Actual Test Coverage for Embedded Systems

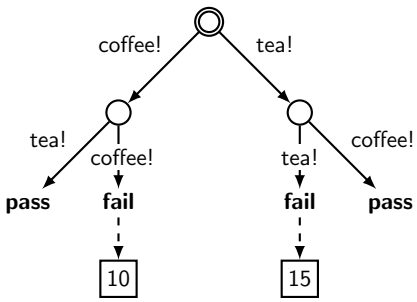
Mark Timmer, University of Twente

Testing embedded systems is inherently incomplete; no test suite will ever be able to test all possible usage scenarios. Therefore, in the past decades many coverage measures have been developed. These measures denote the portion of a system that is tested, that way providing a quality criterion for test suites.

Formulating coverage criteria is not an easy task. The measures provided in the literature are consequently almost all very trivial and syntax-dependent. Well-known examples are statement and path coverage in white-box testing, and state and transition coverage in black-box testing. The complexity of designing coverage measures for embedded systems is contained in the highly dynamic behaviour of such systems, which is state-dependent and subject to many interleavings.

Recently a coverage measure that is suitable for embedded systems has been introduced by Brandán Briones, Brinksma and Stoelinga: semantic coverage. It measures how many faults might be uncovered by a given test suite. A possible disadvantage of semantic coverage is that it focuses on which faults can *potentially* be detected. In practice, however, a single execution or even several executions of a test suite will almost never detect all faults that could potentially be detected, due to non-determinism of the system outputs.

For example, if a system can provide either one of two correct responses, only one of these occurs during a single execution. Therefore, potential faults in the traces starting with the other output will not be detectable. In the test case shown here graphically, we can potentially detect both the failure of providing two coffees and the failure of providing two teas, but in an actual execution only one of these might be detected.



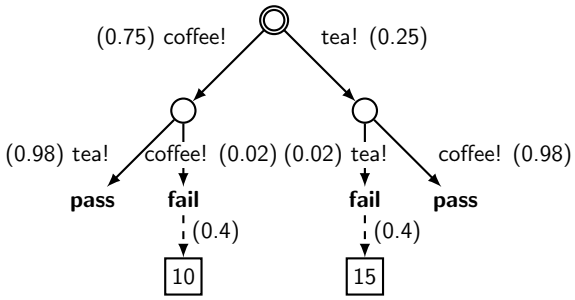
In this talk we introduce a framework on *actual test coverage*. This measure denotes the number of faults *actually* shown present or absent. Our framework contains a method to *evaluate* the actual coverage of a given set of test suite executions after testing has taken place, providing a means to express the quality of a testing process. It also contains a method to *predict* the actual coverage a certain number of executions will yield, providing a means to select the best test suite. Both the evaluation afterwards and the prediction in advance are quite efficient, making it feasible to implement the theory in a tool and use it in a practical context.

Our methods are behavioural, in the sense that actual coverage is invariant to syntactic changes that do not affect system behaviour. That is, we only consider the inputs that might be provided by the user and the outputs that might be provided by the system, and are not concerned in any way with the manner in which this is implemented. Coverage is then defined based on the number of traces that have been shown to be implemented either correctly or incorrectly. When testing did not show the absence of a fault but did increase

our confidence in its absence, we consider it partly covered. Faults can also be given a weight, incorporating their severity.

The framework is based on a probabilistic execution model, describing the probabilistic output behaviour of a system. Using this information, we can estimate how many faults on average will be detectable during a testing process. Moreover, we use estimations of the probability with which faults occur in case they are present, enabling us to calculate the probability of their absence given a certain execution.

As an example, the test case shown earlier has now been updated to also include probabilities. We assumed that the system starts three out of four times by providing a coffee, so, abstracting from the more difficult details, we can state that the probability to actually cover the failure of two coffees is three times as large as the probability to cover the failure of two teas.



In conclusion, our framework is based on a probabilistic execution model, it evaluates the effect of a test suite after testing has taken place, and it predicts the effect of a given number of test suite executions. It is therefore useful for both test evaluation and test selection.

We will illustrate our methods using a small example of a chemical dispenser.

Mark Timmer studied Computer Science at the University of Twente, where he graduated on test coverage in June 2008. He is currently a PhD Student at the Formal Methods and Tools group of the same university, working on Stochastics in Process Algebra. The work to be presented at the 14th Dutch Testing Day was performed in the context of his Master's Thesis.