5. Testing

Angelo Gargantini

Testing e Verifica del Software AA 2526

12 marzo 2026

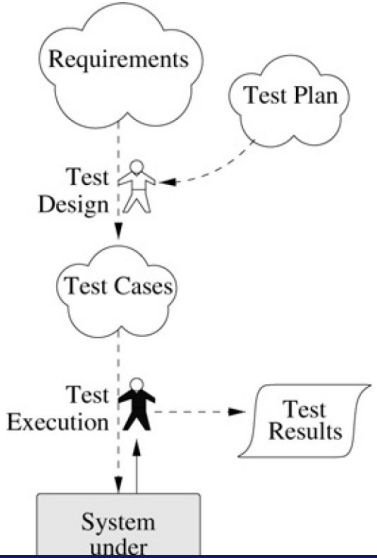
Key issues

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.

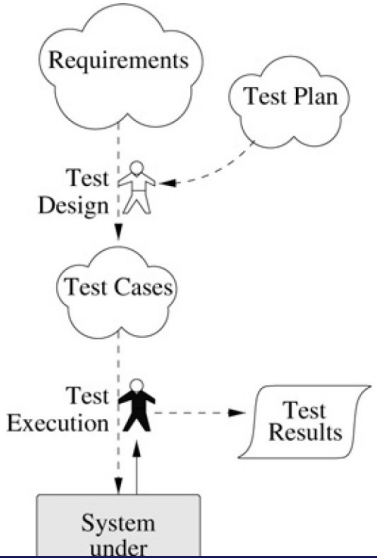
Verifying how the tests cover the requirements: To manage the quality of the testing process (and therefore the quality of the product), one must measure the coverage of the requirements by the test suite. This is usually done by a traceability matrix between test cases and requirements.

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

Manual testing

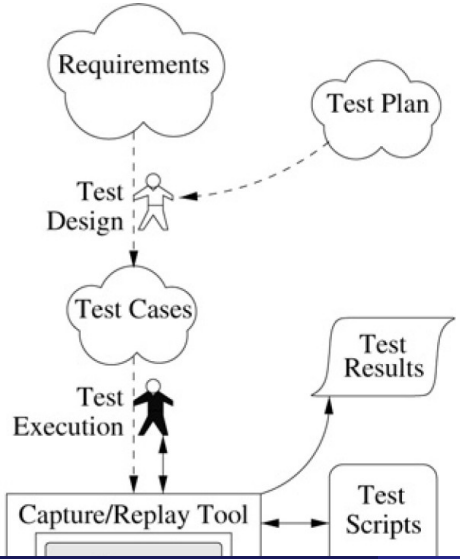


1. test design done by hand (test plan).
2. test execution done by the tester (manually)
3. test verdict done by observation
   - very easy to implement
   - the cost is low for few tests but increases rapidly

crowd testing

- Using the crowd for testing...

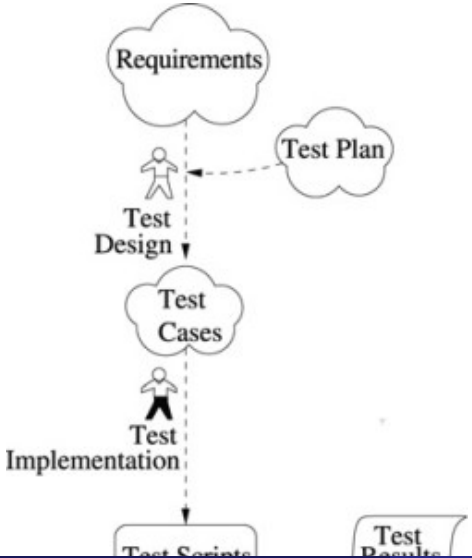
Capture and replay



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

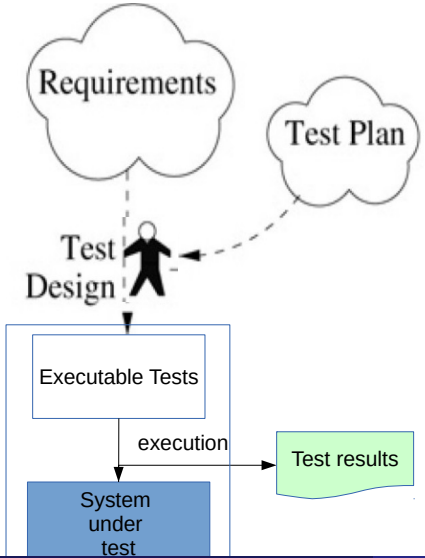
Script-Based TTCN

```
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 System Under Test (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

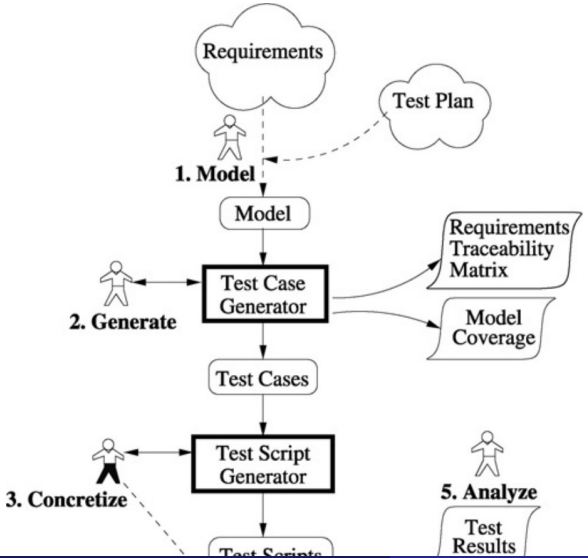


1. the test are designed as before,
2. the tester writes the tests as programs
3. the execution and the verdict are done automaticlly by running the tests
   - very easy to rerun test cases
   - require more time to write the tests (and maintain)

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

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.