SE 465: Testing

Michael Noukhovitch

Winter 2015, University of Waterloo

Notes written from Patrick Lam's lectures.

Contents

1 Introduction

1.1 Types of Problems

- **fault**: static defect in the software
  - **design fault**
  - **mechanical fault**
- **error**: have incorrect state
- **failure**: external incorrect behavior

Example 1.1. Faults

```
static public int findLast (int[] x, int y) {
    for (int i=x.length-1; i>0; --i){
        if (x[i] == y){
            return i;
        }
    }
    return -1;
}
```

fault: should be `i >= 0`
no **fault** input: `x = null`
fault but not **error** input: `x[0] != y`
error but not **failure** input: `y not in x`

1.2 RIP model

RIP model: three things necessary to observe a failure

1. **Reachability**: PC must reach that point in the program
2. **Infection**: after fault, program state must be incorrect
3. **Propogation**: infected state propogates to cause bad output

1.3 Dealing with faults

We have three ways to deal with faults:

- **avoidance**: design, use better language
- **detection**: testing
- **tolerance**: redundancy, isolation

2 Testing

2.1 Testables

- code coverage
- output of a function
- logic coverage
- input space coverage

2.2 Types of testing

static testing: testing without running the code

- compilation
- semantic verification
- code reviews

dynamic testing: testing by running and observing the code

- **test cases**: single input, single output (wrt to some code)
- **black-box testing**: don't look at system implementation
- **white-box testing**: base tests on system's design

2.3 Coverage

We find a reduced space and cover that space with our tests
test requirement: a specific element (of software) that a test case must satisfy or cover
infeasable test requirement: impossible coverage e.g. unreachable code
subsumption: when one testing criterion is strictly more powerful than another criterion

3 Graph Coverage

test path: considering our test as some path through our program from some initial node in N_0, along different nodes that ends up at a final node in N_f
subpath: a path which is a subsequence of a path

3.1 Behaviours

- **deterministic**: 1 test path per test case
- **non-deterministic**: multiple test paths are possible

3.2 Reachability

- **syntactically**: reachable via edges and nodes
- **semantically**: there exist input that gets to a certain node

3.3 Coverage Criterion

Node Coverage: for every statement (node), there must be a test case that executes it
Edge Coverage: for every branch (edge), there must be a test case that goes through it
Edge-Pair Coverage: for every path of length up to 2, there must be a test case that goes through it