Introduction to Testing

Davide Falessi

Introduction

- Testing software is a very important and **challenging** activity.
- Testing is the evaluation of the work products created during a software development effort.
- Testing software well has always been challenging, but the process is fairly well understood.
- The process usually consist of a **combination** of different types of testing.

What is software testing

- Software testing (or just "testing") is the process of uncovering evidence of defects in software systems.
- Testing **comprises** the efforts to find defects.
- Testing **does not include** efforts associated with tracking down defects and fixing them.
  - In other words, testing does *not* include the debugging or repair of bugs

Definitions

- Failure: unexpected system behavior.
- Bugs == Defects: something wrong in the code which causes a failure.

WHY TESTING

- Testing is important because it substantially contributes to ensuring that a software application does **everything** it is supposed to do.
- Some testing efforts extend the focus to ensure an application does **nothing more than** it is supposed to do.

Testing vs. QA

- QA addresses activities designed to **prevent** defects as well as to **remove** those defects that do creep into the product.

Best practices

- Test early
- Test often
- Test the right part in the right way

Test Early

- Instead of engaging system testers toward the end of a project, start them testing at reasonable points during the analysis and design phases of a project.
- Testing analysis and design models not only can help to uncover problems early in the development process (where they are fixed more easily and more cheaply), but it can also help to scope the size of the effort needed to perform adequate system testing by determining what needs to be tested.
- Testing early and often implies that the representations of the software are abstract or incomplete.

Test Often

- An iterative, incremental— sometimes also referred to as **iterative enhancement**— development process is best suited to the vast majority of projects.
- As iterations are completed on analysis, design, and implementation phases, the products should be tested.
- After completion of the first increment, some testing takes the form of regression testing.

Keywords

- Test Case
- Inputs
- Expected Results
- Test Case Execution
- Test suites
- Adequacy
- Coverage
- Black-box
- Structural

Test Cases

- The basic component of testing is a test case. In its most general form, a **test case** is a pair *(input, expected result),* in which *input* is a description of an input to the software under test and *expected result* is a description of the output that the software should exhibit for the associated input.

Inputs

- Inputs and expected results are not necessarily simple data values, such as strings or integer values, but they can be arbitrarily complex.
- Inputs often incorporate system state information as well as user commands and data values to be processed.

Expected result

- ***Expected result*** includes not only perceivable things, such as printed reports, audible sounds, or changes in a display screen, but changes to the software system itself— for example, an update to a database or a change in a system state that affects processing of subsequent inputs.

Test Case Execution

- A **test case execution** is a running of the software that provides the inputs specified in the test case and observes the results and compares them to those specified by the test case.
- If the actual result varies from the expected result, then a failure has been detected and we say the software under test "**fails the test case**.“
- If the actual result is the expected result for a test case, then we say the software "**passes the test case**."

Test Suite

- Test cases are organized into a **test suite.**
- Most test suites have some sort of organization based on the kinds of test cases.
  - E.g., a test suite might have one part containing test cases that are concerned with testing system capacities and another part containing test cases concerned with testing typical uses of the system well within any specified capacities.
- If software passes all the test cases in a test suite, then we say that the software "**passes the test suite**."

Adequacy of test cases

- From practical and economic perspectives, **testing software completely is impossible**.
- A reasonable goal for testing is to develop enough test cases to ensure that the software exhibits no failures in typical uses or in life-critical situations.
- This captures the idea of **adequacy** of testing a software product.
  - Test it enough to be reasonably sure the software works as it is supposed to.

Coverage

- Adequacy can be measured based on the concept of **coverage.**
- Coverage can be measured in at least two ways.
  - In terms of **how many of the requirements** called out in the specification are tested. Of course, some requirements will require many test cases.
  - In terms of **how much of the software** itself was executed as a result of running the test suite. A test suite might be adequate if some proportion of the lines of source code— or possible execution paths through the source code— was executed at least one time during test suite execution.

Black box testing

- In **functional testing,** which is also referred to as **specification-based** or **black box testing,** test cases are constructed based solely on the software's specification and not on how the software is implemented.
- This approach is useful for all levels of testing because it has the advantage that test cases can be developed even before coding begins.
- However, the effectiveness of functional testing depends greatly on the quality of the specification and the ability of the test suite developers to interpret it correctly.

STRUCTURAL TESTING

- In **structural testing,** which is also referred to as **implementation-based** or **white box testing,** test cases are constructed based on the code that implements the software.
- The output of each test case must be determined by the software's specification, but the **inputs can** be determined from analyzing the code itself to determine various values that cause various execution paths to be taken.
- The main advantage of this approach is improved coverage.
- The main disadvantage is that if the programmer did not implement the full specification, then that part of the functionality will not be tested.