Testing process and levels

Zoltan Micskei, Istvan Majzik

Budapest University of Technology and Economics
Fault Tolerant Systems Research Group

Main topics of the course

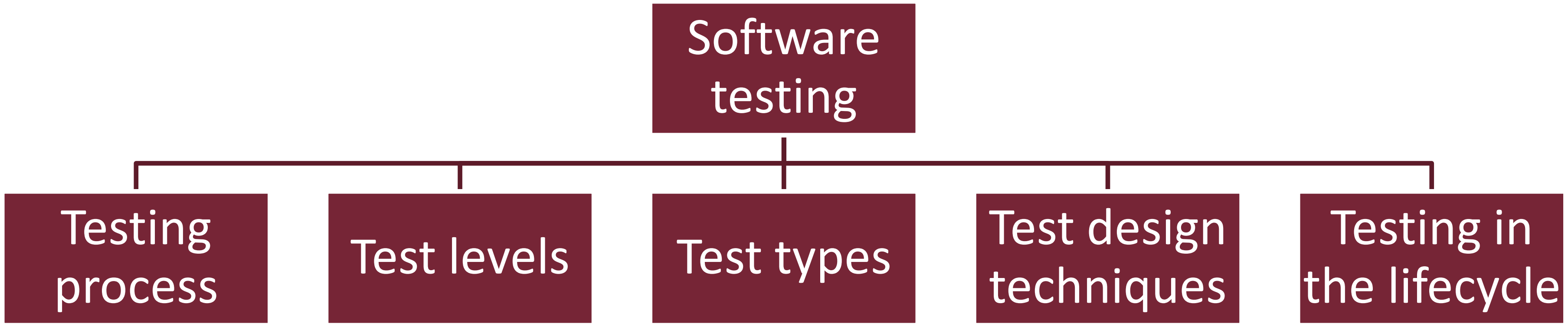
- Overview (1)
    - V&V techniques, Critical systems
- Static techniques (2)
    - Verifying specifications
    - Verifying source code
- **Dynamic techniques: Testing (7)**
    - Developer testing, Test design techniques
    - **Testing process and levels**, Test generation, Automation
- System-level verification (3)
    - Verifying architecture, Dependability analysis
    - Runtime verification

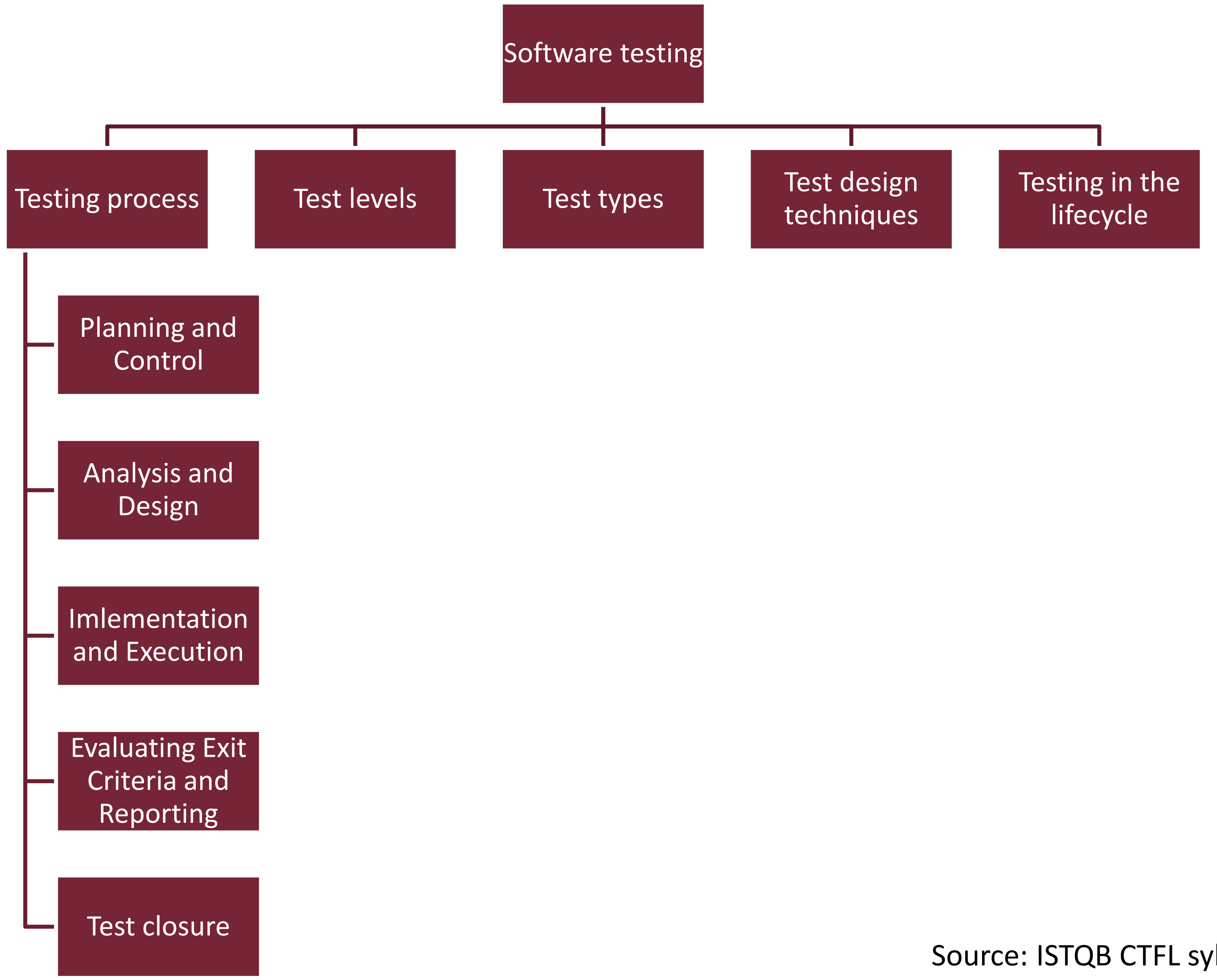
TESTING PROCESS

Learning outcomes

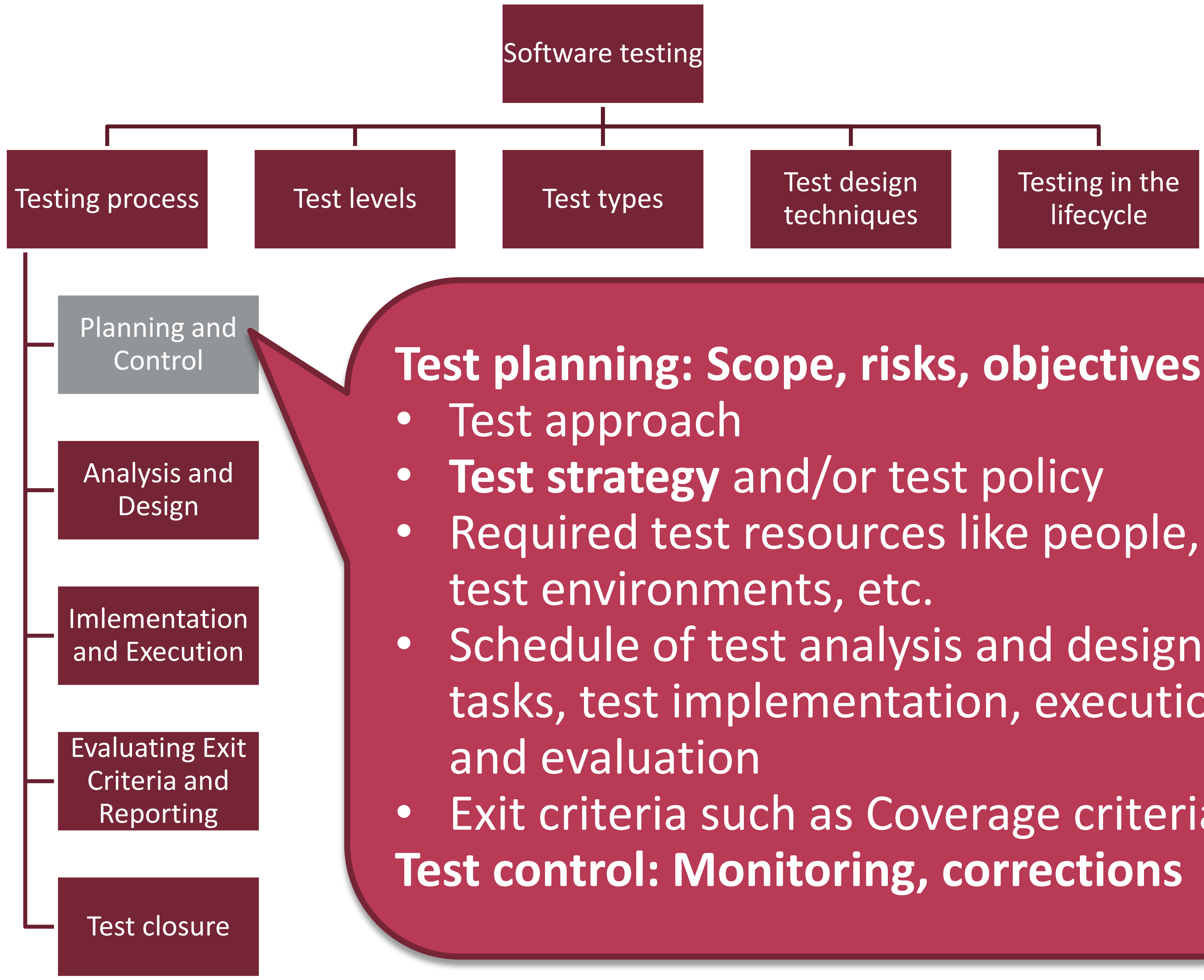
- Explain the activities and tasks in the typical test process (K2)

Overview of testing concepts





Source: ISTQB CTFL syllabus



Test planning: Scope, risks, objectives

- Test approach
- **Test strategy** and/or test policy
- Required test resources like people, test environments, etc.
- Schedule of test analysis and design tasks, test implementation, execution and evaluation
- Exit criteria such as Coverage criteria

Test control: Monitoring, corrections

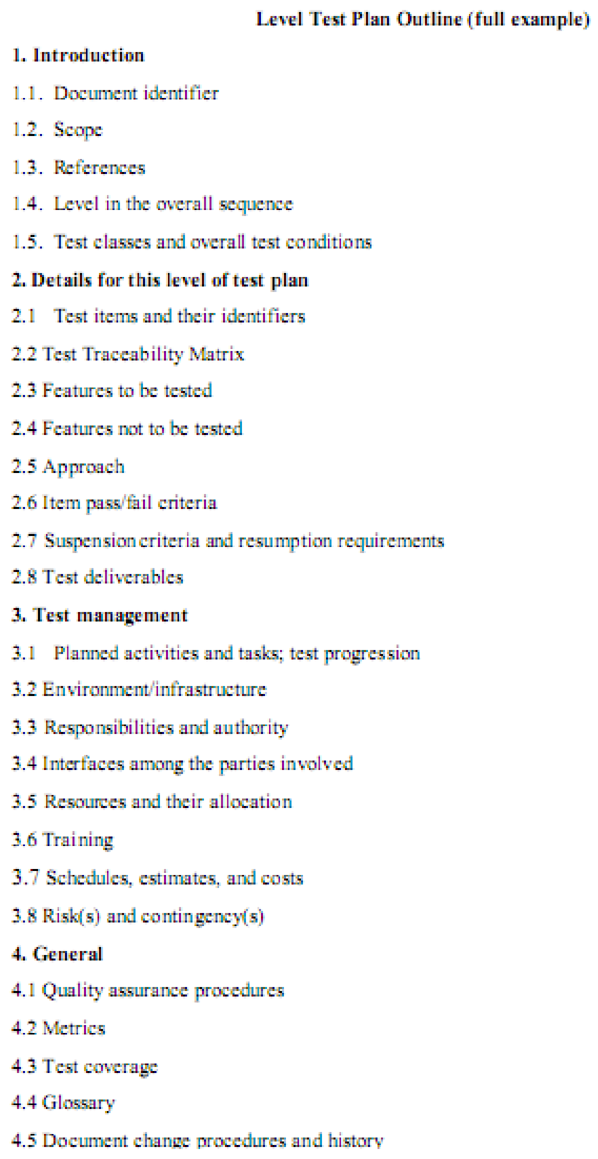
Test strategy

Guidelines for	Possible example:
What methodology?	Test-driven development
What kinds of tests?	Module & system
What tools?	JUnit & GUI Tester
Who will test?	Developers & test engineers
Exit criteria?	At least 90% statement coverage & every use case requirement
What documentation?	Test Report according to IEEE 829

Test plan

- Mapping test strategy to the actual test project
    - Test objectives
    - Test objects, test environment
    - Resources, roles,
    - Schedules
- Defining test phases
    - Length of phase
    - Exit criteria
    - Measuring quality of testing

Test documentation



Level Test Plan Outline (full example)

1. Introduction

1.1. Document identifier

1.2. Scope

1.3. References

1.4. Level in the overall sequence

1.5. Test classes and overall test conditions

2. Details for this level of test plan

2.1 Test items and their identifiers

2.2 Test Traceability Matrix

2.3 Features to be tested

2.4 Features not to be tested

2.5 Approach

2.6 Item pass/fail criteria

2.7 Suspension criteria and resumption requirements

2.8 Test deliverables

3. Test management

3.1 Planned activities and tasks; test progression

3.2 Environment/infrastructure

3.3 Responsibilities and authority

3.4 Interfaces among the parties involved

3.5 Resources and their allocation

3.6 Training

3.7 Schedules, estimates, and costs

3.8 Risk(s) and contingency(s)

4. General

4.1 Quality assurance procedures

4.2 Metrics

4.3 Test coverage

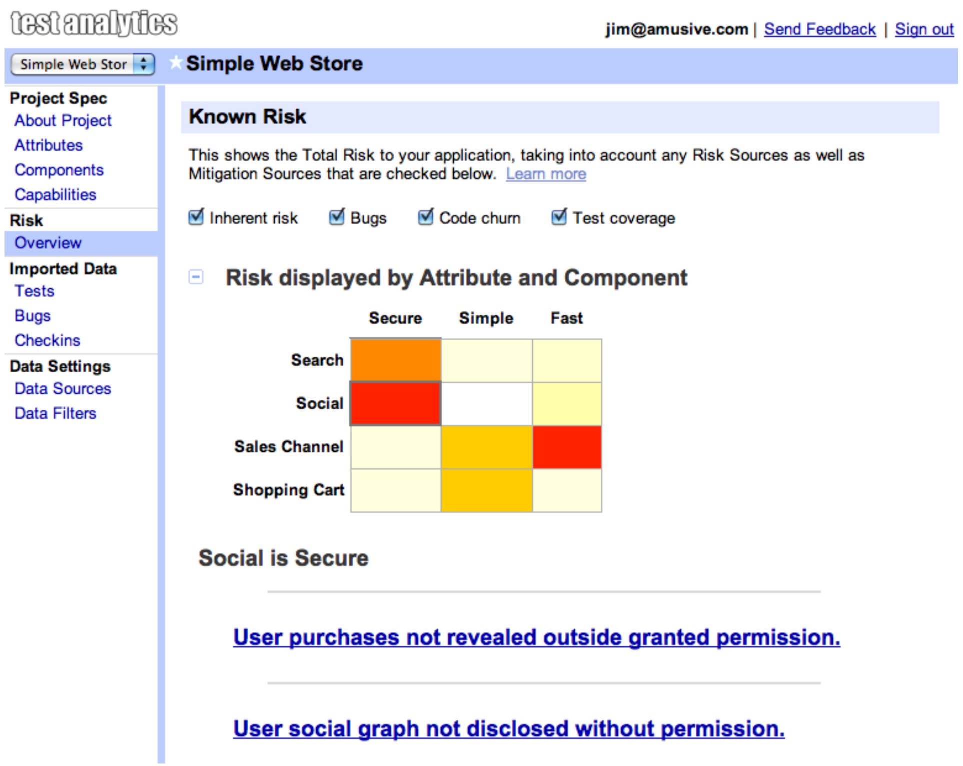
4.4 Glossary

4.5 Document change procedures and history

- IEEE 829 - Standard for Software and System Test Documentation (1998)
  - Test Plan (SPACEDIRT: Scope, People, Approach, Criteria, Environment, Deliverables, Incidentals, Risks, Tasks)
  - Test specifications: Test Design, Test Case, Test Procedure Specifications
  - Test reporting: Test Item Transmittal Report, Test Log, Test Incident Report, Test Summary Report

Google "10 minute test plan"

- Why do write a plan that is not used and updated?
- Keep only the most important
  - Attributes, Components, Capabilities (ACC)



Known Risk

This shows the Total Risk to your application, taking into account any Risk Sources as well as Mitigation Sources that are checked below. Learn more

✓ Inherent risk ✓ Bugs ✓ Code churn ✓ Test coverage

Risk displayed by Attribute and Component

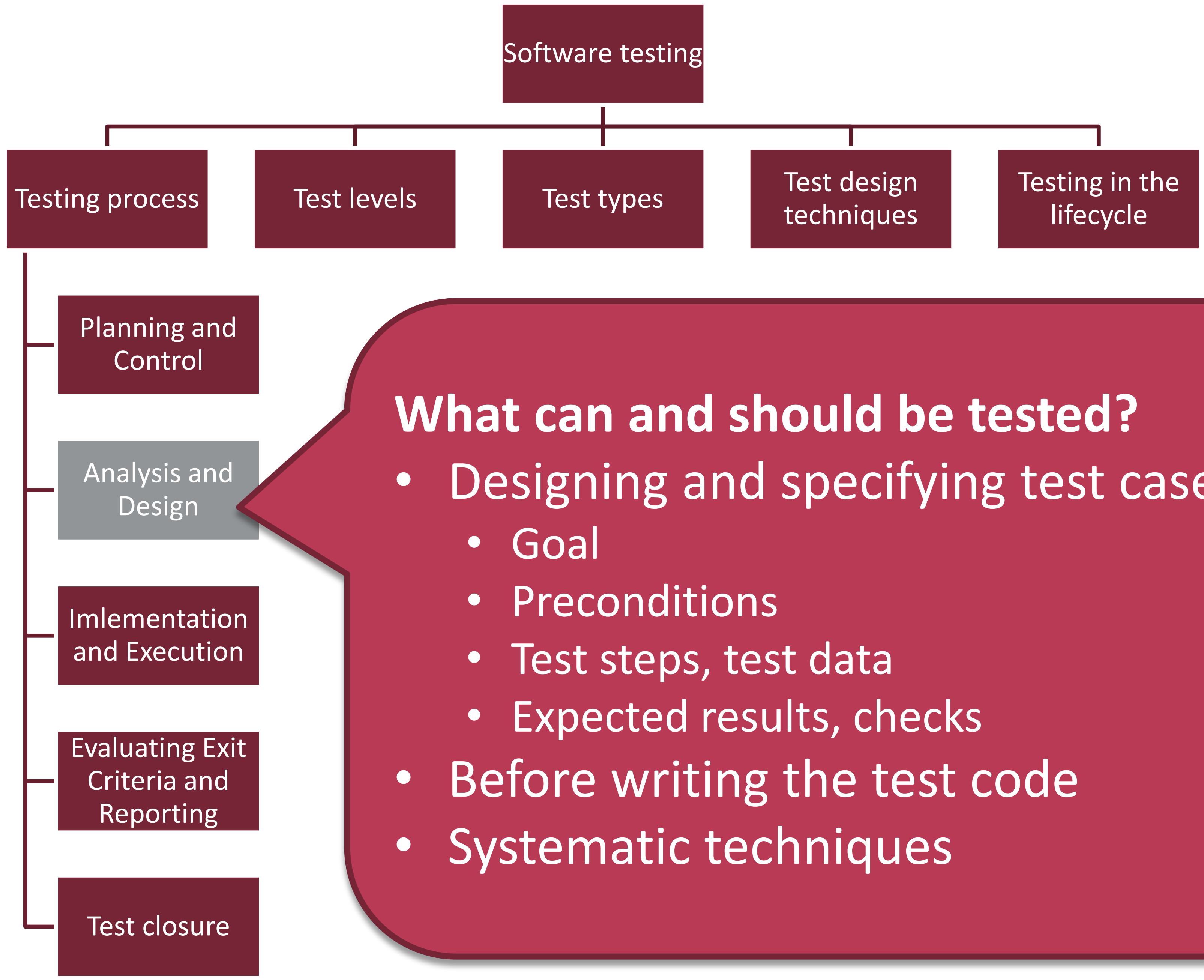
	Secure	Simple	Fast
Search	🟧	⬜	⬜
Social	🟥	⬜	🟨
Sales Channel	⬜	🟨	🟥
Shopping Cart	⬜	🟨	⬜

Social is Secure

User purchases not revealed outside granted permission.

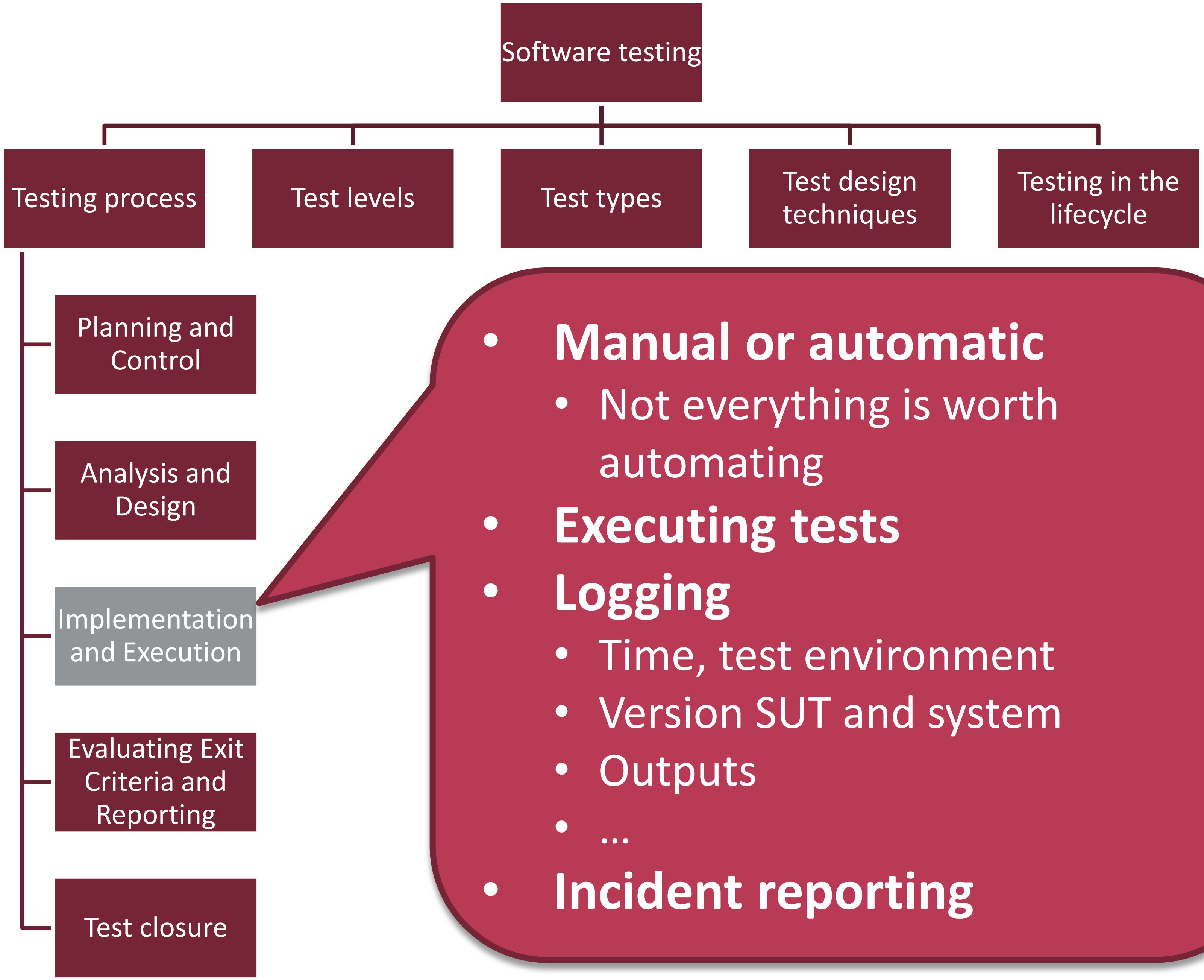
User social graph not disclosed without permission.

Source:
Google Test Analytics - Now in Open Source

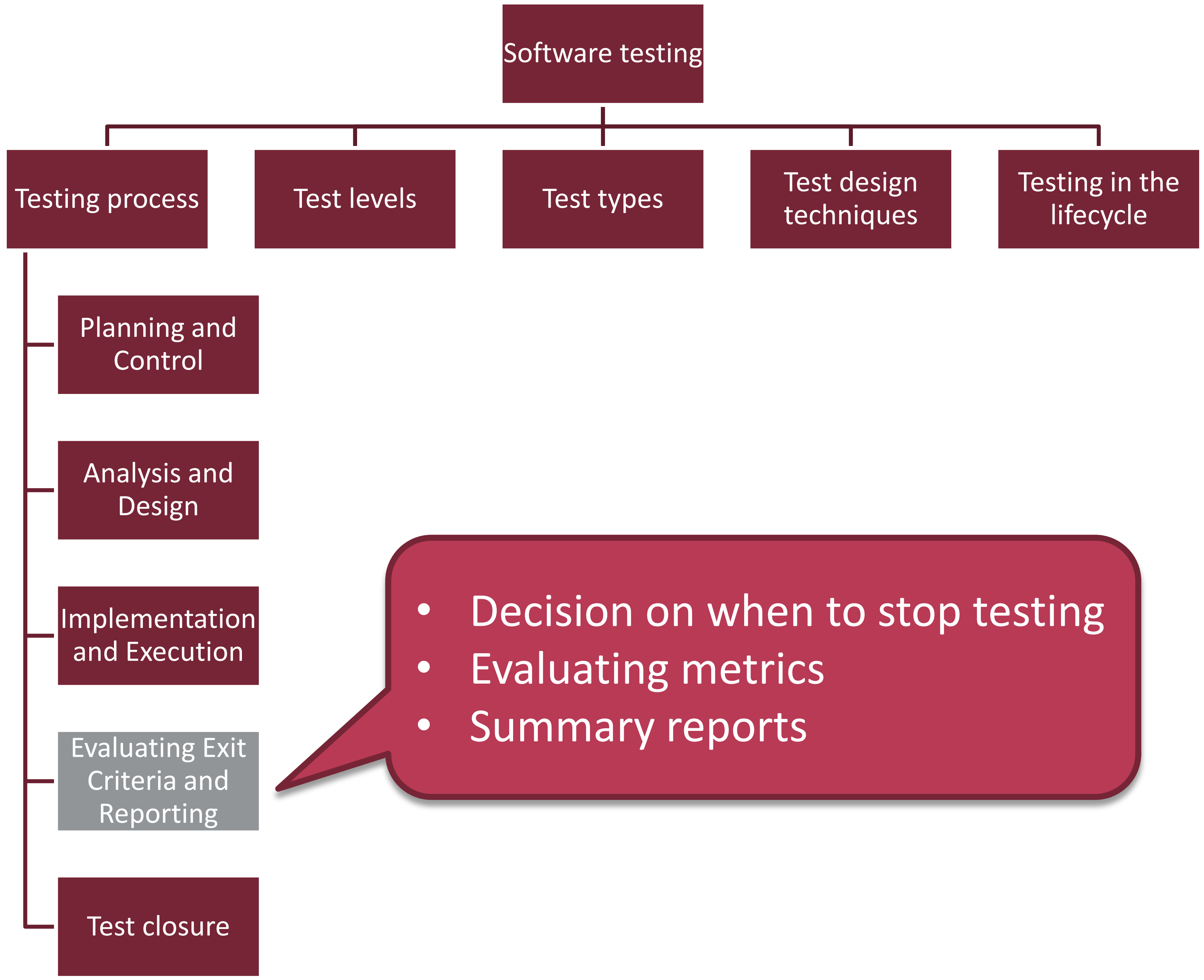


What can and should be tested?

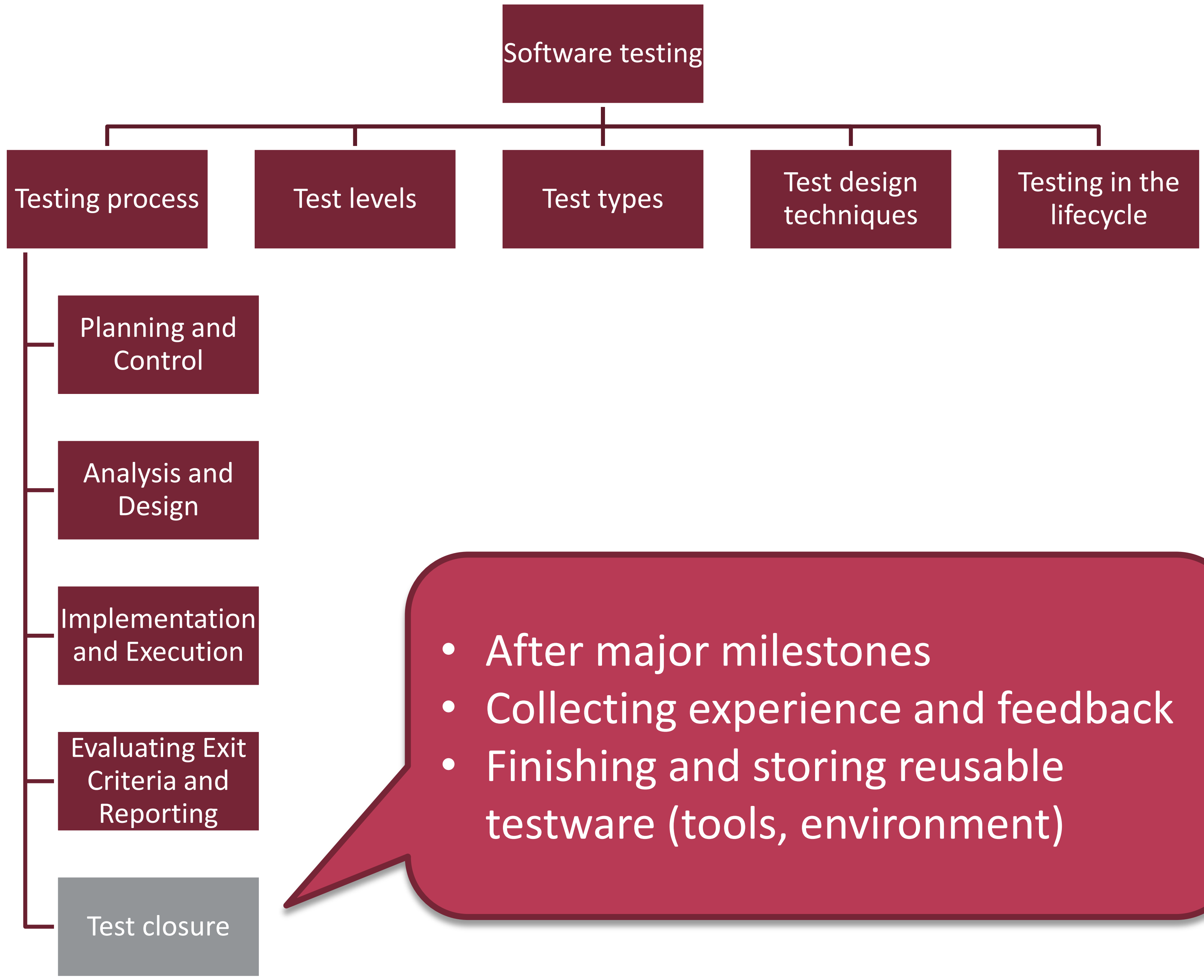
- Designing and specifying test cases
  - Goal
  - Preconditions
  - Test steps, test data
  - Expected results, checks
- Before writing the test code
- Systematic techniques



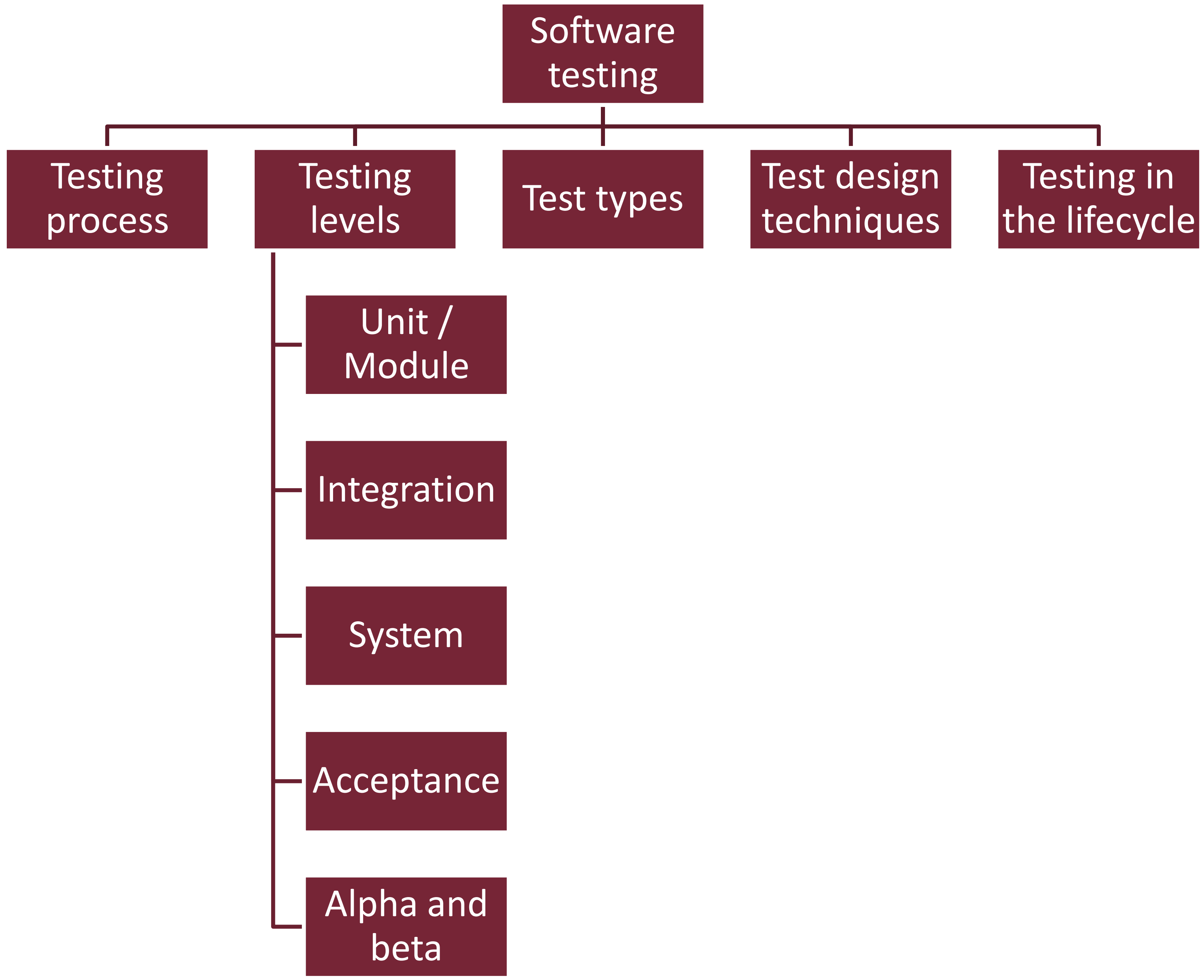
- **Manual or automatic**
  - Not everything is worth automating
- **Executing tests**
- **Logging**
  - Time, test environment
  - Version SUT and system
  - Outputs
  - …
- **Incident reporting**

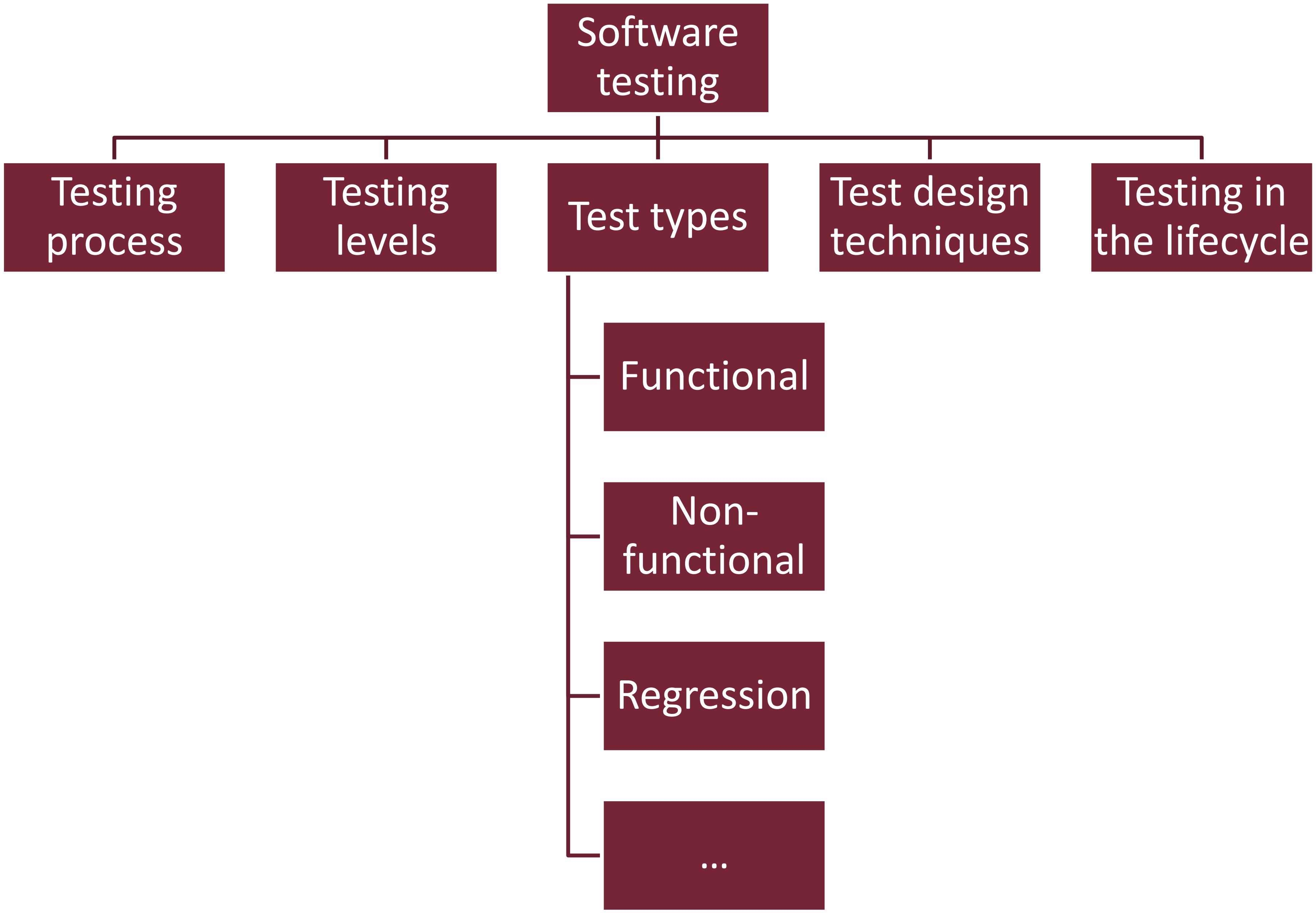


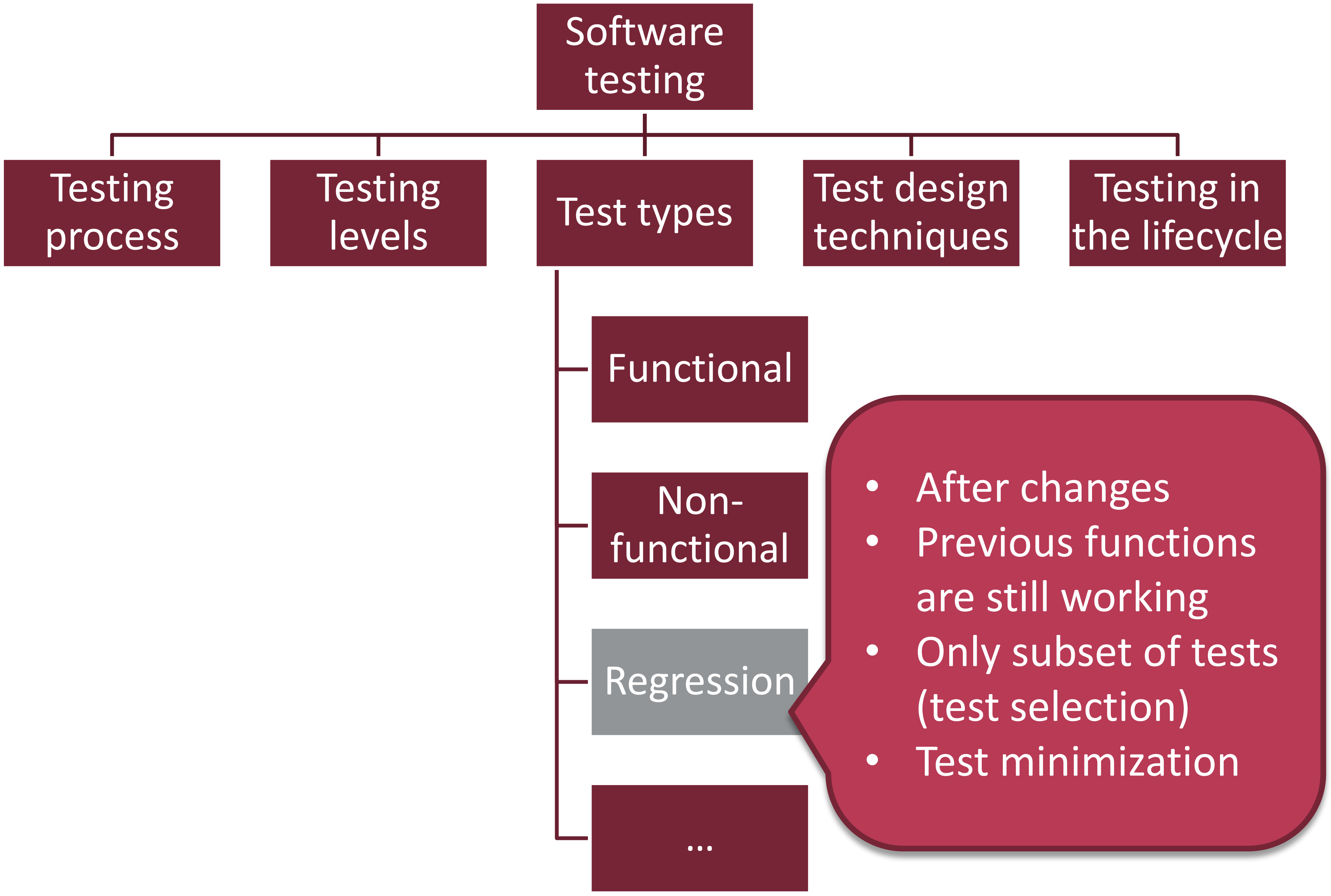
- Decision on when to stop testing
- Evaluating metrics
- Summary reports



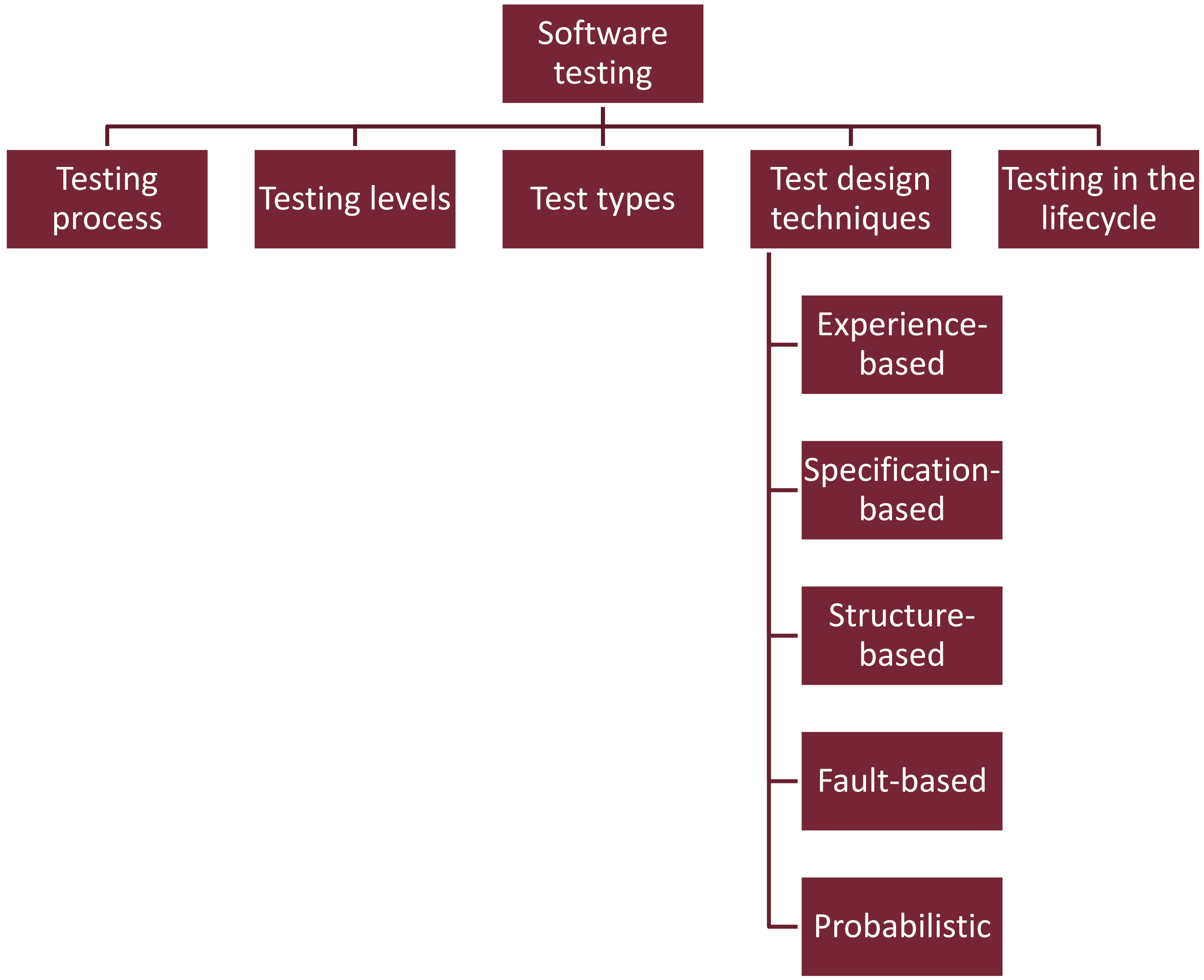
- After major milestones
- Collecting experience and feedback
- Finishing and storing reusable testware (tools, environment)

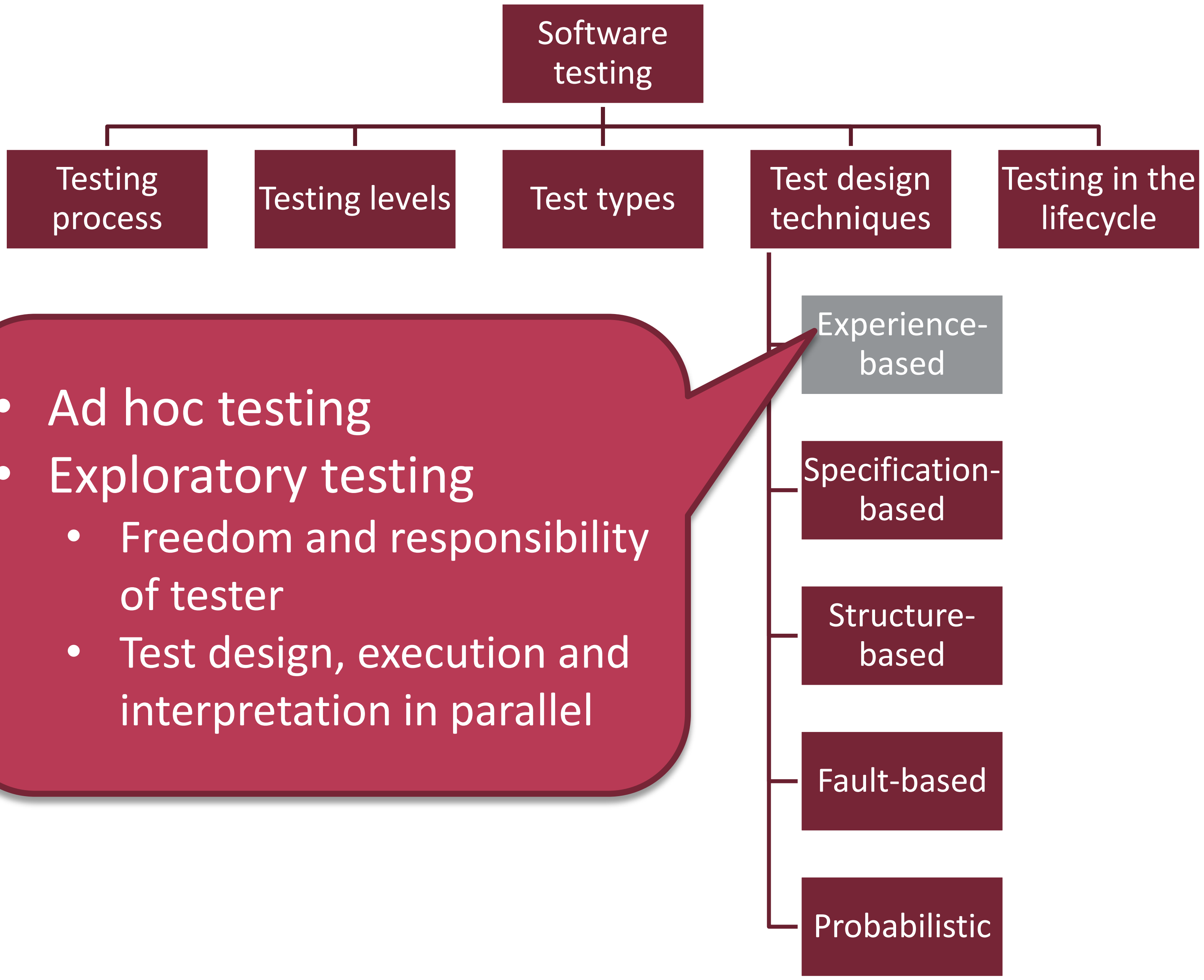

Software testing
Testing process
Testing levels
Test types
Test design techniques
Testing in the lifecycle
Unit / Module
Integration
System
Acceptance
Alpha and beta


Software testing
Testing process
Testing levels
Test types
Test design techniques
Testing in the lifecycle
Functional
Non-functional
Regression
...

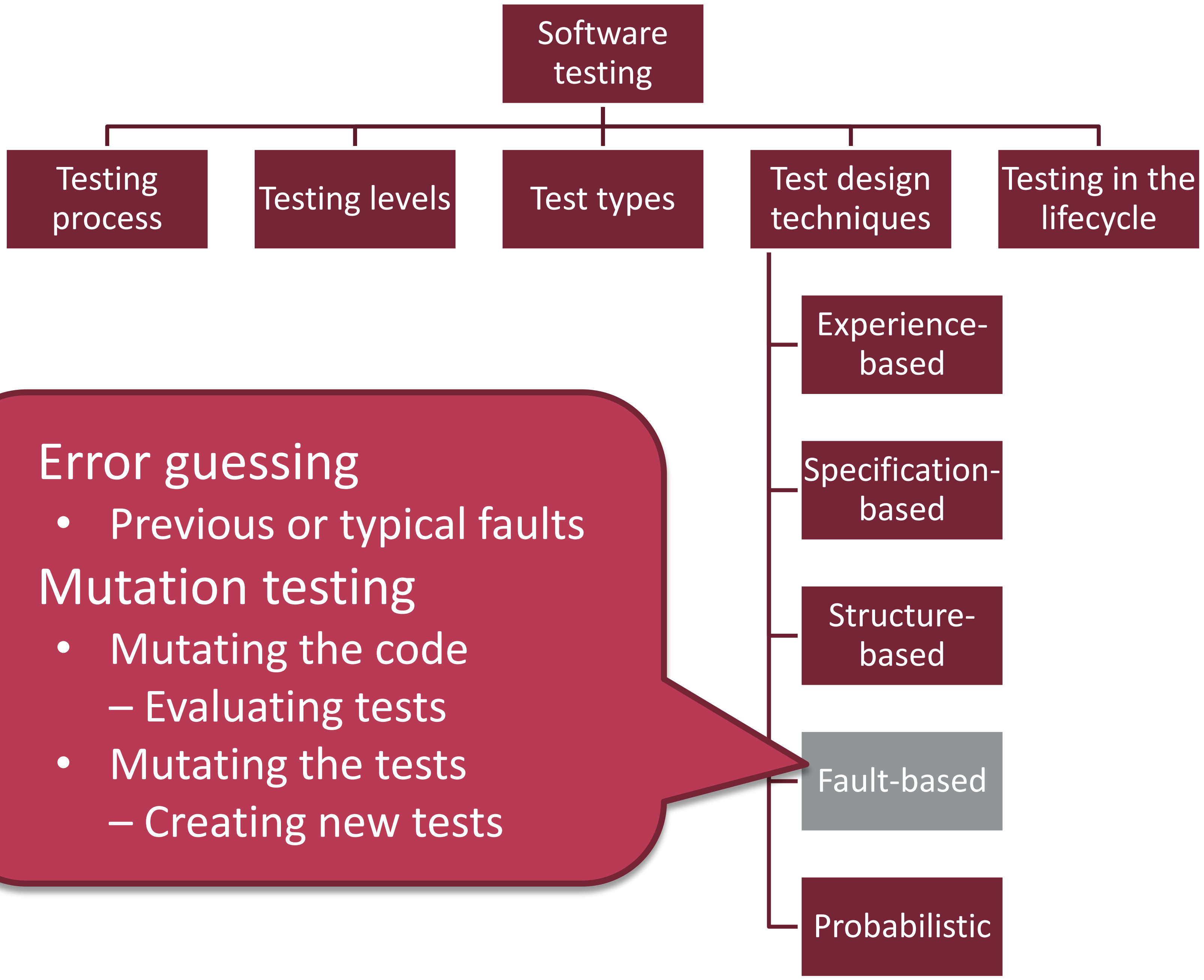


- After changes
- Previous functions are still working
- Only subset of tests (test selection)
- Test minimization

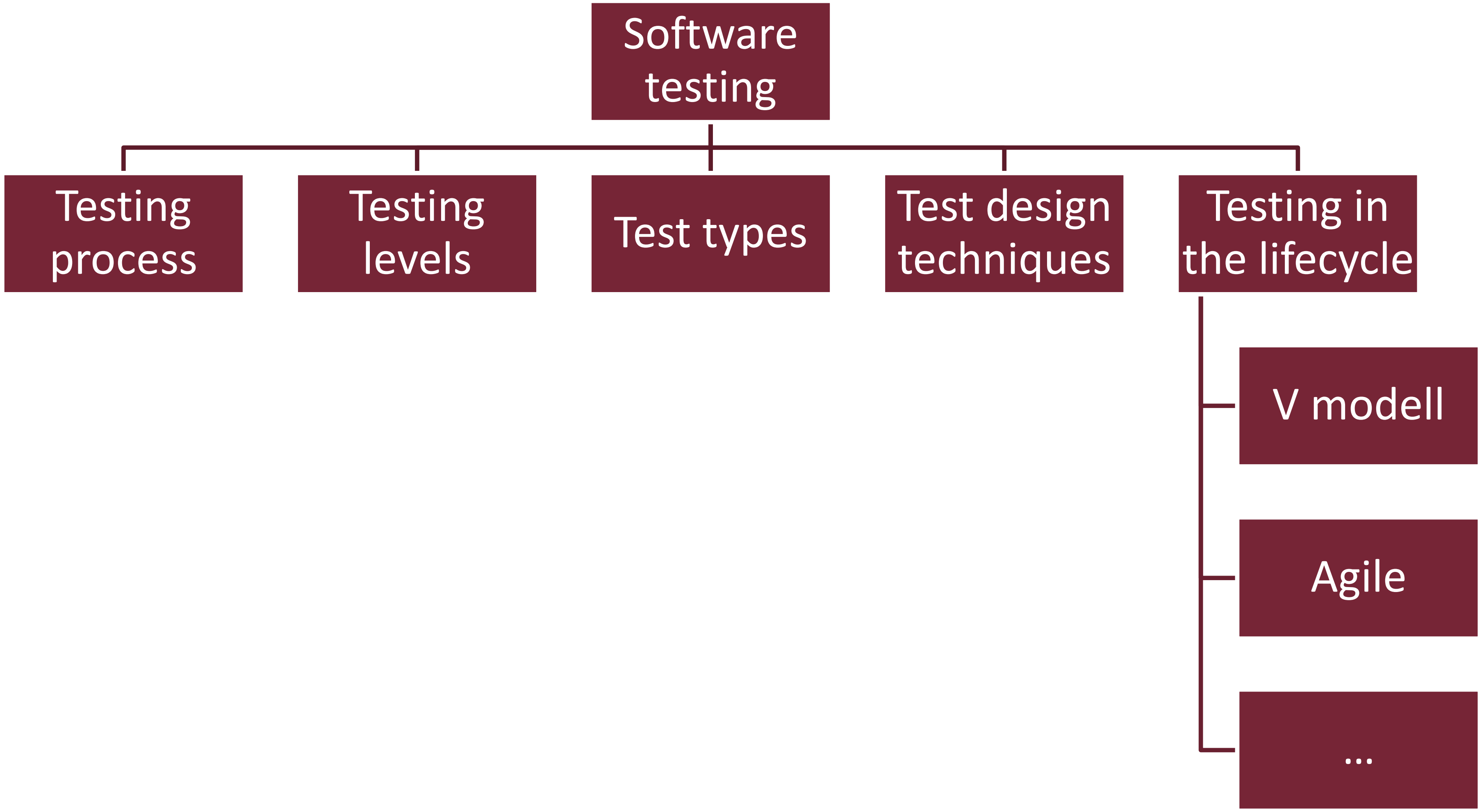

Software testing
Testing process
Testing levels
Test types
Test design techniques
Testing in the lifecycle
Experience-based
Specification-based
Structure-based
Fault-based
Probabilistic



- Ad hoc testing
- Exploratory testing
  - Freedom and responsibility of tester
  - Test design, execution and interpretation in parallel



- Error guessing
  - Previous or typical faults
- Mutation testing
  - Mutating the code
    – Evaluating tests
  - Mutating the tests
    – Creating new tests


Software testing
Testing process
Testing levels
Test types
Test design techniques
Testing in the lifecycle
V modell
Agile
…

- Software testing
  - Testing process
    - Planning and Control
    - Analysis and Design
    - Implementation and Execution
    - Evaluating Exit Criteria and Reporting
    - Test closure
  - Testing levels
    - Unit / Module
    - Integration
    - System
    - Acceptance
    - Alpha and beta
  - Test types
    - Functional
    - Non-functional
    - Regression
    - …
  - Test design techniques
    - Experience-based
    - Specification-based
    - Structure-based
    - Fault-based
    - Probabilistic
  - Testing in the lifecycle
    - V modell
    - Agile
    - …

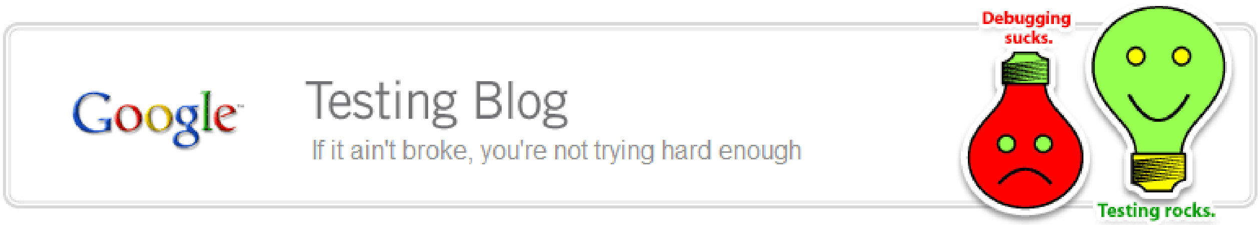
TESTING PRACTICES

Testing @ Microsoft

- Software Developer Engineer in Test (SDET)
- Same career paths as developers
  - Testing is not an entry position
  - Test manager is not a promotion but a different path
- „Hiring testers to pound quality into a product after it’s been developed is a waste of money.”
- 10 year support cycle for major releases
  - Worth investing in good test automation

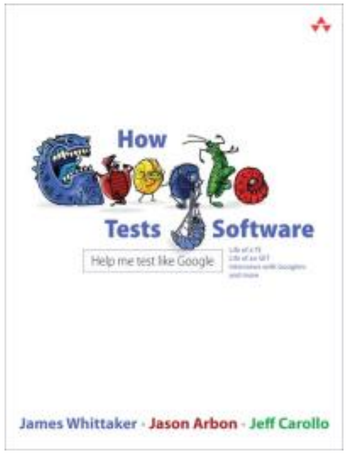
„How we test software at Microsoft”, Microsoft Press, ISBN 0735624259, 2008.

Testing @ Google

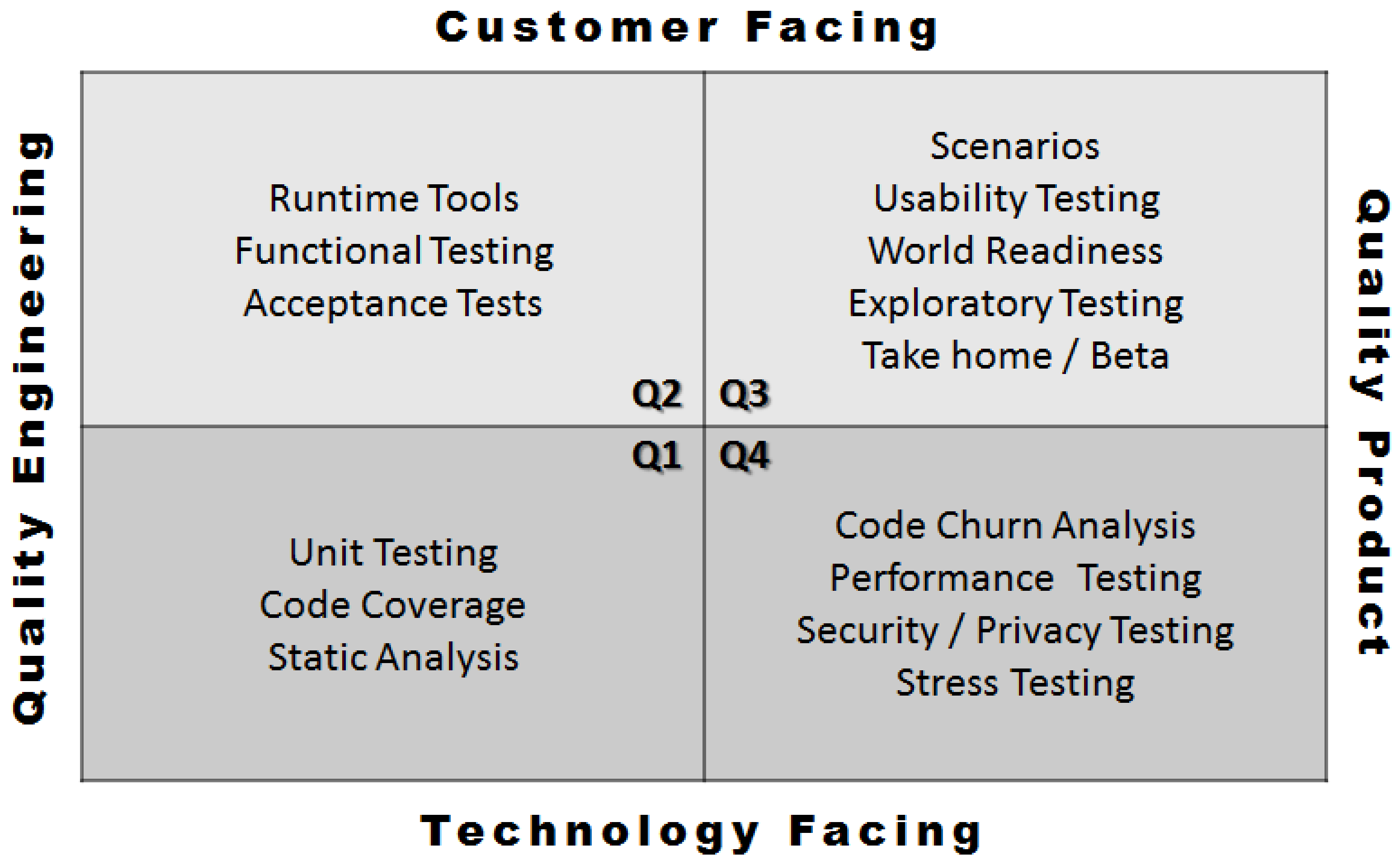


- Roles
  - Test Engineer (TE)
  - Software Engineer in Test & Infrastructure (SETI)

- „The burden of quality is on the shoulders of those writing the code.”
- „Do not hire too many testers.”


How Google Tests Software
Help me test like Google
James Whittaker · Jason Arbon · Jeff Carollo

Testing Quadrants



Source: http://angryweasel.com/blog/riffing-on-the-quadrants/

TEST LEVELS

Learning outcomes

- Distinguish the scope of different test levels (K2)
- Describe the different integration testing approaches (K2)
- Recall the goals of system verification and system validation testing (K1)

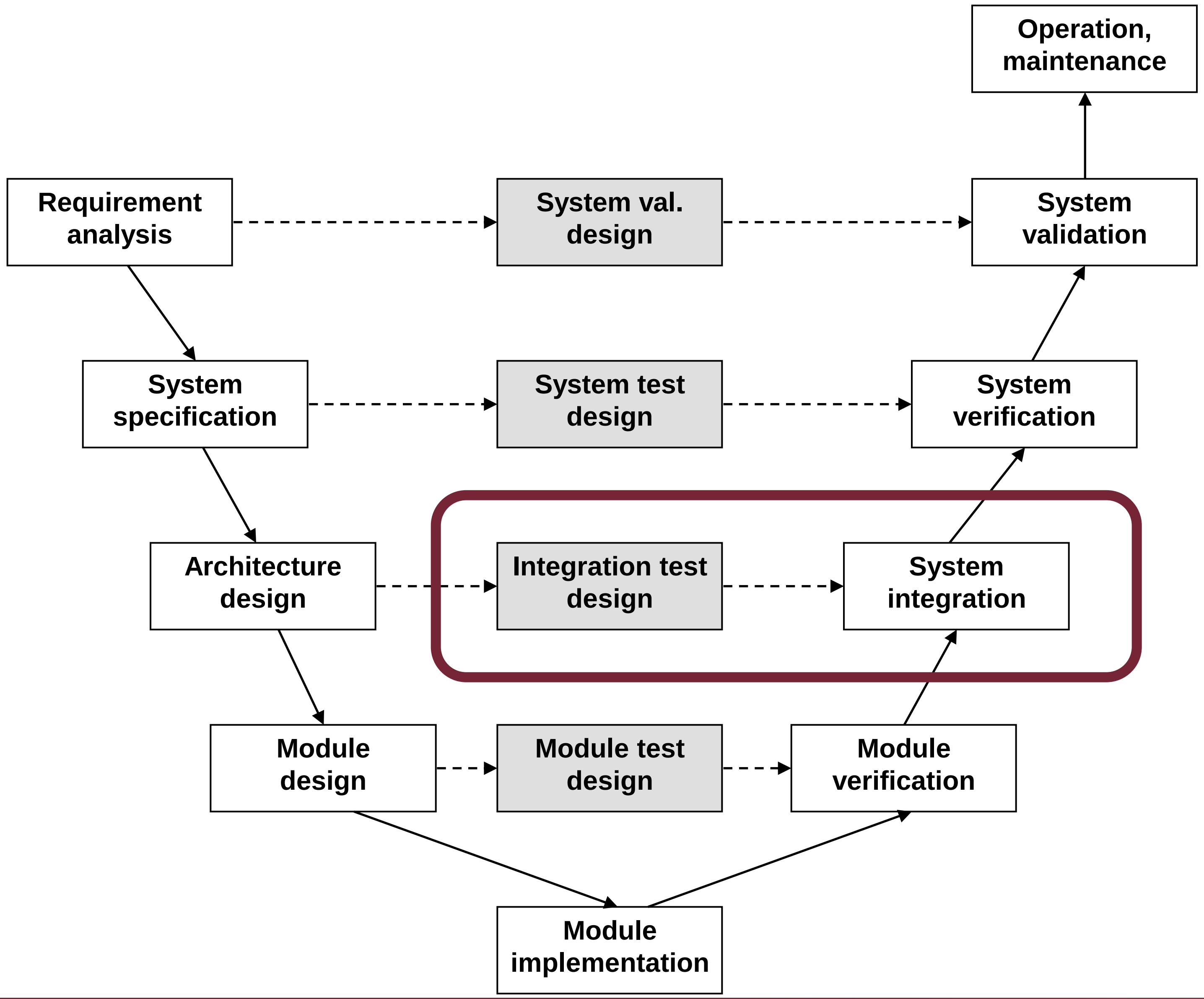
Characteristics of tests in different levels

Recommendations from *How Google Tests Software*:

	Small	Medium	Large
Execution time	< 100 ms	< 1 sec	As fast as poss.
Time limit (kill)	1 minute	5 minutes	1 hour

Resource	Small	Medium	Large
Network (socket)	Mocked	only localhost	Yes
Database	Mocked	Yes	Yes
File access	Mocked	Yes	Yes
System call	No	Not recommended	Yes
Multiple threads	Not recommended	Yes	Yes
Sleep	No	Yes	Yes
System properties	No	Yes	Yes

Testing and test design in the V-model

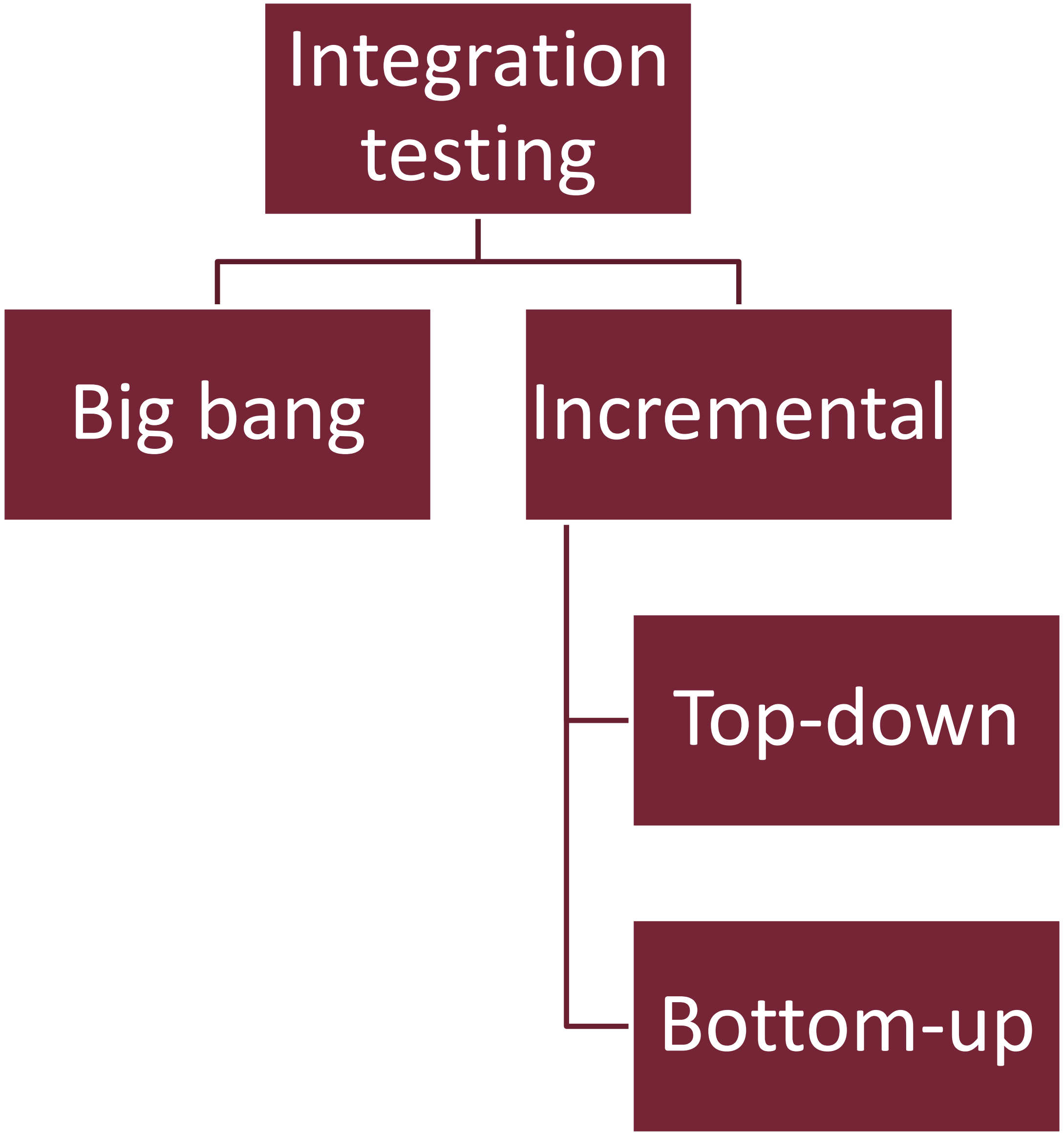


Integration testing

Testing the **interactions** of modules

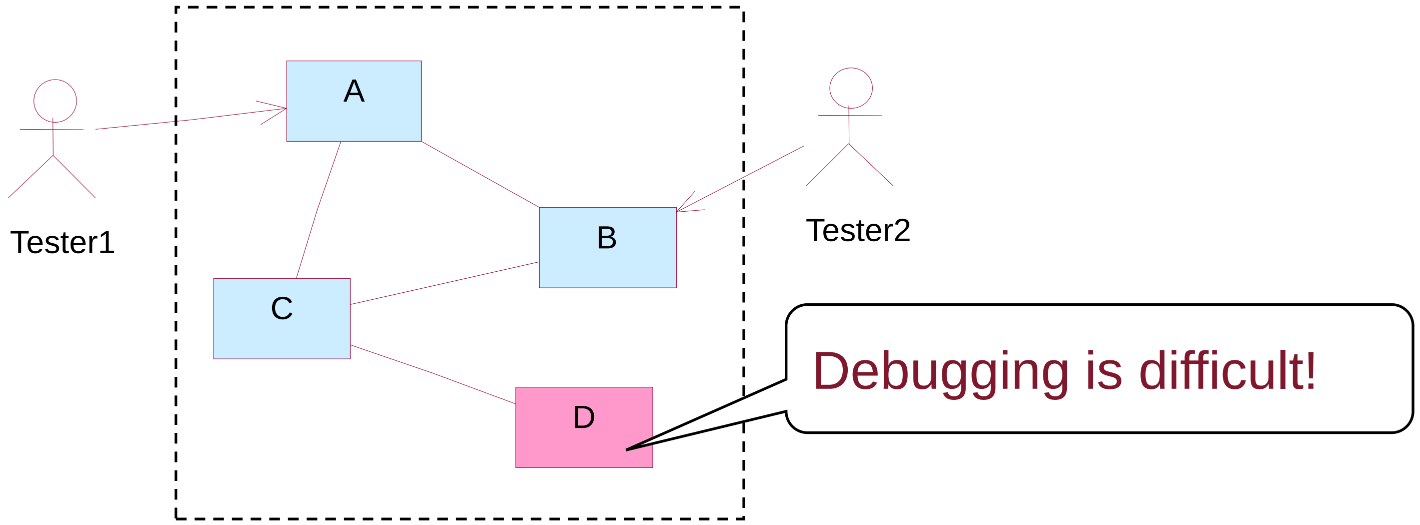
- Motivation
  - The system-level interaction of modules may be incorrect despite the fact that all modules are correct
- Methods
  - Functional testing: Testing scenarios
    - Sometimes the scenarios are part of the specification
  - (Structure based testing at module level)
- Approaches
  - “Big bang” testing: integration of all modules
  - Incremental testing: stepwise integration of modules

Integration testing approaches



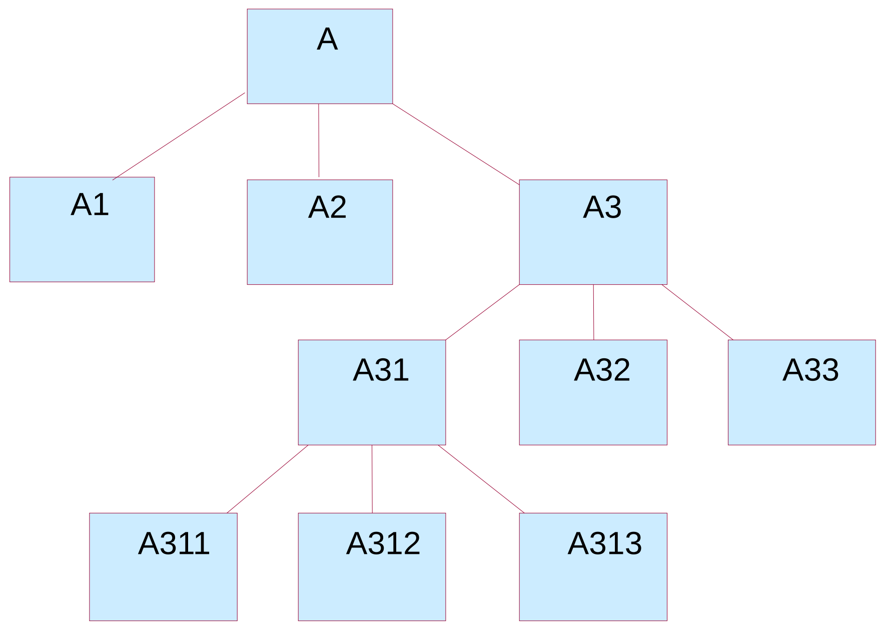
"Big bang" testing

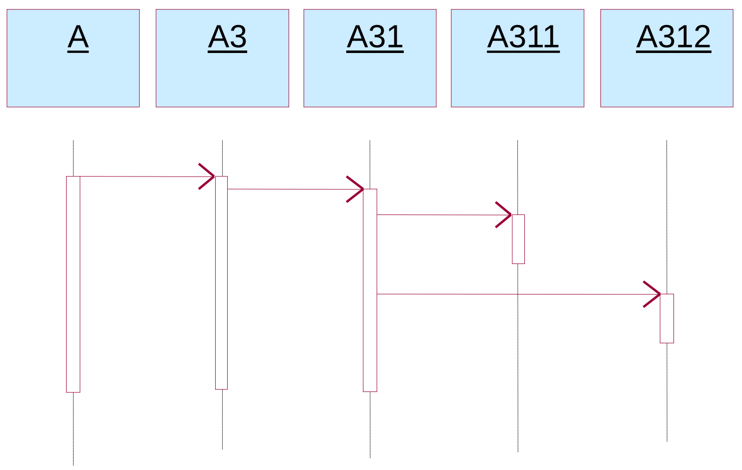
- Integration of all modules and testing using the external interfaces of the integrated system
- External test executor
- Based of the functional specification of the system
- To be applied only in case of small systems



Incremental integration and testing

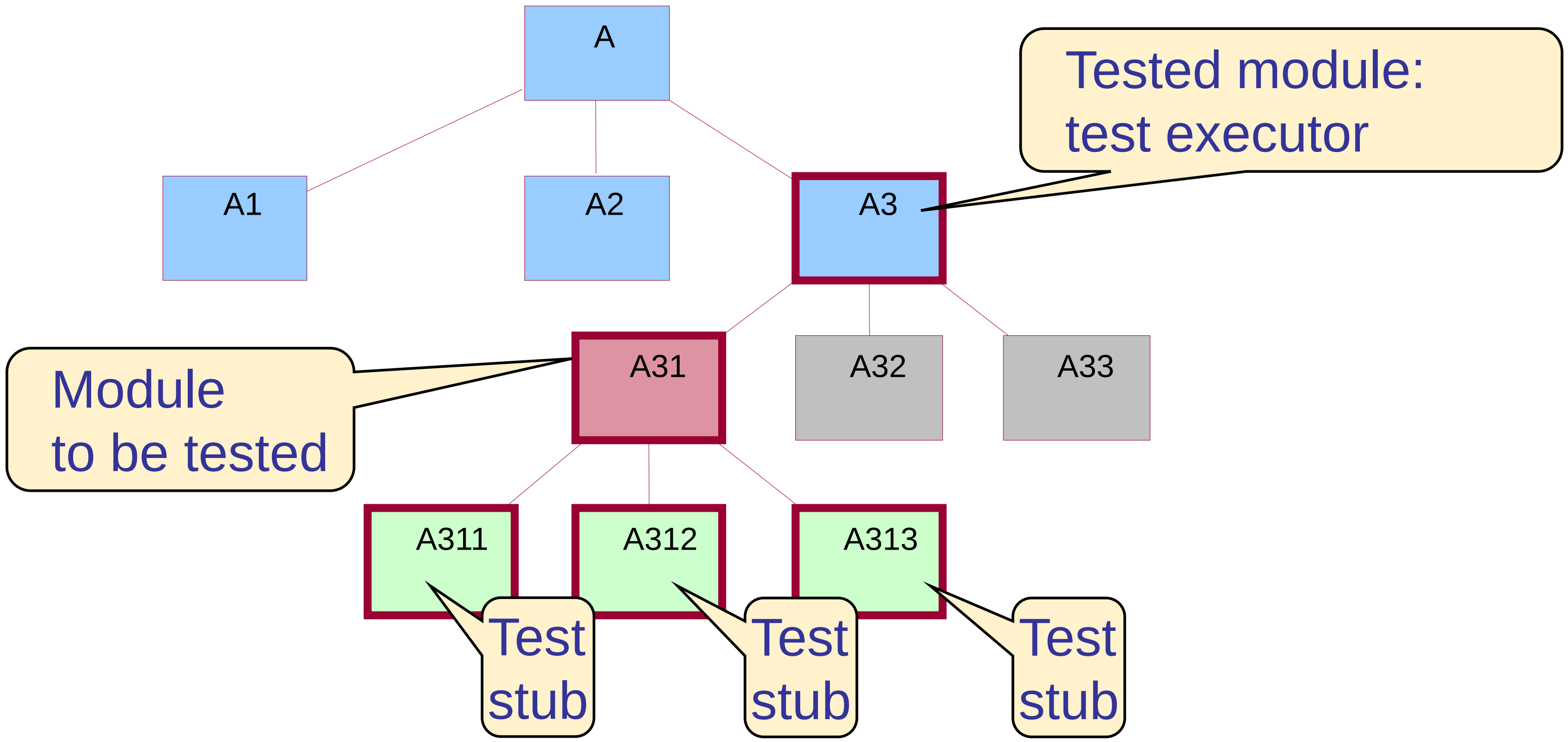
- In case of complex systems (supports debugging)
- Adapts to module hierarchy (calling levels)





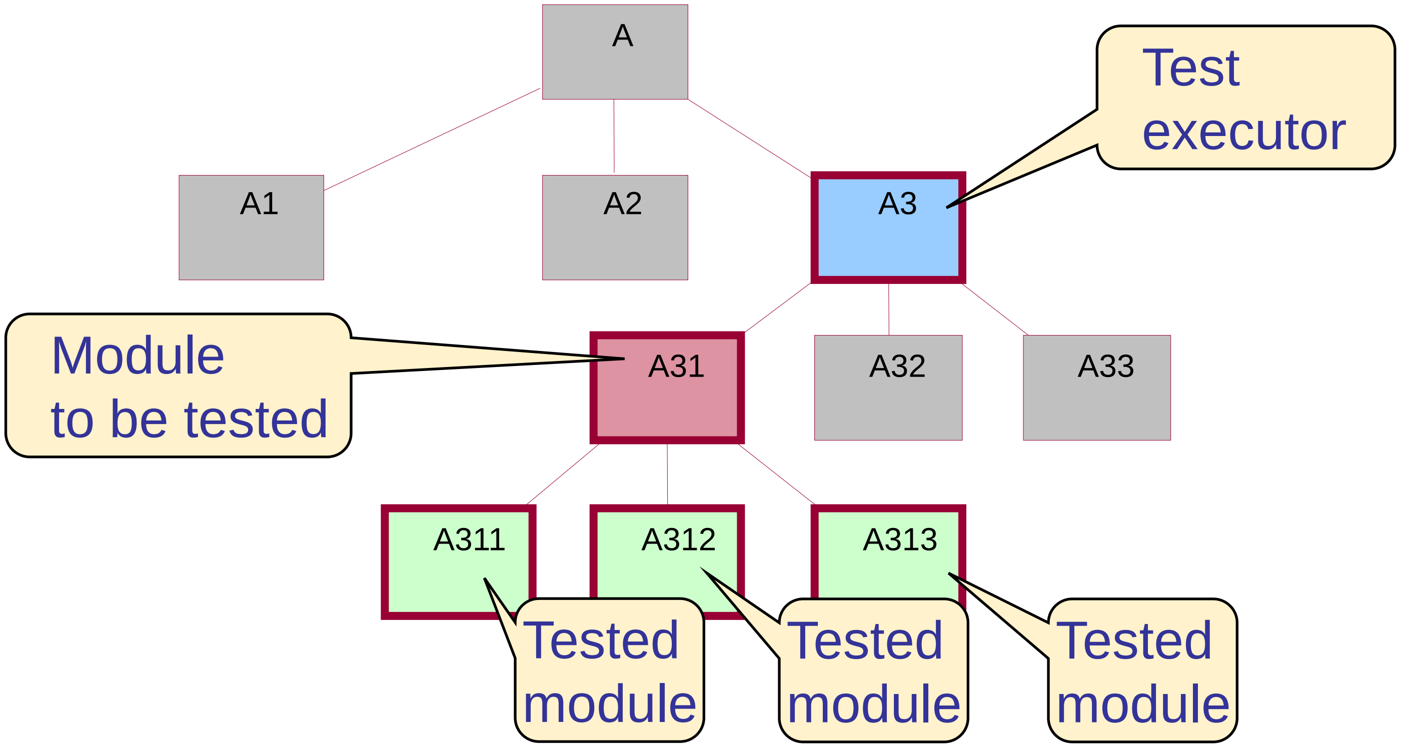
Top-down integration testing

- Modules are tested from the caller modules
- Stubs replace the lower-level modules that are called
- Requirement-oriented testing
- Module modification: modifies the testing of lower levels



Bottom-up integration testing

- Modules use already tested modules
- Test executor is needed
- Testing is performed in parallel with integration
- Module modification: modifies the testing of upper levels



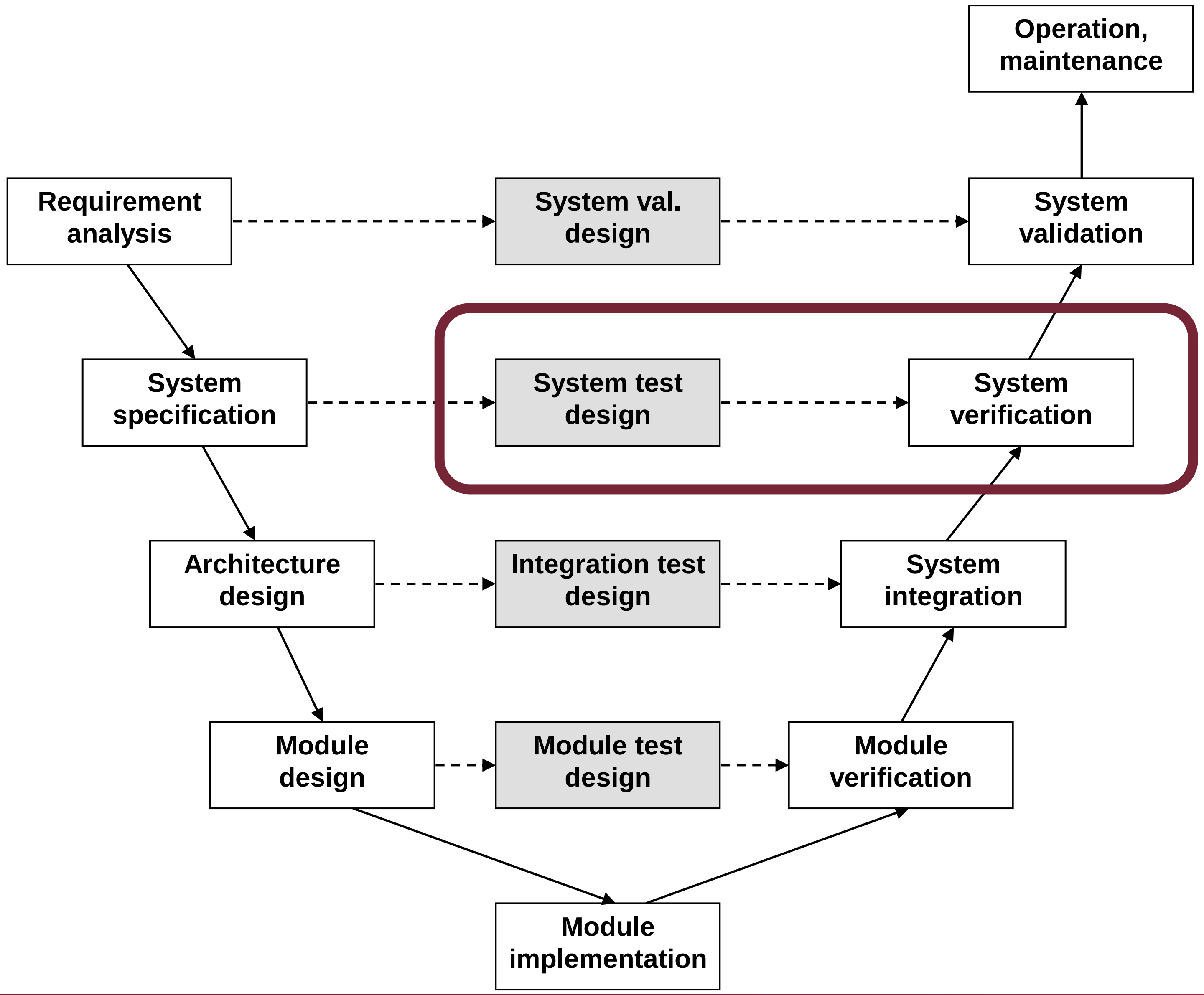
Top down vs. bottom up

- Top down
  + Requirement oriented
  + Working “skeleton” early
  - Harder to create stubs than drivers
  - Tests inputs are far from module to integrate
- Bottom up
  + Integration oriented, more constructive
  + Easier to control and observe the system
  - System is assembled only at the end

Integration with the runtime environment

- Motivation: It is hard to construct stubs for the runtime environment
  - Platform services, RT-OS, task scheduler, …
- Strategy:
  1. Top-down integration of the application modules to the level of the runtime environment
  2. Bottom-up testing of the runtime environment
     - Isolation testing of functions (if necessary)
     - „Big bang” testing with the lowest level of the application module hierarchy
  3. Integration of the application with the runtime environment, finishing top-down integration

Testing and test design in the V-model

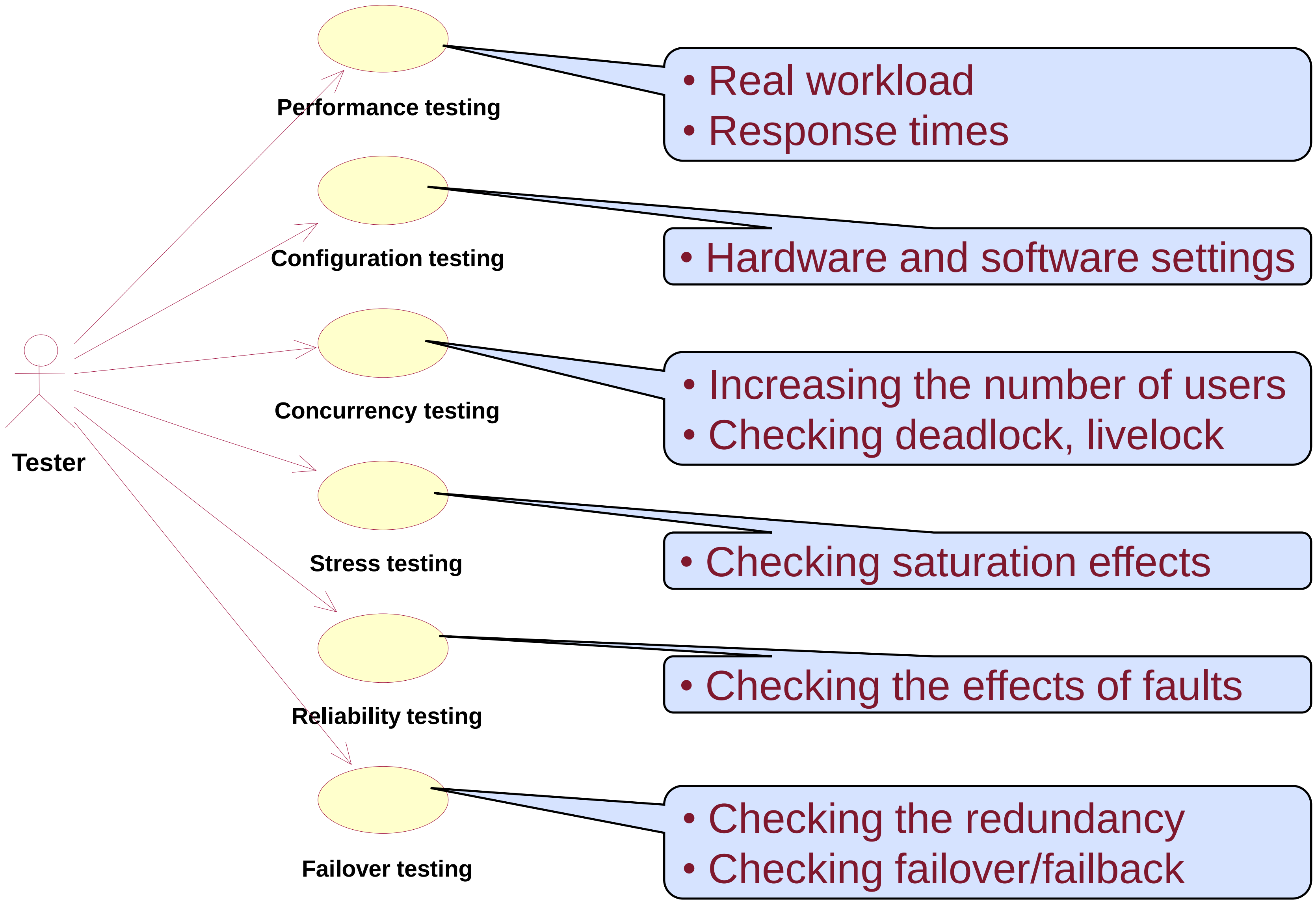


System testing

Testing on the basis of the system specification

- Characteristics:
  - Performed after hardware-software integration
  - Testing functional specification +
    testing extra-functional properties
- Testing aspects:
  - Data integrity
  - User profile (workload)
  - Checking application conditions of the system (resource usage, saturation)
  - Testing fault handling

Types of system tests



- Real workload
- Response times

- Hardware and software settings

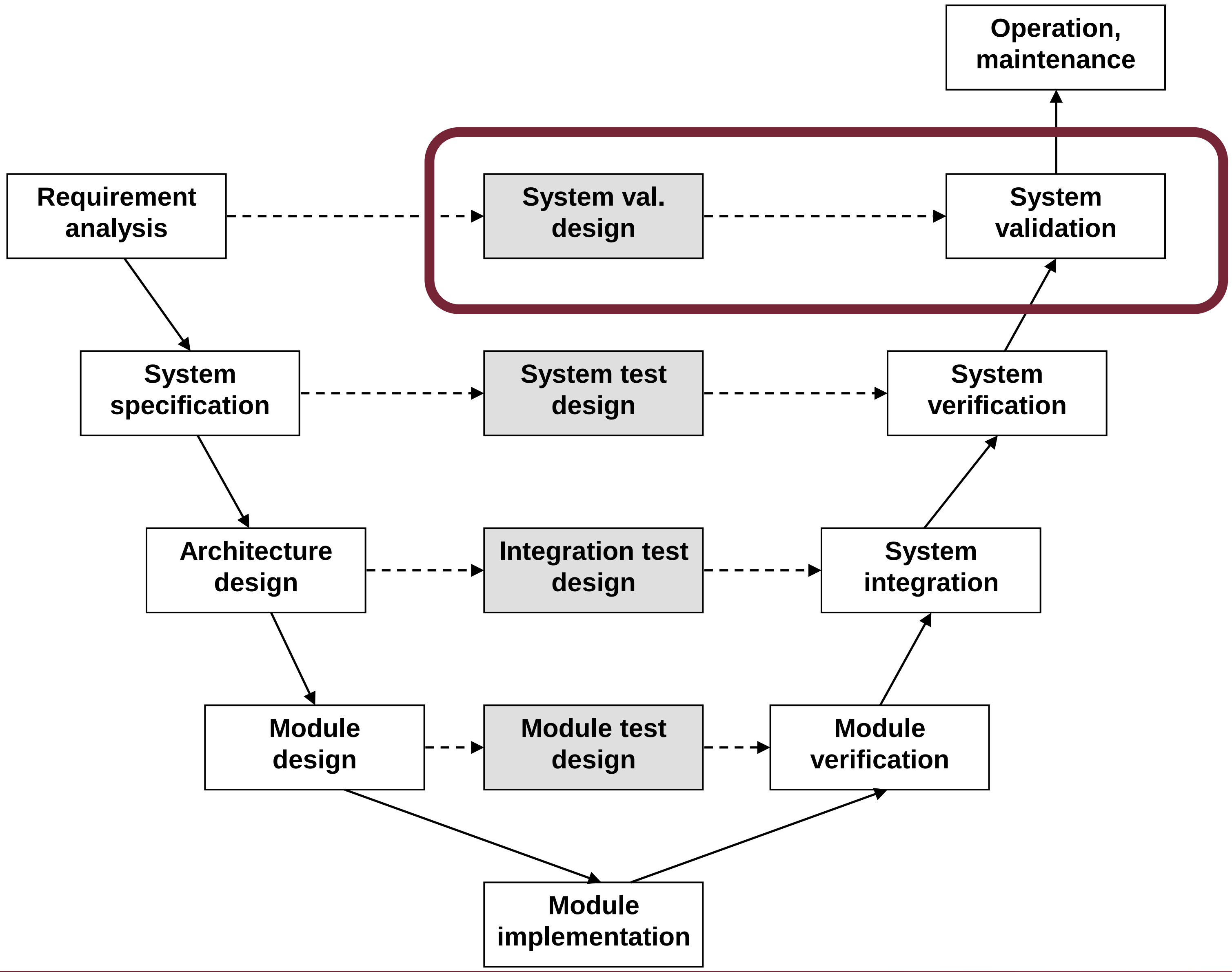
- Increasing the number of users
- Checking deadlock, livelock

- Checking saturation effects

- Checking the effects of faults

- Checking the redundancy
- Checking failover/failback

Testing and test design in the V-model



Validation testing

- Goal: Testing in real environment
  - User requirements are taken into account
  - Non-specified expectations come to light
  - Reaction to unexpected inputs/conditions is checked
  - Events of low probability may appear
- Timing aspects
  - Constraints and conditions of the real environment
  - Real-time testing and monitoring is needed
- Environment simulation
  - If given situations cannot be tested in a real environment (e.g., protection systems)
  - Simulators shall be validated somehow

EXTRA MATERIAL: UML 2 TESTING PROFILE (U2TP)

U2TP: UML 2 Testing Profile (OMG, 2004)

- Able to capture all needed information for functional black-box testing (specification of test artifacts)
  - Mapping rules to TTCN-3, JUnit
- Language (notation) and not a method (how to test)

Packages (concept groups):

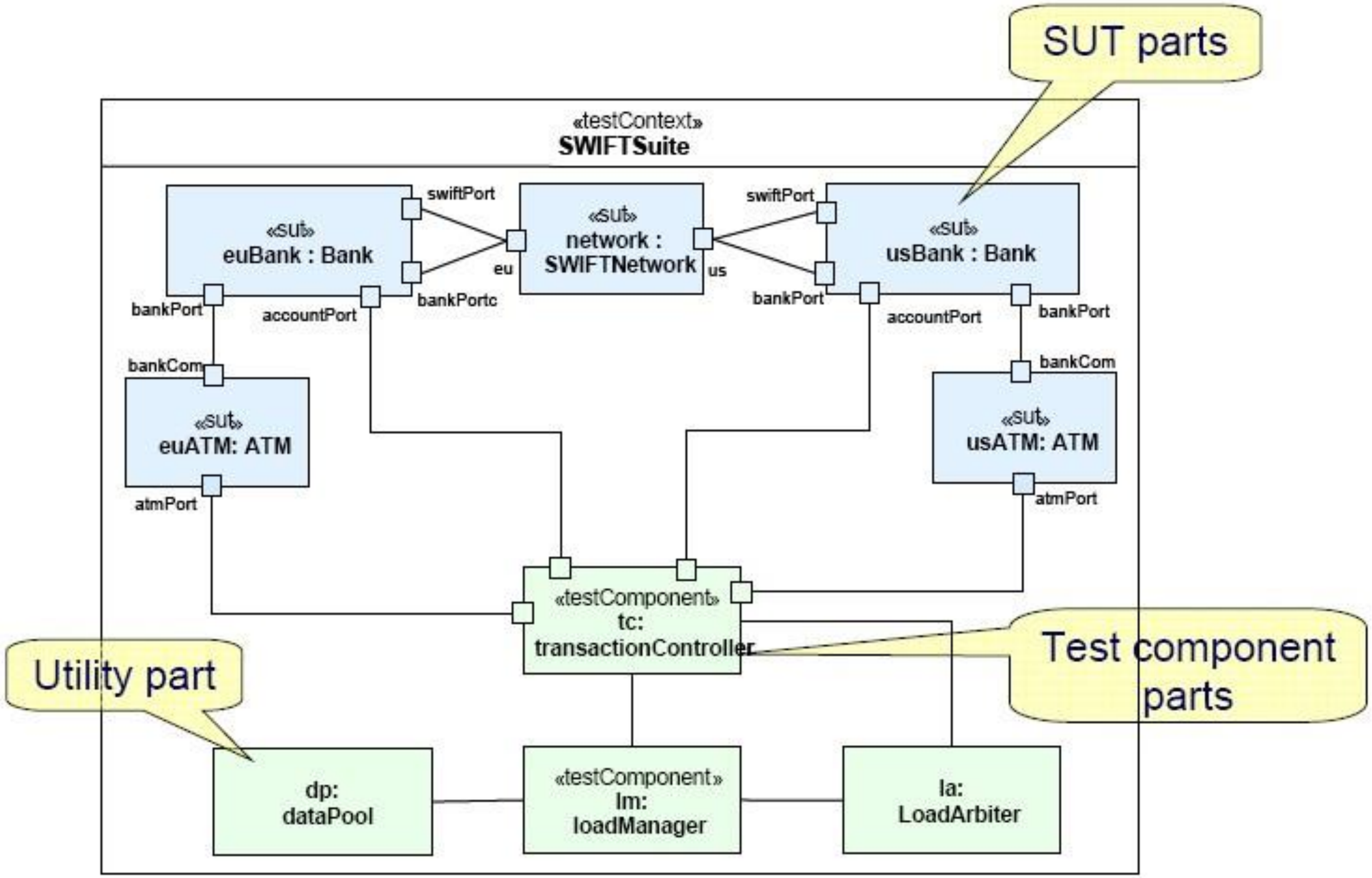
- Test Architecture
  - Elements and relationship involved in test
  - Importing the UML design model of the SUT
- Test Data
  - Structures and values to be processed in a test
- Test Behavior
  - Observations and activities during testing
- Time Concepts
  - Timer (start, stop, read, timeout), TimeZone (synchronized)

U2TP Test Architecture package

Identification of main components:

- SUT: System Under Test
  - Characterized by interfaces to control and observation
  - System, subsystem, component, class, object
- Test Component: part of the test system (e.g., simulator)
  - Realizes the behavior of a test case
    (Test Stimulus, Test Observation, Validation Action, Log Action)
- Test Context: collaboration of test architecture elements
  - Initial test configuration (test components)
  - Test control (decision on execution, e.g., if a test fails)
- Scheduler: controls the execution of test components
  - Creation and destruction of test components
- Arbiter: calculation of final test results
  - E.g., threshold on the basis of test component verdicts

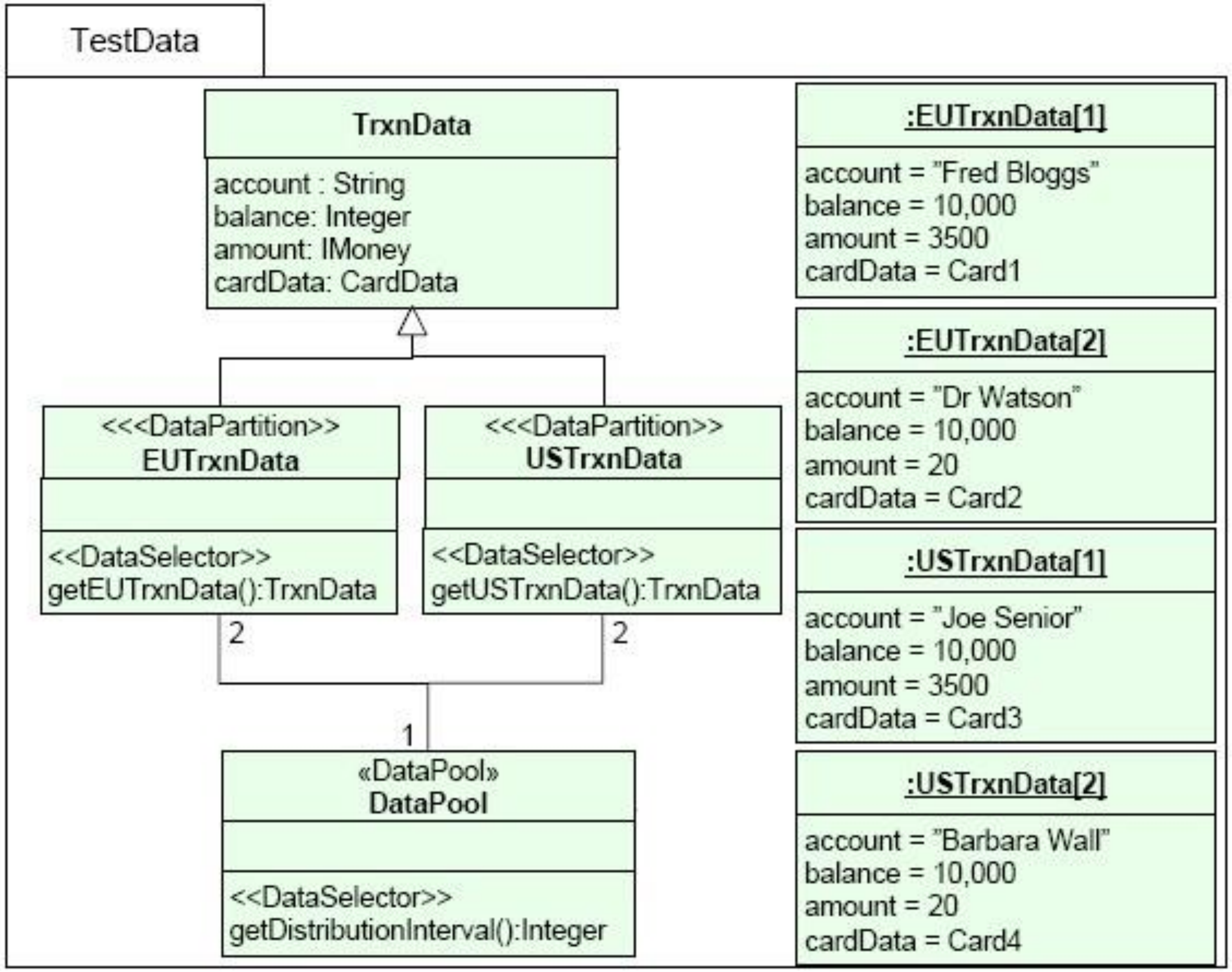
U2TP Test Architecture example



U2TP Test Data package

- Identification of types and values for test (sent and received data)
  - Wildcards (* or ?)
  - Test Parameter
    - Stimulus and observation
  - Argument
    - Concrete physical value
  - Data Partition: Equivalence class for a given type
    - Class of physical values, e.g., valid names
  - Data Selector: Retrieving data out of a data pool
    - Operating on contained values or value sets
  - Templates

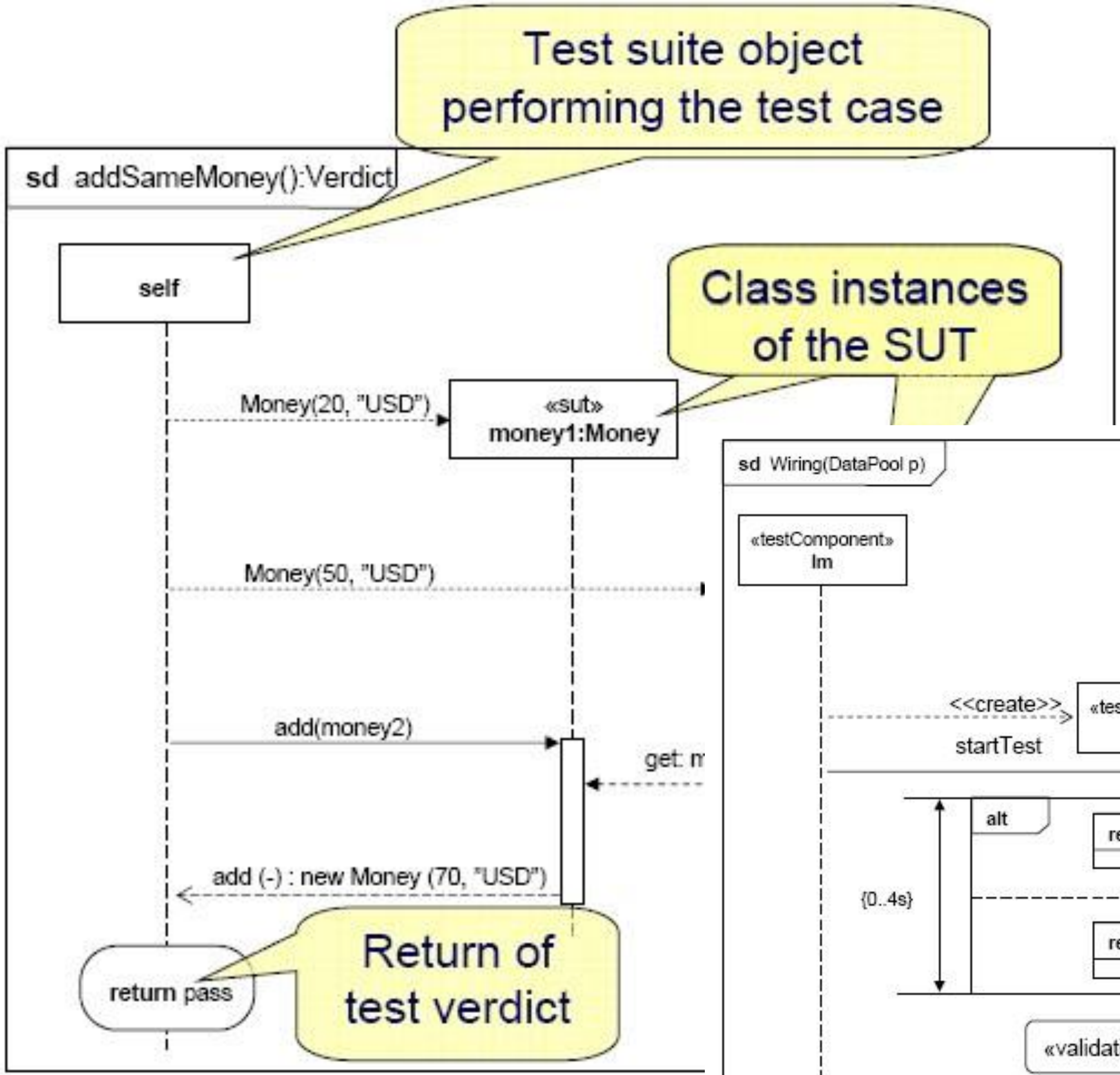
U2TP Test Data example

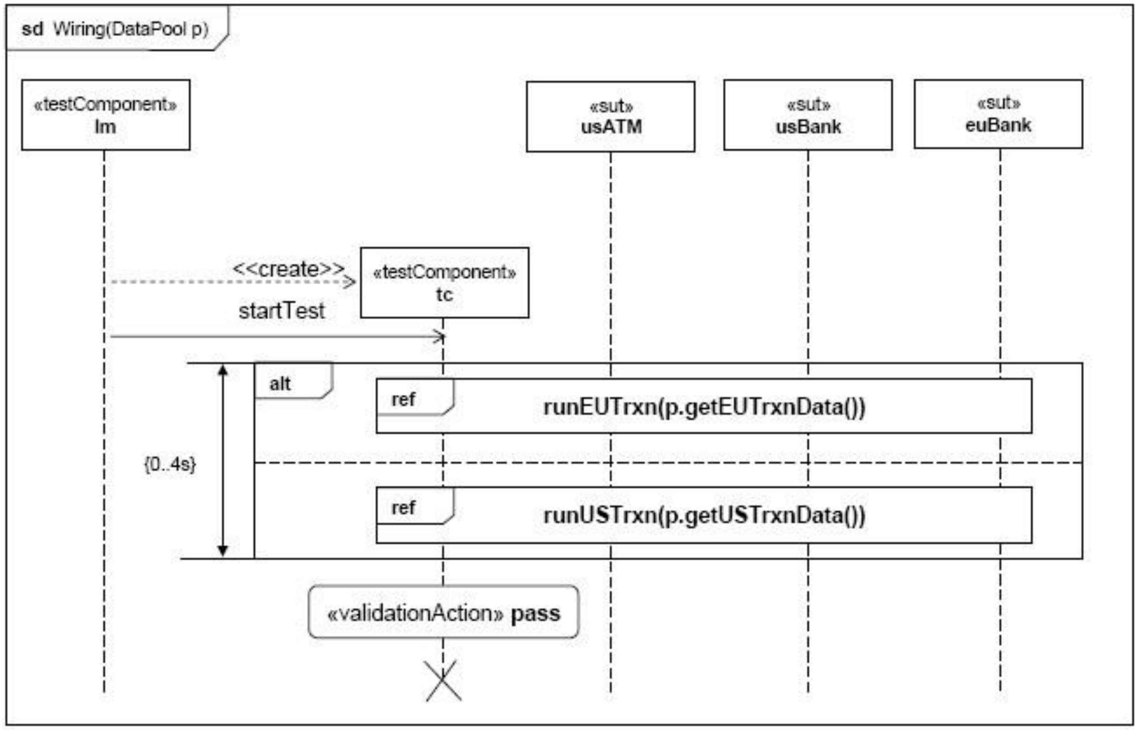


U2TP Test Behavior package

- Specification of default/expected behavior
- Identification of behavioral elements:
  - Test Stimulus: test data sent to SUT
  - Test Observation: reactions from the SUT
  - Verdict: pass, fail, error, inconclusive values
  - Actions: Validation Action (inform Arbiter), Log Action
- Test Case: Specifies one case to test the SUT
  - Test Objective: named element
  - Test Trace: result of test execution
    - Messages exchanged
  - Verdict

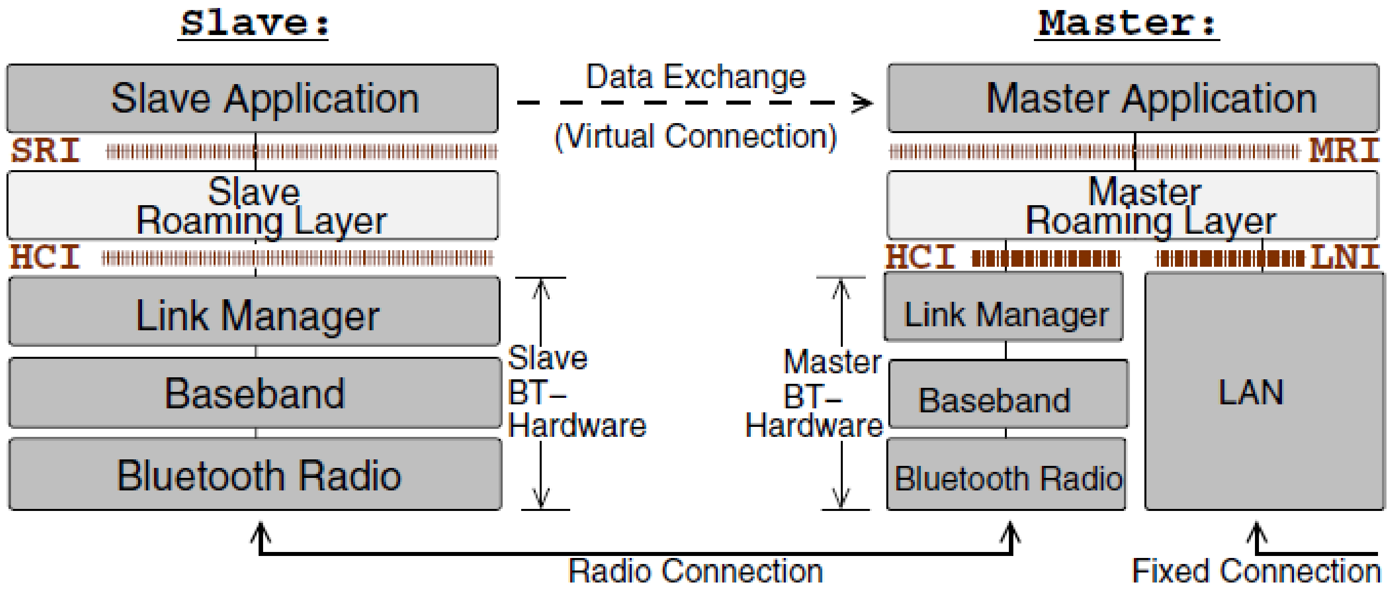
U2TP Test Behavior example


Test suite object performing the test case
sd addSameMoney():Verdict
self
Class instances of the SUT
Money(20, "USD")
«sut» money1:Money
Money(50, "USD")
add(money2)
get: m
add (-) : new Money (70, "USD")
return pass
Return of test verdict



Example: BlueTooth roaming

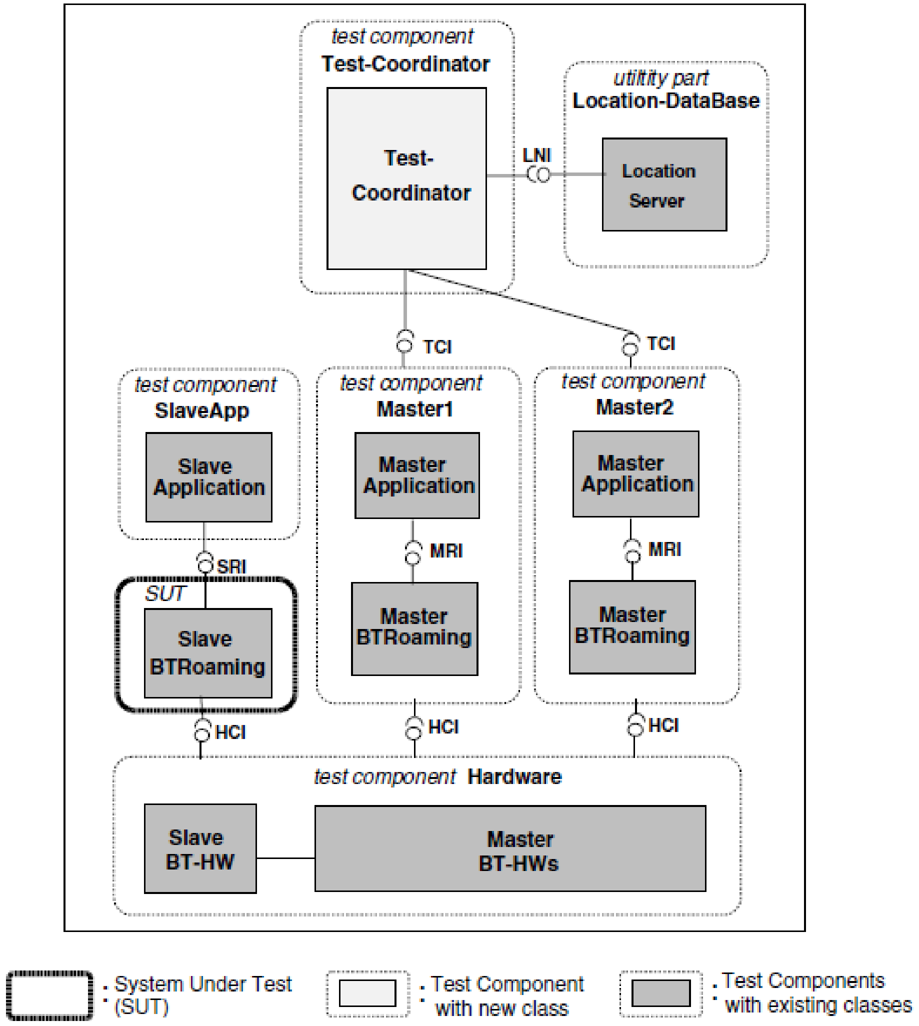
System under test:



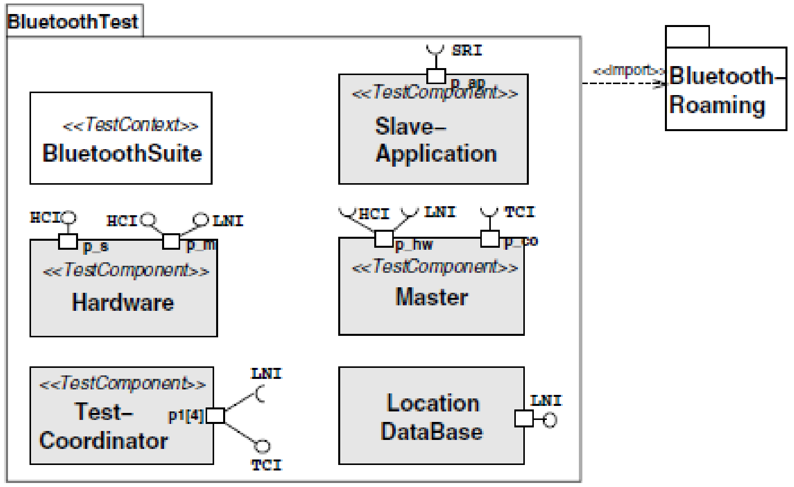
Test objective:

- Slave Roaming Layer functionality
  - Monitoring link quality
  - Connecting to a different master

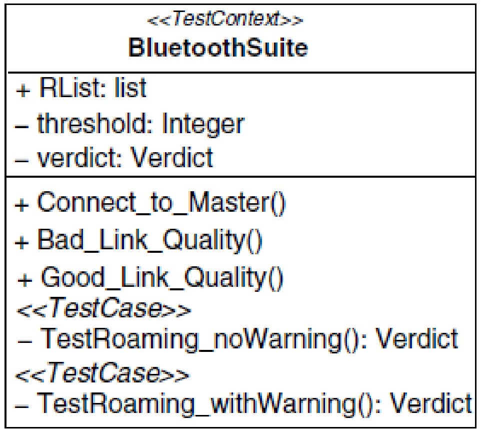
Example: Components



Overview



Test package



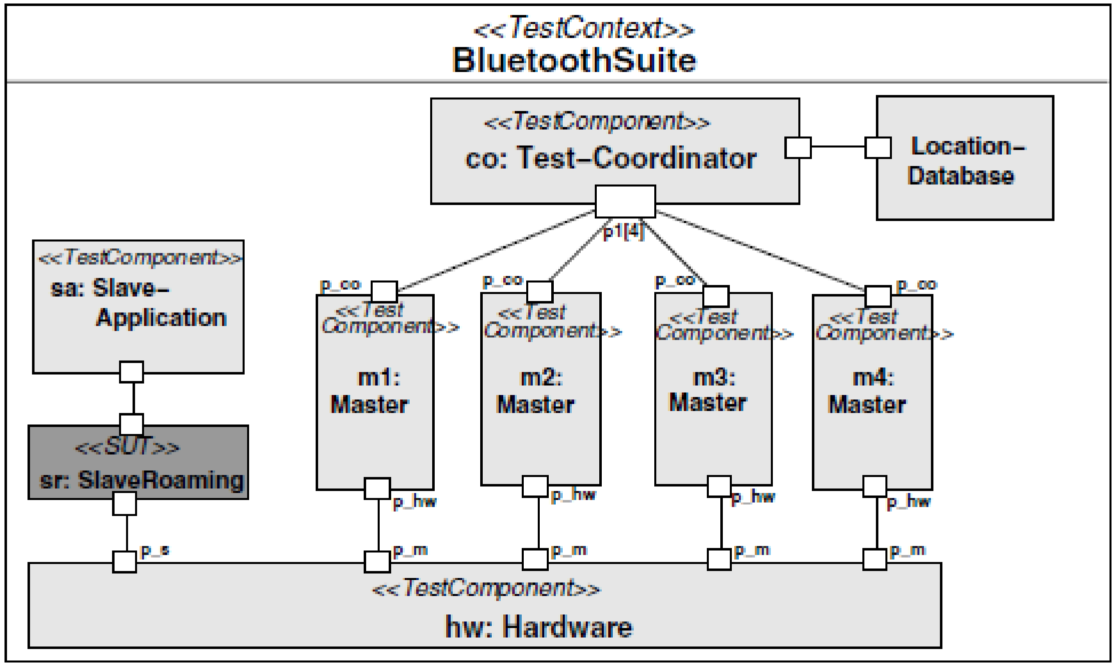
<<TestContext>>
BluetoothSuite

+ RList: list
– threshold: Integer
– verdict: Verdict

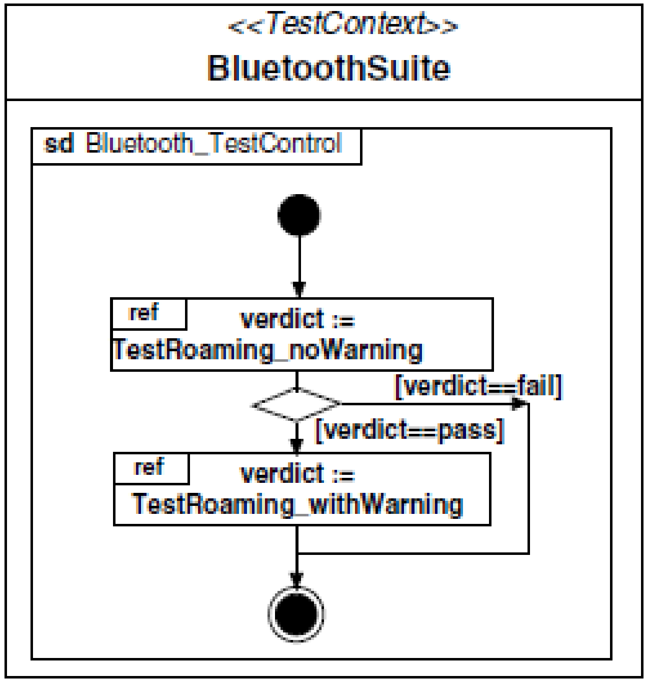
+ Connect_to_Master()
+ Bad_Link_Quality()
+ Good_Link_Quality()
<<TestCase>>
– TestRoaming_noWarning(): Verdict
<<TestCase>>
– TestRoaming_withWarning(): Verdict

Test context

Example: Test configuration and control



Test configuration

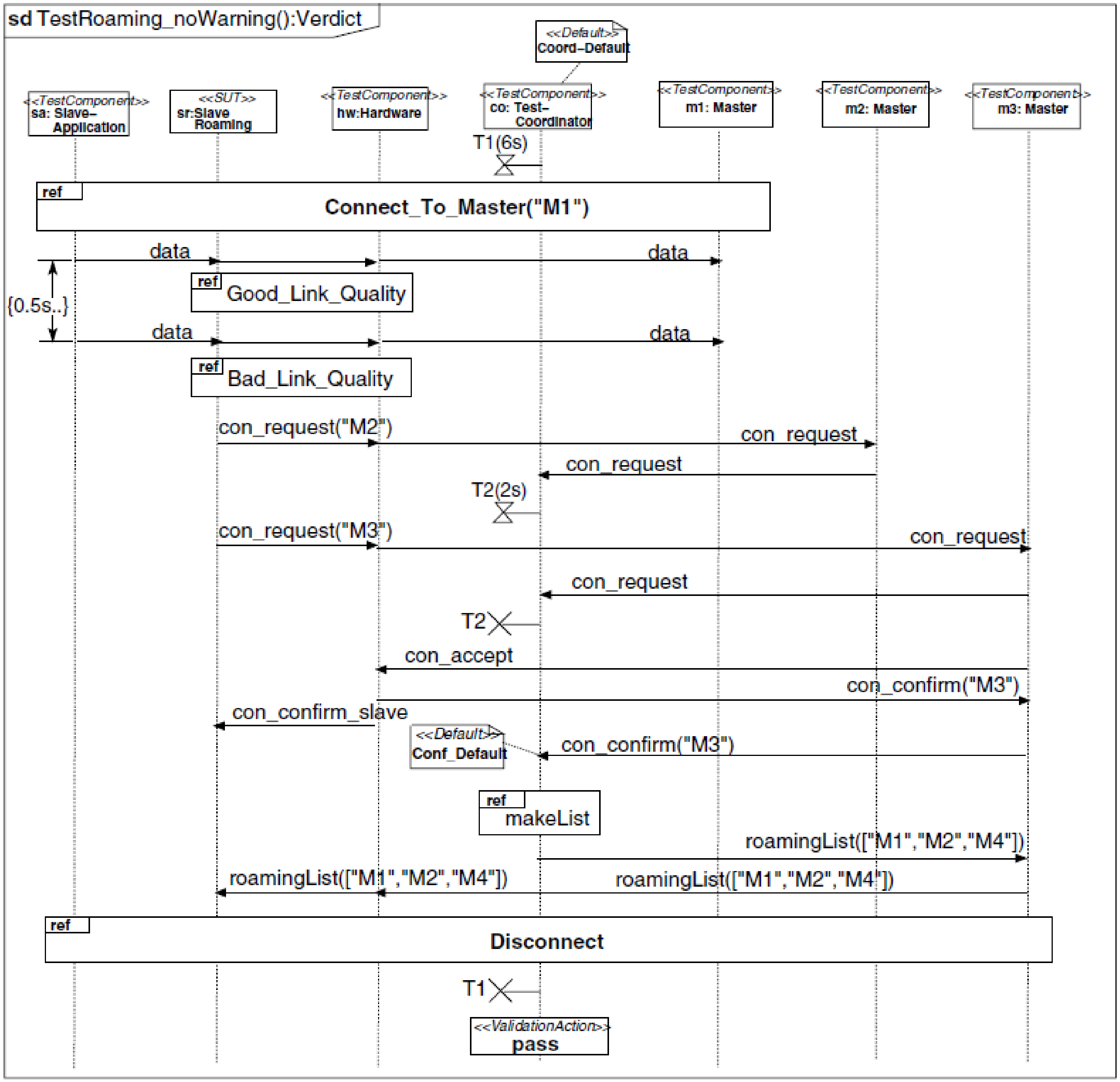


Test control

Test scenario

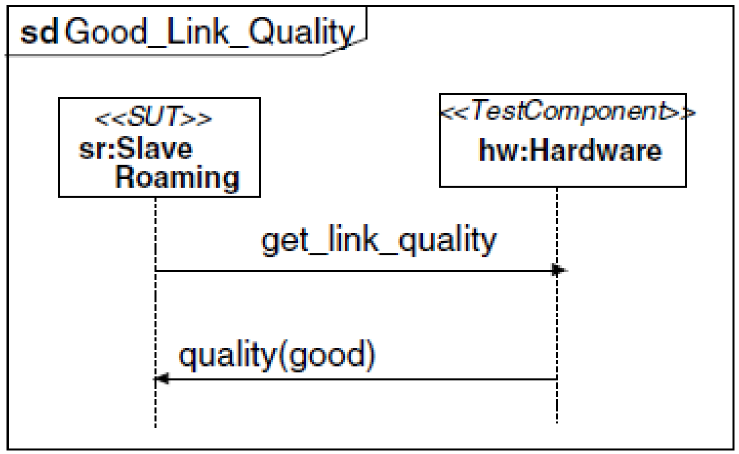
Test case implementa-tion
(see Blue-ToothSuite)

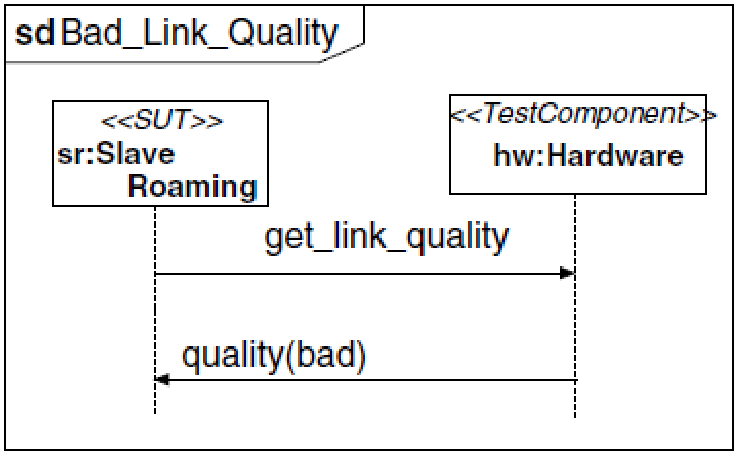
- References
- Timers
- Defaults



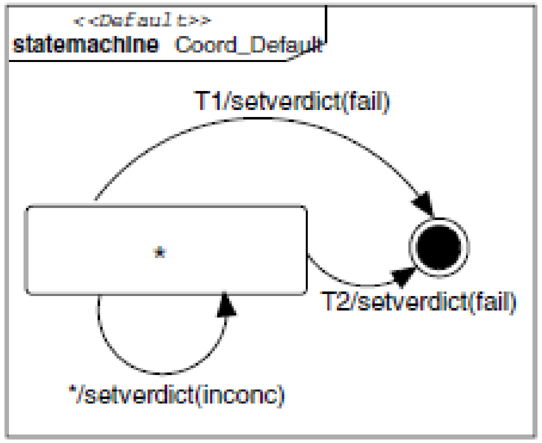
sd TestRoaming_noWarning():Verdict
<<Default>> Coord–Default
<<TestComponent>> sa: Slave–Application
<<SUT>> sr:Slave Roaming
<<TestComponent>> hw:Hardware
<<TestComponent>> co: Test–Coordinator
<<TestComponent>> m1: Master
<<TestComponent>> m2: Master
<<TestComponent>> m3: Master
T1(6s)
ref Connect_To_Master("M1")
data
data
{0.5s..}
ref Good_Link_Quality
data
data
ref Bad_Link_Quality
con_request("M2")
con_request
con_request
T2(2s)
con_request("M3")
con_request
con_request
T2
con_accept
con_confirm("M3")
con_confirm_slave
<<Default>> Conf_Default
con_confirm("M3")
ref makeList
roamingList(["M1","M2","M4"])
roamingList(["M1","M2","M4"])
roamingList(["M1","M2","M4"])
ref Disconnect
T1
<<ValidationAction>> pass

Test scenarios (details)





Sequence diagrams



Default behaviours specified to catch the observations that lead to verdicts

- Here: Processing timer events