



Boosting coverage

How a new approach to coverage is improving the process. By Matteo Bordin.

The development of a high integrity embedded application usually requires close interaction between the design and testing processes. Starting from requirements, some team members refine the specification down to code, whilst others derive a proper test suite to verify the implementation behaves according to the specifications.

This is the V process model applied to software engineering (see figure 1). One key step in the V development process is the assessment of the quality of the testing strategy: coverage is the canonical approach in practice. The idea behind such assessment is to gain confidence in the testing process by checking that the testing suite exercises properly all meaningful constructs – such as statements and branches – present in the application. Coverage is explicitly required by several standards: for example, DO-178B requires achieving different levels of coverage – statement, decision and modified condition/decision (MC/DC) – depending on the application's criticality.

Coverage yesterday

In the past, most coverage tools were based on code instrumentation: before executing test suites, the application code was modified to embed the code that logs the application execution and then compiled. The instrumented application is generally executed on the host machine to facilitate the calculation of coverage results via an execution log.

This strategy puzzles software engineers: questions such as 'are the coverage results representative of the real embedded application?', 'did the code



instrumentation corrupt the application?' or 'are the testing results valid from a certification perspective?' are more than legitimate. The answers are always the same: no. Indeed, two testing campaigns usually need to be run: one on the host, to perform coverage evaluation; and one on the target, to verify the application. The campaigns require two dual compilation/deployment environments and, most likely, adaptation of the test suites.

In addition, and in order to trust coverage results on instrumented code, developers need also to ascertain the behavioural equivalence of the cross compiled application with the instrumented, native executable: this is usually satisfied by comparing the results of the two testing campaigns.

Another approach consists of 'single stepping' the application on the target and collecting coverage information at each step. This technique is very slow and requires a physical connection to the target hardware. Traditional approaches to coverage thus increase the already high costs of verification/certification of critical embedded applications.

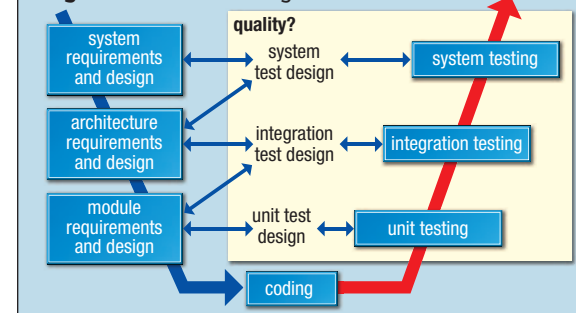
The Couverture approach

The Couverture approach improves current practice by taking advantage of a virtualised execution environment to run cross compiled applications on the host.

The Couverture toolset relies on a virtualisation layer emulating the final target hardware, including attached sensors and actuators. Virtual machines are deployed, thanks to an extended and industrialised version of QEMU (<http://en.wikipedia.org/wiki/QEMU>), an open source machine emulator and virtualiser. The raw processing power of current generation workstations, as well as advances in simulation technology, guarantees the performance required to perform simulation in a production environment.

The idea behind Couverture (see figure 2) is to run the cross compiled test suite just once and the application only on the virtualised environment: no code instrumentation is required, because the virtual machine itself produces an execution trace whilst the testing campaign executes. The Couverture toolset can then compare the execution trace with a dump of the application to evaluate the

Figure 1: The V coverage model





coverage of the test suite.

The coverage evaluation is, in the end, subject to capitalisation and consolidation. Capitalisation consists of unifying the coverage results of different test suites exercising the same unit; consolidation



“Traditional approaches ... increase the already high costs of verification/certification.” **Matteo Bordin, Adacore**

assures the union of all unit tests covers the application as a whole.

Virtualisation technology alleviates the verification and the coverage evaluation processes. It permits early verification of the embedded application itself – even before hardware availability – and of the effectiveness of the testing strategy; and it does not require slow physical connections to the target board. Coverage via virtualisation is also a non intrusive methodology as it doesn't require instrumentation: coverage is evaluated on the same executable code that will be deployed on the final hardware, making unnecessary the dual compilation/deployment environment and traditional testing campaign approach.

In addition to virtualisation, the other distinctive feature of Couverture is its focus on object instructions: the elaboration of execution trace returns coverage results in terms of object instructions.

Focusing on object instructions has two major advantages. First of all, it is independent of programming language:

the Couverture toolset can evaluate the coverage of applications written in Ada, C, C++ and any other compiled language (or in any combination). It can also provide evidence that all executable code the compiler generates – in particular the portion not directly traceable to source code – is exercised and behaves correctly according to the requirements.

This feature satisfies another key requirement of several standards for high integrity software: for example, certification at the highest criticality level for DO-178B requires evidence that either the object code is completely traceable up to source level, or that all compiler added object code is correct and behaves according to requirements.

Regardless of its focus on object code,

and in compliance with standards such as DO-178B, final coverage results produced by the Couverture toolset are traced back (thanks to debug information) to source code in terms of several possible source level metrics – for example, statement coverage, decision coverage or MC/DC.

Beyond its technical merits, the Couverture project exemplifies a new paradigm – Open Qualification Material.

The Couverture project is funded by the Paris Region via the System@tic framework; Adacore is the leading partner for the project, which also involves OpenWide and French universities Telecom ParisTech and LIP6.

To justify the reliance on a tool in the development of a certifiable application, certification authorities require evidence that the tool itself has been developed following certain criteria: qualification material provides such evidence. Usually, qualification material shipping with COTS products has a closed license – it cannot be adapted to user specific needs nor to specific requests from certification authorities. Normally, the license is for a single project only.

The license of qualification material for Couverture will, on the contrary, be open: the end user will be able to modify it, adapt it to specific requirements and reuse it. Tool providers' revenue will come from consulting on the usage, extension and adaptation of the toolset and of the qualification material: in the long term, it will be cheaper to pay for qualified services than to employ and train engineers in key, but narrow, technology niches.

This is the open source philosophy brought to qualification material – and a radical shift for the high integrity software community. ■

Author profile

Matteo Bordin is a software engineer with Adacore (www.adacore.com).

Figure 2: The Couverture approach

