Integration Testing
8.3 Unit Testing

Sidebar 8.4 Fault Discovery Efficiency at Contel IPC

- 17.3% during inspections of the system design
- 19.1% during component design inspection
- 15.1% during code inspection
- 29.4% during integration testing
- 16.6% during system and regression testing
- 0.1% after the system was placed in the field
8.4 Integration Testing

- Bottom-up
- Top-down
- Big-bang
- Sandwich testing
- Modified top-down
- Modified sandwich
8.4 Integration Testing

Terminology

- **Component Driver**: a routine that calls a particular component and passes a test case to it.
- **Stub**: a special-purpose program to simulate the activity of the missing component.
8.4 Integration Testing
View of a System

- System viewed as a hierarchy of components
8.4 Integration Testing
Bottom-Up Integration Example

- The sequence of tests and their dependencies
8.4 Integration Testing
Top-Down Integration Example

- Only A is tested by itself
8.4 Integration Testing
Modified Top–Down Integration Example

- Each level’s components individually tested before the merger takes place
8.4 Integration Testing
Bing–Bang Integration Example

- Requires both stubs and drivers to test the independent components
8.4 Integration Testing
Sandwich Integration Example

- Viewed system as three layers
8.4 Integration Testing
Modified Sandwich Integration Example

- Allows upper-level components to be tested before merging them with others
## 8.4 Integration Testing
Comparison of Integration Strategies

<table>
<thead>
<tr>
<th></th>
<th>Bottom-up</th>
<th>Top-down</th>
<th>Modified top-down</th>
<th>Bing-bang</th>
<th>Sandwich</th>
<th>Modified sandwich</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>Late</td>
<td>Early</td>
<td>Early</td>
<td>Late</td>
<td>Early</td>
<td>Early</td>
</tr>
<tr>
<td>Component drivers needed</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Stubs needed</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Work parallelism at beginning</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Ability to test particular paths</td>
<td>Easy</td>
<td>Hard</td>
<td>Easy</td>
<td>Easy</td>
<td>Medium</td>
<td>Easy</td>
</tr>
<tr>
<td>Ability to plan and control sequence</td>
<td>Easy</td>
<td>Hard</td>
<td>Hard</td>
<td>Easy</td>
<td>Hard</td>
<td>hard</td>
</tr>
</tbody>
</table>
8.4 Integration Testing

Sidebar 8.5 Builds at Microsoft

- The feature teams synchronize their work by building the product and finding and fixing faults on a daily basis.
8.5 Testing Object-Oriented Systems

Questions at the Beginning of Testing OO System

- Is there a path that generates a unique result?
- Is there a way to select a unique result?
- Are there useful cases that are not handled?
8.5 Testing Object-Oriented Systems

Easier and Harder Parts of Testing OO Systems

- OO unit testing is less difficult, but integration testing is more extensive
The farther the gray line is out, the more the difference
8.6 Test Planning

- Establish test objectives
- Design test cases
- Write test cases
- Test test cases
- Execute tests
- Evaluate test results
8.6 Test Planning

Purpose of the Plan

- Test plan explains
  - who does the testing
  - why the tests are performed
  - how tests are conducted
  - when the tests are scheduled
8.6 Test Planning

Contents of the Plan

• What the test objectives are
• How the test will be run
• What criteria will be used to determine when the testing is complete
8.7 Automated Testing Tools

- Code analysis
  - Static analysis
    - code analyzer
    - structure checker
    - data analyzer
    - sequence checker

- Output from static analysis

```
<table>
<thead>
<tr>
<th>Metric</th>
<th>Bad</th>
<th>Average</th>
<th>Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inline faults</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interface faults</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nesting</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decisions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paths</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uninitialized variables</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External coupling</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
8.7 Automated Testing Tools (continued)

• Dynamic analysis
  – program monitors: watch and report program’s behavior

• Test execution
  – Capture and replay
  – Stubs and drivers
  – Automated testing environments

• Test case generators
8.8 When to Stop Testing

More faulty?

- Probability of finding faults during the development
8.8 When to Stop Testing
Stopping Approaches

- Coverage criteria
- Fault seeding
  \[ \frac{\text{detected seeded faults}}{\text{total seeded faults}} - \frac{\text{detected nonseeded faults}}{\text{total nonseeded faults}} \]
- Estimating the number of faults
  \[ N = \frac{S}{s} \]
- Confidence in the software, \( C \)
  \[ C = \begin{cases} 
  1, & \text{if } n > N \\
  \frac{s}{s-N+1}, & \text{if } n \leq N 
  \end{cases} \]
8.8 When to Stop Testing
Identifying Fault-Prone Code

• Track the number of faults found in each component during the development
• Collect measurement (e.g., size, number of decisions) about each component
• Classification trees: a statistical technique that sorts through large arrays of measurement information and creates a decision tree to show best predictors
  – A tree helps in deciding which components are likely to have a large number of errors
8.8 When to Stop Testing
An Example of a Classification Tree

![Classification Tree Diagram]

- Size:
  - < 100 LOC
  - > 300 LOC

- Number of decisions:
  - < 15
  - ≥ 15

- Design review:
  - Yes
  - No

- Code changes:
  - < 5
  - ≥ 5
8.9 Information Systems Example
Piccadilly System

• Consider
  
  CASE 1: Y := X/3;
  CASE 2: Y := 2X-25;
  CASE 3: Y := X MOD 10;
  ENDCASE;

• Using data-flow testing strategy rather than structural
  – Definition-use testing
8.10 Real–Time Example
The Ariane–5 System

• The Ariane–5’s flight control system was tested in four ways
  – equipment testing
  – on–board computer software testing
  – staged integration
  – system validation tests

• The Ariane–5 developers relied on insufficient reviews and test coverage
8.11 What this Chapter Means for You

- It is important to understand the difference between faults and failures
- The goal of testing is to find faults, not to prove correctness