Integration testing, system testing, validation testing

Istvan Majzik
majzik@mit.bme.hu

Budapest University of Technology and Economics
Dept. of Measurement and Information Systems
Typical development steps and V&V tasks

- **Requirement analysis**
  - Checking completeness, consistency, feasibility, verifiability
  - Assuring traceability

- **System specification**
  - Trade-off analysis, interface analysis, fault effects analysis
  - Model based quantitative evaluation

- **Architecture design**
  - Formal verification by (temporal logic based) model checking
  - Equivalence checking

- **Module design**
  - Source code analysis
  - Software model checking with abstraction
  - Proof of program correctness by theorem proving
  - Module testing (unit testing)

- **System integration**
  - Integration testing
  - System testing
  - Validation testing

- **System delivery**

- **Operation, maintenance**
Integration testing
Testing and test design in the V-model

- Requirement analysis
- System specification
- Architecture design
- Module design
- Module implementation
- Module test design
- Integration test design
- System test design
- System integration
- System verification
- System validation
- System integration
- Operation, maintenance
Software integration testing

- Software architecture design
- Software construction design
- Software integration test plan
- Software quality assurance plan

Software integration testing

Software integration test report
Goals, methods and approaches

- **Goal and motivation:**
  - Testing the **interactions** of modules
  - The system-level interaction of modules may be incorrect despite the fact that all modules are correct

- **Methods: Testing interaction scenarios**
  - Sometimes the scenarios are part of the specification
  - **Systematic** testing: Covering all / representative scenarios
  - The concept of equivalence partitions and boundary values applied for interactions (scenario / input data level)

- **Approaches**
  - “Big bang” testing: integration of all modules before testing
  - Incremental testing: stepwise integration+testing
“Big bang” testing

- Integration of all modules then 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

Debugging is difficult!
Incremental integration and testing

- Applied in case of complex systems (to support debugging)
- Adapted to module hierarchy (calling levels)
Module testing: Isolation of modules

- Modules are tested in isolation separately
- Test drivers and test doubles (used for substitution w.r.t dependencies)
- Dependency: Anything collaborating with the SUT (does not belong to it)
General problems: Handling dependencies

- Several approaches for substituting dependencies
  - See ”isolation frameworks” (e.g., Mockito, JMock, ...)
  - Generic name of substitute: **Test double**

- **Stub**
  - Predefined replies to calls
  - Checks the **state** of the SUT

- **Mock**
  - Expected and checked behavior
  - Check the **interactions** of the SUT (number of calls, with parameters ...)

- **Dummy**
  - Not used object (filler)

- **Fake**
  - Working, but not the real one
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 testing

- **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
Functional integration

- **Motivation:**
  - There are several system-level functions
  - Priorities among these regarding criticality → prioritizing testing

- **Basic idea:**
  - Integration on the basis of system functions
  - Each function is integrated and tested, top-down

- **Specific case of top-down integration testing**
  - Requirement oriented (w.r.t. the given function)
  - Test doubles (stubs) are needed
  - Top level is tested with more and more functions
  - Module modification: modifies the testing of lower levels
Integration with the runtime environment

- **Motivation:**
  - It is hard to construct stubs for the runtime environment
  - See e.g., 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)
     - Testing with the lowest level of the application module hierarchy
  3. **Integration** of the application with the runtime environment, finishing top-down integration
Coverage metrics: State based approach

- **Goal:** Coverage of interactions among modules
  - Basic case: Coverage of interface functions (by calls)

- **State based coverage metrics:**
  - Coverage of interface functions for all relevant states (or state partitions) of the caller and the called module
  - Extension: With all triggers and conditions for the call

- opB2() call can be served in two states of comp. B
Coverage metrics: Data flow based approach

- Data flow based metrics:
  - Coverage extended for **coupling paths** (among function calls and returns)
    - Applying def-use labels
  - Coverage metrics:
    - All-coupling-defs
    - all-coupling-uses
    - all-coupling-paths

- Testing robustness of interfaces
  - **Extreme** and boundary values of call parameters
  - **Mutating call scenarios** (omission, duplication, change of ordering, extreme parameters etc.)
System testing
Testing and test design in the V-model

- Requirement analysis
- System specification
- Architecture design
- Module design
- Module implementation
- Module test design
- Integration test design
- System test design
- System integration
- System verification
- System validation
- System design
- Requirement analysis

Operation, maintenance
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
  - ... (depending on the system specification)
Types of system tests (examples)

- Performance testing
  - Real workload
  - Response times

- Configuration testing
  - Hardware and software settings

- Concurrency testing
  - Increasing the number of users
  - Checking deadlock, livelock

- Stress testing
  - Checking saturation effects

- Reliability testing
  - Checking the effects of faults

- Failover testing
  - Checking the use of redundancy by failover/failback
Validation testing
Testing and test design in the V-model

Requirement analysis

System specification

Architecture design

Module design

Module implementation

System test design

Integration test design

Module test design

Module verification

System design

System integration

System verification

System validation

System test design

Operation, maintenance

System integration

System verification
Validation testing

- **Goal: Testing in real environment**
  - User requirements and *expectations* are taken into account
  - Non-specified expectations may come up
  - 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
Summary: Testing levels

1. Module (unit) testing
   - Isolation testing

2. Integration testing
   - ”Big bang” testing
   - Top-down testing
   - Bottom-up testing
   - Functional integration
   - Integration with the runtime environment

3. System testing
   - Testing the integrated system

4. Validation testing
   - Testing user expectations in the real environment
   - Environment simulation
Design and documentation of testing

Standard for **Software Test Documentation**

Test planning:

- **Test Plan**: What is tested, by whom, how, in what time frame, to what quality
  - **SPACEDIRT**: Scope, People, Approach, Criteria, Environment, Deliverables, Incidentals, Risks, Tasks

Test specification:

- **Test Design Specifications**: Test conditions, expected outcome, what is a successful test
- **Test Case Specifications**: The specific test data (test suites)
- **Test Procedure Specifications**: What kind of physical set-up is required, how the tester runs the test, what steps need to be followed

Test reporting:

- **Test Item Transmittal Report**: When specific tested items are passed from one stage of testing to another
- **Test Log**: What tests cases were run, by whom, in what order, and whether individual tests were passed or failed
- **Test Incident Report**: Details of test failure (when, why)
- **Test Summary Report**: Assessment about the quality of the system
Standard test documentation (IEEE 829:2008)

Standard for *Software and System Test* Documentation

Test planning:
- **Master Test Plan (MTP):** Overall test planning for multiple levels
- **Level Test Plan (LTP):** Scope, approach, resources, and schedule of the testing

Test design:
- **Level Test Design (LTD):** Test cases, the expected results, the test pass criteria
- **Level Test Case (LTC):** Specifying the test data for use in running the test cases
- **Level Test Procedure (LTPr):** How to run each test (preconditions and the steps)

Test reporting:
- **Level Test Log (LTL):** Record of relevant details about the execution
- **Anomaly Report (AR):** Events that occur during testing and require investigation
- **Level Interim Test Status Report (LITSR):** Summarize/evaluate interim results
- **Level Test Report (LTR):** Summarize/evaluate the results after test execution has finished for the specific test level
- **Master Test Report (MTR):** Summarize/evaluate the results of the levels
Able to capture all needed information for functional black-box testing (specification of test artifacts)
  o Mapping rules to TTCN-3, JUnit

Language (notation) and not a method (how to test)

Packages (concept groups):

Test Architecture
  o Elements and relationship involved in test
  o Importing the UML design model of the SUT

Test Data
  o Structures and values to be processed in a test

Test Behavior
  o Observations and activities during testing

Time Concepts
  o 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
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

Class instances of the SUT

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

Test context
Example: Test configuration and control

Test configuration

Test control
Example: Test scenario

Test case implementation (see Blue-ToothSuite)

- References
- Timers
- Defaults
Test scenarios (details)

Sequence diagrams

Default behaviors specified to catch the observations that lead to verdicts
• Here: Processing timer events