CSC 221 – Introduction to Software Engineering

debugging, bug finding and bug avoidance

Part 3

Alan Dix

www.hcibook.com/alan/teaching/CSC221/

outline

- part 1 – general issues and heuristics
- part 2 – the system as it is
  - understand and document
- part 3 – locating and fixing bugs
- part 4 – bug engineering
  - design to expose, avoid and recover
  - including fail-fast programming

debugging – part 3
locating and fixing bugs

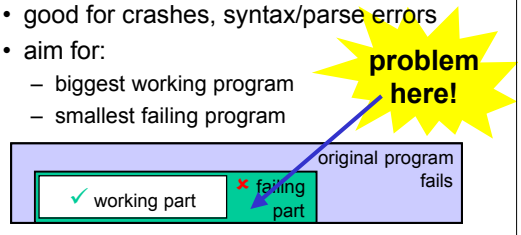
isolate error
don't assume anything!
secondary errors
using tools

locating bugs

- instrument program (print/log)
- isolate error - find subsystem
- simplify program/system

isolate error
binary chop

- comment out parts of program ...
- good for crashes, syntax/parse errors
- aim for:
  - biggest working program
  - smallest failing program



isolate error
use fixed values

- can't comment out beginning
- substitute for calculations/input
- use value from logging

```
val  = complexCalculation();
suspectCalculation(val);
```

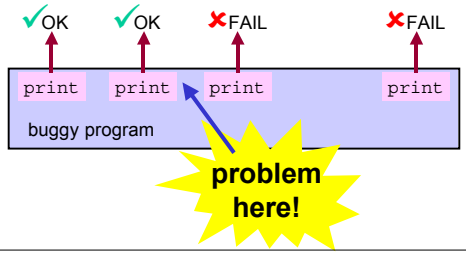
isolate error
use fixed values

- can't comment out beginning
- substitute for calculations/input
- use value from logging

```
// val  = complexCalculation();
// print(val);
val = fixed_value;
suspectCalculation(val);
```

isolate error
binary chop for internal state

- print out internal values ...



isolate error
unfold methods/functions

- problem code between two methods

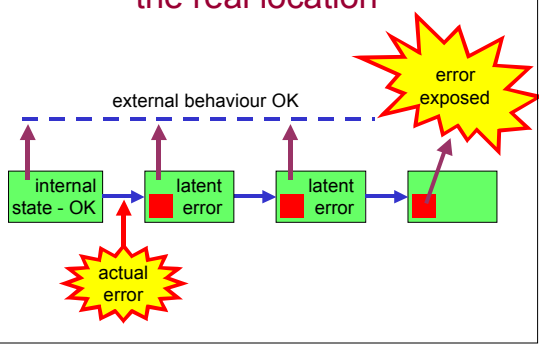
```
Aclas obj;
obj.add(3);
...
class Aclas {
   private int x;
   void add(int y) { x = x + y; }
}
```

isolate error
unfold methods/functions

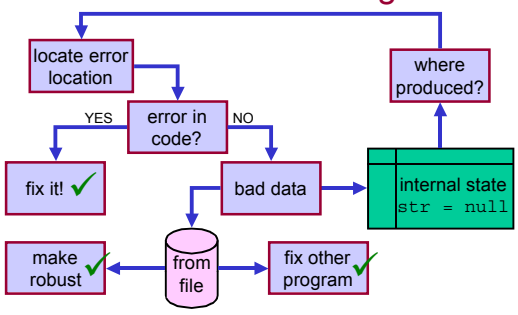
- substitute code from inner method

```
Aclas obj;
// obj.add(3);
obj.x = obj.x + 3; // param y = 3
             // unfold from Aclas
...
class Aclas {
  public int x;
             // remember to change back!
  ...
```

the real location



the real location
find the real bug



don't assume anything!

when all else is impossible ...

- question everything
- check the obvious

```
print("x="+x);
y = x;
print("x="+x+", y="+y);
```

don't assume anything!
capture errors

```
print("x="+x);
f(x);
...
```

```
void f(x) {
    print("f("+x+")");
    ...
}
```

- is it what you think?
  - method / object / function / file
- may not be:
  - classpath, linking, inheritance, overload, file upload directory, caching, compile fail

don't assume anything!
type conversion

```
int secs_yr = 365*24*60*60;
long ms_yr = 1000 * secs_yr;
```

- overflow before conversion
  - also beware integer divide
- C++ more complex conversions
- Java String concatenation

```
print("x+1="+x+1);
```

don't assume anything!
platform/language

- it may not be your fault
  - but be really sure!
  - simplify minimal case
- differences between platforms:
  - Java AWT event order, NullPointerException
- bugs / omissions
  - blank/stub bodies for some Java methods
- JIT / optimisers

secondary errors
debugging causes bugs!

- if capture

```
if ( x != null )
    System.out.println("x is "+x);
    x.val ++;
```

- null print

```
System.out.println("val = "+x.val);
if ( x != null )
    x.val ++;
```

secondary errors - 2

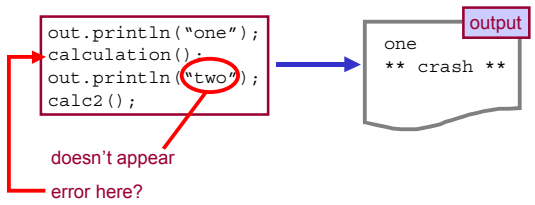
- changes in compiler optimisation

```
int x = obj.z;
print(x);  // forces calculation
// x never used again
```

- changes in memory layout
  - C/C++ array overrun bugs etc.
  - not in Java, but threading changes

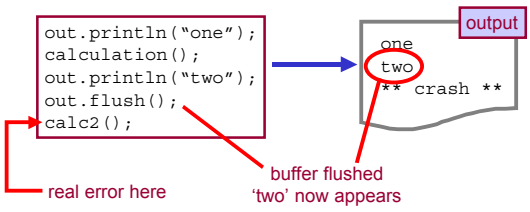
secondary errors
flushing

- serious blank screen / core dump ...
  - insert prints – see when they stop



secondary errors
flushing

- internal buffer for error log output
  - need to flush in logging code



tools
use your compiler

- thin classes

```
class salary { public int amount; }
```

- rename methods/variables

```
int xgetVal() { ... }
```

  - unexpected use ➔ compile errors
- use private/protected
- dynamic errors ➔ static errors

tools
use your debugger

- get to know it!
  - breakpoints, conditional breaks,
  - bugs in the debugger!
- debugger friendly code

```
return complex_expr;
```

```
int x = complex_expr;
return x;
```

... and when you've fixed it

- do you understand what went wrong?
- do you know why the fix works?
- are there similar bugs?
- can you prevent similar bugs in future?