Software Testing and Test Coverage

Overview

• The Purpose of Software Testing
• Test Completion Criteria
• Measuring Test Coverage
  – Requirements
  – Structure
  – Architecture
• Miscellaneous Test Requirements
What role does testing play?

• Testing is one method for assuring that software behaviours are appropriate in the system context.
  – It is not the sole method: reviews and inspections also contribute.

• Software testing can verify that object code satisfies requirements.

• Software testing can also validate the appropriateness of software behaviours.
  – Though need to design specific test scenarios to achieve this.

Testing and Software Assurance

• The purpose of testing is to:
  – demonstrate that software satisfies its requirements
  – demonstrate with a high degree of confidence that errors which could lead to unacceptable failure conditions have been removed.

• Software assurance objectives determine whether the extent of testing is sufficient.
Testing ‘Completeness’

• How do we know when software testing is complete?
  – How do you know when anything is complete?
  – You measure it, and then compare that measurement against completion objectives.

• How do we measure software testing ‘completeness’?
  – Requirements Coverage
  – Structural Coverage
  – Architectural Coverage

• Requirements, structure and architecture are the three software abstractions used to develop and build software.

• Together, they roughly describe the ‘size’ of the software.
  – As such, they can be used as a measure of software testing.
  – The completeness criteria are relatively easy to determine (e.g. number of requirements, number of lines of code, etc).

Testing Completeness Measures

• Requirements Coverage
  – Has the software been tested against all requirements for the normal range of use?
  – Has the software been tested against all requirements for abnormal or unexpected usage?

• Structural Coverage
  – Has each element of the software been exercised during testing?

• Architectural Coverage
  – Have the actual control and data links been utilised during testing?
Some Terminology Issues

• DO-178B often uses obscure or confusing terminology.
  – It’s a camel: a horse designed by committee.

• Comparing DO-178B to MIL-STD-498:
  – High-Level Requirements in DO-178B are Software Requirements in MIL-STD-498.
  – Low-Level Requirements in DO-178B are Software Design in MIL-STD-498.

• When DO-178B talks about requirements coverage, it means both software requirements and software design.

Testing Completeness Measures (cont)

• Requirements Coverage
  – Normal Range Coverage of Software Requirements
  – Normal Range Coverage of Software Design
  – Robustness Coverage of Software Requirements
  – Robustness Coverage of Software Design

• Structural Coverage
  – Code Statements
  – Decisions
  – Conditions and Decisions

• Architectural Coverage
  – Data Paths
  – Control Links
measurement vs. basis of test

- While there are a number of measures for the completeness of testing, these should not be confused with the basis of testing.
- Testing should not be conducted against the source code or against the software architecture.
- All tests should be requirements based.
  - That is, all tests should seek to determine whether software behaviour accurately reflects the requirement.
  - If you don’t test against a requirement, how will you know that the behaviour exhibited by the software is appropriate?
  - If you look at the code and see that a particular input should be tested, but no requirement tells you how the software should respond to that input, there is a shortfall in the requirements.

requirements coverage

- Normal Range Conditions
  - coverage of equivalence classes and boundary values for variables (real, integer, boolean, etc)
  - multiple iterations of time-related functions (e.g. filters, integrators, delays, etc)
  - coverage of valid state transitions

- Robustness Conditions
  - coverage of equivalence classes and boundary values for invalid conditions
  - system initialisation during abnormal conditions
  - failure modes of incoming data
  - out of range loop count values
  - protection mechanisms for exceeded timeframes
  - arithmetic overflow
  - coverage of invalid state transitions
**Equivalence Classes**

- Consider a FADEC for a jet engine with afterburner.
  - Expected throttle setting is 0 – 100.
  - Above 75, afterburner is engaged.
  - 8-Bit Two’s Compliment Integer

- What tests are required to provide coverage of equivalence classes (normal and robustness) and boundary conditions?

<table>
<thead>
<tr>
<th>-128</th>
<th>0</th>
<th>75</th>
<th>100</th>
<th>127</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑↑↑↑</td>
<td>↑↑↑</td>
<td>↑</td>
<td>↑</td>
<td>↑↑↑↑</td>
</tr>
</tbody>
</table>

**Structural Coverage**

- **Statement Coverage**
  - Every line of code exercised at least once during requirements based testing.
    - To show that every line of code is needed.

- **Decision Coverage**
  - Every decision point has taken all possible outcomes during requirements based testing.
    - To show that the behaviour following all possible decision outcomes is appropriate.

- **Modified Condition/Decision Coverage**
  - Every decision point has taken all possible outcomes during requirements based testing.
  - Every condition has taken all possible values during requirements based testing.
    - To show that the response to all possible conditions is appropriate.
  - Every condition has been shown to be able to independently affect the outcome of the decision during requirements based testing.
    - To show that the software only makes decisions based on relevant information.
Definitions

- **Statement**
  - A line of code.

- **Decision**
  - A branch or place in the code where execution could take one of two or more possible paths.

- **Condition**
  - A factor that is used to make a decision.

```java
if (weight_on_wheels) & (air_speed < limit)
    rev_thrust = REVERSE_THRUST_ENABLED;
else
    rev_thrust = REVERSE_THRUST_DISABLED;
```

Test Effort Estimates

- Only considering structural coverage criteria (requirements and architecture may have varying influences).
- Estimate only!
- Number of Tests to Achieve Statement Coverage:
  - 3*(Number of Statements/60)
- Number of Tests to Achieve Decision Coverage
  - 3*(Number of Statements/60) + Number of Empty Decision Paths (i.e. ‘if’s without ‘else’s)
- Number of Tests to Achieve MC/DC
  - 3*(Number of Statements/60) + Number of Empty Decision Paths + Sum (i = 1 to Number of Decisions) (Number of Conditions in Decision (i) + 1) – 2*Number of Decisions
- Example: 10 KSLOC
  - Statement Coverage: 500 tests
  - Decision Coverage: 600 tests
  - MC/DC: 1,500 tests
Example

• How many tests are required for the following fragment?

```plaintext
if (C OR D) AND (F OR G)
    Proc1();
else
    Proc2();
```

• The minimum number of tests to meet the structural coverage criteria for:
  – Statement Coverage
  – Decision Coverage
  – MC/DC
  – Exhaustive Coverage

MC/DC Test Cases

<table>
<thead>
<tr>
<th>Test</th>
<th>C</th>
<th>D</th>
<th>F</th>
<th>G</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

Choosing where to start is the hardest part.

• Result has been both T and F (Decision Coverage)
• Each condition has been both T and F (Condition Coverage)
• Each test has changed one variable and the result has changed.
• Some effort is required to determine the above tests.
  – Would it be easier to just do exhaustive testing?
MC/DC Effectiveness Study

• Most common criticism of DO-178B:
  – MC/DC costs a lot and does not add much value.
• A study was conducted by MIT.
  – http://sunnyday.mit.edu/papers/dupuy.ps
• The study concluded:
  – Tests required to increase coverage from DC to MC/DC account for about 40% of testing effort.
  – MC/DC finds errors not detectable by functional testing without a structural measure of completeness.
  – MC/DC finds more errors than DC or SC.
  – BUT: the errors found by MC/DC and not found by DC or SC are less likely to be the type of errors that contribute to hazardous failure conditions.

Measuring Structural Coverage

• Measuring structural coverage can be a time consuming process.
  – We looked at one decision, most software applications make thousands or millions of decisions.
• Highly recommended: use a software tool.
  – There are plenty available.
• Can be done by hand (e.g. Classic Hornet), but not recommended.
• Question: how do you use a tool to measure structural coverage of tests conducted on target hardware?
• Answer:
  – Run the tests in a simulated environment where the tool works.
  – Measure the structural coverage in the simulated environment.
  – Run the same tests in the target environment.
Shortfalls in Structural Coverage

- What happens if the requirements based tests do not satisfy structural coverage objectives?
- All tests must be requirements based.
  - i.e. can’t just write tests to cover gaps in structural coverage
- Options:
  - Write new requirements based test cases.
  - Write new requirements.
  - The code is dead (i.e. not used, not needed), remove it.
  - Deactivate the code: take measures to prevent execution and verify those measures.
    - Used when one software item can be used on multiple installations.

Architectural Coverage

- Software is a system: there are emergent properties.
- Testing software often requires test stubs and simulations.
  - i.e. simulated interfaces between software components
- The purpose of the architectural coverage measure is to assure that the real interfaces are sufficiently exercised during testing.
- To do this, need to exercise each of the following at least once using actual software modules:
  - Data Paths (passing of data from one software module to another).
  - Control Links (i.e. one software modules control over another).
  - Control Paths (passing of control from one to module to another).
Unit, Integration and System Level Testing

- DOD-STD-2167A and MIL-STD-498 split testing into unit, integration and system level.
- Testing to satisfy DO-178B objectives will be conducted as either unit, integration or system level testing.
  - But this is not the measure of completeness.
- Loose requirement in DO-178B: testing is to be conducted at the highest possible level.
  - i.e. can only rely on unit testing to achieve objectives if it is not possible to test at integration or system level.

Black Box vs. White Box Testing

- This is another view of testing.
- **Black Box**: Testing of the software with no insight into how it works, only know what it is supposed to do.
- **White Box**: Testing of the software based on how it is built.
- All DO-178B tests are black box tests.
  - i.e. they are all requirements based
- But, insight into how the software is built is required in order to demonstrate that testing is sufficient.
  - e.g. decision coverage, equivalence classes, etc require insight into how the software is built
  - Be wary of implicitly establishing requirements through test cases.
Hardware Compatibility

- Also need to test software for compatibility with target hardware.
  - Most testing will be conducted in a simulated environment.
  - Need to check that the actual target hardware satisfies any assumptions inherent in the design process.

- Particular concerns:
  - Hardware Transients and Failures
  - Built In Test Validation
  - Interface Errors
  - Control Loops
  - Resource Management (time, interrupts, memory)
  - Field Loading Operations
  - Protection Boundaries

Test Procedures

- Test Procedures should be defined and approved prior to the commencement of testing.
- Test Procedures should identify the expected outcome of the test.
- It is possible that there are errors in the test procedures:
  - Can be ‘redlined’ during testing if a simple process error.
  - Otherwise require raising of problem report.
Test Environments

• Credit can only be taken for tests conducted on the target hardware or in a qualified test environment.
• It must be shown that the test environment is sufficiently representative of the target environment for the test case to be valid.
• Requires test environment qualification procedures (i.e. tests to verify correct operation) and configuration management of the test environment.

Regression Analysis and Re-Verification

• Often, the ADF will take previously certified software and modify it slightly.
  – Note that previously certified is not the same as ‘off the shelf’.
• How much re-verification is required in these cases?
• Simple part first:
  – Obviously the newly developed and modified aspects of the software need to be verified.
• Hard part:
  – How much of the unchanged software needs to be re-verified in order to demonstrate that the retained functionality has not been negatively affected?
Regression Analysis and Re-Verification

- Need to conduct a regression analysis and determine which aspects of the retained software are linked to the new or modified software through:
  - functional, design and architectural links
  - processor, timing and memory dependencies
- Generally, need to re-verify any software that is directly linked to or dependent on new or modified software.
  - i.e. one step removed using the above links and dependencies
- Also need to consider whether the ADF installation and use invalidates any of the assumptions made during the previous certification.
  - What scope of use did the verification effort consider?
  - Has the severity of failure conditions increased?

Re-Verification or Re-Test?

- Note that the above slides have considered re-verification.
  - Not re-test or regression testing.
- The requirement is to re-verify any aspect of the software that may have been negatively affected (identified through regression analysis). This involves:
  - verification that software requirements are still appropriate
  - verification that the software design and architecture is still appropriate
  - verification that the source code is still appropriate
  - regression testing
Summary – A Check List for Test Completion

- Test Environments Qualified?
- Regression Analysis Demonstrates Sufficiency of Regression Testing?
- Test Procedures (including expected outcomes) Defined?
- Tests Conducted at the Highest Possible Level?
- Requirements Coverage Achieved?
  - Normal and Robustness Testing of Software Requirements (Level D and above)
  - Normal and Robustness Testing of Software Design (Level C and above)
- Structural Coverage Achieved?
  - Statement Coverage (Level C)
  - Decision Coverage (Level B)
  - Modified Condition/Decision Coverage (Level A)
- Architectural Coverage Achieved?
  - Each actual control and data link exercised (Level C and above)
- Compatibility with Target Hardware confirmed?
  - Testing sufficient to validate hardware assumptions (Level D and above)

Questions