



中山大學
SUN YAT-SEN UNIVERSITY

Part II [Problem]

1. Test Oracle



SE-307 Software Testing Techniques

<http://my.ss.sysu.edu.cn/wiki/display/SE307/Home>

Instructor: Dr. Wang Xinming, School of Software, Sun Yat-Sen University

Review: the four dimensions of testing techniques



- **P**roblems

- Fundamental problems: test adequacy, test oracle, test generation
- Important problems: test plan and process, test automation

- **M**ethods

- Exploratory testing, black box testing, white box testing, model-based testing

- **O**bjectives

- Functional testing (unit testing, integration testing, system testing), Performance testing, Validation testing (alpha testing, beta testing), Performance testing (stress testing, load testing), Reliability testing, Regression testing, Security testing, Compatibility testing, ...

- **D**omains

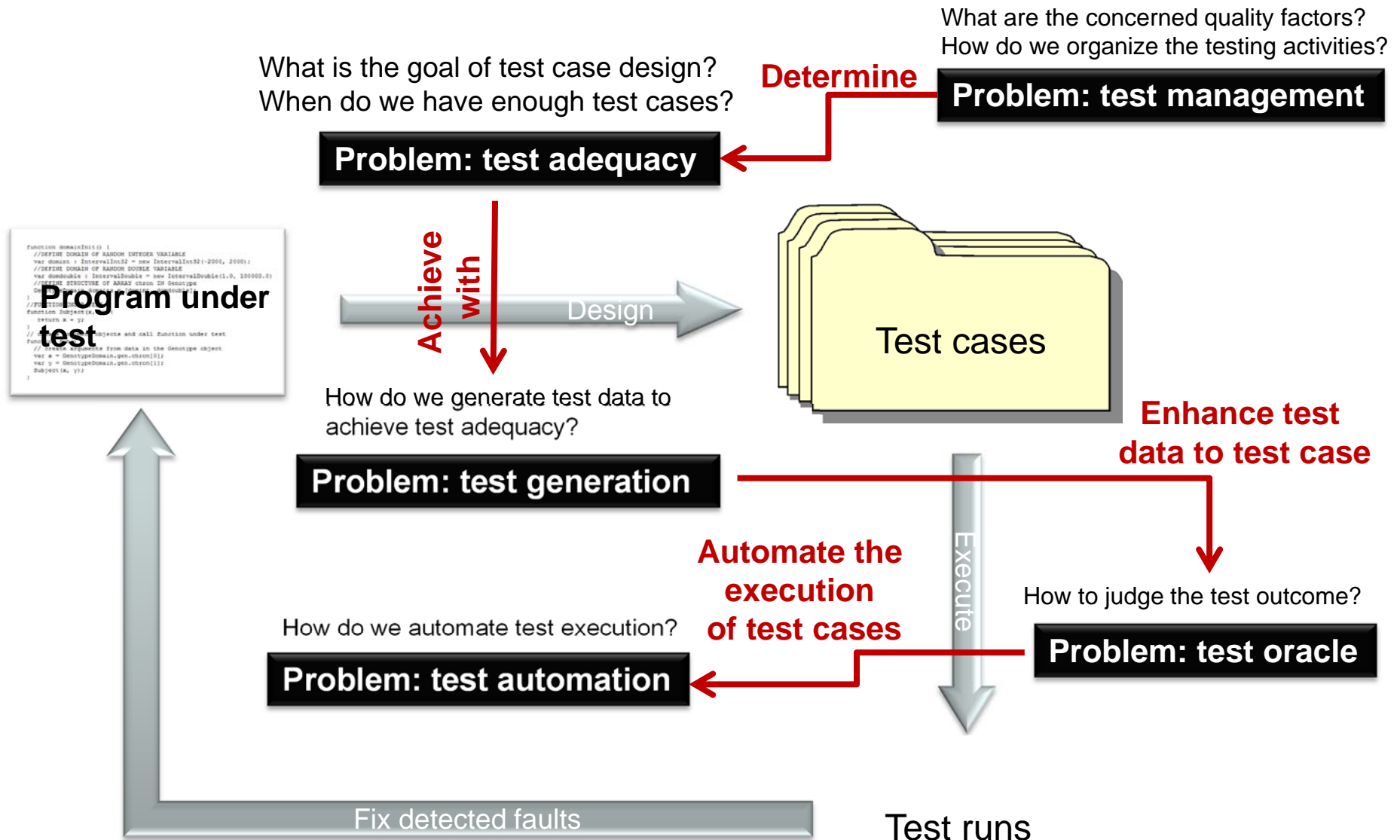
- Unit testing, integration testing, system testing, acceptance testing

The first dimension: Problems

- Why testing is hard?
 - **Fundamental** problems
 - Test oracle
 - Test adequacy
 - Test generation
 - **Important** problems
 - Test process and plan
 - Test automation



Their relation



Fundamental problems in testing

- Test oracle
- Test adequacy
- Test generation

Oracle

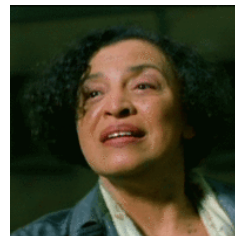
The Oracle offers candy to Neo, the following dialogue ensues ...

Neo: Do you already know if I'm going to take which candy?

The Oracle: Wouldn't be much of an Oracle if I didn't.



- The Oracle in Matrix
 - a program designed to investigate the human psyche
 - predicts choices and outcomes based on input from the Zion mainframe.
 - limited, cannot predict everything.
- Test oracle



Test oracle

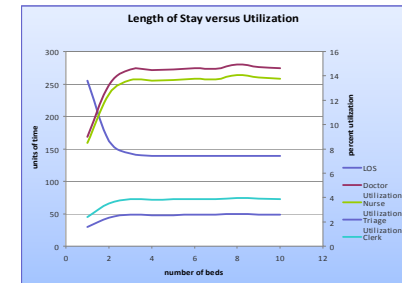
- A **decision procedure** on the program output of a test run (e.g. screen display, logs, controlled behaviors etc.), that makes a **pass/fail** decision about the test outcome

Test outcome: pass/fail



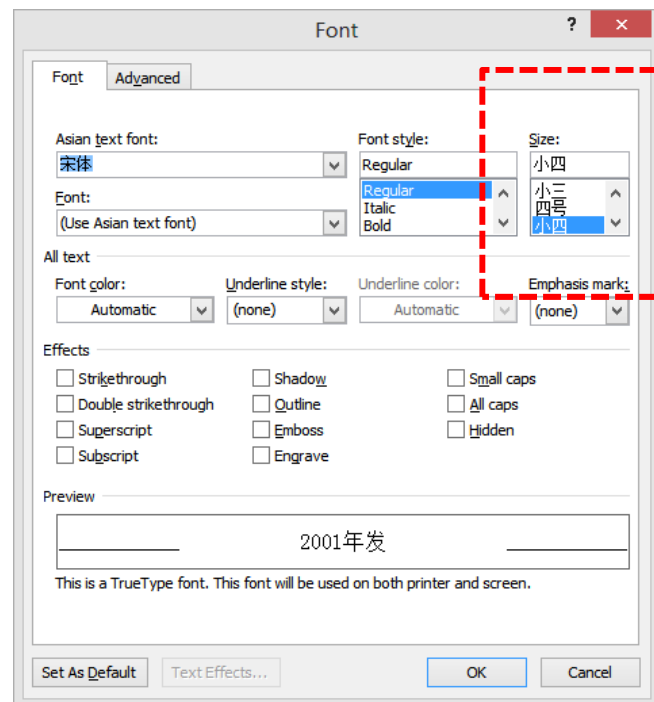
- without them, you have no idea whether you have detect the bug
 - A test case is only as good as its oracle.

- [illegible]



Example

- How do you know that the “font size” feature work in Office Word?



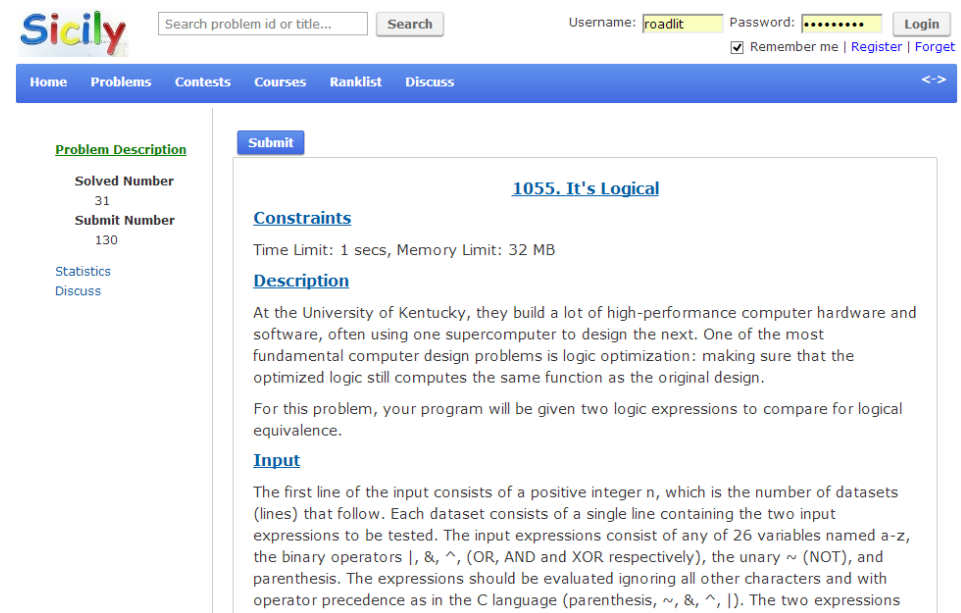
What is your oracle?

Practical Test Oracle

- Specification as test oracle
- Reference implementation as test oracle
- Program assertion as test oracle
- Metamorphic relation as test oracle
- Execution profiling as test oracle
- Heuristic and statistical test oracle

Specification as test oracle

- Extract valid input/output pairs from specifications to verify 'intended behavior' of the system.
 - Simulate what the program does with your brain.
- Example: problems in Sicily



However...

- **Problem 1:** the output maybe too complex or not human-readable.

helloworld.c

```

// helloworld.c : defines the entry point for the console application.
//
#include "stdafx.h"
#include<iostream>
#include<fstream>

using namespace std;

int _tmain(int argc, _TCHAR* argv[])

{
    ifstream infile;
    ofstream outfile;

    infile.open( "C:\\Users\\rubin\\Desktop\\1.txt" );
    outfile.open( "C:\\Users\\rubin\\Desktop\\2.txt" );

    char a;
    while(infile.get(a))
    {
        outfile << a;
    }

    infile.close();
    outfile.close();

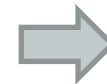
    cout << "success" << endl;

    return 0;
}

```



C Compiler



helloworld.o

```

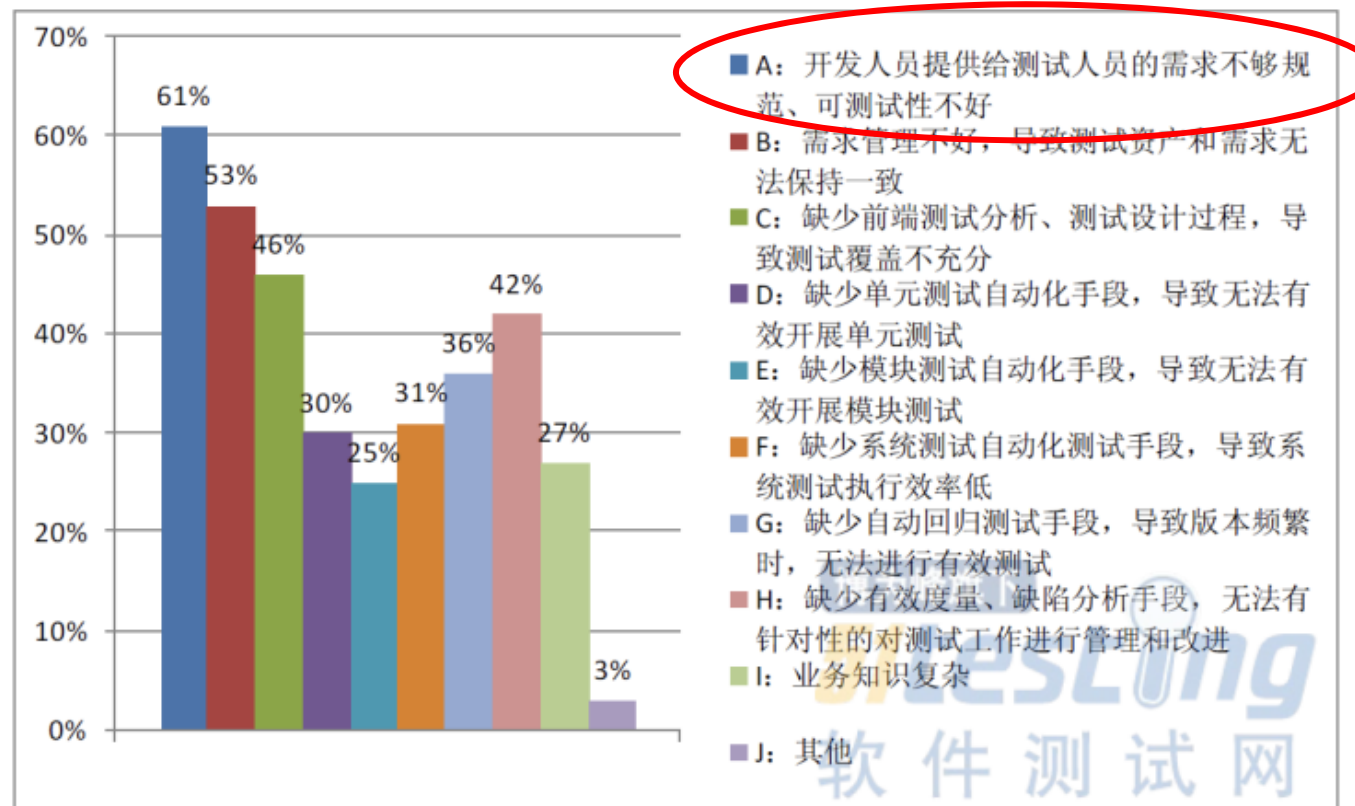
0 1 2 3 4 5 6 7 8 9 a b c d e f
00000000h: 4C 01 BD 01 F3 37 37 51 0C 1D 02 00 75 07 00 00 ; L???Q...u...
00000010h: 00 00 00 00 2E 64 72 65 63 74 76 65 00 00 00 00 ; .....directve....
00000020h: 00 00 00 00 A2 00 00 00 9C 45 00 00 00 00 00 00 ; ....?..?..?..
00000030h: 00 00 00 00 00 00 00 00 00 00 0A 10 00 2E 64 65 62 ; .....deb
00000040h: 75 67 24 53 00 00 00 00 00 00 00 00 20 56 00 00 ; ug$S..... V..
00000050h: 3E 46 00 00 5E 9C 00 00 00 00 00 00 06 00 00 00 ; >F..^?.....
00000060h: 40 00 10 42 2E 72 64 61 74 61 00 00 00 00 00 00 ; @..B.rdata.....
00000070h: 00 00 00 00 01 00 00 00 9A 9C 00 00 00 00 00 00 ; .....?..
00000080h: 00 00 00 00 00 00 00 00 40 10 40 2E 72 64 61 61 ; .....@..@.rda
00000090h: 74 61 00 00 00 00 00 00 00 00 00 00 01 00 00 00 ; ta.....
000000a0h: 9B 9C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; 港.....
000000b0h: 40 10 10 40 2E 72 64 61 74 61 00 00 00 00 00 00 ; @..@.rdata.....
000000c0h: 00 00 00 00 04 00 00 00 9C 9C 00 00 00 00 00 00 ; .....港.....
000000d0h: 00 00 00 00 00 00 00 00 40 10 30 40 2E 72 64 61 ; .....@..@.rda
000000e0h: 74 61 00 00 00 00 00 00 00 00 00 00 04 00 00 00 ; ta.....
000000f0h: A0 9C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; 騰.....
00000100h: 40 10 30 40 2E 72 64 61 74 61 00 00 00 00 00 00 ; @..@.rdata.....
00000110h: 00 00 00 00 04 00 00 00 A4 9C 00 00 00 00 00 00 ; .....
00000120h: 00 00 00 00 00 00 00 00 40 10 30 40 2E 72 64 61 ; .....@..@.rda
00000130h: 74 61 00 00 00 00 00 00 00 00 00 00 04 00 00 00 ; ta.....
00000140h: A8 9C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000150h: 40 10 30 40 2E 72 64 61 74 61 00 00 00 00 00 00 ; @..@.rdata.....
00000160h: 00 00 00 00 04 00 00 00 AC 9C 00 00 00 00 00 00 ; .....設.....
00000170h: 00 00 00 00 00 00 00 00 40 10 30 40 2E 72 64 61 ; .....@..@.rda
00000180h: 74 61 00 00 00 00 00 00 00 00 00 00 04 00 00 00 ; ta.....
00000190h: B0 9C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; 載.....
000001a0h: 40 10 30 40 2E 72 64 61 74 61 00 00 00 00 00 00 ; @..@.rdata.....
000001b0h: 00 00 00 00 04 00 00 00 B4 9C 00 00 00 00 00 00 ; .....礎.....
000001c0h: 00 00 00 00 00 00 00 00 40 10 30 40 2E 72 64 61 ; .....@..@.rda
000001d0h: 74 61 00 00 00 00 00 00 00 00 00 00 04 00 00 00 ; ta.....
000001e0h: B8 9C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; 麓.....
000001f0h: 40 10 30 40 2E 72 64 61 74 61 00 00 00 00 00 00 ; @..@.rdata.....
00000200h: 00 00 00 00 04 00 00 00 BC 9C 00 00 00 00 00 00 ; .....礎.....
00000210h: 00 00 00 00 00 00 00 00 40 10 30 40 2E 72 64 61 ; .....@..@.rda
00000220h: 74 61 00 00 00 00 00 00 00 00 00 00 04 00 00 00 ; ta.....
00000230h: C0 9C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; 魯.....
00000240h: 40 10 30 40 2E 72 64 61 74 61 00 00 00 00 00 00 ; @..@.rdata.....
00000250h: 00 00 00 00 04 00 00 00 C4 9C 00 00 00 00 00 00 ; .....
00000260h: 00 00 00 00 00 00 00 00 40 10 30 40 2E 72 64 61 ; .....@..@.rda

```

Correct or not?

However...

- **Problem 2:** the specification maybe incomplete or ambiguous



2011 年软件测试从业人员在测试工作中存在的障碍分布

However

- Problem 3: the program maybe ***un-deterministic***

Specification of STS

STS enables its users to design tournaments containing a soccer pitch (or field) and a variety of teams. The user can define each team's strategy by prescribing the roaming regions of each of its players. Users may elect a random, rather than a custom strategy, which instructs STS to determine an arbitrary strategy automatically. Once the pitch and teams are specified, STS runs a complete simulated soccer tournament. In the tournament each pair of teams plays against each other twice. At the conclusion of the tournament, STS can output the final scores of the soccer matches and as well as output a tournament standings table, which shows the relative performance of the teams.

- [Specification of STS](#)

- [1. Specify a soccer tournament](#)
 - [Specify a soccer pitch \(球场\)](#)
 - [Specify a soccer team \(球队\)](#)
- [2. Play a tournament](#)
 - [Display tournament scores](#)
 - [Display tournament standings](#)

	Manchester U	Chelsea	Liverpool	Arsenal	Manchester C	Aston Villa	Tottenham	Blackburn Ro
Manchester U	---	2-1	3-0	2-0	7-0	2-1	0-0	3-1
Chelsea	1-2	---	3-0	4-0	5-0	2-0	2-1	4-1
Liverpool	0-5	0-3	---	0-2	0-3	0-1	0-2	0-5
Arsenal	0-5	0-5	1-0	---	4-1	1-3	2-2	2-8
Manchester C	0-9	0-15	0-1	0-3	---	0-0	0-2	3-5
Aston Villa	1-4	0-1	0-0	0-2	0-0	---	1-0	1-2
Tottenham	1-3	0-2	0-0	0-1	2-2	0-0	---	0-3
Blackburn Ro	2-6	4-8	2-0	4-1	14-0	4-1	8-1	---

Practical Test Oracle

- Specification as test oracle
- **Reference implementation as test oracle**
- Program assertion as test oracle
- Metamorphic relation as test oracle
- Execution profiling as test oracle
- Heuristic and statistical test oracle

Reference implementation as test oracle

- Reference implementation
 - Another program that does exactly (or similarly) what the program under test does.
- Situations where reference implementation can be available:
 - **Case 1: deliberately built** to mimic the behavior of the program under test
 - mainly built for accuracy rather than efficiency
 - e.g. Sun's J2EE Reference Implementation
 - **Case 2: alternative products**
 - minor implementation difference
 - e.g. Internet Explorer vs. Firefox
 - e.g. GCC vs. Microsoft C compiler
 - **Case 3: historical version**
 - Regression testing

Internet Explorer



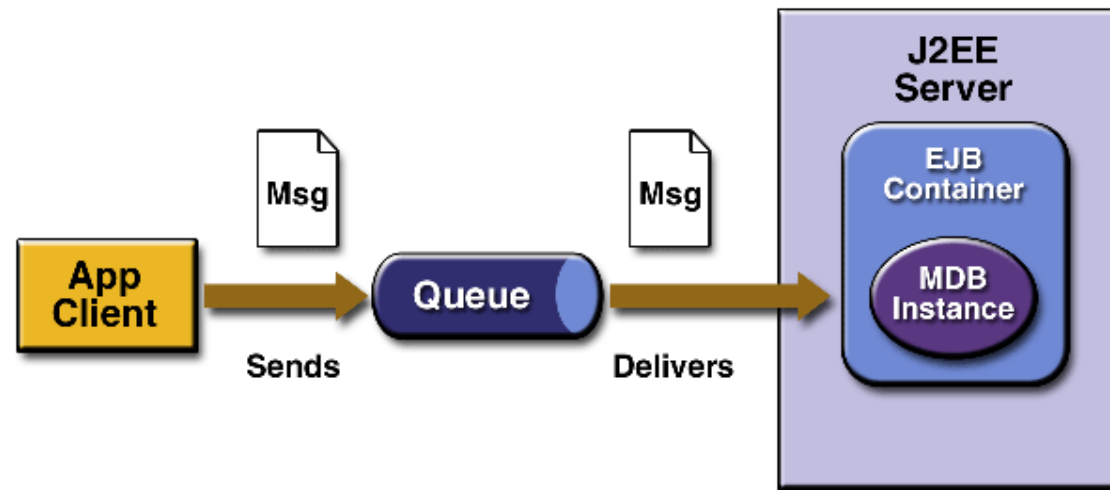
Firefox



Case 1: deliberately built

- **Example 1: J2EE reference implementation**

- J2EE: a component-based approach to the design, development, assembly, and deployment of enterprise applications
- EJB Container: the core of J2EE.
- J2EE Reference Implementation: A reference implementation of EJB container for proto-typing J2EE applications and for providing an operational definition of the J2EE platform.



Example 2: testing an instrumentation tool

- Implementation of test coverage exercises on smart_programmer.
 - Instrument the code to record coverage.
 - Insert Macro to trace the execution.
- How to verify the instrumentation result is correct?

```
#include <stdio.h>

int main(){
    int a, b, c, r;

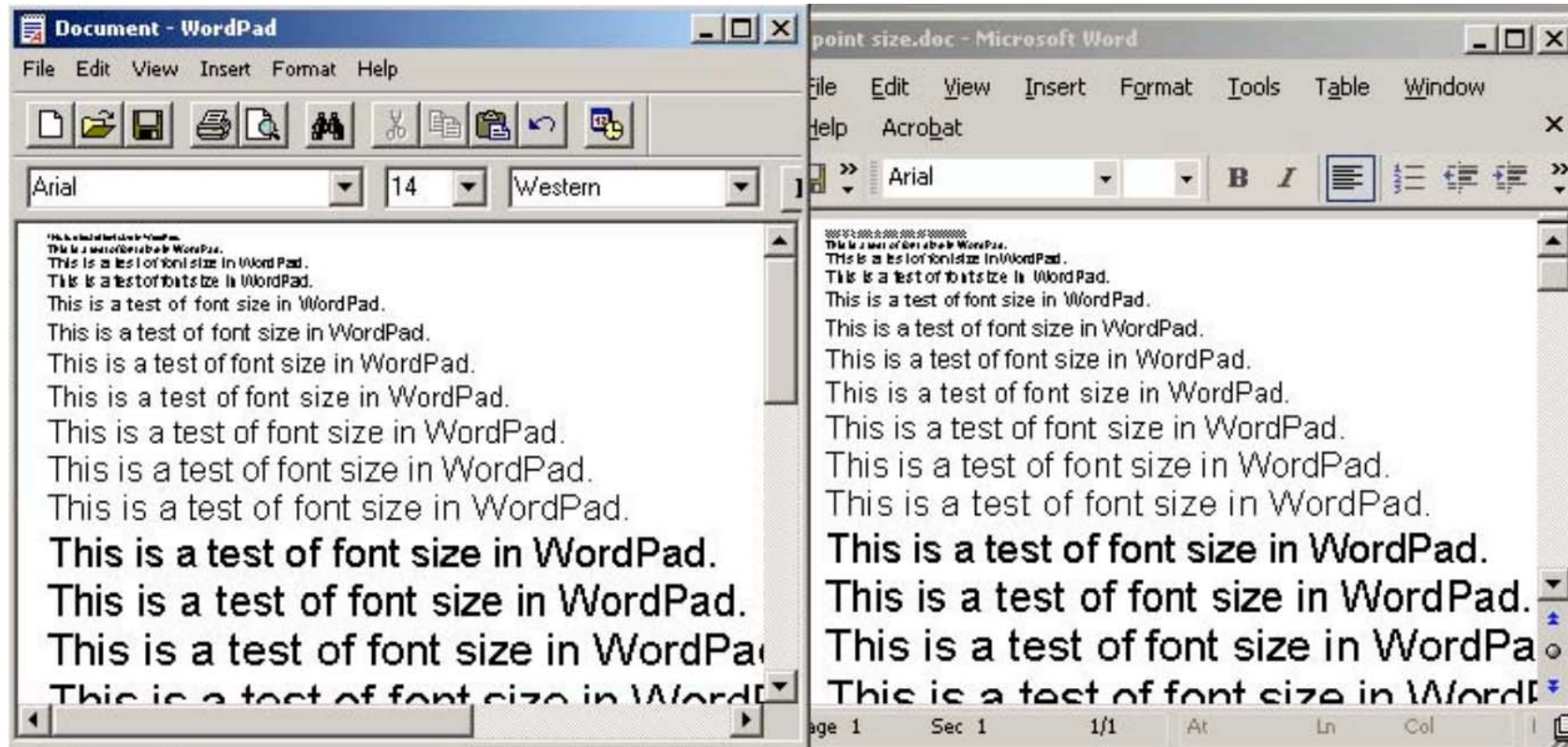
    scanf("%d %d %d", &a, &b, &c);

    if ( a > b ){
        if ( c > a ){
            r = a;
        }else if ( c > b ){
            r = c;
        }else{
            r = b;
        }
    }else{
        if ( c > b ) {
            r = b;
        }else if( c > a ){
            r = c;
        }else{
            r = a;
        }
    }

    printf("Middle: %d\n", r);
    return 0;
}
```

Case 2: alternative product

- Example 1: How do you know that the “font size” feature work in Office Word?
 - Compare it with WordPad, pixel by pixel.



Example 2: testing a file system

- POSIX standard for file system operations
 - IEEE produced, ANSI/ISO recognized standard for file systems
 - Defines operations and what they should do/return, including nominal and fault behavior
- We can use UNIX file systems (ext3fs, tmpfs, etc.) as reference systems to verify the correct behavior of the new file system.

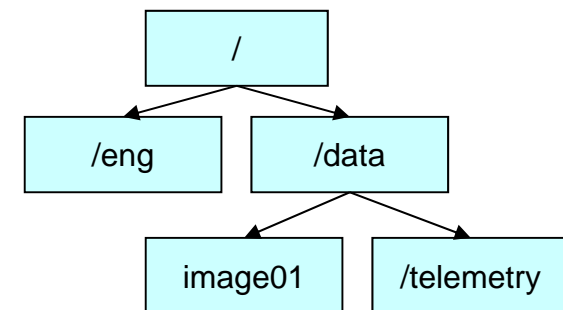
POSIX operation

```
mkdir ("/eng", ...)
mkdir ("/data", ...)
creat ("/data/image01", ...)
creat ("/eng/fsw/code", ...)
mkdir ("/data/telemetry", ...)
unlink ("/data/image01")
```

Result

```
SUCCESS
SUCCESS
SUCCESS
ENOENT
SUCCESS
SUCCESS
```

File system



Case 3: historical version

- Typically during regression testing
 - Use **capture and replay** tool to capture interactions with the system under test.
 - The captured output serves as test oracle for later version.
- During the replay of events, there are various output checking possibilities:
 - **Manual**: user has to watch the system for anomalies
 - **Complete**: all outputs were recorded during capture, and system must reproduce them “exactly”.
 - **Checkpoints**: system output is only checked at certain points for specified values.

Tool: Jacareto

<http://sourceforge.net/apps/mediawiki/jacareto>

- Java-based open source GUI capture + replay tool
 - Uses virtual machine registration.
 - During the capture process, the tool will register as an event listener.
 - During the replay process, the tool will register as a source of keyboard and mouse event.
 - Discover all instances of Swing components with reflection.
 - Replay to detect regression faults at user-specified checkpoints.

Recorded execution as oracle

- At various user-defined checkpoints, the state of a control can be tested.
- Example: Test if a text field contains the specified text
 - Also specify that we want to stop if there is an error (i.e. ignore error is false), and that we do not want to correct the state of the control if it does not conform to the record.

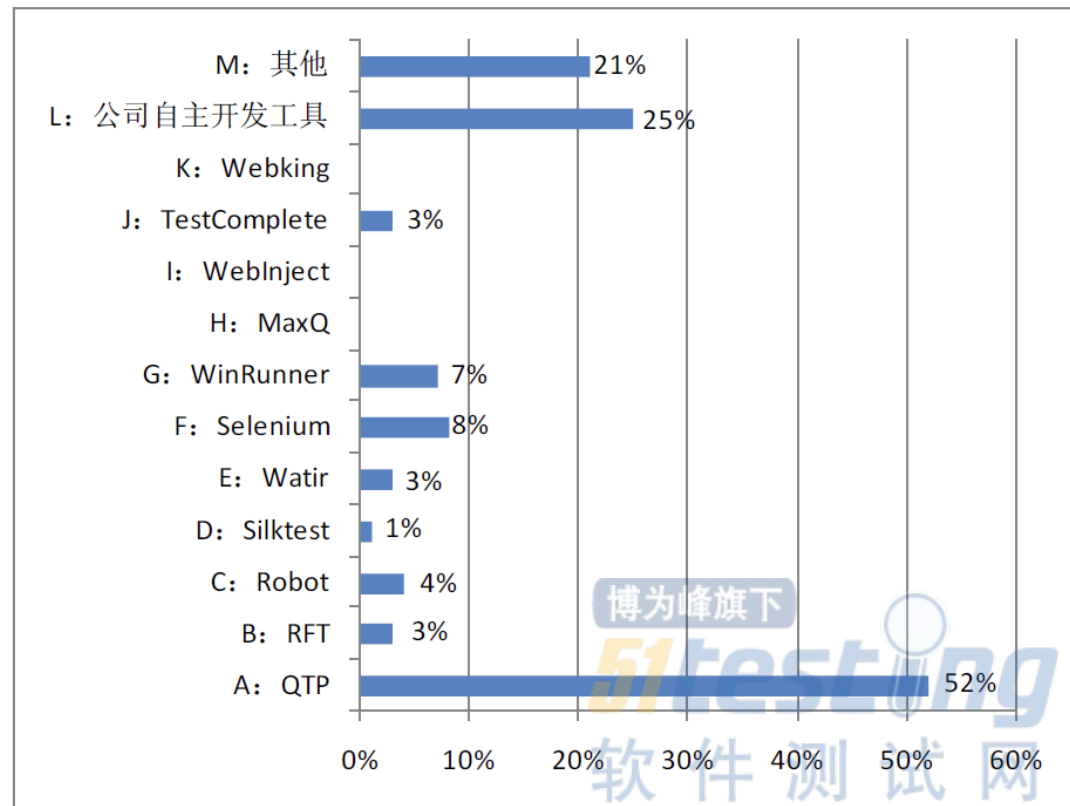
```
<JTextComponentTest
  component="MainFrame_(1).JRootPane_(1).JLayeredPane_(
  1).JPanel_(2).JTextField_(1)" isRegExp="false"
  isCorrecting= "false isEnabled="true"
  isIgnoring="false" hasFocus="false">
  Hello, Alan
</JTextComponentTest>
```

Limitations of capture and replay

- Only feasible with a past implementation
 - Infeasible if we are develop a new application
 - Cannot test new features
- Sensitive to change of interface
 - Change of control layout, size, focus order, etc.
 - Serious test maintenance problem
- Still need human effort at the “capture” part
 - Executing *all* test cases
 - Defining checkpoints

Notable implementations

- Open Source
 - Selenium
 - LDTP
 - Eclipse TPTP Auto GUI recorder
- Commercial
 - TestComplete
 - HP WinRunner
 - IBM Rational Functional Tester (RFT)



2011 年调查中软件测试从业人员常用的功能自动化测试工具分布

Practical Test Oracle

- Specification as test oracle
- Reference implementation as test oracle
- **Program assertion as test oracle**
- Metamorphic property as test oracle
- Execution profiling as test oracle
- Heuristic and statistical test oracle

Program assertion as test oracle

- What is an assertion?
 - An *assertion* is a statement in the program that enables you to test your assumptions about your program.
 - Each assertion contains a Boolean expression that you believe will be true when the assertion executes.
 - By verifying that the Boolean expression is indeed true, the assertion confirms **your assumptions about the behavior of your program**, increasing your confidence that the program is free of errors.

Program assertion as test oracle

- What kind of assumption on the program behavior can be specified with assertion?
 - *Safety property*
 - Can: nothing bad happens
 - Cannot: something good will always happen
 - *Partial specification*: not a complete specification mechanism
 - Does not cover all input domain
 - Cannot detect all possible errors
 - Inspecting internal states instead of output

The assert statement in Java

- An *assertion* is declared using the new Java keyword assert in JDK 1.4 as follows:

assert condition; or *assert condition: valueExpression;*

where *condition* is a Boolean expression and *valueExpression* is an expression that returns primitive-type or an Object (but not `void`).

- Whenever Java executes the assertion statement:
 - The Boolean assertion is evaluated,
 - If it is false, an `AssertionError` will be thrown, Java automatically gives you a stack trace, complete with the line number.
- Disabled by default. Need to be enabled by “**-ea**” switch to JVM.

Notes on Java assert statement

- An **AssertionError** is an **Error**, not an **Exception**
 - You do not need to put it in a **try** statement
 - You do not need to (and cannot) **catch** the error
- The second expression is seldom necessary
 - Use the second expression only if you have useful information to add to the error message

What to assert? (1)

- **Internal Invariants**

- A “fact” that you believe to be true at a certain point in the program

```
public class AssertionDemo {  
    public static void main(String[] args) {  
        int i; int sum = 0;  
        for (i = 0; i < 10; i++) {  
            sum += i;  
        }  
        assert i == 10;  
        assert sum > 10 && sum < 9 * 10 : "sum is " + sum;  
    }  
}
```

What to assert? (2)

- **Cases of conditional statements**
 - Assert the condition for the “default” and “else” branch.

```
if (i % 3 == 0) { ... }  
else if (i % 3 == 1) { ... }  
else {  
    // We know (i % 3 == 2) must be true here  
    assert i % 3 == 2 : i;  
}
```


What to assert? (3)

- **Control flow invariant**
 - If a program should never reach a point, then a constant false assertion may be used

```
void foo() {  
    for (...) {  
        if (...)  
            return;  
    }  
    assert false;  
}
```

What to assert? (4)

- **Loop invariant**

- **Initialization** – condition that is true before first iteration
- **Maintenance** – *if* it is true before an iteration, *then* it remains true before the next iteration
- **Termination** – when loop terminates the invariant gives a useful property to show the correctness of the algorithm

Example: Insertion Sort

- Insertion sort: sort n numbers in $A[0..n-1]$.
- Input: n , numbers in A
- Output: A in sorted order: $\forall i \in [1..n-1], A[i-1] \leq A[i]$

```
for j=1 to length(A)-1
    key=A[j]
    i=j-1
    while i>0 and A[i]>key
        A[i+1]=A[i]
        i--
    A[i+1]=key
```

Example: Insertion Sort

- **Loop invariant:** *At the start of each **for** loop, $A[0...j-1]$ consists of elements originally in $A[0...j-1]$ but in sorted order*

```
for j=1 to length(A)-1
    key=A[j]
    i=j-1
    while i>=0 and A[i]>key
        A[i+1]=A[i]
        i--
    A[i+1]=key
```

Example: Insertion Sort

- **Loop invariant:** *At the start of each **for** loop, $A[0 \dots j-1]$ consists of elements originally in $A[0 \dots j-1]$ but in sorted order*

```
for j=1 to length(A)-1
    key=A[j]
    i=j-1
    while i>=0 and A[i]>key
        A[i+1]=A[i]
        i--
    A[i+1]=key
```

- **Initialization:** $j = 1$, the invariant trivially holds because $A[0]$ is a sorted array.

Example: Insertion Sort

- **Loop invariant:** *At the start of each **for** loop, $A[0...j-1]$ consists of elements originally in $A[0...j-1]$ but in sorted order*

```

for j=1 to length(A)-1
    key=A[j]
    i=j-1
    while i>=0 and A[i]>key
        A[i+1]=A[i]
        i--
    A[i+1]=key

```

- **Maintenance:** the inner **while** loop finds the position i with $A[i] \leq key$, and shifts $A[j-1]$, $A[j-2]$, ..., $A[i+1]$ right by one position. Then key , formerly known as $A[j]$, is placed in position $i+1$ so that $A[i] \leq A[i+1] < A[i+2]$.

$A[1...j-1]$ sorted + $A[j] \rightarrow A[1...j]$ sorted

Example: Insertion Sort

- **Loop invariant:** *At the start of each **for** loop, $A[0...j-1]$ consists of elements originally in $A[0...j-1]$ but in sorted order*

```
for j=1 to length(A)-1
    key=A[j]
    i=j-1
    while i>=0 and A[i]>key
        A[i+1]=A[i]
        i--
    A[i+1]=key
```

- **Termination:** the loop terminates, when $j=n$. Then the invariant states: *" $A[0...n-1]$ consists of elements originally in $A[0...n-1]$ but in sorted order."*

Example: Insertion Sort

- **Loop invariant:** *At the start of each **for** loop, $A[0...j-1]$ consists of elements originally in $A[0...j-1]$ but in sorted order*

```
for j=1 to length(A)-1
    key=A[j]
    i=j-1
    while i>=0 and A[i]>key
        A[i+1]=A[i]
        i--
    A[i+1]=key
```

- Implementation with Java assertion

```
for (int k=1; k<=j-1;k++)assert A[k]>=A[k-1];
```

```
int sum_old=0, sum_new=0;
```

```
for (int k=0; k<=j-1;k++)sum_old+=A[k];
```

```
for (int k=0; k<=j-1;k++)sum_new+=A[k];
```

```
assert sum_old==sum_new;
```


Issue: Assertion or Exception?

Example 1:

```
public void setRadius(double newRadius) {  
    assert newRadius >= 0; or  
    if(newRadius<0) throw new IllegalArgumentException(); ?  
    radius = newRadius;  
}
```

Example 2:

```
private int daysOfMonth(int month)  
    switch (month) {  
        case 1: ... ; break;  
        case 2: ... ; break;  
        ...  
        case 12: ... ; break;  
        default:  
            assert false : "Invalid month: " + month or  
            throw new IllegalArgumentException(); ?  
    }
```

Assertions vs. Exceptions

- When do you use assertions instead of exceptions?
 - Both catch problems in your program, but...
 - The intended usage is *very different!*
- An Exception tells the user of your program that something went wrong
 - You create Exceptions to deal with problems that you know might occur
- An assertion tells *you* that you have a bug
 - You write assertions to state things you know (or *think* you know) about your program
- Exception handling addresses robustness and assertion addresses correctness.

When to throw Exceptions

- Use Exceptions when you:
 - Test whether the parameters to a *public* method or *public* constructor are legal
 - If it's public, then other people besides you can call it.
 - Do any input/output
 - You, the class author, cannot ensure the files will be there or be correct if they are there
- In short,
 - Anything that could go wrong, that you have no control over within *this* class, deserves an Exception. However, ...
 - If you don't know whether the exception will be caught, it might be better to print error message instead.

Quiz: assert, exception, or error message

- Which shall be used to check for 0 args in `main()`?
 - *assert*: bad use!
 - 0 args is due to external input
 - No violation of internal function's precondition
 - *exception*: better
 - Throw an `IllegalArgumentException`.
 - *error message*: best
 - Print error message telling user to type at least one number

Further notes on Java assertions

- If necessary, assertions can be *disallowed* by adding the flag **-source 1.4** to the javac command line.
- By default, assertions are *disabled*. To enable assertions, use the **-enableassertions** (or **-ea**) flag on the java command line.
 - You can also enable or disable assertions for a given package or class.
- The following code, placed at the top of a class, will check whether assertions are *enabled*:

```
static {  
    boolean assertsEnabled = false;  
    assert assertsEnabled = true;  
    if (!assertsEnabled)  
        throw new RuntimeException(  
            "Asserts must be enabled!!!");  
}
```

Assertions in C++

- `assert()` macro

```
#include <assert.h>
assert(expression)
```
- If expression is 1, nothing happens
- If expression is 0, halt program, and display file name, line number and expression
- `#define NDEBUG` turns off assertion testing

assert.h:

```
#ifndef NDEBUG
```

```
#define assert(_Expression) ((void)0)
```

```
#else
```

```
void _wassert(
    const wchar_t * _Message,
    const wchar_t *_File,
    unsigned _Line);
```

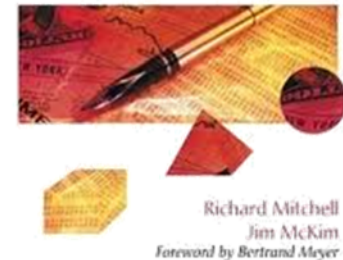
```
#define assert(_Expression)
    (!!( _Expression) || _wassert(
        #_Expression, __FILE__, __LINE__ ))
```

```
#endif
```

Advanced assertion with Design by Contract

- Design by Contract
 - Developed by Bertrand Meyer (2006 ACM Software System Award)
 - Presents a set of principles to produce dependable and robust object oriented software
- Use **contracts** as advanced assertion
 - **Pre-condition**: a condition that must always be true just prior to the execution of a method.
 - **Post-condition**: a condition that must always be true just after the execution of a method.
 - **Class Invariant**: a condition that must always be true after the execution of any method of a class.

Design by Contract,
by Example



What is a Contract?

- There are two parties:
 - **Client** which requests a service
 - **Supplier** which supplies the service
- Contract is the agreement between the client and the supplier
- Two major characteristics of a contract
 - Each party expects some benefits from the contract and is prepared to incur some obligations to obtain them
 - These benefits and obligations are documented in a contract document
- No Hidden Clauses Rule: no requirement other than the obligations written in the contract can be imposed on a party to obtain the benefits

Pre-condition

The programmer who calls a method (***client***) is responsible for **ensuring that the pre-condition of this method is valid** when the method is called.

e.g.

```
double sqrt(double x)  
precondition: x>=0
```

```
bool is_lowercase(char x)  
precondition: x is a letter
```

At this point, my code calls your method, and I make sure that the precondition is valid.



Post-condition

The programmer who writes the method (**supplier**) counts on the precondition being valid, and **ensures that the post-condition becomes true** at the method's end.

*Then my function will execute, when it is done, the postcondition will be true.
I guarantee it.*

e.g.

```
double sqrt(double x)
postcondition: x == return value * return value
```

```
bool is_lowercase(char x)
postcondition: return true if x is between 'a' and 'z'
               return false if x is between 'A' and 'Z'
```



Violations of pre-/post- condition

- What happens if a pre-condition or a post-condition fails (i.e., evaluates to false)?
 - Similar to Java assertion, they can be checked dynamically at run-time to debug the software.
 - A ***pre-condition violation*** would indicate a bug at ...
 - the ***caller***
 - A ***post-condition violation*** would indicate a bug at...
 - the ***callee***

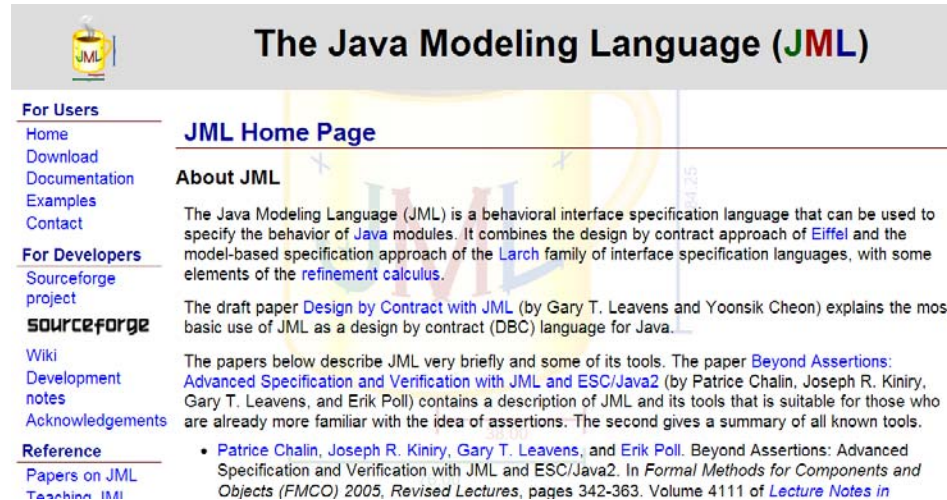
Class Invariants

- A class invariant is an assertion that holds for all instances (objects) of the class
 - A class invariant must be satisfied after creation of every instance of the class.
 - The invariant must be preserved by every method of the class, i.e., if we assume that the invariant holds at the method entry it should hold at the method exit
 - We can think of the class invariant as conjunction added to the pre-condition and post-condition of each method in the class
- For example, a class invariant for a binary search tree:
 - For every node N in the tree,
 - if its left child node N' exists, then $N \geq N'$,
 - if its right child node N'' exists, then $N < N''$.

Tool 1: JML

<http://www.eecs.ucf.edu/~leavens/JML//index.shtml>

- Java Modeling Language (JML)
 - A specification language for Java programs that allows developer to write contracts.
 - Specifications are written as **Java annotation**, which can be compiled and checked by a Java compiler.



The screenshot shows the JML Home Page. At the top, there is a logo for JML and the title "The Java Modeling Language (JML)". Below this, there are two main sections: "For Users" and "For Developers". The "For Users" section includes links to Home, Download, Documentation, Examples, and Contact. The "For Developers" section includes links to Sourceforge project, sourceforge, Wiki, Development notes, Acknowledgements, and Reference. The "Reference" section includes a link to "Papers on JML Teaching JML". The "About JML" section provides a description of JML as a behavioral interface specification language and mentions the draft paper "Design by Contract with JML" by Gary T. Leavens and Yoonsik Cheon. It also lists several papers describing JML and its tools, including "Beyond Assertions: Advanced Specification and Verification with JML and ESC/Java2" by Patrice Chalin, Joseph R. Kiniry, Gary T. Leavens, and Erik Poll.

JMLAnnotations

- JML annotations are added as comments to the Java source code
 - either between `/*@ . . . @*/`
 - or after `//@`
 - These are **annotations** and they are ignored by the Java compiler
- Contracts:
 - Pre-conditions are written as `a requires clauses`
 - Post-conditions are written as `ensures clauses`
 - Class invariants are written as `invariants clauses`
- JML properties are specified as Java boolean expressions
 - JML provides operators to support design by contract style specifications such as `\old` and `\result`
 - JML also provides quantification operators (`\forall`, `\exists`)

Example: pre-/post- condition

```
/*@ requires amount >= 0;  
   @ ensures balance == \old(balance) - amount  
   @           && \result == balance;  
@*/  
int withdraw(int amount) throws PurseException {  
    if (amount <= balance) { balance -= amount; return balance; }  
    else { throw new PurseException("overdrawn by " + amount); }  
}
```

Example: class invariant

```
public class Purse {  
  
    final int MAX_BALANCE;  
    int balance;  
    //@ invariant 0 <= balance && balance <= MAX_BALANCE;  
  
    byte[] pin;  
    /*@ invariant pin != null && pin.length == 4  
        @           && (\forall int i; 0 <= i && i < 4;  
        @           0 <= pin[i] && pin[i] <= 9);  
    @* /  
  
    . . .  
}
```


JML Quantifiers

- JML supports several forms of quantifiers
 - Universal and existential (`\forall` and `\exists`)
 - General quantifiers (`\sum`, `\product`, `\min`, `\max`)
 - Numeric quantifier (`\num_of`)

```
(\forall Student s; class307.contains(s);  
    s.getProject() != null)
```

```
(\forall Student s; class307.contains(s) ==>  
    s.getProject() != null)
```

- Without quantifiers, we would need to write loops to specify these types of constraints

JML Quantifiers (cont)

- Quantifier expressions
 - Start with a declaration that is local to the quantifier expression
`(\forall Student s; ...`
 - Followed by an optional range predicate
`... class307.contains(s); ...`
 - Followed by the body of the quantifier
`... s.getProject() != null)`

JML quantifiers (cont)

- `\sum`, `\product`, `\min`, `\max` return the sum, product, min and max of the values of their body expression when the quantified variables satisfy the given range expression
- For example,
`(\sum int x; 1 <= x && x <= 5; x)` denotes the sum of values between 1 and 5 inclusive
- The numerical quantifier, `\num_of`, returns the number of values for quantified variables for which the range and the body predicate are true

Monitoring JML contracts

- **Step 1:** parsing and type-checking Java programs and their JML annotations
 - JML compiler: `jmlc`
- **Step 2:** runtime assertion checking:
 - Test for violations of assertions (pre-/post- conditions, class invariants) during execution
 - JML runtime library: `jmlrac`
- **Optional step:** extended static checking
 - Automatically prove that contracts are never violated at any execution
 - Automatic verification is done statically with model checking.
 - Tool: **ESC/Java**
(<http://kindsoftware.com/products/opensource/ESCJava2/>)

Tool 2: jContractor

<http://jcontractor.sourceforge.net/>

- Contracts in jContractor are written as Java methods that follow a simple naming convention.
 - Assertions are written as Java methods that return a boolean value
- jContractor provides runtime contract checking by instrumenting the bytecode of classes that define contracts.
- jContractor can
 - either add contract checking code to class files to be executed later,
 - or it can instrument classes at runtime as they are loaded.
- Contracts can be written in the class that they apply to, or in a separate contract class.

An Example Class that use jContractor

```
//Must implement cloneable
class Stack implements Cloneable {

    // Must declare the variable OLD
    private Stack OLD;

    public Stack () { ... }
    public Stack (Object [] initialContents) { ... }
    public void push (Object o) { ... }
    public Object pop () { ... }
    public Object peek () { ... }
    public void clear () { ... }
    public int size () { ... }
    public Object clone () { ... }
    private int searchStack (Object o) { ... }
}
```

Preconditions

- Precondition of a method is written as a Boolean method and its name is the method name followed by "_Precondition"
- A method's precondition is checked when execution enters the method.
- Precondition methods return Boolean and take the same arguments as the non-contract method that they correspond to.

```
protected boolean push_Precondition (Object o) {  
    return o != null;  
}  
private boolean searchStack_Precondition (Object o) {  
    return o != null;  
}  
protected boolean Stack_Precondition (Object [] initialContents) {  
    return (initialContents != null) && (initialContents.length > 0);  
}
```

Post-conditions

- Post-condition of a method is written as a Boolean method and its name is the method name followed by "_Postcondition"
- A method's post-condition is checked just before the method returns.
- Post-condition methods return Boolean and take the same arguments as the non-contract method, plus an additional argument of the method's return type. This argument (called RESULT) must be the last in the list, and holds the value returned by the method.

```
protected boolean push_Postcondition (Object o, Void RESULT) {  
    return implementation.contains(o) && (size() == OLD.size() + 1);  
}
```

```
protected boolean size_Postcondition (int RESULT) {  
    return RESULT >= 0;  
}
```

```
protected boolean Stack_Postcondition (Object [] initialContents,  
Void RESULT) {  
    return size() == initialContents.length;  
}
```


Post-conditions

- Post-conditions may refer to the state of the object at method entry through the `OLD` instance variable.
- This variable must be declared private, and must have the same type as the class that contains it.
- If a class defines an `OLD` variable, it must also implement `Cloneable` and provide a `clone()` method.
- When execution enters a method, a clone of the object will be created and stored in `OLD`.

Class invariants

- Class invariants are checked at the entry and exit of every public method in the class.
- The invariant is defined in a method called "_Invariant" that takes no arguments and returns a boolean.

```
protected boolean _Invariant () { return size() >= 0; }
```

Monitoring Contracts at Runtime

- To run a program with dynamic contract checking:
 - `$ java jContractor MyProgram`
- It is also possible to add contract checking code to class files so that they may be run with a usual Java runtime environment.
 - `$ java jInstrument ./MyProgram.class`
 - `$ java MyProgram`

Review: assertion as test oracle

- Simple assertion
 - `assert` statement in Java.
 - What to assert?
 - Difference with exception and `printf` (error-message)
 - Enable/disable java assertion.
- Advanced assertion with Design by Contract
 - Pre-condition, post-condition, class invariant
 - Tool 1: JML
 - Tool 2: jContractor

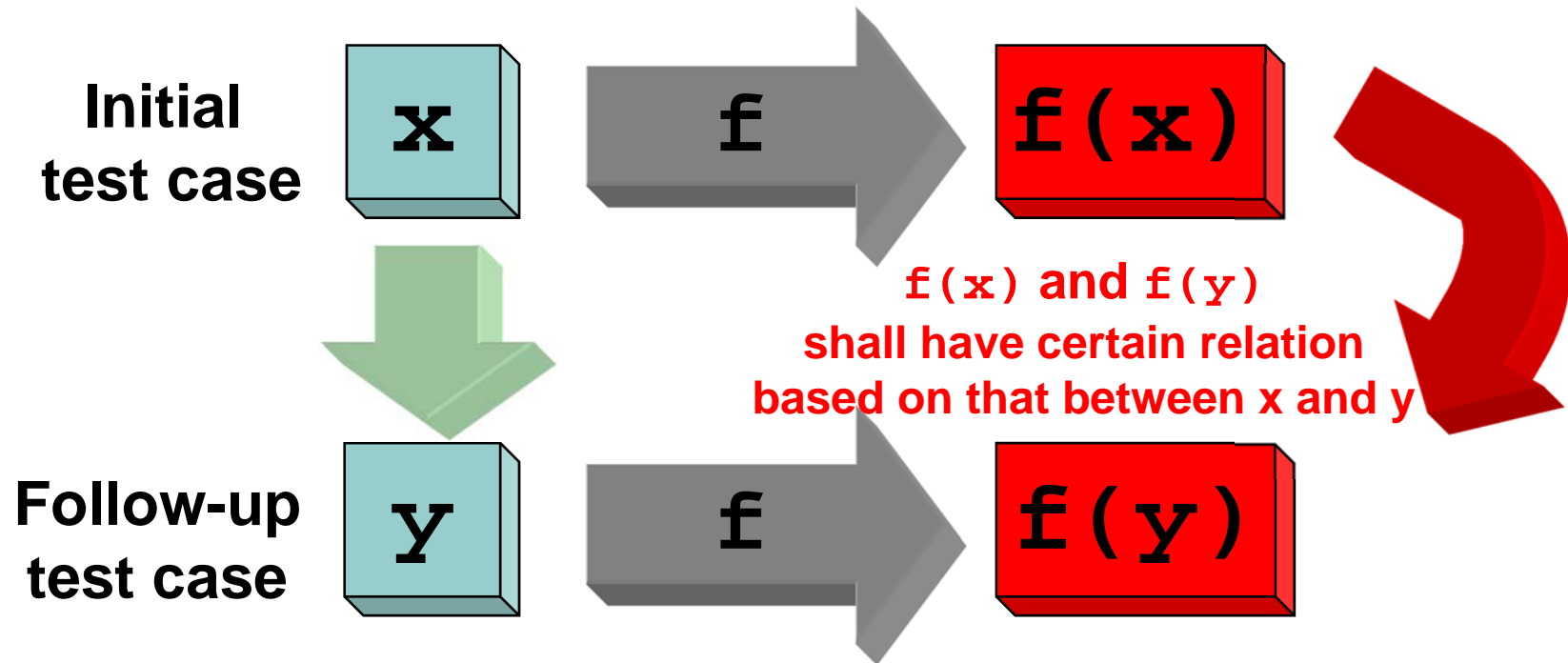
Practical Test Oracle

- Specification as test oracle
- Reference implementation as test oracle
- Program assertion as test oracle
- **Metamorphic property as test oracle**
- Execution profiling as test oracle
- Heuristic and statistical test oracle

Metamorphic property as test oracle

- Many programs have properties such that certain changes to the input yield predictable changes to the output
- We can detect defects in these programs by looking for any violations of these “**metamorphic properties**”
- This is known as “**metamorphic testing**”
 - [T.Y. Chen et al., *Info. & Soft. Tech vol.4*, 2002]

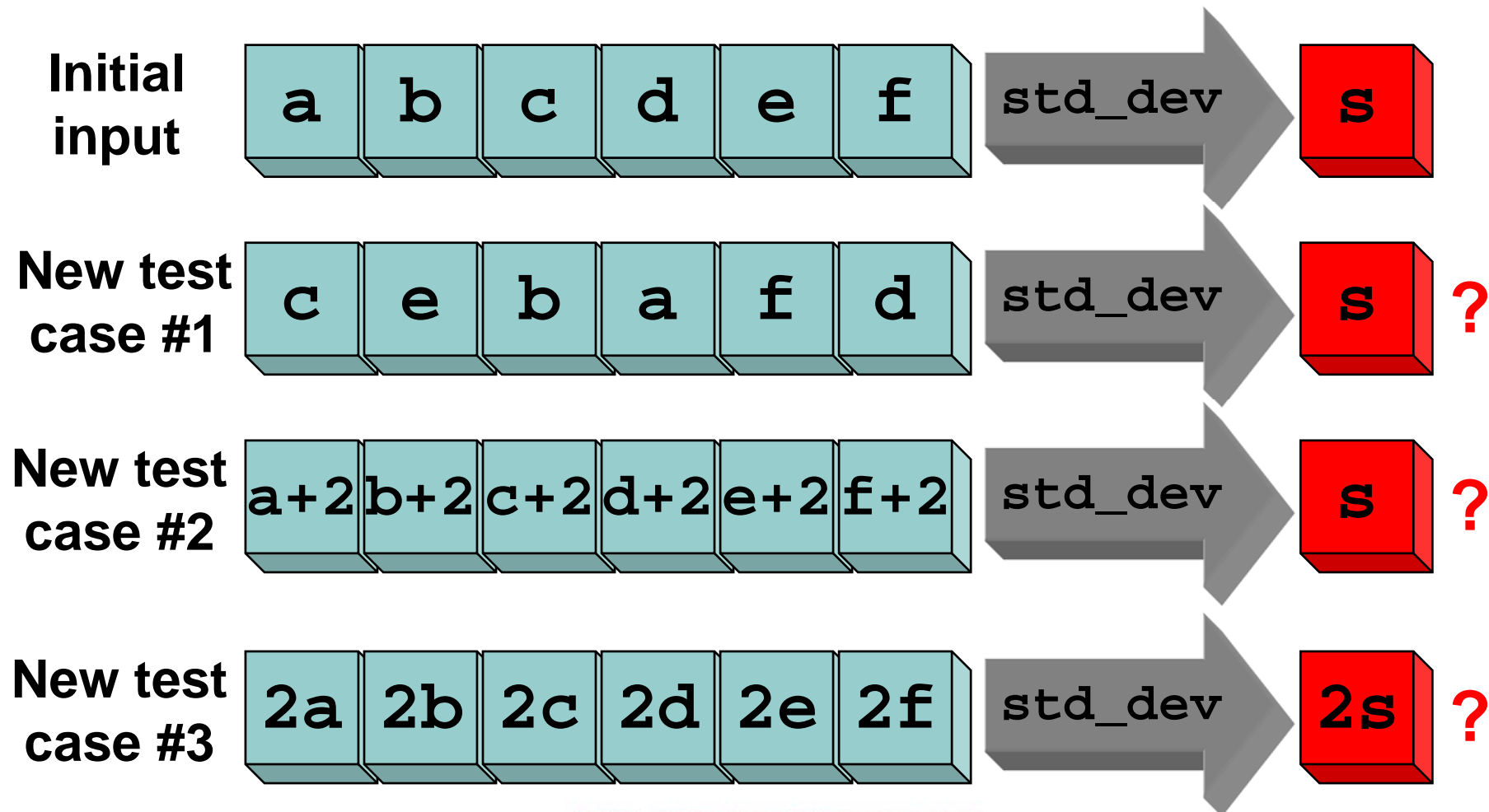
Metamorphic property



- If new test case output $f(y)$ observes the expected relation with $f(x)$, it is **not** necessarily correct
- However, if $f(y)$ does not observe the relation, either $f(x)$ or $f(y)$ – or both! – is wrong

Example

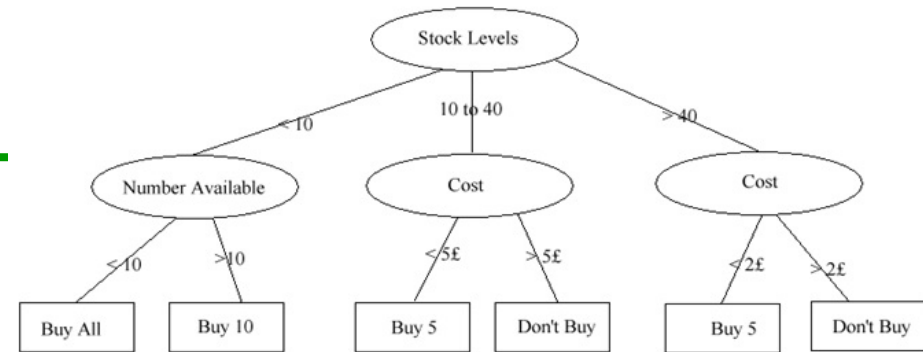
- Consider a function to determine the standard deviation of a set of numbers



Categories of metamorphic properties

- **Additive:** Increase (or decrease) numerical values by a constant
- **Multiplicative:** Multiply numerical values by a constant
- **Permutative:** Randomly permute the order of elements in a set
- **Invertive:** Reverse the order of elements in a set
- **Inclusive:** Add a new element to a set
- **Exclusive:** Remove an element from a set
-
- Machine learning & data-mining apps such as ranking, classification, and anomaly detection exhibit these properties

Example



- Decision tree C4.5

- Permuting the order of the examples in the training data should not affect the model
- If all attribute values in the training data are multiplied by a positive constant, the model should stay the same
- If all attribute values in the training data are increased by a positive constant, the model should stay the same
- Updating a model with a new example should yield the same model created with training data originally containing that example
- If all attribute values in the training data are multiplied by -1, and an example to be classified is also multiplied by -1, the classification should be the same
- Permuting the order of the examples in the testing data should not affect their classification
- If all attribute values in the training data are multiplied by a positive constant, and an example to be classified is also multiplied by the same positive constant, the classification should be the same
- If all attribute values in the training data are increased by a positive constant, and an example to be classified is also increased by the same positive constant, the classification should be the same

Quiz: Metamorphic properties

- You have just developed the following programs. How do you check the correctness of their output using metamorphic properties?
 - A program that computes the length of the shortest path between two nodes in a graph.
 - A program that implements the sin/cos function.
- Can you use metamorphic property as test oracle for one of the programs you wrote previously?

Practical Test Oracle

- Specification as test oracle
- Reference implementation as test oracle
- Program assertion as test oracle
- Metamorphic property as test oracle
- Execution profiling as test oracle
- Heuristic and statistical test oracle

Test oracle: execution profiling

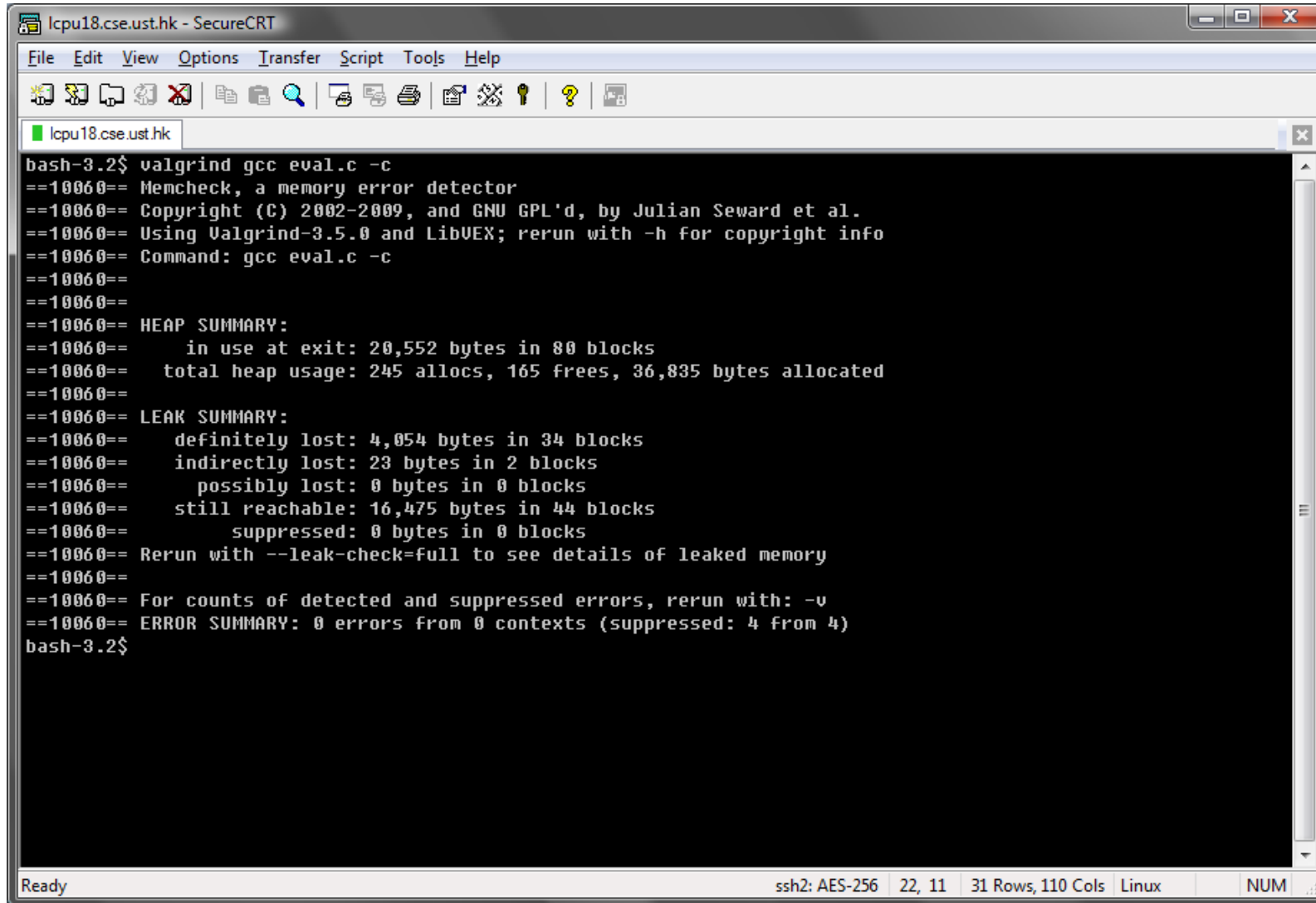
- Execution profilers use instrumentation or sampling to monitor and record program execution details:
 - How many statements have been executed
 - Time elapsed in each function and its descendants
 - How many amounts of memory are allocated/released
 - How many variables have been read/written
 - ...
- Idea: some of them can be directly used to implement a partial oracle

Tool: Valgrind Toolkit

<http://valgrind.org/>

- **Memcheck** is memory debugger
 - Accesses memory it shouldn't.
 - Uses uninitialised values in dangerous ways.
 - Leaks memory.
 - Does bad frees of heap blocks (double frees, mismatched frees).
 - Passes overlapping source and destination memory blocks to memcpy() and related functions.
- **Helgrind** is a thread debugger
 - Finds data races in multithreaded programs
- Using valgrind as a test oracle
 - Memory errors themselves are failure
 - Besides, logic error frequently triggers memory errors, especially in C/C++

Using Valgrind in testing: very simple



The screenshot shows a terminal window titled "lcpu18.cse.ust.hk - SecureCRT". The terminal displays the output of the command `valgrind gcc eval.c -c`. The output includes a copyright notice for Valgrind, a heap summary, and a leak summary. The status bar at the bottom indicates "Ready", "ssh2: AES-256", "22, 11", "31 Rows, 110 Cols", "Linux", and "NUM".

```
lcpu18.cse.ust.hk - SecureCRT
File Edit View Options Transfer Script Tools Help
lcpu18.cse.ust.hk
bash-3.2$ valgrind gcc eval.c -c
==10060== Memcheck, a memory error detector
==10060== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==10060== Using Valgrind-3.5.0 and LibVEX; rerun with -h for copyright info
==10060== Command: gcc eval.c -c
==10060==
==10060== HEAP SUMMARY:
==10060==    in use at exit: 20,552 bytes in 80 blocks
==10060==   total heap usage: 245 allocs, 165 frees, 36,835 bytes allocated
==10060==
==10060== LEAK SUMMARY:
==10060==    definitely lost: 4,054 bytes in 34 blocks
==10060==    indirectly lost: 23 bytes in 2 blocks
==10060==    possibly lost: 0 bytes in 0 blocks
==10060==    still reachable: 16,475 bytes in 44 blocks
==10060==         suppressed: 0 bytes in 0 blocks
==10060== Rerun with --leak-check=full to see details of leaked memory
==10060==
==10060== For counts of detected and suppressed errors, rerun with: -v
==10060== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
bash-3.2$
Ready ssh2: AES-256 22, 11 31 Rows, 110 Cols Linux NUM
```

Practical Test Oracle

- Specification as test oracle
- Reference implementation as test oracle
- Program assertion as test oracle
- Metamorphic property as test oracle
- Execution profiling as test oracle
- **Heuristic and statistical test oracle**

Learning oracle from test runs

- Assumption:
 - In most of the case, the fault is inactive.
- An intuitive solution to identify failed test runs
 - Learning the norms
 - Identifying the outliers

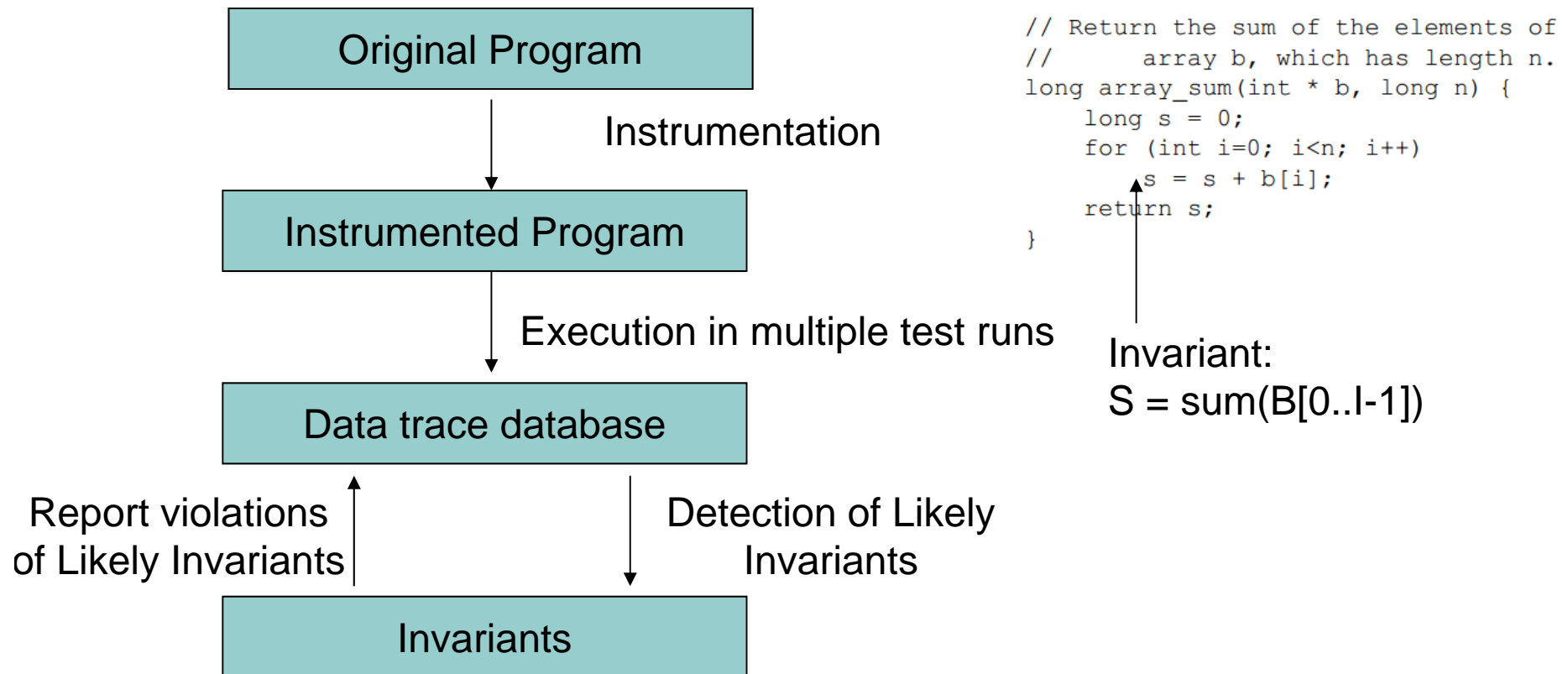
Who is a terrorist?



Tool 1: Daikon

<http://groups.csail.mit.edu/pag/daikon>

- Dynamically Discovering **Likely** Program Invariants



What invariants are inferred?

- Invariants over any variable
 - Constant Value
- Invariants over a single numeric variable
 - Range limits
 - Non-zero
- Invariants over two numeric variables
 - Linear relationship $y = ax + b$
 - Ordering comparison
- Invariants over a single sequence variable
 - Range: Minimum and maximum sequence values
 - Element ordering
- Invariants over two sequence variables
 - Subsequence relationship
 - Reversal
- Invariants over a sequence and a numeric variable
 - Sum, average, ...
- User-implemented templates

How does this idea work?

- Example: BoundedStack

```
public class BoundedStack {
    private int[] elems;
    private int numElems;
    private int max;

    public BoundedStack() { ... }
    public int getNumberOfElements() { ... }
    public int[] getArray() { ... }
    public int maxSize() { ... }
    public boolean isFull() { ... }
    public boolean isEmpty() { ... }
    public boolean isMember(int k) { ... }
    public void push(int k) { ... }
    public int top() { ... }

    public void pop() {
        numElems --;
    }

    public boolean equals(BoundedStack s) {
        if (s.maxSize() != max)
            return false;
        if (s.getNumberOfElements() != numElems)
            return false;
        int[] sElems = s.getArray();
        for (int j=0; j<numElems; j++) {
            if (elems[j] != sElems[j])
                return false;
        }
        return true;
    }
}
```

Faulty

Invariant inferred at the exit of
pop():
 $\text{numElems} \geq 0$

run1: push, pop, push

run2: push, pop, push, pop

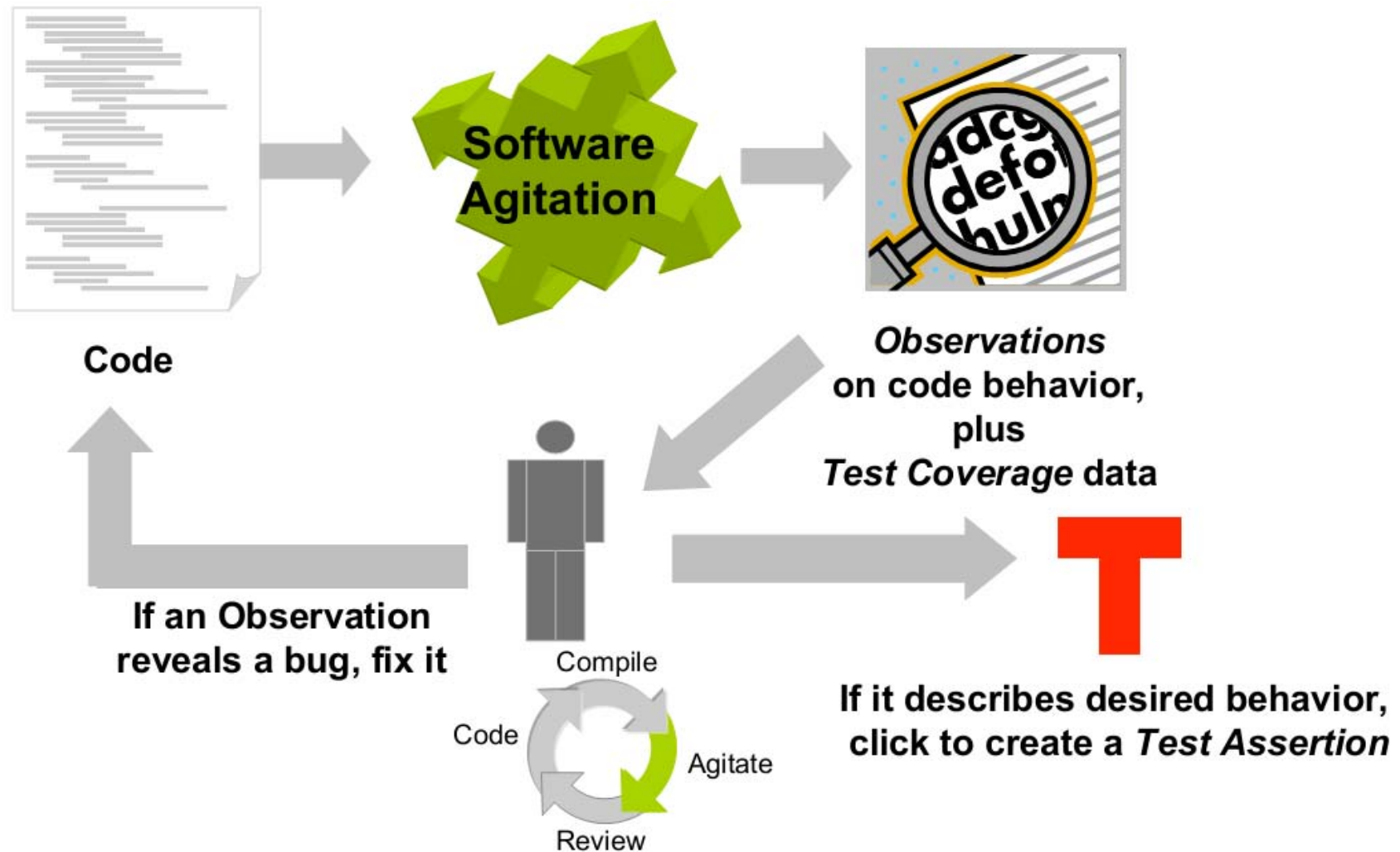
run3: push, push, pop, pop

run4: pop, push, pop

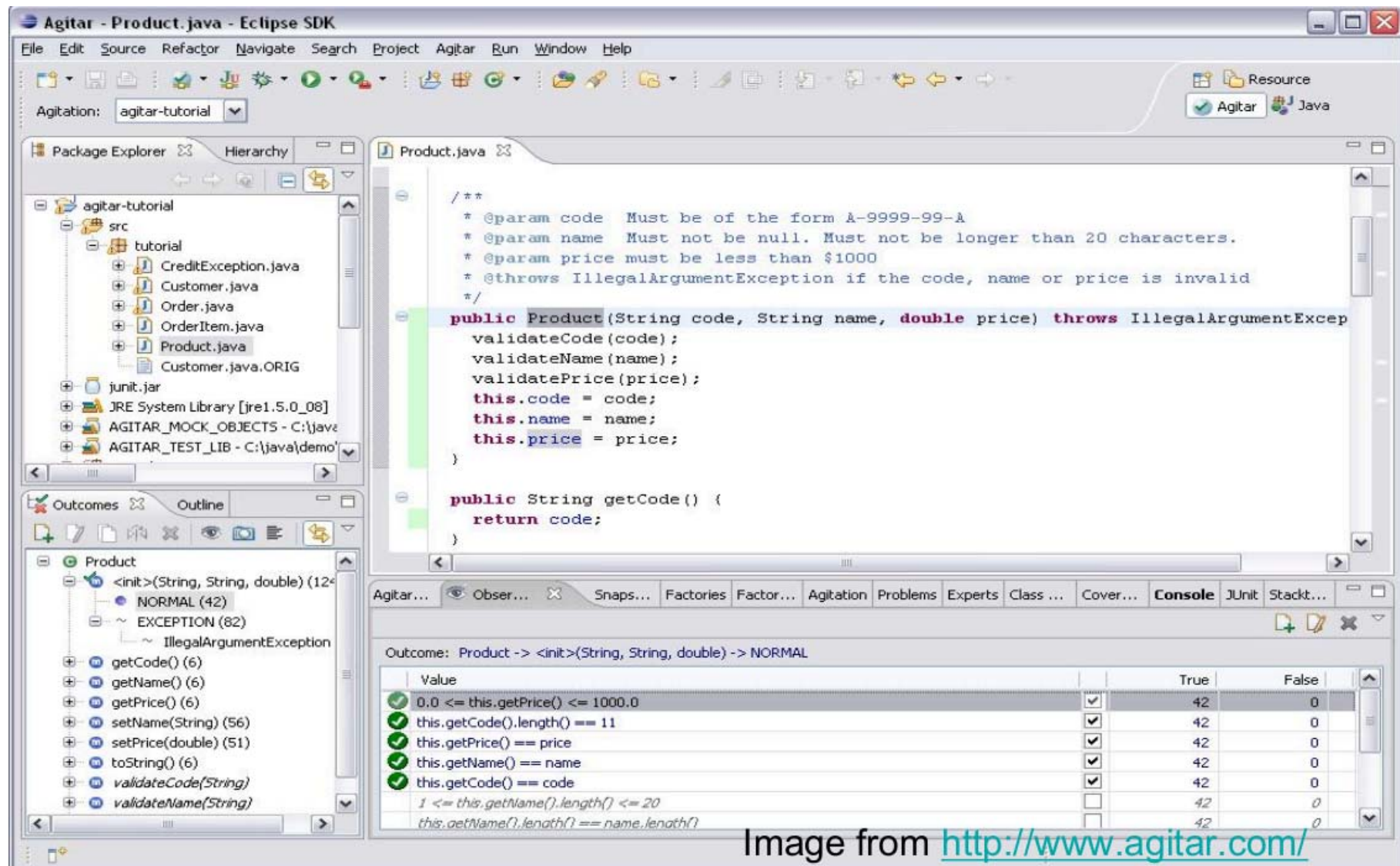
run5: push, push, push, push

Tool 2: AgitarOne

<http://www.agitar.com/solutions/products/agitarone.html>

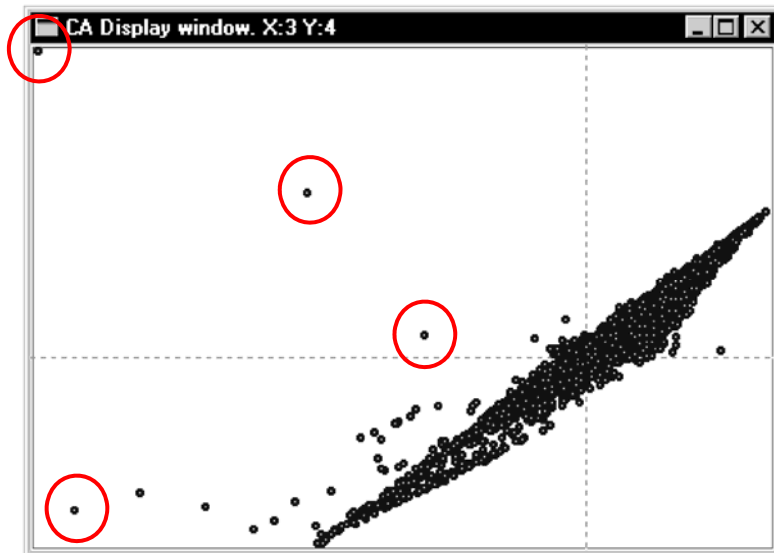


Finding heuristic assertion with AgitarOne



A more radical idea

- Directly use visualization technique to expose outliers



GCC test suite: 7800 test runs
execution profile: function coverage
visualization: multidimensional-scaling
package in matlab

Outliners tend to be failed test runs

ICSE 2000, Multivariate Visualization
in Observation-Based Testing

Review

- Practical Test Oracle
 - Specification as test oracle
 - Reference implementation as test oracle
 - Program assertion as test oracle
 - Metamorphic property as test oracle
 - Execution profiling as test oracle
 - Heuristic and statistical test oracle
- They are not mutually exclusive!

Thank you!

