Verifysoft TECHNOLOGY

EMBRAER

Testwell CTC++ User Testimonial

As a DO-178C development team in Embraer Research and Technology department, verification activities had to be developed in order to accomplish the objectives required by the standard. Regarding the subject of this description, the objectives considered from Table A-7 of DO-178C are the following:

Objective:	Description:
A-7:3	Test coverage of high-level requirements is achieved.
A-7:4	Test coverage of low-level requirements is achieved.
A-7:5	Test coverage of software structure (modified condition/decision coverage) is achieved.
A-7:6	Test coverage of software structure (decision coverage) is achieved.
A-7:7	Test coverage of software structure (statement coverage) is achieved.
A-7:8	Test coverage of software structure (data coupling and control coupling) is achieved.

For critical software, where the Design Assurance Level is maximum (DAL A), all the 6 objectives must be accomplished, while for less critical DALs the objectives are continuously reduced.

Process

Test cases and procedures must be requirement-based, meaning that the source code is not taken into account during the activity of test creation. The first objectives that need to be satisfied are the ones related to requirement coverage, thus test cases are traced to requirements and they must cover completely the requirements considering normal cases and robustness cases. For lesser DALs, the need for covering robustness and to cover low-level requirements are reduced.

Next step is to submit the tests to an informal execution, called dry-run. During this activity the test procedures are evaluated and validated. They must attain satisfactory results, and in the case the tests fail, that should require a change in the source code or in the test case, depending on the faulty side. During dry-run, as a measure of maturity of the tests, it is a good practice to keep track of the structural coverage.

Whenever a lack of structural coverage is detected, an analysis is performed to understand why, despite the requirement coverage achieved, the structural coverage was not.

Usually the issue is on a lack of clarity of the requirement or an unintended code erroneously inserted. When the issue is on the incompleteness of the test, it must be incremented to cover the uncovered path.
That means that the source code is instrumented before the test execution in such a way that it can measure the paths exercised by the tests. There is where Testwell CTC++ comes in our process. It automates the code instrumentation as well as the evaluation of the structural coverage results in a neat report. Since Testwell CTC++ does not create the code infrastructure to run the test, a hand-made script was developed to automatically convert test vectors (tables of data) to create the so called test driver. The test driver is a code that basically defines test inputs, calls the function under test and compares the obtained output with the expected values, assigning a pass or fail verdict.
In contrast with dry-runs, the formal executions are called run-for-score. This activity, sometimes witnessed by Quality Assurance (QA) personnel, is the demonstration that the test can be completely run as described by the test procedures, in a controlled environment, achieving results that are used for certification credits.

Concerns

During the first presentation of the tool to the software community inside the company, some concerns were raised and they were investigated during the use of the tool. Some of them are reproduced below. No critical issue was identified and the use of the tool is satisfying the project needs.
The first concern was about the use of macros in the code. The result was satisfactory since they are processed as if the macros were resolved in the source code before the instrumentation.
The next one was about the ability of the tool to deal with high complexity. Cases were exercised with an excessive number of conditions inside a decision, generating a high number of MC/DC cases. The tool has a limit of 500 MC/DC conditions printed in the report, but this limitation wasn‘t considered an issue. Another test was made involving a high number of inputs and execution cycles, and the tool behavior was normal.
In the case where blocks of code don‘t need to be instrumented, Testwell CTC++ provided a solution using pragma lines of code that satisfied the needs. It was also evaluated the cases where the coverage of a particular block of code is attained by separated executions of tests. The tool operated as expected, accumulating the coverage information from one execution to the others.
These results satisfied the concerns and gave the go-ahead to use it in the R&D project that develops a DO-178C level A software.

Verifysoft TECHNOLOGY

Results

Some insights about the tool worth to be shared.
Testwell CTC++ was integrated in the continuous verification flow. That comprises a series of tools that execute an automatic verification in a periodic basis so that every change in the source code or in the tests is quickly evaluated to anticipate the detection of errors and gaps.
Testwell CTC++ installation is straightforward and the tool usage is based on a command-line interface that is simple and easy to learn.
Below there is a sample of the structural coverage report obtained during one execution in the continuous verification approach:

```
                    1 #include "include/crc32.h"
                    2
                    3 /*
                    4  * @traceto(ASWE-LLR-1130, ASWE-LLR-1132)
                    5  */
Top
         6          6 EMB_UINT32 crc32(EMB_UINT32 initial_value, const void *data, const EMB_UINT data_size)
                    7 {
                    8     EMB_UINT i, j;
                    9     register EMB_UINT32 r;
                   10     static EMB_UINT32 table[256] = {0u};
                   11     const EMB_UINT8 *data_array = data;
                   12
                   13     /* Initialize CRC table */
         1         5   14     if(table[1] == 0u)
                   15     {
       256         1   16         for(i = 0u; i < 256u; i++)
                   17         {
                   18             r = i;
      2048       256   19             for(j = 0u; j < 8u; ++j)
                   20             {
      1024      1024   21                 r = (((r & 1u) != 0u) ? 0u : ((EMB_UINT32)0xEDB88320L)) ^ (r >> 1);
                   22             }
                   23             table[i] = r ^ (EMB_UINT32)0xFF000000L;
                   24         }
                   25     }
                   26
                   27     /* Calculate CRC */
                   28     r = initial_value;
         6         0   29     if(data_array != EMB_NULL)
                   30     {
        29         6   31         for(i = 0u; i < data_size; ++i)
                   32         {
                   33             r = table[((EMB_UINT8)r) ^ data_array[i]] ^ (r >> 8);
                   34         }
                   35     }
                   36
         6             37     return r;
```

The report results are visual and easily understandable. The color code makes easy the identification of gaps where the tests must be enhanced.

Verifysoft TECHNOLOGY

Mateus Lucas Alves Ferreira is a product development engineer of Embraer with 5 years of experience in airborne safety-critical software verification and certification against DO-178C. He graduated as a Control and Automation Engineer at Universidade Federal de Minas Gerais in 2013.

João Carlos Davison has worked for over 10 years with embedded software for safety-critical applications in the aerospace industry, being responsible for the elaboration of the software verification plan, the execution of verification activities and the interaction with Certification Authority. He graduated as an Electrical Engineer at Universidade de São Paulo in 2003.

For further questions please visit www.verifysoft.com and contact us at +49 781 127 8118-0

C Photos: Embraer and Verifysoft Technology GmbH