Softwaretechnik

Lecture 08: Testing and Debugging — Overview

Peter Thiemann

University of Freiburg, Germany

SS 2012

Literature

Essential Reading

- *Why Programs Fail: A Guide to Systematic Debugging*, A Zeller
- *The Art of Software Testing*, 2nd Edition, G J Myers

Further Reading

- *Code Complete*, 2nd Edition, S McConnell

Cost of Software Errors

$ 60 billion

yearly cost of software errors for US economy [NIST 2002]

Cost of Software Errors

$ 180 billion

total sales of software in 2000

697,000 software engineers & 585,000 computer programmers

Cost of Software Errors

estimated

50%

of each software project spent on testing

(spans from 30% to 80%)

Cost of Software Errors

very rough approximation

money spent on testing	$\approx$	cost of remaining errors

Cost of Software Errors

very rough approximation

money spent on testing + cost of remaining errors

=

66% of size of software industry

A Quiz About Testing

A simple program

Input

Read three integer values from the command line.
The three values represent the lengths of the sides of a triangle.

Output

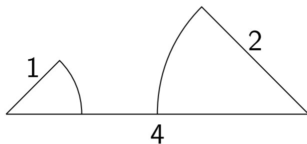
Tells whether the triangle is

Scalene:	no two sides are equal
Isosceles:	exactly two sides are equal
Equilateral:	all sides are equal

Task: Create a Set of Test Cases for this Program

Solution — 1 Point for each Correct Answer

Q 1: (4,1,2) a invalid triangle



Why not a valid triangle? (a,b,c) with $a > b + c$

Define valid triangles: $a \leq b + c$

Solution — 1 Point for each Correct Answer

Q 2: some permutations of previous (1,2,4), (2,1,4)

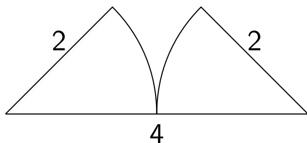
Fulfill above definition, but are still invalid.

Patch definition of valid triangles:

$a \leq b + c$ and $b \leq a + c$ and $c \leq a + b$

Solution — 1 Point for each Correct Answer

Q 3: (4,2,2) a invalid triangle with equal sum



Fulfills above definition, but is invalid (depending on what we want!).

Patch definition of valid triangles:

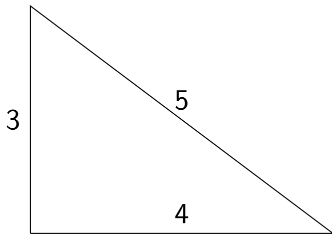
$a < b + c$ and $b < a + c$ and $c < a + b$

Solution — 1 Point for each Correct Answer

Q 4: some permutations of previous (2,2,4), (2,4,2)

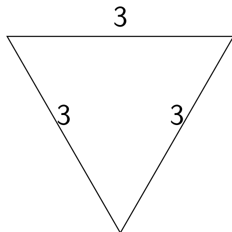
Solution — 1 Point for each Correct Answer

Q 5: (3,4,5) a valid scalene triangle



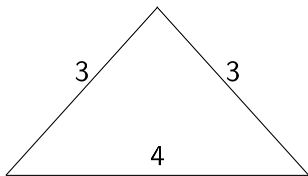
Solution — 1 Point for each Correct Answer

Q 6: (3,3,3) an equilateral triangle



Solution — 1 Point for each Correct Answer

Q 7: (3,4,3) valid isosceles t.



Solution — 1 Point for each Correct Answer

Q 8: all permutations of valid isosceles triangle:

(3,4,3), (3,3,4), (4,3,3)

Solution — 1 Point for each Correct Answer

Q 9: one side with zero value (0,4,3)

Solution — 1 Point for each Correct Answer

Q 10: one side with negative value (-1,4,3)

Solution — 1 Point for each Correct Answer

Q 11: all sides zero (0,0,0)

Solution — 1 Point for each Correct Answer

Q 12: at least one value is non-integer (1,3,2.5)

Solution — 1 Point for each Correct Answer

Q 13: wrong number of arguments (2,4) or (1,2,3,3)

Solution — 1 Point for each Correct Answer

Q 14 (the most important one):

Did you specify the expected output in each case?

About the Quiz

- Q 1–13 correspond to failures that have actually occurred in implementations of the program
- How many questions did you answer?
  < 5? $5 - 7$? $8 - 10$? > 10? All?
- Highly qualified, experienced programmers score 7.8 on average

First Conclusions

- Finding good and sufficiently many test cases is difficult
- Even a good set of test cases cannot exclude more failures
- A specification is required to identify failures

> The discipline of Testing is all about Test Cases

Remark: At Ericsson: 35% of code is test cases!

What is a Bug? Basic Terminology

Photo # NH 96566-KN First Computer "Bug", 1945

Harvard University, Mark II Aiken Relay Calculator

Bug-Related Terminology

1. Defect (aka bug, fault) introduced to code by programmer

Failure and Specification

Some failures are obvious

- obviously wrong output/behaviour
- non-termination
- crash
- freeze

. . . but most are not!

In general, what constitutes a failure, is defined by a specification!

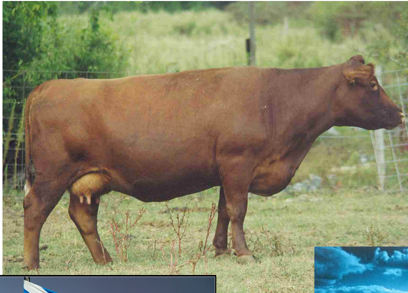
Correctness is a relative notion

— B. Meyer, 1997

Every program is correct with respect to SOME specification

— myself, today

Specification: Intro



Economist:
The cows in Scotland are brown

Logician:
No, there are cows in Scotland of which one at least is brown!

Computer Scientist:
No, there is at least one cow in Scotland, which is brown on one side!!

Specification: Putting it into Practice

Example

A Sorting Program:

```
1 public static Integer[] sort(Integer[] a) { ...
    }
```

Testing `sort()`:

- $\texttt{sort}(\{3, 2, 5\}) == \{2, 3, 5\}$ ✔
- $\texttt{sort}(\{\}) == \{\}$ ✔
- $\texttt{sort}(\{17\}) == \{17\}$ ✔

Specification

Requires: `a` *is an array of integers*

Ensures: *returns the sorted argument array* `a`

Example Cont'd

Example

```
1 public static Integer[] sort(Integer[] a) { ...
    }
```

Specification

Requires: `a` *is a non-null array of integers*

Ensures: *returns the unchanged reference* `a` *containing a permutation of the old contents of* `a` *that is sorted*

`sort(null)` throws `NullPointerException` ✘

The Contract Metaphor

Contract is preferred specification metaphor for procedural and OO PLs

first propagated by B. Meyer, *Computer* 25(10)40–51, 1992

Same Principles as Legal Contract between a Client and Supplier

- **Supplier** aka Implementer, in JAVA, a class or method
- **Client** Mostly a caller object, or human user for `main()`
- **Contract** One or more pairs of ensures/requires clauses defining mutual benefits and obligations of client and implementer

The Meaning of a Contract

Specification (of method `C::m()`)

Requires: *Precondition*
Ensures: *Postcondition*

"If a caller of `C::m()` fulfills the required Precondition, then the class `C` ensures that the Postcondition holds after `m()` finishes."

Often the following wrong interpretations of contracts are seen:

Wrong!

"Any caller of `C::m()` must fulfill the required Precondition."

Wrong!

"Whenever the required Precondition holds, then `C::m()` is executed."

Failure

Definition: failure

A method fails if it is called in a state fulfilling the required precondition of its contract and does not terminate in a state fulfilling the postcondition.

Non-termination, abnormal termination considered as failures here

Notions of Correctness

Definition: partial correctness

A method is partially correct if whenever it is started in a state fulfilling the required precondition and it terminates, then its final state fulfills the postcondition.

This amounts to proving Absence of Failures!

Definition: total correctness

A method is totally correct if whenever it is started in a state fulfilling the required precondition, then it terminates and its final state fulfills the postcondition.

Total correctness implies termination!

Invariant

Objects with non-trivial state

often maintain a class invariant.

Example: a class for dates

```
public class Date {
  public int day;
  public int month;
  public int year;
}
```

Invariant:

```
1 <= day <= 31 /\ 1 <= month <= 12 /\
(month in {4, 6, 9, 11} => day <= 30) /\
(month == 2 => day <= 29) /\
(month == 2 /\ (year % 4 != 0 \/ (year % 100 == 0 /\ year %
       => day <= 28)
```

Invariant II

- All public methods of a class must preserve the class invariant.
- Class invariants can be incorporated into pre- and postconditions.

Specification (of a method)

Requires: *Precondition and Invariant*
Ensures: *Postcondition and Invariant*

Specification (of a constructor)

Requires: *Precondition*
Ensures: *Invariant*

Further Elements of a Contract

Type signature (minimal contract)

Exceptions raised

Temporal properties

- the capacity of the table does not change over time
- a set that is only supposed to grow

Testing vs. Verification

TESTING

Goal: find evidence for presence of failures

Testing: execute a program with the intent of detecting failure

Testing cannot guarantee correctness, i.e., absence of failures

Related techniques: code reviews, program inspections

VERIFICATION

Goal: find evidence for absence of failures

Verification guarantees correctness

Related techniques: code generation, program synthesis (from spec)

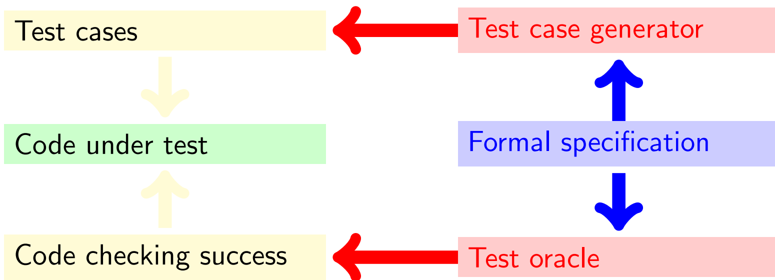
Debugging: from Failures to Defects

- Both, testing and verification attempts exhibit new failures
- Debugging is a systematic process that finds and eliminates the defect that led to an observed failure
- Programs without known failures may still contain defects:
  - if they have not been verified
  - if they have been verified,
    but the failure is not covered by the specification

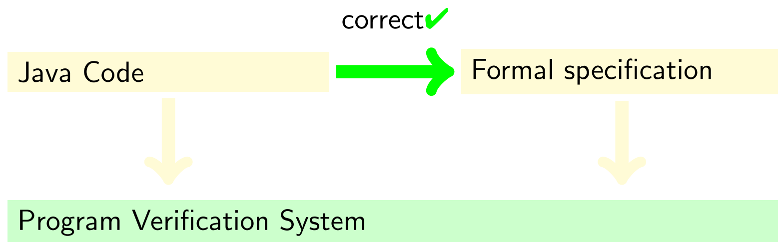
Where Formalization Comes In

Testing is very expensive, even with tool support

30–80% of development time goes into testing



Formal Verification of Program Correctness



Computer support essential for verification of real programs

```
synchronized java.lang.StringBuffer append(char c)
```

- ca. 15.000 proof steps
- ca. 200 case distinctions
- Two human interactions, ca. 1 minute computing time

Tool Support is Essential

Some Reasons for Using Tools

- Automate repetitive tasks
- Avoid typos, etc.
- Cope with large programs

Tools Used

- Automated running of tests: JUnit
- Debugging: Eclipse debugger