Fault-Based Testing
Learning objectives

• Be able to explain the basic ideas of fault-based testing
  • How to use knowledge of a fault model can be used to create useful tests and judge the quality of test cases
  • Explain the rationale of fault-based testing well enough to distinguish between valid and invalid uses

• Be able to explain and use mutation testing as one application of fault-based testing principles
Let’s count marbles ... a lot of marbles

• Suppose we have a big bowl of marbles. How can we estimate how many?
  • I don’t want to count every marble individually
  • I have a bag of 100 other marbles of the same size, but a different color
  • What if I mix them?

Photo credit: (c) KaCey97007 on Flickr, Creative Commons license
Estimating marbles

- I mix 100 black marbles into the bowl
  - Stir well ...
- I draw out 100 marbles at random
- 20 of them are black
- How many marbles were in the bowl to begin with?
Estimating Test Suite Quality

- Now, instead of a bowl of marbles, I have a program with bugs
- I add 100 new bugs
  - Assume they are exactly like real bugs in every way
  - I make 100 copies of my program, each with one of my 100 new bugs
- I run my test suite on the programs with seeded bugs ...
  - ... and the tests reveal 20 of the bugs
    - (the other 80 program copies do not fail)
- What can I infer about my test suite?
Basic Assumptions

• We’d like to judge effectiveness of a test suite in finding real faults, by measuring how well it finds seeded fake faults.

• Valid to the extent that the seeded bugs are representative of real bugs
  • Not necessarily identical (e.g., black marbles are not identical to clear marbles); but the differences should not affect the selection
    • E.g., if I mix metal ball bearings into the marbles, and pull them out with a magnet, I don’t learn anything about how many marbles were in the bowl
Mutation testing

- A mutant is a copy of a program with a mutation
- A mutation is a syntactic change (a seeded bug)
  - Example: change \( i < 0 \) to \( i <= 0 \)

- Run test suite on all the mutant programs
- A mutant is killed if it fails on at least one test case

- If many mutants are killed, infer that the test suite is also effective at finding real bugs
What do I need to believe?

• Mutation testing uses seeded faults (syntactic mutations) as black marbles

• Does it make sense? What must I assume?
  • What must be true of black marbles, if they are to be useful in counting a bowl of pink and red marbles?
Mutation testing assumptions

• Competent programmer hypothesis:
  • Programs are nearly correct
    • Real faults are small variations from the correct program
    • => Mutants are reasonable models of real buggy programs

• Coupling effect hypothesis:
  • Tests that find simple faults also find more complex faults
    • Even if mutants are not perfect representatives of real faults, a test suite that kills mutants is good at finding real faults too
Mutation Operators

• Syntactic change from legal program to legal program
  • So: Specific to each programming language. C++ mutations don’t work for Java, Java mutations don’t work for Python

• Examples:
  • crp: constant for constant replacement
    • for instance: from (x < 5) to (x < 12)
    • select from constants found somewhere in program text
  • ror: relational operator replacement
    • for instance: from (x <= 5) to (x < 5)
  • vie: variable initialization elimination
    • change int x =5; to int x;
Live Mutants

• Scenario:
  • We create 100 mutants from our program
  • We run our test suite on all 100 mutants, plus the original program
  • The original program passes all tests
  • 94 mutant programs are killed (fail at least one test)
  • 6 mutants remain alive

• What can we learn from the living mutants?
How mutants survive

• A mutant may be equivalent to the original program
  • Maybe changing \((x < 0)\) to \((x <= 0)\) didn’t change the output at all! The seeded “fault” is not really a “fault”.
    • Determining whether a mutant is equivalent may be easy or hard; in the worst case it is undecideable

• Or the test suite could be inadequate
  • If the mutant could have been killed, but was not, it indicates a weakness in the test suite
  • But adding a test case for just this mutant is a bad idea. We care about the real bugs, not the fakes!
Variations on Mutation

• Weak mutation
• Statistical mutation
Weak mutation

• Problem: There are lots of mutants. Running each test case to completion on every mutant is expensive
  • Number of mutants grows with the square of program size

• Approach:
  • Execute meta-mutant (with many seeded faults) together with original program
  • Mark a seeded fault as “killed” as soon as a difference in intermediate state is found
    • Without waiting for program completion
    • Restart with new mutant selection after each “kill”
Statistical Mutation

• Problem: There are lots of mutants. Running each test case on every mutant is expensive
  • It’s just too expensive to create $N^2$ mutants for a program of $N$ lines (even if we don’t run each test case separately to completion)

• Approach: Just create a random sample of mutants
  • May be just as good for assessing a test suite
    • Provided we don’t design test cases to kill particular mutants (which would be like selectively picking out black marbles anyway)
In real life ...

- Fault-based testing is a widely used in semiconductor manufacturing
  - With good *fault models* of typical manufacturing faults, e.g., “stuck-at-one” for a transistor
  - But fault-based testing for *design errors* is more challenging (as in software)
- Mutation testing is not widely used in industry
  - But plays a role in software testing research, to compare effectiveness of testing techniques
  - BUT, there are more tools like Stryker, Mutatest, and PIT that make mutation testing more easily adoptable.
- Some use of fault models to design test cases is important and widely practiced
Summary

• If bugs were marbles ...  
  • We could get some nice black marbles to judge the quality of test suites

• Since bugs aren’t marbles ...  
  • Mutation testing rests on some troubling assumptions about seeded faults, which may not be statistically representative of real faults

• Nonetheless ...  
  • A model of typical or important faults is invaluable information for designing and assessing test suites