Software Testing, Quality Assurance and Maintenance Winter 2010

Lecture 2 — January 6, 2010

Patrick Lam *version 1*

RIP Fault Model

Recall that a *fault* is something that's latent or hiding, while a *failure* is visible (e.g. EPIC FAIL).

To get from a fault to a failure:

1. Fault must be *reachable*;
2. Program state subsequent to reaching fault must be incorrect: *infection*; and
3. Infected state must *propagate* to output to cause a visible failure.

Applications of the RIP model: automatic generation of test data, mutation testing.

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:

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

Definition 3

- Test Case Values*: input values necessary to complete some execution of the software.*
- 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.

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.

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.*

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.*

(An example of a coverage level might be "5/6", if there are 6 test requirements and a test set satisfies 5 of them.)

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();
```

Coverage levels help us evaluate the goodness of a test set, especially in the presence of infeasible test requirements.

Example. Here is a non-software example of test requirements:

> MATH 117: Functions of engineering importance; review of polynomial, exponential and logarithmic functions; trigonometric functions and identities. Inverse functions (logarithmic and trigonometric). Limits and continuity. Derivatives, rules of differentiations, derivatives of elementary functions. ...

In this case, a test set is the set of questions on a final exam. (Note that a final exam usually won't cover the whole syllabus; also, we'd usually think of test requirements as, for instance, "a partial fraction expansion" or "definition of the derivative", etc).

Running the test set then consists of giving the exam to students.

Here are some examples of non-exhaustive coverage criteria:

- student understands how to compute and use limits;
- student can solve arbitrary word problems (could subsume criterion of being able to solve all min-max problems).