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 (component) testing

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

- **System delivery**
  - Validation testing

- **Operation, maintenance**
Testing and test design in the V-model
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 validation
- System verification
- System integration
- System design
- System integration
- System verification
- System validation
- 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:
  o Testing the interactions of modules
  o The system-level interaction of modules may be incorrect despite the fact that all modules are correct

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

▪ Approaches
  o “Big bang” testing: integration of all modules before testing
  o Incremental testing: stepwise integration + testing
“Big bang” testing

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

Diagram:

Tester1

- A
- B
- C
- D

Error in this component: Debugging is difficult!
Incremental integration and testing

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

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

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

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

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

- **Dummy**
  - Not used component (just “filler”)

- **Fake**
  - Working component, 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” is available and tested early
  - - Harder to create stubs than drivers

- **Bottom up**
  - + Integration oriented, more constructive
  - + Easier to control and observe the subsystems
  - - System is assembled only at the end
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 in a top-down way → 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 down 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
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 transitions) of the caller and the called module
- Extension: With all triggers and conditions for the call

opB2() call can be served by two transitions 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 design
- System validation
- System verification
- System integration
- System test design
- System integration
- System validation
- System verification
- System validation
- System integration
- System test design
- System design
- Module test design
- Integration test design
- Architecture design
- System specification
- Requirement analysis
- Operation, maintenance
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:**
  - User workload (according to user profile)
  - Checking **application conditions** of the system (resource usage, saturation)
  - Testing **fault handling**
  - Data integrity
  - ... (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
Software validation

- System requirements specification
- Software requirements specification
- Software requirements test specification
- Software validation plan
- Software validation test report
- Software validation report
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
   o Isolation testing

2. Integration testing
   o ("Big bang" testing)
   o Top-down testing
   o Bottom-up testing
   o Functional integration
   o Integration with the runtime environment

3. System testing
   o Testing the integrated system

4. Validation testing
   o Testing user expectations in the real environment
   o 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 Plans (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
U2TP: UML 2 Testing Profile (OMG, 2004)

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

Packages (concept groups):

- **Test Architecture**
  - Components and relationship involved in test
  - Importing the UML design model of the SUT
- **Test Data**
  - Data structures and values to be processed in a test
- **Test Behavior**
  - Activities and observations 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
  - Can be: System, subsystem, component, object

- **Test Component**: Part of the test system (e.g., a 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 (e.g., sent and received data)
  - Wildcards (* or ?) can be used
  - 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

 TestData

 TrxnData
 account : String
 balance: Integer
 amount: IMoney
 cardData: CardData

<<DataPartition>>
EUTxnData
<<DataSelector>>
getEUTxnData(): TrxnData

<<DataPartition>>
USTxnData
<<DataSelector>>
getUSTxnData(): TrxnData

: EUTxnData[1]
account = "Fred Bloggs"
balance = 10,000
amount = 3500
cardData = Card1

: EUTxnData[2]
account = "Dr Watson"
balance = 10,000
amount = 20
cardData = Card2

: USTxnData[1]
account = "Joe Senior"
balance = 10,000
amount = 3500
cardData = Card3

: USTxnData[2]
account = "Barbara Wall"
balance = 10,000
amount = 20
cardData = Card4
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, or inconclusive
  - **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:

![Diagram showing slave and master components with various layers and connections]

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