Software Testing, Quality Assurance and Maintenance Winter 2015

Lecture 4 — January 12, 2015

Patrick Lam *version 1*

About Testing

We can look at testing statically or dynamically.

Static Testing (ahead-of-time): this includes static analysis, which is typically automated and runs at compile time (or, say, nightly), as well human-driven static testing—walk-throughs (informal) and code inspection (formal).

Dynamic Testing (at run-time): observe program behaviour by executing it; includes black-box testing and white-box testing.

Usually the word "testing" means *dynamic testing*.

Naughty words. People like to talk about "complete testing", "exhaustive testing", and "full coverage". However, for many systems, the number of potential inputs is infinite. It's therefore impossible to completely test a nontrivial system, i.e. run it on all possible inputs. There are both practical limitations (time and cost) and theoretical limitations (i.e. the halting problem).

In the absence of complete testing, we will define *testing criteria* and evaluate test suites with them.

Test cases

Informally, a *test case* contains:

- what you feed to software; and
- what the software should output in response.

Here are two definitions to help evaluate how hard it might be to create test cases.

Definition 1 Observability *is how easy it is to observe the system's behaviour, e.g. its outputs, effects on the environment, hardware and software.*

Definition 2 Controlability *is how easy it is to provide the system with needed inputs and to get the system into the right state.*

Anatomy of a Test Case

Consider testing a cellphone from the "off" state:

$$\underbrace{\langle \text{ on } \rangle}_{\text{prefix values}} \quad \underbrace{1\ 519\ 888\ 4567}_{\text{test case values}} \quad \underbrace{\underbrace{\langle \text{ talk } \rangle}_{\text{verification values}} \quad \underbrace{\langle \text{ end } \rangle}_{\text{exit codes}}}_{\text{postfix values}}$$

Definition 3

- Test Case Values*: input values necessary to complete some execution of the software. (often called the test case itself)*
- Expected Results*: result to be produced iff program satisfies intended behaviour on a test case.*
- Prefix Values*: inputs to prepare software for test case values.*
- Postfix Values*: inputs for software after test case values;*
  - verification values*: inputs to show results of test case values;*
  - exit commands*: inputs to terminate program or to return it to initial state.*

Definition 4

- Test Case*: test case values, expected results, prefix values, and postfix values necessary to evaluate software under test.*
- Test Set*: set of test cases.*
- Executable Test Script*: test case prepared in a form to be executable automatically and which generates a report.*

On Coverage

Ideally, we'd run the program on the whole input space and find bugs. Unfortunately, such a plan is usually infeasible: there are too many potential inputs.

Key Idea: Coverage. Find a reduced space and cover that space.

We hope that covering the reduced space is going to be more exhaustive than arbitrarily creating test cases. It at least tells us when we can plausibly stop testing.

The following definition helps us evaluate coverage.

Definition 5 *A* test requirement *is a specific element of a (software) artifact that a test case must satisfy or cover.*

We write TR for a set of test requirements; a test set may cover a set of TRs.

For instance, consider three ice cream cone flavours: vanilla, chocolate and mint. A possible test requirement would be to test one chocolate cone. (Volunteers?)

Two software examples:

- cover all decisions in a program (branch coverage); each decision gives two test requirements: branch is true; branch is false.
- each method must be called at least once; each method gives one test requirement.

Definition 6 *A* coverage criterion *is a rule or collection of rules that impose test requirements on a test set.*

A test set may or may not satisfy a coverage criterion. The coverage criterion gives a recipe for generating TRs systemically.

Returning to the ice cream example, a flavour criterion might be "cover all flavours", and that would generate three TRs: {flavour: chocolate, flavour: vanilla, flavour: mint}.

We can test an ice cream stand by running two test sets on it, for instance: test set 1 includes 3 chocolate cones and 1 vanilla cone, while test set 2 includes 1 chocolate cone, 1 vanilla cone, and 1 mint cone.

Definition 7 *(Coverage). Given a set of test requirements TR for a coverage criterion C, a test set T satisfies C iff for every test requirement $tr \in TR$, at least one $t \in T$ satisfies TR.*

Infeasible Test Requirements. Sometimes, no test case will satisfy a test requirement. For instance, dead code can make statement coverage infeasible, e.g.:

```
if (false)
  unreachableCall();
```

or, a real example from the Linux kernel:

```
while (0)
  {local_irq_disable();}
```

Hence, a criterion which says "test every statement" is going to be infeasible for many programs.

Quantifying Coverage. How good is a test set? It's great if it covers everything, but sometimes that's impossible. We can instead assign a number.

Definition 8 *(Coverage Level). Given a set of test requirements TR and a test set T, the coverage level is the ratio of the number of test requirements satisfied by T to the size of TR.*

Returning to our example, say TR = {flavour: chocolate, flavour: vanilla, flavour: mint}, and test set T1 contains {3 chocolate, 1 vanilla}, then the coverage level is "2/3" or about 67%.

Subsumption

Sometimes one coverage criterion is strictly more powerful than another one: any test set that satisfies C_1 might automatically satisfy C_2.

Definition 9 *Criteria subsumption: coverage criterion* C_1 subsumes C_2 *iff* every *test set that satisfies* C_1 *also satisfies* C_2*.*

Software example: branch coverage ("Edge Coverage") subsumes statement coverage ("Node Coverage"). Which is stronger?

Evaluating coverage criteria. Subsumption is a rough guide for comparing criteria, but it's hard to use in practice. Consider also:

1. difficulty of generating test requirements;
2. difficulty of generating tests;
3. how well tests reveal faults.