PRACTICAL SOFTWARE DEBUGGING AND TESTING

Yung-Pin Cheng (鄭永斌)

Associate Professor

Department of Computer Science and Information Engineering

National Central University

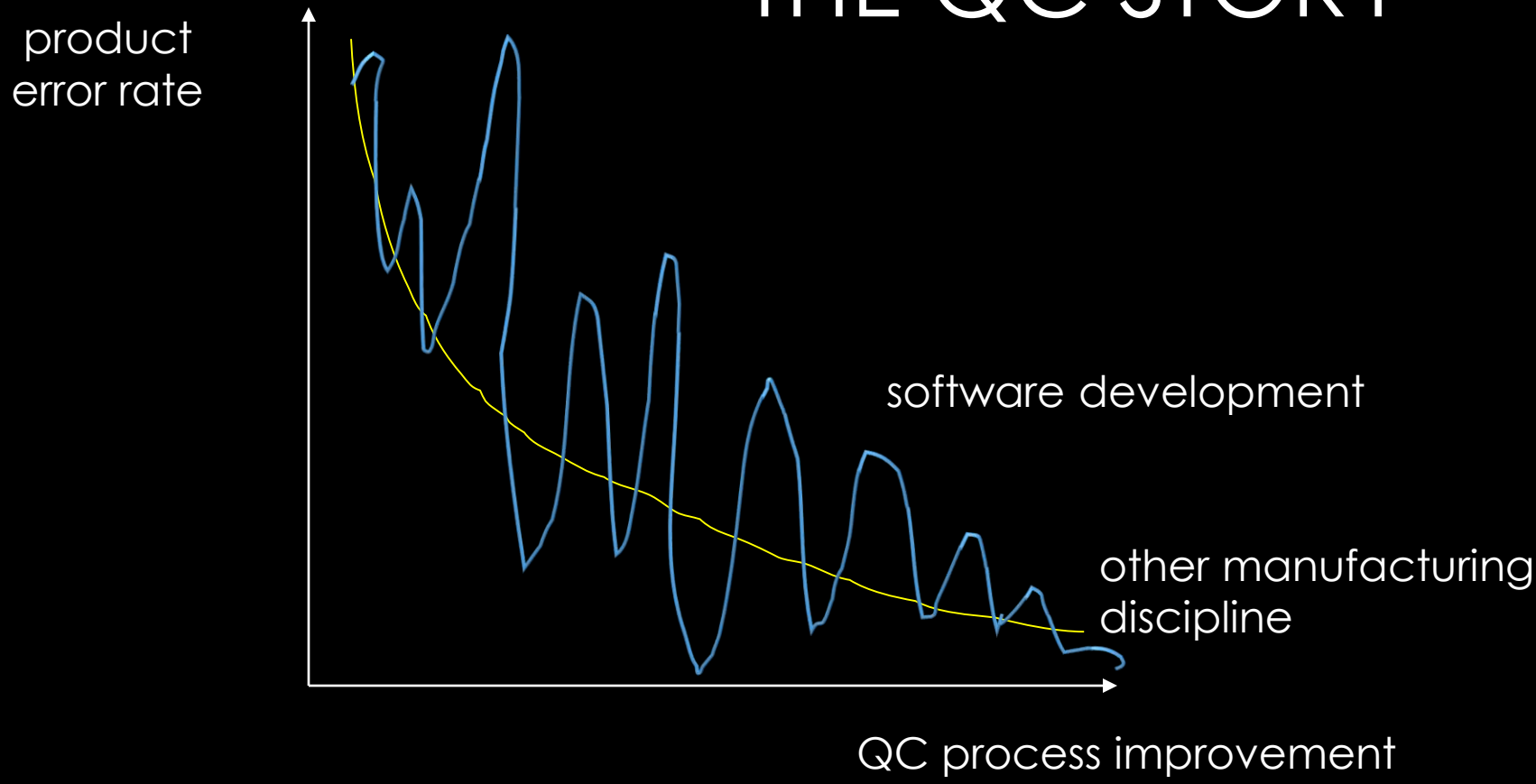
WHERE AM I FROM?

TESTING VS. DEBUGGING

- Two techniques are sometimes mixed and confused
- **Software testing** is
  - use testing tools and testing methodology to find BUGs
  - done by QAs(Testers)
  - Many testing tools are used.
- **debugging** is
  - when a BUG is found, find the root cause, fix the program, and make sure it can pass the tests.
  - by developers
  - use debugging tools.

BUGS ARE EVERYWHERE

THE QC STORY



SOFTWARE QUALITY

- Unlike other engineering disciplines, software industry is one area that **humans** are the main task forces.
- No humans (programmers) are perfect
- This explains why software testing is important because you cannot fully trust **humans.**

HOW PROGRAMMERS CAN CREATE BUGS EASILY

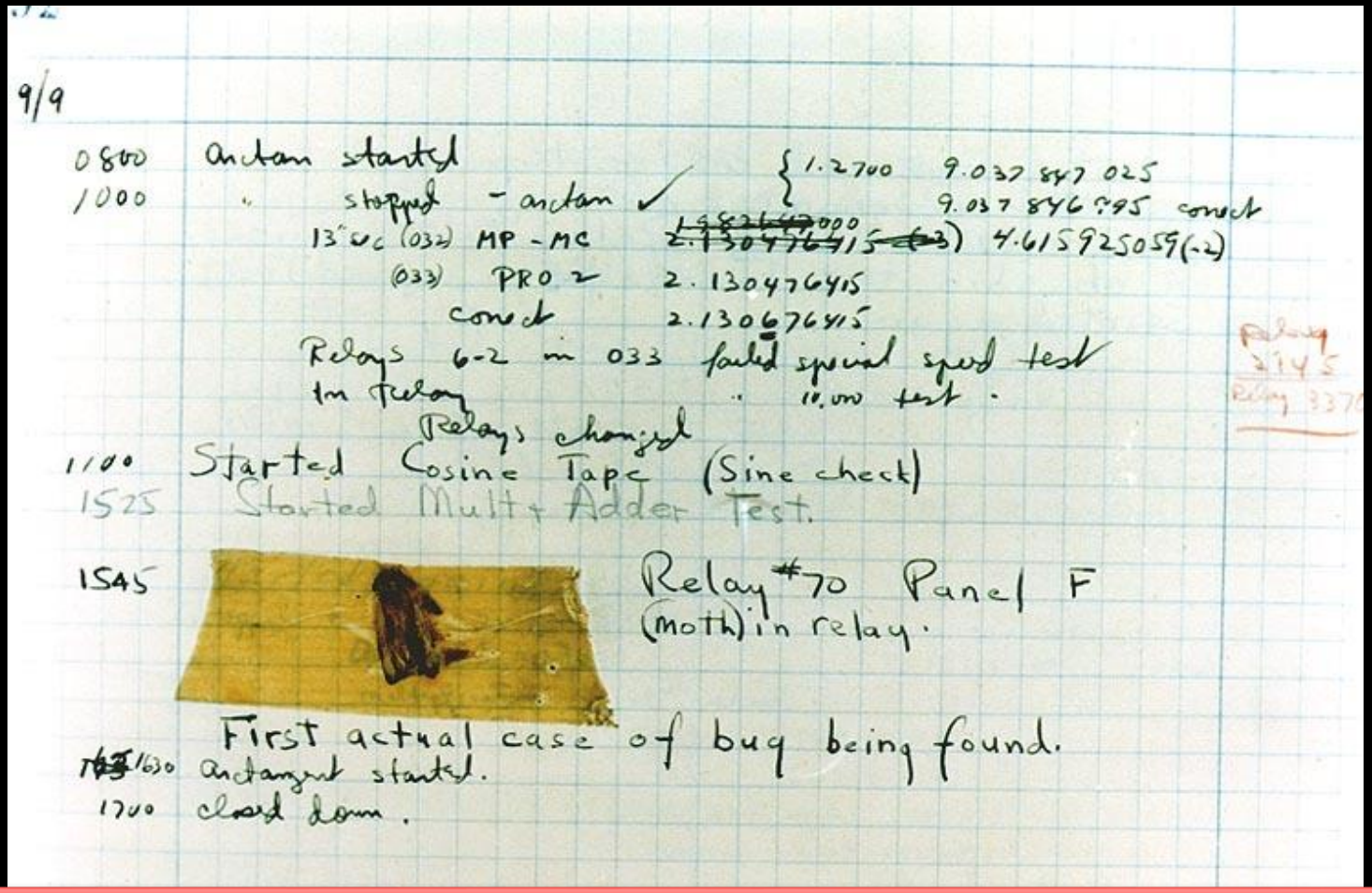
Linux Device Driver bugs

[Dingo: Taming device drivers, 2009]

Driver	#loc	#bugs
USB		
RTL8150 USB-to-Ethernet adapter	827	16
EL1210a USB-to-Ethernet adapter	710	2
KL5kusb101 USB-to-Ethernet apapter	925	15
Generic USB network driver	1028	45
USB hub	2234	67
USB-to-serial converter	989	50
USB mass storage	803	23
Firewire		
IEEE1394 Ethernet controller	1413	22
SBP-2 transport protocol	1713	46
PCI		
Mellanox InfiniHost InfiniBand adapter	11718	123
BNX2 Ethernet adapter	5412	51
i810 frame buffer	2920	16
CMI8338 audio	2660	22
		498

WHAT IS A BUG?

A **software bug** is the common term used to describe an error, flaw, mistake, failure, or fault in a computer program or system that produces an incorrect or unexpected result, or causes it to behave in unintended ways.

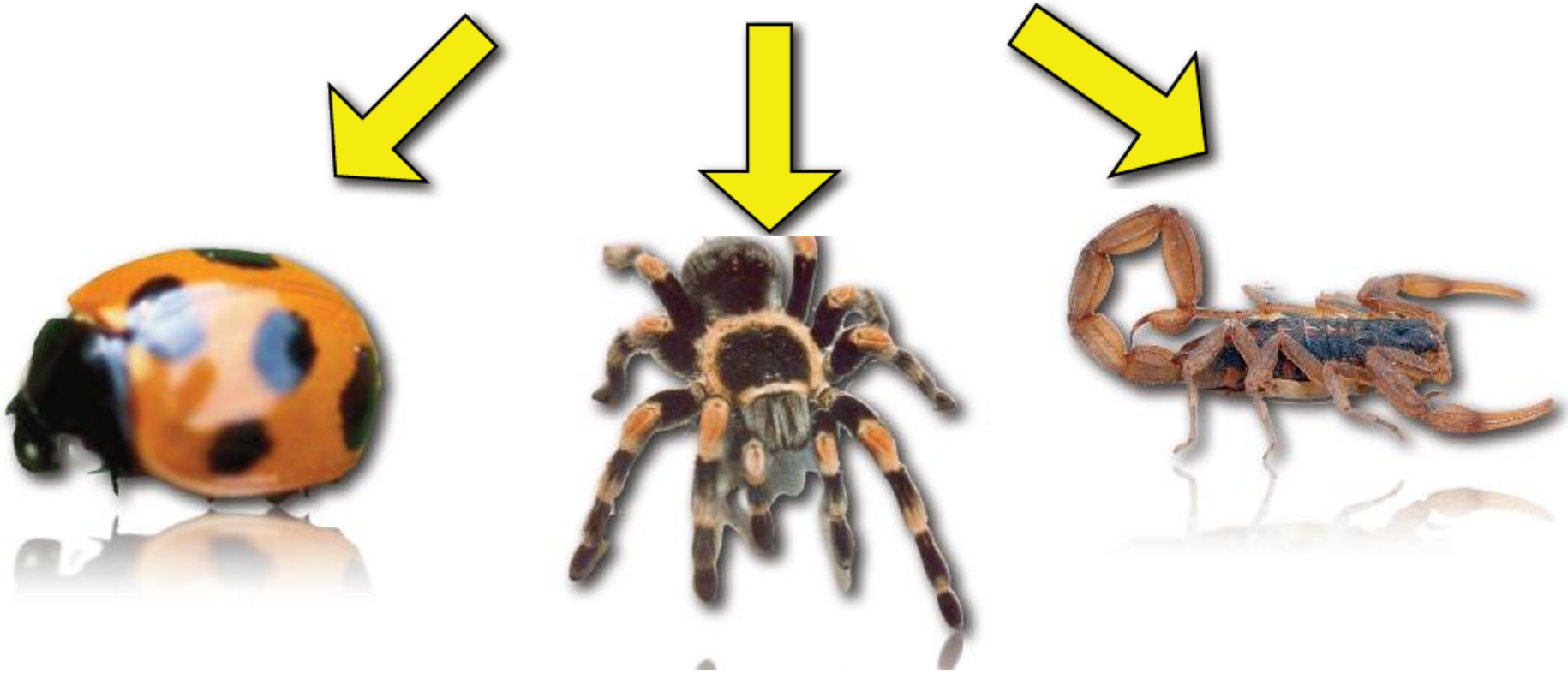


1946 Grace MurrayHopper (who invented COBOL)
find a moth in Mark III National Museum of American History

HOW BUGS ARE PRODUCED?

- programming errors (e.g.: typos)
- Design and logic errors
- Compiler errors (rarely)
- Memory management faults
- Specs errors
- Third party component (library)errors

Bugs

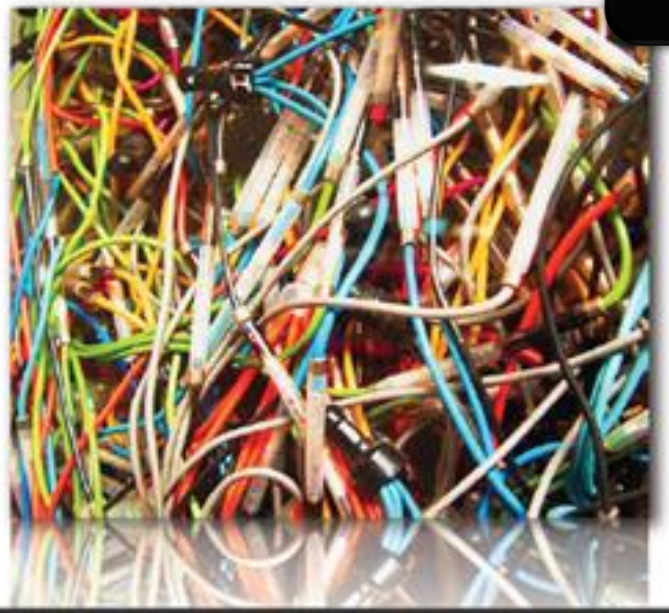


Logical Bugs

created by poor system design , poor data structure, and poor logic planning.

The cost of fixing a bug can be very high, i.e., it may require fixing many code and data structures, which require more testing efforts.

Google



Wiring Bug

Caused by poor hardwiring programming. hardwiring programming produces too many possible execution paths.

Such code lacks of deep understanding and thinking of the problem, which often requires good data structure design and algorithms.

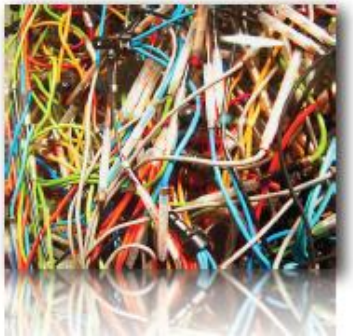
Fixing a bug in the problem may look straightforward, however, it is getting complicated as time goes by

引用自 Google

Rendering Bug
Once you find the bug,
fixing it is straightforward.

Google

Bugs



Cost of Fixing the Bug

	Probability of Finding a Bug	Difficulty of Finding the Bug	Difficulty of Fixing the Bug
Logical Bugs	HIGH	HARD	HARD
Wiring Bugs	MEDIUM	MEDIUM	MEDIUM
Rendering Bugs	MEDIUM	EASY	EASY

COMMENTS

- probability to find the bugs –
  - Low: You need to run many test cases to manifest a bug however, you don't need to design hard test case specifically to manifest the bug
  - High: a bug is easily manifested when you execute some test cases.
- difficulty to find the bugs –
  - Low: you don't need to design tricky test cases to find the bugs.
  - High: You need to design tricky test cases in order to trigger the conditions for manifesting a bug.

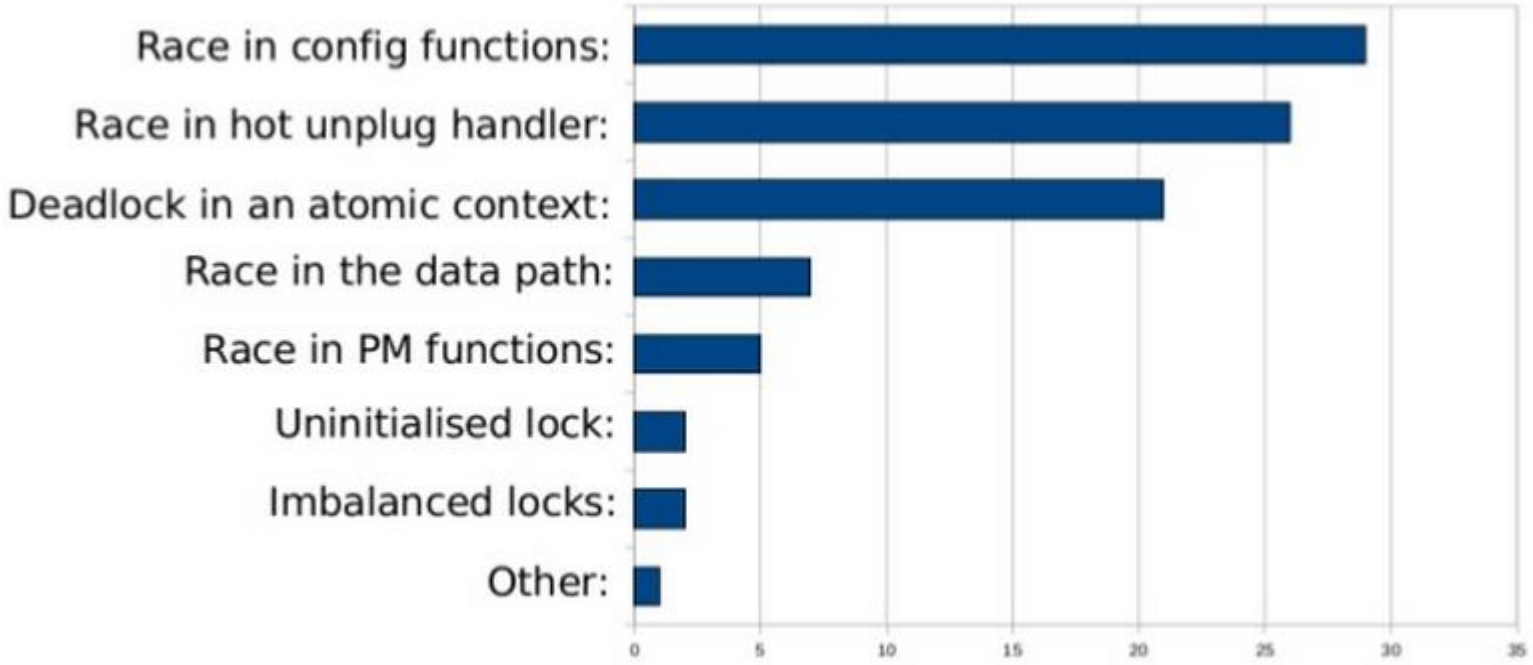
UNUSUAL BUG)

- Due to the progress of modern software techniques, new techniques are introduced or old techniques become prevalent and widely adopt.
  - Multithreading
  - Networking
  - GUI and object-oriented techniques.
    - The thread behaviors can be obscured by encapsulation, inheritance, and polymorphism
  - Signal/CallBack (Interrupt-Driven Like System)
- These techniques can introduce unusual bugs

UNUSUAL BUG

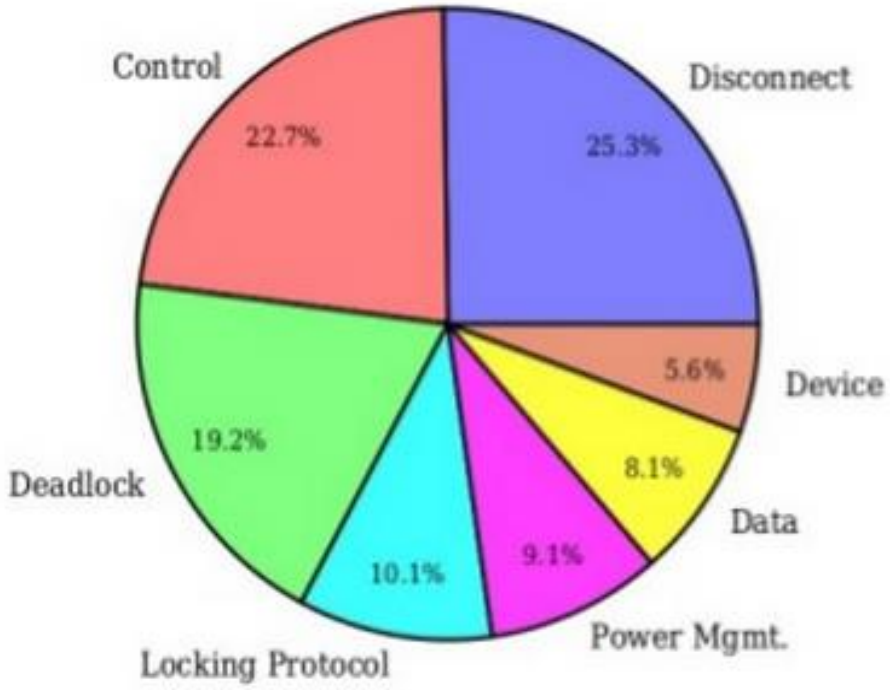
- Heisengbug
  - borrowed from Heisenberg Uncertainty Principle
  - bug that is difficult to reproduce
  - even it is reproducible, it disappears once you **observe** it.
  - program behave differently in debug mode, optimization mode, release mode
- Examples
  - race condition
  - deadlock
  - memory not properly cleaned and initiated.

Concurrency errors



Further study about concurrency bugs

- Markus Peloquin, Lena Olson, Andrew Coonce, University of Wisconsin–Madison, *"Simultaneity Safari: A Study of Concurrency Bugs in Device Drivers"* (2009)
- Types of Device Driver Bugs

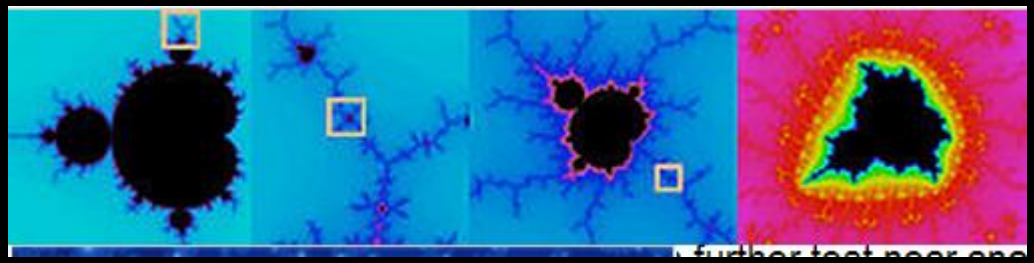


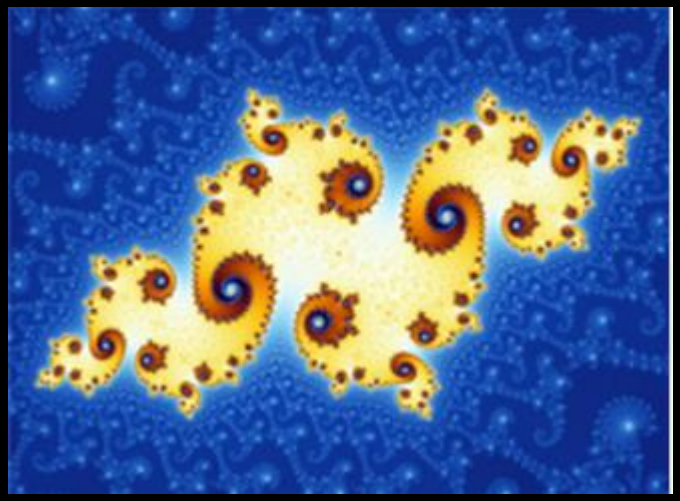
BUG

- Bohrbug
  - A repeatable *bug*; one that manifests reliably under a possibly unknown but well-defined set of conditions. Sometimes it is not easy to trigger the conditions.
- Examples
  - an overflow bug

UNUSUAL BUG

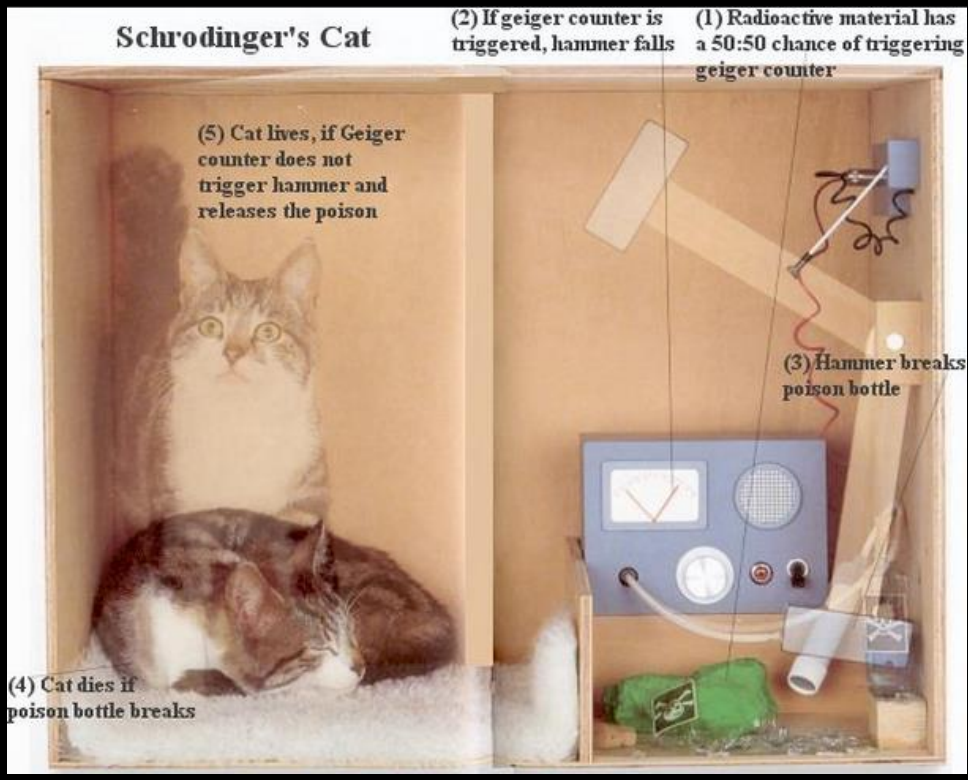
- Mandelbug [from the Mandelbrot set]
- A bug whose underlying causes are so complex and obscure as to make its behavior appear chaotic or even non-deterministic.





UNUSUAL BUG

- Schrödinbug
  - when you read the code and then you know actually it is very easy to trigger such bugs. Such bugs not triggered, you are just lucky.
  - Often written by poor programmers which likes to write weird code and your team never do code review.
  - PS: This programmer should never be hired in the first place.



UNUSUAL BUG

- Statistical bug
  - Introduce random number generator, probability makes your program bugs irreproducible.
  - You many need to accumulate large number of runs to find the bugs

REPRODUCE THE BUGS

- In practices, reproducing a bug may not be an easy/intuitive task, particularly
  - bugs are unusual bugs
  - reproducing the test environment is too costly
  - bug only happens in accumulated data, but you cannot access the data due to security or authentication issues.

DEBUGGER

教育部資訊人才培育計畫

AN OVERVIEW OF DEBUGGING

- Debug is an inevitable process of software development. According to a survey, programmer may spend 1/3 work time (or more) in debugging.
- Seldom taught formally in school.
- CS professors think students can learn the skill by themselves if they write more code (?)
- The difficulties of debugging increases as the system complexity increase.

TYPICAL DEBUGGING PROCESS

1. Reproduce bug (can be a non-trivial task in many applications)
2. Simplify the problem (by smaller test data)
    1. example 1: A program may crash while reading or processing large amount of data. If you can find a simpler test data that can reproduce the bug, debugging can be much more efficient and easy. This is one of the debugging wisdom.
3. Isolate the culprits
    - example 2: there are too many steps that can crash the program. By skipping some steps and replace the steps by input/output pair as much as possible，you can exam if the skipped step create some side effects which crash the program (Divide and Conquer)
4. Use the debugger to trace and examine the code step by step

TRACING TECHNIQUES – PRINT DEBUG

- PRINT debug
  - A basic technique in programming 101
  - insert printf/cout to print variable values at critical places in code.
  - print to file or screen
  - unless you insert "pause" command, the printing is non-stop (unless you know CTRL-S)
  - It is still an important debugging technique. In practice, programmer may create log which is a PRINT debug

TRACING TECHNIQUES – PRINT DEBUG

PRINT debug may have the following problems

- If you need to modify/add print contents, you need to change the code. Changing the code requires re-compilation. It could takes time if the scale of system is large.
- When there are lots of PRINTs you may need to delete/comment them one by one.
- PRINT debug may be used to print formatted output (e.g., you print a tree in a tree layout in the screen). As the visualization code is complicated, bugs can be created in the debugging code.
- In some applications, printing to screen may be not available
  - GUI applications
  - High performance program

EXAMPLE

- Please compare the elapse time of the two programs

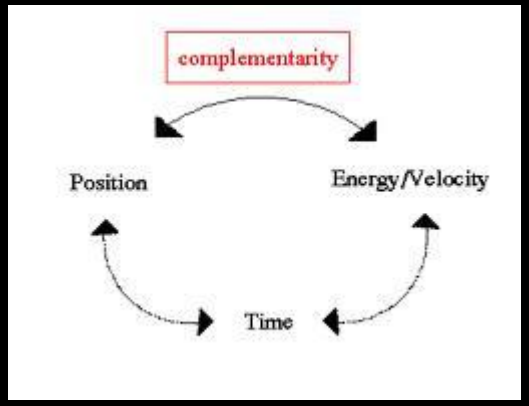
```
for (i=0;i<1000000; i++) {
      value = i * i ;
}
```

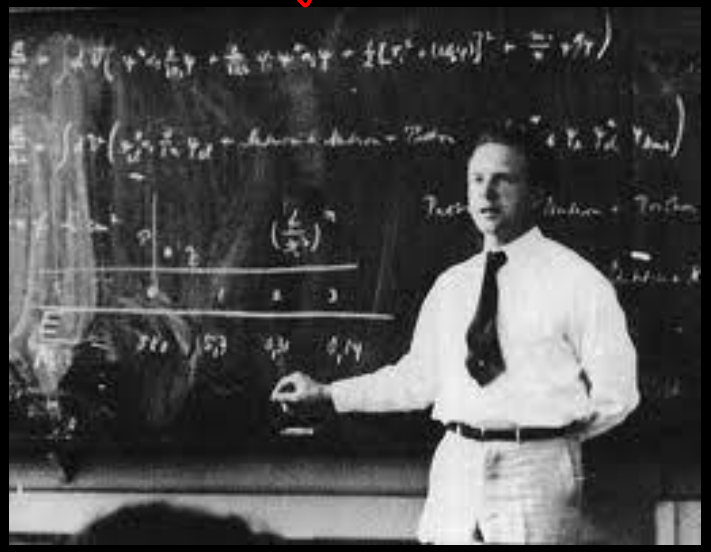
```
for (i=0; i< 1000000; i++) {
      value = i * i ;
      cout << i << endl ;
}
```

- We all took operating systems courses. I/O is very slow.

PROBE EFFECT

- In order to understand the internal program states, you add PRINT commands (so does debugger)
- Because you observe, in someways, you change the program behaviors. This is called probe effect
- Uncertainty principle





PROBE EFFECT IN PRACTICE

- If your program (debuggee) contains only one thread (main()), it is called a sequential program
  - Adding PRINT only decrease performance and does not change the correctness of the program (School exercises are the major kind of the program). Your program simply run slower.
  - Bug can be reproduce repeatedly

PROBE EFFECT

- If your program contains multi-threading, distributed/networked communications, parallel processing
  - Networking apps, multithreading apps, apps with GUIs… Programs in real world belonged to this category.
  - Probe effect (PRINT/debugger) not only slow down your program but can also alter the interleaving of threads.
  - Program may not reach the point you want to reproduce for debugging
  - bugs may not be reproduced
  - Be AWARE !!

TRACING TECHNIQUES - DEBUGGER

- Debugger (System Program)
  - allow you to set break points and there is no need to recompile the system
  - At break point you can watch variables
  - At break point, you can check call stack
  - Source level debug
  - You can execute program one by one and then observe the results immediately and easily.
  - It is much more efficient compared to PRINT debug
  - If you never use it before and don't think it is necessary, it is a sign that your programs are small (ACM programs?)

AN INTRODUCTION TO DEBUGGER

- Debugger is an important system program
  - Any CPU manufacturer wants to attract developers, debugger is a must support system program.
  - System programs include compiler, linker, debugger, assembler …
- debugger is a very complicated system program
- A debugger of its original form is console-mode (command line)

MOST WELL-KNOWN C/C++ DEBUGGER GDB

- **GDB: The GNU Project Debugger (http://www.gnu.org/s/gdb/)**
- **It can debug the following languages**
  - Ada, C, C++, Objective-C, Pascal, Python (and many other languages),
  - Unix based, but it is also ported to Windows, Mac
- **Console mode based**

THE EXAMPLE TO DEBUG A C PROGRAM BY GDB

EX ex1.c

```
#include<stdio.h>

int main(int argc,char * arg[]){
int a,b,c,d,e,f,i;
a = 0;
b = 8;
c = 29;
d = 44;
e = 444;

printf ("%d %d %d %d %d",a,b,c,d,e);

for (i=0;i<argc;i++) printf("@@ %s \n",arg[i]);
return 0;
}
```

ADD DEBUGGING INFORMATION

To use GDB, give the following command

```
is93007@bsd1:[~]$ gcc -g -o ex1 ex1.c
```

gcc: GNU C compiler

- “-g” tells gcc to include debugging information to object files.
- “-o” tells gcc to generate executable
- By default, compiler does not turn on debugging option.
- If debugger is integrated in IDEs (visual studio, eclipse), debug mode is a default mode when you open a project.

GDB STARTUP

```
is93007@bsd1:[~]$ gdb
GNU gdb 4.18 (FreeBSD)
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public L
welcome to change it and/or distribute copies of it under
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warra
This GDB was configured as "i386-unknown-freebsd"

(gdb)
```

READ THE DEBUGGEE

```
(gdb) file ex1
Reading symbols from ex1...Deprecated bfd_read called
at /usr/src/gnu/usr.bin/binutils/gdb/../../../../contrib/gc
Deprecated bfd_read called at /usr/src/gnu/usr.bin/binutils
in fill_symbuf
done.
```

You can also run the debugger by giving debuggee as the first argument

```
is93007@bsd1:[~]$ gdb ex1
```

SETTING ARGUMENTS

- main(int argc, char * argv[])

```
(gdb) show args
Argument list to give program being debugged when it is started is "".

(gdb) set args Barney is handsome
(gdb) show args
Argument list to give program being debugged when it is started is "Barney is handsome".
(gdb) run
Starting program: /.amd/home3/is93/is93007/ex1 Barney is handsome
0 8 29 44 444@@ /.amd/home3/is93/is93007/ex1
@@ Barney
@@ is
@@ handsome

Program exited normally.
```

LIST SOURCE CODE IN GDB

```
(gdb) l
1 #include<stdio.h>
2
3 int main(int argc,char * arg[]){
4
5 int a,b,c,d,e,f,i;
6 a = 0;
7 b = 8;
8 c = 29;
9 d = 44;
10 e = 444;
(gdb) list
11
12 printf ("%d %d %d %d %d",a,b,c,d,e);
13
14
15
16 for (i=0;i<argc;i++) printf("@@ %s \n",arg[i]);
17 return 0;
18
19
20 }
(gdb)
```

SET A BREAK POINT

`break ... if cond`

Set a breakpoint with condition *cond*; evaluate the expression *cond* each time the breakpoint is reached, and stop only if the value is nonzero--that is, if *cond* evaluates as true. `...' stands for one of the possible arguments described above (or no argument) specifying where to break. See section Break conditions, for more information on breakpoint conditions.

`rbreak regex`

Set breakpoints on all functions matching the regular expression *regex*. This command sets an unconditional breakpoint on all matches, printing a list of all breakpoints it set. Once these breakpoints are set, they are treated just like the breakpoints set with the `break` command. You can delete them, disable them, or make them conditional the same way as any other breakpoint.

SET A BREAK POINT AND RUN

```
(gdb) b 8
Breakpoint 1 at 0x80484b4: file ex1.c, line 8.
(gdb) info break
Num Type Disp Enb Address What
1 breakpoint keep y 0x080484b4 in main at ex1.c:8
(gdb) r
Starting program: /.amd/home3/is93/is93007/ex1

Breakpoint 1, main (argc=1, arg=0xbfbffaec) at ex1.c:8
8 c = 29;
(gdb)
```

PRINT/WATCH VARIABLES

When your program stop at a break point you can use print to print a variable

```
(gdb) print e
e = 444
```

If you want your program print the variable every time a break point is hit, use display

```
(gdb) display e
```

CONTINUE YOUR PROGRAM

`continue [ignore-count]`
`c [ignore-count]`
`fg [ignore-count]`

Resume program execution, at the address where your program last stopped; any breakpoints set at that address are bypassed. The optional argument *ignore-count* allows you to specify a further number of times to ignore a breakpoint at this location; its effect is like that of `ignore` (see section Break conditions).

`step`

Continue running your program until control reaches a different source line, then stop it and return control to GDB. This command is abbreviated `s`.

Warning: If you use the `step` command while control is within a function that was compiled without debugging information, execution proceeds until control reaches a function that does have debugging information. Likewise, it will not step into a function which is compiled without debugging information. To step through functions without debugging information, use the `stepi` command, described below.

`step count`

Continue running as in `step`, but do so *count* times. If a breakpoint is reached, or a signal not related to stepping occurs before *count* steps, stepping stops right away.

`next [count]`

Continue to the next source line in the current (innermost) stack frame. This is similar to `step`, but function calls that appear within the line of code are executed without stopping. Execution stops when control reaches a different line of code at the original stack level that was executing when you gave the `next` command. This command is abbreviated `n`.

TRACEPOINTS

The **trace** command defines a tracepoint, which is a point in the target program where the debugger will **briefly stop, collect some data,** and then allow the program to continue. Setting a tracepoint or changing its commands doesn't take effect until the next `tstart` command; thus, you cannot change the tracepoint attributes once a trace experiment is running.

Here are some examples of using the `trace` command:

```
(gdb) trace foo.c:121     // a source file and line number
(gdb) trace +2            // 2 lines forward
(gdb) trace my_function   // first source line of function
(gdb) trace *my_function // EXACT start address of function
(gdb) trace *0x2117c4     // an address
```

TURN ON DEBUGGING INFORMATION

- Debugging information will increase10-20%的 object file
- Using compiler's optimization option may make your debugging information useless.

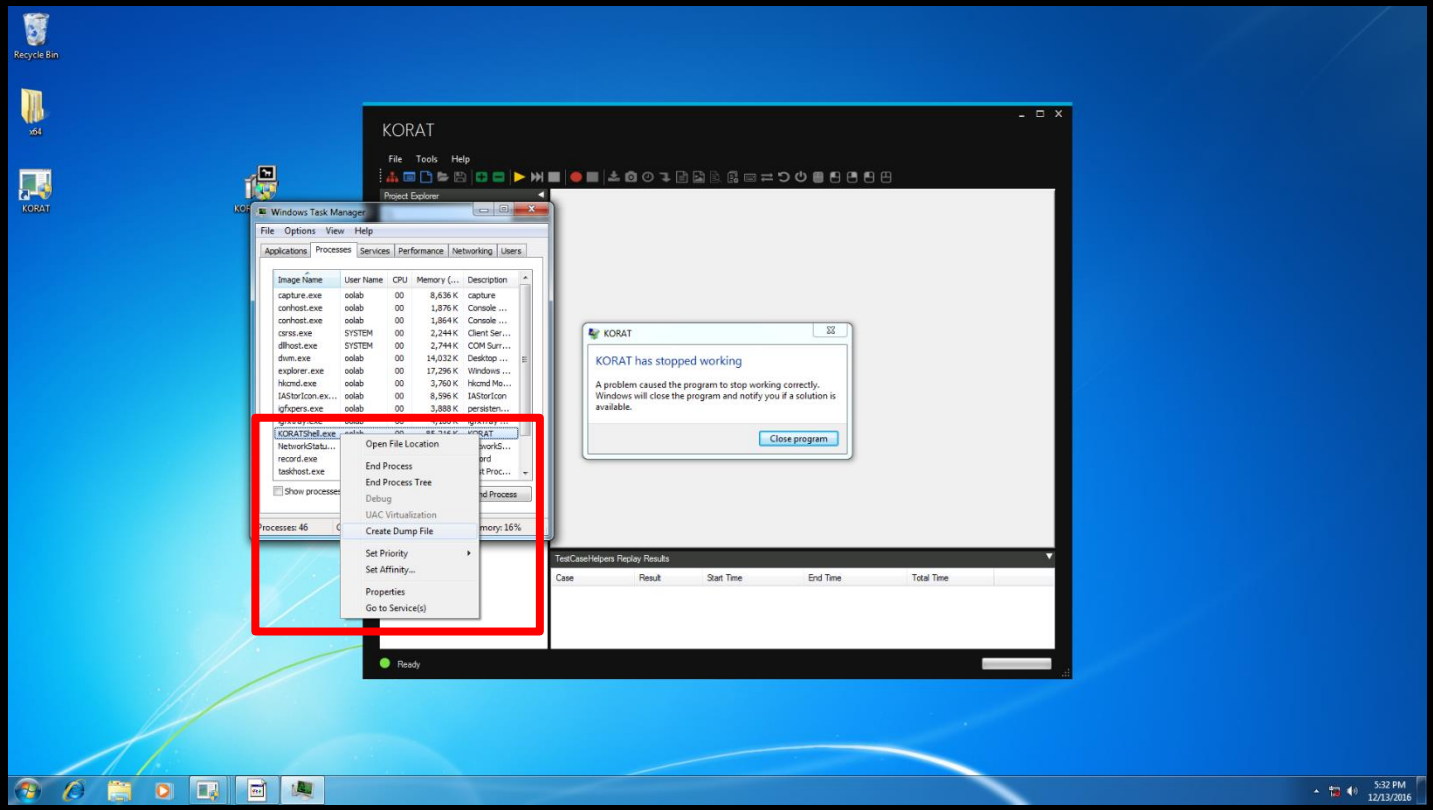
CORE DUMP ANALYSIS

```
$ ./crash -p param1 -o param2
Segmentation fault (core dumped)
$ gdb ./crash core
GNU gdb (GDB) 7.1-ubuntu
...
Core was generated by `./crash -p param1 -o param2'. <<<<< See this line shows cra
Program terminated with signal 11, Segmentation fault.
#0  __strlen_ia32 () at ../sysdeps/i386/i686/multiarch/../../i586/strlen.S:99
99  ../sysdeps/i386/i686/multiarch/../../i586/strlen.S: No such file or directory.
    in ../sysdeps/i386/i686/multiarch/../../i586/strlen.S
(gdb)
```

THE STORY OF AN UNUSUAL BUG

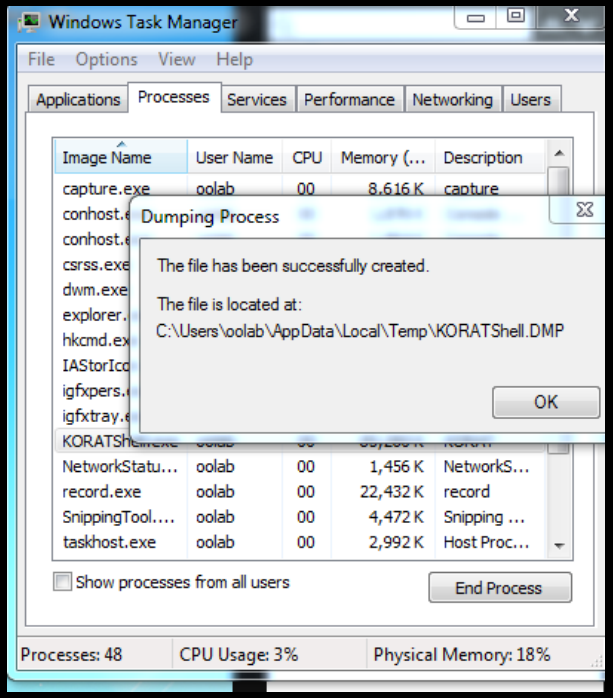
- In our lab, we have a system called KORAT
- It is intensively used daily in ADLINK (2nd largest industrial computer company)
- One day, new features/many lines of code are added
- Everything tested in debugging mode is fine
- Unfortunately, when the program is packed into a release build and it crashes when a key is pressed
- Students were stuck. The question is how to debug it?

WHEN PROGRAM "KORATSHELL" CRASH …. OPEN TASK MANAGER AND CREATE DUMP FILE



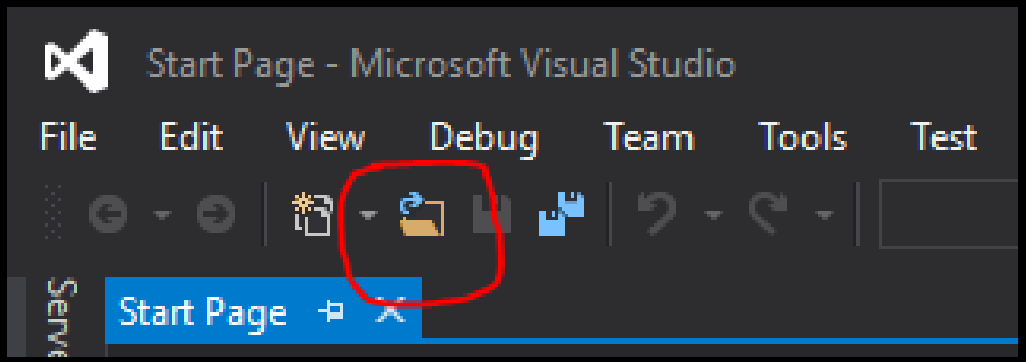
教育部顧問室
Ministry of Education
Advisory Office

TO GET THE DUMP FILE

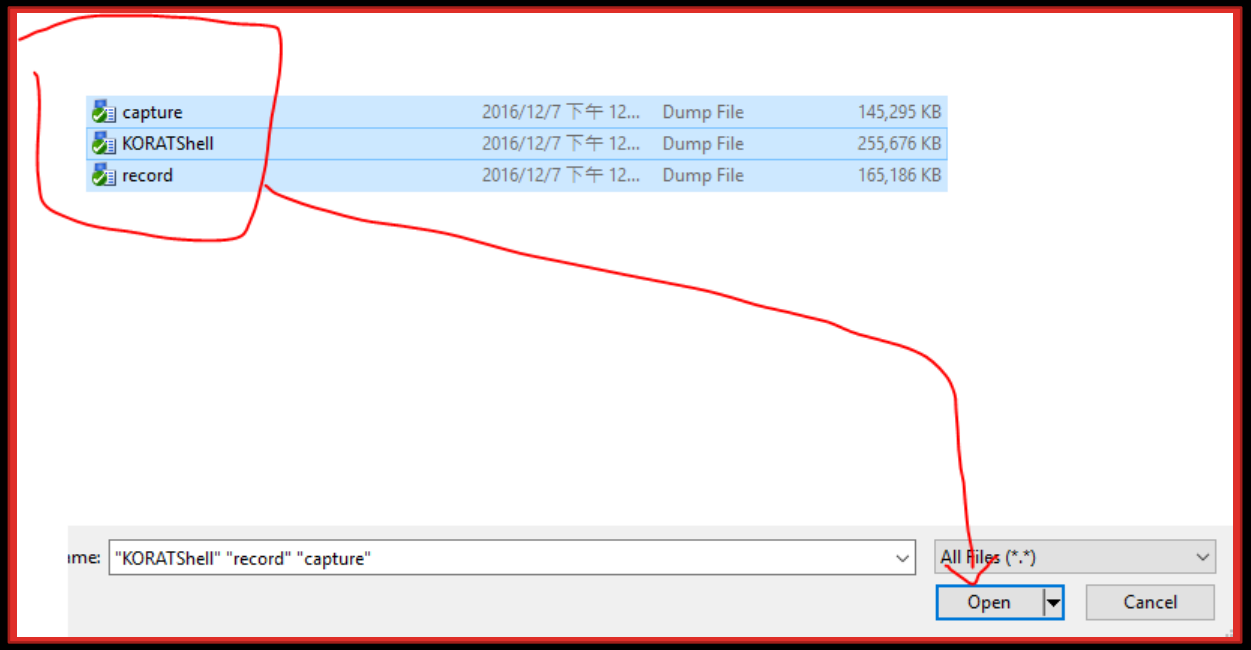


教育部顧問室
Ministry of Education
Advisory Office

OPEN VS AND CLICK “OPEN FILE”

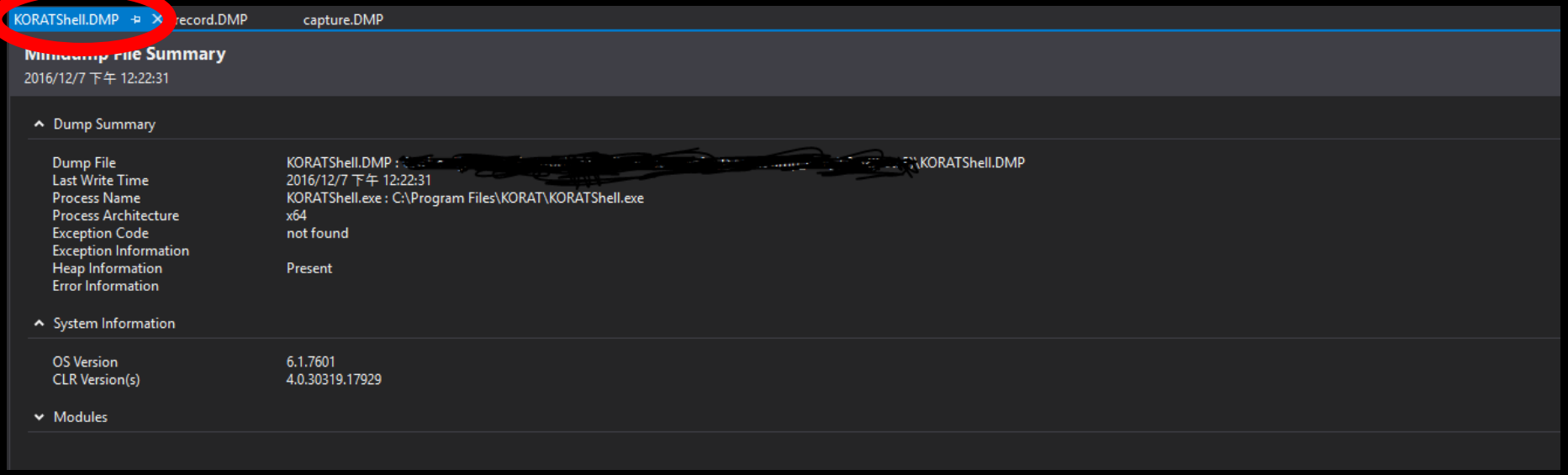


CHOOSE THE DUMP FILE AND OPEN IT



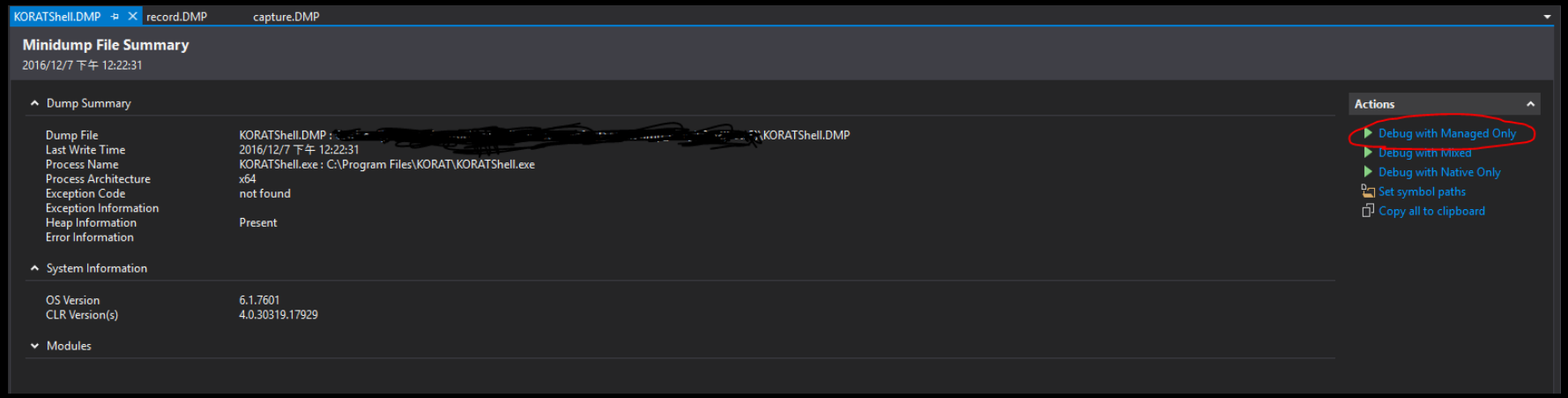
教育部顧問室 Ministry of Education Advisory Office

SWITCH TAB TO "KORATSHELL.DMP"

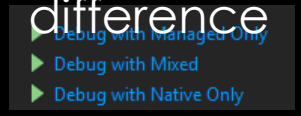


教育部顧問室
Ministry of Education
Advisory Office

CLICK THE “DEBUG WITH MANAGED ONLY”

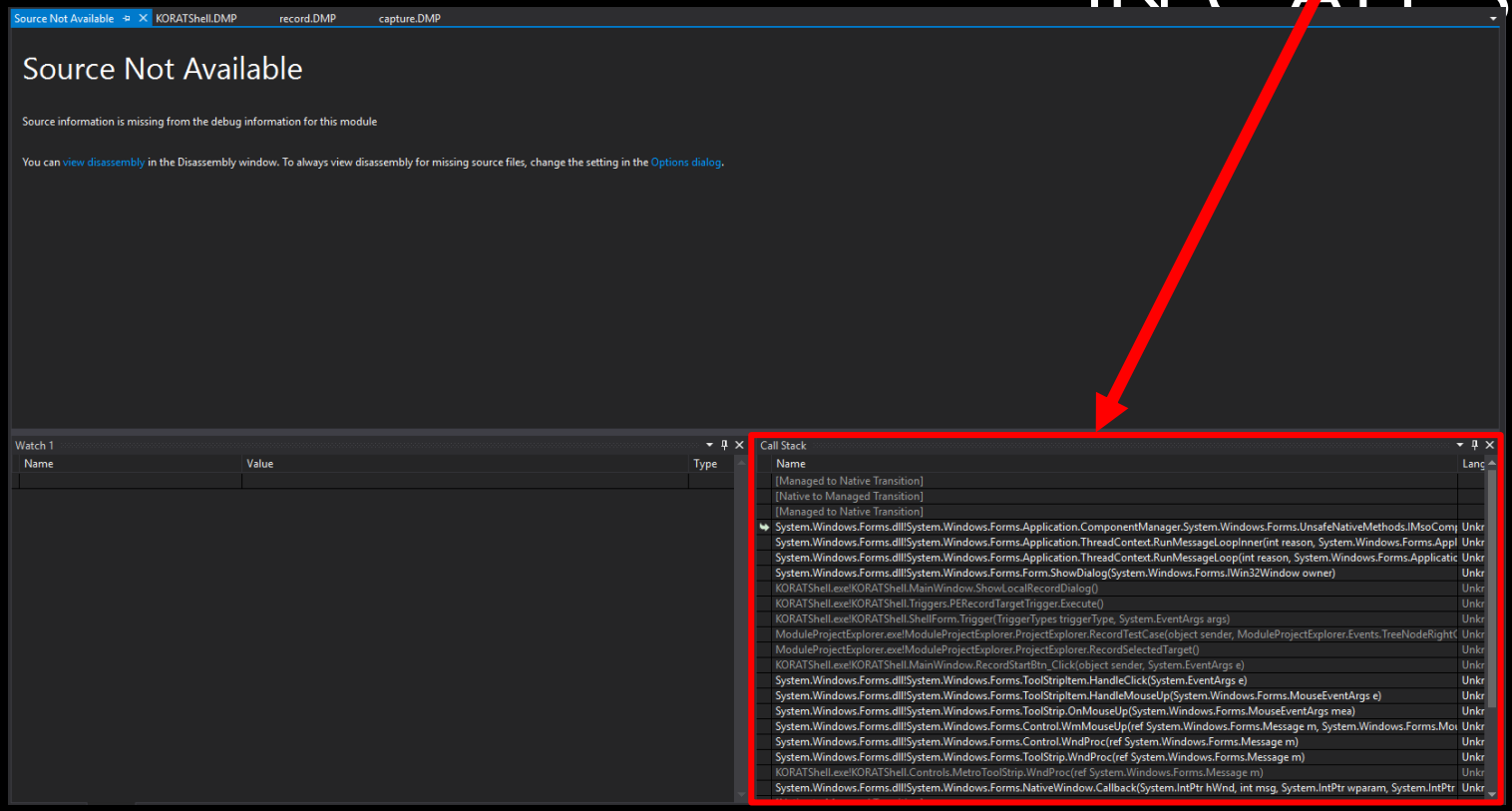


p.s : please see the reference page to know difference



教育部顧問室
Ministry of Education
Advisory Office

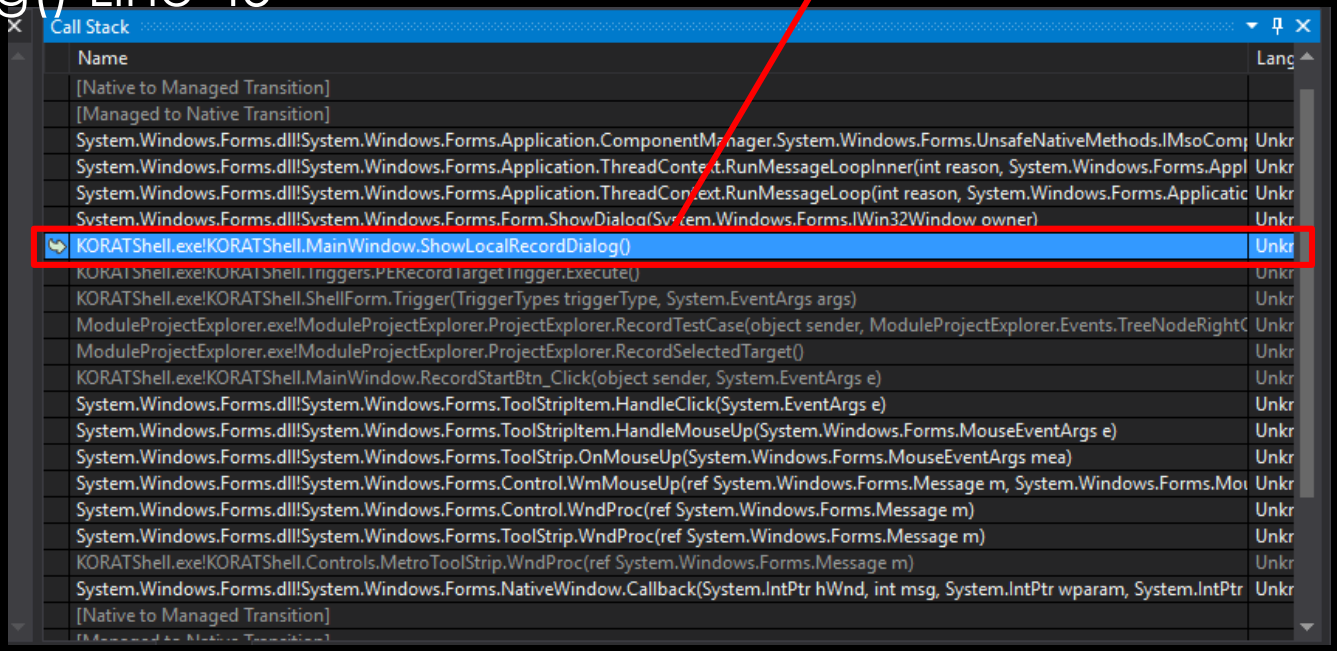
NOW, WE NEED TO FIND OUR CODE IN CALL STACK



教育部顧問室
Ministry of Education
Advisory Office

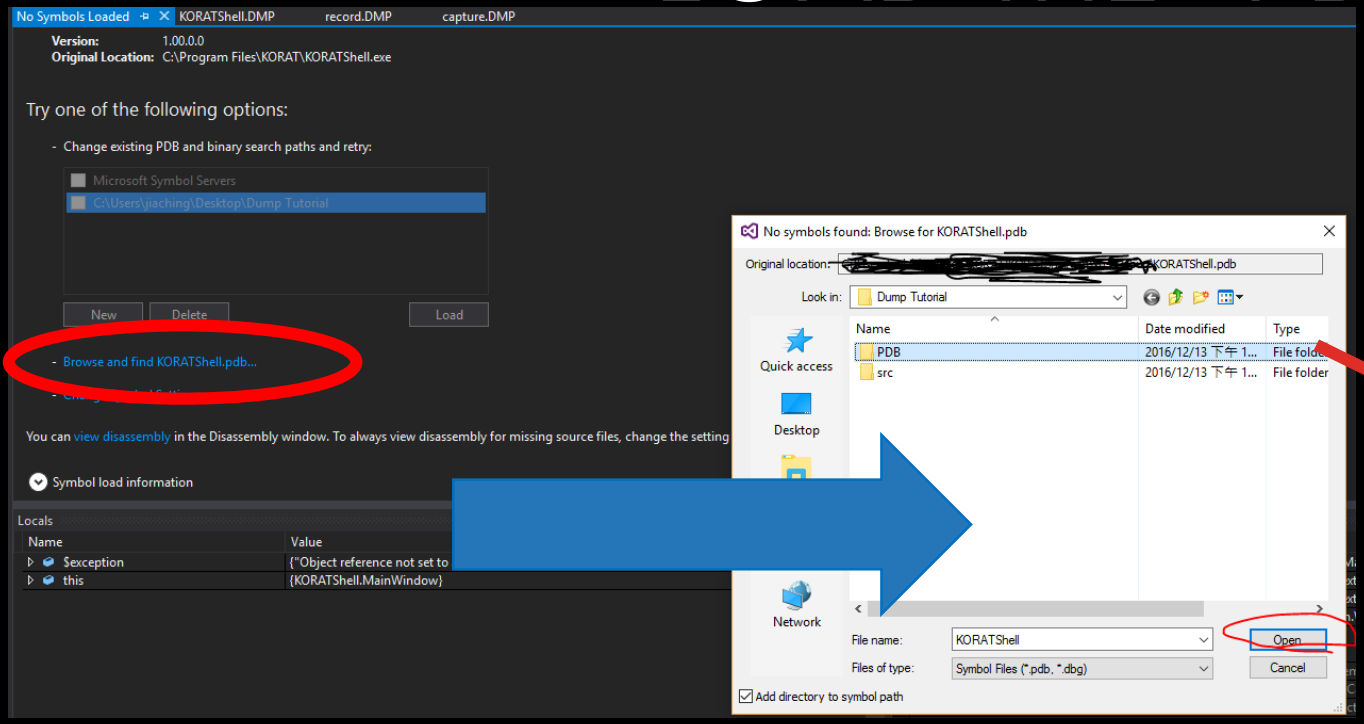
DOUBLE CLICK THIS LINE IN CALL STACK

“KORATShell.exe!KORATShell.MainWindow.ShowLocalRecordDialog() Line 45“

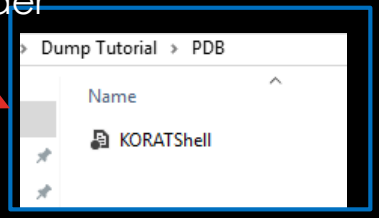


教育部顧問室
Ministry of Education
Advisory Office

LOAD THE "PDB" FILE

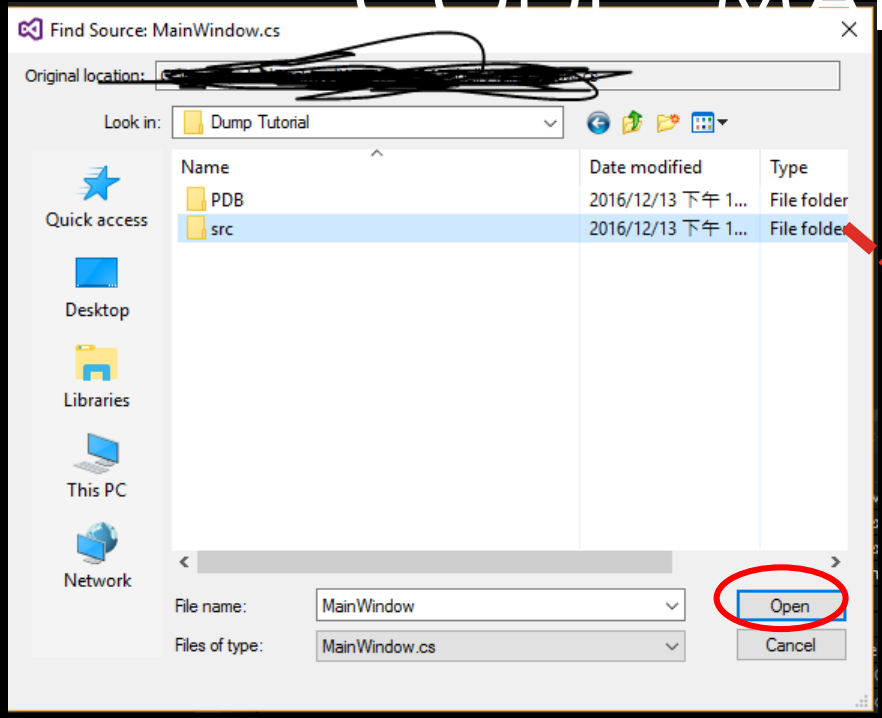


KORATShell.pdb in the PDB folder

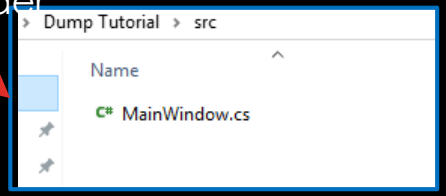


教育部顧問室
Ministry of Education
Advisory Office

AND THEN LOAD THE SOURCE CODE "MAINWINDOWS.CS"



MainWindows.cs in the src folder



教育部顧問室
Ministry of Education
Advisory Office

LET'S DEBUG !!!!!!!!!!!

```
            _localRecordDialog = new MetroForm();
            _localRecordDialog.Text = Constants.RECORD;
            _localRecordDialog.Controls.Add(toolbar);
            _localRecordDialog.Size = new System.Drawing.Size(500, 90);
            _localRecordDialog.Theme = MetroFramework.MetroThemeStyle.Dark;
            _localRecordDialog.TopMost = true;

            _shouldHookKey = true;
            _localRecordWindow = _localRecordDialog;
            hookId = SetHook(Proc);
            _localRecordDialog.ShowDialog();

            TooBar.Items.AddRange(GetRecordButtons());
            WindowState = FormWindowState.Normal;
        }
        finally
```

```
_localRecordDialog.TopMost = true;

_shouldHookKey = true;
_localRecordWindow = _localRecordDialog;
hookId = SetHook(Proc);
_localRecordDialog.ShowDialog();
```

Locals

Name	Value	Type
$exception	{"Object reference not set to an instance of an object."}	System.I
this	{KORATShell.MainWindow}	KORATSI
hookId	{14615141}	System.I
toolbar	{KORATShell.Controls.MetroToolStrip, Name: , Items: 22}	KORATSI

Call Stack

Name	Lang
[Managed to Native Transition]	
[Native to Managed Transition]	
[Managed to Native Transition]	
System.Windows.Forms.dll!System.Windows.Forms.Application.ComponentManager.System.Windows.Forms.UnsafeNativeMethods.IMsoComp	Unkr
System.Windows.Forms.dll!System.Windows.Forms.Application.ThreadContext.RunMessageLoopInner(int reason, System.Windows.Forms.Appl	Unkr
System.Windows.Forms.dll!System.Windows.Forms.Application.ThreadContext.RunMessageLoop(int reason, System.Windows.Forms.Applicatic	Unkr
System.Windows.Forms.dll!System.Windows.Forms.Form.ShowDialog(System.Windows.Forms.IWin32Window owner)	Unkr
KORATShell.exe!KORATShell.MainWindow.ShowLocalRecordDialog() Line 45	C#
KORATShell.exe!KORATShell.Triggers.PERecordTargetTrigger.Execute() Line 34	C#
KORATShell.exe!KORATShell.ShellForm.Trigger(TriggerTypes triggerType, System.EventArgs args) Line 14	C#
ModuleProjectExplorer.exe!ModuleProjectExplorer.ProjectExplorer.RecordTestCase(object sender, ModuleProjectExplorer.Events.TreeNodeRight(	Unkr
ModuleProjectExplorer.exe!ModuleProjectExplorer.ProjectExplorer.RecordSelectedTarget()	Unkr
KORATShell.exe!KORATShell.MainWindow.RecordStartBtn_Click(object sender, System.EventArgs e) Line 326	C#
System.Windows.Forms.dll!System.Windows.Forms.ToolStripItem.HandleClick(System.EventArgs e)	Unkr
System.Windows.Forms.dll!System.Windows.Forms.ToolStripItem.HandleMouseUp(System.Windows.Forms.MouseEventArgs e)	Unkr
System.Windows.Forms.dll!System.Windows.Forms.ToolStrip.OnMouseUp(System.Windows.Forms.MouseEventArgs mea)	Unkr
System.Windows.Forms.dll!System.Windows.Forms.Control.WmMouseUp(ref System.Windows.Forms.Message m, System.Windows.Forms.Mo	Unkr
System.Windows.Forms.dll!System.Windows.Forms.Control.WndProc(ref System.Windows.Forms.Message m)	Unkr
System.Windows.Forms.dll!System.Windows.Forms.ToolStrip.WndProc(ref System.Windows.Forms.Message m)	Unkr
KORATShell.exe!KORATShell.Controls.MetroToolStrip.WndProc(ref System.Windows.Forms.Message m) Line 25	C#
System.Windows.Forms.dll!System.Windows.Forms.NativeWindow.Callback(System.IntPtr hWnd, int msg, System.IntPtr wparam, System.IntPtr	Unkr

教育部顧問室
Ministry of Education Advisory Office

OBSERVATIONS

- When Window core system component throw an exception and nobody catch, your program just crash
  - How about C++
  - How about Java
- The core dump analysis shows an exception is raised in user32.dll
- The exception is raised due to an null reference exception
- Now at least we have a clue

FINALLY, WE REACH MSDN

```
public delegate void MyCallback();
[DllImport("MYDLL.DLL")]
public static extern void MyFunction(MyCallback callback);
```

Also, make sure the lifetime of the delegate instance covers the lifetime of the unmanaged code; otherwise, the delegate will not be available after it is garbage-collected.

See Also

C# Tutorials

GARBAGE COLLECTED? WHAT AN UNUSUAL BUG FINAL RESOLUTION

```
_shouldHookKey = true;
_localRecordWindow = _localRecordDialog;
HookProcedure = new LowLevelKeyboardProc(Proc);
hookId = SetHook(HookProcedure);
_localRecordDialog.ShowDialog();
```

REFERENCE

Using Dump Files

What Is Managed Code?

Difference between native and managed code?

Debug With Mixed vs Debug with Native Only when debugging dump files

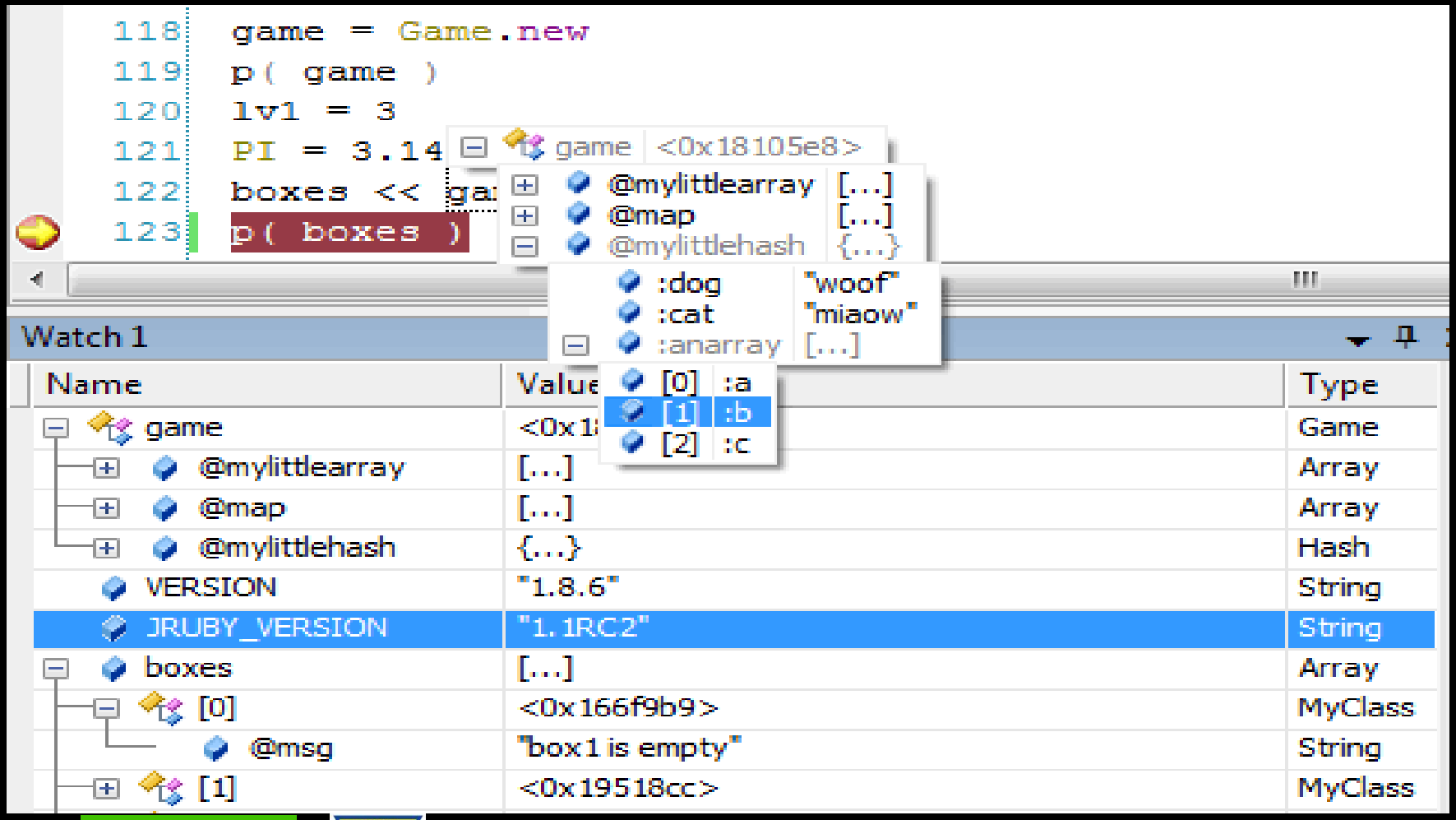
CONSOLE-MODE DEBUGGER SUCKS?

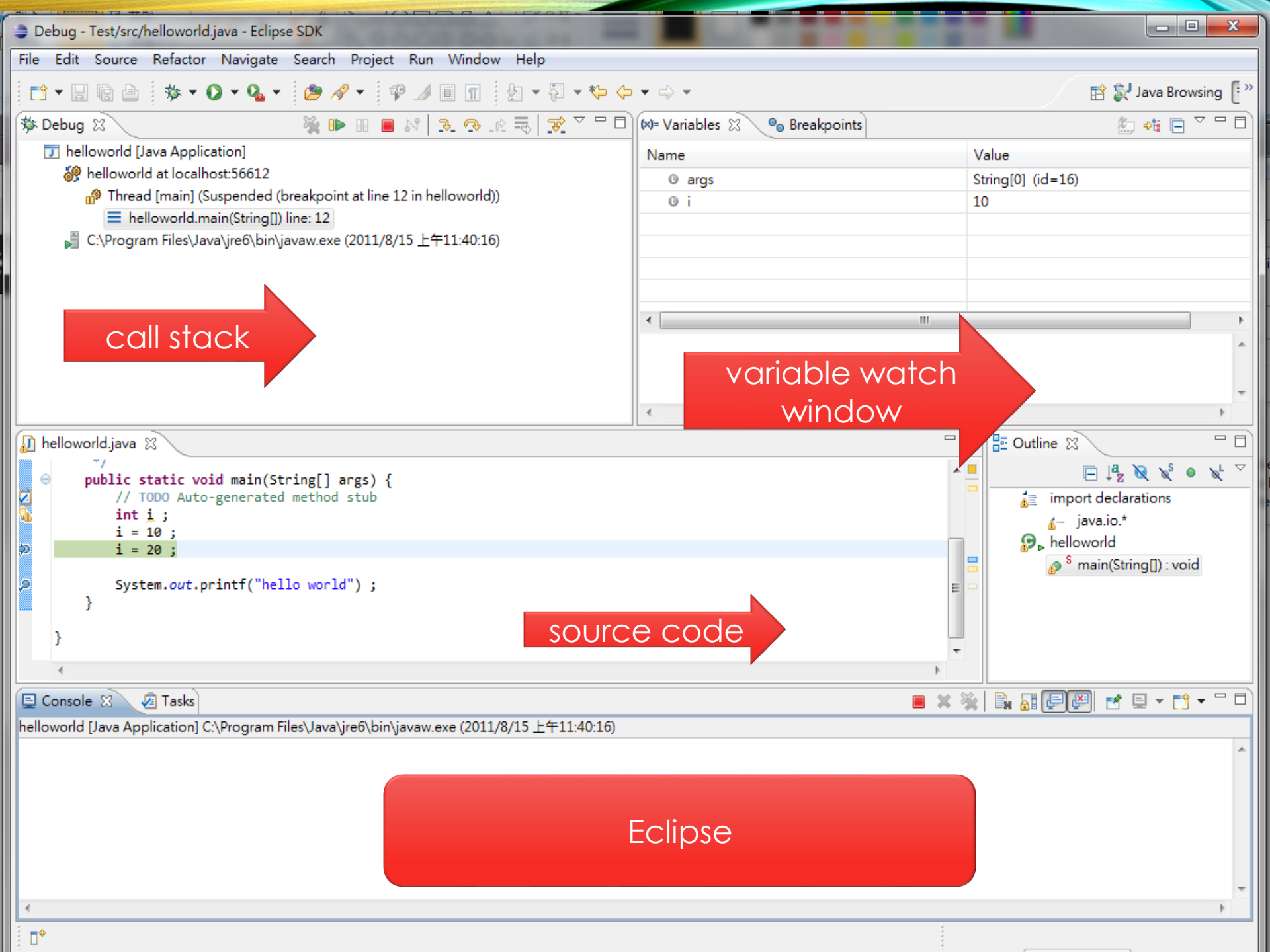
- Yes, 一般而言命令列模式的debugger 使用起來其實不夠方便(還要背指令)，對習慣視窗世代的學生而言，接受度很低。
- Yes, console mode debugger is inconvenient to use
- 不過，做為主修電腦科學的學生，了解debugger 是怎麼樣以最原始的形式存在還是很重要的。
- However, it is important for a CS major students to understand the original form of a debugger.
- 未來有許許多多非 win-tel 的開發平台。如果開發平台尚不支援Window GUI，則通常 debugger 的操作方式就會退回到 console mode 的使用介面

DEBUGGER WRAPPED BY GUI

- 為了解決 console mode debugger 不容易使用的問題，目前整合開發環境都會將debugger 與 compiler, editor 做整合，使得除錯過程能更友善
  to resolve the inconvenience of using console mode debugger, most IDEs nowadays integrate a debugger with editors in IDE.
- 有時候稱之為 Visual Debugger

DEBUGGING GUI – VISUAL STUDIO

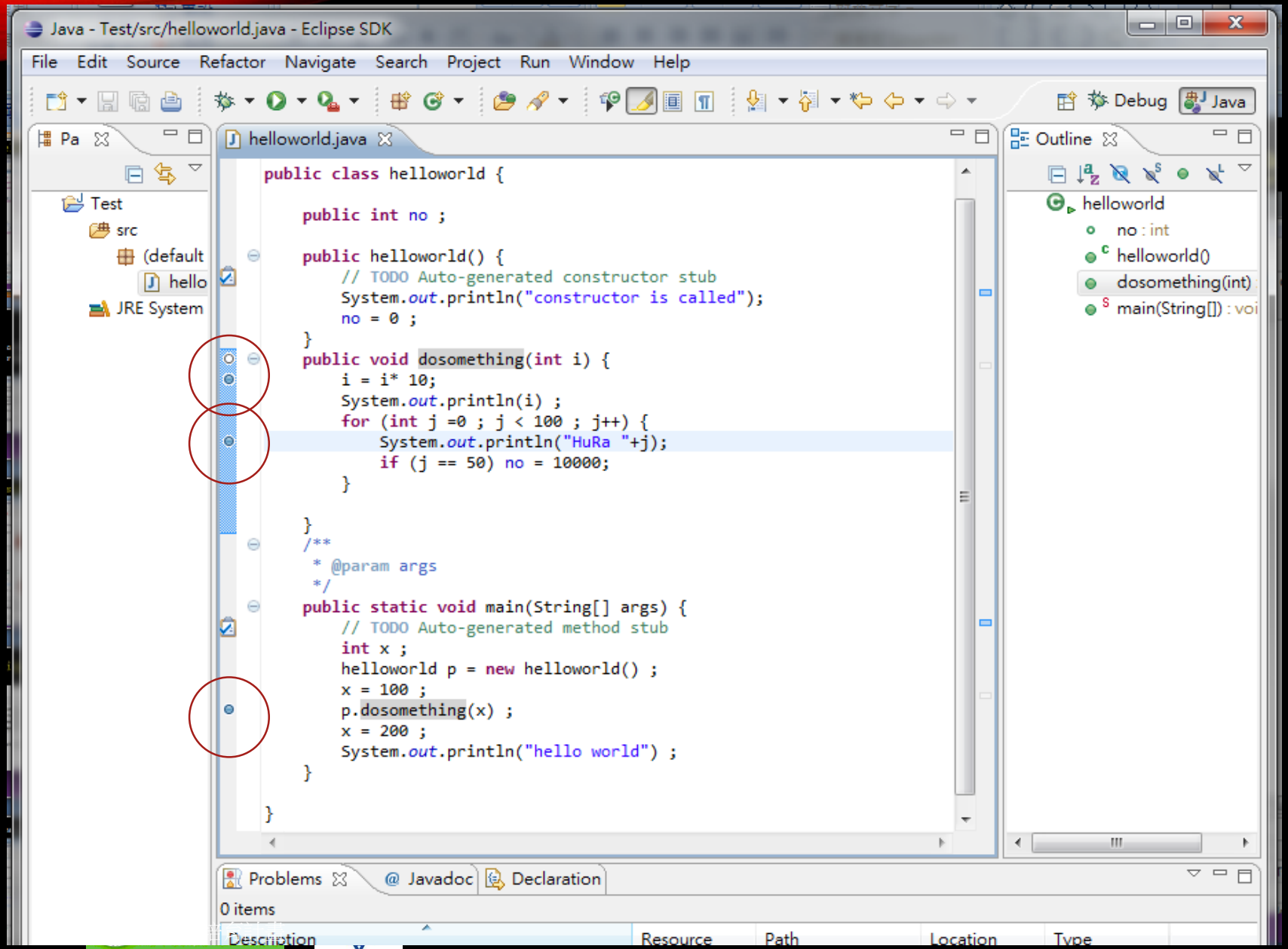



Debug - Test/src/helloworld.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java Browsing
Debug
helloworld [Java Application]
helloworld at localhost:56612
Thread [main] (Suspended (breakpoint at line 12 in helloworld))
helloworld.main(String[]) line: 12
C:\Program Files\Java\jre6\bin\javaw.exe (2011/8/15 上午11:40:16)
call stack
Variables
Breakpoints
Name
Value
args
String[0] (id=16)
i
10
variable watch window
helloworld.java
public static void main(String[] args) {
// TODO Auto-generated method stub
int i ;
i = 10 ;
i = 20 ;
System.out.printf("hello world") ;
}
}
source code
Outline
import declarations
java.io.*
helloworld
main(String[]) : void
Console
Tasks
helloworld [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (2011/8/15 上午11:40:16)
Eclipse

LAB 1 – LET'S PLAY WITH JAVA DEBUGGER UNDER ECLIPSE

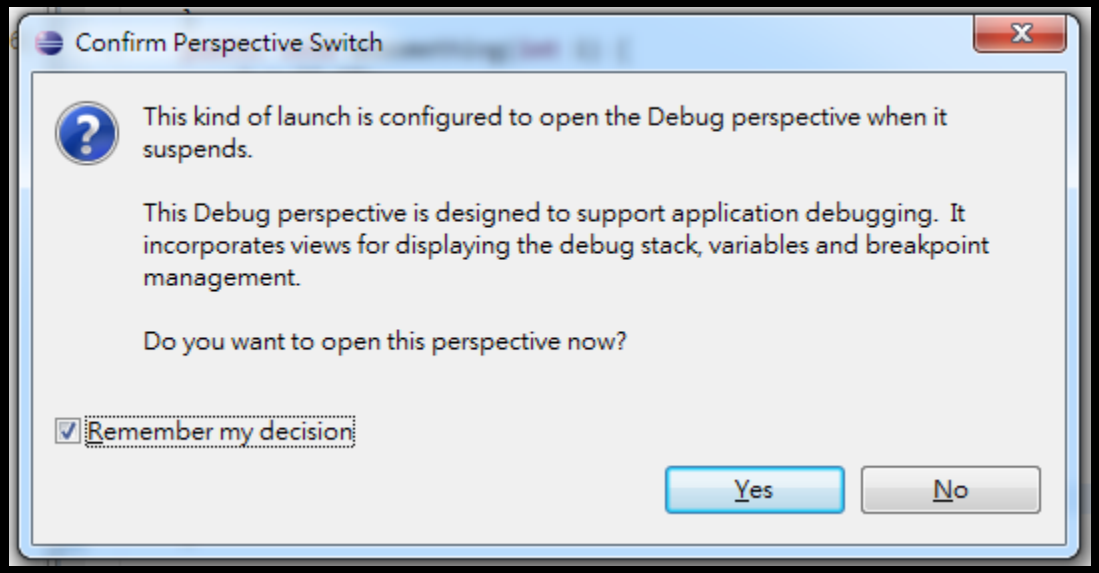
- 請開啟 Eclipse Classic 3.7
- New a project called test
- Enter a helloworld.java as follows

ENTER A HELLO WORLD EXAMPLE

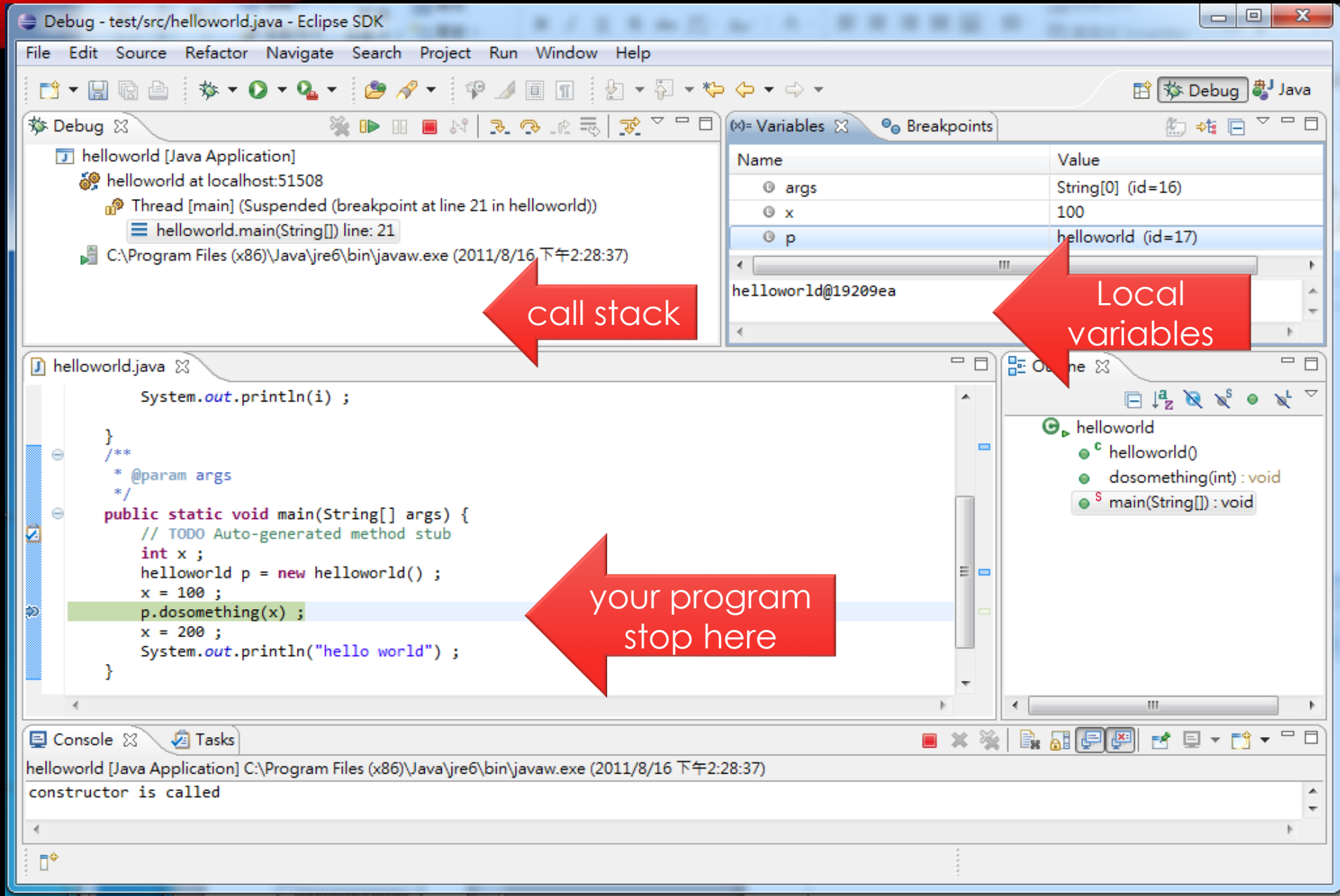


LET'S GO

- Set break points
- Debug the program (F11)



A DEBUG PERSPECTIVE



KEY BINDINGS

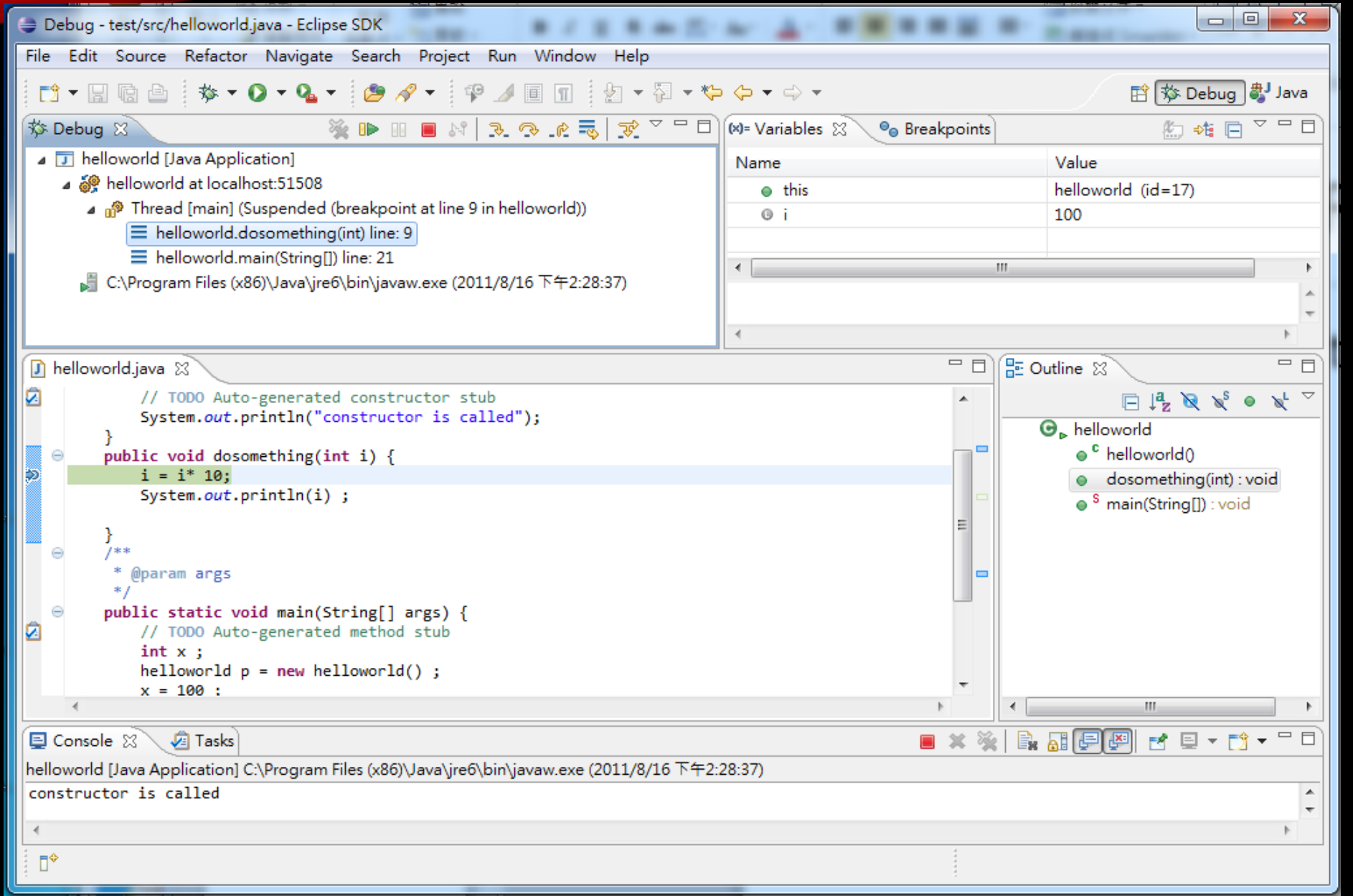
Table 1. Debugging Key bindings

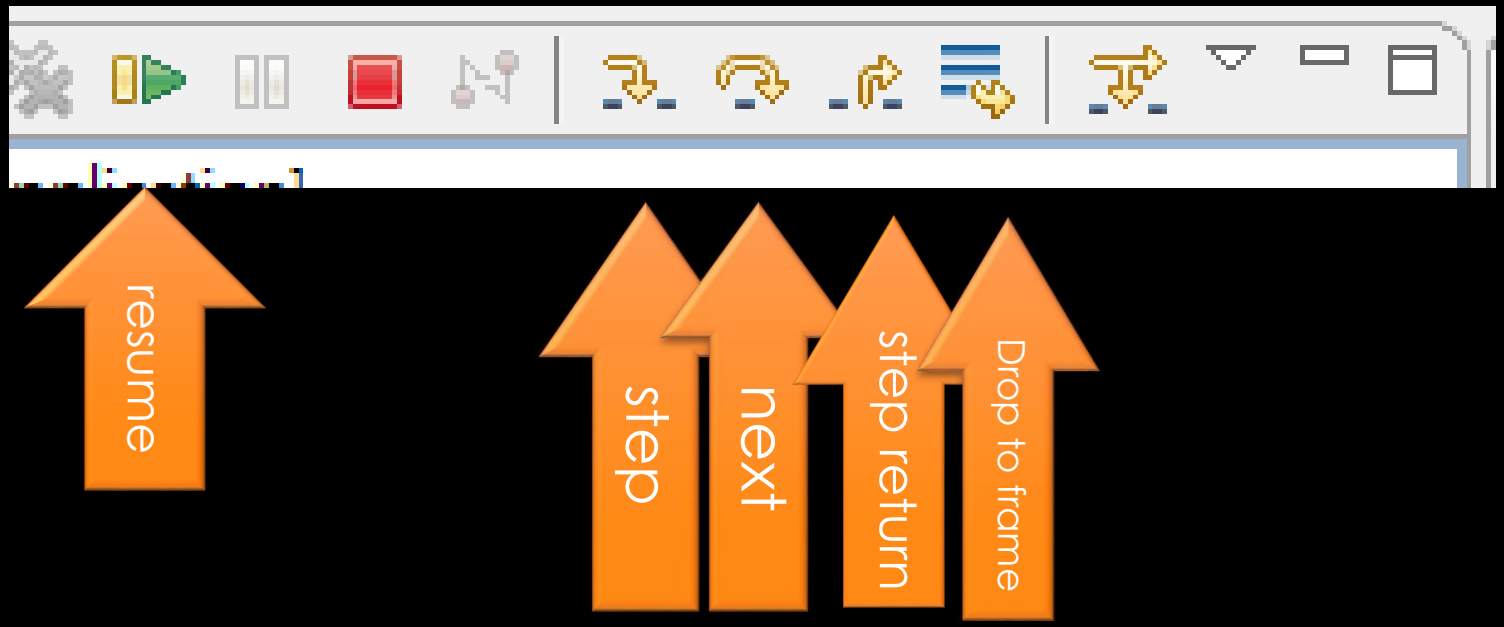
Command	Description
F5	Goes to the next step in your program. If the next step is a method / function this command will jump into the associated code.
F6	F6 will step over the call, e.g. it will call a method / function without entering the associated code.
F7	F7 will go to the caller of the method/ function. So this will leave the current code and go to the calling code.
F8	Use F8 to go to the next breakpoint. If no further breakpoint is encountered then the program will normally run.

You can of course use the ui to debug. The following displays the keybindings for the debug buttons.

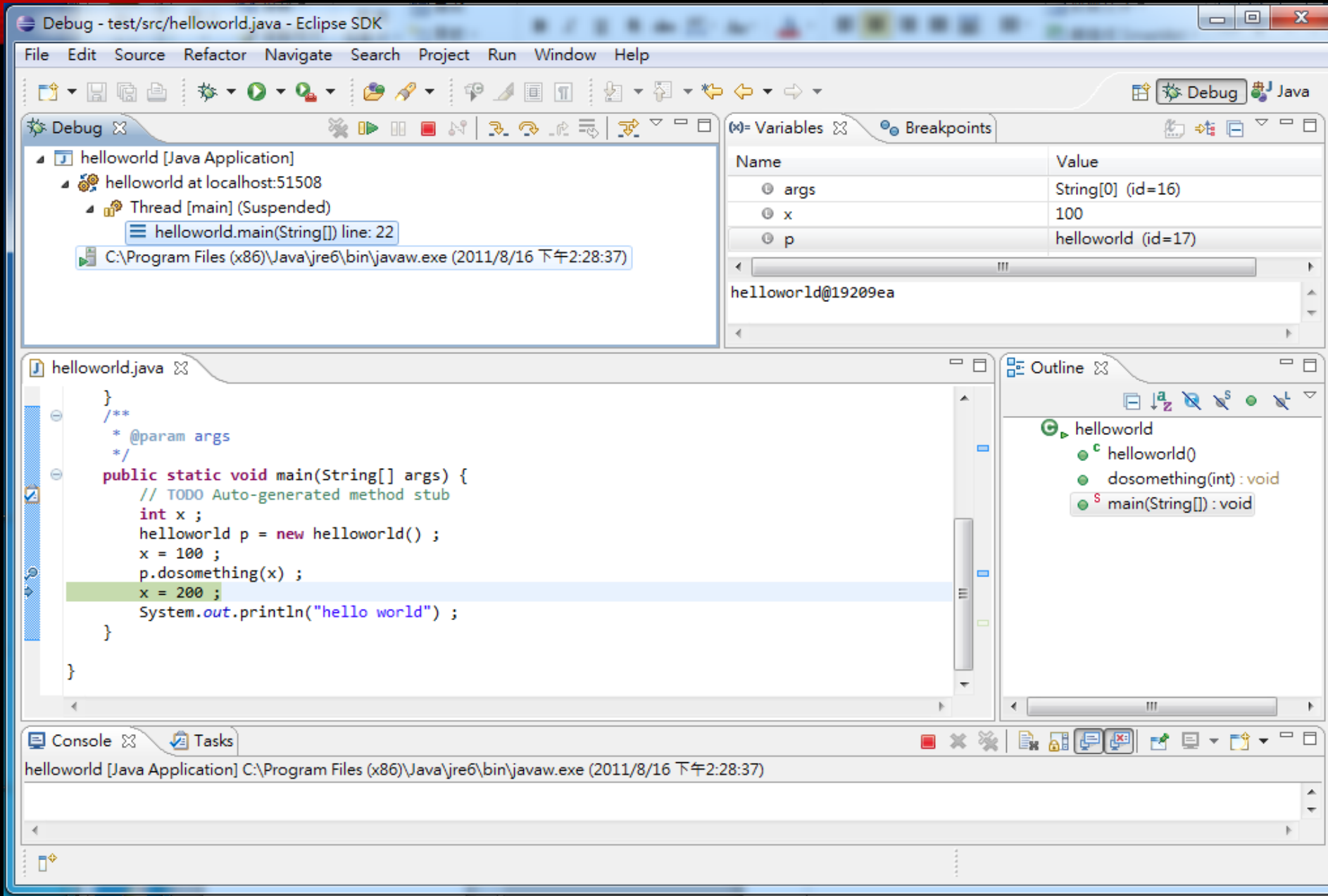
F8 Stop F5 F6 F7

- Click resume to hit the next break point

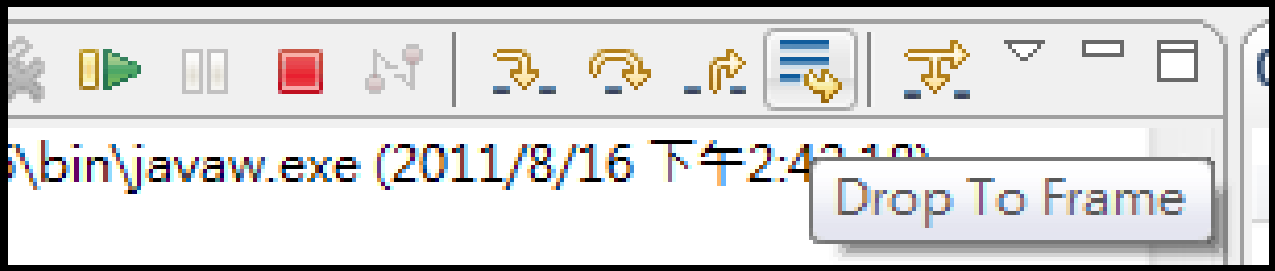



resume
step
next
step return
Drop to frame

STEP RETURN


Debug - test/src/helloworld.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Java
Debug
helloworld [Java Application]
helloworld at localhost:51508
Thread [main] (Suspended)
helloworld.main(String[]) line: 22
C:\Program Files (x86)\Java\jre6\bin\javaw.exe (2011/8/16 下午2:28:37)
Variables
Breakpoints
Name
Value
args
String[0] (id=16)
x
100
p
helloworld (id=17)
helloworld@19209ea
helloworld.java
} /** * @param args */ public static void main(String[] args) { // TODO Auto-generated method stub int x ; helloworld p = new helloworld() ; x = 100 ; p.dosomething(x) ; x = 200 ; System.out.println("hello world") ; } }
Outline
helloworld
helloworld()
dosomething(int) : void
main(String[]) : void
Console
Tasks
helloworld [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (2011/8/16 下午2:28:37)

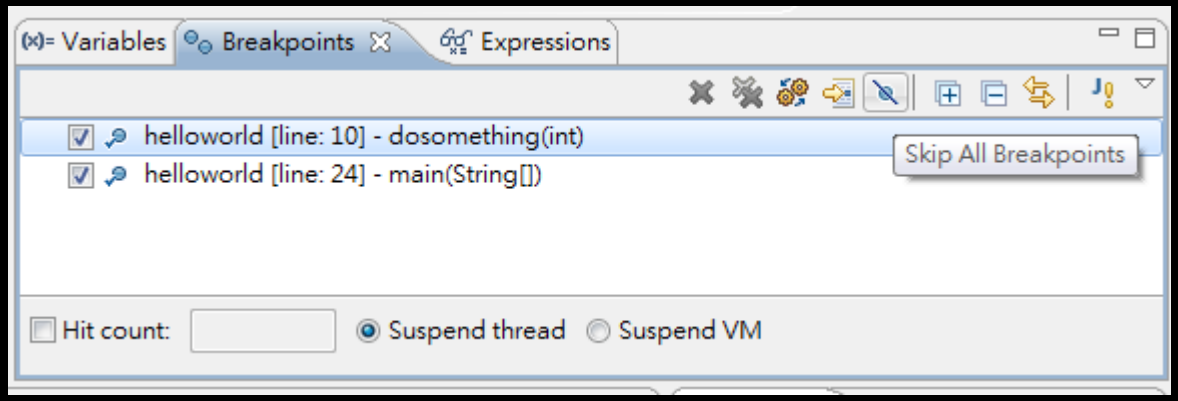
DROP TO FRAME

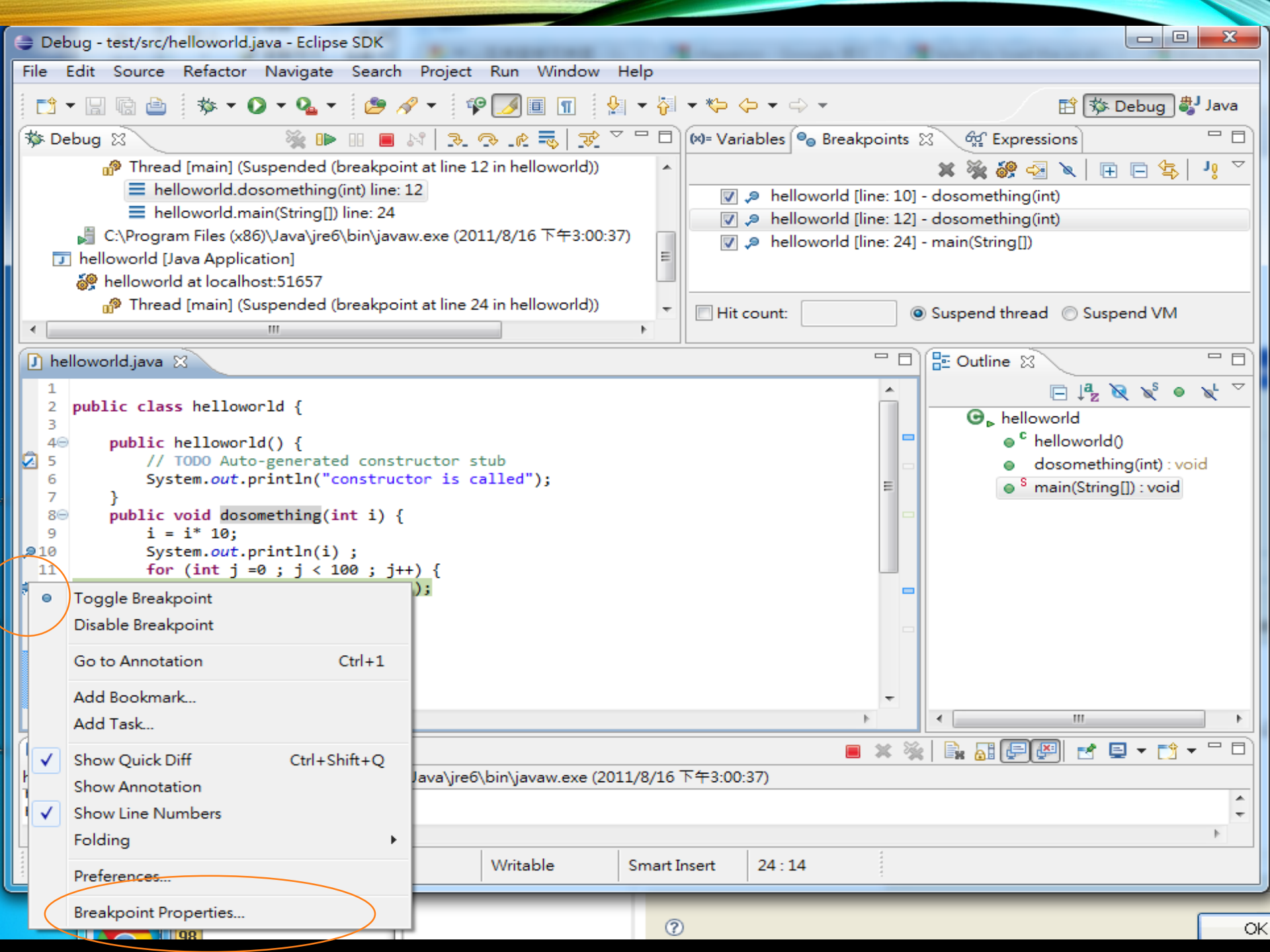


- Drop to frame
  - Full reverse debugger (too costly)
  - We often set a break point in the middle of a method. When the break point is hit, we may just find that some thing is wrong before this break point. Drop to the frame let you return to the start of method.
  - A kind of compromising reverse debugger

SKIP ALL BREAKPOINTS

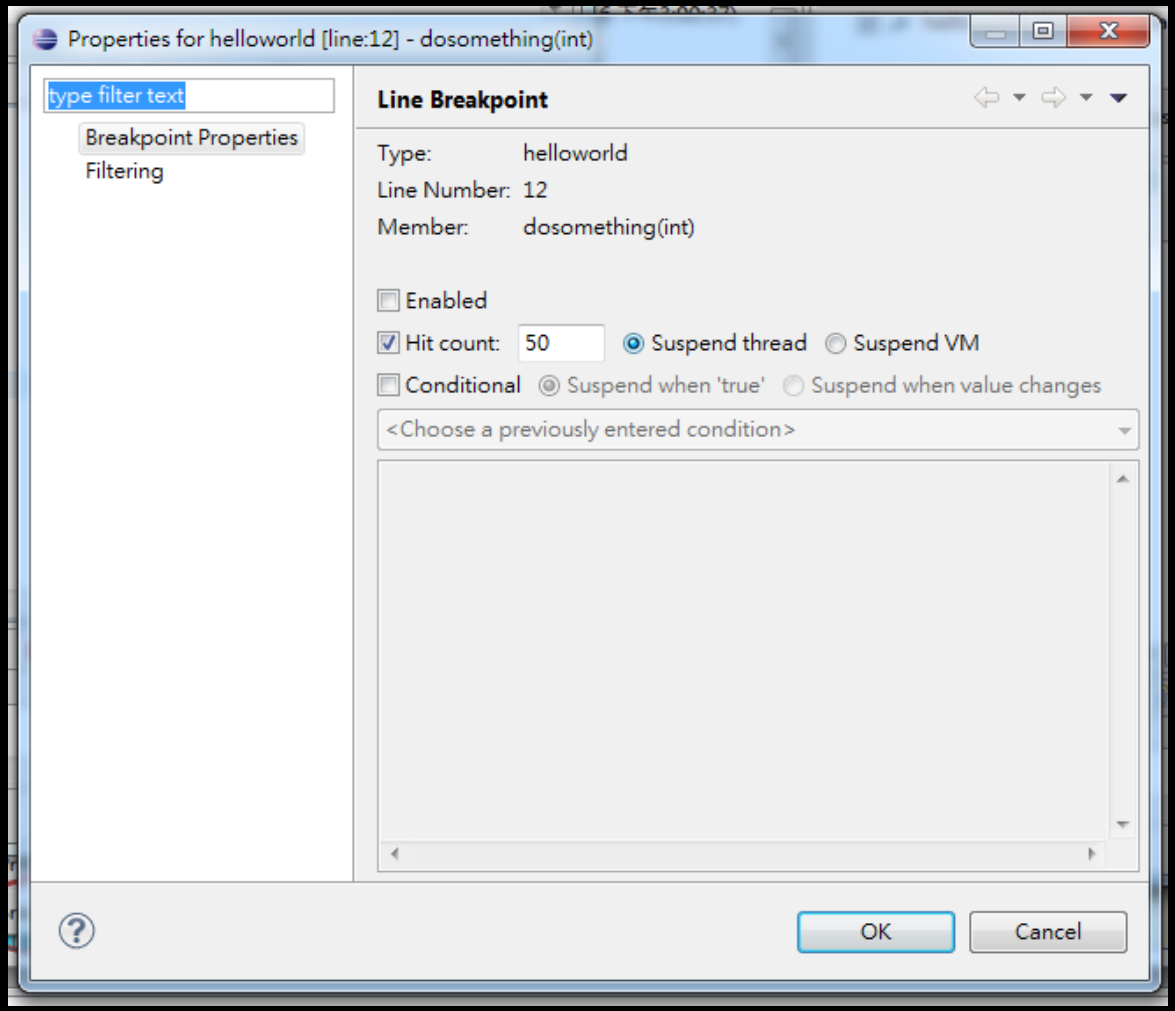
- Allow you to skip (disable) break points




Debug - test/src/helloworld.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Java
Debug
Thread [main] (Suspended (breakpoint at line 12 in helloworld))
helloworld.dosomething(int) line: 12
helloworld.main(String[]) line: 24
C:\Program Files (x86)\Java\jre6\bin\javaw.exe (2011/8/16 下午3:00:37)
helloworld [Java Application]
helloworld at localhost:51657
Thread [main] (Suspended (breakpoint at line 24 in helloworld))
Variables
Breakpoints
Expressions
helloworld [line: 10] - dosomething(int)
helloworld [line: 12] - dosomething(int)
helloworld [line: 24] - main(String[])
Hit count:
Suspend thread
Suspend VM
helloworld.java
1
2 public class helloworld {
3
4 public helloworld() {
5 // TODO Auto-generated constructor stub
6 System.out.println("constructor is called");
7 }
8 public void dosomething(int i) {
9 i = i* 10;
10 System.out.println(i) ;
11 for (int j =0 ; j < 100 ; j++) {
Toggle Breakpoint
Disable Breakpoint
Go to Annotation Ctrl+1
Add Bookmark...
Add Task...
Show Quick Diff Ctrl+Shift+Q
Show Annotation
Show Line Numbers
Folding
Preferences...
Breakpoint Properties...
Outline
helloworld
helloworld()
dosomething(int) : void
main(String[]) : void
Java\jre6\bin\javaw.exe (2011/8/16 下午3:00:37)
Writable
Smart Insert
24 : 14
OK

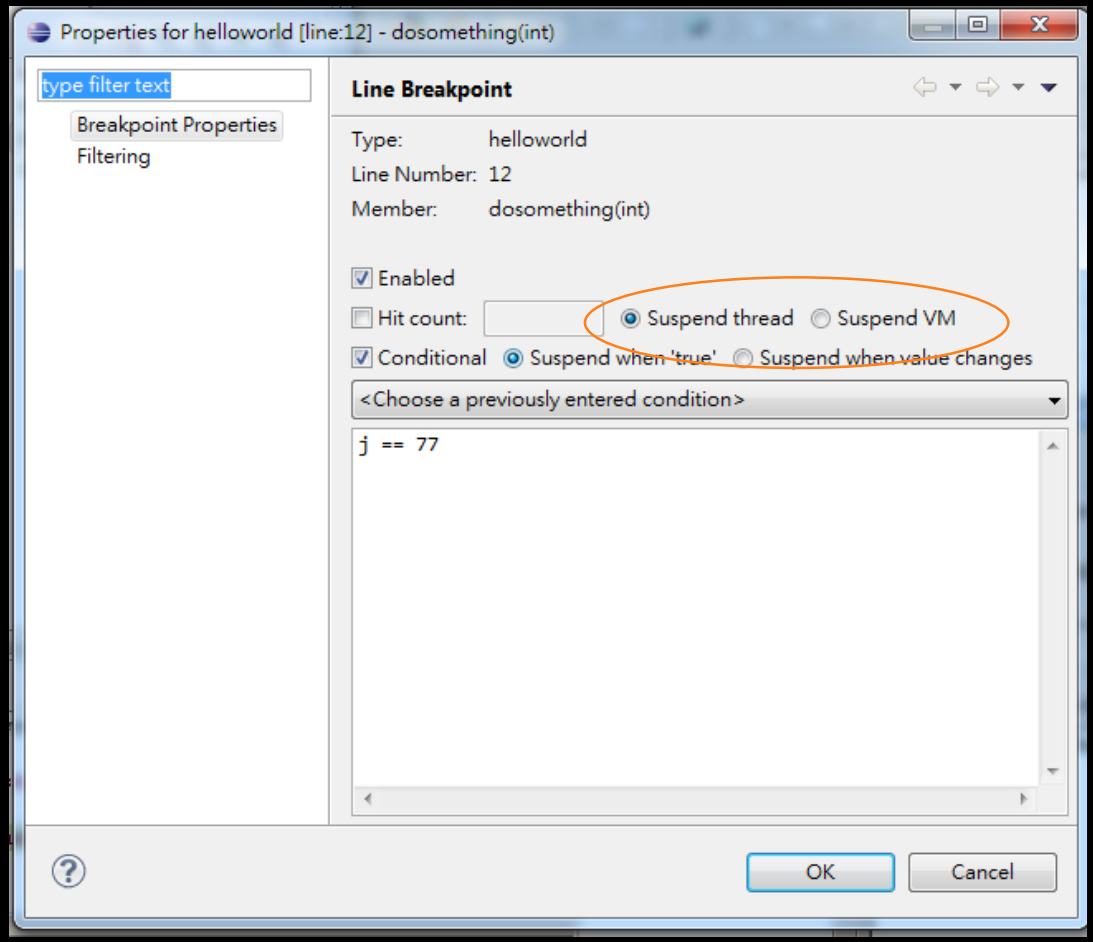
HIT COUNT

- Set the hit count to 50
- Resume the program
- Check the value of j in the for loop



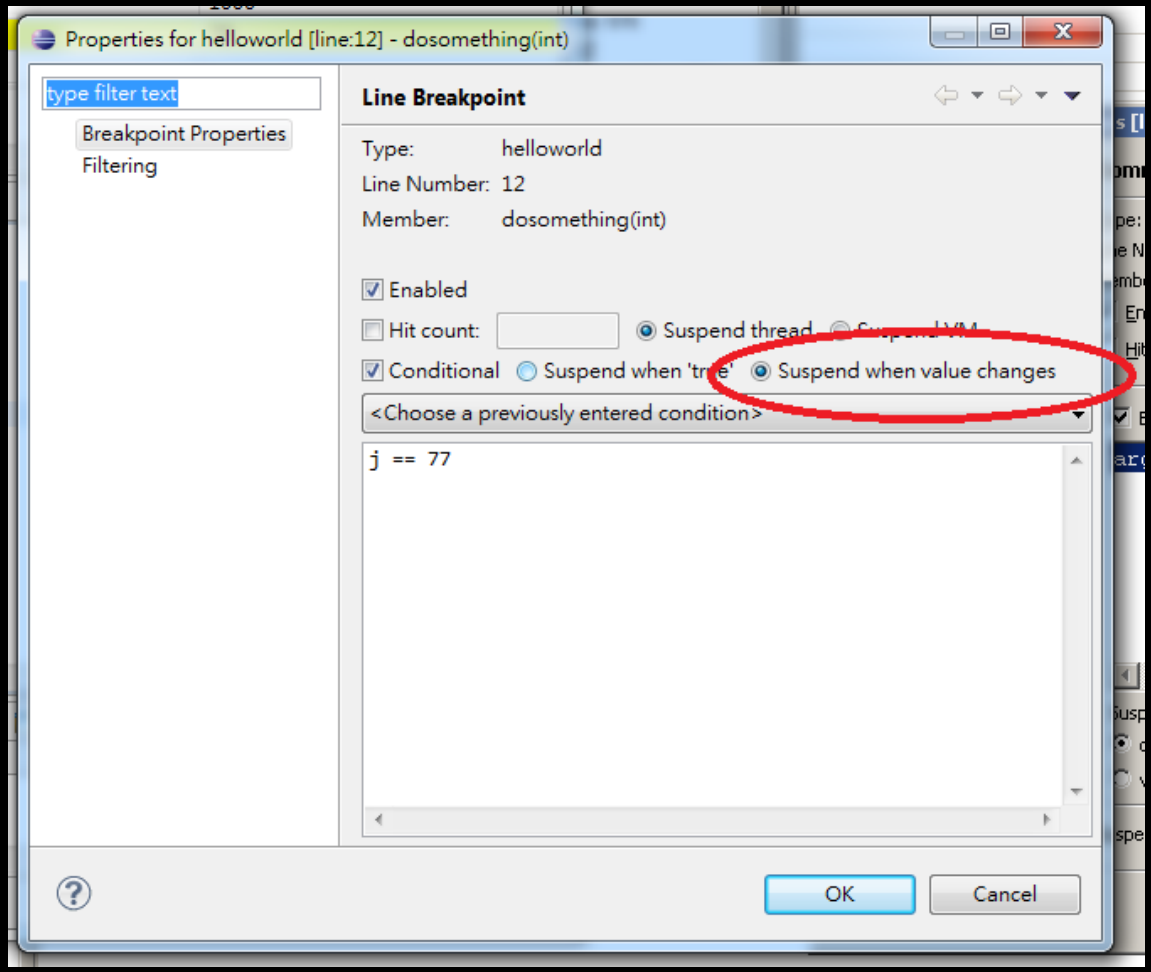
CONDITIONAL BREAK POINT

- Enter a Boolean expression
- The breakpoint will be hit when the condition is true



CONDITIONAL BREAK POINT

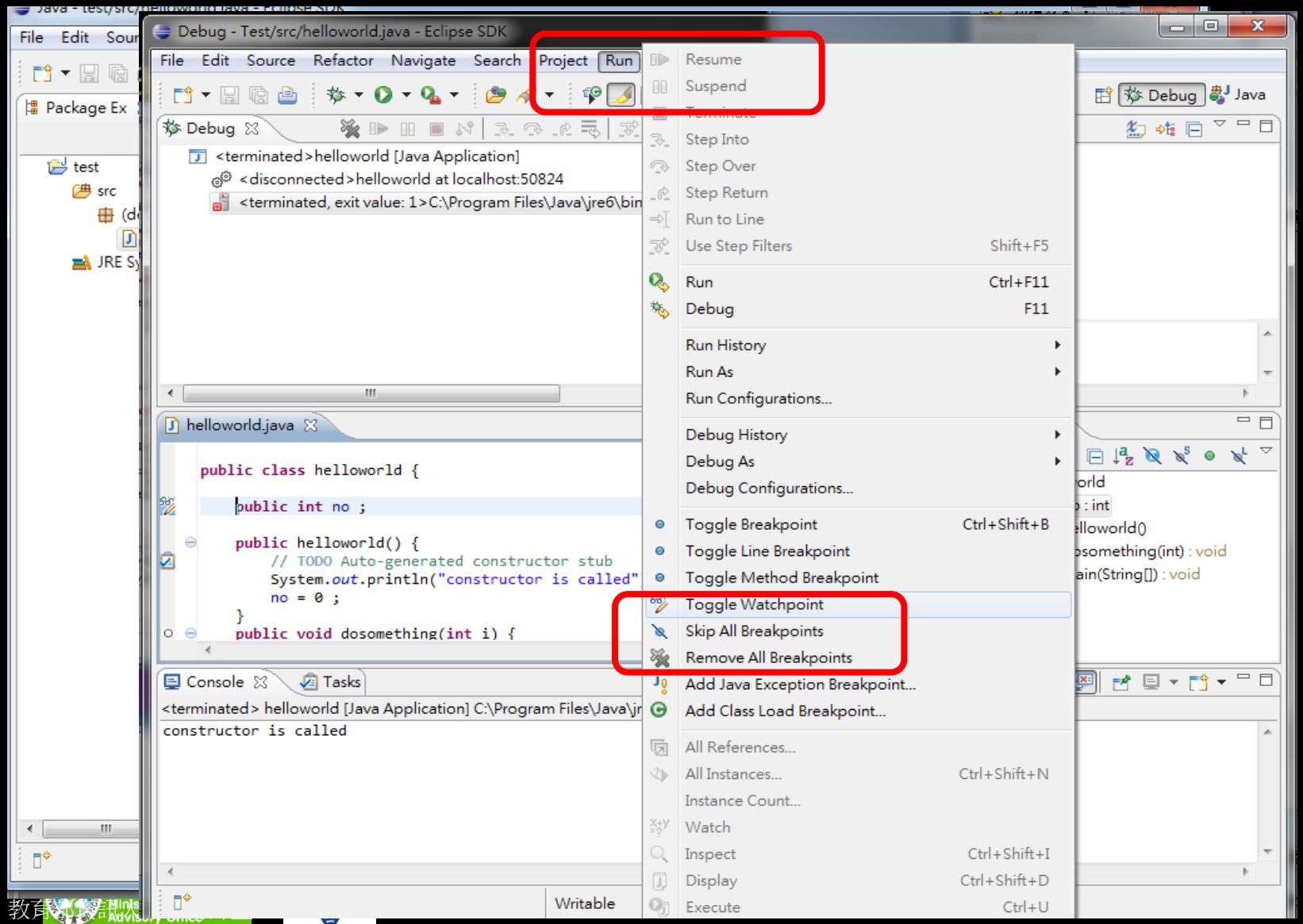
- A break point is hit when the condition evaluation changes.
- So, the break point will be hit at j=0,77,78



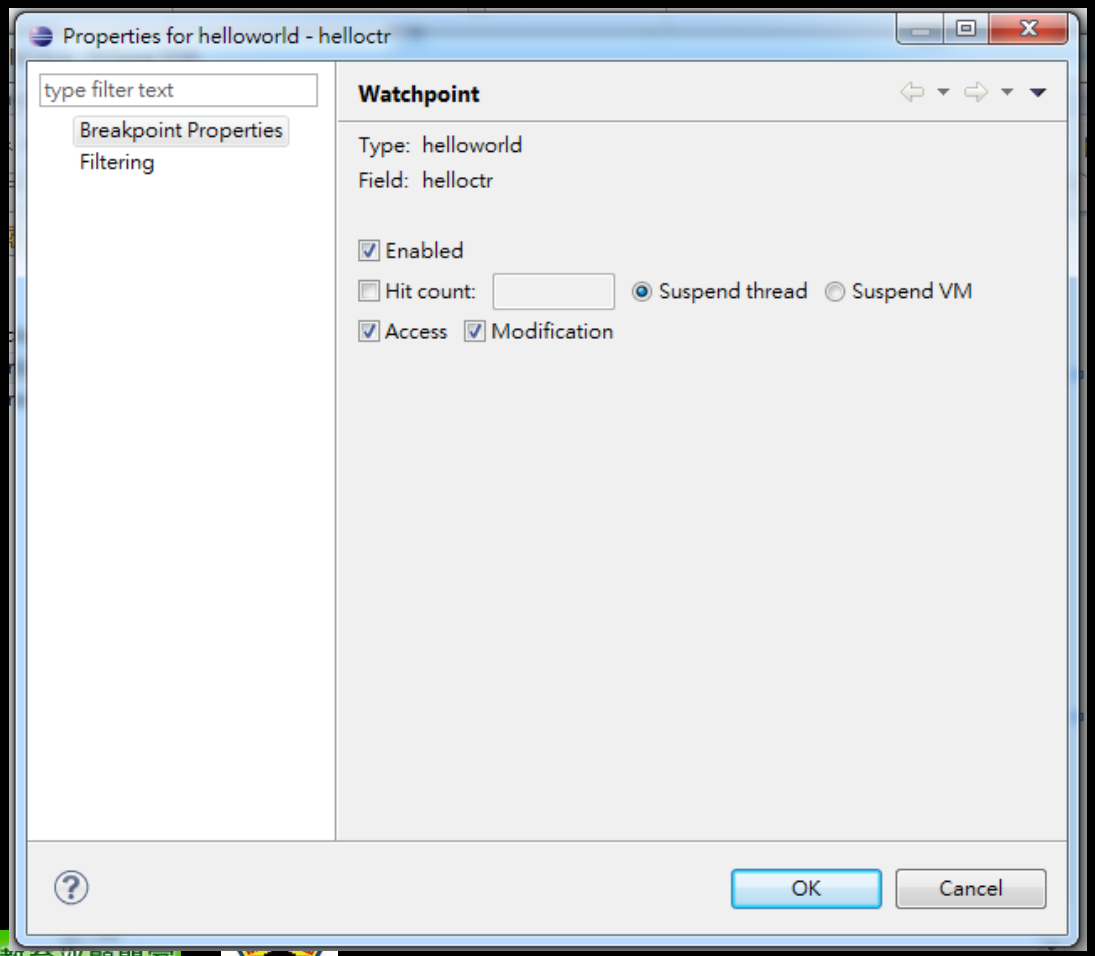
WATCH POINT

- Sometimes, computer programs can behave obscurely and weirdly. (e.g., using pointer, polymorphic, inheritance)
- So sometimes you may want to stop a program when a variable is accessed/changed.
- It can be particularly important if your system scale is large and it is implemented by a team, where modularity may not be clean and fully-decoupled.
- You may want to know **whose code** mess up your data structures.

PLACE A WATCH POINT ON A FIELD



THE OPTIONS OF A WATCH POINT



ATTACH A RUNNING PROCESS TO DEBUGGER

- A very common feature when you have mult-processes to debug
  - E.g., your program when started, fork a child process to do something else.
- Just google "attach a process to a debugger"

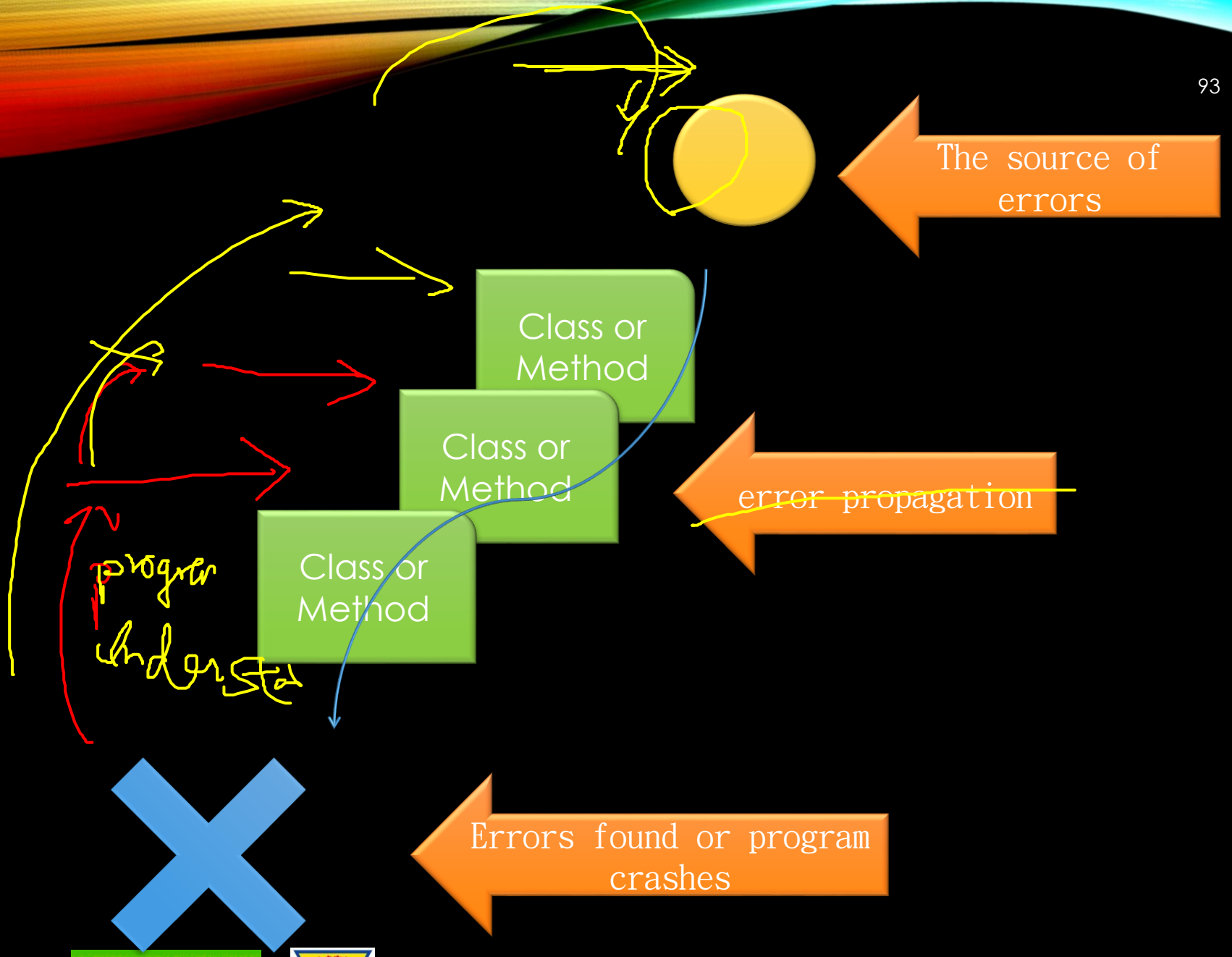
WHERE THE BUGS ARE HIDDEN?

- OK, talk is cheap.
- Even you become a master of debugging tools, unfortunately debugging requires a lot of
  - Experiences -- (so that you can come out with better hypothesis)
  - Hypothesis – evaluate the symptoms so that you can guess where could go wrong
  - Set break points
- Poor hypothesis -> useless break points -> waste of time-> make other guesses
- It can get much more complicated if your system is
  - Large and complicated
  - Built by a lot of people in different teams.
  - Your system is not well designed or modularized.

THE STORY OF GOD OF CODING

教育部資訊人才培育計畫

NEWBIE, SENIOR, AND GURU


The source of errors
Class or Method
Class or Method
Class or Method
error propagation
Errors found or program crashes

SOME COMMON DEBUGGING APPROACHES

- Read the source code and find the bugs
  - Stupid and inefficient
  - Sometimes works because the bug is so trivial
- Step by Step check if each step has problems
  - Using steps in a debugger and check variables in each break point
  - Doable but time consuming
- Divide and Conquer
  - Check middle point results (set break points)
  - If the results are correct, the chance that the bug is hidden in the second half of the program is much higher
  - Keep narrow the search by setting break points
- Simplify the test case so that the error can still be reproduced.

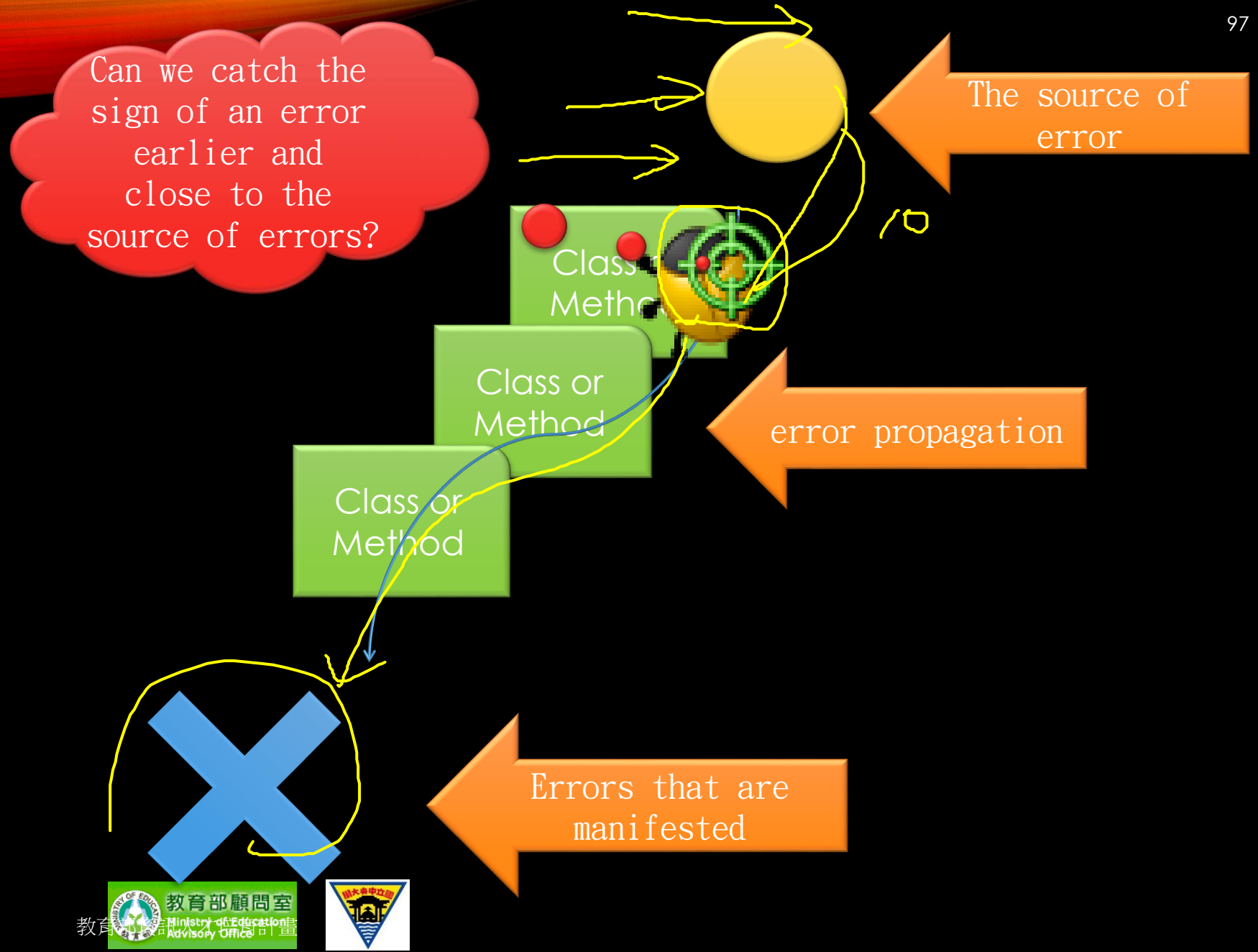
THE RIGHT SOLUTION

- Mastering debugging tools is a must for a good developer.
- However, even you can debug quickly
  - The question is can you really debug every kinds of bugs in short time
- If you spend a lot of time in locating the source of errors, it could mean
  - Your coding quality is poor
  - Your system modularity could be poor so that bugs can be hidden in your code.
  -

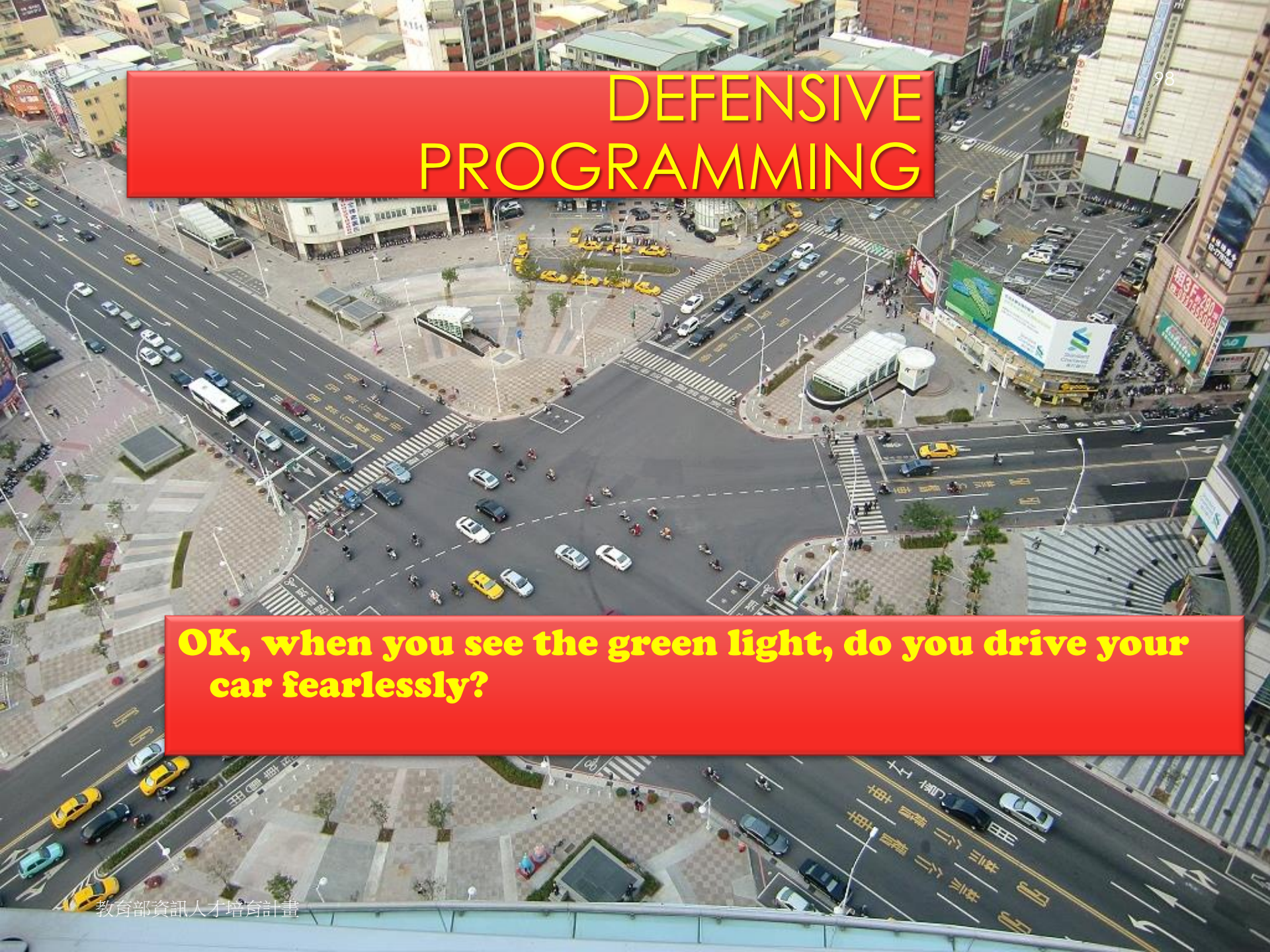
BEST DEBUGGING STRATEGY?

- Let the bug be manifested by themselves.
- But HOW?

Can you see the bug?


Can we catch the sign of an error earlier and close to the source of errors?
The source of error
Class or Method
Class or Method
Class or Method
error propagation
Errors that are manifested

DEFENSIVE PROGRAMMING



OK, when you see the green light, do you drive your car fearlessly?

WHY WE NEED DEFENSIVE PROGRAMMING?

- In school, mostly you complete your programming work alone.
- In real world, you work with others to complete a product
- The stupidest phenomenon
  - You join all your source codes
  - Build
  - Run
  - Crash
  - Stare at each other
  - Accuse each other
  - Push one debugging leader
  - A great amount of time is spent until who is to blame is settled.


I'M WITH
STUPID

When a ship encounters disaster, how their ship body design prepare to deal with it?

TACTIC NO. 1 - ASSERTION

- An assertion is code that's used during development – typically exists through alpha testing and beta testing and removed in release
- It is instrumented into your code, which is called instrumentation
- Assertions are especially useful in large, complicated programs and in high reliability programs

ASSERTION IN C

fail fast

- In standard C, you can use #include <assert.h>

```
void foo(int x) {
    assert(x > 10) ;
}
```

- You can also define your own ASSERT

```
#define ASSERT(condition, message) { \
  if (!(condition)) {   \
      LogError("Assertion Failed:", \
                  #condition, message) ; \
      exit(EXIT_FAILURE);\
   } \
}
```

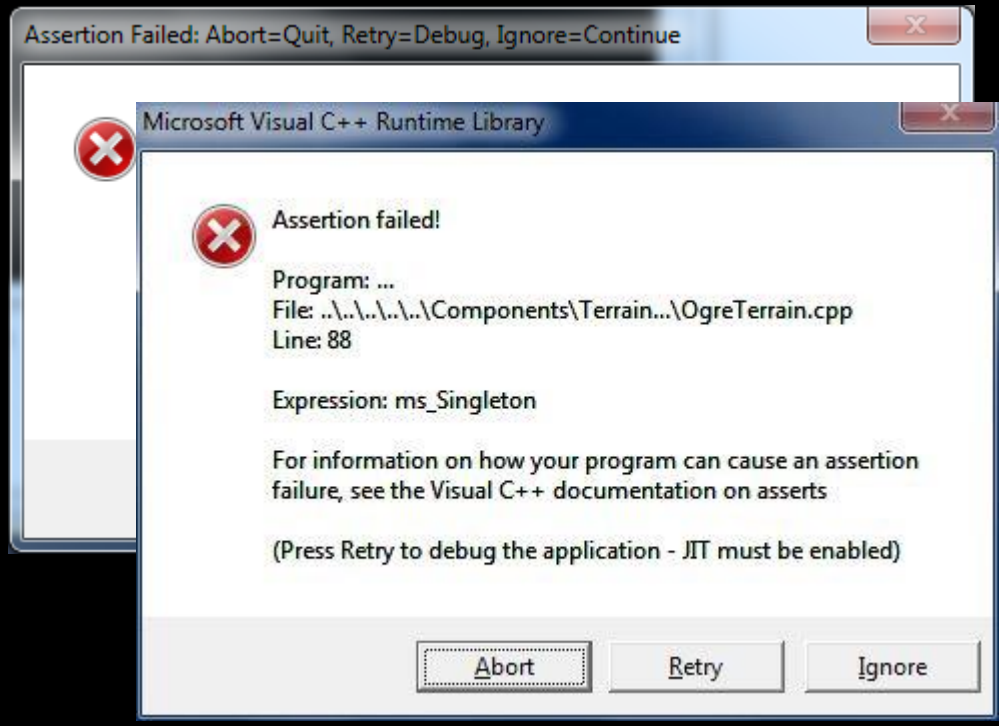
THE BODY OF ASSERT IN C

```
#ifdef NDEBUG
#define assert(_Expression)     ((void)0)
#else
#ifdef  __cplusplus
extern "C" {
#endif
_CRTIMP void __cdecl _wassert(_In_z_ const wchar_t * _Message, _In_z_ const
    wchar_t *_File, _In_ unsigned _Line);
#ifdef  __cplusplus
}
#endif
#define assert(_Expression) (void)( (!!(_Expression)) ||
    (_wassert(_CRT_WIDE(#_Expression), _CRT_WIDE(__FILE__), __LINE__), 0) )
#endif  /* NDEBUG */
```

NDEBUG is defined in release build

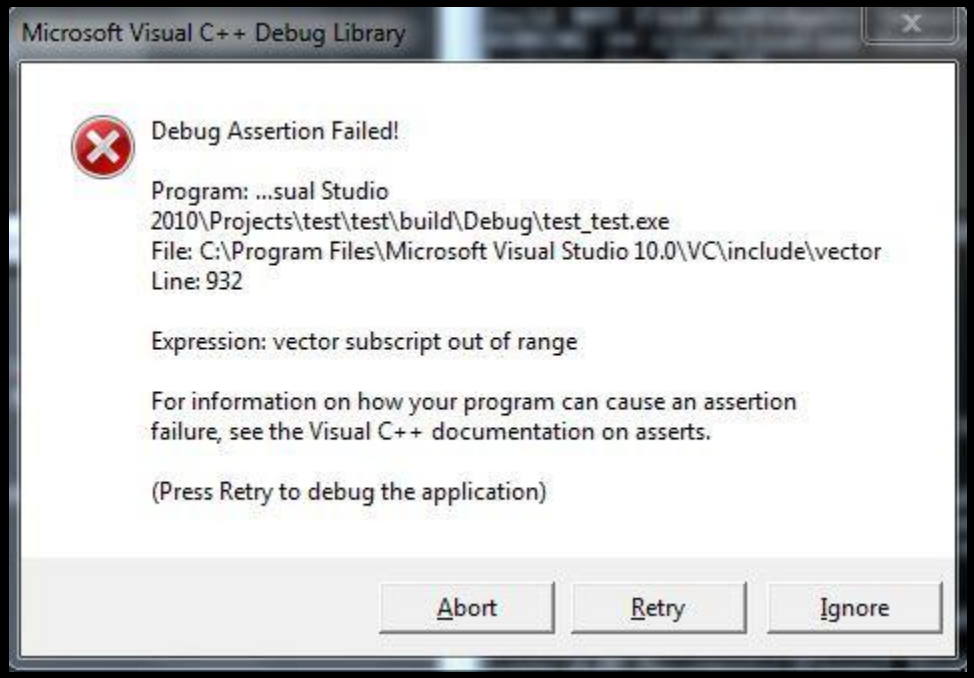
ABOUT ASSERT

- In nowadays IDEs, when your program hit an assert failed, IDEs ask you if you want to stop/retry your executions.
- At this time, you can invoke debugger to debug, check variables before assertion, and use call stack to examine how the assertion is hit

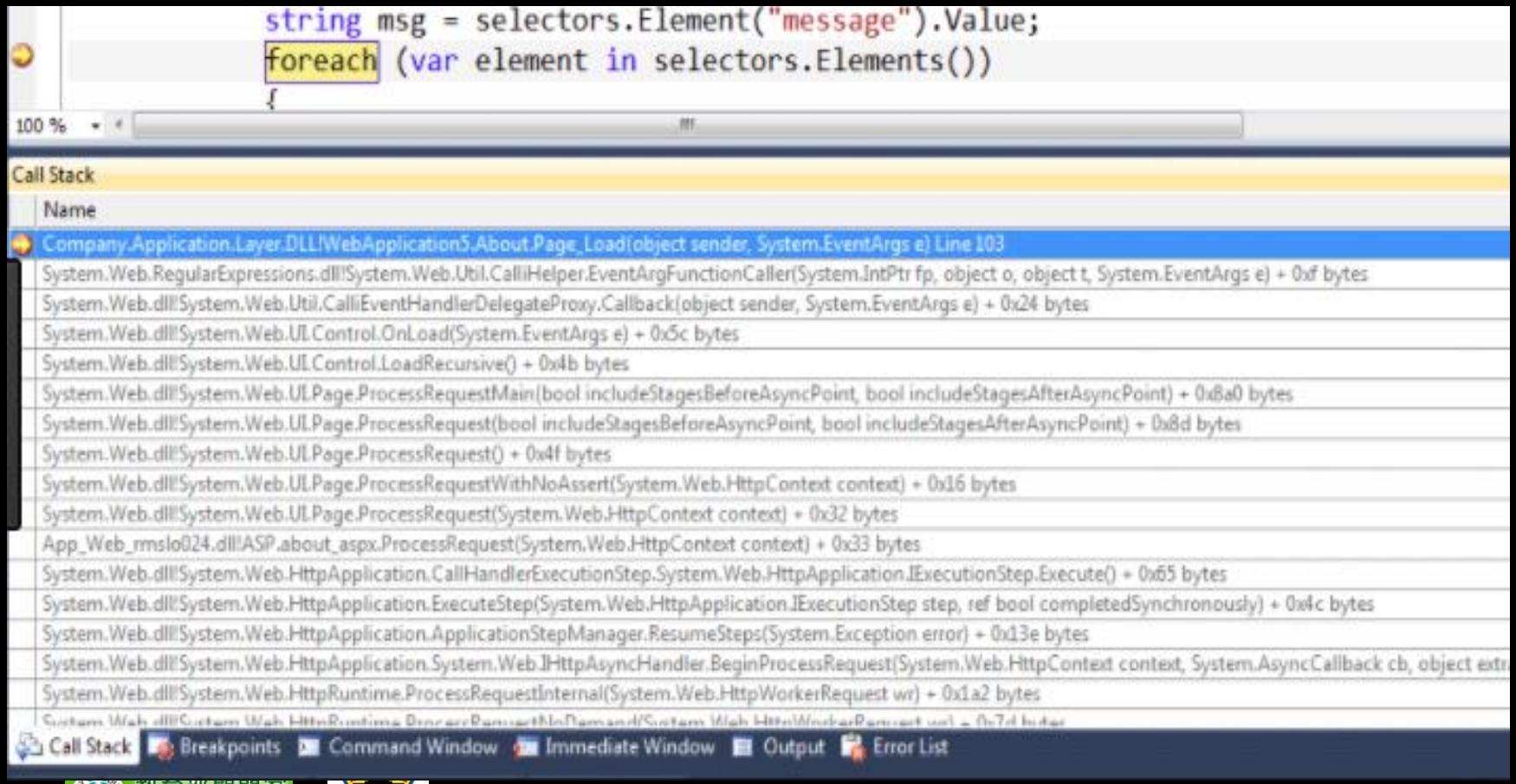


C++ STL ASSERTION

- When you use C++ STL inappropriately, your program will hit an assertion failed.
- So at this time, you can invoke the debugger to check where in your code use the STL incorrectly.



CALL STACK TRACE



ASSERTION EXAMPLES

- In Java
  - assert denominator != 0 : “denominator is unexpectedly equal to 0.”
- Example

```
.....
assert allcost !=0 : “assert failed: in foo(), allcost is
unexpectedly equal to 0.” ;
 averagecost = this.salary/ allcost ;
```

WHAT TO ASSERT ? SOME EXAMPLES

- That an input parameter's value falls within its expected range (or an output parameter's value does)

```
void p(int dir, int velocity) {
        assert(dir >=0 && dir < 4 ) ;
        assert(velocity >= 0 && velocity < 10);
        ……..
    }
```

- That a file or stream is open (or closed) when a routine begins executing (or when it ends executing)

```
void p(FILE *fp, string &msg) {
     assert(fp != null) ;
     assert(msg.getLength() >= 1 && msg.getLength() <= 255);
   }
```

- That a file or stream is at the beginning (or end) when a routine begins executing (or when it ends executing)
- That a file or stream is open for read-only, write-only, or both read and write~ I . That the value of an input-only variable is not changed by a routine
- That a pointer is non-null
- That an array or other container passed into a routine can contain at least XI number of data element
- That a table has been initialized to contain real values

```
void p(Object table[], int no) {
    assert(no != 0) ;
  for (int i=0;i< no; i++) {
      assert(table[i] != null) ;
      assert(table[i].getval() >= -1 && table[i].getval() <= 1) ;
  }
```

- That a container is empty (or full) when a routine begins executing (or when it finishes)
- That the results from a highly optimized, complicated routine match the results from a slower but clearly written routine 1

ASSERT AS A SAFETY GUARD FOR THE WORK IN THE MIDDLE

- When you left some part of the code to be dealt later or refined later.

```
If  (something) {
      …… // do some normal things
 } else {  // right now, it should not happen
           // you are not sure how this can
           // be reached.
     assert(false); // “under construction”
}
```

DO NOT CONFUSE ASSERT WITH EXCEPTION HANDLING

- Assertion are used to handle errors that should never occurs in the code
- Error handling (Exception Handling) is handing error you expect to occur. Such as file open errors
- Assertion should be disabled in official release
- Error handling is a part of system code
- Assertion is inserted as debugging aid.

ADVANCED TECHNIQUES TO STOP ERROR PROPAGATION

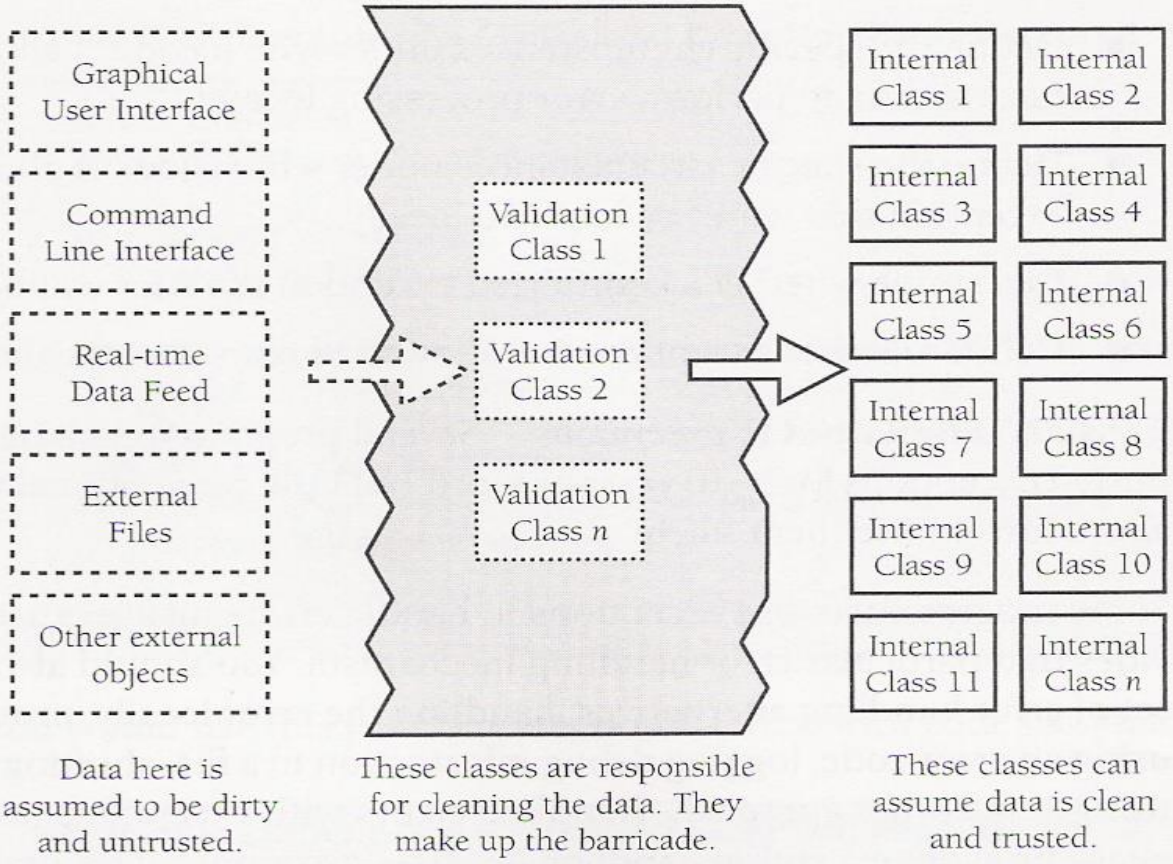
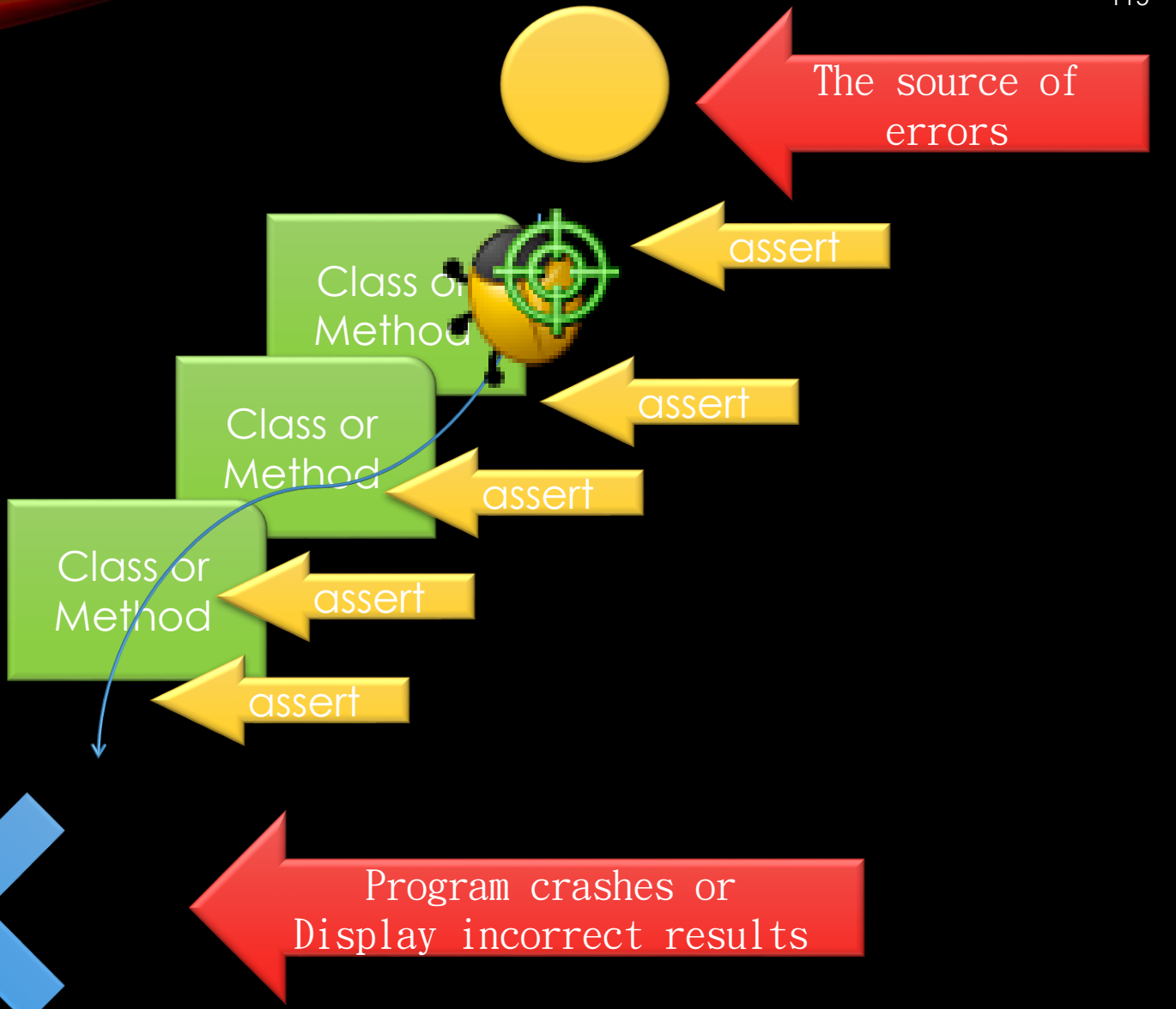


Figure 8-2 Defining some parts of the software that work with dirty data and some that work with clean data can be an effective way to relieve the majority of the code of the responsibility for checking for bad data.

GOOD CODING HABIT?

- The good habits and good discipline
  - Never assume your data file is correct.
  - If you assume it, sometime it causes you days in debugging but actually your program works correctly.
  - Data file can be corrupted for different kind of reasons which you can never know and expect


The source of errors
assert
Class or Method
assert
Class or Method
assert
Class or Method
assert
assert
Program crashes or Display incorrect results

SUMMARY

- Assertion can help your signal an error could already occurs, which is closer to the source of error you need to look for.
- Assertions are never too many,
- Remember,
  - crashes on an assert is good during development.


Well, if your system crashes without being catched by asserts ?
I could be a really bad sign
well..... i'm f****d
GrumpyCatPics.com

SOFTWARE TESTING

教育部資訊人才培育計畫

A JOKE

有天晚上我參加一個老同學的喜宴，
同桌的人有律師、會計師，
只有我一個是電腦業界的人。
為了引起話題，我說我羨慕其他行業的人，像是律師、會計師等，在學校學的那一套終生都受用，每年就算有新法條也不會改太多。
不像我們電腦界的人，每天都在K新的技術手冊，新產品新技術隨時大翻新，改朝換代的速度讓人措手不及，所以學電腦的人看起來都比較蒼老，因為太辛苦了。

我話剛說完，一位會計師馬上回我一句：「你錯了，其實我們最羨慕你們電腦界的人，因為沒有任何行業的消費者像你們電腦界的消費者一樣好欺負。」

頓時我成了眾矢之的，
這些電腦使用者的抱怨全部集中到我身上來。
其中一位仁兄還舉了個有趣的例子，

他說：
「如果傢俱業和電腦業一樣，世界會變成什麼樣子？」以下是他講的故事：

如果傢俱業跟電腦業一樣，比如說我到傢俱店買了一張桌子，搬回家往地板上一放，啪啦一聲桌子就塌了。這時候我不會生氣、不會罵人，我會先自己檢查一下出了什麼錯。

我會先檢討自己，
是不是我做錯了什麼，
或是我對桌子的使用不夠熟悉。

於是我會去買書來看
(書名可能是《快快樂樂學修桌子》、
《21天學會修桌子》、《修桌子技巧與實例》、
或是《修桌子的聖經》)。

要是書看得不太懂，
我會再花錢去報名上修桌子的課程。

學完之後還是修不好，
我會請其他比較懂得修桌子的朋友來幫忙。

最後沒有辦法，
終於我打電話給原先的傢俱行，
(可能還要購買《技術支援方案》)，
結果他們跟我說：
「唉呀！你買到的是搶鮮版啦！
本來就應該有問題的」。

於是我恍然大悟原來是自己的錯，
我就再去買一張「正式版」的桌子。

回家一擺還是啪啦又塌了！

修了半天還是有問題，
再請傢俱行的技術人員來做仔細檢查，
最後終於發現問題的所在——
「我家的地板和桌子不相容」，
又是我自己的錯，
於是我得趕快幫家裡的地板升級.....。

等一切都忙完了，
桌子可以使用了，
我趴在桌上寫字，
心裡充滿了成就感，
我很得意地跟網友分享我修理桌子的經驗，
並暗自慶幸自己在科技的潮流上沒有落
伍.....。

THE IMPORTANCE OF TESTING

- Seldom recognized by academic
- Professors don't care
- students don't care
- but ACM survey more than 1000 software companies. Three topics are suggested to be important to CS students
  - Teamwork
  - Software Engineering
  - Software Testing

教育部顧問室
Ministry of Education
Advisory Office

WHY TESTING?

- Let's look at some well-known software defects
  - 台灣高鐵售票系統：系統上線之後當機連連，系統無法應付突然湧現的購票人潮（no stress testing)
  - 台灣彩卷系統：類似的問題
  - 2000-2005- 巴拿馬國家癌症中心—5個病人接受過量的迦瑪射線照射死亡。15人引發了嚴重的併發症
  - 2003 —軟體造成美國東北部及加拿大停電。5000萬人受影響，3人喪生
  - 2000 美國海軍飛機墜落，4人喪生 （控制軟體問題）
  - 1997 韓國空難，225人喪生 （雷達控制軟體問題）
  - 1995 美國航空在哥倫比亞撞山159人喪生（導航軟體問題）

STATISTICS

- 2002- 美國國家標準局報告─軟體品質每年造成595億美元（0.6% GDP）
- 國內又如何？
  - 台北縣政府校務行政系統案例
  - 台師大林口校區學務系統案例

QUALITY CONTROL AND QUALITY ASSURANCE

- Software is a product.
- Good quality product requires 品質控管quality control (QC) and quality assurance (QA) 品質確保

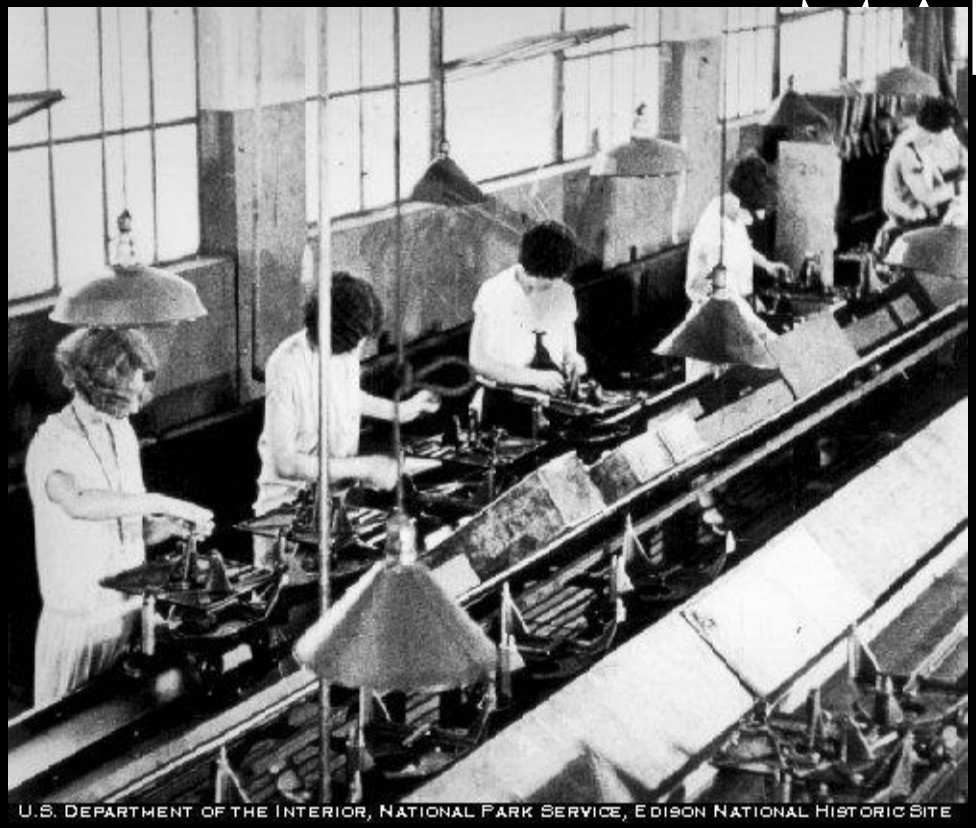
MANUFACTURING IN OTHER DISCIPLINE

- 在製造業的世界裡面
  - 產品品質(quality)是很重要的一個因素
  - 當你投入非常多成本與原物料時你永遠希望你生產出來100個產品有100個都是可以賣的
- QC (Quality Control) 品質控管
  - 製造過程從設計到生產可能有數十道到幾百道程序
  - 每一道程序都可能影響到後面的品質
  - 如何透過製造流程的改善(process improvement)來提高良率，一直都是製造工業的重要課題

HOW TO IMPROVE QUALITY?

- 在製造過程中如果能夠能夠使用機器，最好的改善品質的方式就是自動化。機器不容易犯錯而且可以不斷地重複單調無聊的工作。機器會出錯的時候通常是由於製造機器的磨損，必須重新校正。
  - Ex. 日本的步進馬達精確度超高的秘密。
- Question
  - 當某部分的工作不能由機器來做的時候，製造過程如何做到品質控管？

ASSEMBLY LINE IN MANUFACTURING



U.S. Department of the Interior, National Park Service, Edison National Historic Site

生產線

- 每個生產線員工只做一件夠簡單到不容易出錯的工作
- 生產線員工不需要夠聰明
- 生產線將複雜的產品製造切割成許多小步驟可以在短時間以及最少的技術內可以完成
- 發現產品的問題並不難.

AGAIN, LET'S REVIEW THE ENGINEERING PROCESS OF OTHER ENGINEERING FIELDS.

1. Idea（概念）
2. Marketing analysis (Requirement Analysis)（市場分析）
3. Analysis and Design（分析與設計）
4. Manufacturing (QC)（製造）
5. Testing (QA)（測試）
6. Release product（產品出廠）

SOFTWARE ENGINEERING

1. Idea（概念）
2. Requirement analysis and specification (100% design)（需求分析與規格）
3. Design and analysis（分析與設計）(100% design, QC here?)
4. Implementation（程式實做）(95% design?, QC here?)
5. Manufacturing（製造）（compilation – no cost manufacturing)
6. Testing（測試）

QUESTION?

- Is coding a design process or a manufacturing process
- Is Code a design or a product?
- Again, the analog breaks

More questions

- If coding is not a design process, we should be able to hire many 生產線女工 to control our quality
- If coding is a design process, bad news, errors are inevitable because programmers are always doing complicated design jobs, not simple and easy job.

FACTS

- Now, you should know why software has so many bugs or notorious for being flawed

PROGRAMMERS V.S. ASSEMBLY LINE WORKER

- Programming is a job which requires perfection !! Hu
  complete perfe
- Programmers m
  complexity ano
- Humans are no
  perfect jobs (we
  "Natural Selecti

TESTING

I FIND YOUR LACK OF TESTS DISTURBING.

SOFTWARE QUALITY

- Unfortunately, software must be completed by humans, and humans are error-prone and a lot of errors !!
- You should learn not to trust programmers
- This is why testing is so important in matured software industry !!

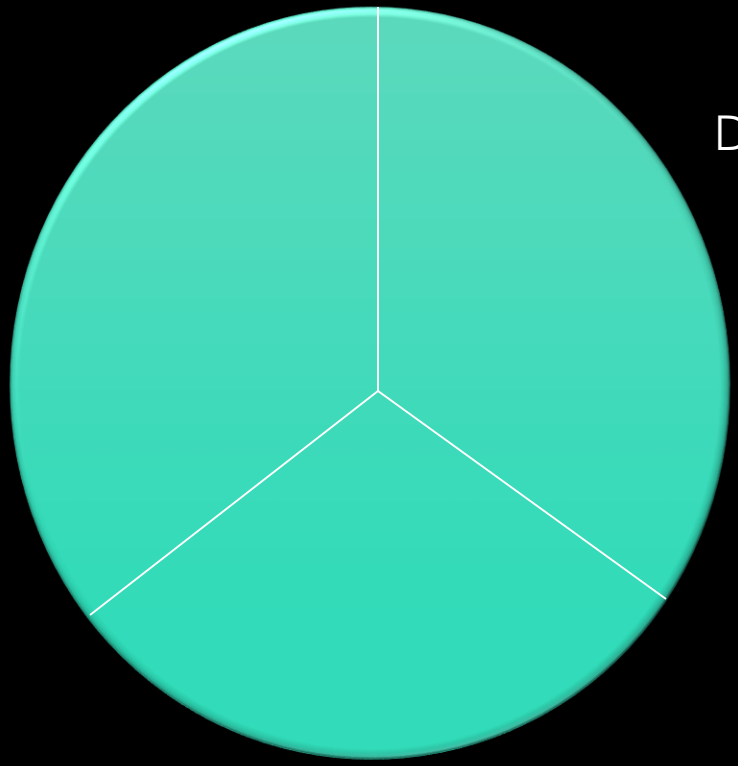
PRACTICAL TESTING PRACTICES

教育部資訊人才培育計畫

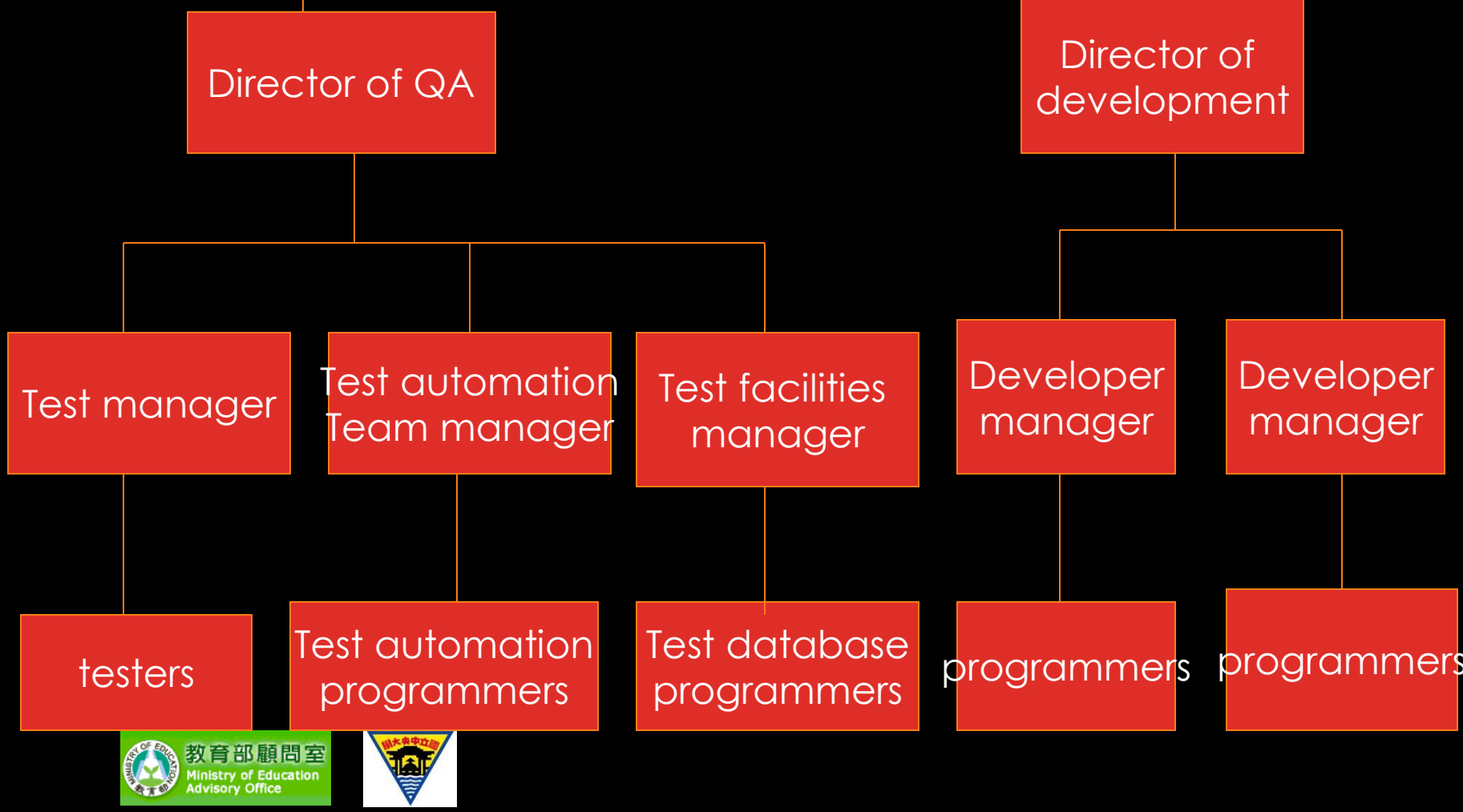
SDLC (SOFTWARE DEVELOPMENT LIFE CYCLE)

- Requirement analysis (marketing research)
- Software function/performance specification (QAs should join the project here to write a test plan)
- Analysis and Design 分析
- Coding and unit testing (by programmers)
- Integration testing (by programmers/QAs)
- **Alpha testing (by QAs – a set of test cases is prepared)**
  - Most software functions and features are basically completed
  - All functions are tested, no functions will be added beyond this point
  - Serious flaws (high severity) are solved and addressed (show stopper)
- **Beta testing (by Beta users)**
  - sub-serious bugs are all fixed
  - Test plan has been completely executed
  - Bug discovering rate is lower than bug fixing rate
- **Release**
  - Bug discovering rate is lower than bug fixing rate for a long period of time.
  - The version after fixing bugs has been regressively tested (regression test)
  - Quality is formally proved by QA team
  - All documents are ready

MATURED SOFTWARE COMPANY



JOB HIERARCHY



WHY QA MUST BE INDEPENDENT FROM DEVELOPMENT TEAM?

- Only VIRGO programmers may have the idea of perfection.
- To tell programmers to play the role of a tester is just like to have a judge admits he is wrong.
  - if he finds so many bugs, it proves him is a poor programmer
  - programmer like to finish a job not to complete it to its perfection.
- From the above
  - QA must stand on the opposite side of developers regarding quality issues.
  - QA got promoted when he catches more serious bugs.
  - QA cannot be managed under developer team

Marketing

balance of terrors

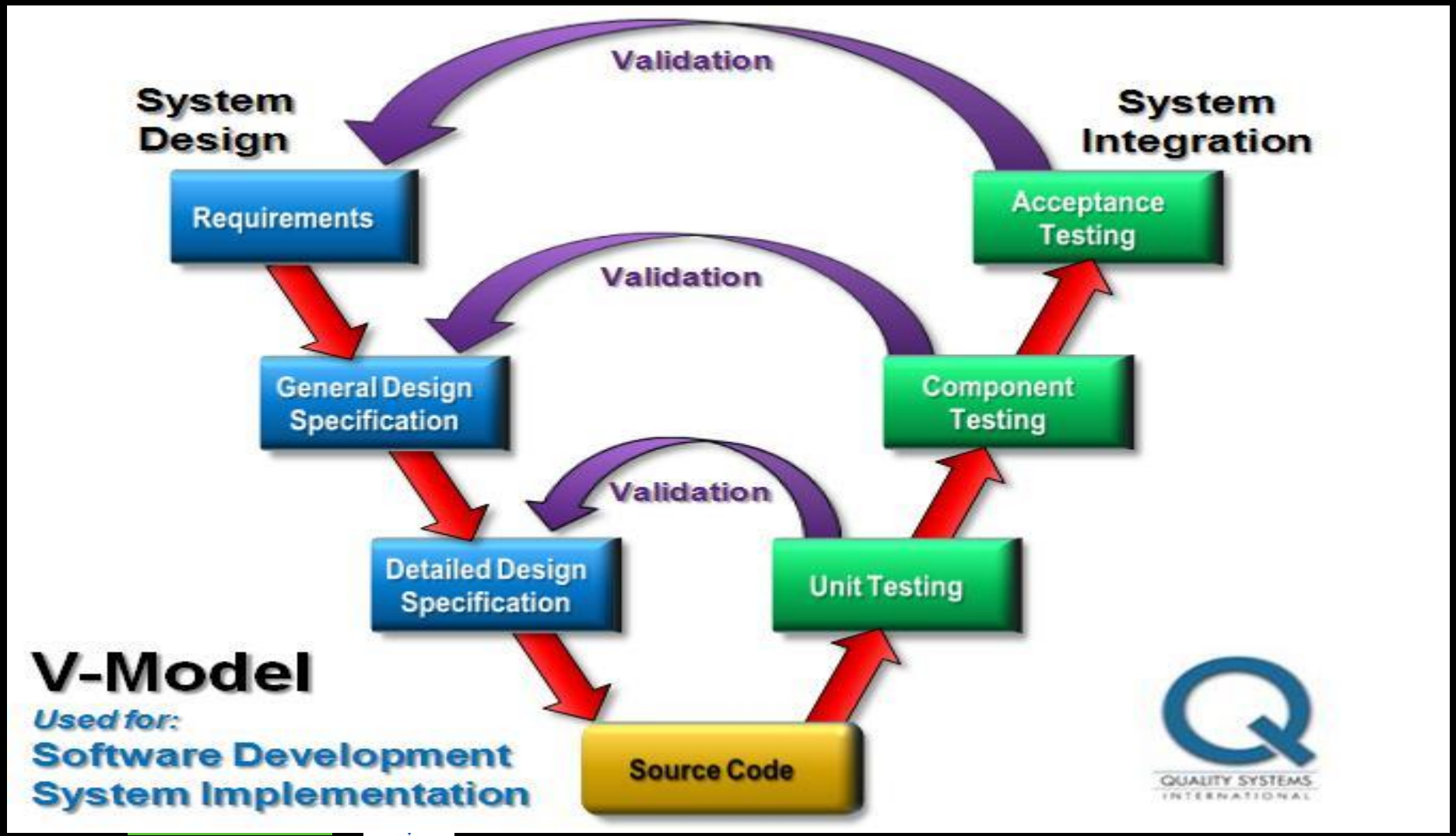
QA (testers)

Developer

WHAT TO TEST IN SOFTWARE TESTING?

- completeness
  - whether the software functionality is conformed to software specs (SRS)
- correctness (functional specs)
- reliability (nonfunctional specs)
- compatibility (nonfunctional specs)
- efficiency(nonfunctional specs)
- usability (nonfunctional specs)
- portability (nonfunctional specs)
- scalability (nonfunctional specs)
- testability (nonfunctional specs)

SOFTWARE TESTING V-MODEL



TESTING TASKS TO BE DONE

1. **Write Test plan (often seen in large software company with formal QA team)**
2. write test cases, prepare test data
3. test case management
4. file bugs
5. regression testing
6. test automation (with tools)
7. stress testing/load testing
8. security testing
9. test as assets.

TEST PLAN?

- There are many test plan template
- The major contents in a test plan
    - what to test?
    - what not to test?
    - estimate the cost and efforts of testing
    - prioritize testing tasks
    - how test data should be prepared.
    - testability
    - testing schedule
- via review, developers and testers can communicate and check
    - if specs are understood mutually (communication is not as intuitive as you think)
    - raise any testing issues while studying the system specs

TESTING TASKS TO BE DONE

1. Write Test plan (often seen in large software company with formal QA team)
2. **write test cases, prepare test data**
3. test case management
4. file bugs
5. regression testing
6. test automation (with tools)
7. stress testing/load testing
8. security testing
9. test as assets.

TestID

T-VM-PLUG-0031A-Test plug-in : Minerva plug-in

Revision History

2013/01/21 created

Summary

利用視覺化一個cube_uvbm來測試Minerva plug-in是否能與xDIVA正常溝通

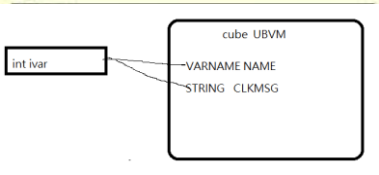
Tested Story

Creator

Zoe

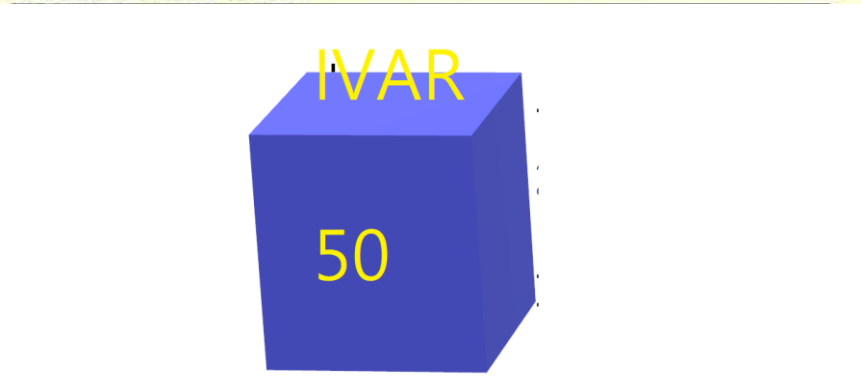
Test Steps

1. open the ubvm.java with the Minerva
2. Set break points at line 15 // ivar = 100
3. Run xDIVA
4. click the Debug menu -> Visualize ivar
5. When the mapping dialog pops up, do mapping
   ->



1. Place a cube_ubvm (select tab "ubvm")
2. Set the attribute of cube: select the tab "Input" on the right, check the sizex to make sizex visible
3. Link ivar to ports "_vm_name" , _vm_sizex and _vm_clickmsg
4. Press "Apply"

6. -> cube sizex is 50



A REAL TEST CASE Design When build is not ready

assertions

assertions

TestID

T-VM-PLUG-0031A-Test plug-in : Minerva plug-in

Revision History

2013/01/21 created

Summary

利用視覺化一個cube_uvbm來測試Minerva plug-in是否能與xDIVA正常溝通

Tested Story

Creator

Zoe

Test Steps

1. open the ubvm.java with the Minerva
2. Set break points at line 15 // ivar = 100
3. Run xDIVA
4. click the Debug menu -> Visualize ivar
5. When the mapping dialog pops up, do mapping ->

   1. Place a cube_ubvm (select tab "ubvm")
   2. Set the attribute of cube: select the tab "Input" on the right, check the sizex to make sizex visible
   3. Link ivar to ports "_vm_name" , _vm_sizex and _vm_clickmsg
   4. Press "Apply"

6. -> cube sizex is 50

When it becomes a regression test cases with Stable golden cases

TEST CASE ID EXAMPLE

- **T-SSSS-CCCCC-XXX-DDDDD TITLE**
- **SSSS**: Test Suite name
- **CCCCC**: Component or Requirement abbreviation name –
  - by this name, a tester can quickly grasp what this test case is designed for
  - it can be a component or requirement abbreviation names.
- **XXX**: X 的值可以是
  - ‘A’ the test case be automatically executed
  - ‘M’ the test case must be manually executed
  - ‘P’ this is a positive case
  - ‘N’ this is negative case
  - 'B' this is a boundary case
  - '0' test case priority (smoke test case can be 0)
  - '1' ..'9'
- **DDDD**: test case ID in a test suite. ID is not necessary be an integer. If it is an integer like “0008”, an alternative 0008A means a similar test cases.
- **TITLE**: 50 words that can be easily displayed in a test case management tools.

TESTING TASKS TO BE DONE

1. Write Test plan (often seen in large software company with formal QA team)
2. write test cases, prepare test data
3. **test case management**
4. file bugs
5. regression testing
6. test automation (with tools)
7. stress testing/load testing
8. security testing
9. test as assets.

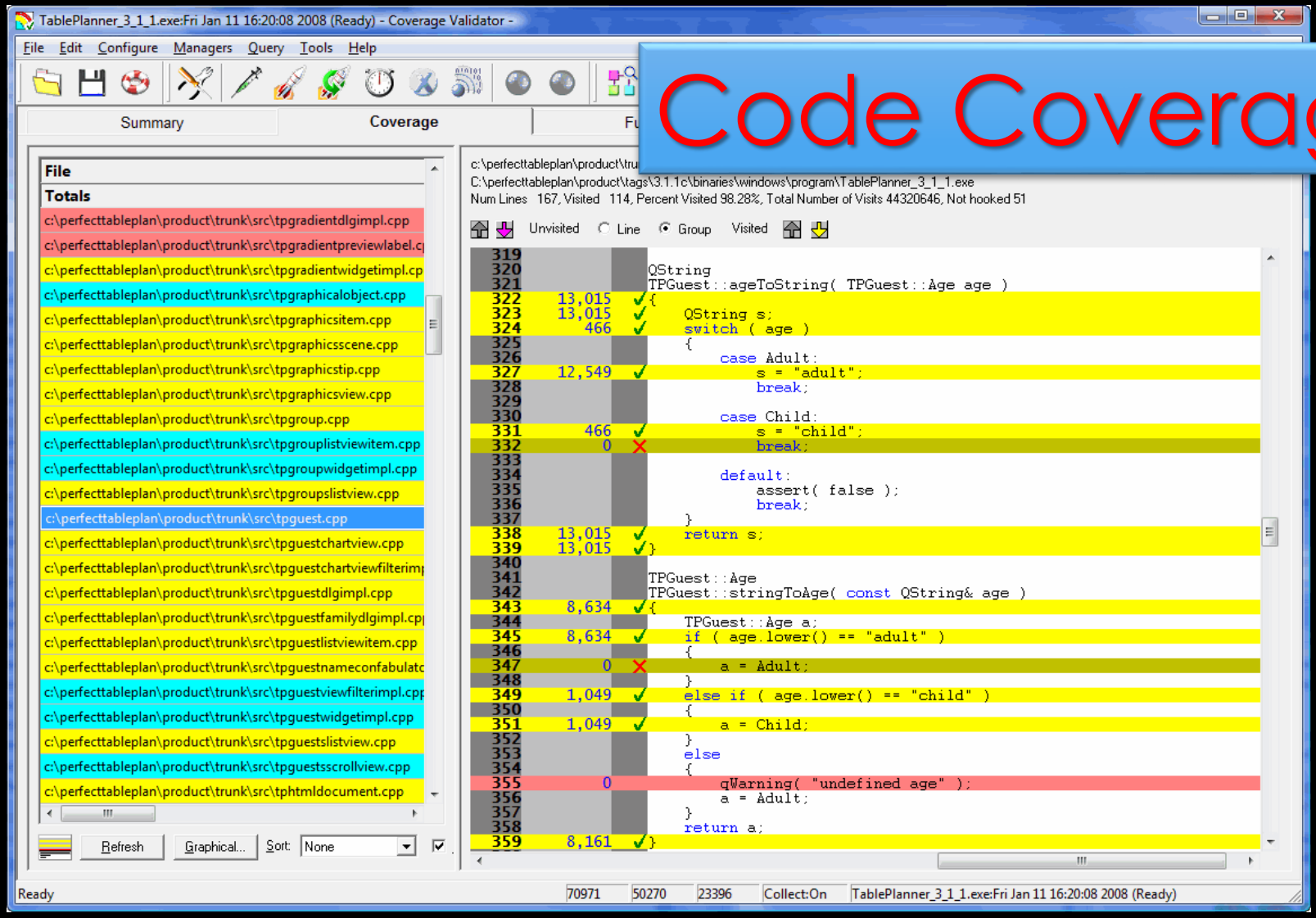
TEST CASE MANAGEMENT EXAMPLE USING REDMINE

#	Tracker	Status	Priority	Subject	Assignee	Updated	test result
985	test case	New	Normal	T-VM-LAYOT-0033-Varname : used in layout	珮宜 何	01/21/2013 12:38 pm	passed
984	test case	New	Normal	T-VM-PLUG-0032B-Test to visualize two variables with difference plug-in : Eclipse plug-in		01/21/2013 12:35 pm	none
983	test case	New	Normal	T-VM-PLUG-0032A-Test to visualize two variables with difference plug-in : Minerva plug-in		01/21/2013 12:32 pm	none
982	test case	New	Normal	T-VM-PLUG-0032-Test to visualize two variables with difference plug-in : VS plug-in		01/21/2013 12:27 pm	none
981	test case	New	Normal	T-VM-PLUG-0031B-Test plug-in : Eclipse plug-in	Ricky Chien	01/21/2013 12:19 pm	passed
980	test case	New	Normal	T-VM-PLUG-0031A-Test plug-in : Minerva plug-in	珮宜 何	01/21/2013 12:18 pm	passed
979	test case	New	Normal	T-VM-PLUG-0031-Test plug-in : VS plug-in	冠華 丁	01/21/2013 12:16 pm	passed
978	test case	New	Normal	T-VM-VP-0030-animation : by visual point		01/21/2013 12:11 pm	none
977	test case	New	Normal	T-VM-LAYOT-0029-Layout : attribute in the container		01/21/2013 12:04 pm	none
976	test case	New	Normal	T-VM-VP-0028A- visual point : when the variable doesn't exist		01/21/2013 11:57 am	none
975	test case	New	Normal	T-VM-VP-0028- visual point : a ball change with its radius		01/21/2013 11:52 am	none
974	test case	New	Normal	T-VM-TM-0027B- Array : size 0 array		01/21/2013 11:42 am	none
973	test case	New	Normal	T-VM-TM-0027A- Array : array unfold		01/21/2013 11:36 am	none
972	test case	New	Normal	T-VM-TM-0027- Array : array type mapping		01/21/2013 11:28 am	none
971	test case	New	Normal	T-VM-TM-0026D-Test unfolding : Can't unfold when pointer is null	冠華 丁	01/21/2013 11:22 am	passed
970	test case	New	Normal	T-VM-TM-0026C-Test unfolding : Can't unfold when the type is loop reference	新志 陳	01/21/2013 11:17 am	none
968	test case	New	Normal	T-VM-TM-0026B-Test unfolding : Can't unfold when the type is the same as class type	Ricky Chien	01/21/2013 10:44 am	passed
967	test case	New	Normal	T-VM-TM-0026A-Test unfolding : Can't unfold when the type is doing mapping	珮宜 何	01/21/2013 10:40 am	passed
966	test case	New	Normal	T-VM-TM-0026-Test unfolding : binary tree	冠華 丁	01/21/2013 10:36 am	passed
965	test case	New	Normal	T-VM-TM-0025E-Test type mapping dialog : ref type variable only connect to reference type VM	新志 陳	01/21/2013 10:24 am	passed
964	test case	New	Normal	T-VM-TM-0025D-Test type mapping dialog : Type mapping should has single root	Ricky Chien	01/21/2013 10:13 am	failed
963	test case	New	Normal	T-VM-TM-0025C-Test type mapping dialog : TMD doesn't have the "apply all"		01/21/2013 10:03 am	none
962	test case	New	Normal	T-VM-TM-0025B-Test type mapping dialog : TMD input port has default gates	冠華 丁	01/21/2013 10:00 am	passed
961	test case	New	Normal	T-VM-TM-0025A-Test type mapping dialog : save a binary tree type node and import to toolbar	新志 陳	01/21/2013 09:56 am	failed
960	test case	New	Normal	T-VM-TM-0025-Test type mapping : binary tree	Ricky Chien	01/21/2013 09:48 am	passed

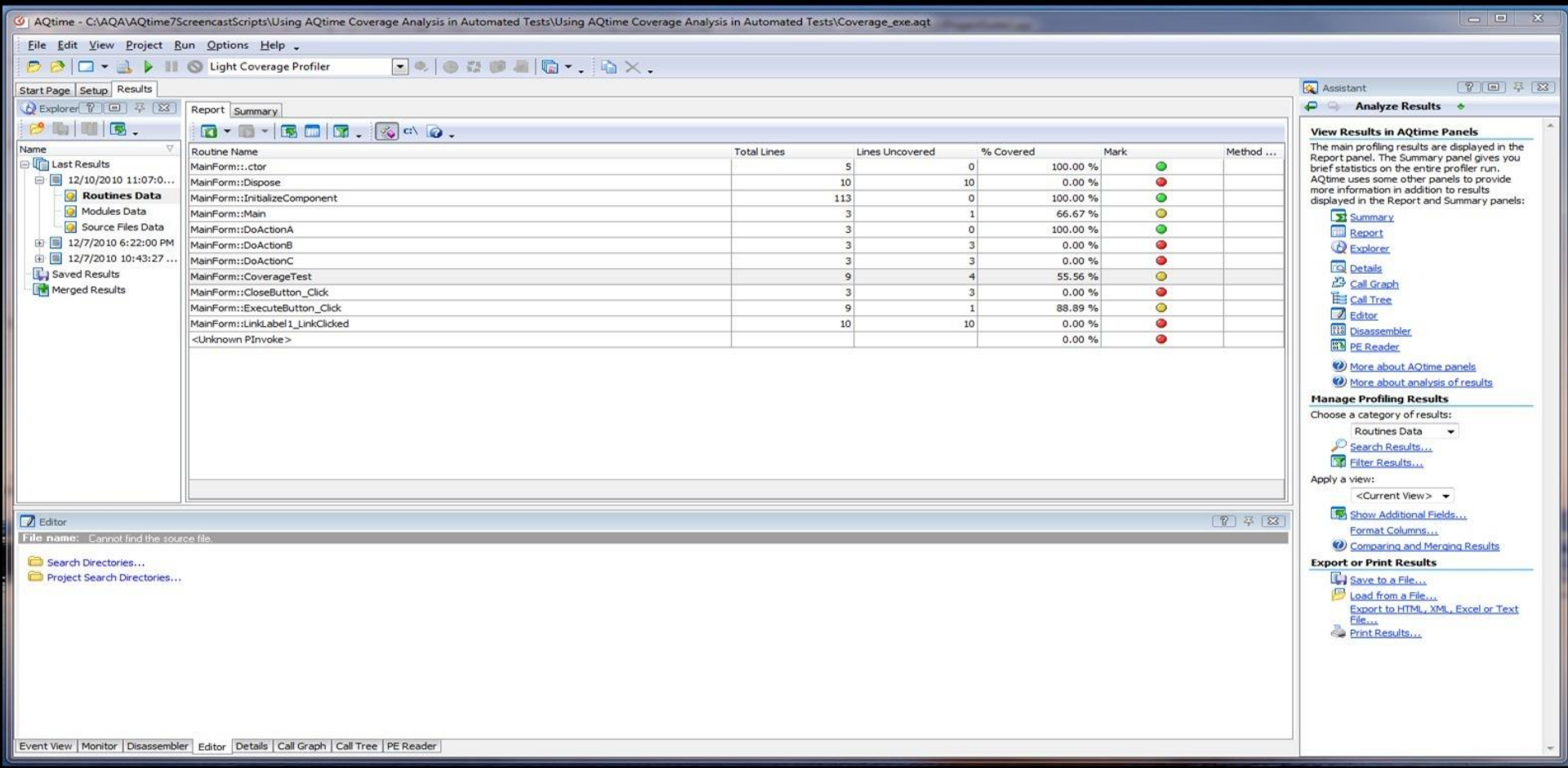
KORAT'S DREAM

- Remember !! No test cases no quality !
- However, programmers hate writing test cases, particularly the paper work parts.
- QA team does it (if you have one)
- Can a tool record my testing activities and then make them into regression test cases?
- So, I don't need to write test cases

HOW DO YOU VERIFY IF THE TEST CASES ARE GOOD?



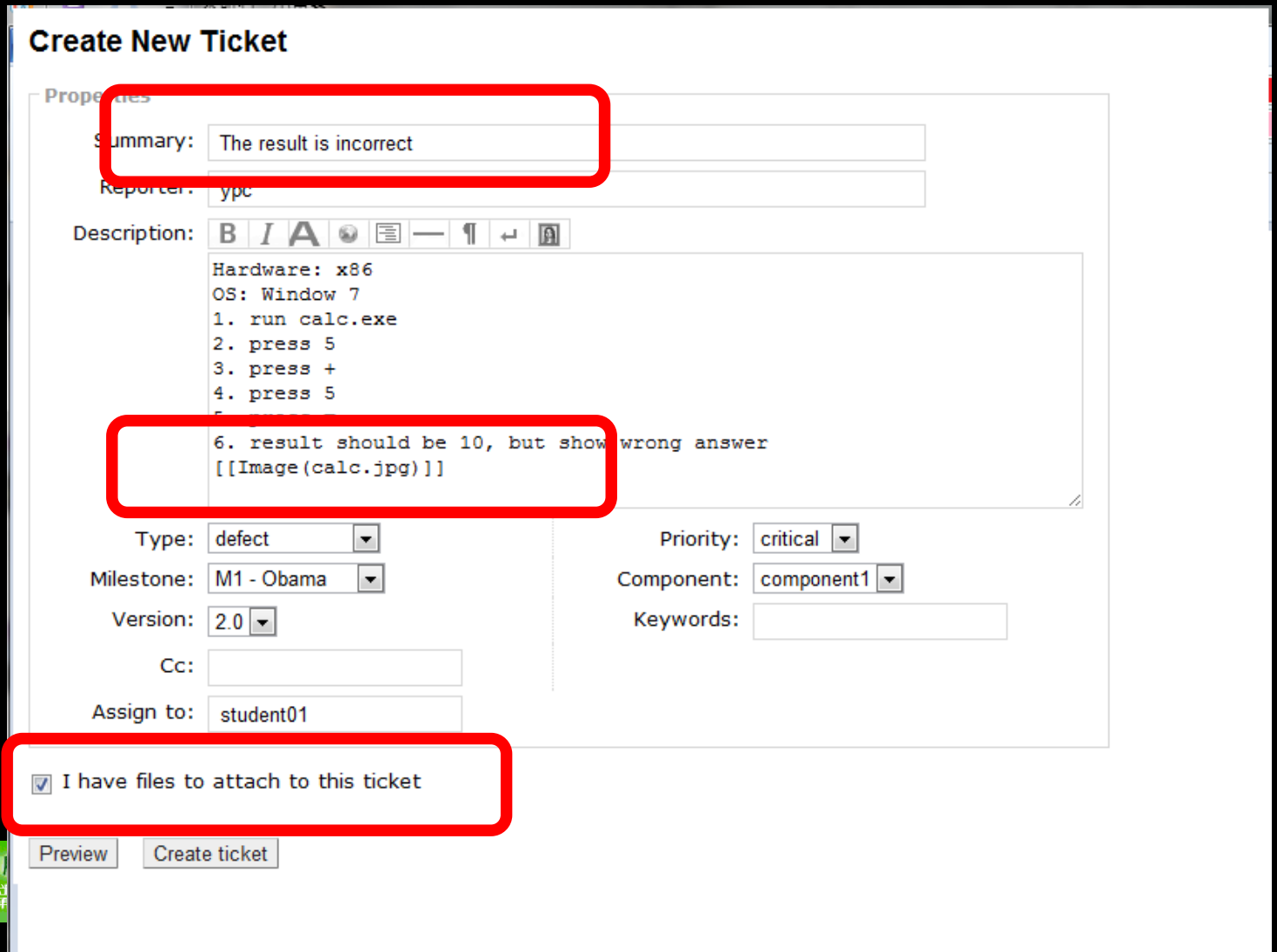
CODE COVERAGE ANALYSIS



TESTING TASKS TO BE DONE

1. Write Test plan (often seen in large software company with formal QA team)
2. write test cases, prepare test data
3. test case management
4. **file bugs in bug report system**
5. regression testing
6. test automation (with tools)
7. stress testing/load testing
8. security testing
9. test as assets.

FILE A BUG IN A MODERN WIKI BUG REPORT SYSTEM



Ticket #1 (new defect)

The result is in correct

Opened 6 minutes ago

Reported by:	ypc	Owned by:	student01
Priority:	critical	Milestone:	M1 - Obama
Component:	component1	Version:	2.0
Keywords:		Cc:	

Description

Reply

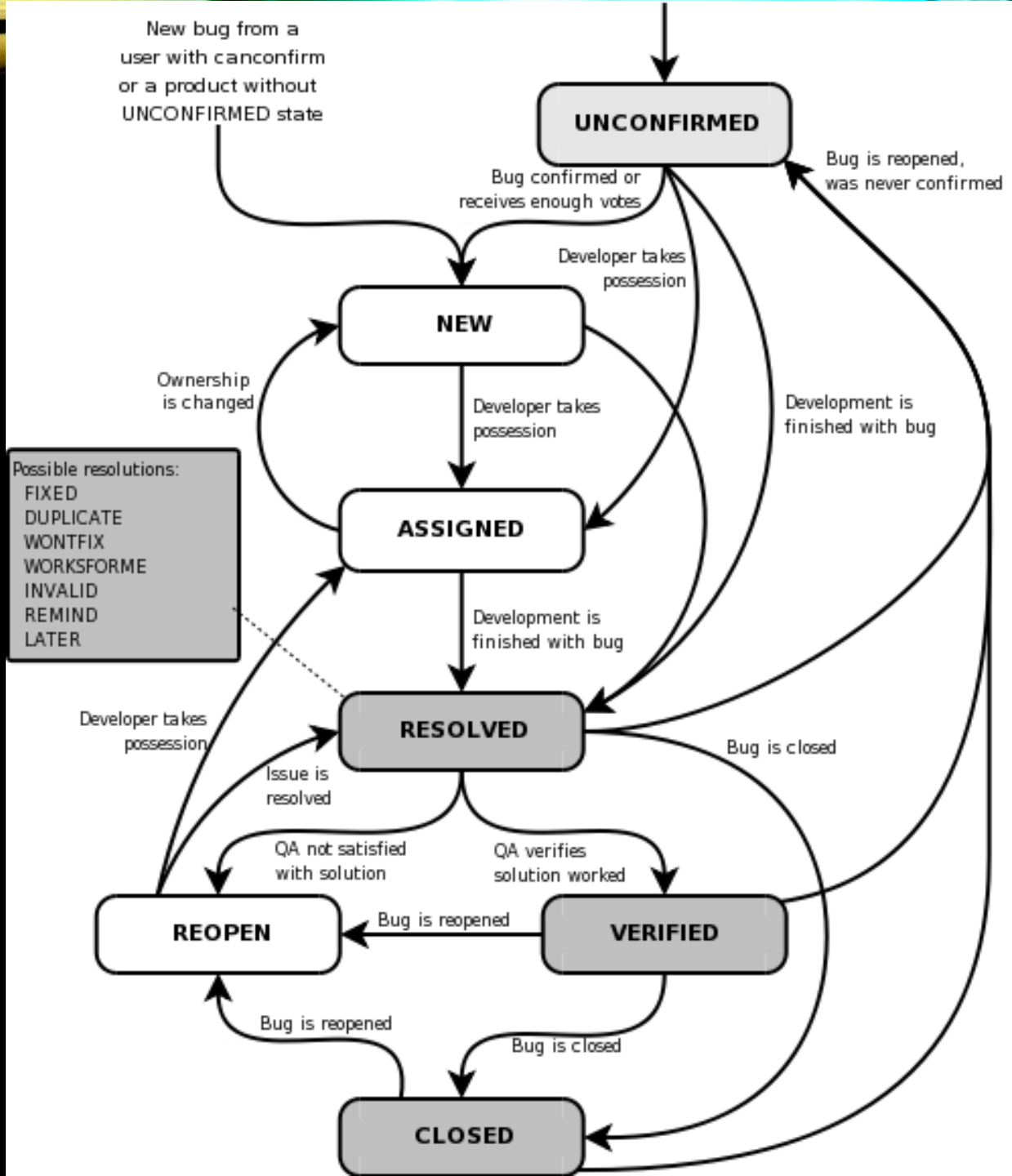
Hardware: x86
OS: Window 7
1. run calc.exe
2. press 5
3. press +
4. press 5
5. press =
6. result should be 10, but show wrong answer

A sample bug report

ATTACH ANY FILES THAT CONCERN THE BUG

- Files to attach include
  - input file/data
  - output file/data
  - Log files
  - Core dump (memory dump)
  - Error log files
  - Tracing information (generated by Tracing tools)
  - ……
- A test report should provide any information as much as possible.

- bug life cycle



SEVERITY OF SOFTWARE DEFECTS

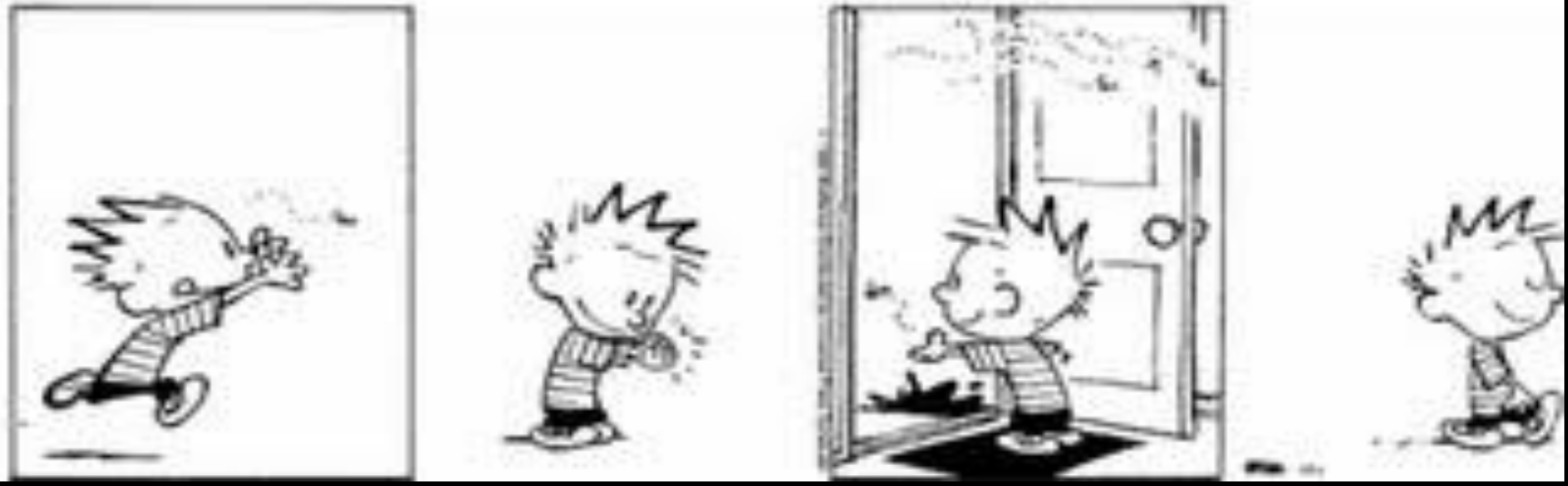
- show stopper (severity 1)
- unusable software (severity 1)
- Microsoft ranks the bug severity from 1-30
- killing people, waste a lot of money.

TESTING TASKS TO BE DONE

1. write Test plan (often seen in large software company with formal QA team)
2. write test cases, prepare test data
3. test case management
4. file bugs in bug report system
5. **regression testing & Test Automation (with tools)**
6. stress testing/load testing
7. security testing
8. test as assets.

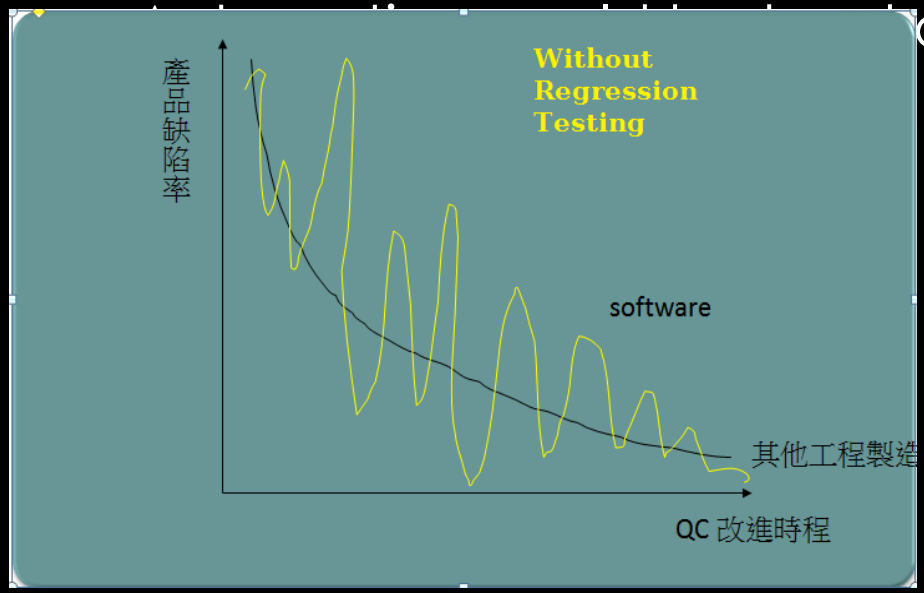
REGRESSION TESTING

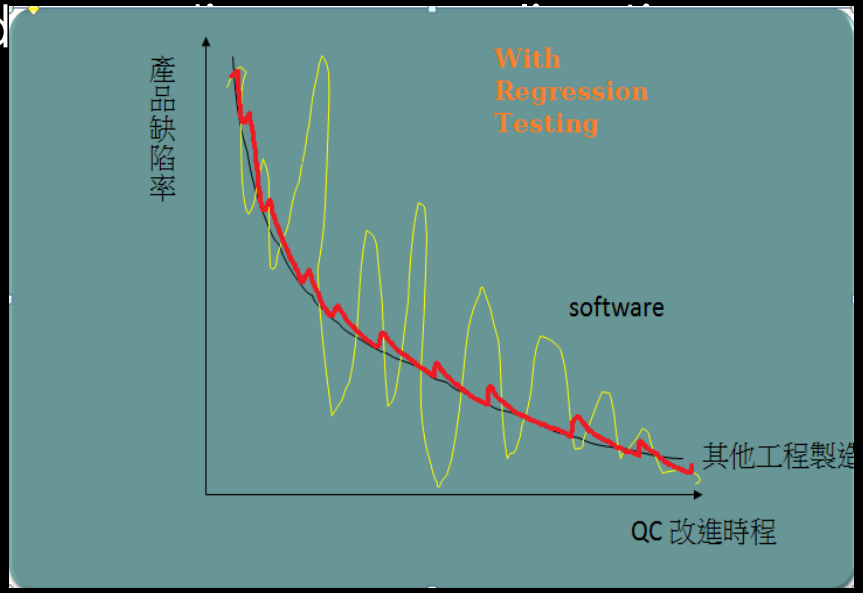
Regression:
"when you fix one bug, you introduce several newer bugs."

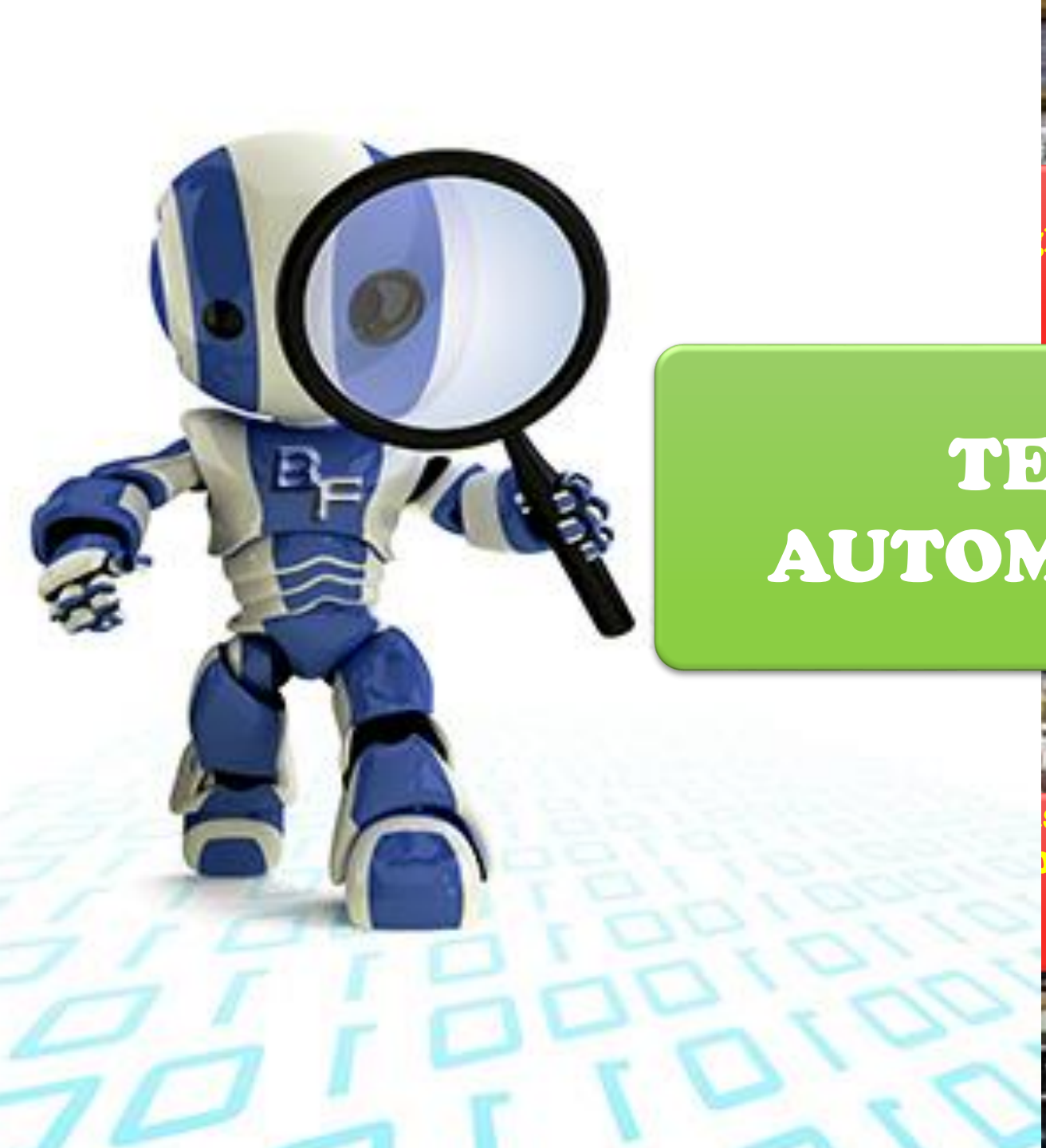


REGRESSION TESTING

- Rerun all the test cases when the program is modified, extended, or revised.







TEST AUTOMATION

ht-

sands of test cases?
00 test cases, they

TESTING AUTOMATION

- In many applications or functionalities, testing can be automated at different degrees. In contrast, many functionalities and applications are not amenable to automatic testing, where humans are needed
- examples that is difficult to automate testing
  - testing WORD.
- examples that is easy to automate
  - testing a sorting algorithm – you can easily write a program to check if the output is listed in ascending or descending order

AUTOMATIC TEST CASE GENERATION AND TEST ORACLES

- If test cases can be derived automatically and then these test cases can be executed automatically and then report the error – **What a perfect world !** We can fire all the testers.
- In practice, this is only a dream
- In many applications, Effective test cases can only be derived by human
- In many applications, Executions requires human to monitor and validate if program behaviors are conformed to specs.

TEST ORACLES

- Oracle – In ancient Greece, an oracle was a priest or priestess who made statements about future events or about the truths

- A test oracle is a program that can determine if the program behaviors or program input/output are conformed to specs.

教育部顧問室
Ministry of Education
Advisory Office

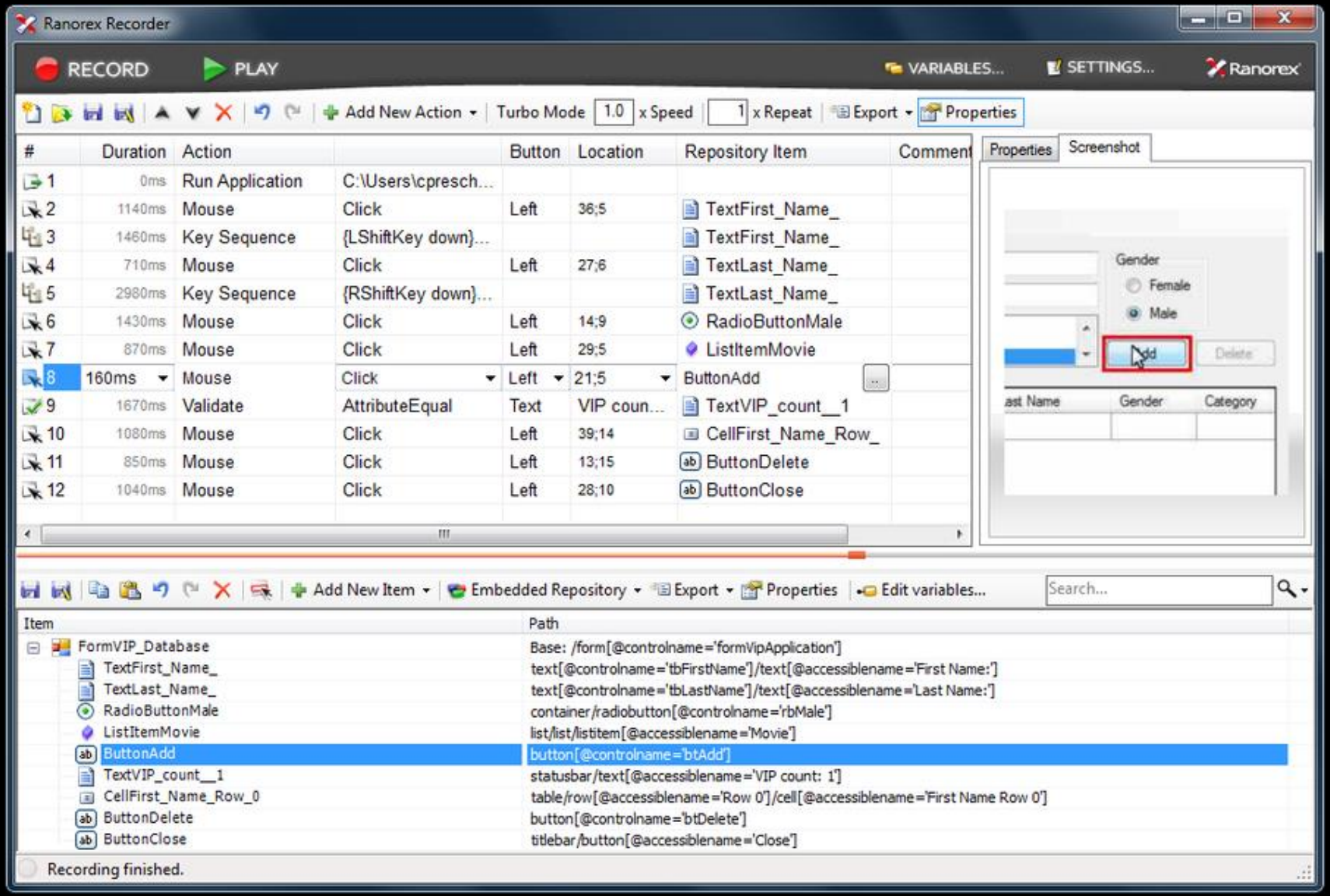
TESTING AUTOMATION

- For testing automation, test oracle must be presented to determine the conformity between specs and programs.
- In some applications, test oracles are easy to derive
- In most applications, test oracles are difficult to implement or in theory, impossible to derive automatically

TEST AUTOMATION TOOLS

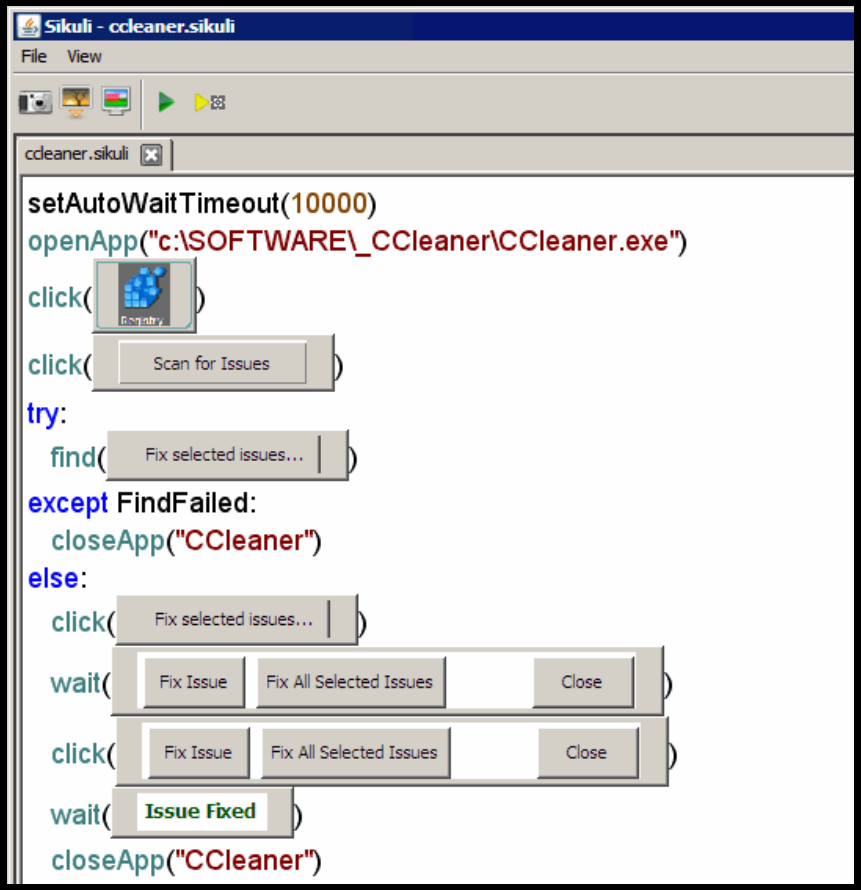
- Applications with GUI
  - Capture/Replay Testing Tools
    - When UI changes, problems could occurs.
  - Android testing framework (based on Junit, xUnit test framework)
    - test cases can be created by coding (many companies adopt such an approach.
- xUnit test framework
  - test automation is done at a unit level (class, packages)
  - can escape from the troubles of setting a system test cases
  - coding intensive (most tasks are done by programmers)

TYPICAL CAPTURE/REPLAY TESTING TOOLS



INNOVATIVE TOOL - SIKULI (FROM MIT)

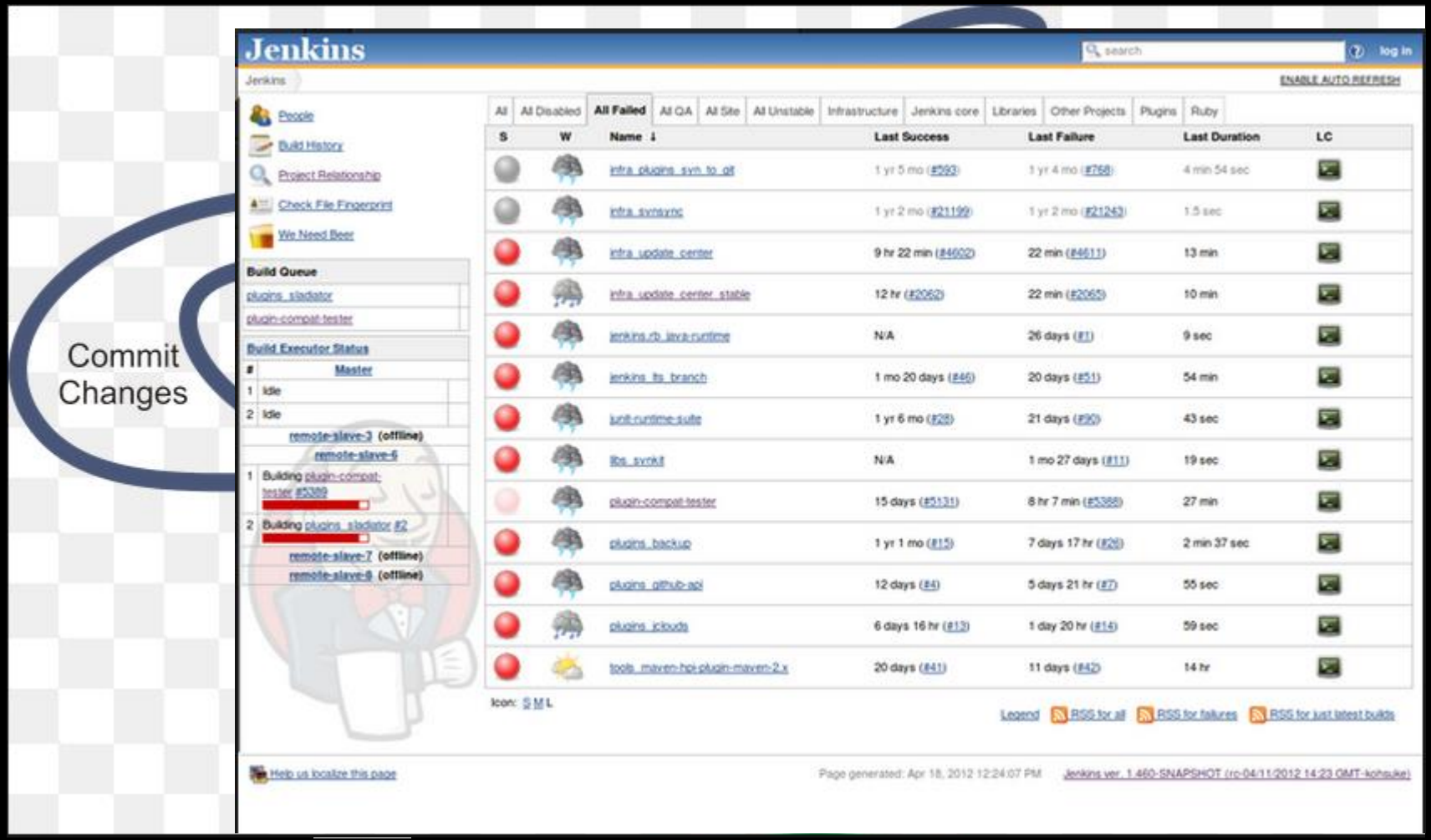
```
1  click( <image> )
2  click( <image: All Programs> )
3  a = getLastMatch().getTarget()
4  a = a.above(100)
5  while True:
6      try:
7          b = find( <image: Accessories> )
8          break
9      except FindFailed:
10         wheel(a, WHEEL_DOWN, 1)
11 click(b)
12 click( <image: Calculator> )
13
```



PROBLEMS WITH CONVENTIONAL CAPTURE/REPLAY

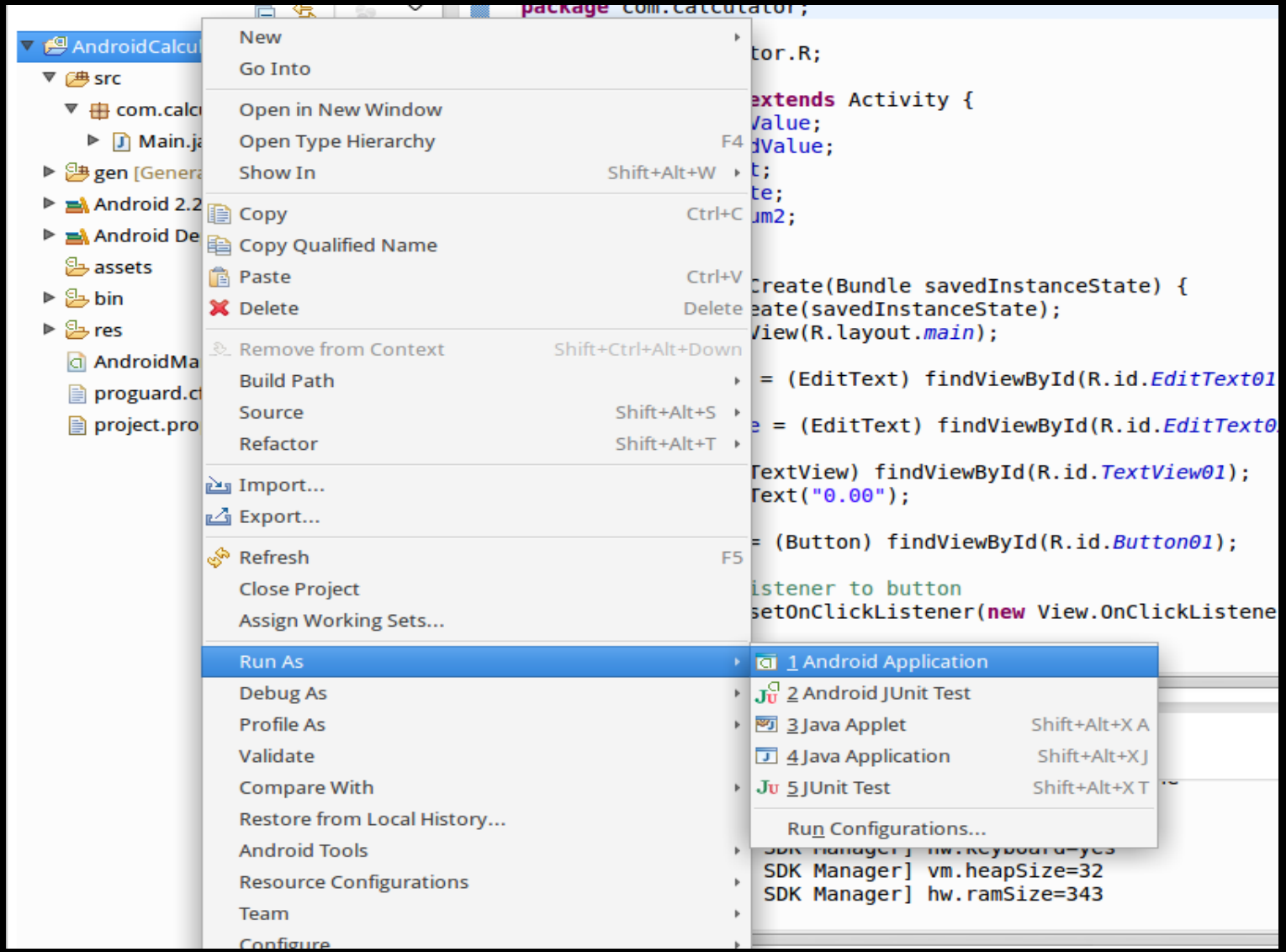
- not adapt to UI changes very well
- not applicable to drag-and-drop events which are common in nowadays multi-touch applications.

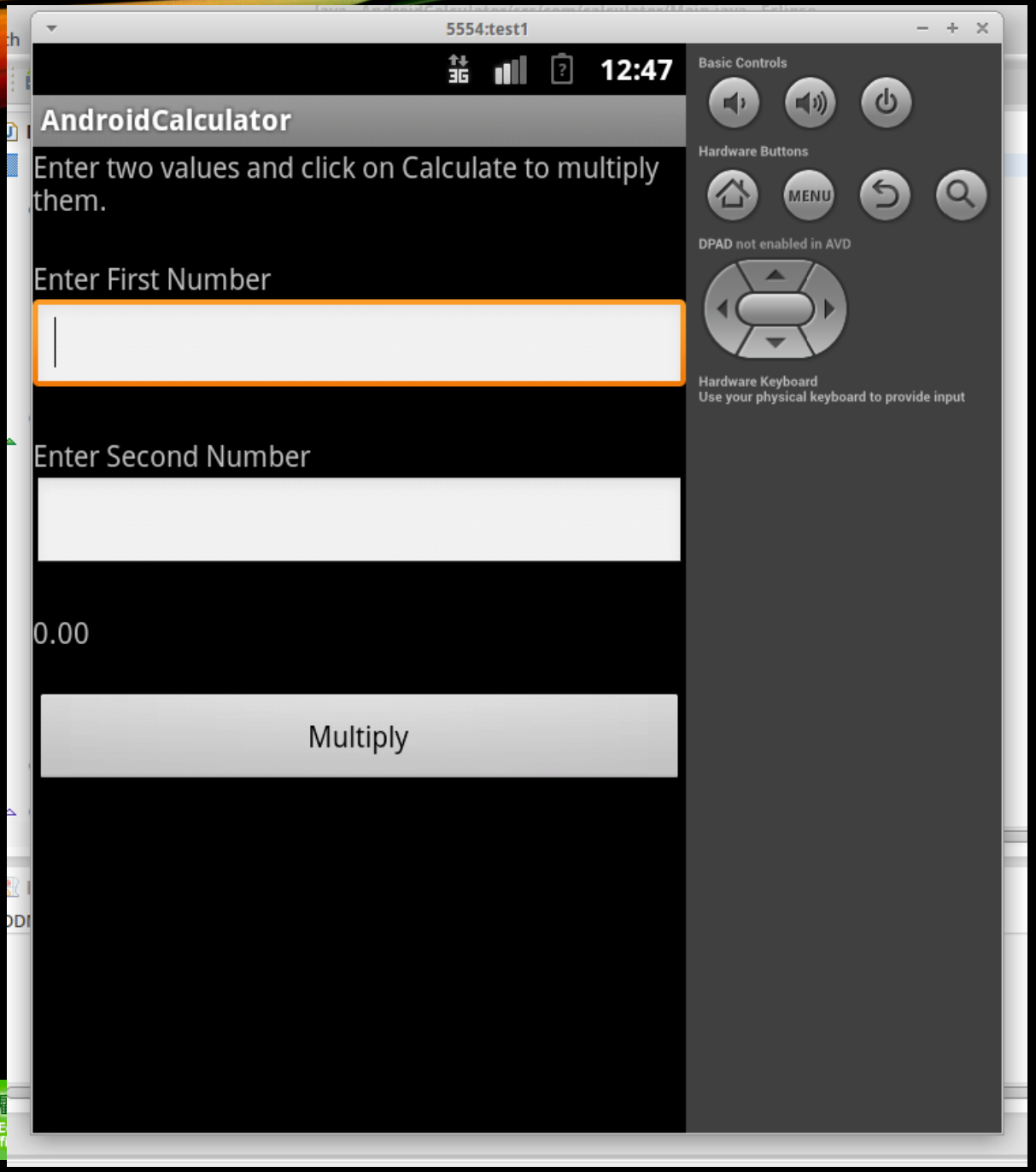
ADVANCED- CONTINUOUS INTEGRATION TOOLS



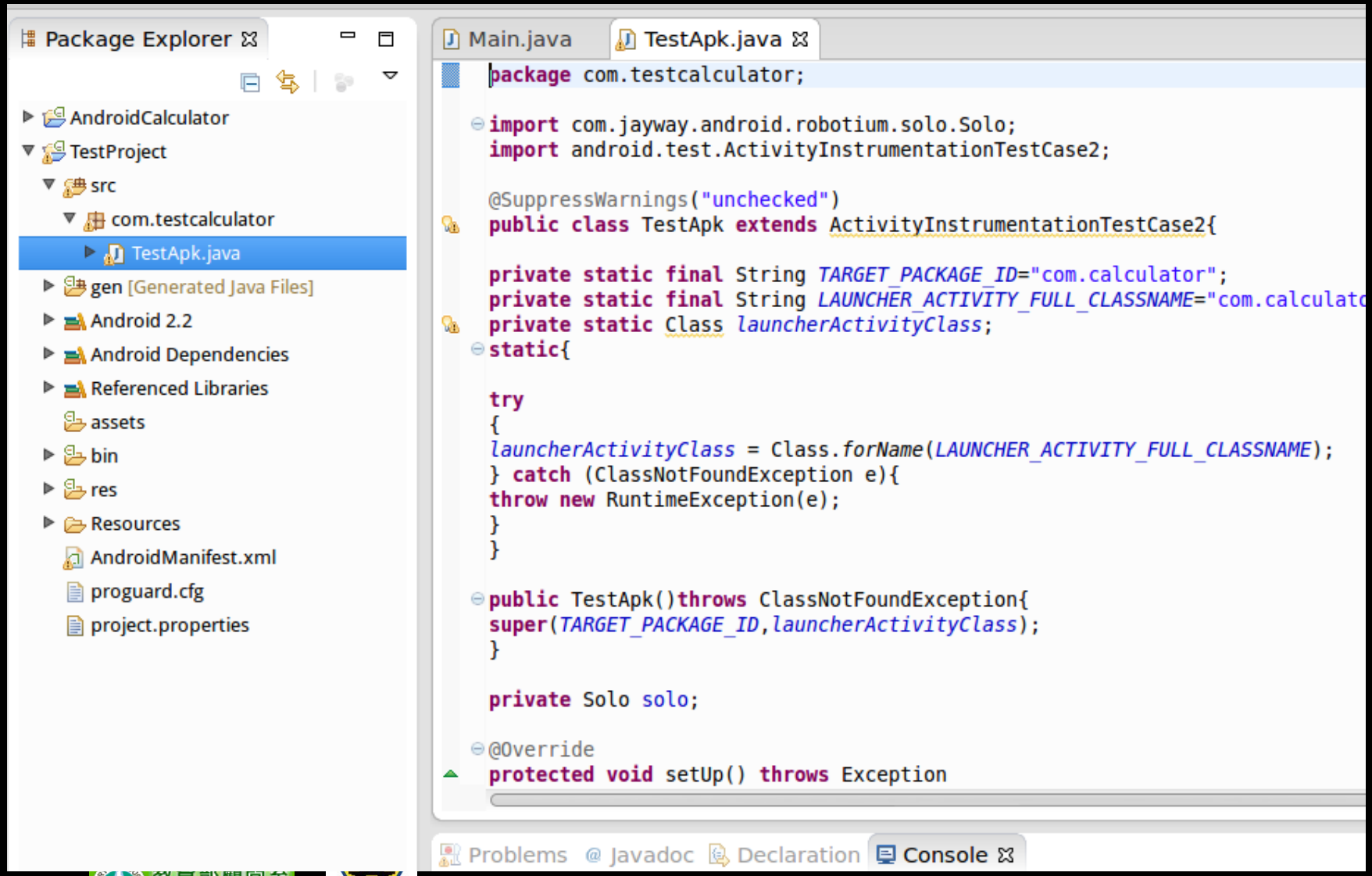
ANDROID TESTING FRAMEWORK – RUN THIS APPLICATION IN VIRTUAL DEVICE

- Right click->Run As->Android Application




5554:test1
12:47
AndroidCalculator
Enter two values and click on Calculate to multiply them.
Enter First Number
Enter Second Number
0.00
Multiply
Basic Controls
Hardware Buttons
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input

- Browse->Desktop->TestProject
- Open TestApk.java

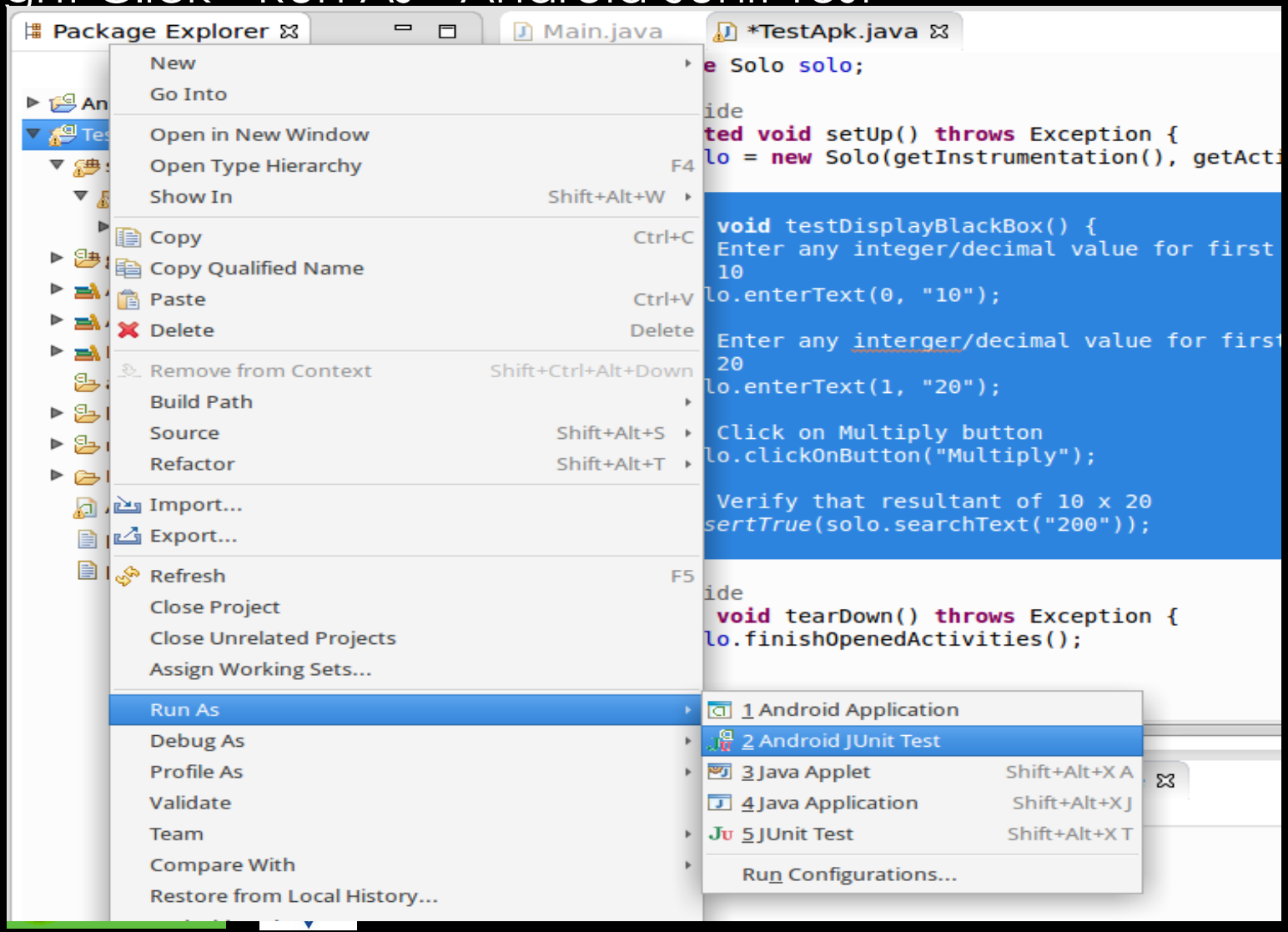


- public void testDisplayBlackBox()

```
public void testDisplayBlackBox() {
    // Enter any integer/decimal value for first editfield, we are writing
    // 10
    solo.enterText(0, "10");

    // Enter any interger/decimal value for first editfield, we are writing
    // 20
    solo.enterText(1, "20");

    // Click on Multiply button
    solo.clickOnButton("Multiply");

    // Verify that resultant of 10 x 20
    assertTrue(solo.searchText("200"));
}
```

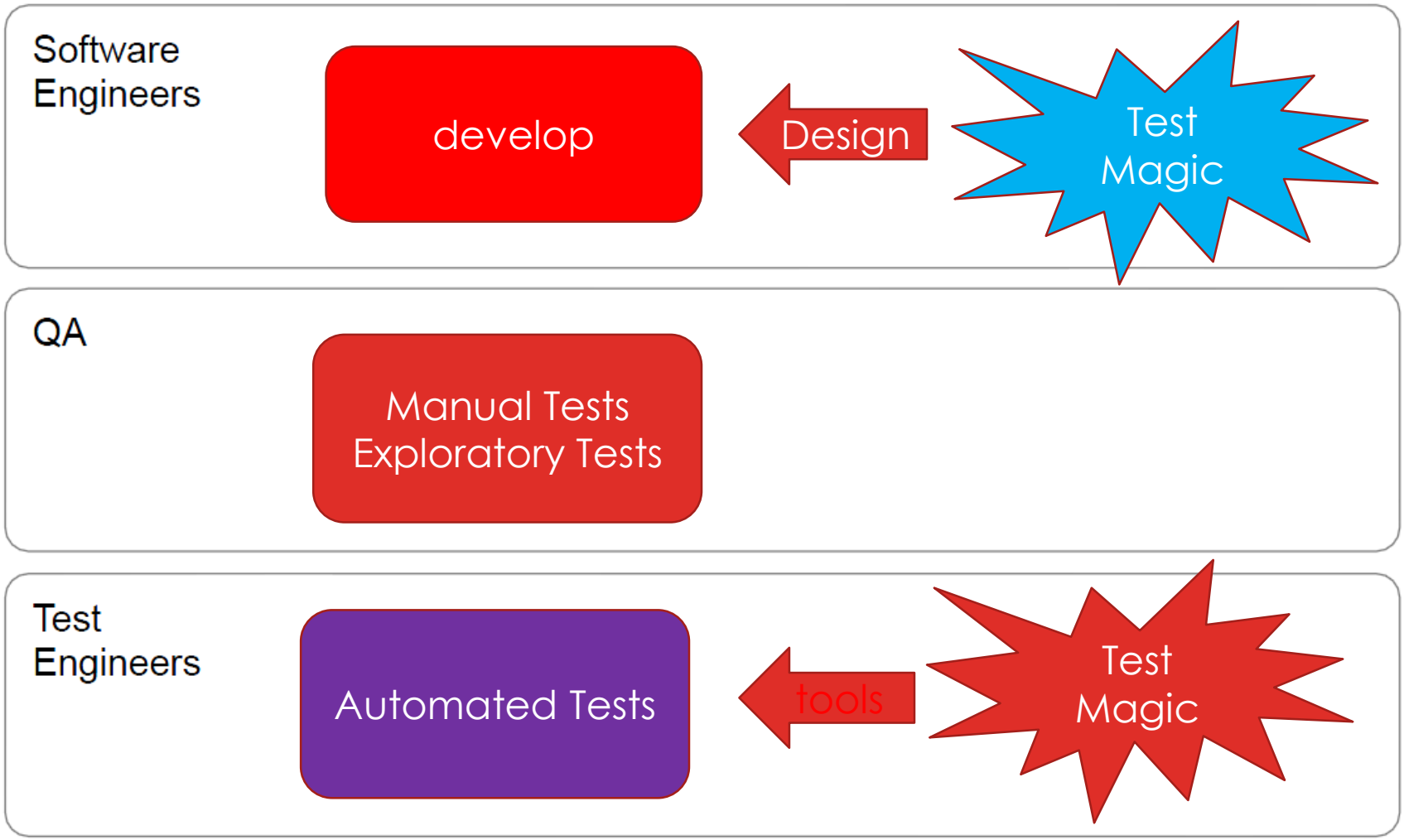
RUN ANDROID JUNIT TEST

- Right Click->Run As->Android Junit Test

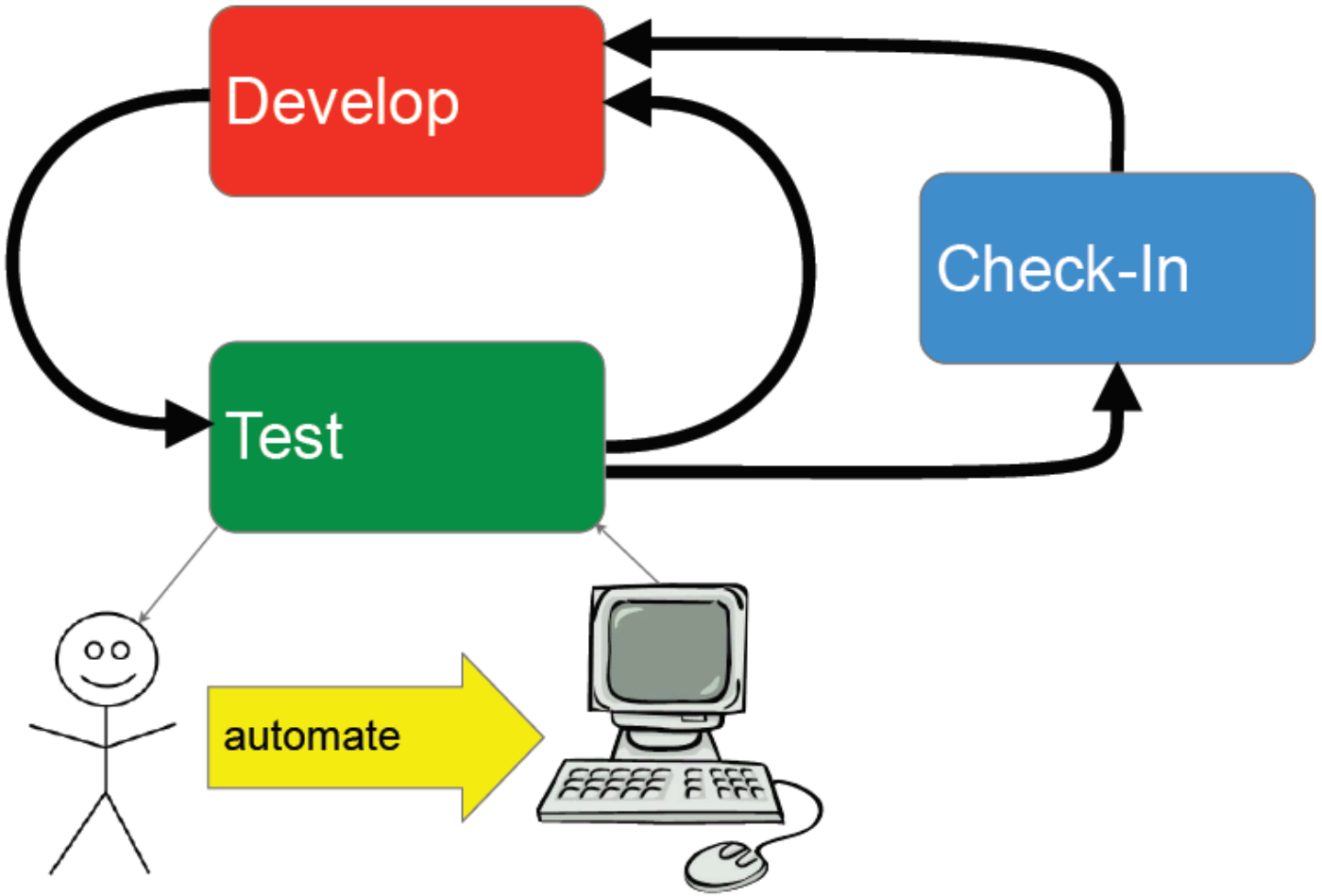


TEST AUTOMATION WITH CONTINUOUS INTEGRATION

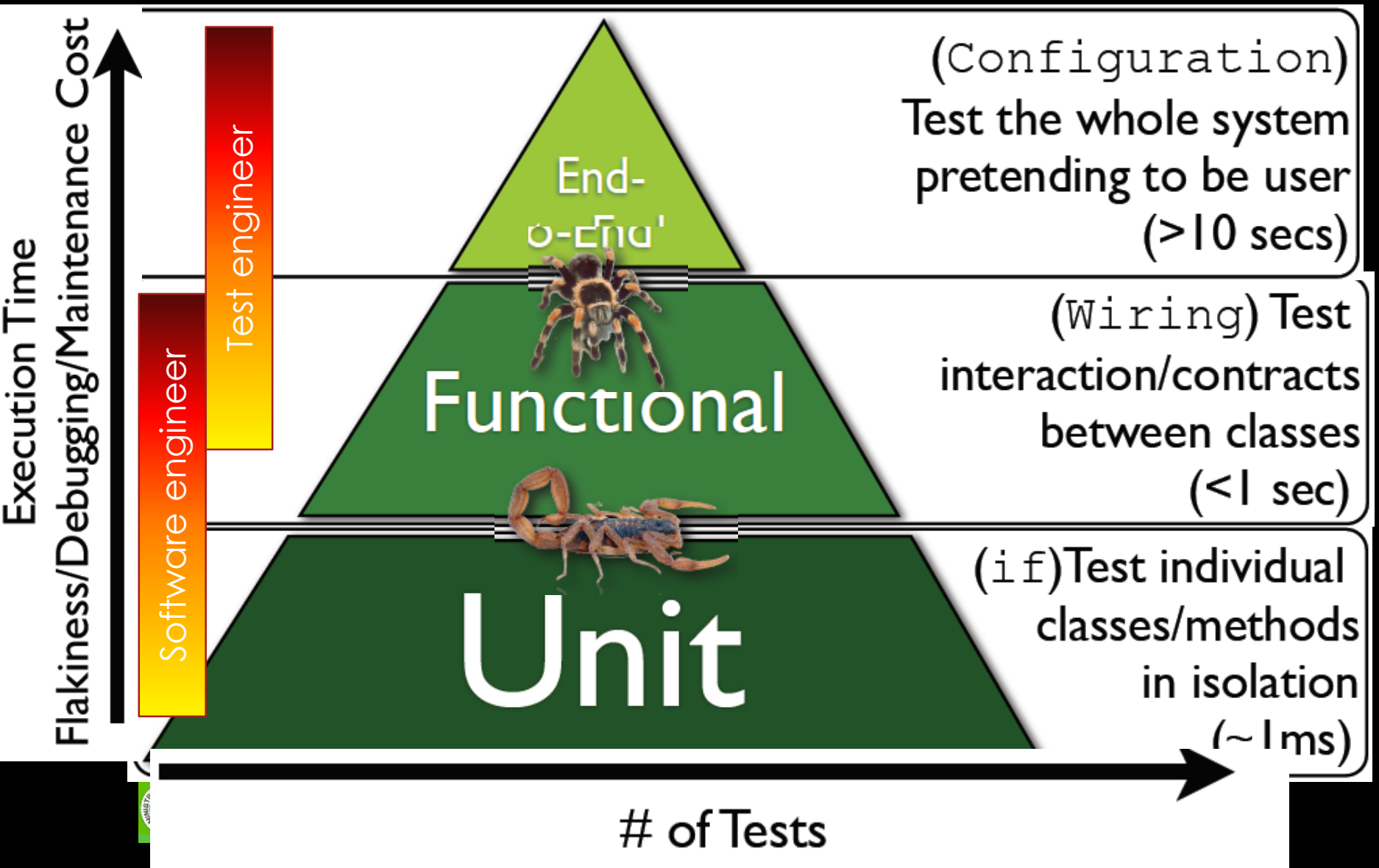
Development Model



Micro Development Model



DIFFERENT KIND OF TESTING



THE XUNIT FRAMEWORKS

- Original was for SmallTalk
  - Kent Beck and Erich Gamma
- Ported to Various languages and platforms
  - JUnit, CppUnit, DUnit, VBUnit, RUnit, PyUnit, Sunit, HtmlUnit, …
  - Good list at www.xprogramming.com
- Standard test architecture

WRITING UNIT TESTS

- **1. Write a test**

  Test-driven development always begins with writing a test.

- **2. Run all tests and see the new one fail**

  This validates that the test harness is working correctly and that the new test does not mistakenly pass without requiring any new code.

- **3. Write some code**

  The next step is to write some code that will pass the test. The new code written at this stage will not be perfect and may, for example, pass the test in an inelegant way. That is acceptable as later steps will improve and hone it. It is important that the code written is *only* designed to pass the test, no further (and therefore untested) functionality must be predicted and 'allowed for' at any stage.

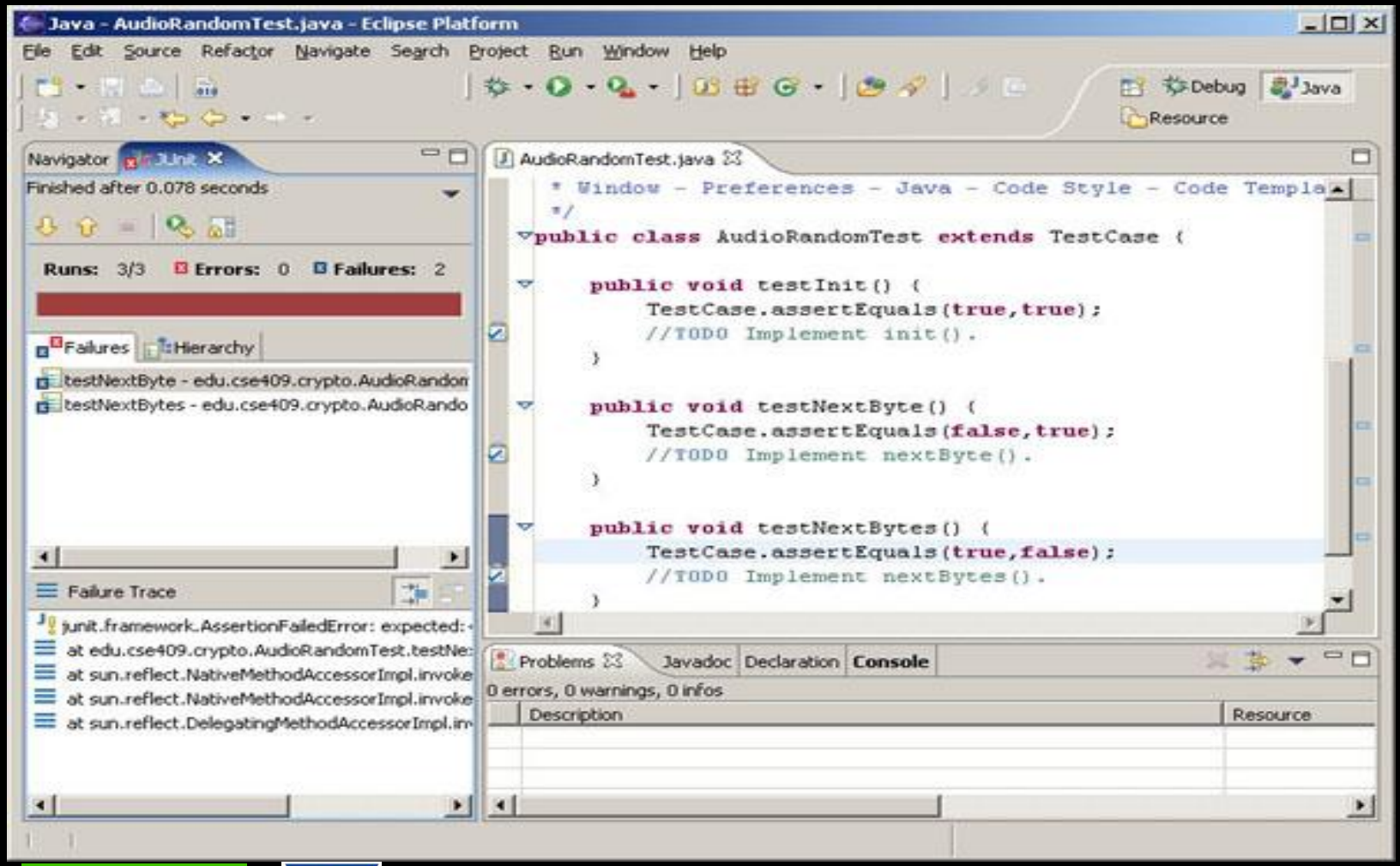
- **4. Run the automated tests and see them succeed**

  If all test cases now pass, the programmer can be confident that the code meets all the tested requirements. This is a good point from which to begin the final step of the cycle.

- **5. Refactor to remove duplication**

  Now the code can be cleaned up as necessary. By re-running the test cases the developer can be confident that refactoring is not damaging any existing functionality. The concept of removing duplication is an important aspect of

JUNIT FOR ECLIPSE



XUNIT TEST SAMPLE CODE

```
public class MoneyTest extends TestCase {
//…
   public void testSimpleAdd() {

       Money m12CHF= new Money(12, "CHF");  // (1)
       Money m14CHF= new Money(14, "CHF");
       Money expected= new Money(26, "CHF");
       Money result=  m12CHF.add(m14CHF);    // (2)

       Assert.assertTrue(expected.equals(result));     // (3)
   }
}
```

(1) Creates the objects we will interact with during the test. This testing context is commonly referred to as a test's *fixture*. All we need for the testSimpleAdd test are some Money objects.

(2) Exercises the objects in the fixture.

(3) Verifies the r

TESTING TASKS TO BE DONE

1. write Test plan (often seen in large software company with formal QA team)
2. write test cases, prepare test data
3. test case management
4. file bugs in bug report system
5. regression testing & Test Automation (with tools)
6. **stress testing/load testing**
7. security testing
8. test as assets.

THE TREND OF SOFTWARE TECHNOLOGY

- Technology Trend
  - web-based (browser-based) and service-based
  - Cloud
  - Connected to Public Internet
  - open to security threats
  - The number of users can be unexpected.
- Software Testing Trend
  - Cloud as a testing service
  - Stress Testing/Load testing
  - Security Testing


WELCOME TO THE
CLOUD ERA
COME
WELCOME

STRESS TESTING

教育部顧問室
Ministry of Education
Advisory Office

THE PREPARATION OF A STRESS TESTING

Test Plan (capture a scenario)

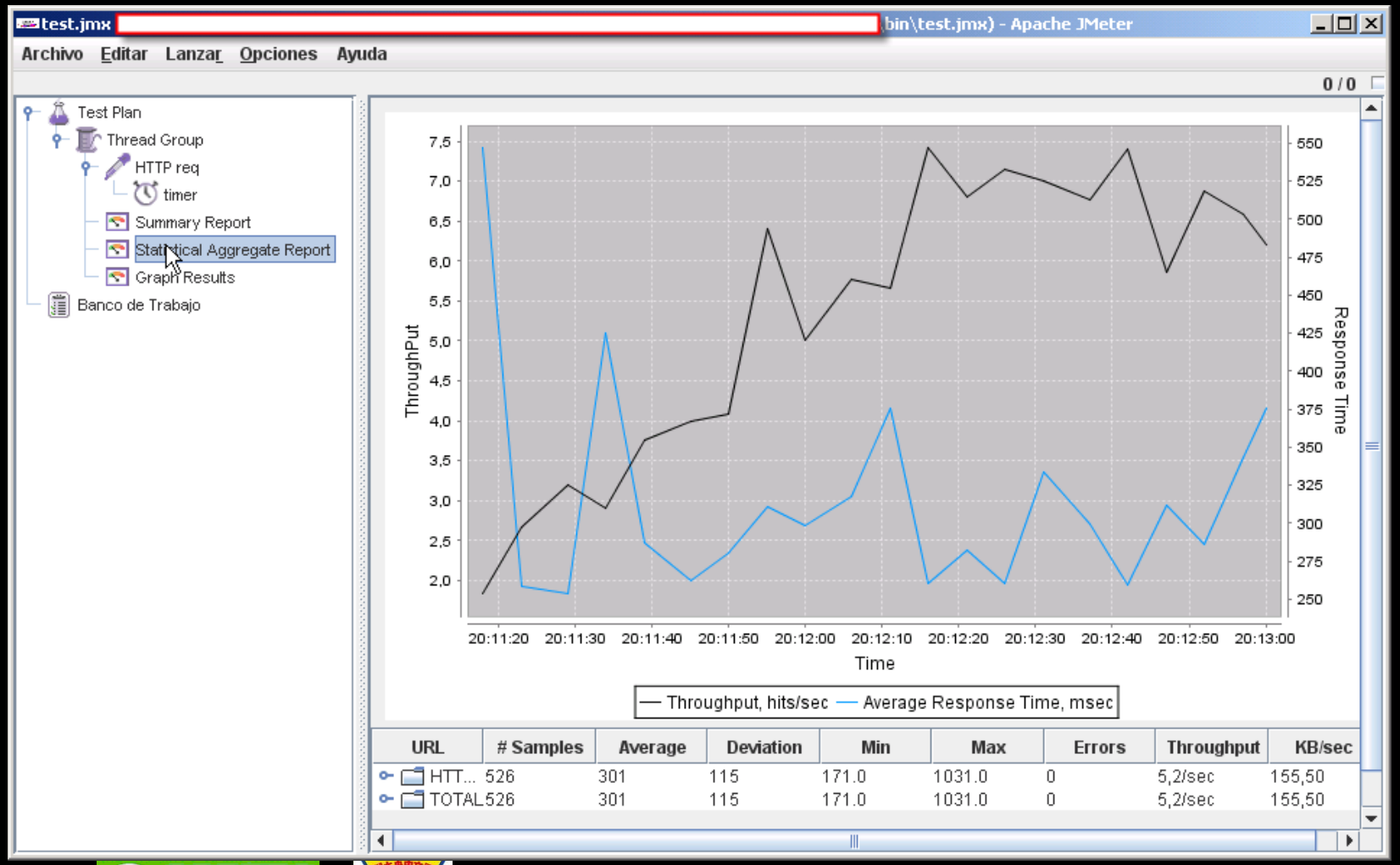
http requests

Web Server

Thread Groups

statistical data

STRESS TESTING REPORT



教育部顧問室
Ministry of Education
Advisory Office

LOAD TESTING VS. STRESS TESTING

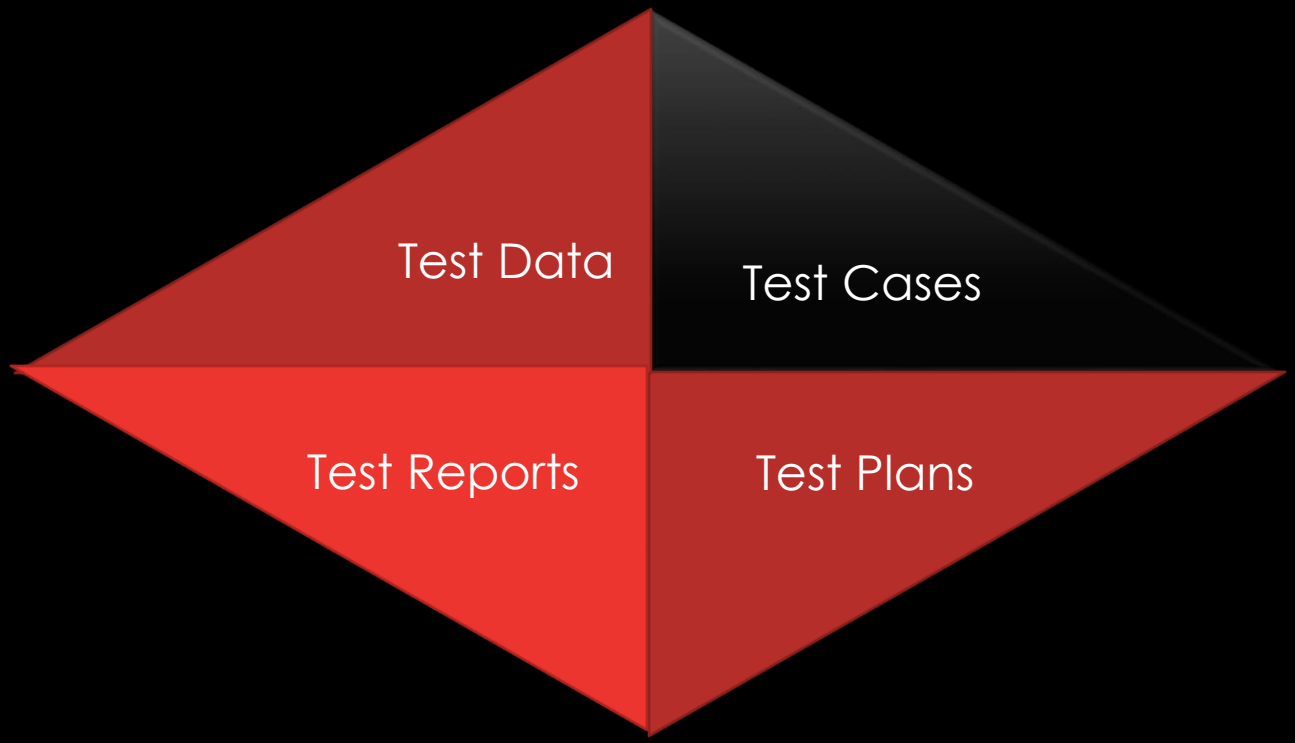
- Load Testing – test the system with regular loads to see if it can behave regularly within specs
- Stress Testing – test the system with loads beyond expected throughput to see how system reacts

TESTING TASKS TO BE DONE

1. write Test plan (often seen in large software company with formal QA team)
2. write test cases, prepare test data
3. test case management
4. file bugs in bug report system
5. regression testing & Test Automation (with tools)
6. stress testing/load testing
7. security testing
8. **test as assets.**

TEST AS ASSET

- Most people think source code is the unique asset of a software company
- Actually, tests are another important asset, particularly when a company invest so many resources.
- example of mobile phone testing out sourcing. (50000 test cases)



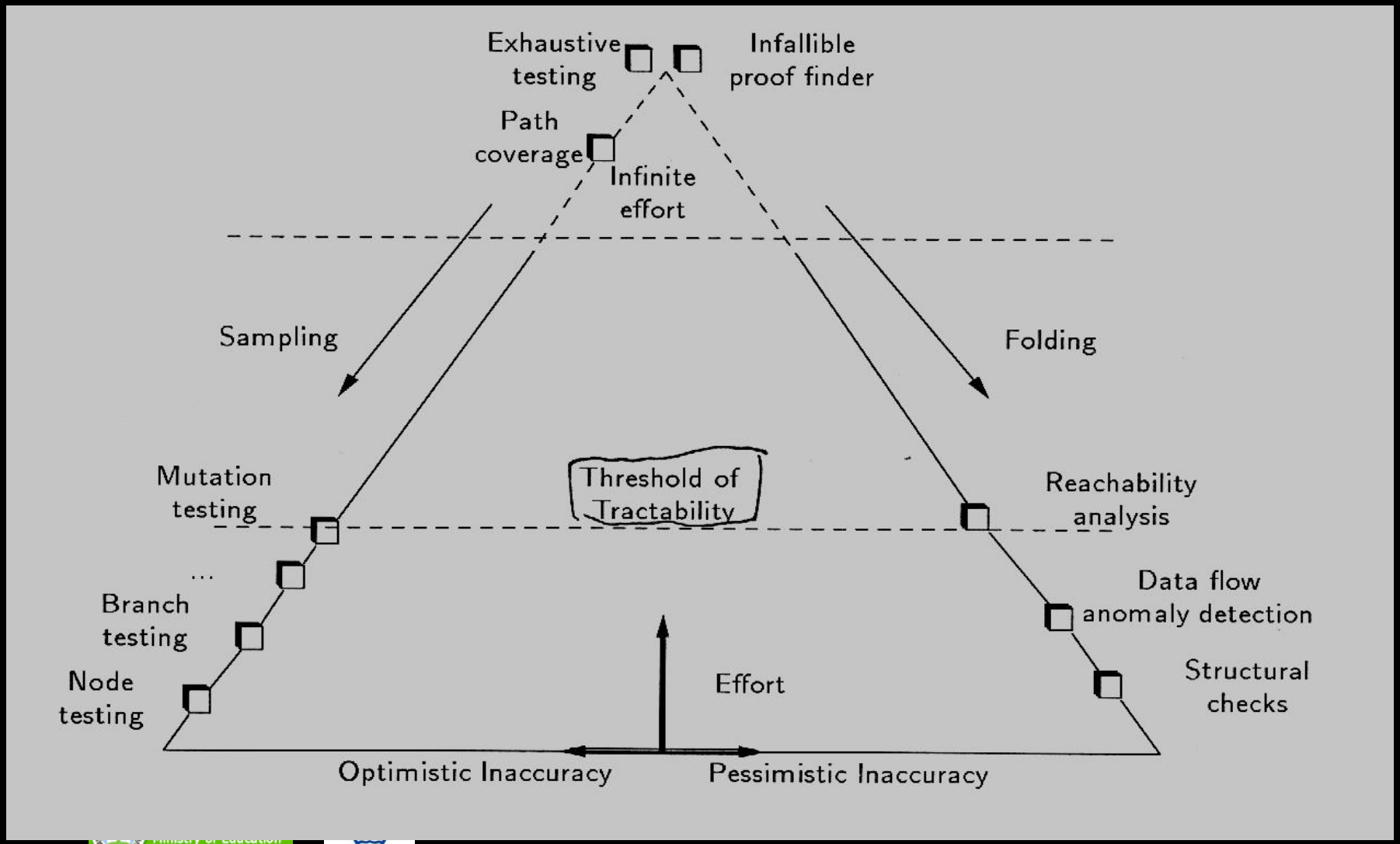
TESTING THEORY

If I have time.

SOFTWARE DEFECTS DETECTION TECHNIQUES CLASSIFICATION

- optimistic
  - testing
  - using optimistic approach can only show how good your code may be. It can not show the absence of faults
- pessimistic
  - static analyzer, code review
  - mostly want to show the absence of faults
  - may report spurious results
    - compiler complains about usage of noninitialized variable

OPTIMISTIC V.S.



COMMENTS

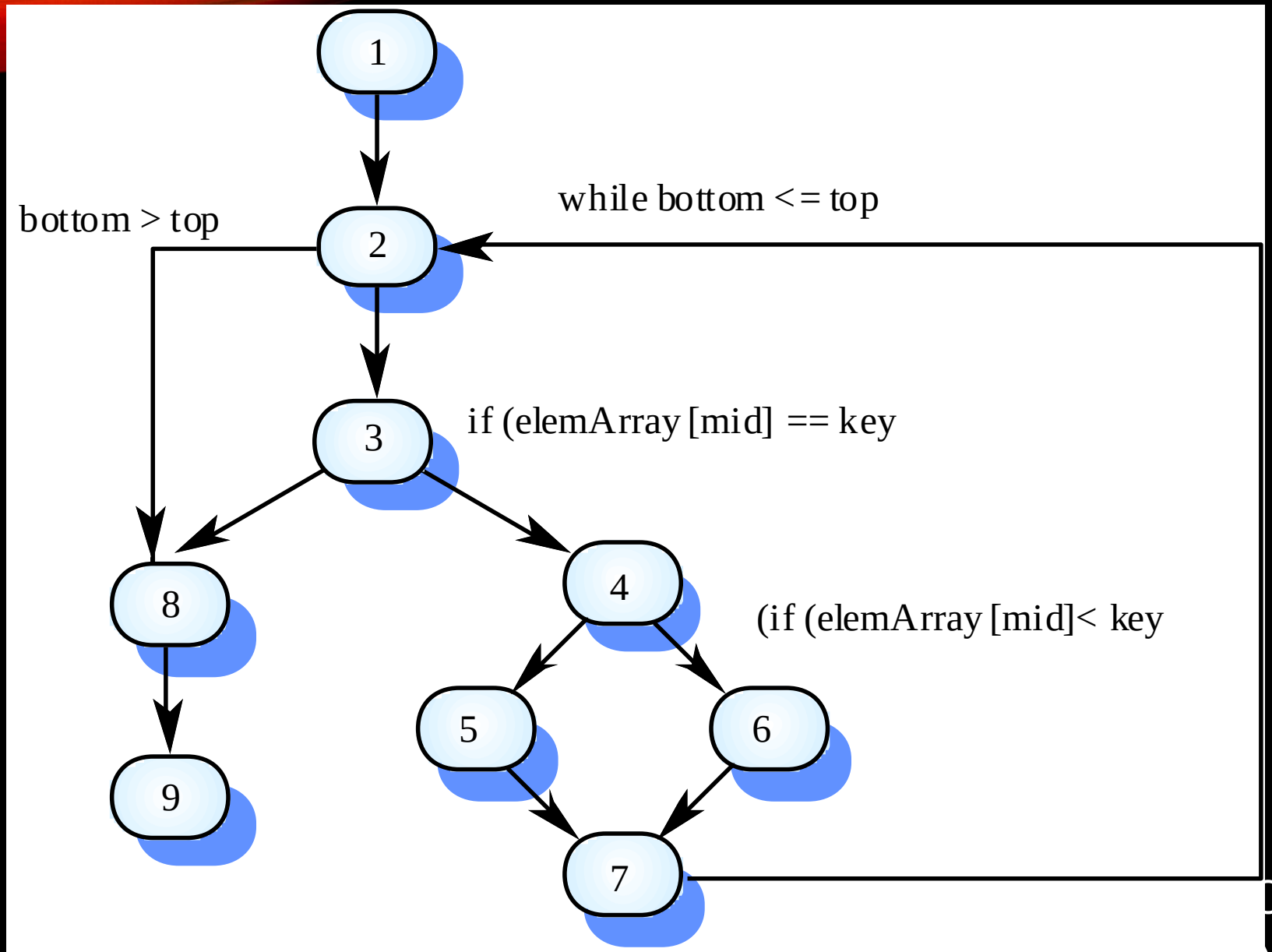
- In principle, testing cannot show the absence of faults, so it is impossible to prove a program is correct by testing
- E.g., you can not prove a sorting algorithm is correct totally by testing. To prove a sorting program is correct exhaustively, you need to test N! cases. If N = 100, it intractable.
- Using exhaustive testing to obtain correctness of a program is often infeasible for common problem.
- Question: How Sorting algorithms in Algorithm Course are proved correct?

COMMENTS

- Proof is difficult. You need to show that your solution works for all cases. Even the cases are N! and N can be arbitrary
- Testing only works for a comparatively small set of input domain under reasonable cost and resources.

CRITERIA OF CONFIDENCE

- Test Coverage (from weak to strong)
  - statement coverage – each statement is at least executed once.
  - branch coverage – each branch should be exercised at least once
  - path coverage – each path should be at least exercised at least once


1
while bottom <= top
bottom > top
2
3
if (elemArray [mid] == key
8
4
(if (elemArray [mid]< key
5
6
9
7

OW
GRAPH

THE WEAKEST CRITERIA

- Statement coverage
  1 test case to run 1 2 3 8 9

  1 test case to run 1 2 4 6 7 2

  1 test case to run 1 2 4 5 7 2 8
- You only need 3 test cases to have each statement at least executed once.

BRANCH COVERAGE

- each branch can produce two choices, every choice combinations should all be exercised
- there are 3 branches, so at most 2 * 2 * 2 branching choices should be exercised if loop is not considered.
- So at least you need to find test cases to meet this criteria

  1 2 8 9

  1 2 3 8 9

  1 2 3 4 5 7 2 8 9

  1 2 3 4 5 7 2 3 8 9

  1 2 3 4 6 7 2 8 9

  1 2 3 4 6 7 2 3 8 9

- In the example, there are less than 8 branching choices because it is not a complete tree

PATH COVERAGE

- The highest coverage
- Try to find test cases which can cover all the paths
- equal to exhaustive testing if you want to cover them all, which is impossible
- You need to find infinite test cases which have finite (if the program must stop) or infinite length (if the program may run forever)

  1 2 8 9 (finite length)

  1 2 3 8 9 (finite length)

  1 2 3 4 5 7 2 8 9

  1 2 3 4 5 7 2 3 4 5 7 2 8 9

  1 2 3 4 5 7 2 3 4 6 7 2 3 8 9

  1 2 (3 4 5 7 2) * 8 9 (finite length but very long sequence)

  1 2 (3 4 5 7 2 | 3 4 6 7 2) * 8 9 (finite length with permutations)

  1 2 (3 4 5 7 2) ω 8 9

  .....

  .....
- where * is finite iteration in automata theory
- where ω is infinite iteration in automata theory

PATH COVERAGE

- in a program which can run for ever, full path testing is equal to exhaustive testing
- it is infeasible in practice
- So, we must discover more practical path coverage criteria.

INDEPENDENT PATHS (A WEAKER CRITERIA)

- 1, 2, 3, 8, 9
- 1, 2, 3, 4, 6, 7, 2
- 1, 2, 3, 4, 5, 7, 2
- 1, 2, 3, 4, 6, 7, 2, 8, 9
- each test case should explore statements that are not explored by the previous test cases to reduce redundant test cases
- A dynamic program analyser may be used to check that paths have been executed
- In practice, this is often difficult too

TEST COVERAGE IN REAL WORLD

- Since different coverage criteria involve different degree of cost and time
- Often statement coverage is the lower bound.
- Branching coverage or path coverage can be applied to important components
- Test coverage testing is widely applied in software industry. Although coverage rate mean nothing to program correctness. It is only an index to show if testing has been done adequately. Most people accept that index is closely related to software quality since test cases are executed.

TEST COVERAGE

- IMPORTANT

  If you have 100% statement coverage, it does not mean your program is correct.

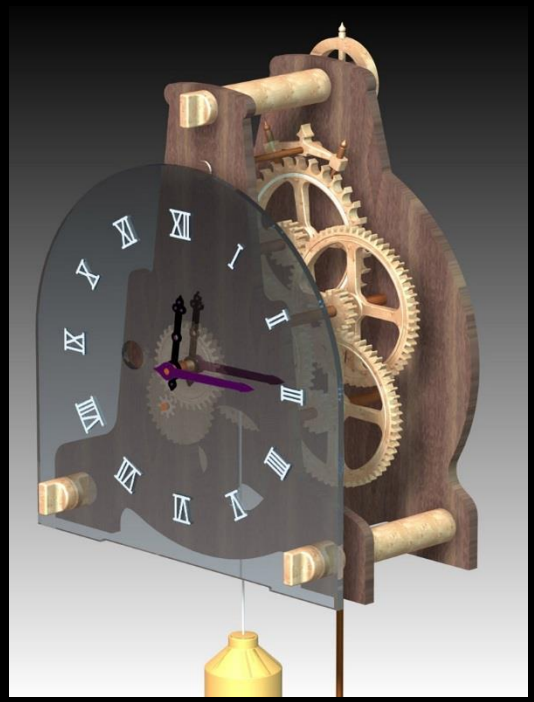
- Sorting algorithm is an example
  - you can easy derive only few test cases to have 100% coverage of your program
  - But to prove your program is correct is totally a different scale of the program

HOW TO PROCEED TEST COVERAGE TESTING?

- You need to have source code to know the coverage.
- You need to have tool to display the coverage rate
- Each test case, you need to have someone (or in very few cases, tools) to judge the correctness of program behaviors.

WHITE BOX TESTING

Test cases



output

You test an equipment by observing the internal behaviors of the equipment also known as structural testing (text book)

TEST COVERAGE

- Test coverage must be done with white-box testing. The tester must have source code for testing
- Effective test cases must be derived from source code, which require programming skills.
- So, test coverage can be done by programmers or 3rd party testing company.
- In USA, Europe, there are many 3rd party testing companies.
- In Taiwan, never heard of it. Maybe you should start a new one.

BLACK BOX TESTING

Test Cases

Outputs

教育部顧問室
Ministry of Education
Advisory Office

BLACK BOX TESTING

- A testing process proceed without knowing the internal behaviors of the system.
- Often done by QA (Testers)
- QA only interact with system by
  - feeding test cases by user interfaces
  - feeding test cases by files
  - feeding test cases by networking traffic
  - feeding test cases by consoles

BLACKBOX TESTING AND COVERAGE

- In principle, Black Box testing can still combine with test coverage tools, if source code is available.
- Test coverage tools can be used to show the index of "how good the test cases are derived by the QAs."

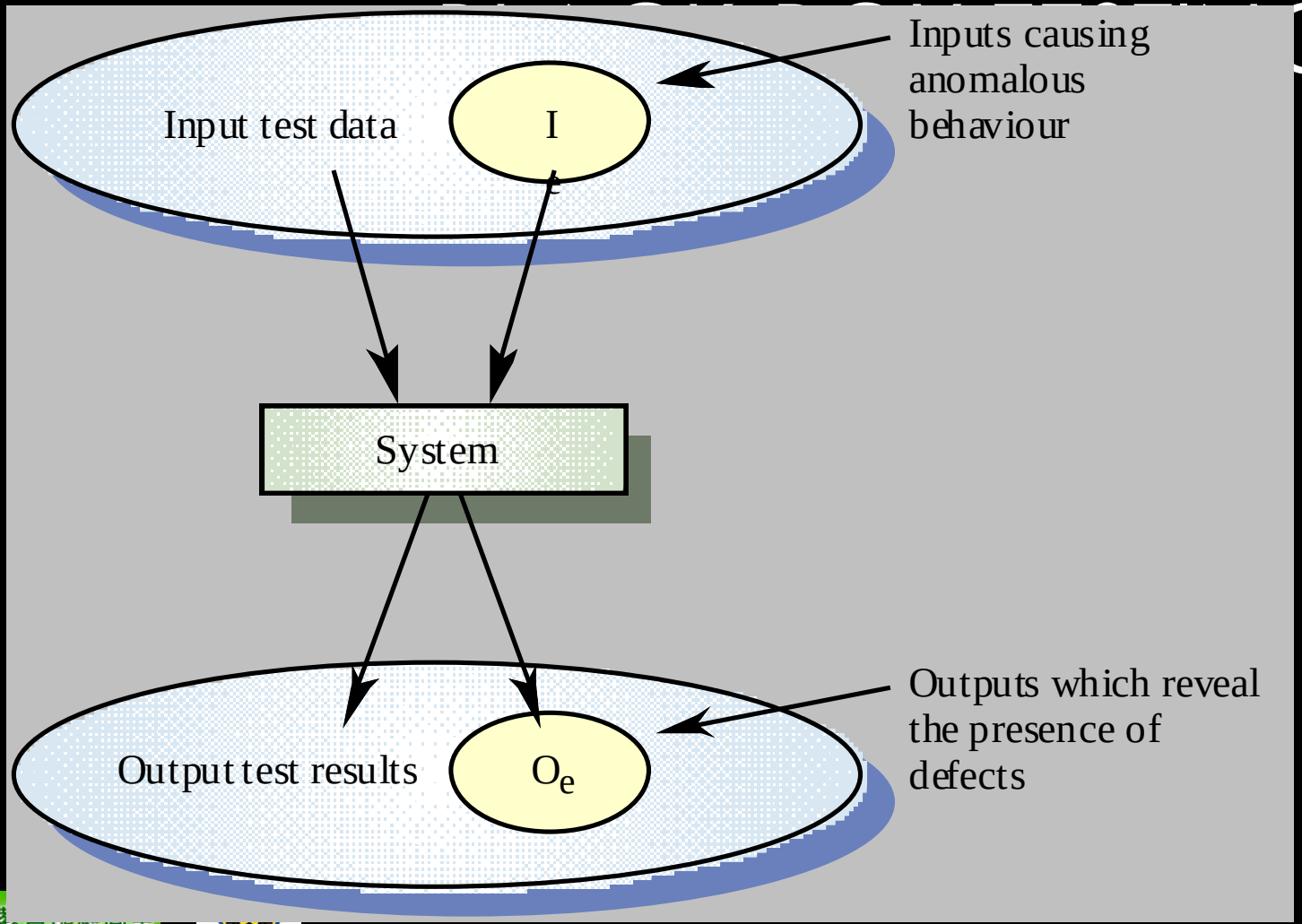
BLACKBOX TESTING

- In practice, QAs use blackbox testing to assure Software quality after unit testing and integration testing. They take over from developers
- They focus on checking

whether the program is conformed to specs.

BLACKBOX (ALPHA TESTING) TESTING

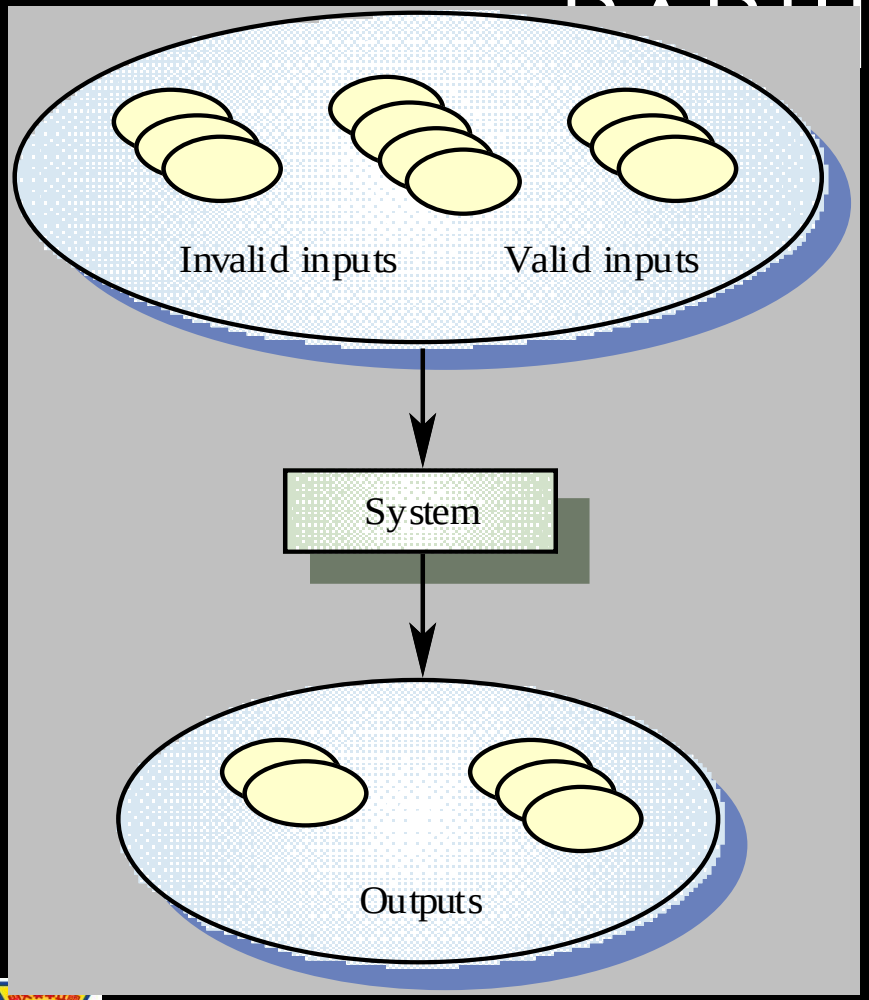
- QAs test program by specs.
- In common cases, specs may describe
  - correct behaviors
  - incorrect behaviors that should be avoid.
- QA's job goal
  - to show the presence of software errors as much as possible
  - test the software from a user's perspective
  - test all positive, negative, and boundary cases



EQUIVALENCE PARTITIONING (PARTITION TESTING)

- Input data and output results often fall into different classes where all members of a class are related
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member
- Test cases should be chosen from each partition

EQUIVALENCE PARTITIONING



EQUIVALENCE PARTITIONING WITH BOUNDARY CASES

- Partition system inputs and outputs into 'equivalence sets'
  - If input is a 5-digit integer between 10,000 and 99,999, equivalence partitions are <10,000, 10,000-99, 999 and > 10, 000
- Choose test cases at the boundary of these sets
  - 00000, 09999, 10000, 99999, 10001

EQUIVALENCE PARTITIONS