

Metric Tools for Java Programs

Zoran Putnik

Department of Informatics and Mathematics,
Faculty of Science, University of Novi Sad

Free Metric Tools for Java

- 
- **JCSC**
 - **CheckStyle**
 - **JavaNCSC**
 - **JMT**
 - **Eclipse plug-in**

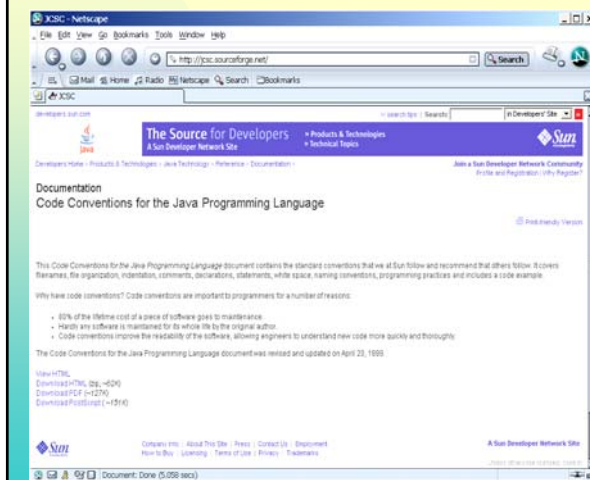
JCSC – Java Coding Standard Checker

JCSC - Netscape
File Edit View Go Bookmarks Tools Window Help
http://jcsc.sourceforge.net/
JCSC
Java Coding Standard Checker
Check and feel better about your code!
(c) 1999-2004 by Ralph Jocham (rjocham72@netscape.net)
JCSC is released under the terms of the [GNU General Public License](#)
JCSC
Overview
JCSC is a powerful tool to check source code against a highly definable coding standard and potential bad code. The standard covers naming conventions for class, interfaces, fields, parameter, ... Also the structural layout of the type (class/interface) can be defined. Like where to place fields, either before or after the methods and in which order. The order can be defined through the visibility or by type (instance, class, constant). The same is applicable for methods. Each of those rules is highly customizable. Readability is enhanced by defining where to put white spaces in the code and when to use braces. The existence of correct JavaDoc can be enforced and various levels. Apart from that, it finds weaknesses in the code -- potential bugs -- like empty catch/finally block, switch without default, throwing of type 'Exception', slow code, ... JC is inspired by lint.
JCSC features a graphical UI for easy rule configuration, a command line interface, an Ant [task](#) which produces a [JavaDoc style](#) webpage and is [CruiseControl](#) compatible (see [screenshot](#)). Also, the [NCSS/CCN](#) metrics have been added for easier code quality assessment.
Currently JCSC supports over 100 enforceable rules -- this is not all and there is more to come!

Overview

- JCSC is a powerful tool to check source code against a highly definable coding standard and potential bad code.
- The standard covers:
 - ◆ naming conventions for class, interfaces, fields, parameter, ...
 - ◆ the structural layout of the type (class/interface)
 - ◆ finds weaknesses in the the code -- potential bugs -- like empty catch/finally block, switch without default, throwing of type 'Exception', slow code, ...
- It can be downloaded at:
<http://jcsc.sourceforge.net/>

Performed Checks



4th Workshop on SEE and RE

5

- The default checking rules adhere to the Sun Code Conventions with some additional auditing for weak code.

General Checks

- correct class/interface header
- line length
- NCCS (non commenting source statements = real code) count for class
- NCCS checking for method length
- CCN (cyclomatic complexity number) checking for methods
- tabulators in source code allowed
- `.*` imports allowed or use fully qualified imports
- only catch subclassed exceptions; not `Throwable` and `Exception`
- check declaration modifier order (also in nested classes)
- interface are not declared `abstract`
- `'l'` for long values allowed (`'l'` if often mistaken for `'1'` -> use `'L'`; i.e. `40l` or `40L`)
- space after statement (`if`, `else`, `while`, ...)
- space after method name
- spaces around binary expression
- throwing of `Exception` or only of subclasses types allowed
- only catching of specialized Exceptions (not `Throwable`, `Exception`) is allowed
- switch statement requires default
- assignments in conditional expressions allowed (`if (a = 5)`)
- only one declaration per line
- allow `'get'` prefix for method returning a boolean or enforce `'is'`, `'has'`, `'are'` instead
- allow type `'Vector'` to be returned, use `'List'` or `'Collection'` instead; new faster `Collection` API
- allow type `'Hashtable'` to be returned, use `'HashMap'` instead; new faster `Collection` API
- allow `String` literals or only constants (`final static String`) in code. Important for internationalization
- empty catch block allowed or indicated
- empty finally block allowed or indicated
- complex loop expression allowed or customizable
- conditional expression allowed or indicated
- number of arguments of method calls; too many arguments indicate procedural programming
- semicolon after type declaration allowed (this is C++)
- single line block without `'{'`, `'}'` allowed (`if`, `else`, ...)
- Array specifier at type (ie. `String[]` names and not `String names[]`)
- allow `public`, `protected` or `package private` fields
- allow `public` fields or not
- check `[]` at type and not at field name
- indicate when too many parameters are passed

4th Workshop on SEE and RE

6

Metrics - 1

- NCSS (non commenting source statements- real code) are being calculated for the whole project, individual classes and methods
- NCSS is an acronym for Non Commenting Source Statements. This number represents pure functionality code lines in a source file. Comparing this number and the violations count, the quality can be easier assest.

Metrics - 2

- CNN (cyclomatic complexity number - possible number of pathes through a method) are generated for all methods and constructors
- CCN is an acronym for Cyclomatic Complexity Number. This number indicates the number in how many branches the flow is split. Each method has a CCN of 1 per default.

Recommendations



- Bruce Eckel used JCSC to validate the code examples in his 3rd edition of Thinking in Java

Usage

- Used as a command-line tool
- Through **commercial** extensions, offers several GUI's for work:
 - ◆ Rules Editor UI
 - ◆ Ant
 - ◆ IntelliJ IDEA
 - ◆ CruiseControl
- Cannot scan more than one file at the time, cannot scan the whole folder recursively.

Usage – command line

■ `jcsc [option] <file>`

■ with the option being:

- ◆ `-h` : show the help
- ◆ `-r <rule>` :the rule file is read from the file system
- ◆ `-j <rule>` :the rule file is read from the `jcsc.jar` file

Violations:

```
c:\SemOrg\test\timeverifier.java:11:1:class Declaration
JavaDoc does not provide the required '@author'
tag:TypeDeclarationAuthor:3
```

```
c:\SemOrg\test\timeverifier.java:11:1:class Declaration
JavaDoc does not provide the required '@version'
tag:TypeDeclarationVersion:3
```

...

5 violation(s) found

Metrics:

```
13:43:TimeVerifier.verify():NCSS-10:CCN-6
```

```
NCSS count : 16
```

```
Lines count : 30
```

```
Methods count : 1
```

```
Unit Test Class count : 0
```

```
Unit Tests count : 0
```

4th Workshop on SEE and RE

11

Usage – Ant tool (demo data)

■ `ant -buildfile jcsc.xml all`

Results are stored in a XML file that can be viewed via XSL compliant browsers.

■ **REPORT**

Tested with Mozilla 1+ and IE6

```
/cygdrive/e/jakarta-ant-1.5/lib/ xml-apis.jar:
/cygdrive/e/ jakarta-ant-1.5/lib/xercesImpl.jar:
/cygdrive/e/jakarta-ant-1.5/lib/optional.jar:
/cygdrive/e/jakarta-ant-1.5/lib/gnu-regexp.jar:
/cygdrive/e/jakarta-ant-1.5/
...
e:\jaxp-1.1\jaxp.jar;e:\jaxp-1.1\crimson.jar;
e:\jaxp-1.1\xalan.jar;e:\jakarta-ant\lib\ant.jar;
e:\jakarta-ant\lib\jakarta-ant-1.4.1-optional.jar
```

Buildfile: jcsc.xml

all:

```
[jcsc] Package Count:      10
[jcsc] Class Count:       26
[jcsc] Methods Count:     281
[jcsc] Total Violations Count : 153
[jcsc] Avg Violations per Class: 5.8846154
[jcsc] Total NCSS Count:   3430
[jcsc] Avg Violations per NCSS: 0.044606414
[jcsc] Unit Test Class Count: 2
[jcsc] Unit Tests Count:   83
```

BUILD SUCCESSFUL

4th Workshop on SEE and RE

12

Metrics results for SemOrg

Name	NCSS	CCN
CompanyBookingList	16; 14	8; 7
ClientBookingList	16; 14	8; 7
ClientPresentationList	15; 13	8; 7
CanConductList	15; 14	8; 7
CompanyPresentationList	15; 13	8; 7
ClientList	15; 13	8; 7

4th Workshop on SEE and RE

13

Free Metric Tools for Java

- **JCSC**
- ➔ ■ **CheckStyle**
- **JavaNCSC**
- **JMT**
- **Eclipse plug-in**

4th Workshop on SEE and RE

14

CheckStyle

- Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard.
- It automates the process of checking Java code to spare humans of this boring (but important) task.
- This makes it ideal for projects that want to enforce a coding standard.

Usage

- Checkstyle is highly configurable and can be made to support almost any coding standard.
- It can be used as:
 - ◆ An Ant task.
 - ◆ A command line tool.
- It can be downloaded at:
<http://checkstyle.sourceforge.net/>

Features

- The things that Checkstyle can check for are:
 - ◆ Javadoc Comments
 - ◆ Naming Conventions
 - ◆ Headers
 - ◆ Imports
 - ◆ Size Violations
 - ◆ Whitespace
 - ◆ Modifiers
 - ◆ Blocks
 - ◆ Coding Problems
 - ◆ Class Design
 - ◆ Duplicate Code
 - ◆ **Metrics Checks**
 - ◆ Miscellaneous Checks
 - ◆ Optional Checks

Metrics Checks

- BooleanExpressionComplexity
- ClassDataAbstractionCoupling
- ClassFanOutComplexity
- CyclomaticComplexity
- NPathComplexity

Command line usage

- The command line usage is:

- ◆ `java -D<property>=<value> \ com.puppycrawl.tools.checkstyle.Main \ -c <configurationFile> [-n <packageNameFile>] \ [-f <format>] [-p <propertiesFile>] [-o <file>] \ [-r <dir>] file...`

- Command line options are:

- ◆ `-n packageNameFile` - specify a package names file to use.
- ◆ `-f format` - specify the output format. Options are "plain" for the DefaultLogger and "xml" for the XMLLogger. Defaults to "plain".
- ◆ `-p propertiesFile` - specify a properties file to use.
- ◆ `-o file` - specify the file to output to.
- ◆ `-r dir` - specify the directory to traverse for Java source files.

Features

- Checkstyle will process the specified file and report errors to standard output in plain format.
- Checkstyle requires a configuration XML file that configures the checks to apply.
- Checkstyle reports errors, but reports "metric errors" **only** if a program has measured value greater than default.

An example of configuration XML file is:

```
<module name="Checker">
  <module name="TreeWalker">

    <!-- Default value is 3 -->
    <module name="BooleanExpressionComplexity">
      <property name="max" value="1"/>
    </module>

    <!-- Default value is 7 -->
    <module name="ClassDataAbstractionCoupling">
      <property name="max" value="1"/>
    </module>

    <!-- Default value is 20 -->
    <module name="ClassFanOutComplexity">
      <property name="max" value="1"/>
    </module>

    <!-- Default value is 10 -->
    <module name="CyclomaticComplexity">
      <property name="max" value="1"/>
    </module>

    <!-- Default value is 200 -->
    <module name="NPathComplexity">
      <property name="max" value="1"/>
    </module>
  </module>
</module>
```

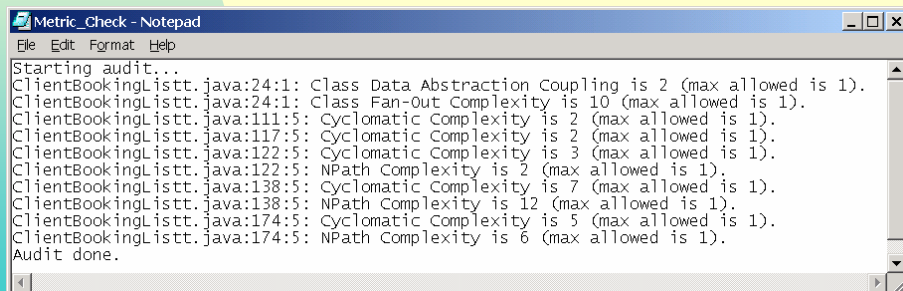
4th Workshop on SEE and RE

21

■ An example of a result:

```
C:\Jezic1\Metric\CheckStyle\checkstyle-3.4>java com.puppycrawl.tools.checkstyle.
Main -c docs/metric_checks.xml \ test.java
Starting audit...
test.java:21:1: Class Fan-Out Complexity is 3 (max allowed is 1).
test.java:36:1: Cyclomatic Complexity is 3 (max allowed is 1).
test.java:36:1: NPath Complexity is 2 (max allowed is 1).
Audit done.
```

■ An example of a result on a known "bad" function:



```
Metric_Check - Notepad
File Edit Format Help
Starting audit...
ClientBookingListt.java:24:1: Class Data Abstraction Coupling is 2 (max allowed is 1).
ClientBookingListt.java:24:1: Class Fan-Out Complexity is 10 (max allowed is 1).
ClientBookingListt.java:111:5: Cyclomatic Complexity is 2 (max allowed is 1).
ClientBookingListt.java:117:5: Cyclomatic Complexity is 2 (max allowed is 1).
ClientBookingListt.java:122:5: Cyclomatic Complexity is 3 (max allowed is 1).
ClientBookingListt.java:122:5: NPath Complexity is 2 (max allowed is 1).
ClientBookingListt.java:138:5: Cyclomatic Complexity is 7 (max allowed is 1).
ClientBookingListt.java:138:5: NPath Complexity is 12 (max allowed is 1).
ClientBookingListt.java:174:5: Cyclomatic Complexity is 5 (max allowed is 1).
ClientBookingListt.java:174:5: NPath Complexity is 6 (max allowed is 1).
Audit done.
```

4th Workshop on SEE and RE

22

Most distinct characteristics

- It **checks** a project as a whole, yet a it gives a report **only** on items greater than the default values.
- It gives different results than all of the other checks.

Free Metric Tools for Java

- **JCSC**
- **CheckStyle**
- ➔ ■ **JavaNCSC**
- **JMT**
- **Eclipse plug-in**

JavaNCSS extensions

- JavaNCSS can optionally present its output with a little graphical user interface.
- ... or as a result in a command line. (If no option is given, JavaNCSS only calculates the total non commenting source statements (NCSS) of the given input.)

Nr.	Classes	Functions	NCSS	Javadocs	Package
1	1	3	7	0	semorg.gui

	1	3	7	0	Total

Packages	Classes	Functions	NCSS	Javadocs	per

1.00	1.00	3.00	7.00	0.00	Project
	1.00	3.00	7.00	0.00	Package
		3.00	7.00	0.00	Class
			2.33	0.00	Function

```
C:\Jozsi\Metric\JCSC\javancss21.41\javancss21.41\bin>javancss c:\semorg\test\changelistener.java
Java NCSS: 7
```

Complete results for "SemOrg" - Packages

Nr.	Classes	Functions	NCSS	Javadocs	Package
1	2	33	194	28	semorg
2	6	108	510	74	semorg.booking
3	8	128	439	124	semorg.classes
4	2	47	202	35	semorg.company
5	137	439	3431	0	semorg.gui
6	8	122	564	82	semorg.person
7	6	104	488	73	semorg.presentation
8	5	88	367	64	semorg.seminar

	174	1069	6195	480	Total

Packages	Classes	Functions	NCSS	Javadocs	per

8.00	174.00	1069.00	6195.00	480.00	Project
	21.75	133.63	774.38	60.00	Package
		6.14	35.60	2.76	Class
			5.80	0.45	Function

Complete results for "SemOrg" - Classes

Packages	Classes	Methods	
159	6	2	0
160	6	2	0
161	6	2	0
162	6	2	0
163	6	2	0
164	6	2	0
165	6	2	0
166	6	2	0
167	6	2	0
168	6	2	0
169	138	36	0
170	24	2	1
171	146	31	0
172	27	2	0
173	69	18	0
174	11	1	0
Average Object NCSS:			32.05
Average Object Functions:			6.14
Average Object Inner Classes:			0.01
Average Object Javadoc Comments:			2.76
Program NCSS:			6,195.00

Complete results for "SemOrg" - Methods

Packages	Classes	Methods	
1054	9	2	1
1055	2	1	1
1056	2	1	1
1057	2	1	1
1058	2	1	1
1059	2	1	1
1060	2	1	1
1061	2	1	1
1062	2	1	1
1063	2	1	1
1064	2	1	1
1065	2	1	1
1066	2	1	1
1067	2	1	0
1068	4	1	0
1069	20	6	0
1070	10	6	0
Average Function NCSS:			4.43
Average Function CCN:			1.40
Average Function JVDC:			0.41
Program NCSS:			6,195.00

- For "Methods", two additional metrics are added:
 - CCN – cyclomatic complexity number
 - JVDC – indication whether this method is formally documented or not.
- Results are the same as with the previous tool.

Usage

- Multiple java source files can be specified in the command line:
 - ◆ If a '@' char is put in front of a file name, then not this file will be measured but its content will be interpreted as a list of Java source files that shall be counted.
 - ◆ Wild cards are not supported yet. (If the operating system processes the command line for the program, then you are lucky. Windows doesn't do that.)
 - ◆ Yet ... the following line processes all files:

```
javancss -gui *.java
```

Usage and options 1/2

- **Synopsis**
 - ◆ javancss [-option] stdin | [@]source_file*
- **Options**
 - ◆ -ncss
This is the default which counts total NCSS.
 - ◆ -package
Collects the metrics data for each package. This is the most top level view javancss offers for your projects.
 - ◆ -object
Collects the metrics data for each class/interface.
 - ◆ -function
Collects the metrics data for each function.
 - ◆ -all
The same as '-package -object -function'.

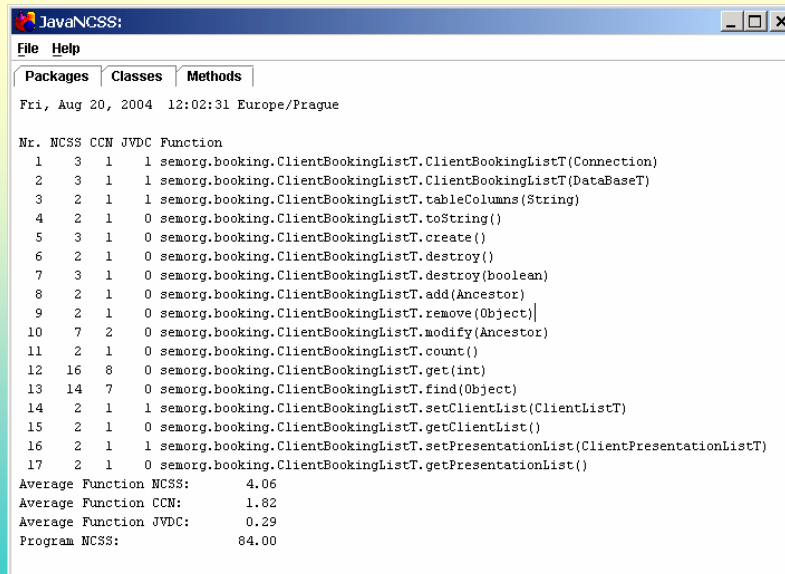
Usage and options 2/2

- ◆ -gui
Opens a gui to presents the '-all' output in tabbed panels.
- ◆ -xml
Output in xml and not in ascii format. Additional option '-all' is recommended.
- ◆ -out file
Output goes normally to standard output, with this option an output file can be specified.
- ◆ -recursive
Java file in sub directories will be parsed as well.
- ◆ -check
Trigger JavaNCSS self test suite.
- ◆ -version
Prints out the version of JavaNCSS.
- ◆ -help
Prints out some basic information.

4th Workshop on SEE and RE

33

Result for known "bad" functions



JavaNCSS: File Help

Packages Classes Methods

Fri, Aug 20, 2004 12:02:31 Europe/Prague

Nr.	NCSS	CCN	JVDC	Function
1	3	1	1	semorg.booking.ClientBookingListT.ClientBookingListT(Connection)
2	3	1	1	semorg.booking.ClientBookingListT.ClientBookingListT(DataBaseT)
3	2	1	1	semorg.booking.ClientBookingListT.tableColumns(String)
4	2	1	0	semorg.booking.ClientBookingListT.toString()
5	3	1	0	semorg.booking.ClientBookingListT.create()
6	2	1	0	semorg.booking.ClientBookingListT.destroy()
7	3	1	0	semorg.booking.ClientBookingListT.destroy(boolean)
8	2	1	0	semorg.booking.ClientBookingListT.add(Ancestor)
9	2	1	0	semorg.booking.ClientBookingListT.remove(Object)
10	7	2	0	semorg.booking.ClientBookingListT.modify(Ancestor)
11	2	1	0	semorg.booking.ClientBookingListT.count()
12	16	8	0	semorg.booking.ClientBookingListT.get(int)
13	14	7	0	semorg.booking.ClientBookingListT.find(Object)
14	2	1	1	semorg.booking.ClientBookingListT.setClientList(ClientListT)
15	2	1	0	semorg.booking.ClientBookingListT.getClientList()
16	2	1	1	semorg.booking.ClientBookingListT.setPresentationList(ClientPresentationListT)
17	2	1	0	semorg.booking.ClientBookingListT.getPresentationList()

Average Function NCSS: 4.06
Average Function CCN: 1.82
Average Function JVDC: 0.29
Program NCSS: 84.00

4th Workshop on SEE and RE

34

Free Metric Tools for Java

- **JCSC**
- **CheckStyle**
- **JavaNCSC**
- ➔ ■ **JMT**
- **Eclipse plug-in**

JMT – Java Measurement Tool

- **Authors:**
 - ◆ Otto-von-Guericke-Universität Magdeburg, Germany,
 - ◆ Department of Software Engineering
 - ◆ Developed by **Ingo Patett** as his Diplom-Arbeit
 - ◆ Supervised by Prof. Dumke and Dr. Köppe. It was improved by Christian Kolbe under supervision of Dipl.-Inf. Wille
 - ◆ Can be downloaded from <http://ivs.cs.uni-magdeburg.de/sw-eng/agruppe/forschung/tools/>
- **Purpose:**
 - ◆ The Java Measurement Tool realizes a static measurement of Java applications. It analyzes the Java classes and the relations between them.

Application usage

- Two ways of analyzing Java classes.
 - ◆ The first way is to analyze one single Java class - select '*Analyze a File*'.
 - ◆ There is also the possibility to analyze an entire project - select '*Analyze a Project*'.
 - ★ To process an entire project you have to analyze it file by file. Click the button '*Load File*' and select a Java file.
- File to be analyzed must be a source code file, it is impossible to process a compiled class file.

Procedure

- After loading the file click the button '*Analyze File*' to process it.
- Message about the success or an occurring error will appear.
- If you are analyzing a project, continue with the next file.
- After processing the files, the results of the measurement are displayed.
- It is possible to have a look:
 - ◆ at the data of the classes, or
 - ◆ the data of the methods contained within the project.

JMT Tool

Current File: Analyze a File Analyze a Project

Values of the Project

Number of Classes	0	Avg. Number of Parameters per Method	0
Number of Methods	0	Attribut Inheritance Factor	0
Avg. Number of Methods per Class	0	Method Inheritance Factor	0
Avg. Number of Attributs per Class	0	Coupling Factor	0

Values of Classes and Methods

Class Data Method Data

DIT NOC WMC WAC CBO RFC LOC

Exit Analyze File Load File

4th Workshop on SLE and RE

39

Results for a file analysis – “Method Data”

Current File: Analyze a File Analyze a Project

Values of the Project

Number of Classes	1	Avg. Number of Parameters per Method	0.6
Number of Methods	5	Attribut Inheritance Factor	0.0
Avg. Number of Methods per Class	5.0	Method Inheritance Factor	0.0
Avg. Number of Attributs per Class	19.0	Coupling Factor	0.0

Values of Classes and Methods

Class Data Method Data

DIT NOC WMC WAC CBO RFC LOC

Nr.	Method Name	Class	Access Type	NOP	LOC
1	AboutBox	AboutBox	public	1	5
2	!bInIt	AboutBox	private	0	29
3	processWind...	AboutBox	protected	1	3
4	cancel	AboutBox	friendly	0	1
5	actionPerform...	AboutBox	public	1	2

Exit Analyze File Load File

4th Workshop on SLE and RE

40

Results for a file analysis – “Class Data”

Current File: Analyze a File Analyze a Project

Values of the Project

Number of Classes	1	Avg. Number of Parameters per Method	0.6
Number of Methods	5	Attribut Inheritance Factor	0.0
Avg. Number of Methods per Class	5.0	Method Inheritance Factor	0.0
Avg. Number of Attributs per Class	19.0	Coupling Factor	0.0

Values of Classes and Methods

Class Data Method Data

DIT NOC WMC WAC CBO RFC LOC

Nr.	Class Name	Parent Class	DIT	NOC	WMC	WAC	CBO	PIM	NMI	NAI	NMO	RFC	LOC
1	AboutBox	JDialog	0	0	5	19	12	2	0	0	0	17	40

Exit Analyze File Load File

4th Workshop on SEE and RE

41

An example of a single measure

Current File: Analyze a File Analyze a Project

Values of the Project

Number of Classes	1	Avg. Number of Parameters per Method	0.6666667
Number of Methods	12	Attribut Inheritance Factor	0.0
Avg. Number of Methods per Class	12.0	Method Inheritance Factor	0.0
Avg. Number of Attributs per Class	2.0	Coupling Factor	0.0

Values of Classes and Methods

Class Data Method Data

DIT NOC WMC WAC CBO RFC LOC

CBO - Coupling between Object Classes

Class	CBO
1	7
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0

Exit Analyze File Load File

4th Workshop on SEE and RE

42

List of available metrics 2/4

■ *Metrics on Class Level*

- ◆ **PIM** – Number of Public Methods
- ◆ **NMI** – Number of Methods inherited
Number of methods of the direct parent class
- ◆ **NAI** – Number of Attributes inherited
Number of attributes of the direct parent class
- ◆ **NMO** – Number of Methods overwritten
- ◆ **RFC** – Response for a class
Number of methods used by the class plus the methods of the class. Is the highest possible number of methods, which can be invoked by a message to this class.
- ◆ **LOC** – Lines of Code

List of available metrics 3/4

■ *Metrics on Method Level*

- ◆ **NOP** – Number of Parameter
- ◆ **LOC** – Lines of Code

■ *Metrics on Inheritance Level*

- ◆ **MIF** – Method Inheritance Factor
Relation of inherited methods to total number of methods.
- ◆ **AIF** – Attribute Inheritance Factor
Relation of inherited attributes to total number of attributes.

List of available metrics 4/4

- *Metrics on System Level*
 - ◆ **COF** – Coupling Factor
(= (total number of couplings) / (n²-n) ;
with n = number of defined classes)
A high COF points to a high complexity of
the system.
 - ◆ **ANM** – Average Number of Methods per
Class
 - ◆ **ANM** – Average Number of Attributes per
Class
 - ◆ **ANM** – Average Number of Parameter
per Method

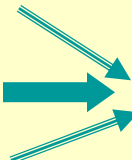
Most distinct characteristics

- It **does** have a possibility to
analyse a project as a whole, yet a
user has to load individually tenths,
hundreds or thousands of files.
- File analysis requires several (un-
necessary) steps: Analyse a File +
Load File + Select File + Open File
+ Analyse File + OK

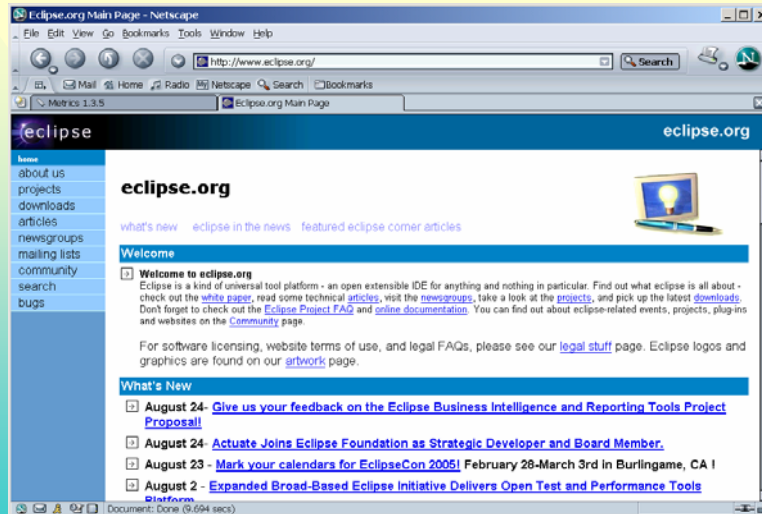
Resume

- **JCSC** – OK, but too simple – only 2 checks, only file-per-file checking
- **CheckStyle** – reports only errors, not check result, gives “wrong” results
- **JavaNCSS** – good, small number of metrics, but easy to use and with solid user interface
- **JMT** – most serious, the greatest number of checks, a slightly too complicated usage

Free Metric Tools for Java

- **JCSC**
 - **CheckStyle**
 - **JavaNCSC**
 - **JMT**
 - **Eclipse plug-in**
- 

Eclipse project



4th Workshop on SEE and RE

51

Basics

- Eclipse is an open platform for tool integration built by an open community of tool providers.
- It operates under a open source paradigm, with a common public license that provides royalty free source code and worldwide redistribution rights.
- Eclipse platform provides tool developers with ultimate flexibility and control over their software technology.

4th Workshop on SEE and RE

52

History

- Industry leaders: Borland, IBM, MERANT, QNX Software Systems, Rational Software³, Red Hat, SuSE, TogetherSoft³ and Webgain² formed the initial eclipse.org Board of Stewards in November 2001.
- Since then, a lot of new members joined. For example: Fujitsu, Hitachi, HP, Oracle, Object Management Group (OMG) Fraunhofer Institute, Ericsson, QA Systems, Advanced Systems Concepts, Genuitec, INNOOPRACT Informationssysteme GmbH, Intel ...

“Metrics” plug – in

- Installation of the whole project is (extremely) simple, and consists of un-zipping.
- The same stands for adding the “metric” plug-in that should be un-zipped at the appropriate place and enabled as an option inside Eclipse.

GUI of Eclipse + Metric plug-in

The screenshot shows the Eclipse IDE with the Metrics view open. The Metrics view displays a table of metrics for various packages and classes. The table has columns for Metric, Total, Mean, Std. Dev., and Max. The metrics include Lines of Code, Number of State Methods, Affluent Coupling, Normalized Distance, Number of Classes, Specification Index, Instability, Number of Attributes, Number of Packages, Weighted Methods per Class, Number of Overridden Methods, Number of Method Depth, Number of Static Attributes, Number of Methods, Lines of Code, Lock of Cohesion of Methods, and McCabe Cyclomatic Complexity.

Metric	Total	Mean	Std. Dev.	Max
Lines of Code (avg/max per method)	5018	4.694	12.845	204
Number of State Methods (avg/max per type)	57	0.328	0.782	4
Affluent Coupling (avg/max per package/fragment)	12.375	13.964		
Normalized Distance (avg/max per package/fragment)	0.907	0.242		
Number of Classes (avg/max per package/fragment)	174	23.75	43.614	
Specification Index (avg/max per type)	0.232	0.372		
Instability (avg/max per package/fragment)	0.462	0.281	0	
Number of Attributes (avg/max per type)	669	3.695	8.001	
Number of Packages	8			
Weighted Methods per Class (avg/max per type)	1441	0.282	10.522	
Number of Overridden Methods (avg/max per type)	80	0.46	0.973	
Number of Method Depth (avg/max per method)	1.394	0.52		
Number of Static Attributes (avg/max per type)	7	0.04	0.377	
Number of Methods (avg/max per type)	1012	5.616	7.056	
Lines of Code (avg/max per type)	5018	28.639	50.836	
Lock of Cohesion of Methods (avg/max per type)	0.22	0.345	0	
M McCabe Cyclomatic Complexity (avg/max per met...)	1.340	0.936		

4th Workshop on SEE and RE

55

Line-of-code metrics for SemOrg

The screenshot shows the Eclipse IDE with the Metrics view open, displaying the Lines of Code metric. The table lists metrics for various packages and classes, including Lines of Code, Total, Mean, Std. Dev., and Maximum.

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Lines of Code (avg/max per method)	5018	4.694	12.845	204	/SemOrg/SemOrg/gui/MainFrame.java	btn
(default package)	5018	4.694	12.845	204	/SemOrg/SemOrg/gui/MainFrame.java	btn
semorg.gui	2749	6.262	19.054	204	/SemOrg/SemOrg/gui/MainFrame.java	btn
MainFrame.java	313	3.557	21.517	204	/SemOrg/SemOrg/gui/MainFrame.java	btn
PresenterPanel.java	347	9.379	31.333	192	/SemOrg/SemOrg/gui/PresenterPanel.java	btn
SeminarPanel.java	241	5.605	21.737	145	/SemOrg/SemOrg/gui/SeminarPanel.java	btn
CompanyPanel.java	216	4.968	21.898	124	/SemOrg/SemOrg/gui/CompanyPanel.java	btn
AncestorDialog.java	199	5.237	13.618	85	/SemOrg/SemOrg/gui/AncestorDialog.java	btn
BookingPanel.java	135	6.136	16.515	81	/SemOrg/SemOrg/gui/BookingPanel.java	btn
AddressPanel.java	117	6.158	16.617	76	/SemOrg/SemOrg/gui/AddressPanel.java	btn
PersonPanel.java	100	6.316	15.506	71	/SemOrg/SemOrg/gui/PersonPanel.java	btn
ContactPanel.java	107	6.294	16.109	70	/SemOrg/SemOrg/gui/ContactPanel.java	btn
ClientPresentationPanel.java	112	7	15.7	67	/SemOrg/SemOrg/gui/ClientPresentationPanel.java	btn
ClientBookingPanel.java	101	11.312	15.003	57	/SemOrg/SemOrg/gui/ClientBookingPanel.java	btn
ClientPanel.java	107	8.231	13.779	52	/SemOrg/SemOrg/gui/ClientPanel.java	btn
NamePanel.java	77	5.923	12.313	48	/SemOrg/SemOrg/gui/NamePanel.java	btn
AssociatePanel.java	70	6	12.050	47	/SemOrg/SemOrg/gui/AssociatePanel.java	btn
BoardPanel.java	63	6.3	9.829	36	/SemOrg/SemOrg/gui/BoardPanel.java	btn
CompanyBookingPanel.java	101	10.1	10.568	31	/SemOrg/SemOrg/gui/CompanyBookingPanel.java	btn
AboutBox.java	46	9.2	10.447	30	/SemOrg/SemOrg/gui/AboutBox.java	btn
ClientContactPanel.java	91	9.1	9.246	25	/SemOrg/SemOrg/gui/ClientContactPanel.java	btn
CompanyPresentationPanel.java	42	6	6.437	21	/SemOrg/SemOrg/gui/CompanyPresentationPan...	btn
SQLPanel.java	19	9.5	4.5	14	/SemOrg/SemOrg/gui/SQLPanel.java	btn
TimeInfer.java	13	13	0	13	/SemOrg/SemOrg/gui/TimeInfer.java	verify
IntegerVerifier.java	10	10	0	10	/SemOrg/SemOrg/gui/IntegerVerifier.java	verify
FloatVerifier.java	10	10	0	10	/SemOrg/SemOrg/gui/FloatVerifier.java	verify
AncestorPanel.java	4	1	0.707	2	/SemOrg/SemOrg/gui/AncestorPanel.java	setMode
ChangeListener.java	0	0	0	0		
semorg.company	193	4.106	6.544	29	/SemOrg/SemOrg/company/Company.java	toString
semorg.presentation	434	4.173	5.79	29	/SemOrg/SemOrg/presentation/Presentation.java	Presentation
semorg.seminar	296	3.25	6.028	29	/SemOrg/SemOrg/seminar/Seminar.java	Seminar
semorg	129	3.909	5.518	26	/SemOrg/SemOrg/SemOrg.java	createTables
semorg.classes	342	2.672	3.398	22	/SemOrg/SemOrg/classes/TimeT.java	valueOf
semorg.booking	433	4.009	4.61	21	/SemOrg/SemOrg/booking/CompanyBookingList...	get
semorg.person	452	3.705	4.515	19	/SemOrg/SemOrg/person/AssociateList.java	get
Number of Static Methods (avg/max per type)	57	0.328	0.782	4	/SemOrg/SemOrg/classes/TimeT.java	

4th Workshop on SEE and RE

56

Example of another "bad" metrics

Metric	Total	Mean	Std. Dev	Maximum	Resource causing Maximum	Method
(H) Lines of Code (avg/max per method)	5018	4.694	12.895	204	/SemOrg/semorg/gui/MainFrame.java	blank
(H) Number of Static Methods (avg/max per type)	57	0.320	0.702	4	/SemOrg/semorg/classes/TimeT.java	
(H) Affarent Coupling (avg/max per package/fragment)	12.375	13.964	47		/SemOrg/semorg/classes	
(H) Normalized Distance (avg/max per package/fragment)	0.507	0.242	0.75		/SemOrg/semorg/company	
(H) Number of Classes (avg/max per package/fragment)	174	21.75	43.614	137	/SemOrg/semorg/gui	
(H) Specialization Index (avg/max per type)	0.232	0.372	1.2		/SemOrg/semorg/gui/CompanyPresentationPanel.java	
(H) Instability (avg/max per package/fragment)	0.462	0.281	0.957		/SemOrg/semorg/gui	
(H) Number of Attributes (avg/max per type)	669	3.845	8.001	70	/SemOrg/semorg/gui/MainFrame.java	
(H) Number of Packages	8					
(H) Weighted methods per Class (avg/max per type)	1441	8.282	10.522	48	/SemOrg/semorg/booking/ClientBooking.java	
(H) Number of Overridden Methods (avg/max per type)	80	0.46	0.573	2	/SemOrg/semorg/booking/ClientBooking.java	
(H) Nested Block Depth (avg/max per method)	3.204	0.82	4		/SemOrg/semorg/classes/TimeT.java	
(H) Number of Static Attributes (avg/max per type)	7	0.04	0.377	4	/SemOrg/semorg/SemOrgListT.java	count
(H) Number of Methods (avg/max per type)	1012	5.816	7.056	35	/SemOrg/semorg/presentation/PresentationT.java	
(H) Lines of Code (avg/max per type)	5018	28.839	90.934	325	/SemOrg/semorg/gui/PresentationPanel.java	
(H) Lack of Cohesion of Methods (avg/max per type)	0.22	0.345	0.992		/SemOrg/semorg/gui/MainFrame.java	
(H) McCabe Cyclomatic Complexity (avg/max per method)	1.348	0.936	8		/SemOrg/semorg/gui/PresentationPanel.java	setElement
(H) Number of Parameters (avg/max per method)	0.865	1.272	17		/SemOrg/semorg/presentation/ClientPresentationT.java	ClientPresentationT
(H) Number of Parameters (avg/max per method)	0.865	1.272	17		/SemOrg/semorg/presentation/ClientPresentationT.java	ClientPresentationT
(H) Number of Parameters (avg/max per method)	1.019	2.43	17		/SemOrg/semorg/presentation/ClientPresentationT.java	ClientPresentationT
(H) Number of Parameters (avg/max per method)	0.75	1.432	13		/SemOrg/semorg/seminar/SeminarT.java	SeminarT
(H) Number of Parameters (avg/max per method)	0.824	1.353	10		/SemOrg/semorg/booking/ClientBookingT.java	ClientBookingT
(H) Number of Parameters (avg/max per method)	0.766	1.462	10		/SemOrg/semorg/company/CompanyT.java	CompanyT
(H) Number of Parameters (avg/max per method)	0.893	1.603	10		/SemOrg/semorg/person/AssociateT.java	AssociateT
(H) Number of Parameters (avg/max per method)	0.888	0.556	7		/SemOrg/semorg/gui/MainFrame.java	MainFrame
(H) Number of Parameters (avg/max per method)	0.867	1.114	6		/SemOrg/semorg/classes/TimeT.java	modify
(H) Number of Parameters (avg/max per method)	0.545	0.782	4		/SemOrg/semorg/SemOrgListT.java	SemOrgListT
(H) Abstractness (avg/max per package/fragment)	0.031	0.003	0.25		/SemOrg/semorg/classes	
(H) Number of Interfaces (avg/max per package/fragment)	0	0	0			
(H) Different Coupling (avg/max per package/fragment)	7.125	6.133	22		/SemOrg/semorg/gui	
(H) Number of Children (avg/max per type)	48	0.276	1.83	17	/SemOrg/semorg/gui/AncestorPanel.java	
(H) Depth of Inheritance Tree (avg/max per type)	1.954	1.579	6		/SemOrg/semorg/gui/PersonPanel.java	

4th Workshop on SEE and RE

57

More detailed view of the same results

Metric	Total	Mean	Std. Dev	Maximum	Resource causing Maximum	Method
(H) McCabe Cyclomatic Complexity (avg/max per method)	1.348	0.936	8		/SemOrg/semorg/gui/PresentationPanel.java	setElement
(H) Number of Parameters (avg/max per method)	0.865	1.272	17		/SemOrg/semorg/presentation/ClientPresentationT.java	ClientPresentationT
(H) Number of Parameters (avg/max per method)	0.865	1.272	17		/SemOrg/semorg/presentation/ClientPresentationT.java	ClientPresentationT
(H) Number of Parameters (avg/max per method)	1.019	2.43	17		/SemOrg/semorg/presentation/ClientPresentationT.java	ClientPresentationT
(H) Number of Parameters (avg/max per method)	1.625	4.014	17		/SemOrg/semorg/presentation/ClientPresentationT.java	ClientPresentationT
(H) Number of Parameters (avg/max per method)	1.625	4.014	17		/SemOrg/semorg/presentation/ClientPresentationT.java	ClientPresentationT
(H) Number of Parameters (avg/max per method)	0.865	1.272	17		/SemOrg/semorg/presentation/ClientPresentationT.java	ClientPresentationT
(H) Number of Parameters (avg/max per method)	0.865	1.272	17		/SemOrg/semorg/presentation/ClientPresentationT.java	ClientPresentationT
(H) Number of Parameters (avg/max per method)	0.692	0.462	1		/SemOrg/semorg/presentation/CompanyPresentationT.java	CompanyPresentationT
(H) Number of Parameters (avg/max per method)	0.714	0.452	1		/SemOrg/semorg/presentation/ClientPresentationListT.java	ClientPresentationListT
(H) Number of Parameters (avg/max per method)	0.643	0.479	1		/SemOrg/semorg/presentation/PresentationListT.java	PresentationListT
(H) Number of Parameters (avg/max per method)	0.75	1.432	13		/SemOrg/semorg/seminar/SeminarT.java	SeminarT
(H) Number of Parameters (avg/max per method)	0.861	2.123	13		/SemOrg/semorg/seminar/SeminarT.java	SeminarT
(H) Number of Parameters (avg/max per method)	0.667	0.745	2		/SemOrg/semorg/seminar/CanConductT.java	CanConductT
(H) Number of Parameters (avg/max per method)	0.736	0.666	2		/SemOrg/semorg/seminar/CanConductListT.java	CanConductListT
(H) Number of Parameters (avg/max per method)	0.625	0.404	1		/SemOrg/semorg/seminar/CanConductListT.java	CanConductListT
(H) Number of Parameters (avg/max per method)	0.692	0.462	1		/SemOrg/semorg/seminar/SeminarListT.java	SeminarListT
(H) Number of Parameters (avg/max per method)	0.654	1.353	10		/SemOrg/semorg/booking/ClientBookingT.java	ClientBookingT
(H) Number of Parameters (avg/max per method)	1.134	2.007	10		/SemOrg/semorg/booking/ClientBookingT.java	ClientBookingT
(H) Number of Parameters (avg/max per method)	1.188	1.845	8		/SemOrg/semorg/booking/CompanyBookingT.java	CompanyBookingT
(H) Number of Parameters (avg/max per method)	0.64	1.229	6		/SemOrg/semorg/booking/BookingT.java	BookingT
(H) Number of Parameters (avg/max per method)	0.625	0.404	1		/SemOrg/semorg/booking/CompanyBookingListT.java	CompanyBookingListT
(H) Number of Parameters (avg/max per method)	0.647	0.478	1		/SemOrg/semorg/booking/BookingListT.java	BookingListT
(H) Number of Parameters (avg/max per method)	0.647	0.478	1		/SemOrg/semorg/booking/ClientBookingListT.java	ClientBookingListT
(H) Number of Parameters (avg/max per method)	0.766	1.462	10		/SemOrg/semorg/company/CompanyT.java	CompanyT
(H) Number of Parameters (avg/max per method)	0.818	1.714	10		/SemOrg/semorg/company/CompanyT.java	CompanyT
(H) Number of Parameters (avg/max per method)	0.643	0.479	1		/SemOrg/semorg/company/CompanyListT.java	CompanyListT
(H) Number of Parameters (avg/max per method)	0.892	1.603	10		/SemOrg/semorg/person/AssociateT.java	AssociateT
(H) Number of Parameters (avg/max per method)	0.888	0.556	7		/SemOrg/semorg/gui/MainFrame.java	MainFrame
(H) Number of Parameters (avg/max per method)	0.867	1.114	6		/SemOrg/semorg/classes/TimeT.java	modify
(H) Number of Parameters (avg/max per method)	0.545	0.782	4		/SemOrg/semorg/SemOrgListT.java	SemOrgListT
(H) Abstractness (avg/max per package/fragment)	0.031	0.003	0.25		/SemOrg/semorg/classes	
(H) Number of Interfaces (avg/max per package/fragment)	0	0	0			
(H) Different Coupling (avg/max per package/fragment)	7.125	6.133	22		/SemOrg/semorg/gui	
(H) Number of Children (avg/max per type)	48	0.276	1.83	17	/SemOrg/semorg/gui/AncestorPanel.java	
(H) Depth of Inheritance Tree (avg/max per type)	1.954	1.579	6		/SemOrg/semorg/gui/PersonPanel.java	

4th Workshop on SEE and RE

58

Cyclomatic Complexity Number for known "bad" functions

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Lines of Code (avg/max per type)	5018	28.839	50.934	325	/SemOrg/semorg/gui/PresenterPanel.java	
Lack of Cohesion of Methods (avg/max per type)		0.22	0.345	0.992	/SemOrg/semorg/gui/MainFrame.java	
McCabe Cyclomatic Complexity (avg/max per method)		1.348	0.936	8	/SemOrg/semorg/gui/PresenterPanel.java	setElement
(default package)		1.348	0.936	8	/SemOrg/semorg/gui/PresenterPanel.java	setElement
semorg.gui		1.320	0.896	8	/SemOrg/semorg/gui/PresenterPanel.java	setElement
semorg.booking		1.593	1.171	6	/SemOrg/semorg/booking/CompanyBookingList.java	get
CompanyBookingList.java		1.562	1.368	6	/SemOrg/semorg/booking/CompanyBookingList.java	get
CompanyBookingList						get
get		6				
find		4				
modify		2				
CompanyBookingList		1				
CompanyBookingList		1				
tableColumns		1				
toString		1				
create		1				
destroy		1				
add		1				
remove		1				
count		1				
setCompanyList		1				
getCompanyList		1				
setPresentatorkist		1				
getPresentatorkist		1				
BookingList.java		1.75	1.534	4	/SemOrg/semorg/booking/BookingList.java	get
ClientBookingList.java		1.529	1.334	6	/SemOrg/semorg/booking/ClientBookingList.java	get
ClientBookingList.java		2.045	1.224	5	/SemOrg/semorg/booking/ClientBookingList.java	toString
CompanyBookingList.java		1.812	0.982	3	/SemOrg/semorg/booking/CompanyBookingList.java	toString
BookingList.java		1.04	0.196	2	/SemOrg/semorg/booking/BookingList.java	BookingList
semorg.company		1.319	0.902	6	/SemOrg/semorg/company/CompanyList.java	get
semorg.person		1.418	1.078	6	/SemOrg/semorg/person/PersonList.java	get
semorg.presentation		1.394	1.042	6	/SemOrg/semorg/presentation/CompanyPresent...	get
semorg.senior		1.294	0.891	6	/SemOrg/semorg/senior/CarConductList.java	get
semorg.classes		1.211	0.681	5	/SemOrg/semorg/classes/ContactList.java	toString
semorg		1.182	0.987	4	/SemOrg/semorg/semorgapp.java	SemOrgapp
Number of Parameters (avg/max per method)		0.865	1.272	17	/SemOrg/semorg/presentation/ClientPresentati...	ClientPresentationT

4th Workshop on SEE and RE

59

Metrics for the project as a whole

Metric	Total	Mean	Std. Dev.	Maximum
Lines of Code (avg/max per method)	5018	4.694	12.945	20
Number of Static Methods (avg/max per type)	57	0.526	0.252	-
Allered Coupling (avg/max per package/fragment)		12.375	15.964	4
Normalized Distance (avg/max per package/fragment)		0.507	0.242	0.7
Number of Classes (avg/max per package/fragment)	174	21.75	43.614	13
Specialization Index (avg/max per type)		0.232	0.232	1
Instability (avg/max per package/fragment)		0.462	0.281	0.95
Number of Attributes (avg/max per type)	669	3.945	8.001	7
Number of Packages	8			
Weighted methods per Class (avg/max per type)	1441	8.282	10.522	4
Number of Overridden Methods (avg/max per type)	80	0.46	0.573	-
Handled Block Depth (avg/max per method)		1.204	0.52	-
Number of Static Attributes (avg/max per type)	7	0.04	0.277	-
Number of Methods (avg/max per type)	1012	5.816	7.056	3
Lines of Code (avg/max per type)	5018	28.839	50.934	325
Lack of Cohesion of Methods (avg/max per type)		0.22	0.345	0.992
McCabe Cyclomatic Complexity (avg/max per method)		1.348	0.936	8
Number of Parameters (avg/max per method)		0.865	1.272	17
Abstractness (avg/max per package/fragment)		0.031	0.063	0.2
Number of Interfaces (avg/max per package/fragment)		0	0	0
Efferent Coupling (avg/max per package/fragment)		7.125	6.133	2
Number of Children (avg/max per type)	48	0.276	1.83	1
Depth of Inheritance Tree (avg/max per type)		1.954	1.579	-

4th Workshop on SEE and RE

60

Metrics for the certain package

Metric	Total	Mean	Std. Dev.	Maximum
Lines of Code (avg/max per method)	2749	6.262	19.054	20
Number of Static Methods (avg/max per type)	0	0	0	0
Affluent Coupling	1			
Normalized Distance	0.043			
Number of Classes	137			
Specialization Index (avg/max per type)		0.244	0.41	1.1
Instability	0.957			
Number of Attributes (avg/max per type)	503	3.672	8.856	74
Weighted methods per Class (avg/max per type)	583	4.255	6.556	38
Number of Overridden Methods (avg/max per type)	39	0.277	0.449	
Number of Static Attributes (avg/max per method)		1.223	0.496	
Number of Static Attributes (avg/max per type)		0.002	0.205	
Number of Methods (avg/max per type)	439	3.204	3.934	3
Lines of Code (avg/max per type)	2749	20.066	52.052	321
Lack of Cohesion of Methods (avg/max per type)		0.124	0.302	0.99
McCabe Cyclomatic Complexity (avg/max per met...)		1.320	0.806	
Number of Parameters (avg/max per method)		0.888	0.556	
Abstractness	0			
Number of Interfaces	0			
Affluent Coupling	22			
Number of Children (avg/max per type)	17	0.124	1.447	3
Depth of Inheritance Tree (avg/max per type)		1.934	1.752	

4th Workshop on SEE and RE

61

Metrics for the certain class

Metric	Total	Mean	Std. Dev.	Maximum
Lines of Code (avg/max per method)	123	12.3	23.031	8
Number of Static Methods	0			
Specialization Index	0.6			
Number of Attributes	14			
Weighted methods per Class	13			
Number of Overridden Methods	1	1.3	0.458	
Number of Static Attributes	0			
Number of Methods	10			
Lines of Code	123	0.839		
Lack of Cohesion of Methods		1.3	0.458	
McCabe Cyclomatic Complexity (avg/max per met...)		0.7	0.458	
Number of Parameters (avg/max per method)		0		
Number of Children	0			
Depth of Inheritance Tree	6			

4th Workshop on SEE and RE

62

Metrics results can be exported to XML

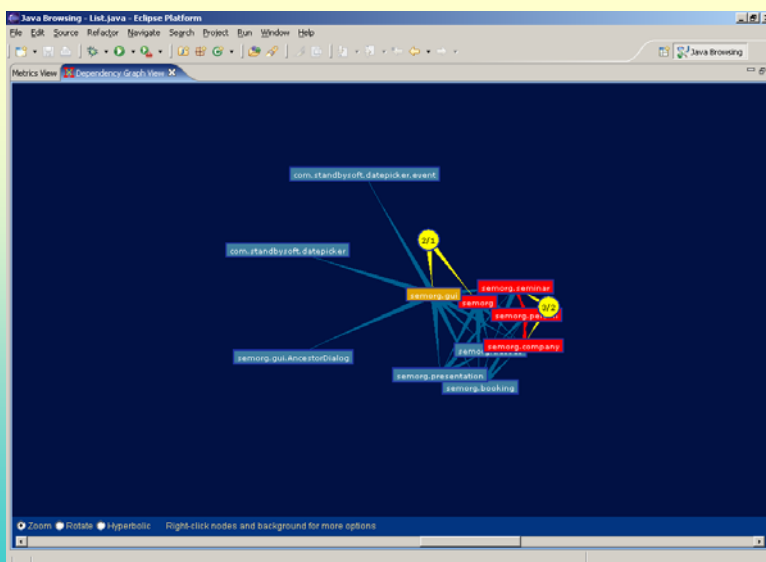
The screenshot shows the Eclipse IDE interface. A 'Save As' dialog is open, allowing the user to save a file. In the background, the 'Metrics View' is active, displaying a table of code metrics for the current file. The metrics include:

Metric	Total	Mean	Std. Dev.	Maximr
avg/max per method	2749	6.762	19.694	20
cls (avg/max per type)	0	0	0	0
l	1	0.043		1
lst	137			
pkg/max per type		0.244	0.41	1
0.967				
avg/max per type)	503	3.672	8.856	7
Class (avg/max per type)	583	4.255	6.556	3
Methods (avg/max per type)	38	0.277	0.448	
1.223				
Number of Static Attributes (avg/max per type)	3	0.022	0.255	
Number of Methods (avg/max per type)	439	3.204	3.934	3
Lines of Code (avg/max per type)	2749	20.066	52.052	20
ts risk of Cohesion of Methods (avg/max per type)	0.124	0.302	0.96	
M McCabe Cyclomatic Complexity (avg/max per met...	1.328	0.886		
Number of Parameters (avg/max per method)	0.000	0.956		
Abstractness	0			
Number of Interfaces	0			
Effort Coupling	22			
Number of Children (avg/max per type)	17	0.124	1.447	1
Length of Inheritance Tree (avg/max per type)	1.934	1.752		

4th Workshop on SEE and RE

63

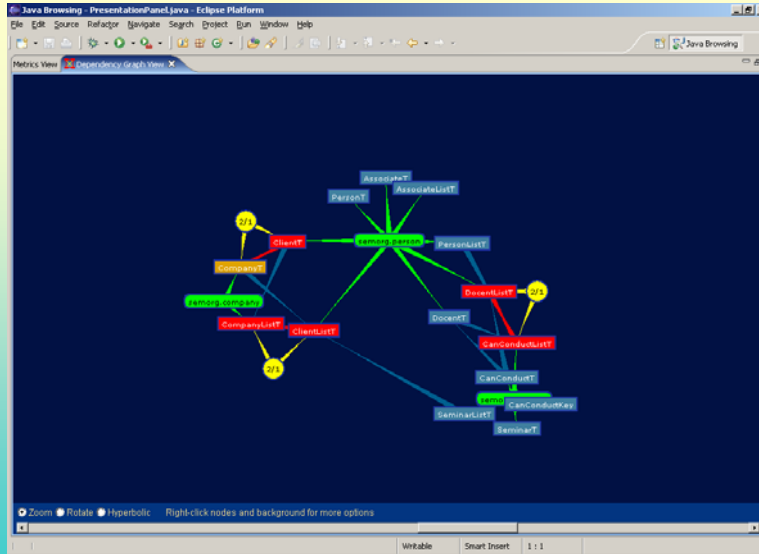
Dependency graph of a project



4th Workshop on SEE and RE

64

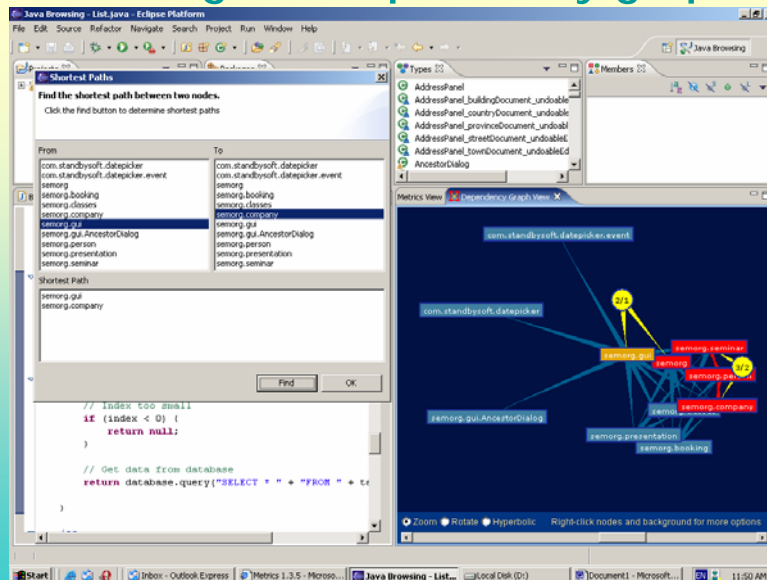
More detailed dependency graph



4th Workshop on SEE and RE

65

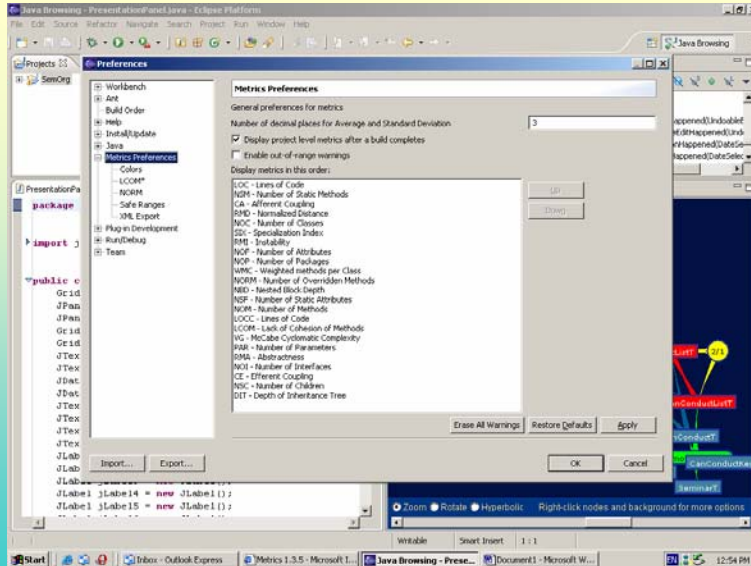
Path finding in a dependency graph



4th Workshop on SEE and RE

66

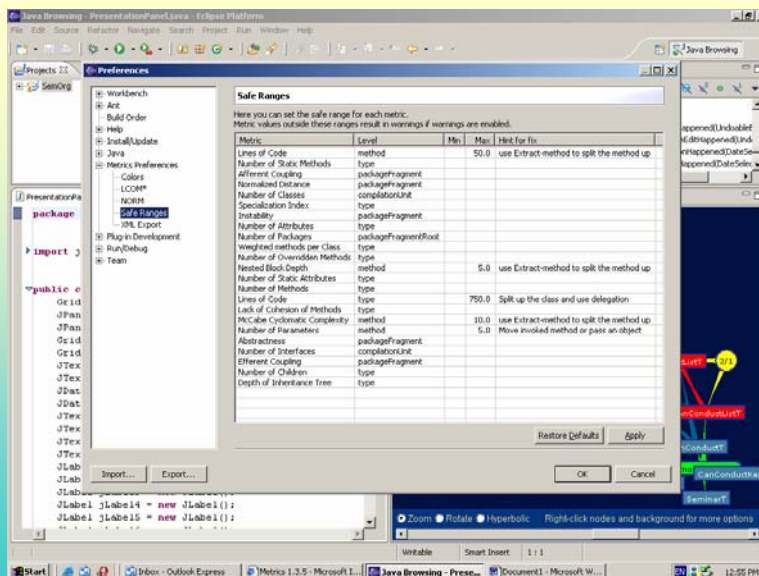
Adjusting the preferences for metrics 1/2



4th Workshop on SEE and RE

67

Adjusting the preferences for metrics 2/2



4th Workshop on SEE and RE

68

