CSC 221 – Introduction to Software Engineering

debugging, bug finding and bug avoidance

Part 2

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 2
the system as it is

understand your system
make assumptions explicit
document interactions

understand your system

- collaborative walkthrough
- personal walkthrough
  - what do you think it does?
- print statements
  - what it really does!
- make assumptions explicit
- make interactions explicit

understand your system
when in doubt

- document

```
s = s.substring(idx+1);
    // +1 to skip current char
```

- bracket

```
d = b3<<24 + b2<<16 + b1<<8 + b0;
```

```
d = (b3<<24) + (b2<<16) + (b1<<8) + b0;
```

understand your system
bugs are where you'd expect

- what was difficult to code?
  look there!
- where would you expect errors?
  look there!

understand your system
bugs are where you'd expect

- some years ago ...
- testing third party floating point library
  - ❶ try simple case 1+2=3 – OK ✓
  - ❷ try cases to force 16 bit carry – FAIL ✘
- soon after - problem with Microsoft C
  - 32 bit integer subtract
  - carry error ⇒ answer is 65536 out

understand your system
beginnings and endings

- starting up
  - loading data (files exist, in right format?)
  - initialisation (doing it, doing it right, in the right order)
- ending
  - saving data
  - releasing resources
  - closing/flushing files

make assumptions explicit
bad - implicit assumption

```
int len(String str) {
    return str.length();
}
```

◆ fails if `str` is null

make assumptions explicit
better - documented

```
/**
 * @param str - must not be null
 */
int len(String str) {
    return str.length();
}
```

make assumptions explicit
defensive - recover

```
/**
 * @param str - must not be null
 */
int len(String str) {
    if ( str == null ) return 0;
    return str.length();
}
```

make assumptions explicit
diagnostic - log

```
/**
 * @param str - must not be null
 */
int len(String str) {
    if ( str == null ) {
         log("len:  str is null");
         return 0;
    }
    return str.length();
}
```

document interactions

- external interactions (real world effects!)
  - mouse events, robot controls
- sequence information
  - order of events, method calls
- shared data
  - static variables, references to objects
- hidden effects
  - influence on other objects / components

document interactions
external interactions

- say what causes events

  e.g. a component receives a `MOUSE_PRESS` event when and only when the user presses down a mouse button over it
  - N.B. not true of all Java AWT toolkits!
- say what happens in the real world

  e.g. the `move(x,y)` method sends a signal to the robot arm control motor moving it x cm in the X direction and y cm in the Y direction

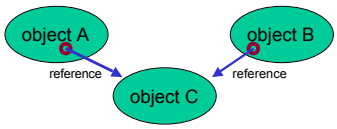
document interactions
sequence information

- define events order explicitly

  e.g. `MOUSE_PRESS` and `MOUSE_RELEASE` events always alternate in the event sequence for any mouse button
  - N.B. not true of all Java AWT toolkits!
- explain restrictions on method calls

  e.g. always call the `xinit()` method before other methods and call the `close()` method last
  - but check inside your code

document interactions
shared data ➔ hidden effects



- can be difficult to describe

  e.g. if you create a TCP/IP socket in Java, what happens to the output part if you close the input?

  answer: it gets closed too

document interactions
scenarios

- typical paths through system
  - a 'story' about the system
  - what happens
    which objects involved
    how they affect data
  - several levels of detail

e.g. see aQtive documentation at:
http://www.aqtive.net/developer/developers-pack/onCue-hiw/onCue-hiw.html