CISC327 - Software Quality Assurance

Lecture 8

Introduction to Systematic Testing, part 1

Introduction to Systematic Testing

- Outline
  - Today we begin a thorough look at software testing
  - Definitions: What is software testing?
  - Role of specifications
  - Levels of testing:
    unit, integration, system, acceptance

What is Testing?

- Testing is the process of executing software in a controlled manner to answer a question:
  - "Does the software behave as specified?"
- Implies that we have a specification*
  - Possibly that the tests are the specification
- Or implies that we have some property we wish to test for independently of the specification
  - e.g., "All paths in the code are reachable, no dead code"
- Testing is often associated with the words validation and verification

* compare http://www.freeindiegam.es/2013/08/become-a-great-artist-in-just-10-seconds-andi-mcclure-michael-brough/

What is Systematic Testing?

- An explicit discipline or procedure (a system) for
  - choosing and creating test cases
  - executing the tests and documenting the results
  - evaluating the results, possibly automatically
  - deciding when we are done (enough testing)

What is Systematic Testing?

- Because in general it is impossible to ever test completely, each systematic method chooses a particular point of view and tests only from that point of view (the test criterion)
  - e.g., test only that every decision (if statement) can be executed either way

Verification vs. Validation

- Verification
  - The checking or testing of software (or anything else) for conformance and consistency with a given specification
    - Answers the question "are we doing the job right?"
  - Testing is most useful in verification, although it is just one part of it
    - Inspection, measurement, analysis and formal methods are also important

Verification vs. **Validation**

- Validation
  - The process of checking that what has been specified is what the user actually wanted
    - Answers the question "are we doing the right job?"
  - Validation usually involves meetings, reviews, and discussions to check that what has been specified is what was intended
  - Testing is less useful in validation, although it can have a role

Validation vs. Verification

- Verification
  - Check that the software meets its stated functional and non-functional requirements
- Validation
  - More general than verification, ensure that the software meets the customer's expectations
  - Requirements specifications do not always reflect the real wishes or needs of system customers and users

Testing vs. Debugging

- Debugging is not Testing
  - Debugging is the process of analyzing and locating bugs when the software does not behave as expected
  - Testing plays the much more comprehensive role of methodically searching for and exposing bugs, not just fixing those that happen to show up by playing with the software
  - Debugging therefore supports testing but cannot replace it
  - However, no amount of testing is guaranteed to find all bugs
    - (except possibly exhaustive testing, where practical)

Exhaustive Testing vs. ...

- Occasionally you **can** test exhaustively:
  - Let's write all the tests for a NAND ("not AND" or "Sheffer stroke") operation
  - Tests are just the same as the specification (truth table):
    - A B A NAND B
    - true true false
    - true false true
    - false true true
    - false false true
  - Only 4 possible combinations of A and B, so 4 test cases

Exhaustive Testing vs. …

- Most software, even most individual methods, can't be exhaustively tested
  - Examples:
    - There are infinitely many integers
    - Even a 32-bit integer (small these days) has ~4 billion possible values
    - Strings: way too many strings to test
  - Compilers are an extreme case: the set of possible inputs to a C compiler is the set of all C programs (and many "almost" C programs, e.g. programs with syntax errors)

The Role of Specifications

- The Need for Specification
  - Validation and verification activities, such as testing, cannot be meaningful unless we have a specification for the software
  - The software we are building could be a single module or class, or could be an entire system
  - Depending on the size of the project and the development methods, specifications can range from a single page to a complex hierarchy of interrelated documents

Levels of Specification

- Three Levels
  - Specifications of large systems usually contain at least three levels of software specification documents
  1. Functional specifications (or requirements)
  2. Design specifications
  3. Detailed design specifications

Levels of Specification

- 1) Functional specifications (requirements)
  - Give a precise description of the required behaviour (functionality) of the system
  - Describe what the software should do, not how it should do it
  - May also describe constraints on how this can be achieved
    - Example: When the user chooses the "Exit" menu item, bring up the "Save" dialog if the current document has not been saved, otherwise terminate the program

Levels of Specification

- 2) Design specifications
  - Describe the architecture of the design to implement the functional specification
  - Describe the components of the software and how they are to relate to one another
    - Example: A UML diagram and associated documentation describing the relationship between a document object and a spelling checker

Levels of Specification

- 3) Detailed design specifications
  - Describe how each component of the architecture, down to the individual code units, is to be implemented
    - Example: A detailed description of the Document object, including the data structures used to store the information, relationships with other objects, and so on

Levels of Testing

- Corresponding Test Levels
  - Given the hierarchy of specifications, it is usual to structure testing into three (or more) corresponding levels
    - **3)** (detailed design) **Unit** Testing
    - **2)** (design specifications) **Integration** Testing
    - **1)** (functional specifications) **System** Testing
  - To these levels, we usually add:
    - **0) Acceptance** Testing

Levels of Testing

- Corresponding Test Levels
  - 3) **Unit** testing addresses the verification that individual components of the architecture meet their detailed design specification
  - 2) **Integration** testing (a.k.a. component testing) verifies that the groups of units corresponding to architectural elements of the design specification can be integrated to work as a whole

Levels of Testing

- Corresponding Test Levels
  - 1) **System** testing verifies that the integrated total product has the functionality specified in the functional specification
  - 0) **Acceptance** testing, in which the actual customers validate that the software meets their real intentions as well as what has been functionally specified, and accept the result

An Integral Task: Tests as Goals

- Once each level of specification is written, the next step is to write the tests for that level
  - XP speeds this by making the tests themselves the specification
- It is important that the tests be designed without knowledge of the software implementation
  - In XP, before implementation
- Otherwise we are tempted to simply test the software for what it actually does, not what it should do

Using Tests

- Evaluating Tests
  - Within each level of testing, once the tests have been applied, test results are evaluated
  - If a problem is encountered, then either:
    a) the **tests are wrong**:
       the tests are revised and applied again, or
    b) the **software is wrong**:
       the software is fixed and the tests are applied again
  - In either case, the tests are applied again, and so on, until no more problems are found, at which point development can proceed to the next level of testing

Test Evolution

- Tests Don't Die!
  - Testing does not end when the software is accepted by the customer
  - Tests must be repeated, modified and extended to ensure that no existing functionality has been broken, and that any new functionality is implemented according to the revised specifications and design

Test Evolution

- Tests Don't Die!
  - Maintenance of the tests for a system is a major part of the effort to maintain and evolve a software system while retaining a high level of quality
  - To make this continual testing practical, automation plays a large role in software testing methods

Summary

- Introduction to Testing
  - Testing addresses primarily the verification that software meets its specifications
  - Without some kind of specification, we cannot test
  - Testing is done at several levels, corresponding to the levels of functional, design, and detailed specifications in reverse order
  - Testing is not finished at acceptance, it remains for the life of the software system

Summary

- References
  - Sommerville, ch. 8, "Software Testing"
  - The Software Test Page (on the web)