Testing II

Lecture 8a

Software Engineering
TDDC88/TDDC93
autumn 2013

Kristian Sandahl
Department of Computer and Information Science
Linköping University, Sweden
Kristian.sandahl@ida.liu.se
A Software Life-cycle Model
Which part will we talk about today?

- **Part I**: Integration Testing
- **Part II**: System Testing
- **Part III**: Acceptance Testing

**Validate Requirements, Verify Specification**

- **Requirements**
- **System Design**: (Architecture, High-level Design)
- **Module Design**: (Program Design, Detailed Design)

**Verify System Design**

- **System Testing**: (Integration testing of modules)

**Verify Module Design**

- **Module Testing**: (Integration testing of units)

**Verify Implementation**

- **Implementation of Units**: (classes, procedures, functions)
- **Unit testing**

**Acceptance Test**: (Release testing)

- **Maintenance**

Project Management, Software Quality Assurance (SQA), Supporting Tools, Education
Lower levels (from last lecture)

- Unit testing performed by the programmers – small distance design – test
  - Module testing can be performed by automated builds – regression testing a la JUnit feasible.
  - Module testing would benefit from independent testers.
  - Test-Driven Development fully feasible.
  - XP: ALL test-cases are written before coding – based on User Stories
  - Long learning curve
  - Use for some parts in student project
  - Always think of testing early!
Agenda - What will you learn today?

Part I
Integration Testing

Part II
System Testing

Part III
Acceptance Testing
Part I
Integration testing
Part I
Integration Testing

Part II
System Testing

Part III
Acceptance Testing
Integration Testing strategies

1. Big-bang
2. Bottom-up
3. Top-down
4. Sandwich
Three level functional decomposition tree

Part I
Integration Testing

Part II
System Testing

Part III
Acceptance Testing

Kristian.Sandahl@liu.se
Big-Bang testing

Part I
Integration Testing

Part II
System Testing

Part III
Acceptance Testing

Environment:
A, B, C, D, E, F, G, H

Unit test A
Unit test B
Unit test H

System-wide test
- A pretend module that requires a sub-system and passes a test case to it

**Diagram:**
- **Driver**
  - Setup driver
  - SUT(x)
  - Verification

**SUT**

**System Under Test**

**Black-box view**

**Part I**
Integration Testing

**Part II**
System Testing

**Part III**
Acceptance Testing
Bottom-up testing

Part I
Integration Testing

Part II
System Testing

Part III
Acceptance Testing

Kristian.Sandahl@liu.se
Is bottom-up smart?

- If the basic functions are complicated, error-prone or has development risks
  - If bottom-up development strategy is used
  - If there are strict performance or real-time requirements

Problems:
- Lower level functions are often off-the-shelf or trivial
- Complicated User Interface testing is postponed
- End-user feed-back postponed
- Effort to write drivers.
A program or a method that simulates the input-output functionality of a missing sub-system by answering to the decomposition sequence of the calling sub-system and returning back simulated or "canned" data.
Top-down testing

A
B C D
E F G H

A, B, C, D

A, B, E, F, C, D, G, H
Is top-down smart?

- Test cases are defined for functional requirements of the system
  - Defects in general design can be found early
  - Works well with many incremental development methods
  - No need for drivers

Problems:
- Technical details postponed, potential show-stoppers
- Many stubs are required
- Stubs with many conditions are hard to write
Sandwich testing

Part I
Integration Testing

Part II
System Testing

Part III
Acceptance Testing

Target level

A, B, C, D
E, F, B
G, H, D
A, B, E, F, C, D, G, H
Part II
System Testing
Part I
Integration Testing

Part II
System Testing

Part III
Acceptance Testing

Component code

Unit test

Tested components

Design Specification

Integration test

Integrated modules

Component code

Unit test

Tested components
System functional requirements

- Integrated modules
  - Function test

Functioning systems

- Performance test

Accepted system

- Installation test

Verified validated software

Customer requirements spec.

- Acceptance test

User environment

- System In Use!

**Part I**
Integration Testing

**Part II**
System Testing

**Part III**
Acceptance Testing
Function testing/Thread testing

(testing one function at a time)

functional requirements

A function test checks that the integrated system performs its function as specified in the requirement

- Guidelines
  - use a test team independent of the designers and programmers
  - know the expected actions and output
  - test both valid and invalid input
  - never modify the system just to make testing easier
  - have stopping criteria
Performance Testing
nonfunctional requirements

- Stress tests
- Volume tests
- Configuration tests
- Compatibility tests
- Regression tests
- Security tests
- Timing tests

- Environment tests
- Quality tests
- Recovery tests
- Maintenance tests
- Documentation tests
- Human factors tests / usability tests
Software reliability engineering

- Define target failure intensity
- Develop operational profile
- Plan tests
- Execute test
- Apply data to decisions
Part III
Acceptance Testing
Acceptance Testing

customers, users need

- **Benchmark test**: a set of special test cases

- **Pilot test**: everyday working
  - Alpha test: at the developer’s site, controlled environment
  - Beta test: at one or more customer site.

- **Parallel test**: new system in parallel with previous one
Acceptance test at developers site
→ installation test at users site, otherwise installation test might not be needed!
Termination Problem
How decide when to stop testing

- The main problem for managers!

- Termination takes place when
  - resources (time & budget) are over
  - found the seeded faults
  - some coverage is reached
Control-flow based coverage

Statement coverage

All statements executed

Part I
Integration Testing

Part II
System Testing

Part III
Acceptance Testing

Kristian.Sandahl@liu.se

27
Control-flow based coverage

Branch coverage

All decision branches tried

Part I
Integration Testing

Part II
System Testing

Part III
Acceptance Testing
Control-flow based coverage

Full path coverage

All possible paths executed

Part I
Integration Testing

Part II
System Testing

Part III
Acceptance Testing

Kristian.Sandahl@liu.se
GUI application is **event driven**; users can cause any of several events in **any order**

- GUI applications offer one small benefit to testers:
  - There is a **little need** for integration testing

- Unit testing is typically at the “button level”; that is **buttons have functions**, and these can be tested in the **usual unit-level sense**.

- The essence of **system-level testing** for GUI applications is to exercise the event-driven nature of application

A wide range of GUI testing tools has appeared on the market over the past few years.

TDDC88 has a lab on Selenium
Example: event-driven

Currency Converter

U. S. Dollar amount

Equivalent in …

- Brazil
- Canada
- European Community
- Japan

Compute
Clear
Quit
Smoke test

- Important selected tests on module, or system
- Possible to run fast
- Build as large parts as possible as often as possible
- Run smoke tests to make sure you are on the right way