Tutorial 5
More on Software Quality Measurements

How to measure performance?

How to measure complexity?

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 optimizeit*

Time Performance

- Tool: /bin/time

  werewolf~/>/usr/bin/time

  Usage: /usr/bin/time [-apvV] [-f format] [-o file] [--append] [--verbose]

  [--portability] [--format=format] [--output=file] [--version] [--quiet] [--help] 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

```
index % time    self  children    called     name
                0.02    0.00       1/1       main [2]
[1]     100.0   0.02    0.00         1       hibernate [1]
 -----------------------------------------------
[2]     100.0   0.00    0.02                 main [2]
                0.02    0.00       1/1       hibernate [1]
                0.00    0.00       1/1       nap [3]
-----------------------------------------------
                0.00    0.00       1/1       main [2]
[3]       0.0   0.00    0.00         1       nap [3]
```

Profiler Case Study - Hello.java

```
1.    public class Hello {
2.
3.        static private void hibernate(){
4.                long i;
5.                for(i=0; i<100; i++){
6.                        System.out.println("- A long hibernate...");
7.                }
8.        }
9.        static private void nap(){
10.               long i;
11.               System.out.println("- Just a nap...");
12.       }
13.
14.       public static void main(String args[]){
15.               System.out.println("Take a nap!");
16.               Hello.nap();
17.               System.out.println("Go to hibernate!");
18.               Hello.hibernate();
19.       }
20.   }
```

java –prof

- javac Hello.java
- java –prof Hello

```
count                callee                                caller
100    java.io.PrintStream.println(Ljava/lang/String;)V Hello.hibernate()
1      java.io.PrintStream.println(Ljava/lang/String;)V Hello.nap()
1      Hello.nap()V Hello.main([Ljava/lang/String;)
1      Hello.hibernate()V Hello.main([Ljava/lang/String;)
```

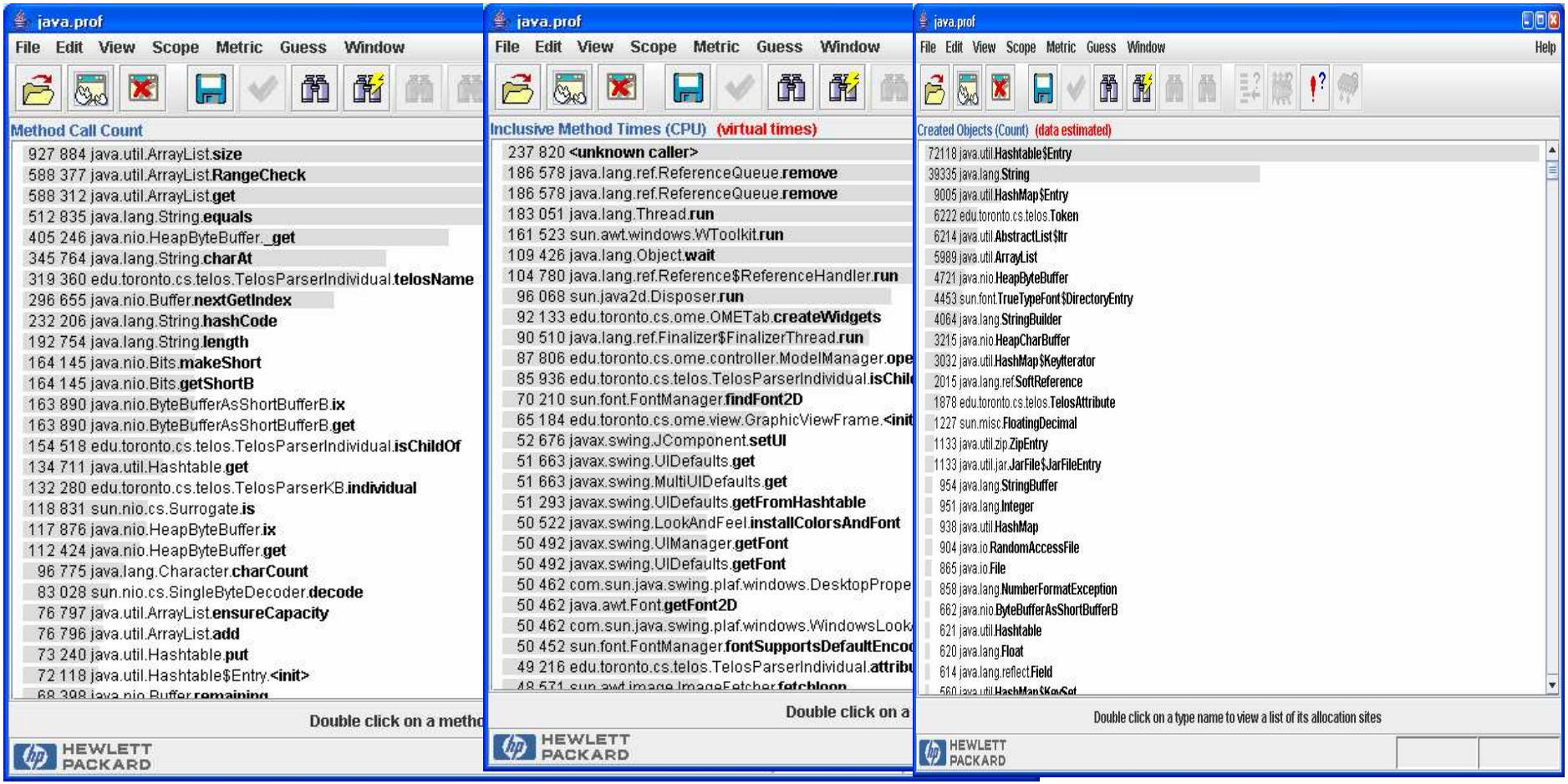
Tools for space performance

- Monitoring memory consumptions
- Gabage collection: System.gc()
- Jim Patrick, “Handling memory leaks in Java programs”, http://www-106.ibm.com/developworks/java/library/j-leaks.
- 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
  - …

Some screenshots for OpenOME measurements



Complexity Metrics

- LOC
- McCabe complexity:
  - Control flow graph <V,E>
  - |V| + |E| - 2
  - It is also measure for test coverage
- Halstead complexity:
  - Operators: N1, unique n1, Operands: N2, unique n2
  - Length = (N1+N2)
    Volume = (N1+N2) $\log_2$ (n1+n2)
    Level = (2/n1)*(n2/N2)
    Efforts (mental discrimination) = Volume / Level
    Time = Efforts / 180000 (hours)
    (50 discrimination per second)
  - It is also useful for project estimations

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://www.teaminabox.co.uk/downloads/metrics

...

Results for the VIM 6.3

mccabe *.c | less

File	Name	Complexity	No. of returns
arabic.c	chg_c_f2m	34	1
arabic.c	A_is_a	38	2
arabic.c	A_is_special	1	1
arabic.c	A_is_f	40	2
arabic.c	A_is_iso	1	1
arabic.c	chg_c_i2m	25	1
arabic.c	A_is_s	37	2
arabic.c	chg_c_a2f	39	1
arabic.c	chg_c_a2i	39	1
arabic.c	half_shape	3	3
arabic.c	A_is_ok	1	1
arabic.c	chg_c_a2m	39	1
arabic.c	chg_c_a2s	39	1
arabic.c	A_is_valid	1	1
arabic.c	arabic_shape	12	2
arabic.c	chg_c_laa2i	6	1
arabic.c	chg_c_laa2f	6	1
arabic.c	A_firstc_laa	2	2
arabic.c	A_is_formb	1	1
arabic.c	A_is_harakat	1	1
buffer.c	fileinfo	18	0

…

3923 functions

…

halstead *.c | less

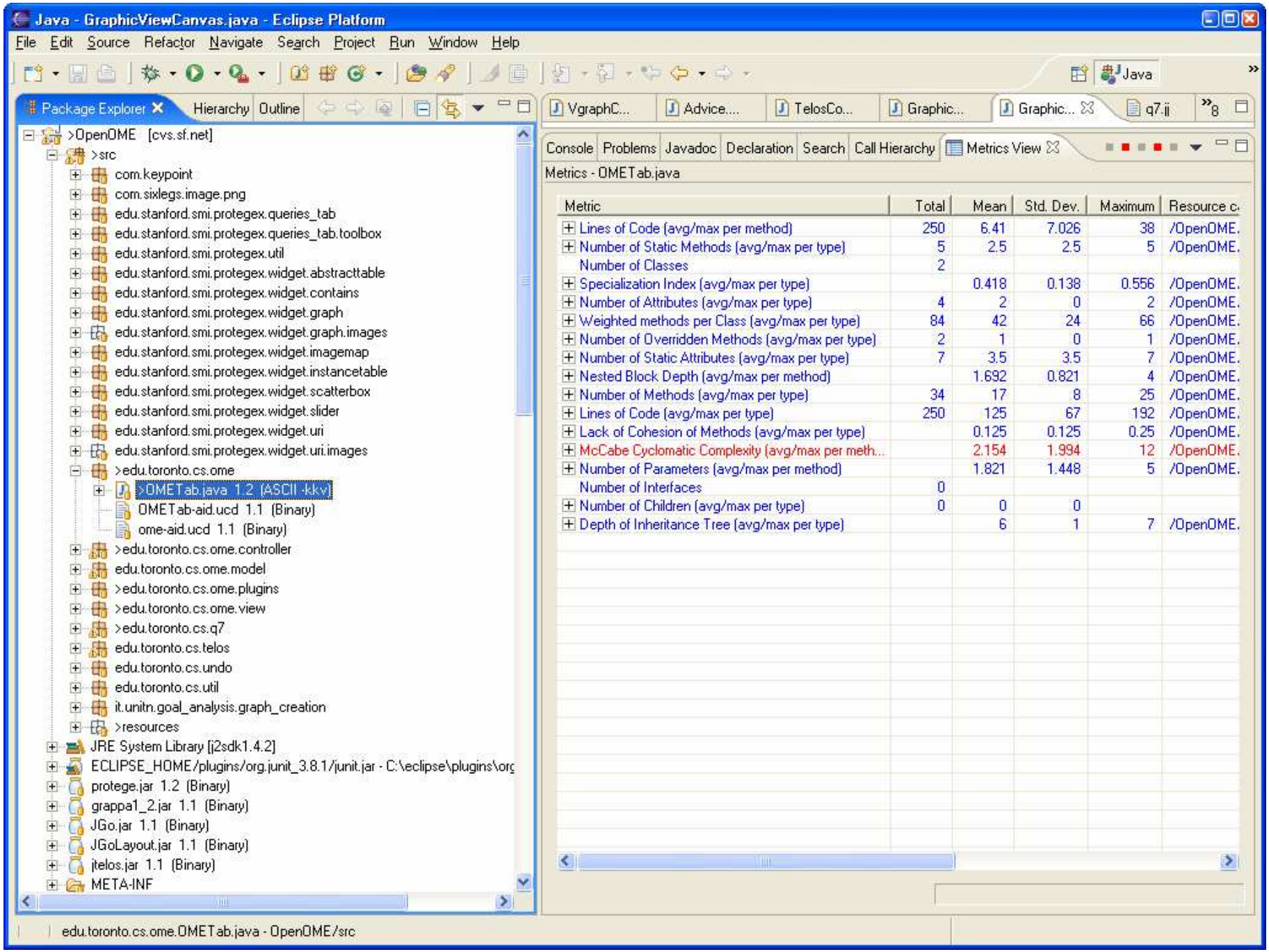
File	length	volume	level	effort	time
arabic.c	2334	18441	0.016967	1086882	6.0
buffer.c	16778	166692	0.004443	37516321	208.4
charset.c	6106	56173	0.007947	7068181	39.3
diff.c	7598	65257	0.003931	16600691	92.2
digraph.c	14522	157014	0.009542	16455714	91.4
edit.c	22527	230316	0.004175	55170619	306.5
eval.c	37745	397470	0.002855	139198597	773.3
ex_cmds.c	19611	198997	0.004440	44821078	249.0
ex_cmds2.c	18711	191075	0.004472	42729876	237.4
ex_docmd.c	32895	355904	0.004319	82409564	457.8
ex_eval.c	5541	45754	0.004331	10564642	58.7
ex_getln.c	18338	182191	0.003881	46943439	260.8
farsi.c	6161	52634	0.006575	8004959	44.5
fileio.c	26949	278105	0.003520	79013285	439.0
fold.c	10686	93084	0.002786	33409135	185.6
getchar.c	13661	129337	0.003870	33416492	185.6
gui.c	15406	153995	0.004973	30963256	172.0
gui_at_fs.c	11929	110062	0.003880	28365559	157.6
gui_at_sb.c	5690	48870	0.005467	8939484	49.7
gui_athena.c	7797	71321	0.005297	13464608	74.8
gui_beval.c	4587	41152	0.008172	5035965	28.0
gui_gtk.c	11409	111274	0.005997	18553811	103.1
gui_gtk_f.c	3136	25477	0.007770	3278996	18.2

… 65 files …

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


Java - GraphicViewCanvas.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explorer
Hierarchy
Outline
>OpenOME [cvs.sf.net]
>src
com.keypoint
com.sixlegs.image.png
edu.stanford.smi.protegex.queries_tab
edu.stanford.smi.protegex.queries_tab.toolbox
edu.stanford.smi.protegex.util
edu.stanford.smi.protegex.widget.abstracttable
edu.stanford.smi.protegex.widget.contains
edu.stanford.smi.protegex.widget.graph
edu.stanford.smi.protegex.widget.graph.images
edu.stanford.smi.protegex.widget.imagemap
edu.stanford.smi.protegex.widget.instancetable
edu.stanford.smi.protegex.widget.scatterbox
edu.stanford.smi.protegex.widget.slider
edu.stanford.smi.protegex.widget.uri
edu.stanford.smi.protegex.widget.uri.images
>edu.toronto.cs.ome
>OMETab.java 1.2 [ASCII -kkv]
OMETab-aid.ucd 1.1 (Binary)
ome-aid.ucd 1.1 (Binary)
>edu.toronto.cs.ome.controller
edu.toronto.cs.ome.model
>edu.toronto.cs.ome.plugins
>edu.toronto.cs.ome.view
>edu.toronto.cs.q7
edu.toronto.cs.telos
edu.toronto.cs.undo
edu.toronto.cs.util
it.unitn.goal_analysis.graph_creation
>resources
JRE System Library [j2sdk1.4.2]
ECLIPSE_HOME/plugins/org.junit_3.8.1/junit.jar - C:\eclipse\plugins\org
protege.jar 1.2 (Binary)
grappa1_2.jar 1.1 (Binary)
JGo.jar 1.1 (Binary)
JGoLayout.jar 1.1 (Binary)
jtelos.jar 1.1 (Binary)
META-INF
VgraphC...
Advice....
TelosCo...
Graphic...
Graphic...
q7.jj
Console
Problems
Javadoc
Declaration
Search
Call Hierarchy
Metrics View
Metrics - OMETab.java
Metric
Total
Mean
Std. Dev.
Maximum
Resource c
Lines of Code (avg/max per method) 250 6.41 7.026 38 /OpenOME.
Number of Static Methods (avg/max per type) 5 2.5 2.5 5 /OpenOME.
Number of Classes 2
Specialization Index (avg/max per type) 0.418 0.138 0.556 /OpenOME.
Number of Attributes (avg/max per type) 4 2 0 2 /OpenOME.
Weighted methods per Class (avg/max per type) 84 42 24 66 /OpenOME.
Number of Overridden Methods (avg/max per type) 2 1 0 1 /OpenOME.
Number of Static Attributes (avg/max per type) 7 3.5 3.5 7 /OpenOME.
Nested Block Depth (avg/max per method) 1.692 0.821 4 /OpenOME.
Number of Methods (avg/max per type) 34 17 8 25 /OpenOME.
Lines of Code (avg/max per type) 250 125 67 192 /OpenOME.
Lack of Cohesion of Methods (avg/max per type) 0.125 0.125 0.25 /OpenOME.
McCabe Cyclomatic Complexity (avg/max per meth... 2.154 1.994 12 /OpenOME.
Number of Parameters (avg/max per method) 1.821 1.448 5 /OpenOME.
Number of Interfaces 0
Number of Children (avg/max per type) 0 0 0
Depth of Inheritance Tree (avg/max per type) 6 1 7 /OpenOME.
edu.toronto.cs.ome.OMETab.java - OpenOME/src

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.