code as follows:

```
function res = unitTesting(...)
.../// NDA
warning ('off','all');
cvtFileName = [PathTo.TestHarness 'FILE_NAME.cvt'];
disp ('ERROR MSG');
 disp (['ERROR MSG additional' NameReq '.slx']);
%% UNIT TEST
cd (PathTo.TestHarness)
switch ToolchainSelector
   case (1) %NDA
      ...
      /// NDA
```

```
            // alternative method of the testing
            end
        otherwise % Custom Tool
            disp (' /// NDA');
            disp ([' In the TestSuite exists ' num2str(NDA) ' TestCases']);
            warning ('off','all');
            SimOut = sim (['PATH TO FILE ' NameReq '.slx'],'ReturnWorkspaceOutputs','on');
            warning ('on','all');
            res = 'PASS'; % Default - The test is Done
                for TestCaseNum = 1:HowManyTestCasesAreHere % Repeat for eatch TestCase
              % Find Fail testcase. ('-1' - The Test is FAIL;
              %                     '1' - The Test is PASS.
              if (min (eval (['NDA.REQ_' NameReq '_result.TestCase' num2str(TestCaseNum) '.Data']))
== -1)
                 res = 'FAIL';
                 TestStatus(TestCaseNum) = -1;
                 TestCaseStatus = 'FAIL';
              else
                 TestStatus(TestCaseNum) = 1;
                 TestCaseStatus = 'PASS';
              end
              disp (['   Unit Test for TestCase #' num2str(TestCaseNum) ' is ' TestCaseStatus]);
            end
    end

     disp ('- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -');
     disp ('    <strong>UNIT TEST RESULT:</strong>');
     disp ([' for REQ-' num2str(NameReq)]);
     disp ([' ' TaskTicket]);
     disp ([' ' res]);
     TestSuiteStatus = [' REQ-' num2str(NameReq) '(' TaskTicket ') is ' res];
     disp ('- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -');
     res = TestSuiteStatus;

    %% COVERAGE CALCULATION
    % precondition
       coverage.complexity = -1;
       coverage.condition = -1;
       coverage.decision = -1;
       coverage.execution = -1;
       coverage.mcdc = -1;
       warning ('off','all');

    % if option is activated then ...
    if (TestMode.coverage.condition == true || ...
          TestMode.coverage.decision == true || ...
          TestMode.coverage.mcdc == true)
       disp ('    <strong>Coverage and Complexity:</strong>');
       mdl = ['NDA_' NameReq];
       mdl_subsys = [PathTo.DesignModel 'NDA_' NameUUT];
       open_system(mdl);
       open_system(mdl_subsys);

       [coverage.tests, coverage.data] = cvload (cvtFileName);
       if TestMode.coverage.decision == true
```

```
        cov = decisioninfo (coverage.data{1}, ['DesignModel_' NameUUT]);

        coverage.decision = 100 * (cov(1)/cov(2));
        disp (['      Decision  : ' num2str(coverage.decision)]);
    else
        coverage.decision = -1;
        disp ('      Decision  : is N/A');
    end
    if (TestMode.coverage.condition == true);
        cov = conditioninfo (coverage.data{1}, ['DesignModel_' NameUUT]);
        coverage.condition = 100 * (cov(1)/cov(2));
        disp (['      Condition : ' num2str(coverage.condition)]);
    else
        coverage.condition = -1;
        disp ('      Condition : is N/A');
    end
    if (TestMode.coverage.mcdc == true)
        cov = mcdcinfo (coverage.data{1}, ['DesignModel_' NameUUT]);
        coverage.mcdc = 100 * (cov(1)/cov(2));
        disp (['      MCDC      : ' num2str(coverage.mcdc)]);
    else
        coverage.mcdc = -1;
        disp ('      MCDC      : is NA');
    end
  end
  warning ('on','all');
  % SV Save.
  cvdo = cvsim (mdl);
  cvsave (([PathTo.TestHarness 'Coverage_REQ_' NameReq]), cvdo);

  res = [res '; COVERAGE: Condition: ' num2str(coverage.condition) ...
                '; Decision: ' num2str(coverage.decision) ...
                '; MCDC: ' num2str(coverage.mcdc)];
   %% REPORTING
  /// NDA
  end
  warning ('on','all');
 end
```

5. Conclusion

During operation and augmentation, the method described by Mathworks showed critical dependencies, specifically:

1. in a component, all interfaces must be shared by blocks;
2. if a model is connected as a linked model, then the linked model must have the same dimensionality as the model to which it is authenticated.

If these conditions are not met, the method is invalid.

Promising for further consideration is a method that is based on the creation of Simulink libraries, which would contain a component entirely. In this case, all the blocks included in the components would have common interfaces with each other, so they would be suitable for collecting cumulative coverage. However, this approach to estimating coverage is redundant due to the need to build and maintain a library component. It is also not the typical, recommended by Mathworks, solution, hence causing additional difficulties in the qualification and certification phase of the software product, methods and tools that were used to build the vehicle.

6. Acknowledgements

I thank my colleagues, namely Oleksandr Denysenko and Oleksii Chkalov, for their support in the process of working on this topic, without whom the work would have dragged on for a silly time.

Last but not least, I thank Igor Kozakevych and Artem Kharchenko for organizational support in producing the publication and Bogdan Poklad for editing and translation.

7. References

[1] Yoon, Sang-Ho. "Automatic code generation." (2013). // https://kr.mathworks.com/content/dam/mathworks/mathworks-dot-com/solutions/automotive/files/kr-expo-2013/Track_2_4.pdf

[2] Drozdenko, Benjamin, et al. "Implementing a matlab-based self-configurable software defined radio transceiver." International Conference on Cognitive Radio Oriented Wireless Networks. Springer, Cham, 2015. // https://genesys-lab.org/papers/Crown-Com-SDR.pdf

[3] Bispo, João, Luís Reis, and João MP Cardoso. "Techniques for efficient MATLAB-to-C compilation." Proceedings of the 2nd ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming. 2015. // https://repositorio.inesctec.pt/server/api/core/bitstreams/275e8d49-c5c3-486e-adb6-4eb3d0460294/content

[4] Klaponin Y. Structural model of robot-manipulator for capture of no-cooperation client spacecraft / Klaponin Y., Humennyi D., Parkhomey I., Rudnitska O // CEUR Workshop Proceedings (.http://ceur.ws.org) Vol-2067 urn:nbn:de:0074-2067-8-0 P. 151 – 157. – ISSN 1613-0073 (http://ceurws.org)

[5] Pelliccione, Patrizio, et al. "Automotive architecture framework: The experience of volvo cars." *Journal of systems architecture* 77 (2017): 83-100. // https://www.researchgate.net/profile/Patrizio-Pelliccione/publication/314033799_Automotive_Architecture_Framework_The_Experience_of_Volvo_Cars/links/5a1604020f7e9bc6481c7eab/Automotive-Architecture-Framework-The-Experience-of-Volvo-Cars.pdf

[6] Anton Albinsson, Fredrik Bruzelius, at al. Validation of vehicle-based tyre testing methods.