2. Testing and verification process

Angelo Gargantini

Testing e verifica del software

AA 2015-16

Section 1

What is testing

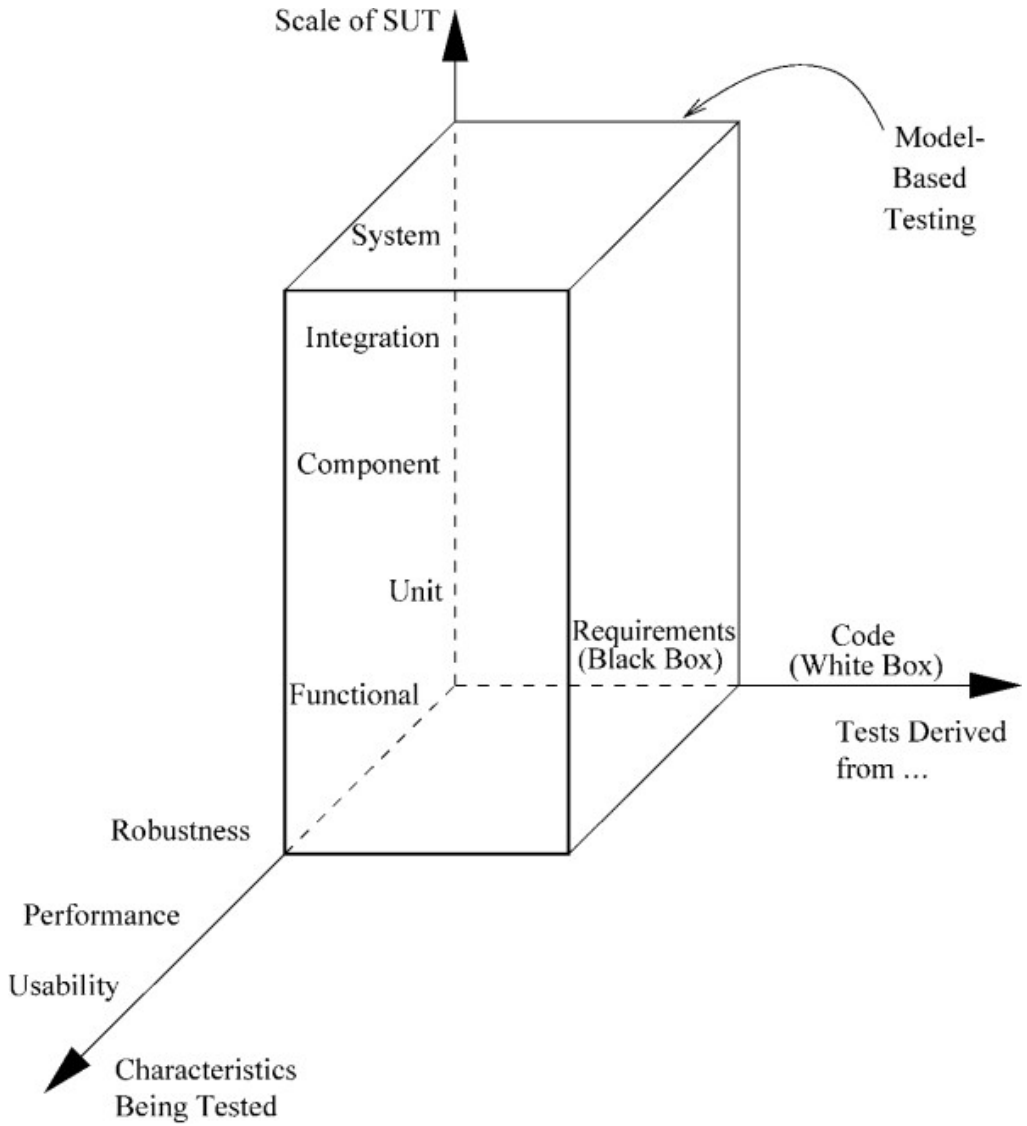
What is testing

Definition by IEE SE Body of knowledge

Testing is an activity performed for evaluating product quality, and for improving it, by identifying defects and problems. Software testing consists of the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior.

Dynamic	it requires the sw to be executed
Finite	only a subset of possible inputs
Selected	selected according to some criteria
Expected	there must be a way to check sw correcteness

What is testing



Section 2

Classic testing processes (program-based)

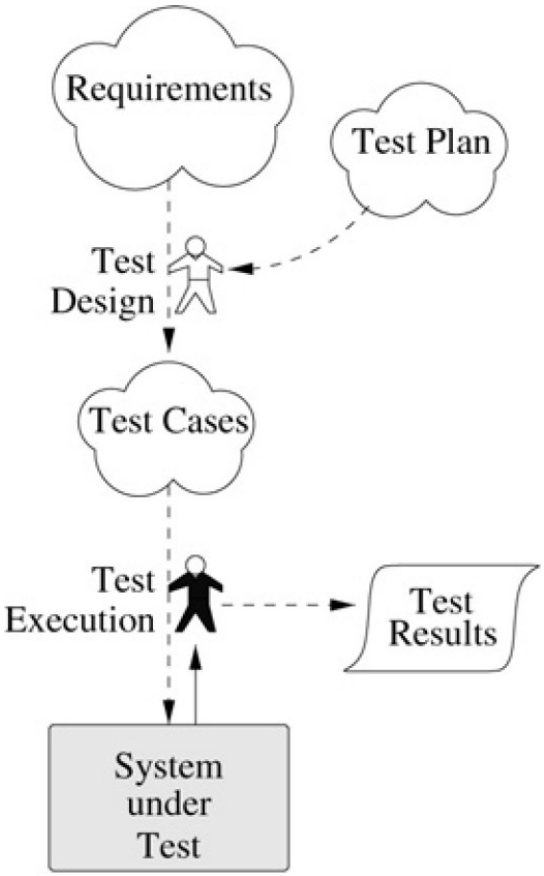
Key issues

Designing the test cases: The test cases have to be designed starting from the system requirements and taking into consideration the high-level test objectives and policies. Each test case is defined by a test context, a scenario, and some pass/fail criteria.

Executing the tests and analyzing the results: The test cases have to be executed on the system under test (SUT). The test results are then analyzed to determine the cause of each test execution failure.

Verifying how the tests cover the requirements: To manage the quality of the testing process (and therefore the quality of the product), one must measure the coverage of the requirements by the test suite. This is usually done by a traceability matrix between test cases and requirements.

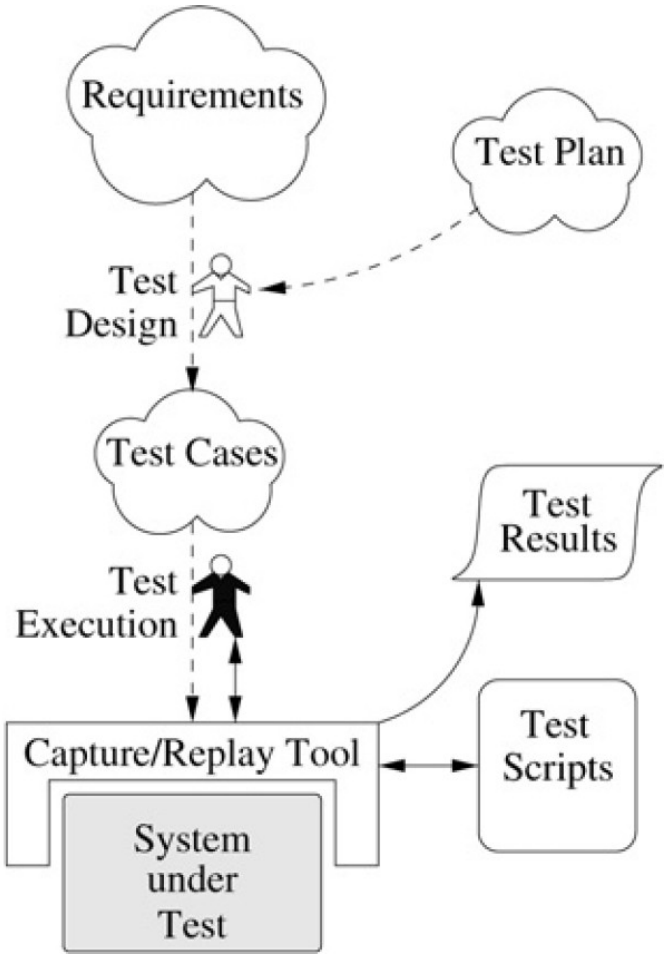
Manual testing



1. test design done by hand (test plan).
2. test execution done by the tester (manually)
3. test verdict done by observation

- very easy to implement
- the cost is low for few tests but increases rapidly

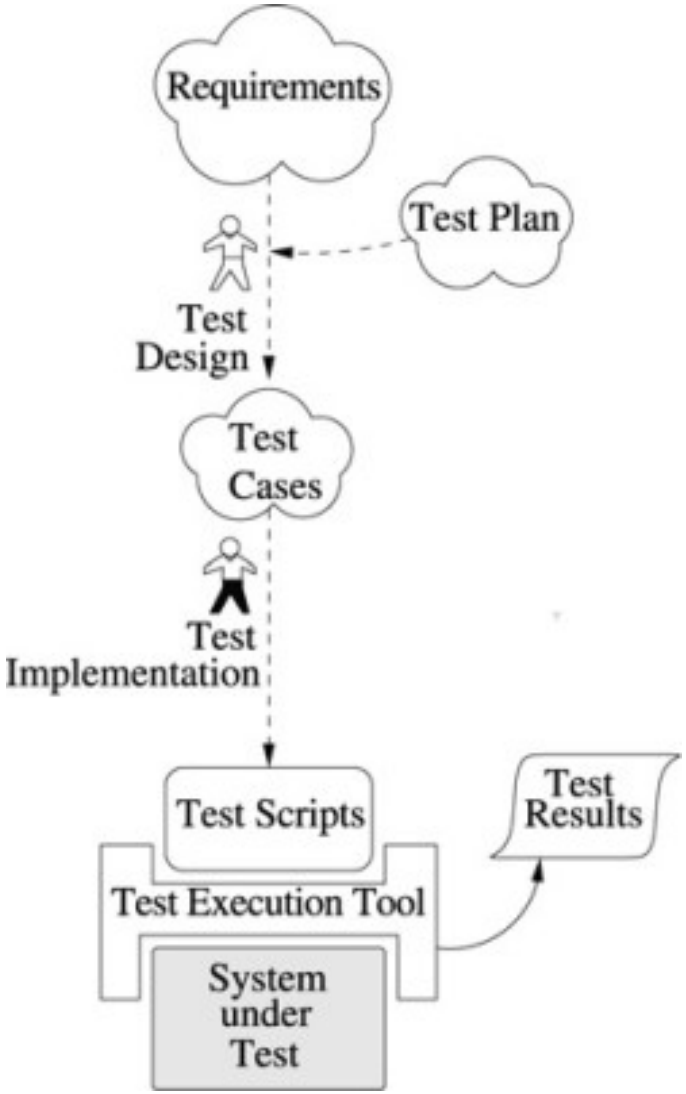
Capture and replay



1. the test are designed and executed as before,
2. test execution is recorded
3. Then when a new release of the SUT must be tested, the capture/replay tool can attempt to rerun all the recorded tests and report which ones fail.

- very easy to rerun test cases
- not robust to sw changes (e.g. the GUI)

Script-Based



1. the tests are scripts written by testers
2. tests can be automatically executed

- very easy to rerun test cases
- maintenance can become costly

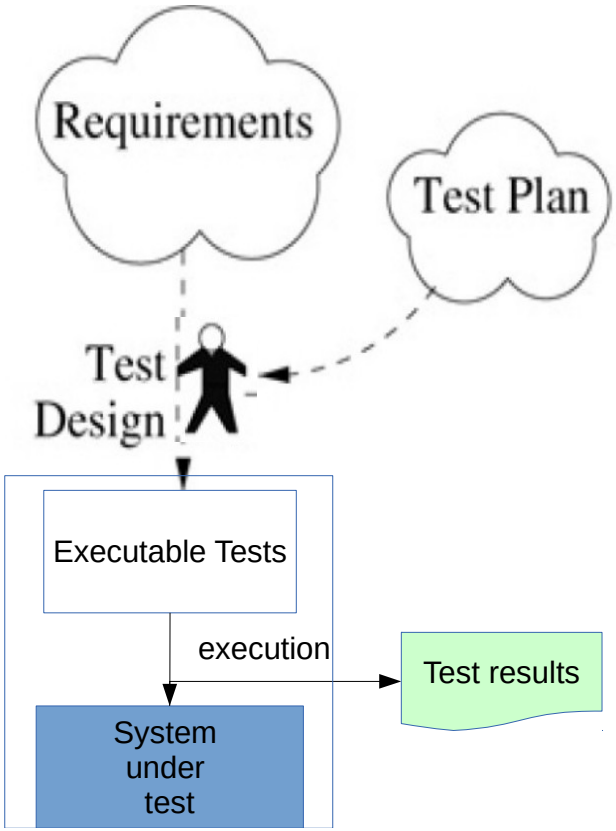
Script-Based TTCN

```
testcase TwoCoffeesPlease () runs on EmptyComponentType{
 var CoffeeMachineComponentType CoffeeMachine;
 var CoffeeDrinkerComponentType CoffeeDrinker;
 CoffeeMachine := CoffeeMachineComponentType.create;
 CoffeeDrinker := CoffeeDrinkerComponentType.create;
 connect(CoffeeDrinker:OutputPort, CoffeeMachine:InputPort);
 connect(CoffeeDrinker:InputPort, CoffeeMachine:OutputPort);
 CoffeeMachine.start( CoffeeMachineFunction() );
 CoffeeDrinker.start( CoffeeDrinkerFunction() );
 timer t; t.start(6.0); t.timeout;
 CoffeeMachine.stop;
}
```

Script-Based typical program

1. initializing the SUT,
2. putting the SUT in the required context,
3. creating the test input values,
4. passing those inputs to the SUT,
5. recording the SUT response,
6. comparing that response with the expected outputs,
7. assigning a pass/fail verdict to each test.

Writing tests as programs



1. the test are designed as before,
2. the tester writes the tests as programs
3. the execution and the verdict are done automaticlly by running the tests

- very easy to rerun test cases
- require more time to write the tests (and maintain)

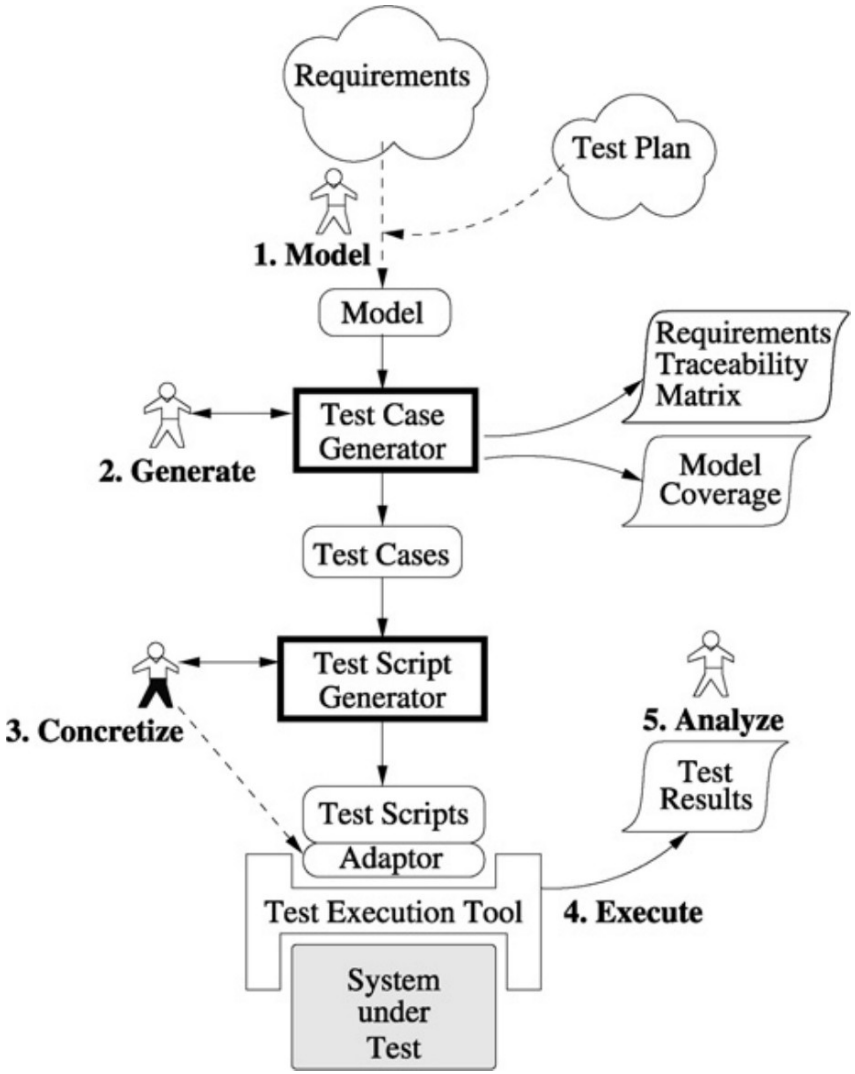
Solved and Remaining Problems

Testing Process	Solved Problems	Remaining Problems
Manual Testing	Functional testing	Imprecise coverage of SUT functionality No capabilities for regression testing Very costly process (every test execution is done manually) No effective measurement of test coverage
Capture/ Replay	Makes it possible to automatically reexecute captured test cases	Imprecise coverage of SUT functionality Weak capabilities for regression testing (very sensitive to GUI changes) Costly process (each change implies recapturing test cases manually)
Script- Based Testing	Makes it possible to automatically execute and reexecute test scripts	Imprecise coverage of SUT functionality Complex scripts are difficult to write and maintain Requirements traceability is developed manually (costly process)
Program-based Testing	No extra language is required	It may require additional effort during maintenance

Section 3

The model-based testing process

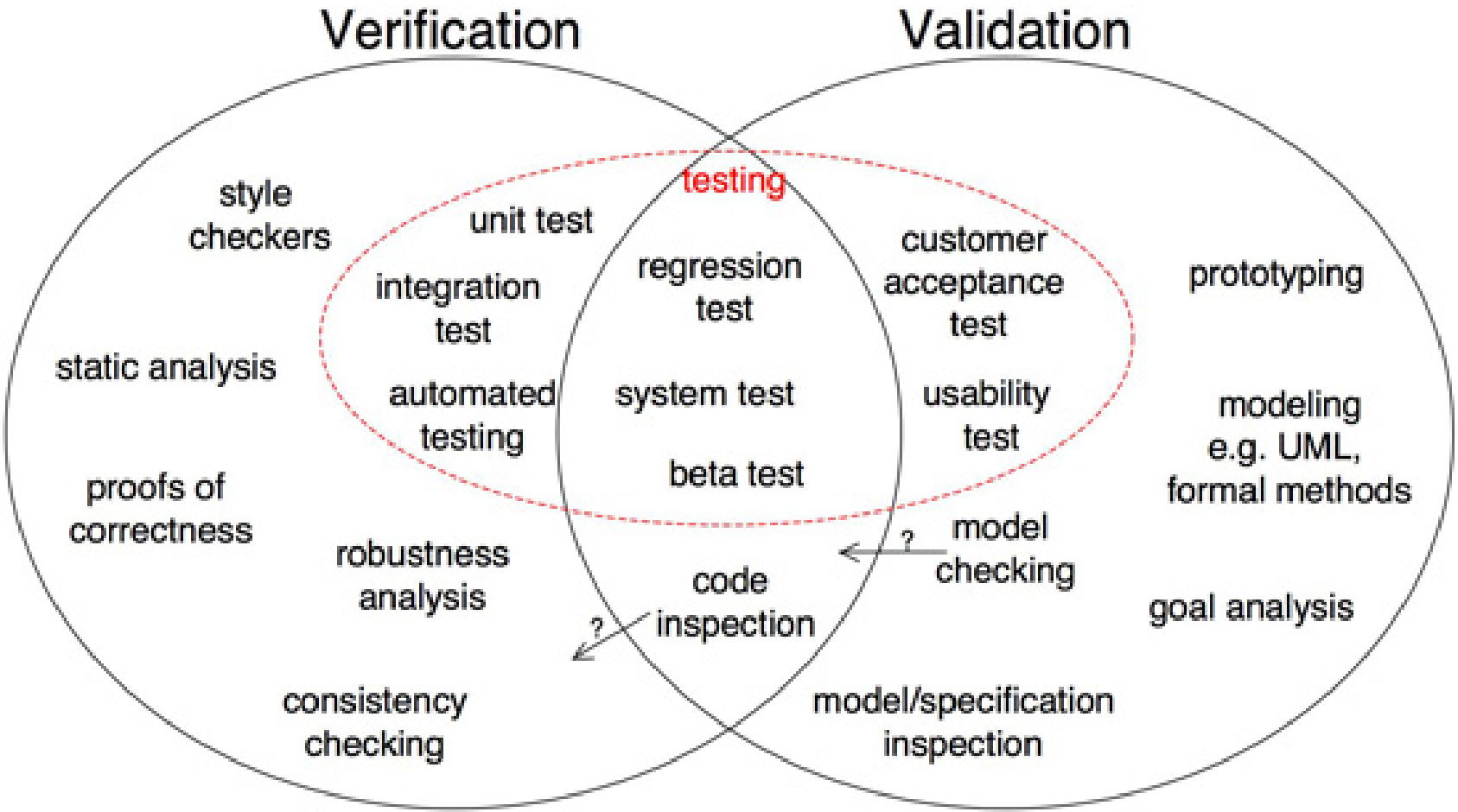
Model-Based testing



steps for MBT

1. Model the SUT and/or its environment.
2. Generate abstract tests from the model.
3. Concretize the abstract tests to make them executable.
4. Execute the tests on the SUT and assign verdicts.
5. Analyze the test results.

Is testing verification or validation



Section 4

Formal verification

What is formal verification

formal verification

techniques that **construct** a **mathematical** proof of **consistency** between some formal representation of a program or design and a formal specification.

construct it is often an human activity

mathematical based on mathematics and logics

consistency between two "artifacts" (or two roles of the same artifact)

- formal representation of a program or design
- formal specification of the properties the program or design should have

Program verification

- Current practice is to gather evidence for program correctness by testing
- However, exhaustive testing is difficult even for small programs
- Testing cannot prove that a program is correct.
- Program verification can prove that a program is correct
  1. starts with the formal description of a specification for a program (It may be implicit, e.g. a null pointer is never dereferenced)
  2. a proof (in some proof system) that the program meets the formal specification.

Runtime verification

- Checking at runtime that a program behaves as expected.

Formal model verification

- Formal verification is the process of checking whether a design satisfies some requirements (properties).
- The design must be converted in a “verifiable” format. For example a FSM
- The property must be given in formal way

Example

- A light is initially off. If the user presses a button becomes on if it is off and viceversa.
- Model: the FSM
- Properties
  - if the user never presses the button, the light stays off.
  - whenever the light is off and the user presses the button it becomes on
  - whenever the light is on and the user presses the button it becomes off
- differenze con il testing

model checking

- The most popular method for automatic formal verification is model checking.
- Given a model of a system, exhaustively and automatically check whether this model meets a given specification.
- Example: software model checker of Java programs (Java Path Finder) used by NASA