Learning Objectives

- Understand the concepts of black box tests, white box tests, observability, and controllability
- Understand the principles of debugging
- Understand when a unit test is used and how it is constructed
- Understand when an integration test is used and how it is constructed
- Understand when an acceptance test is used and how it is constructed
Design and Testing

- A design process is really a continual increase in specificity from engineering requirements to the detailed design.
- We now consider the question of how to test that the resulting system meets design requirements.
- Use a test structure

A Design-Test Structure

- This model starts with the engineering requirements, proceeds to implementation, and then onto testing.
- Every level of design has a corresponding level of test.
- The test process is actually split between the two halves.
- Each test performed on the right must be engineered during the development of the system in the left.
- An acceptance test plan should be written with the requirements specification, integration tests defined and written during the system design, etc.
Avoid Smoke Tests

• In our enthusiasm to complete a project many of us all too often rely on a “smoke test”
  – Turn on a system to see if it works.
  – The name of this test is a reference to what may happen to the system if the test fails - it burns up and smokes.
• Beyond being a potentially expensive way to test a system, a smoke test is not a systematic approach to verify that the system behaves as expected.
• Customers will not be impressed with "Hey, it didn't catch on fire!" as the test result.

Need for Clear Tests

• The test cases define exactly what the module must do
• Testing prevents "feature creep"
  – The development of a module is complete when its test is passed
• Test cases motivate developers by providing immediate feedback
• Test cases force designers to think about extreme cases
• Test cases are a form of documentation
• Test cases force the designer to consider the design of the module before building it
Types of Testing

• Tests fall into two general types
  – **Black box**
    • Those that are performed without any knowledge of the system’s internal organization
    • Testing is typically conducted by changing the inputs and comparing the system outputs to their expected values
    • Input and output values can be classified as typical, boundary, extreme, and invalid
  – **White box**
    • Conducted with knowledge of the internal working of the system
    • Build tests which target specific internal nodes of the system to check that they are operating as expected
    • Tests should be written to check that the node can handle typical, boundary, extreme and illegal situations

Observability and Controllability

• One of the many goals in designing a system is to increase its testability
  – A design is *testable* when a failure of a component or subsystem can be quickly located
• One way to increase the testability of a system is to increase *controllability* and *observability*
  – **Controllability**
    • The ability to set any node of the system to a prescribed value
  – **Observability**
    • The ability to observe any node of a system
• In black box testing, controllability and observability are low
• In white box testing, controllability and observability may be higher, depending on the design
Black Box Test Example

- Only points x1, x2, x3, and f can be controlled/observed
- Inputs (x1-x3) can be varied and the output f can be observed
- Low controllability and observability

White Box Test Example

- Test points A, B, and C are added to the circuit
- Inputs (x1-x3) can be varied and the output f and test points A-C can be observed
- High observability
- Low controllability
Stubs

- A stub is a device that is used to simulate a subcomponent of a system
- This might be done for multiple reasons:
  - The subcomponent has not yet been built or is built and is at another location
  - The risk of damaging the subcomponent warrants using a stand-in
- Stubs are used to simulate inputs or monitor outputs of the UUT
- Both hardware and software stubs can be used in designing a system
- In software testing, stub routines are developed to either call other functions or act as those to be called by the UUT

- For example
  - A function generator for a audio input
  - A `printf()` instead of a file write
  - DIP switch instead of a bus connection

Stub Example

- Assume that the circuit is ultimately to be integrated into a larger system
- The input to this system is a time varying source with certain resistive and capacitive characteristics, while the output is connected to another system with a known input resistance range
- On the input side is a function generator, an off-the-shelf component, connected to a resistor and capacitor that models the expected characteristics of the final system
- The stub on the output side is simply a resistor, whose value can be varied over the expected load
Test Case Properties

- **Accurate** - The test should check what it is supposed to and exercise an area of intent
- **Economical** - The test should be performed in a minimal number of steps
- **Limited in complexity** - Tests should consist of a moderate number (10-15) of steps
- **Repeatable** - The test should be able to be performed and repeated by another person
- **Appropriate** - The complexity of the test should be such that it is able to be performed by other individuals who are assigned the testing task
- **Traceable** - The test should verify a specific requirement
- **Self cleaning** - The system should return to the pre-test state after the test is complete

Debugging Process

- Observe the problem under different operating conditions
- Form a hypothesis as to what the potential problem is
- Conduct experiments to confirm or eliminate the hypothesized source of the problem
- Repeat until the problem is eliminated
- Check easiest problems first
  - You can perform more in a given time
- Start at lowest levels of abstraction
  - Upper levels rely on lower level
- Example
  - Is the system powered up?
  - Is the testing equipment adjusted properly?
  - Are the bus lines being correctly manipulated?
  - Have you initialized the system?
  - Are you printing out the right variable/type?
Unit Testing

- A unit test is a test of the functionality of a system module in isolation
- Should be traceable to the detailed design
- Consists of a set of test cases
- Each test case establish that a subsystem performs a single unit of functionality to some specification
- Test cases should be written with the express intent of uncovering undiscovered defects

Unit Testing (continued)

- Write unit test during implementation; why?
  - White box tests
  - Thinking about test situations and conditions may lead the designer to uncover errors before they are ever designed into the system
  - Unit tests need to be written with an understanding of the internal organization of the component, and a system is best understood during its development
Unit Test (example)

```c
if (16 < Celsius Temperature < 32)  
   Fahrenheit Temperature = ROM[input-16];  
else  
   Fahrenheit Temperature = (9 * Celsius Temperature)/5 + 32;
```

- What inputs are used to check the ROM?
- What inputs check the conditionals?
- Processing path – sequence from A to B
  - Test coverage
  - Path complete coverage

Testing Methods

- Matrix Test
- Automated Scripts Tests
  - Useful for regression testing
- Step-by-step Test

- These can be applied to any level of test
  - Unit
  - Integration
  - Acceptance, etc.
Matrix Test

Test Writer: Joe Engineer

Test Case Name: ADC function test
Test ID #: ADC-FT-01

Description: Verify conversion range and clock frequency. Output goes to 0 in presence of null clock.

Type: ☑ white box ☑ black box

Tester Information

Name of Tester: Date:
Hardware Ver: 1.0
Time:

Setup: Isolate the ADC from the system by removing configuration jumpers.

<table>
<thead>
<tr>
<th>Test</th>
<th>Vr</th>
<th>Clock</th>
<th>Expected output</th>
<th>Pass</th>
<th>Fail</th>
<th>NA</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0V</td>
<td>10kHz</td>
<td>0</td>
<td>0x000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2.0V</td>
<td>0Hz</td>
<td>0</td>
<td>0x000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Overall test result:

Step-by-Step Test

Test Writer: Joe Engineer

Test Case Name: Finite State Machine Path Test #1
Test ID #: FSM-Path-01

Description: Simulate insertion of money with a mix of nickels and dimes. Verifies FSM, outputs candy in response to a total deposit of $0.30.

Type: ☑ white box ☑ black box

Tester Information

Name of Tester: Date:
Hardware Ver: 1.0
Time:

Setup: Make sure that the system was reset and is in state $0.00.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Expected Result</th>
<th>Pass</th>
<th>Fail</th>
<th>NA</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Strobe Nickel</td>
<td>State should go to $0.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Strobe Dime</td>
<td>State should go to $0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Wait</td>
<td>State should remain $0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Strobe Nickel</td>
<td>State should go to $0.20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Strobe Dime</td>
<td>State should go to $0.30, output candy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Nothing</td>
<td>State should go to $0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Overall test result:
Integration Testing

• Write integration test during initial design
  – Help insure requirements are being met
  – Help to firm-up design
  – Requires the designer think about the expected behavior of the subsystems
  – Requires designer to think about extreme behaviors of subsystems

Write the Integration Test

• What are the different paths of execution through the system?
• Are all modules exercised at least once during integration testing?
• Have all the interface signals been tested?
• Have all the interface modes been exercised?
• Does the system process information at the required rate and met timing requirements?
Acceptance Testing

- Might be formal legal document
  - May involved regulatory agencies and/or standards acceptance tests
- Written along with requirements
- Traceable to engineering requirements
- Identifies
  - Scope – how much of the system is tested?
  - Level – how deep will testing be performed?

Example: Autonomous Robot

- Autonomous navigating robot
- Engineering requirements
  - The robot’s center must stay within 12 to 18 centimeters of the wall over 90% of the course, while traveling parallel to a wall over a 3 meter course
  - The robot’s heading should never deviate more than 10 degrees from the wall’s axis, while traveling parallel to a straight wall over a 3 meter course
Robot Acceptance Test

Test Writer: Sue L. Engineer

Test Case Name: Robot acceptance test #1
Test ID #: Robot-AT-01

Description: Checks the engineering requirement: The robot’s center must stay within 12 to 18 centimeters of the wall over 90% of the course, while traveling parallel to a wall over a 3 meter course.

Tester Information

Name of Tester: Date:
Hardware Ver: Robot 1.0
Time:

Setup: Completed robot should be fully charged and placed on 3 meter test track.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Expected Result</th>
<th>Pass</th>
<th>Fail</th>
<th>N/A</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Write a program to monitor the robot’s position from the wall.</td>
<td>Program should be statically tested to verify accuracy. Should sample wall at a sufficient rate depending on speed.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Put robot on test track, run test, and download data.</td>
<td>The robot should travel down the entire length of the test track and then stop.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Plot test data in a spreadsheet program.</td>
<td>Plot of position vs. time should be within 12-18 cm 90% of the time.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Overall test result:

Example: Robot Architecture

![Robot Architecture Diagram]

Electrical & Computer Engineering Engineering Test and Validation II (25 of 32)
Unit Testing Possibilities

- MCU (hardware)
- LCD
- Switches
- Compass
- Range finder
- H-bridge
- Motors
- Chassis
- MCU (software)

Unit Test: Compass

<table>
<thead>
<tr>
<th>Module</th>
<th>Digital Compass – Geosensor version 2.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>Earth’s magnetic field: An orientated field of magnetic force beginning and ending at the earth’s magnetic poles.</td>
</tr>
<tr>
<td></td>
<td>SClk – Clock signal to clock data through the module. Maximum Frequency is 10Mhz.</td>
</tr>
<tr>
<td></td>
<td>SDIn – Serial data input to send data into the compass module. Date is valid on positive SClk edges.</td>
</tr>
<tr>
<td>Outputs</td>
<td>SDOut – Serial data output from the compass module. Data is valid on negative clock edges.</td>
</tr>
<tr>
<td>Functionality</td>
<td>Senses the earth’s magnetic field and determines the orientation of the compass with respect to the field. This orientation is stored in an internal register and can be retrieved through the SPI interface.</td>
</tr>
<tr>
<td>Test</td>
<td>Comp-UT-01</td>
</tr>
</tbody>
</table>
Unit Test: Compass (Step-by-Step)

<table>
<thead>
<tr>
<th>Test Writer: Sue L. Engineer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case Name: Compass unit test 1</td>
</tr>
<tr>
<td>Test ID #: Comp‐UT‐01</td>
</tr>
</tbody>
</table>

**Description:** Checks that the compass returns correct angular measurements to the MCU. Test program is in ./test/compass_unit_test_1.c

**Type:**
- White box
- Black box

**Tester Information**

<table>
<thead>
<tr>
<th>Name of Tester:</th>
<th>Date:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Ver: Compass Module - Geosensor version 2.3</td>
<td></td>
</tr>
<tr>
<td>Time:</td>
<td></td>
</tr>
</tbody>
</table>

**Setup:** Compass module should be wired to the MCU through the SPI interface pins. The MCU should be connected to an RS232 terminal through its SCI interface. The terminal should be configured to run at 9600 baud. Cardinal directions map should be aligned using the magnetic compass.

<table>
<thead>
<tr>
<th>Action</th>
<th>Expected Result</th>
<th>#</th>
<th>#</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Compile compass in test directory</td>
<td>IDE should generate no warnings or errors.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Download</td>
<td>MCU should report &quot;download successful&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Execute</td>
<td>MCU should display compass splash screen on terminal interface.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Orientate compass to 0 degrees.</td>
<td>Terminal interface should display 0 degrees +/- 10 degrees.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Orientate compass to 30 degrees.</td>
<td>Terminal interface should display 30 degrees +/- 10 degrees.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Orientate compass to 45 degrees.</td>
<td>Terminal interface should display 45 degrees +/- 10 degrees.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. Orientate compass to 145 degrees.</td>
<td>Terminal interface should display 315 degrees +/- 10 degrees.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Overall test result:**

Some Integration Test Possibilities

- MCU + motors + bridge + switches
- Chassis + digital compass + MCU + motors + bridge + LCD
- Chassis + range finder + MCU + motors + bridge
Step-by-step Integration test

Test Writer: Sue L. Engineer

Test Case Name: Robot integration test #1
Test ID #: Robot:IT-01

Description: Checks interaction of DC motors on the magnetic compass.

Type: □ white box □ black box

Tester Information

Name of Tester: Date:

Hardware Ver: Robot 1.0

Setup: A wooden turn-table should be placed on top of the cardinal direction map. This map should be aligned with a magnetic compass. There should be no metal present while the alignment is being performed. Next, the partially-assembled robot should be placed on the turn-table. The MCU should be connected to a terminal to observe and record data.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Write program to spool compass readings while simultaneously driving motors.</td>
<td>Program should be statically tested to verify accuracy. Should sample compass at a sufficient rate depending on speed.</td>
</tr>
<tr>
<td>2</td>
<td>Run acceptance test</td>
<td>Test program should prompt user to turn the robot to an orientation and then spin the motors will then spin up and down.</td>
</tr>
<tr>
<td>3</td>
<td>Plot spooled data in spreadsheet program.</td>
<td>Plots should be analyzed to see if compass deviated any more than 10 degrees from set point.</td>
</tr>
</tbody>
</table>

Overall test result:

---

Reasons to Develop and Conduct Tests

- Testing reduces the number of bugs in existing and new features
- Tests are good documentation
- Tests improve design
- Tests constrain features
- Tests defend against other designers
- Testing forces you to slow down and think
- Testing makes development faster