

**SWEBOK Breakdown of topics in the KA Description of Software Testing, V.
1.0 - A. Bertolino, CNR (Italy)**

SOFTWARE TESTING

Software testing consists of the dynamic verification of the behaviour of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the specified expected behaviour.

In the above definition, and in the following as well, underlined words correspond to key issues to identify the KA of software testing. In particular:

- dynamic: static analysis techniques, such as peer review and inspection, are not considered as part of this KA;
- finite: testing always implies a trade-off between limited resources and inherently unlimited test requirements; this therefore points to practical problems, of both technical (criteria for test adequacy) and managerial (estimating the effort to put in testing) nature;
- selected: test techniques differ in how they select the (finite) test set, and different selection criteria may yield very different effectiveness. The problem of identifying the most suitable selection criterion under given conditions is still under investigation.
- expected: trying the program is useless if the observed outcome cannot then be judged as acceptable or not; this is referred to as the *oracle problem*, that becomes especially difficult in statistical approaches.

Software testing is usually performed at different levels in the subsequent phases of a development process. That is to say, the object of the test can vary: a whole program, part of it (functionally or structurally related), a single module.

Testing is conducted in view of a specific purpose (test objective), which is stated more or less explicitly, and with varying degrees of precision (stating the objective in precise, quantitative terms allows for establishing control over the test process).

The test objective is strictly related with the selection of the test set, both in its consistency (*how much testing is enough for achieving the stated objective?*) and in its composition (*which test cases are the most appropriate for achieving the stated objective?*). Very often, testing is aimed at exposing failures (as many as possible), and many popular test techniques have been developed for this purpose. These techniques variously attempt to "break" the program, by trying it over identified classes of (deemed equivalent) executions: being systematic is the matter here. However, a comprehensive view of the KA of testing must include other as important objectives for testing, e.g., reliability measurement, usability evaluation, contractor acceptance, for which different approaches would be taken. The objective varies with the test object, i.e., different purposes are addressed in different phases.

Sometimes, it can happen that confusion is made between test objectives and techniques. For instance, branch coverage is a popular test technique. Achieving a specified branch coverage measure should not be considered *per se* as the objective of

testing: it is a means to improve the chances of finding failures (in the guidebook, I would like to make explicit that testing can be performed with different objectives, and that its effectiveness can only be measured against the objective. There are some recent excellent papers discussing this point, and I will refer one of these).

An alternative useful view of testing is that testing amounts at observing a sample of program executions: i.e., testing can be seen as a *scientific experiment*, in that a sample (some executions) is observed in order to draw some general conclusion over the population (the whole set of potential executions). In this view, the fundamental concepts of testing can be described in terms of the representativeness of the sample and the extent to which general conclusions can be inferred based on the test results.

Following this premise, a breakdown for the KA of software testing is proposed as follows:

- (A) "Basic concepts and definitions", which establishes the terminology and sets the boundaries of the KA wrt to related issues.
- (B) "Test Levels": establishes the object of the test; this is the topic where the traditional test phases fit in.
- (C) "Techniques": this is the place for well-known test criteria, as well as for more specific branches: surely, the most widely expanded topic in the testing literature.
- (D) "Test related measures": distinguishing between measures relative to the product, and measures relative to the performed tests.
- (E) "Testing in practice": this topic covers the management of the test process, and its constituent activities.
- (F) "Automated testing": a list of the tools currently available to support the testing. (Also, I wonder if it would be appropriate here a reference to well-known sites (mostly web sources) maintaining up-to-date catalogs of commercial products)

A. Basic Concepts and Definitions

A.1 Faults vs. Failures

- The Fault-Error-Failure chain
- Types, classification and statistics of faults

A.2 Theoretical foundations

- Definitions of testing and related terminology
- Test selection criterion/Test adequacy criterion (or stopping rule)
- Testing effectiveness/Objectives for testing
- Debug testing, or testing for defect removal
- Reliability achievement and evaluation by testing
- Test oracle
- Functional and non-functional testing
- Theoretical and practical limitations of testing
- Software testability

A.3 Setting the boundaries of the KA

- Testing vs. Static Analysis Techniques
- Testing vs. Correctness Proofs
- Testing vs. Debugging

Testing vs. Developing
Testing within SQA
Testing within CMM [Testing within other process models?]
Testing and Certification

B. Test Levels

B.1 Test phases

Unit testing
Module ("Component") testing
Integration testing
System testing
Acceptance testing
Alpha, Beta and Field Testing
Regression testing

B.2 Special purpose testing

Installation testing
Performance testing
Stress testing
Volume testing
Configuration testing
Usability testing

C. Techniques

C.1 Specification-based

Equivalence partitioning
Cause-effect graphing
Boundary-value analysis
Category-partition
Testing from formal specifications

C.2 Code-based

Reference models for code-based testing (flowgraph, call graph, dependency graph)
Control flow-based criteria
Data flow-based criteria

C.3 Fault-based

Error guessing
Mutation
Fault seeding

C.4 Usage-based

Operational testing (statistical testing)
(Musa's) SRET

C.5 Specific techniques

Object-oriented testing
Component-based testing

D. Test related measures

D.1 Evaluation of the tested program

Reliability growth models
Life test, reliability evaluation
Remaining number of defects/Fault density
Cyclomatic complexity

D.2 Evaluation of the tests performed

Coverage/thoroughness measures
Comparison and relative effectiveness of different techniques
Saturation effect

E. Testing in practice

E.1 Management

Attitudes/Egoless programming
Test process
Test documentation
Cost/effort estimation
Internal vs. independent test team
Standards/Guidelines/Interest groups

E.2 Steps within a test process

Planning
Test case generation
Execution
Test output evaluation
Trouble reporting/Test log
Defect Tracking

F. Automated Testing

F.1 General overview of test environments

What can be automated, and what cannot
Surveys/Reference sources for commercial tools

F.2 Tools

Capture/Replay
Test harness (drivers, stubs)
Test generators
Oracle/file comparators
Coverage analyzer/Instrumentor
Tracers
Reliability evaluation tools