SE310
Analysis and Design of Software

Lecture 13 – Testing Designs
(Prototype, Code Generation, Simulation, Walk-throughs, Mock-Ups)
Discussion

Why is it Hard to Test a Design?

List 3 Key Challenges

What can We Test?

- A Prototype?
- Code Generated from UML 2.5 (Class, Object, OIM)
- Simulation [Executable State Machines, MATLAB and Toolboxes, System Toolkit, Petri-Nets, etc.]
- Walkthroughs of Models (Default method for Design V&V)
- Mock-Ups (Default for GUI and HCI)

How will you Test your Design before Final Inspection?
New Trends in Testing

Test Driven (Design) and Development – **TDD**
- **Kent Beck** “rediscovered” (the XP guy!)
- Is it just old wine in new bottles?
- Who cares, it works, and should be expanded to all phases of design and development (verify as you go!)

**Continuous Integration for Dev Ops**
- Inspired by Cloud Services and the Web
- GitLab and GitHub have embraced
- Packages tools we use in SE 420 in the Cloud for “verify as you go” during phases of development mostly (design to a degree)
- Idea is to automate testing and collect metrics
- Is it just old wine in new bottles?
- Who cares, it’s a great idea and automates better what has been done by successful organizations for decades

© Sam Siewert
Testing Strategies

- The "V-Chart for Software Development"

- Requirements Analysis
- System Design
- Architecture Design
- Module Design
- Coding
- Unit Test Design
- Integration Test Design
- System Test Design
- Acceptance Test Design
- Acceptance Test
- System Test
- Integration Test
- Unit Testing
Domain Models – Use Case Details

- Complete a Design
- Provide Code Re-Use Candidates and PoC

Start Here!

https://www.modelio.org/

USE Modelio as your DESIGN TOOL

Helpful Validation and Verification Features for Design

- Integrated Models
- Checklists – Completeness
- CPP and Java Code Generation
Testing Levels

- **Acceptance Tests** – Requirements, Deployment Diagram
  - Test for each Requirement
  - Gating Tests for Shipment, Release Notes

- **System Test** – Analysis Use Cases, Domain Model, OIM Sequence
  Message Passing, CFD/DFD, Block Diagrams
  - End-to-End (Fully Integrated)
  - Performance
  - Major Features
  - Fault-Injection

- **Integration and Test** - OM, OIM Methods and Collaboration, Activity
  Diagrams, Package Diagrams, Subsystems
  - Components, Subsystems
  - Interfaces, Protocols
  - Verification of Public Interfaces (Methods and Attributes)

- **Unit Tests** - State Machines, Method Details
  - Functional, Basic Block, Test Cases (Vectors)
  - Test Drivers to Instantiate and Exercise Methods (Functions)
Methods to Test Design

- Brainstorm …
Methods to Test Design – Brainstorm
Results from 2016

1. Design Walk-throughs and Inspections
2. Concurrent SQA, QA, QC (“V” model)
3. Structural Code Generation (Modelio – From Classes to OOP)
4. Behavioral Simulation (Model Based SE)
   1. Telelogic State Machine Simulation / Emulation ([IBM Rational Tools for SWE – Telelogic Support])
   2. Mathworks Stateflow
   3. Yakindu Statecharts, Webratio, Xamarin, PIPE2 Petr Net Editor, Timed Arc Petri Nets
   4. Write Your Own Emulator
5. Alpha and Beta Early Customer Programs (for Prototypes)
6. Prototyping to Prove out Design (PoC)
7. Voice of the Customer (Focus Group for Alpha / Beta)
8. Compare to Existing Products (Your Own, Competitors, Product Ecosystem)

© Sam Siewert
What Is Software Testing?

- Software testing is a dynamic validation activity to detect defects and undesirable behavior in the software, and demonstrate that the software satisfies its requirements and constraints.
Objectives of Software Testing

• Testing is a process of executing a program with the intent of finding an error.

• A good test case is one that has a high probability of finding an undetected error.

• A successful test is one that uncovers an undiscovered error.
What is Testing and What is Not?

• Testing is a process for finding semantic or logical errors.

• Testing is not aimed to find syntax errors.

• Testing can reveal the presence of errors, NOT their absence.

• Class exercise: what is the difference between testing and debugging?
Why Software Testing?

• Review and inspection are not adequate because they do not execute the program. Can you test a car without driving it?

• Why need testing:
  – Instrumentation systems & appliances
    • coded in assembly/machine language ===> costly to repair
  – Process control systems
    • failure is politically unacceptable & economically undesirable.
  – Military systems
    • involve risk of human life
Software Testing in Practice

• Testing amounts to 40% -- 80% of total development costs
  – ~ 40% for information systems
  – more than 80% for real-time embedded systems

• On one occasion the average costs of a field detected bug was US$15,000 (1991).

• Testing receives the least attention, and often not enough time and resources.

• Testers are often forced to abandon test efforts because changes have been made.
Software Testing in Practice

• **Most companies’ new hires are testers.**

• Most testing work is manually testing the software system; help from tools is limited.

• In many cases, testing is not performed by using systematic testing methods or techniques. It is not effective.

• There are “conflict of interest” between testers and developers, and testers and management.
Benefit of Software Testing

• Benefits:
  – Conscious of need for error-free software.
  – Ensure the more flagrant kinds of errors will be detected.
  – Improving software quality.
  – Adding a layer of quality assurance to inspection, walkthrough, and peer review.
  – Helping in data collection for root cause analysis and process improvement.
  – Providing a framework for applying SQA techniques.

• Systematic software testing is more effective than programmers’ casual testing.
V&V and Testing

• Verification: “are we building the product right?”

• Validation: “are we building the right product?”

• Testing is a pragmatic approach to Software Quality Assurance (SQA).

• Program correctness proof is a theoretical approach (to SQA).
V&V and Testing

- Testing is aimed at proving that errors exist; it cannot assert that a program is error free.

- Program correctness proof attempts to prove that a program is free from errors. Once proved, the program is correct forever.

- A combination of above two is symbolic execution. It executes programs using symbolic values and rules developed by program correctness proof.
Test Case Generation – White Box

• Knowing the internal workings of a product
  • focus on the program’s structure and internal logic
  • test cases are designed according to the structure and internal logic

• Well-known techniques
  • Basis path testing: test cases are designed to exercise the control flow paths of a program
  • Condition testing: test cases are designed to exercise each outcome of a condition
  • Data flow testing: test cases are designed to test data elements’ define and use relationships
Test Case Generation – Black Box

- **Knowing the functional specification**
  - focus on functional requirements
  - test cases are designed to test the functionality of the program

- **Well-known techniques**
  - boundary value analysis: test cases are derived according to the boundary values of variables
  - causal-effect analysis: test cases are derived according to the stimuli and responses, and input output relationships
  - equivalence partitioning: test cases are derived from partitions of the input and output domains
Test Categories

- Unit Testing: testing at the individual module level, functions, modules, classes, etc.
- Integration Testing: testing the interfaces among the modules or components.
- Validation Testing: test according to customer’s functional requirements.
- System Testing
  - Recovery Testing
  - Security Testing
  - Stress Testing
  - Performance Testing
Test Driven Development

1. Prepare for TDD
2. Write/edit tests for features to be implemented next
3. Run new tests
4. All new tests fail?
   - No: Go back to step 3
   - Yes: Write/edit production code
5. Run all tests
6. All tests pass?
   - No: Refactor code
   - Yes: Go to step 7
7. More features to implement?
   - Yes: Go back to step 2
   - No: Test coverage accomplished?
8. Test coverage accomplished?
   - Yes: Halt
   - No: Add tests to increase coverage
Software Testing Budget

- Size of testing budget:
- expressed in terms of x% of total SW development costs.
- in terms of system size: $x per statement.
- compute in terms of productivity of the test team in terms of time: planned & spent.
Test Methods at Each Level for Concurrent Process
- Acceptance Tests – Requirements, Deployment Diagram
- System Test – Analysis Use Cases, Domain Model, OIM Sequence Message Passing, CFD/DFD, Block Diagrams
- Integration and Test - OM, OIM Methods and Collaboration, Activity Diagrams, Package Diagrams, Subsystems
- Unit Tests - State Machines, Method Details

Details of Composing Test Drivers, Cases, and Automation Verification and Validation

Regression Testing

Methods of Black-box and White-box Test Development
Testing Levels

- Acceptance Tests – Requirements, Deployment Diagram

- System Test – Analysis Use Cases, Domain Model, OIM Sequence Message Passing, CFD/DFD, Block Diagrams
  - End-to-End (Fully Integrated)
  - Performance
  - Major Features
  - Fault-Injection

- Integration and Test - OM, OIM Methods and Collaboration, Activity Diagrams, Package Diagrams, Subsystems

- Unit Tests - State Machines, Method Details
Use Case Based Testing

Step 1. Prioritizing the use cases using the requirement-use case traceability matrix.

Step 2. Generating test scenarios from the expanded use cases.

Step 3. Identifying test cases.

Step 4. Generating test data (values).

Step 5. Generating test scripts.
Prioritizing Use Cases

• Why prioritizing
  – We may not have needed resources or time.
  – Prioritizing ensures that high-priority use cases are tested.

• The requirement-use case traceability matrix
  – rows are requirements and columns are use cases
  – an entry is checked if the use case realizes the requirement
  – the bottom row sums up the priority of each use case.
## Requirement-Use Case Traceability Matrix

<table>
<thead>
<tr>
<th>Priority Weight</th>
<th>UC1</th>
<th>UC2</th>
<th>UC3</th>
<th>UC4</th>
<th>UC5</th>
<th>UC6</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1 3</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>R3 2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4 1</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>R6 1</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

| Score | 5 | 5 | 1 | 1 | 3 | 1 |

Copyright © 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.
Generating Use Case Scenarios

• A scenario is an instance of a use case.

• A scenario is a concrete example showing how a user would use the system.

• A use case has a primary scenario (the normal case) and a number of secondary scenarios (the rare or exceptional cases).

• This step should generate a sufficient set of scenarios for testing the use case.
Identifying Test Cases

- Identifying test cases from the scenarios using a test case matrix:
  - the rows list the test cases identified
  - the columns list the scenarios, the user input variables and system state variables, and the expected outcome for each scenario
  - each entry is either
    - V, indicating a valid variable value for the scenario
    - I, indicating an invalid variable value for the scenario
    - N/A, indicating a not applicable variable value for the scenario
## Use Case Based Test Case Generation

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>Scenario</th>
<th>Email ID</th>
<th>Password</th>
<th>Registered</th>
<th>Event Info</th>
<th>Event Open</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC1</td>
<td>Successful Registration</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>Display confirmation</td>
</tr>
<tr>
<td>TC2</td>
<td>User Not Found</td>
<td>I</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>Error msg</td>
</tr>
<tr>
<td>TC3</td>
<td>Invalid Info</td>
<td>V</td>
<td>V</td>
<td>NA</td>
<td>I</td>
<td>NA</td>
<td>Error msg</td>
</tr>
<tr>
<td>TC4</td>
<td>User Quits</td>
<td>V</td>
<td>V</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>Back to login</td>
</tr>
<tr>
<td>TC5</td>
<td>System Unavailable</td>
<td>V</td>
<td>V</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>Error msg</td>
</tr>
<tr>
<td>TC6</td>
<td>Registration Closed</td>
<td>V</td>
<td>V</td>
<td>NA</td>
<td>NA</td>
<td>I</td>
<td>Error msg</td>
</tr>
<tr>
<td>TC7</td>
<td>Duplicate Registration</td>
<td>V</td>
<td>V</td>
<td>I</td>
<td>NA</td>
<td>NA</td>
<td>Error msg</td>
</tr>
</tbody>
</table>
# Identifying Test Data Values

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>Scenario</th>
<th>Email ID</th>
<th>Password</th>
<th>Registered</th>
<th>Event Info</th>
<th>Event Open</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC1</td>
<td>Successful Registration</td>
<td><a href="mailto:lee@ca.com">lee@ca.com</a></td>
<td>Lee123</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Display confirmation</td>
</tr>
<tr>
<td>TC2</td>
<td>User Not Found</td>
<td><a href="mailto:unknow@ca.com">unknow@ca.com</a></td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>Error msg</td>
</tr>
<tr>
<td>TC3</td>
<td>Invalid Info</td>
<td><a href="mailto:lee@ca.com">lee@ca.com</a></td>
<td>Lee123</td>
<td>NA</td>
<td>I</td>
<td>NA</td>
<td>Error msg</td>
</tr>
<tr>
<td>TC4</td>
<td>User Quits</td>
<td><a href="mailto:lee@ca.com">lee@ca.com</a></td>
<td>Lee123</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>Back to login</td>
</tr>
<tr>
<td>TC5</td>
<td>System Unavailable</td>
<td><a href="mailto:lee@ca.com">lee@ca.com</a></td>
<td>Lee123</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>Error msg</td>
</tr>
<tr>
<td>TC6</td>
<td>Registration Closed</td>
<td><a href="mailto:lee@ca.com">lee@ca.com</a></td>
<td>Lee123</td>
<td>NA</td>
<td>NA</td>
<td>No</td>
<td>Error msg</td>
</tr>
<tr>
<td>TC7</td>
<td>Duplicate Registration</td>
<td><a href="mailto:lee@ca.com">lee@ca.com</a></td>
<td>Lee123</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>Error msg</td>
</tr>
</tbody>
</table>
Take Away

- Test Definition and Development Should Start With Requirements
- **Testing Concurrent with Analysis, Design, Implementation**
- Define Test Cases, Methods, and Use Tools at All Levels of Analysis, Design, Implementation
- Define Exit Criteria, So Test Completion is not Arbitrary – E.g. Code Path Coverage
- Gating Tests Prevent Shipment of Defective Software
- Some Defects and Deficiencies May Be Documented as Limitations in Release Notes
- Can’t Prove Code is Bug Free - SWE Formal Methods – Advanced Concepts to Prove Correctness [Research Topic]
  - Pre-condition, Post-condition (Axiomatic Proofs)
  - Prove Ability to Meet Deadlines (Rate Monotonic)
  - Termination and [Halting Problem and Undecidable Problems](#)