Search-Enhanced Testing
an approach for automating defect-testing with harvested oracles

Step 1: Create a characterising test

Functionality description. The goal of a characterising test is to specify the functionality of potential oracles so that the test-driven search [1] can identify components with the desired functionality but at the same time minimizes the number of components with undesired functionality.

Finding oracles. The quality of the characterising test is essential to finding "good" oracles that semantically match the CUT.

Algebraic Specification. Characterising tests are created in a format similar to algebraic specifications.

Minimalistic. By the fact, that algebraic specifications do not require determined input / return value tuples, but make use of the universal quantifier, they can be rather minimalistic.

Step 2: Test-Driven Search

Finding oracles. Our code-search engine merobase.com performs a test-driven search to find multiple different implementations of the functionality defined by the characterising test.

Precise. Test-Driven Search has turned out to be the most precise kind of search technique available [1] and is practically usable.

Insensitive to names. Any component with the required method profiles is included in the list of candidates to be tested even if the names of its operations and parameter order do not match those expected by the characterising test.

Step 3: Multi-Version Testing

Testing multiple versions. The idea of multi-version testing (MVT) is built upon foundations developed in back-to-back testing [2] and n-version programming [3], where two or more distinct versions of a program "vote" on the correct outcome of an invocation.

MVT approach. For multi-version testing, the CUT and the discovered oracles are invoked by a MVT broadcaster, which logs their invocation responses and marks occurring discrepancies between them.

One class to test them all... The MVT broadcaster is a facade class, which enables any class that would normally invoke operations on the CUT to do so by mimicking the CUT’s provided interface, but internally distributing the invocations to all the oracles and the CUT.

En masse invocation. The MVT broadcaster, by mimicking the CUT’s interface, makes it possible to perform large numbers of randomly generated invocations on the CUT and the oracles.

Step 4: Discrepancy Analysis

Invocation profiles. The results of the invocations performed during step three are automatically assigned to an equivalence class (e.g. the CUT and all oracles agree or there is a disagreement between all oracles and the CUT).

Discrepancy table. Based on the created invocation profiles, the SET environment creates a human readable discrepancy table that lists controversial invocation results.

Identify interesting test cases. The discrepancy table enables a human tester to identify important test cases for the CUT by inspecting only the "interesting" ones and thus helps to write a comprehensive test suite more efficiently.

References


Read more: Colin Atkinson, Oliver Hummel and Werner Janjic, Search-Enhanced Testing (NIER Track), ICSE 2011, Honolulu, USA.
Werner Janjic et al., Discrepancy Discovery in Search-Enhanced Testing, SUITE 2011, Honolulu, USA.