Making tests executable

Angelo Gargantini

Testing e verifica del software

AA 2015-16

Concretization

- how to take a suite of abstract tests, generated from an abstract model, and make them executable on the real SUT?
- This concretization phase is an important part of the model-based testing process, and it can take a significant amount of effort.
  - molte volte fatto a mano - vedremo dei tool in laboratorio
- sometime, test execution require some human activities (like using the system) and this makes necessary the human intervention.

Section 1

Principles of test adaptation

Common abstractions

- Model only one aspect of the SUT, not all its behavior.
- Omit inputs and outputs that are not relevant to the test goals.
- Take a simplified view of complex data values, such as enumerating a few typical values.
- Assume that the SUT has already been initialized to match a particular testing scenario.
- Define a single model operation that corresponds to a sequence of SUT operations, or to just one part of an SUT operation.

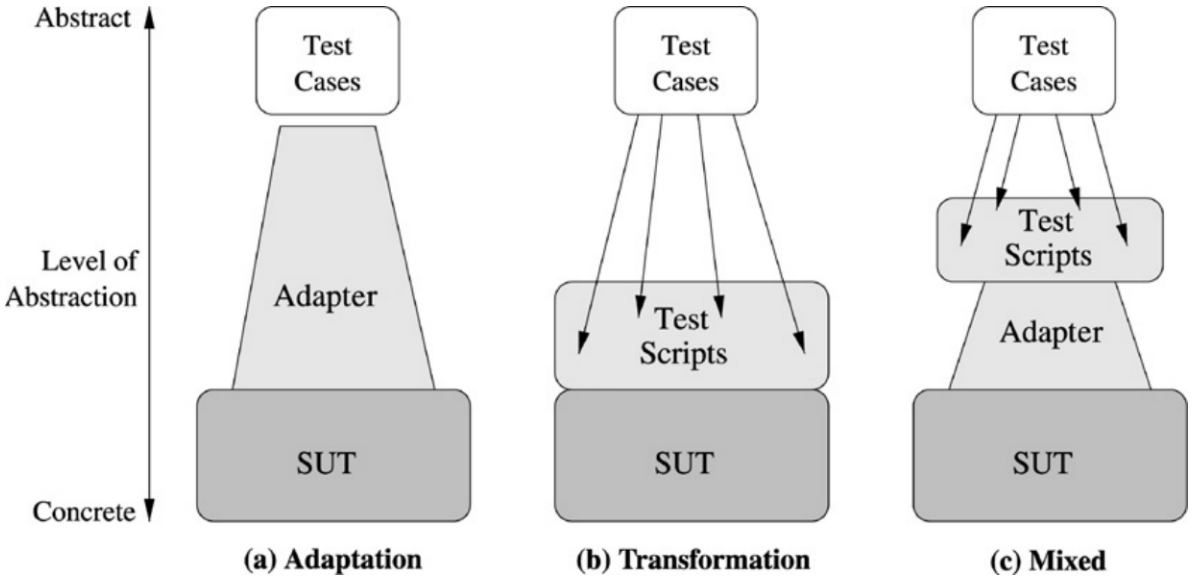
Online vs offline testing

Online testing is where tests are executed as they are generated, so the model-based testing tool is tightly coupled to the SUT.

Offline testing decouples the generation and execution phases, so the test execution can be completely independent of the model-based test generation process. This book contains several examples of each of these. Online testing is

bridging the gap between abstract and concrete

- Three approaches to bridging the semantic gap between abstract tests and the concrete SUT



Bridging the gap

- ADAPTATION: manually write some adapter code that bridges the gap. This is essentially a wrapper around the SUT that provides a more abstract view of the SUT to match the abstraction level of the model.
- TRANSFORMATION: The transformation approach, (b), is to transform the abstract tests into concrete test scripts.
- MIX: use an intermediate notation

Adaptation

Setup: Set up the SUT so that it is ready for testing. This involves configuring and initializing the SUT so that it reflects the test scenario assumed by the model.

Concretization: Translate each model-level abstract operation call and its abstract input values into one or more concrete SUT calls with and its the appropriate input values.

Abstraction: Obtain the SUT results from those concrete calls, and translate them back into abstract values, and then pass them back to the model for comparison with the expected results in order to produce the test verdict.

Teardown: Shut down the SUT at the end of each test sequence or at the end of each batch of tests.

The Transformation Approach

The transformation approach involves transforming each abstract test into an executable test script:

- A standard programming language, such as Java or C
- A scripting language, such as TCL, JavaScript, or VBScript
- A standard test notation, such as TTCN-3 [WDT+ 05]
- A proprietary test notation, such as the TSL (Test Script Language) of Mercury WinRunner or some company-specific test notation

The Transformation Approach

- Some **concrete setup** and **teardown** code will be needed.
- The **template** for each operation may be quite complex because there is not necessarily a one-to-one mapping between the signature of the abstract operation and the SUT operations.
- The model uses abstract **constants** and **values**—these must be translated into the **concrete** values used by the SUT.
- When testing nondeterministic SUTs, an abstract test may have a **tree** structure rather than being a simple sequence. This requires the transformation engine to be more sophisticated to handle such structures. It must generate an executable test script that includes **conditional statements** to check the SUT outputs and then take the appropriate branch through the tree to the next part of the test.
- **Traceability** between the concrete test scripts and the abstract tests must be maintained

The Transformation Approach

- The structure of the transformed test suite is almost as important as the code within each test script.
- In addition to generating the executable code of the test scripts, it may be desirable to generate a **test plan**, which describes, in English, the structure of the test suite, the rationale for each test in the suite, the settings used to generate the tests, who generated the tests, when they were generated, and so on.

The transformation approach can produce test scripts that fit smoothly into your existing test management practices, with similar language, structure, and naming conventions as manually written test scripts.

Which Approach Is Better?

- For **online testing**, it is almost always better to use the adaptation approach because online testing requires a tightly integrated, two-way connection between the model-based testing tool and the SUT.
- For **offline testing**, we can choose between the adaptation approach or the transformation approach, or use a mixture of the two.
  - If we use the transformation approach, then we obtain a suite of executable test scripts that can be executed directly on the SUT.
  - If we use the adaptation approach, then our suite of abstract tests effectively becomes executable because the adapter acts as an interpreter, mapping each abstract call into SUT operations and translating the SUT results back to the abstract level.
  - With the mixed approach, we transform the abstract tests into executable test scripts that call an adapter layer to handle the low-level details of SUT interaction.

Simple rule

For online testing, use the adaptation approach. For offline testing, the transformation approach has some advantages (less disruption), and it is often useful to combine it with the adaptation approach.