Software Testing

Lecture Notes 3 (of 4)

Outline of the Lecture

- Integration Testing
  - Top-down
  - Bottom-up
  - Big-bang
  - Sandwich
- System Testing
- Acceptance Testing
- Distribution of Faults in a large Industrial Software System (ISSTA 2002)

Unit & Integration Testing

Objective: to ensure that code implemented the design properly.

Component code

Tested components

Unit test

Integration test

Components

Driver

Component to be tested

Mock

Test cases

Boundary conditions

independent paths

interface
• System Testing Steps
  – Function testing / Thread testing
  – Performance testing
  – Acceptance testing
  – Installation testing

• Test Automation
• Termination Problem

Comparison of Integration Strategies

<table>
<thead>
<tr>
<th></th>
<th>Top-down</th>
<th>Modified</th>
<th>Bottom-up</th>
<th>Big-Bang</th>
<th>Sandwich</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration</td>
<td>Early</td>
<td>Early</td>
<td>Early</td>
<td>Late</td>
<td>Early</td>
<td>Early</td>
</tr>
<tr>
<td>Time to basic working program</td>
<td>Early</td>
<td>Early</td>
<td>Late</td>
<td>Late</td>
<td>Early</td>
<td>Early</td>
</tr>
<tr>
<td>Driver needed</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Stubs needed</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

System Testing

Objective: to ensure that the system does what the customer wants it to do.
Function testing/Thread testing

**functional requirements**

- **Threads**:  
  - A scenario of normal usage  
  - A stimulus/response pair  
  - Behavior that results from a sequence of system-level input  
  - An interleaved sequence of port input and output events  
  - A sequence of atomic system functions (ASF)  

- **ASF**: an atomic system function is an action that is observable at the system level in terms of port input and output events

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

**Cause-and-Effect-Graph**

(test case generation from req.)

- **causes**: inputs  
- **effects**: outputs and transformations  
- **causes-and-effect graph**:  
  - boolean graph reflecting causes and effects relationships  
  - is a formal language into which a natural language specification is translated

**Basic cause-effect graph symbols**

- Identity: if $a$ then $b$
- And: if ($a$ and $b$) then $c$
- Or: if ($a$ or $b$ or $c$) then $d$
- Identity: if (not $a$) then $b$
**Specification:** the character in column 1 must be an “A” or a “B”. The character in column 2 must be a digit. In this situation, the file update is made. If the first character is incorrect, message X12 is issued. If the second character is not a digit, message X13 is issued.

**Causes**
- C1: character in column 1 is “A”
- C2: character in column 1 is “B”
- C3: character in column 2 is a digit

**Effects**
- E1: update made
- E2: message X12 is issued
- E3: message X13 is issued

---

**Constraint symbols**
- E cause-constraint: at most one of a, b, and c can be true
- I cause-constraint: at least one of a, b, and c must always be true
- O cause-constraint: one, and only one, of a and b must be true
- R cause-constraint: for a to be true, b must be true
- M effect-constraint: if effect a to is true, effect b is forced to be false

---

**Decision table for cause-and effect graph**

<table>
<thead>
<tr>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
<th>Test 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cause 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cause 2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Cause 3</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>Effect E1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Effect E2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Effect E3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**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
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

Installation Testing users site

Acceptance test at developers site
\rightarrow installation test at users site, otherwise may not be needed!!

Test Automation

- Automating parts of the testing process can provide long-term benefits to organization, such as:
  - reducing the amount of time it takes to execute a suite of tests
  - reducing the tester’s involvement in executing tests
  - facilitating regression testing
  - allowing for the simulation of hundreds of users
  - avoiding human mistakes by having tools control repetitive and tedious tasks
- Test automation refers to two key testing activities:
  - Executing the tests
  - Evaluating the output

Automated Testing Tools

- Code Analysis tools
  - Static, Dynamic
- Test execution tools
  - Capture-and-Replay
  - Stubs & Drivers
  - Comparators
- Test case generator

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