Non-intrusive Software Coverage Estimation for Safety-Critical System Certification

Martin Heininger
CEO
HEICON – Global Engineering GmbH
Schwendi, Germany
martin.heininger@heicon-ulm.de

Rainer Makowitz
CMO
Accemic Technologies GmbH
Kiefersfelden, Germany

***Abstract*— Proof of functional safety requires the collection of structural coverage information to confirm that the structural coverage is appropriate for the required safety level.**

A new observation methodology based on processor traces provides the key means to gain non-intrusive insight into the execution of production code for multi-core SoCs. Trace data analysis must be automated to cope with the enormous bandwidth of trace data streams.

This new technique supports certification of test coverage and enables automated detection of emerging timing constraints. Most importantly, it allows structural tests to be executed on the basis of the production code without the need for software instrumentation.

The paper provides an overview of the applicable functional safety standards, explains the advantages of executing structural tests even at higher test levels, and gives practical hints for hardware and software architecture considerations for providing best observability for executing structural tests.

Non-intrusive, Functional Safety, structural Software Coverage, Data- and Control flow, Requirement Coverage, Certification

I. Introduction

Assessment and Certification of Safety-Critical Systems are based on functional safety standards, like ISO 26262 [1] (Automotive), EN 50128 [2] (Railway), IEC 61508 [3] (Industrial Automation), ISO 25119 [4] (Agriculture and forestry), DO 178C (Aerospace) and others. Each of the standards defines 4 to 5 levels of criticality.

The goal of all standards is, to minimize the failures in safety-critical systems. There are two types of failures which have to be distinguished:

- Failures due to random hardware faults
- Failures due to systematic hardware and software faults

For random hardware faults the standards define several constructive measures which are able to detect this failure type and move the system in a safe state.

For systematic faults the functional safety standards define techniques and measures, which have to be applied during the development, to minimize these failures in the field to an acceptable minimum.

For critical systems the functional safety standards mandate a very high quality of the testing strategy which is traditionally assessed by structural code coverage tests [5]. Key coverage metric targets are 100% structural source code coverage and the completeness of a data- and control flow analysis. Both measures have for many years been required by the safety standards. However, there have been several drawbacks with the implementation of each of the measures, mainly due to technological limitations.

Our new non-intrusive system observation technology [7] provides for the first time new opportunities for the usage of both measures.

In the following sections we address first the structural code coverage measure. In the following section we discuss data- and control flow analysis and how it can be improved using our new methodology.

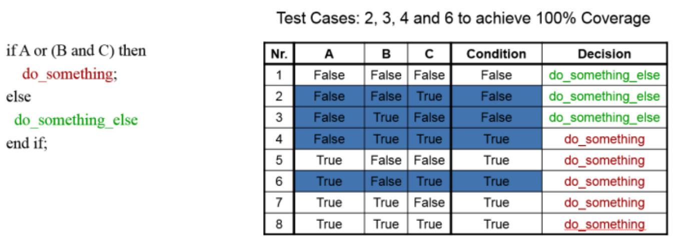
II. Structural Source Code Coverage

A. *Introduction of structural source code coverage*

The structural source code coverage provides a figure about the coverage of the source code structure by executed tests. There are different kind of coverage measures possible [6]. The coverage of the executable source code statements is very often used. This means that each source code statement has to be executed to achieve 100% coverage.

The well-known MC/DC Coverage (Modified Condition/Decision) proves that each variable within a condition has an effect on the result of the decision. The minimum number of test cases required is the number of variables in a condition +1.

The following figure 1 illustrates test cases to achieve MC/DC coverage:



```
if A or (B and C) then
   do_something;
else
  do_something_else
end if;
```

Test Cases: 2, 3, 4 and 6 to achieve 100% Coverage

Nr.	A	B	C	Condition	Decision
1	False	False	False	False	do_something_else
2	False	False	True	False	do_something_else
3	False	True	False	False	do_something_else
4	False	True	True	True	do_something
5	True	False	False	True	do_something
6	True	False	True	True	do_something
7	True	True	False	True	do_something
8	True	True	True	True	do_something

Fig. 1. MC/DC Coverage example

In test case 3 and 4 variables A, and B are kept constant, only C is changing. As the decision is changed from do_something_else to do_something, these two test cases prove that C does have an impact on the decision.

In test cases 2 and 6 B and C are constant and A is changing. These test cases prove that A does have an impact on the result.

In test cases 2 and 4 A and C are constant and B is changing. That's the prove that B does have an impact on the result.

B. *Pros and Cons of structural coverage measurement in practical work*

Across the relevant industries, it has become clear in recent years that White-box testing (a method of software testing that tests internal structures or workings of an application) and the simple measurement of structural coverage is not overly meaningful as an assessment of the quality of the software test strategy [6]. This is even true in cases where 100% MC/DC coverage has been reached. Here is the reason why:

Tests that only aim to measure structural coverage find hardly any errors of functional behavior. Obviously, good tests always verify the intended functionality. Most of the faults originate in the (mis-) understanding of the intended functionality (defined by the customer – not by the software structure itself). Practice shows that significantly more failures are found by Black-box testing (a method of software testing that examines the functionality of an application without peering into its internal structures or workings) where the tests are based on requirements [6].

Requirements-based testing, however, has its own weakness: the definition of completeness in requirement engineering. Completeness metrics are very difficult to achieve for requirements. It is generally accepted that it is practically impossible to create 100% complete requirements [6]. The same applies for the test of software intensive systems. Creating tests that achieve 100% coverage of functionality is not possible.

Test methods like equivalence class testing are introduced, in order to cover as many aspects of the system as possible and derive the test cases in a systematic way. Such methods support the tester to decide when a sufficient test coverage is achieved but up to now coverage is typically far below 100% [6].

After all, the strength of the structural source code coverage is the determination of the 100% coverage because it can be enumerated in the test program.

This situation has driven the aerospace industry to emphasize the measurement of structural test coverage [6]. The basic idea is to create requirements-based tests and observe the structural coverage of such tests.

The measurement of structural coverage represents a meaningful quality statement about the software only if requirements are considered during test creation. As it is practically impossible to create complete requirements and corresponding tests for a system, structural coverage is a very efficient corrective for this methodological weakness.

If the tests are requirements-based, then a low structural coverage can have only one of the following three root causes:

1. the requirements are not complete

2. the tests are not complete

3. the uncovered source code is not needed at all (dead code)

Tests which are measuring the structural coverage are not an independent test method in contrast to the requirement-based tests. It's the combination of both in the same test, which is most beneficial.

C. *Non-intrusive Coverage measurement strengthen the structural coverage measurement*

As explained above, pure White-box tests don't guarantee intended system behavior and there is a desire to measure the structural coverage at higher test levels in the V-diagram, as this would allow to create tests based on functional requirements and measure the structural coverage that these tests achieve. In Embedded Systems, this is typically the level at which the hardware and software are tested -- the integration test. Only for this test level meaningful functional requirements can be formulated with reasonable effort.

However, up to now there was no technology to determine the structural source code coverage with a reasonable effort at higher test levels. This is where non-intrusive observation of program and data flow is changing the picture.

Assisted by the availability of high-bandwidth trace ports in most high-performance microprocessors [ARM, INTEL, NXP] a new class of analysis tools is emerging. Developed in the context of several German and European funded research projects we have developed the CEDARtools platform that is capable of reconstructing processor instruction and data flow in real-time in live equipment. Non-intrusive measurement of structural source code coverage is one of the derived capabilities of this system and it is very likely that it will replace the classic source code instrumentation in the next years. Structural source code coverage can play a completely new role in safety-relevant software development.

So far many believe that structural coverage should and could only be determined in white-box tests. In many textbooks, the measurement of structural source code coverage is even promoted as an independent test method. With our non-intrusive measurement of structural coverage we make this methodology universally useful on all test levels.

The non-intrusive system observation technology can also increase the quality of other software techniques and measures, like the data- and control flow analysis.

III. Completing requirement by use of data- and control flow analysis results

Carrying out a data- and control flow analysis is required in almost all functional safety standards (ISO 26262-6 [1] Table 7 measures 1f/g, DO 178C [7] Table A-7 measure 8 and EN 50128 [2] Table A19 measures 3/4). Most of the functional safety software projects are challenged by performing the data- and control flow analysis. The reason is, that a proper data- and control flow analysis requires tool support, but the available tools on the market are not providing all the information required.

The goal of the data- and control flow analysis is to proof that the specified data- and control flows in architecture are correctly implemented in the source code.

A. Definition of data- and control flow

Wikipedia defines the data flow analysis [9 as follows: Data-flow analysis is a technique for gathering information about the possible set of values calculated at various points in a computer program. A program's control flow graph (CFG) is used to determine those parts of a program to which a particular value assigned to a variable might propagate.

In contrast control flow analysis is defined loosely as [10] a static-code-analysis technique for determining the control flow of a computer program; where control flow (or flow of control) is the order in which individual statements, instructions or function calls of an imperative program are executed or evaluated.

Very often there is a close coupling between data flow and control flow. An example is the error handling within a software:

Requirement SW-1: A error message TRG4 shall be send from ECU 1 to ECU 2 and the software of ECU 1 shall initiate the switch off the ECU 1 in case the battery voltage represented in a float value is above 14 Volt. In case the battery voltage is within an accepted range (9 V to 14V) the software shall continue with normal operation.

In this example the control flow depends on the content of the float value. Moreover, the example shows that there is also a relationship between the data- and control flow and the functional requirements.

B. Challenges of data- and control flow analysis

As shown above, it is very difficult in a practical application to clearly separate data flow and control flow.

In addition, software tends to have an infinite number of data- and control flows. A complete proof of the correctness of all data- and control flows is impossible.

Most importantly, dynamic verification of data- and control flows is today very challenging due to technical constraints. In the Aerospace industry unit tests and software integration tests have been used most of the time to verify important data- and control flow requirements. However, since these tests are not performed with the real hardware, no data- and control flows could be verified under stress conditions for the entire embedded system. Also, for time-critical applications, which represent the majority of embedded systems, these proofs are difficult.

C. Non-intrusive system observation

The non-intrusive system observation offers the possibility to monitor software parameters and simultaneously other external and internal events without influencing the behavior of the system. Such tests can be fully automated.

Since these tests also prove the functional software requirements, the non-intrusive system observation technology supports the completeness check of requirements. This is due to the fact that important data- and control flows are reflected in the functionality of the software. And the functionality of the system is defined in the requirements.

With the current technologies, the results of the data- and control flow analysis are decoupled from the functional requirements, as the unit test and a static analysis do not test functional requirement. The non-intrusive system observation proofs the data- and control flow with hardware/software integration tests. Equally these are the tests which also proof the functional software requirements. Tests which are only derived from data- and control flow analysis may demonstrate a gap in the requirements.

D. Activity diagrams specify data and control flows

The new possibilities, that result from non-intrusive system observation, also draw attention to a further weakness of the existing development approach. As there are almost an infinite number of data- and control flows in a software, it's not possible to define and document all flows in the architecture. Furthermore, it is not a simple task to identify and specify the most important data- and control flows.

With the new opportunities offered by non-intrusive system observation technology, this more effort will be spent to address this weakness. Both the systematic proof of the data- and control flows by test, as well as the closing of existing functional specification gaps are only successful if at least the most important data- and control flows are defined in the software architecture. A promising solution are activity diagrams that result from a systematic description of the software architecture in UML [6].

IV. Conclusion

Most of the functional safety standards define the structural source code coverage and the data- and control flow analysis as important techniques in the software development to minimize systematic software faults.

Performing a meaningful data- and control flow analysis and structural source code measurement is a challenge, as the available tools cannot fully support these two techniques.

Non-intrusive system observation enables the execution of the data- and control flow analysis and the measurement of the structural coverage within the integrated embedded system. As a consequence, non-intrusive system observation will support a combined completeness check of the functional and performance requirements.

References

[1] International Organization for Standardization Std., „ISO 26262:2018. Road vehicles – Functional safety“. 2018.

[2] European Commitee for Electrotechnical Standardization, „EN 50128:2011 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems“. 2011.

[3] British Standard, „EN 61508:2011 Functional safety of electrical/electronic/programmable electronics safety related systems“. 2011.

[4] International Organization for Standardization Std., „ISO 25119:2018 Tractors and machinery for agriculture and forestry - Safety related parts of control systems -“. 2018.

[5] „Accemic“. [Online]. http://accemic.com/. [17-Jan-2020].

[6] „Heininger, Martin: Professional experience of the author, resulting from 20+ years in consulting“. .

[7] EUROCAE, „ED-12C - Software Considerations in Airborne Systems and Equipment Certification“. 2011.

[8] „Wikipedia, data flow analysis“. [Online]. https://en.wikipedia.org/wiki/Data-flow_analysis. [17-Jan-2020].

[9] „Wikipedia, control flow“. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/Control_flow. [17-Jan-2020].