Mutation Testing in UTP

Introduction

Bernhard K. Aichernig

Institute for Software Technology
Graz University of Technology
Graz, Austria

UNU-IIST: United Nations University
International Institute for Software Technology
Macau S.A.R. China

PSSE 2007

Outline

- Introduction
- Unifying Theories of Programming (UTP)
- Formalization: Faults and Test Cases
- Mutation Testing for Pre-postcondition Contracts
- Mutation Testing for Programs
- Mutation Testing for Protocol Specifications

Mutation Testing

- Mutation Testing focuses on faults
  - not on structural coverage (e.g. cover all statements)
- Basic idea:
  1. anticipate faults
  2. design test cases that would detect such faults
  3. run these tests to detect such faults
- we model faults on the specification level
  - by mutating the specification text

Aim

generate test cases preventing programs from implementing mutated (faulty) specifications

Triangle Example: Original in OCL

```
context Ttype(a: int, b: int, c: int): String
pre:  a < (b+c) and b < (a+c) and c < (a+b)
post: if((a = b) and (b = c))
      then result = "equilateral"
      else
        if ((a = b) or (a = c) or (b = c))
        then result = "isosceles"
        else result = "scalene"
        endif
      endif
```

Test case

$$a = 2,\ b = 2,\ c = 1,\ (\textit{expected})\ \textit{result} = \text{“}\textit{isosceles}\text{”}$$

Triangle Example: Mutant

```
context Ttype(a: int, b: int, c: int): String
pre:  a < (b+c) and b < (a+c) and c < (a+b)
post: if((a = a) and (b = c))
      then result = "equilateral"
      else
        if ((a = b) or (a = c) or (b = c))
        then result = "isosceles"
        else result = "scalene"
        endif
      endif
```

Test case :(

$$a = 2,\ b = 2,\ c = 1,\ (\textit{expected})\ \textit{result} = \text{“}\textit{isosceles}\text{”}$$

Triangle Example: Mutant

```
context Ttype(a: int, b: int, c: int): String
pre:  a < (b+c) and b < (a+c) and c < (a+b)
post: if((a = a) and (b = c))
      then result = "equilateral"
      else
        if ((a = b) or (a = c) or (b = c))
        then result = "isosceles"
        else result = "scalene"
        endif
      endif
```

Fault-detecting test case :)

$$a = 1,\ b = 2,\ c = 2,\ (\textit{expected})\ \textit{result} = \text{“}\textit{isosceles}\text{”}$$

Mutations

- Error guessing is a common strategy of testers who are domain experts.
- Claim: with a model more systematic way of error guessing
- Kinds of fault injection (mutation):
  - automatically by a set of mutation operators,
  - manually by interactively altering the specification.
- one fault per mutant

Test Hypothesis

Assumption

- We can anticipate the errors possibly made during implementation and
- are able to represent the faults in a given model

Dijkstra

Testing can never show the absence of faults but only their presence.

Our Reply

Testing can show the absence of faults, if we have a knowledge of what can go wrong.

Test Hypothesis

Assumption

- We can anticipate the errors possibly made during implementation and
- are able to represent the faults in a given model

Dijkstra

Testing can never show the absence of faults but only their presence.

Our Reply

Testing can show the absence of faults, if we have a knowledge of what can go wrong.

Naming the Incorrect

IEEE[1] classified in 1990 as follows in order to name the incorrect:

error: mistake made by a developer; located in people's head; may lead to one or more

fault, defect: flaw in the *software* with the potential to cause a

failure: manifestation of one or more faults, detected by comparing expected vs. actual observations.

Many call it simply "bug".

[1] Institute of Electronics and Electrical Engineering

Questions

Interesting questions when focusing on faults:

- Does an error made by a designer or programmer lead to an observable fault?
- Do my test cases detect such faults?
- How do I find a test case that uncovers a certain fault?
- What are the equivalent test cases that would uncover such a fault?
- How to automatically generate test cases that will reveal certain faults?

A theory will help answering them: UTP