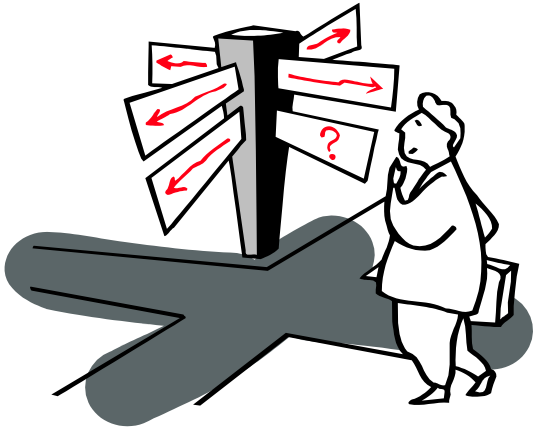
Software Testing

1. Introduction

Universiteit Antwerpen

1. Introduction



(Based on "Part I: Preliminaries" of Testing Object-Oriented Systems)

- Challenge
- Test Adequacy (vs. Test Inadequacy)
- The V model
- Terminology
- Limits of testing
- What is special about object-oriented testing ?
- Challenge revisited

Challenge

Devise a test plan (i.e. a set of test cases) for a program that ...

> … reads three integer values from a card(*). The three integer values are interpreted as representing the lengths of the side of a triangle. The program prints a message that states whether the triangle is scalene, isosceles, or equilateral.
>
> From “The Art of Software Testing” (Myers, 1978)

(*) A “Card” was common input medium in 1978, interpret as “File”.

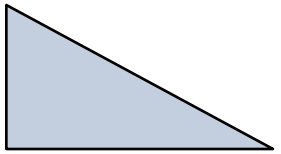
Challenge (help)

A valid triangle must meet two conditions

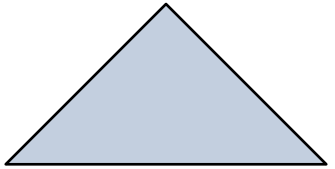
- No sides may have a length of zero
- each side must be shorter than the sum of all sides divided by 2

A triangle is

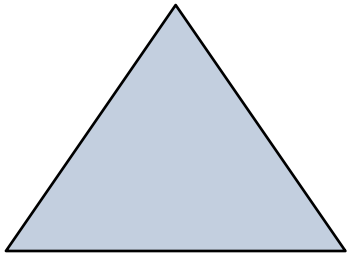
- scalene: no sides are equal in length



- isosceles: there exist two sides which are equal in length



- equilateral: all sides are equal in length



Challenge (solution)

- 3 one valid for each scalene, isosceles, or equilateral
- 3 permutations for equal sides (all isosceles)
- 1 one side a zero length
- 1 one side negative length
- 3 permutations for equal sides (all invalid)
- 6 permutations (one side smaller than sum of all sides divided by 2)
- 1 all sides zero
- 3 non-integer inputs
- 3 missing inputs
- 6 permutations (one side equals the sum of the other two)
- 3 three, two and one sides at maximum value (MAXINT)

  - 33 test cases are possible !
  - Highly experienced programmers score on the average 7.8/14

What is Testing ?

Testing = design and implementation of a special software system:
exercises another software system with the intent of finding bugs

Test Design = analyze system under test and decide where bugs are likely to occur

Test Implementation = automate as much as possible; i.e. apply test cases and evaluate results

Test Models = abstraction to conquer astronomical complexity.

Fault Model = Focus on places where bugs are likely to occur

Test Adequacy vs. Test Inadequacy

The "Test Adequacy Utopia"

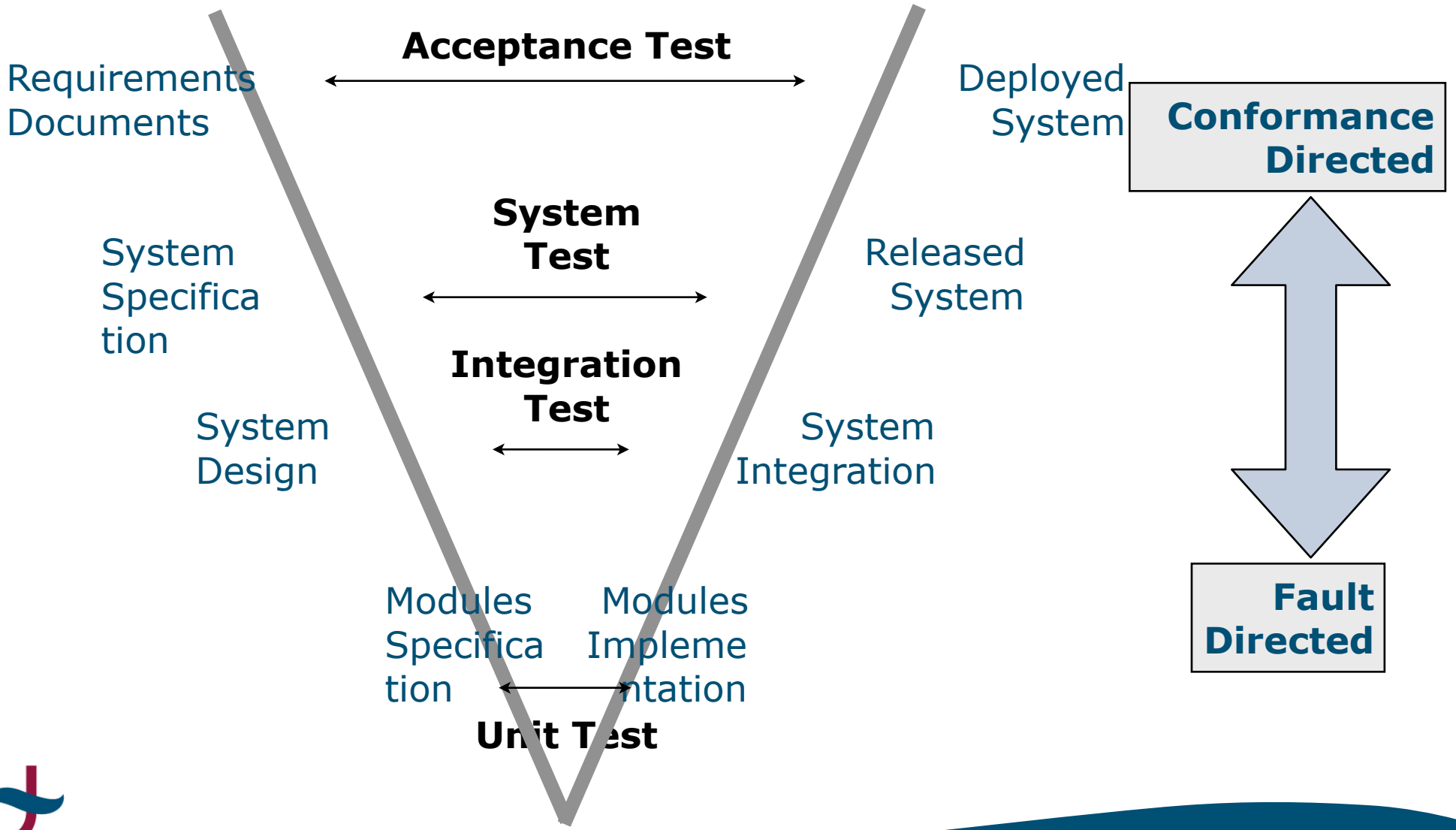
- If a system passes an ADEQUATE suite of test cases, then it must be correct
  - impossible: provable undecidable

Weaker proxies for adequacy

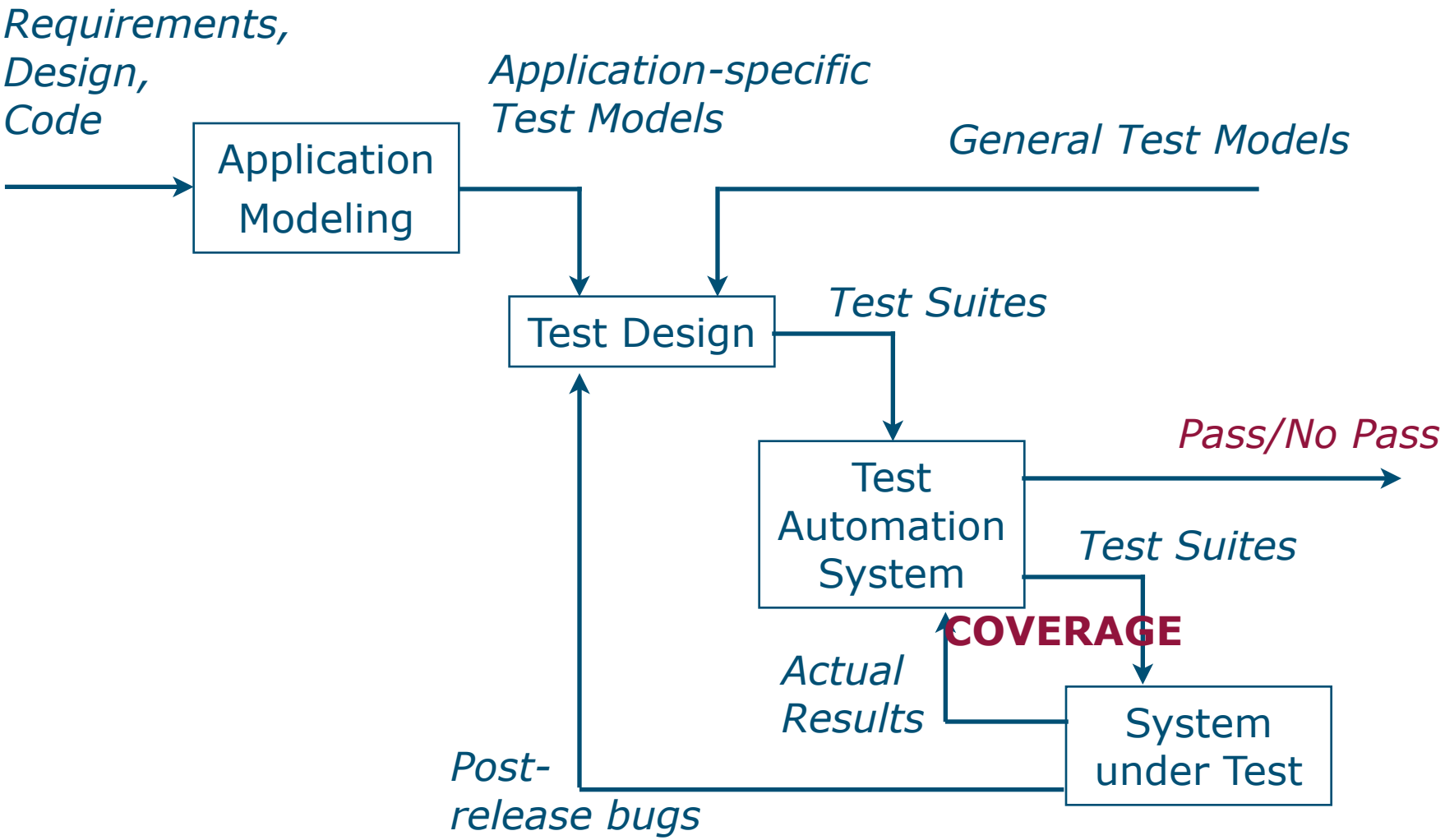
- Design rules to highlight INADEQUACY of test suites
  - If a given suite of test cases does not satisfy the design rules … reconsider carefully

Testing is risk assessment !

The V-model


Acceptance Test
Requirements Documents
Deployed System
Conformance Directed
System Test
System Specification
Released System
Integration Test
System Design
System Integration
Modules Specification
Modules Implementation
Unit Test
Fault Directed

Testing: systems engineering view



Terminology (1/4)

- software testing = execution of code using combinations of input and state to reveal bugs

  (not requirements validation ! not code/design/… reviews !)
- component (under test) = any software aggregate that has visibility in the development environment (method, class, object, function, module, executable, task, subsystem, ...)
- scope of test = collection of components to be verified

- implementation under test
  method under test
  object under test
  class under test
  component under test
  system under test

Perspective of a forensic investigator dissecting suspicious samples

Terminology (2/4)

- unit test = test scope is small executable (object of a class, method)
- integration test = test scope is complete system or subsystem (software AND hardware)
- system test = test scope is a complete and integrated application

- fault-directed testing: intent is to reveal faults through failures
- conformance-directed testing: intent is to demonstrate conformance to required capabilities

- confidence = assessment of the likelihood of unrevealed bugs
- (estimated) failure rate = the probability that a failure will occur after a certain period of usage

  ➡ both are external quality attributes !

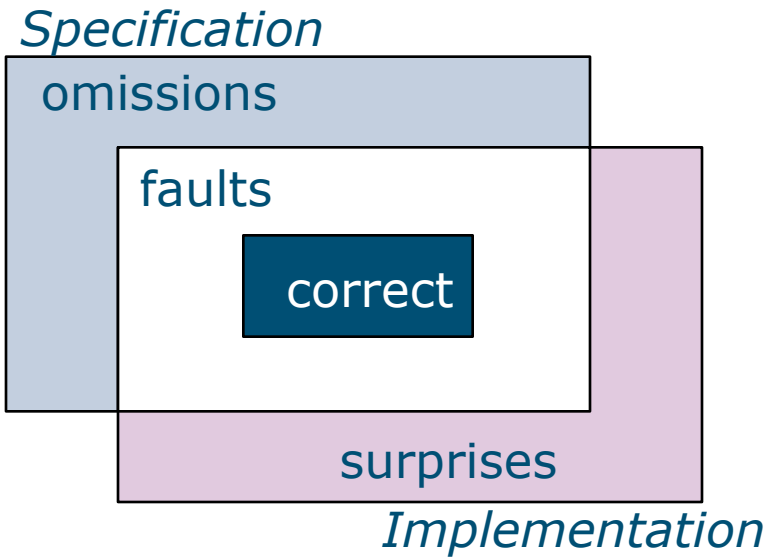
Terminology (3/4)

- test case = pretest state of implementation under test + test inputs or conditions + expected results
  - expected results: generated messages + thrown exceptions + returned values + resultant state
  - oracle = means to produce expected result
  - test point = specific value for test case input and state variables

- domain = a set of values that input or state variables of the implementation under test may take
- domain analysis: places constraints on input/state/output to select test points
  - equivalence classes (partition testing)
  - boundary value analysis, special values testing

Terminology (4/4)

- Failure = manifested inability of a system
- Software fault = missing or incorrect code
- Error = human action that produces a fault
- bug = error or fault
- omission = required capability that is not present
- surprise = code that does not support any required capability



Limits of Testing: State space explosion

Input space is surprisingly large

- Simplified case
  - Triangle example with points in coordinate system [1..10, 1..10]
  - 10^2 = 100 possible end-points; 10^4 = 10.000 possible lines; $10^{4*3} = 10^{12}$ possible triangles
- Less simplified
  - display of 1024 x 768 pixels; 786.432^2 possible lines; 786.432^6 possible triangles = $2{,}36574 \times 10^{35}$
- Full integer coordinate system
  - $2^{16 \times 4}$ possible lines; $2^{16 \times 4 \times 3} = 2^{192} = 6.277 \times 10^{57}$ possible triangles (number of particles in the universe = $+\text{-}10^{80}$)

Limits of Testing: Loops

```
for (int i = 0; i < n; i++) {
      if (a.get(i) == b.get(i))
          x[i] = x[i] + 100;
      else
          x[i] = x[i] / 2;
}
```

Without iteration:

- 3 entry exit paths

With two iterations:

- 5 possible paths

For n iterations

- $2^n + 1$ possible paths

Limits of Testing: Coincidental Correctness

- Coincidental correctness: buggy code may produce correct results under some circumstances
  - example: write x + x instead of x * x
    will produce correct result for x = 2 !

  - example 2:

```
int scale(int j) {
    j = j -1 ;   // should be j = j + 1;
    j = j / 30000;
    return j;
}
```

    out of 65.536 possible values for j, only six will reveal the fault !
    (-30001, - 30000, -1, 0, 29999 and 30000)

Coincidental Correctness in Inheritance

```
public class Account extends Object {
      protected Date lastTxDate, today;
      // ...
      int quartersSinceLastTx () {
            return (90 / daysSinceLastTx ());
      }
      int daysSinceLastTx () {
            return (today.day() - lastTxDay.txDate + 1);
            // Correct - today's transactions return 1 day elapsed
}}
public class TimeDepositAccount extends Account {
      // ...
      int daysSinceLastTx () {
            return (today.day() - lastTxDay.txDate);
            // Incorrect - today's transactions return 0 days
      }
}
```

quartersSinceLastTx will produce "divide by zero" exception when last transaction occurs on the current day

Fault Models for Object-Oriented Programming

Object-oriented programming is based on powerful constructs

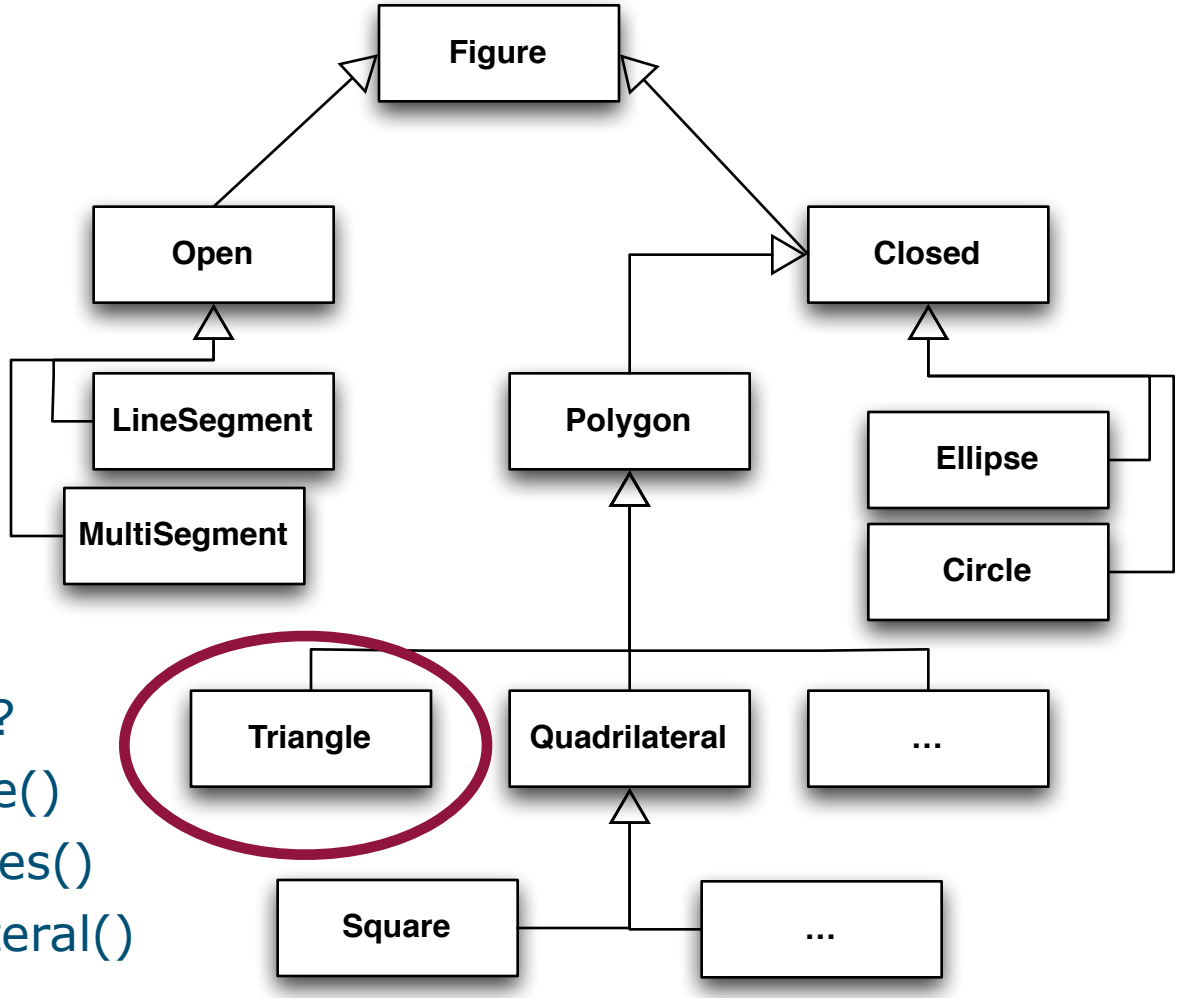
- easy to use, easy to abuse

- dynamic binding + complex inheritance
  - faults due to unanticipated bindings and misinterpretations
- many small components with lots of interfaces
  - interface programming are a known cause of faults
- objects preserve state, but state control is distributed over program
  - state control errors are likely

Object-Oriented Testing ≠ Traditional testing

- Encapsulation:
  - difficult to bring an object in desired state
  - difficult to verify whether it is in a desired state
- Inheritance:
  - Weakens encapsulation
  - Subtle interactions (initialize & new, copy & isEqual, == & hash)
  - type/subtype relationship (Liskov's substitution principle)
  - multiple inheritance: even more possibilities for subtle interactions
  - abstract classes: some methods are intentionally empty
  - generic classes: interaction between type parameter and the generic class
- Polymorphism
  - should be considered if- or switch statement (branch testing !)
  - subtle interactions (double dispatch of + on Number hierarchy)

Challenge revisited (class diagram)



How to test ?

- is_scalene()
- is_isosceles()
- is_equilateral()

Challenge revisited (java code)

```
class Polygon extends Figure {
    abstract void draw (int r, int g, int b);
    abstract void erase ();
    abstract float area (); abstract float perimeter (); abstract float center ();
}
class Triangle extends Polygon {
    public Triangle (LineSegment a, LineSegement b, LineSegement c);
     public void setA(LineSegment a);
       … setB … ; … setC …;… getA …;… getB …;… getC …;
    public boolean is_scalene(); … is_isosceles(); …is_equilateral()
}
class LineSegment extends Figure {
    public LineSegment (int x1, int y1, int x2, int y2);
       … setx1 … ; … sety1 …;… setx2 …;… sety2 …;… getx1 …;… gety1 …; …
}
```

Challenge (object-oriented - solution 1/2)

- constructor faithfully creates line segments
- only one of the predicates (is_*) is true ate a given time
- minimum/maximum values for each LineSegment parameter
- repeat a result
- repeat result after permuting line lengths
- repeat result after sending erase, draw, …
- repeat result more than 2 times
- 1 single point line, 2 single point lines, 3 single point lines
- three lines: 1 minimum length + 2 maximum lengths
- two nonintersecting single lines
- two overlapping lines of different lengths
- three overlapping lines of different lengths
- two intersecting lines
- two parallel lines
- three nonintersecting parallel lines
- three lines intersecting at one point
- three parallel lines
- three non intersecting, non parallel lines
- three lines that form an interior triangle but extend beyond the intersections

Challenge (object-oriented - solution 2/2)

- three lines with two intersections (“open” triangle)
- three lines with one intersections (“open” triangle)
- three lines, each of maximum length
- three lines, each of maximum length
- a figure with one line entirely on each boundary of the coordinate space
- above for two and three lines
- two lines that intersect and originate from the same boundary of the coordinate space

➡ 62 test cases in total

Original Myers Tests

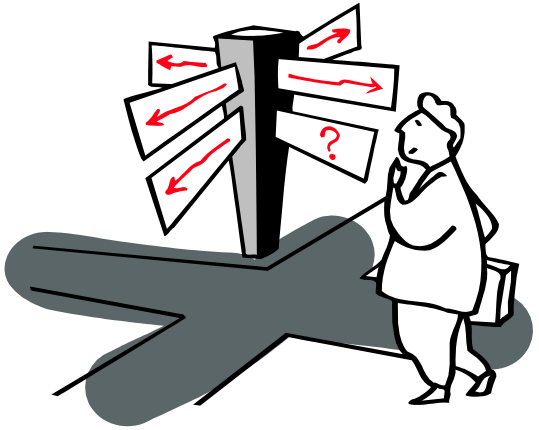
- 33 test cases
  - 6 not possible (non integer input & missing input)
- 62 + 27 = 89 test cases !

Inheritance & Polymorphism

- All methods defined in *Figure* inherited or overridden by Triangle provide a response that is consistent with the original definition
- above for *Closed*
- above for *Polygon*

➡ 3 x other tests with a substituted triangle

Conclusion



(Based on "Part I: Preliminaries" of Testing Object-Oriented Systems)

- Challenge
- *Test Adequacy (vs. Test Inadequacy)*
- *The V model*
- Terminology
- Limits of testing
- *What is special about object-oriented testing ?*
- Challenge revisited