CSE70: Lecture 4

• Today
  – First milestone deadline now!
  – Did you submit your feedback?

• For Today:
  – FLOOT (in the reader)

• Today
  – Yesterday lab
  – Testing
  – Coverage
  – Questions on the midterm?
Full Lifecycle Object Oriented Testing (FLOOT)

• Regression Testing
  – Avoids *regression*
  – Test suites need to be maintained
  – Fundamental for incremental development

• QA
  – Verify compliance with …
  – So what is quality?
Not only the code...

- Check and validate as soon as possible!
- Use Case Scenario Testing
- Prototype Walkthrough
- User Requirement Review
- Model Review and Walkthroughs
  - Reported to catch ?% of design errors (Grady 1992)
- User-Interface Testing (GUI Testing)
Testing the code

- Black-Box Testing
- White/Clear/Glass-Box Testing
- Boundary Value Testing
- Class Testing
- Class-Integration Testing
- Code Inspections
- Coverage and Path Testing
- Inheritance-Regression Testing
- Method Testing
Testing the system

- Function Testing
- Installation Testing
- Operations and Support Testing
- Stress Testing
User Testing

• *Pilot* Testing
  – *Alpha, Beta*

• User Acceptance Testing (UAT)
Finding and reporting defects

- Humphrey (1997)
  - Description
  - Stage in which the defect was found
  - Stage in which the defect was introduced
  - Stage in which the defect was removed
  - Steps to recreate the defect
  - Effort to fix it
  - Solution
  - other…
Coverage

• Exercise statements with testing
• Why check coverage?
  – Only 50-60% of the code exercised (Wiegers 2002)
• Define coverage
  – Control-flow
    • Statement, path, ...
  – Data-flow
    • Define-use, ...
Statement Coverage

• Exercise every statement

```java
void one(int x) {
    if(x==1)
        System.out.println(“One!”);
}

• one(1);
```
void two(int[] x) {
    int i=-1;
    if(x[0]==2) {
        System.out.println("Two!");
        i=0;
    }
    x[i]=1;
}

• int[] x = {2,1}; two(x);
• int[] x = {1,1}; two(x);
Branch Coverage

• Produce every condition outcome

```java
void two(int[] x) {
    int i=-1;
    if(x[0]==2) {
        System.out.println("Two!");
        i=0;
    }
    x[i]=1;
}
```

- int[] x = {2,1}; two(x);
- int[] x = {1,1}; two(x);
void three(int[] x) {
    int i=-1;
    if(x[0]==2) {
        System.out.println("Three!");
        i=0;
    }
    if(x[0]==1)
        i=1;
    x[i]=0;
}
• int[] x = {2,1}; three(x);
• int[] x = {1,1}; three(x);
• int[] x = {3,1}; three(x);
Path Coverage

- Exercise every path

```java
void foo(int[] x) {
    ...;
    condition 1{
        ...;
    }
    condition 2
        ...;
        ...
}
```
Example

void four(int[] x) {
    if(x[0]==2 && x[1]==1)
        x[0]=4;
}

• Statement?
• Branch?
• Path?
Cyclomatic Complexity

• McCabe (1976)
• M = decision statements + 1
• M = e - n + p
• *Simple* code has low complexity
• How does this relate to coverage and testing?
  – As coverage, quantifies the level of effort for testing
  – S ≤ B ≤ M ≤ P
void five(int[] x) {
    if(x[0]>2 && x[1]>1)
        x[0]=5;
    if(x[0]>1)
        x[1]=5;
}

• Statement?
• Branch?
• M?
• Path?

• {3,2}
• {3,2}, {0,1}, {3,0}
• 4
• {0,1}, {2,1}, {3,0},
  {3,2}
Data-flow testing

• Beizer (1990)
• Data has a state, the state determines the errors
• Data is:
  – Defined
  – Used
  – Killed
  – ...

Data-flow analysis

• Warnings!
  – Defined-Defined
  – Defined-Killed

• Coverage and Testing
  – Defined-used

\[
\begin{align*}
x & = 3 & \text{if}\(y\) \\
if(y) & \quad y = x; & \text{else} \\
\text{else} & \quad w = x; \\
\end{align*}
\]
Final remarks on testing

• Testing takes a considerable amount of time

• There is no perfect code
  – Errors often come from lack of/incorrect information
  – Complexity increases the chances of error

• 80/20 rule (Boehm 1987)
  – 20% of the code contributes 80% of the errors
Testing for the project

• Base code
  – LOC given
    • 794 functional code, 443 tests
  – Statement Coverage
    • IRCClient 73%
    • IRCMessage 88.5%
    • IRCWindow ?
  – Total tests
    • 37 tests

• TDD
Summary

• Testing
  – Design and code
  – Developer and user

• Coverage
  – Control-flow
  – Data-flow

• References
  – McCabe (1976): *Design Complexity Measurement and Testing*
  – Beizer (1990): *Software Testing Techniques*
In the midterm

**Everything seen so far**
- Serial, iterative, and incremental processes
- XP (4 Values and 13 practices)

**Pair Programming**
- Benefits

**TDD**
- TFD + Refactoring
- Benefits

**Testing**
- While writing and systematically
- FLOOT
- Coverage
  - Control-flow and data-flow
  - McCabe

**Tools**
- Management
  - Trac (milestones, tickets, ...)
- Version Control
  - Subversion (SVN)
    - make and observe changes
- Unit Testing
  - JUnit
- Mocking
  - JMocks
- Automation
  - Ant

**IRC**
- Raw format of messages
  - JOIN, PART
  - PRIVMSG, NOTICE
  - ...

**Others:**
- YAGNI, BDUF