Testing Methodologies

Muntazir Fadhel

October 18, 2016

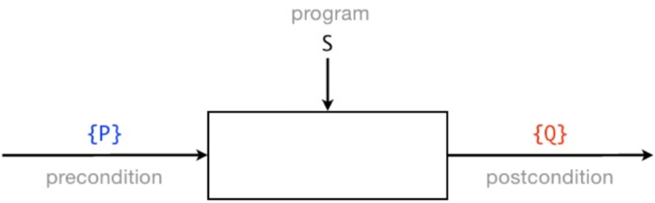
Comparing Testing and Verification

Testing

Involves running the program on selected inputs and comparing the result with the expected result.

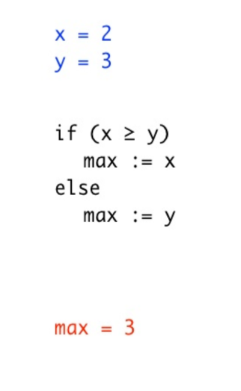
Verification

The argument that the program works as expected for all inputs.



Example: Find the maximum value of two natural numbers

A Testing Activity

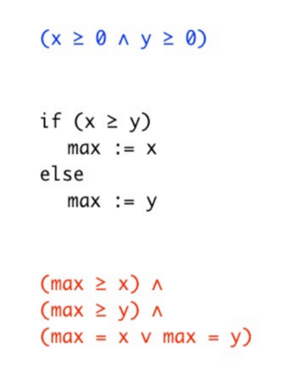


```
x = 2
y = 3

if (x ≥ y)
   max := x
else
   max := y


max = 3
```

A Verification Activity



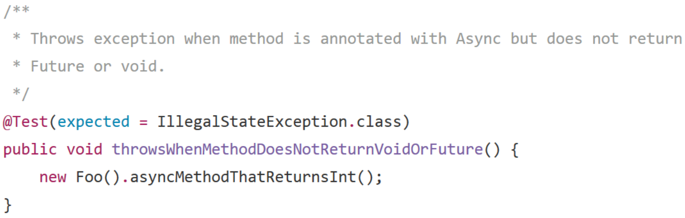
```
(x ≥ 0 ∧ y ≥ 0)


if (x ≥ y)
   max := x
else
   max := y


(max ≥ x) ∧
(max ≥ y) ∧
(max = x ∨ max = y)
```

If verification is more preferable, why bother with testing?

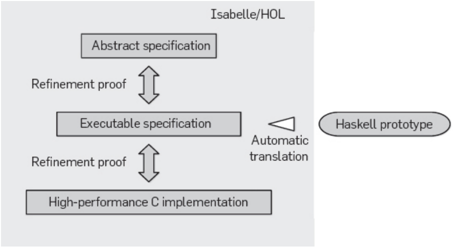
- Verification can only show correctness relative to a model.
- Some non-functional requirements cannot be verified.
- Verification is typically impractical for larger systems, even with tooling support.
- Because we don't necessarily need a correct program, only one that we are reasonably confident in.
- Testing often provides useful, **readable** documentation for software maintainers.



```
/**
 * Throws exception when method is annotated with Async but does not return
 * Future or void.
 */
@Test(expected = IllegalStateException.class)
public void throwsWhenMethodDoesNotReturnVoidOrFuture() {
    new Foo().asyncMethodThatReturnsInt();
}
```

The SEL4 Micro Kernel

The seL4 micro kernel was formally verified in 2009.



- Contained 10 000 lines of C code, it took twelve person years to verify. [1]
  - Required 200 000 lines worth of Isabelle/HOL proof scripts!
- Verification assumed correct functioning of hardware, hardware management, virtual memory, etc.

What To Test Against

Note

Test cases must be developed against a previously determined specification.

Example 1: Testing of the program against a pre- and post condition.

function max(Integer **x**, Integer **y,** Integer **result**)

Condition	Result
x >= y	**result = x**
x < y	**result = y**

$$\{x >= y\}\ \mathbf{max(x,\ y\ ,\ result)}\ \{result = x\}$$

- Input: x = 6, y = 5
- Output: result = 5, x unchanged, y unchanged, test /alertfailed

Example 2: Testing of the program against an abstract statement.

$$(result >= x) \wedge (result >= y) \wedge (result = x \vee result = y)$$

- Input: x = 6, y = 5
- Output: result = 6, x unchanged, y unchanged, test passed.

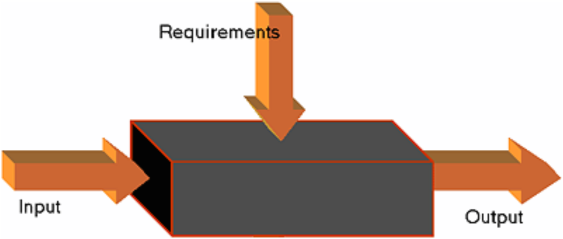
We evaluate the predicate of the specification statement to determine whether the test failed or passed.

Black Box Testing

Definition

Tests are derived from specification of the program without making use of the program's internals.

- **Software requirements** serve as the specification against which test cases are developed.
- Should be the emphasis for testers/quality assurance.
- Code access is not required.



White Box Testing

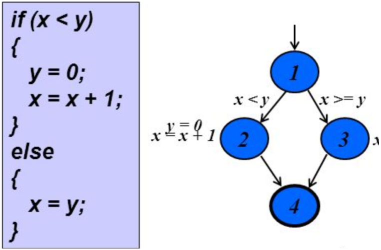
Definition

Tests are derived based on the internal structure of the program.

- **Software implementation** serves as the specification against which test cases are developed.
- Should be the emphasis for developers.
- Should ensure that all possible program paths, statements, loops and conditions are thoroughly tested.

Example

Branch Testing: ensures that each one of the possible branches from each decision point is tested.



```
if (x < y)
{
   y = 0;
   x = x + 1;
}
else
{
   x = y;
}
```

Selecting Test Cases

- Criteria:
  1. Test cases that reflect the normal operation of a program.
  2. Test cases that reflect areas of the program where problems are expected to arise.

Partition Testing

Choose test cases from groups of inputs that should be processed in the same way.

Guideline-Based Testing

The guidelines are based on testing core functionality of the software, components that have frequent defects or have undergone recent changes.

Common Testing Strategies

Unit Testing

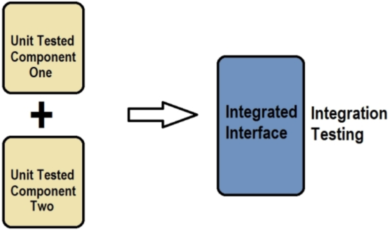
Tests small individual units of the program for proper operation.

- Encourage development of code adhering to the Single Responsibility Principle.
- Execute quickly thereby offering developers rapid feedback.

Integration Testing

Tests the interfaces between the units/modules for correct operation.

- Generally require more resources to execute in comparison to unit tests.
- Should be performed after unit tests have been developed.

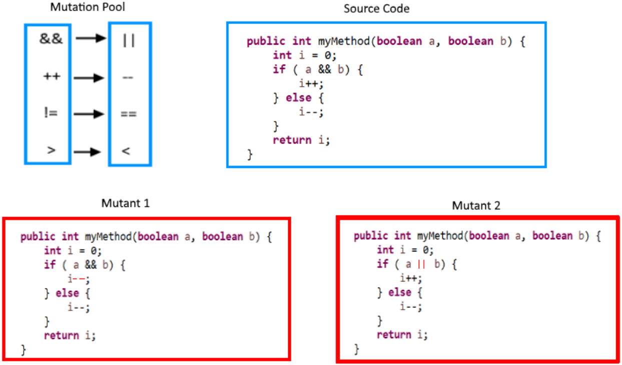


Mutation Testing

Definition

Inserts faults into the programs to test whether the tests pick them up thereby validating or invalidating existing tests.

1. Program is analyzed to generate mutants.
2. Mutants are run against existing test cases.
3. Passing tests indicate undetected mutants and testing gaps.



Mutation Pool

Source Code

```
public int myMethod(boolean a, boolean b) {
    int i = 0;
    if ( a && b) {
        i++;
    } else {
        i--;
    }
    return i;
}
```

Mutant 1

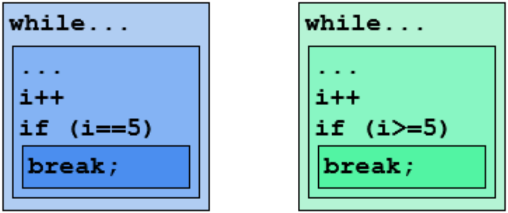
```
public int myMethod(boolean a, boolean b) {
    int i = 0;
    if ( a && b) {
        i--;
    } else {
        i--;
    }
    return i;
}
```

Mutant 2

```
public int myMethod(boolean a, boolean b) {
    int i = 0;
    if ( a || b) {
        i++;
    } else {
        i--;
    }
    return i;
}
```

Mutation Testing Discussion

- Quality of existing tests can be gauged from the percentage of mutations killed.
- Generally increases testing time by many factors.
- Requires manual verification of test output.
- How do we deal with Mutant Equivalence?



```
while...
  ...
  i++
  if (i==5)
    break;
```

```
while...
  ...
  i++
  if (i>=5)
    break;
```

Fuzzy Testing Overview

Definition

Input large amounts of random data in attempt to make the system fail or behave unexpectedly.

Mutation-Based

- Samples of valid input are mutated randomly to produce malformed input.

Generation-Based

- Generate input from scratch rather than mutating existing input.

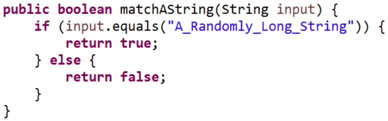
Evolutionary

- Uses feedback from each test case to learn the format of the input over time.

Fuzzy Testing Discussion

- Has been successfully used in industry to detect important bugs.
- Work best for problems that can cause a program to crash such as buffer overflow, cross-site scripting and denial of service attacks.
- Programs with complex inputs can require much more work to produce a smart enough fuzzer to get sufficient code coverage.

- Needs to be supplemented with white box testing!

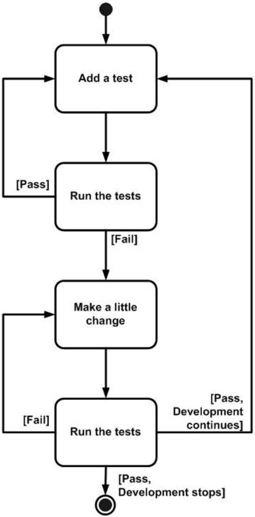


```java
public boolean matchAString(String input) {
    if (input.equals("A_Randomly_Long_String")) {
        return true;
    } else {
        return false;
    }
}
```

Test Driven Development

A **software development process** that relies on the repetition of a very short development cycle:

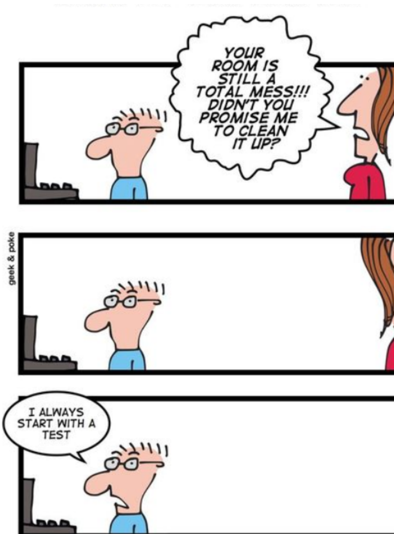
1. Requirements are turned into very specific test cases.
2. Software is improved to pass the new tests, only.



Test Driven Development

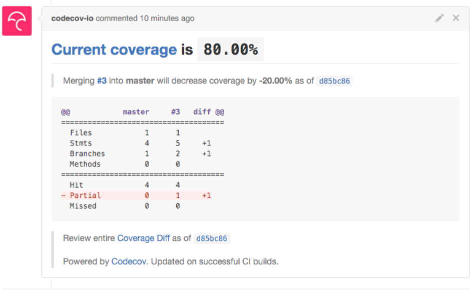
- Forces you to think about the interface to the code first.
- Keep developers focused on writing code that matches the software's original specifications.
- Gives developers confidence in making changes to the software.

- Does TDD improve software design?
  - Studies show test-first programmers are more likely to write software in more and smaller units that are less complex than non-TDD programmers [2].
  - At the end of the day, TDD doesn't create design. You do.


YOUR ROOM IS STILL A TOTAL MESS!!! DIDN'T YOU PROMISE ME TO CLEAN IT UP?
geek & poke
I ALWAYS START WITH A TEST

Concluding Thoughts on Testing

- Tests on their own do not increase confidence in the program, running test do!
  - Always try to automate test cases and run them often.
- Best results are achieved by a careful combination of verification and testing activities.
- Be wary of **"coverage"** statistics, not always an accurate indication of how well the software is tested!



codecov-io commented 10 minutes ago

Current coverage is 80.00%

Merging **#3** into **master** will decrease coverage by **-20.00%** as of `d85bc86`

```
@@            master     #3   diff @@
=====================================
  Files              1      1
  Stmts              4      5     +1
  Branches           1      2     +1
  Methods            0      0
=====================================
  Hit                4      4
- Partial            0      1     +1
  Missed             0      0
```

Review entire Coverage Diff as of `d85bc86`

Powered by Codecov. Updated on successful CI builds.

For Further Reading I

- Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. 2009. seL4: formal verification of an OS kernel. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP '09). ACM, New York, NY, USA, 207-220. DOI=http://dx.doi.org/10.1145/1629575.1629596
- D. Janzen and H. Saiedian, ”Does Test-Driven Development Really Improve Software Design Quality?,” in IEEE Software, vol. 25, no. 2, pp. 77-84, March-April 2008. doi: 10.1109/MS.2008.34
- P. Runeson, ”A Survey of Unit Testing Practices,” IEEE Software, no. pp. 22-29, July/Aug 2006.

For Further Reading II

- Williams, L. (n.d.). WhiteBox Testing. Realsearch.
- N. Nagappan, E. M. Maximilien, T. Bhat, and L. Williams, ”Realizing Quality Improvement Through Test Driven Development: Results and Experiences of Four Industrial Teams,” Empirical Software Engineering, vol. 13, no. 3, pp. 289-302, June 2008.
- Jia, Y., Harman, M. (n.d.). An Analysis and Survey of the Development of Mutation Testing. IEEE Transactions on Software Engineering
- Sekerinski, E. (2012). Notes on Software Design: Testing. McMaster University.

For Further Reading III

- Ahamed, R., Dr. (2009). Studying The Feasibility and Importance of Software Testing: An Analysis (Vol. 1(3)). International Journal of Engineering Sciences and Technology.
- E. H. Spafford, Extending Mutation Testing to Find Environmental Bugs, Software:Practice and Experience, vol. 20, no. 2, pp. 181189, February 1990
- K. Beck, Test Driven Development – by Example. Boston: Addison Wesley, 2003
- C.-w. Ho, M. J. Johnson, L. Williams, and E. M. Maximilien, ”On Agile Performance Requirements Specification and Testing,” in Agile 2006, Minneapolis, MN, 2006, pp. 47-52.