Test Integration Strategies

Brian Nielsen
bnielsen@cs.auc.dk
Center of Embedded Software Systems
Aalborg University, Denmark
**Driver:** a code fragment that invokes the Unit under test

- Supplies input data
- Invokes Unit
- Collects and compares results
- ~xUnit testing framework

**Stub:** a code fragment that simulates a missing unit

- Simulate
- Pre-programmed return values
- Simplified impl.

**Component Hierarchy**

- A
  - B
  - C
  - D
  - E
  - F
  - G

**Unit under test**

**Test driver**

- Function call
- Method invocation
- Message passing
- Shared variables
Integration Testing

• Goal: Systematic construction of program structure and testing for interfacing errors
  • Loss of data/information across interfaces
    • `int f(){ ... g(y,x)...; }
      `Int g(int x,y) {};
  • Side-effects (shared data-structures)
  • Accumulation of errors (imprecision)
• Incremental
• Consider required cost and thoroughness
• Mostly blackbox, supplemented with white-box tests to ensure coverage
Big-bang integration

Integration Order

1. Test all components in isolation
2. Integrate all in one step

Component Hierarchy

- Stubs and drivers needed!
- Difficult to localize cause of errors
- Difficult to distinguish interface faults
Bottom-Up integration

Component Hierarchy

1. Test lower levels first (utility routines)
2. Implement driver (easy) to invoke module
3. Use tested modules as stubs

Integration Order

- No stubs needed!
- Mundane low-level utilities tested first, but most important top level modules tested late
  - Critical errors found late
  - Main design problems are at higher-levels
  - Timing determined by high-level modules
Top-Down integration

1. Test higher levels first (controlling modules)
2. Implement stubs to deliver responses
3. Use tested modules as drivers

Component Hierarchy

Breadth-First Integration Order

- No drivers needed!
- Many stubs may be needed
- Stubs may be problematic to write

Depth-First Integration Order

- Must provide values as expected by tester
- May require frequent alteration
Sandwich integration

Component Hierarchy

1. Combination of B-up and T-Down
2. Divide system into upper, middle (target), and lower levels
3. B-up in low-levels
4. T-Down in upper levels

Integration Order

+ combines B-up and T-Dn
+ Integration possible early
- Individual components not tested thoroughly before integration, especially middle level
System Test (Pfleger)

- Unit & integration: code implements design correctly
- **System testing**: system does what customers expect
  - Constructed software tested against established requirements during requirements analysis
  - *Validation criteria* satisfied?
  - Are all *user-visible* functional, behavioral, performance requirements satisfied?

---

Integrated modules

- **Function test**
- **Non-functional test**
- **Acceptance test**
- **Installation/Production test**

Customer

Requirements

For other Quality aspects

System functional requirements

System in use

Environment

---

- **Performance**
- **Stress / load**
- **Reliability**
- **Recoverability**
- **Security**
- **Installability**
- **...**

System accepted, validated
Acceptance Test (Pfleger)

- By customers,
  - User’s understanding of requirements
  - Customer writes, conducts, and evaluates acceptance tests
- Functional Acceptance Test (FAT)
- Production Acceptance Test (PAT)

Techniques
- Benchmark tests
- Pilot tests ~ experimental, limited edition
  - Alpha test: in-house, controlled environment, developer supervised
  - Beta test: “live” test at customer sites, unsupervised
- Parallel test: run concurrently with old system