Tutorial 5
More on Software Quality Measurements
How to measure performance?
How to measure complexity?

Last lecture...
On Software Quality Measurements

- We explained the basics of software measurements and metrics
- We gave the metrics related to some quality attributes (-illities)
- We showed performance/complexity measurement results with a toy example

Today…

http://www.cs.toronto.edu/~yijun/ece450h/handouts/tools

1. **Performance** is broken down into Time and Space performance
   *How to measure performance?*

2. **Complexity** is the major reason for low Understandability, Testability, Maintainability
   *How to measure complexity?*

Some examples in Eclipse…

1. Measuring Performance
   1. **System performance**
      - Windows Task Manager
      - Linux “top” command, /proc/cpuinfo
   2. **Application performance**
      1. **Time metrics:**
         - clockticks, # instructions, #cache(TLB) misses
         - **Timing:** /bin/time
         - Profiler: gprof, java –prof, HPjmeter.jar
         - **More comprehensive tools**
           - Simulators: cache simulator/visualizer
           - Hardware performance counters (PCL, perfmon)
           - Intel VTune
      2. **Space metrics:**
         - memory size, network traffic
         - Borland optimizet
Time Performance

- Tool: /bin/time
  werewolf~:/usr/bin/time
  command [arg...]

- Example:
  werewolf~:/software/axis-1_1:/usr/bin/time client.sh

IBM: Armonk, NY
1.48user 0.07system 0:01.59elapsed 97%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (2527major+2588minor)pagefaults 0swaps

- What are User time, CPU time, System Time?
- How to know where the time is spent?

Profiler

- Profiler instruments the code by book-keeping instructions. When the program runs, these instructions can be used to tell
  – Which functions are called the most
  – How is time distributed among different functions?
  – From these data, one can pinpoint the bottleneck of the execution time
- Profiler usually comes together with Compilers
- Many different profilers:
  – For GNUCC (C/C++/Java/Fortran/Ada), use gprof
  – For Java, use "java -prof"
  – HPjmeter.jar (see handouts/tools)

Profiler Case Study - Hello.c

1. #include<stdio.h>
2. void hibernate(void)
3. {
4.   long i;
5.   for(i = 0; i<1000000; i=i+1)
6.   {
7.     printf(" A long hibernate...\n");
8.   }
9. }
10. void nap(void)
11. {
12.   printf(" Just a short nap!\n");
13. }
14. int main()
15. {
16.   printf("Take a nap!\n");
17.   nap();
18.   printf("Go to hibernate!\n");
19.   hibernate();
20. }

Gprof

- gcc –p –pg hello.c
  (gmon.out is generated)
- gprof a.out

<table>
<thead>
<tr>
<th>index</th>
<th>% time</th>
<th>self</th>
<th>children</th>
<th>called</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>100.0</td>
<td>0.02</td>
<td>0.00</td>
<td>1</td>
<td>main [2]</td>
</tr>
<tr>
<td>[2]</td>
<td>100.0</td>
<td>0.02</td>
<td>0.00</td>
<td>1</td>
<td>hibernate [1]</td>
</tr>
<tr>
<td>[3]</td>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
<td>nap [3]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Spring 2005 ECE450H1S Software Engineering II
Profiler Case Study - Hello.java

```java
public class Hello {
    static private void hibernate() {
        long i;
        for (i = 0; i < 100; i++) {
            System.out.println("A long hibernate...");
        }
    }

    static private void nap() {
        long i;
        System.out.println("Just a nap...");
    }

    public static void main(String args[]) {
        System.out.println("Take a nap!");
        Hello.nap();
        System.out.println("Go to hibernate!");
        Hello.hibernate();
    }
}
```

java –prof

```sh
javanac Hello.java
java –prof Hello
```

<table>
<thead>
<tr>
<th>count</th>
<th>callee</th>
<th>caller</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>java.io.PrintStream.println(java/lang/String;)</td>
<td>V Hello.hibernate()</td>
</tr>
<tr>
<td>1</td>
<td>java.io.PrintStream.println(java/lang/String;)</td>
<td>V Hello.nap()</td>
</tr>
<tr>
<td>1</td>
<td>Hello.nap()</td>
<td>V Hello.main(java/lang/String;)</td>
</tr>
<tr>
<td>1</td>
<td>Hello.hibernate()</td>
<td>V Hello.main(java/lang/String;)</td>
</tr>
</tbody>
</table>

Tools for space performance

- Monitoring memory consumptions
- Gabage collection: System.gc()
- Borland Optimizeit (It may be used for evaluation purposes)

HPjmeter.jar

- Java –jar HPjmeter.jar
  - Method call counts
  - Call graph tree
  - Objects created by Method
  - Created Objects
  - …
2. Measuring Complexity

- **LOC**
  - `wc * .java`

- **McCabe and Halstead metrics**
  - For C programs, see the handouts/tools/metric.tar.gz
  - `mccabe *.c`, `halstead *.c`
  - For Java programs in Eclipse
    - [http://metrics.sourceforge.net](http://metrics.sourceforge.net)
    - [http://www.teaminabox.co.uk/downloads/metrics](http://www.teaminabox.co.uk/downloads/metrics)

### Results for the VIM 6.3

<table>
<thead>
<tr>
<th>File</th>
<th>mccabe *.c</th>
<th>less</th>
<th>Complexity</th>
<th>No. of returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>arabic.c</td>
<td>3334</td>
<td>18441</td>
<td>0.016967</td>
<td>1086882.60</td>
</tr>
<tr>
<td>buffer.c</td>
<td>16778</td>
<td>166992</td>
<td>0.004443</td>
<td>37516321</td>
</tr>
<tr>
<td>charlie.c</td>
<td>6106</td>
<td>56173</td>
<td>0.007947</td>
<td>7068181139.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File</th>
<th>halstead *.c</th>
<th>less</th>
<th>Complexity</th>
<th>No. of returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>arabic.c</td>
<td>3334</td>
<td>18441</td>
<td>0.016967</td>
<td>1086882.60</td>
</tr>
<tr>
<td>buffer.c</td>
<td>16778</td>
<td>166992</td>
<td>0.004443</td>
<td>37516321</td>
</tr>
<tr>
<td>charlie.c</td>
<td>6106</td>
<td>56173</td>
<td>0.007947</td>
<td>7068181139.3</td>
</tr>
</tbody>
</table>

For C programs, see the handouts/tools/metric.tar.gz

- `mccabe *.c`, `halstead *.c`
- For Java programs in Eclipse
  - [http://metrics.sourceforge.net](http://metrics.sourceforge.net)
  - [http://www.teaminabox.co.uk/downloads/metrics](http://www.teaminabox.co.uk/downloads/metrics)
Metrics @ sourceforge.net

Net.sourceforge.metrics-site-1.3.5.zip

1. Expand the package into Eclipse
2. Open a “metrics” view
3. Follow the instructions
4. Enable “metric” calculation
5. Rebuild your project

3. Relation to your project

• Opportunities:
  – You may add junit test cases to the code base to reveal bugs (publish it to the bug tracking system) and fix them (+5%)
  – **You may apply design patterns, refactoring techniques on this legacy code base, showing as an improved complexity metrics (+2.5%)**
  – You may tune the performance of the system to speed up the display, load/save for scalable graphs (+2.5%)
• Don’t forget your major project task (up to 100%)!
  – To study the editor methods in the OpenOME and adapt them to the OmniGraphEditor web service.