Test adequacy

Test adequacy criteria specify when a software under test is sufficiently tested. The suggested articles look into different aspects of structural test adequacy criteria, which are among the most popular adequacy criteria. The standard structural coverage metrics that virtually every test tool reports are statement coverage and branch coverage. Other widely used criteria are modified condition/decision coverage (MC/DC) and path coverage.

References:

Meng, Gay, Whalen (2018):
Ensuring the Observability of Structural Test Obligations.
In: IEEE Transactions on Software Engineering.
Available online at https://doi.org/10.1109/TSE.2018.2869146

McMinn, Wright, McCurdy, Kapfhammer (2018):
Automatic Detection and Removal of Ineffective Mutants for the Mutation Analysis of Relational Database Schemas.
In IEEE Transactions on Software Engineering.
Available online at https://doi.org/10.1109/TSE.2017.2786286

Marchetto, Scanniello, Susi (2018):
Combining Code and Requirements Coverage with Execution Cost for Test Suite Reduction.
In IEEE Transactions on Software Engineering.
Available online at https://doi.org/10.1109/TSE.2017.2777831
Test input selection

Once a test adequacy criterion is chosen, suitable test inputs need to be chosen to achieve it. We will discuss several techniques to generate test inputs and to select and prioritize test inputs from an existing set of input candidates. Surprisingly, the random selection of test input data is more difficult to beat than should be expected. The suggested articles consider systematic approaches to test input selection or generation, partially in comparative assessments with randomized input selection strategies.

References:

An interleaving approach to combinatorial testing and failure-inducing interaction identification.
In: IEEE Transactions on Software Engineering.
Available online at https://doi.org/10.1109/TSE.2018.2865772

ConTesa: Directed Test Suite Augmentation for Concurrent Software.
In: IEEE Transactions on Software Engineering.
Available online at https://doi.org/10.1109/TSE.2018.2861392

In: IEEE Transactions on Software Engineering.
Available online at https://doi.org/10.1109/TSE.2018.2852744

Rodrigues, Delamaro, Corrêa, Nunes (2018):
In: ACM Comput. Surv. 51, 2, Article 41, May 2018
Available online at https://doi.org/10.1145/3182659

Su, Wu, Miao, Pu, He, Chen, Su (2017):
A Survey on Data-Flow Testing.
In: ACM Comput. Surv. 50, 1, Article 5, March 2017
Available online at https://doi.org/10.1145/3020266
**Test oracle selection**

In order to make sense of test results, the observed software behavior during test needs to be compared against some expected behavior, which is called the "test oracle". We will discuss why oracle generation and selection is generally difficult to support by program analyses and look at a number of recent approaches targeting different aspects of the oracle problem.

References:

Metamorphic Testing: A Review of Challenges and Opportunities.
In: ACM Comput. Surv. 51, 1, Article 4, January 2018
Available online at https://doi.org/10.1145/3143561

Gay, Rayadurgam, Heimdahl (2018):
Automated Steering of Model-Based Test Oracles to Admit Real Program Behaviors.
In IEEE Transactions on Software Engineering, vol. 43, no. 6, pp. 531-555, 1 June 2017.
Available online at https://doi.org/10.1109/TSE.2016.2615311

Li, Offutt (2017):
Test Oracle Strategies for Model-Based Testing
Available online at https://doi.org/10.1109/TSE.2016.2597136
**Threats to test result validity**

The purpose of tests is to detect defects. If no defect is detected, the tester and the developers should be confident that there actually are no defects for the tested usage. Unfortunately, there are a number of procedural mistakes or unjustified assumptions that can render test results invalid. In such cases, the obtained test results are not representative for the actual behavior of the system under test in operation. The following list of papers deals with such problems.

References:

R. Natella, S. Winter, D. Cotroneo and N. Suri (2018):
Analyzing the Effects of Bugs on Software Interfaces.
In *IEEE Transactions on Software Engineering*.
Available online at [https://doi.org/10.1109/TSE.2018.2850755](https://doi.org/10.1109/TSE.2018.2850755)

Niu, Changhai, Lei, Leung, Wang (2018):
Identifying failure-causing schemas in the presence of multiple faults.
In *IEEE Transactions on Software Engineering*.
Available online at [https://doi.org/10.1109/TSE.2018.2844259](https://doi.org/10.1109/TSE.2018.2844259)