

# CSC 7003 : Basics of Software Engineering

**J Paul Gibson, A207**

`paul.gibson@int-edu.eu`

<http://www-public.it-sudparis.eu/~gibson/Teaching/CSC7003/>

## **Metrics**

<http://www-public.it-sudparis.eu/~gibson/Teaching/CSC7003/L9-Metrics.pdf>

# Software Metrics Come In Many Different Kinds

Metrics are expensive to collect, report, and analyze so if no one is using a metric, producing it is a waste of time and money.

If a metric does not have a customer, it should not be produced. Customers may include:

- **Functional Management:** Interested in applying greater control to the software development process, reducing risk and maximizing return on investment.
- **Project Management:** Interested in being able to accurately predict and control project size, effort, resources, budgets and schedules. Interested in controlling the projects they are in charge of and communicating facts to their management.
- **Software Engineers/Programmers:** The people that actually do the software development. Interested in making informed decisions about their work and work products. These people are responsible for collecting a significant amount of the data required for the metrics program.

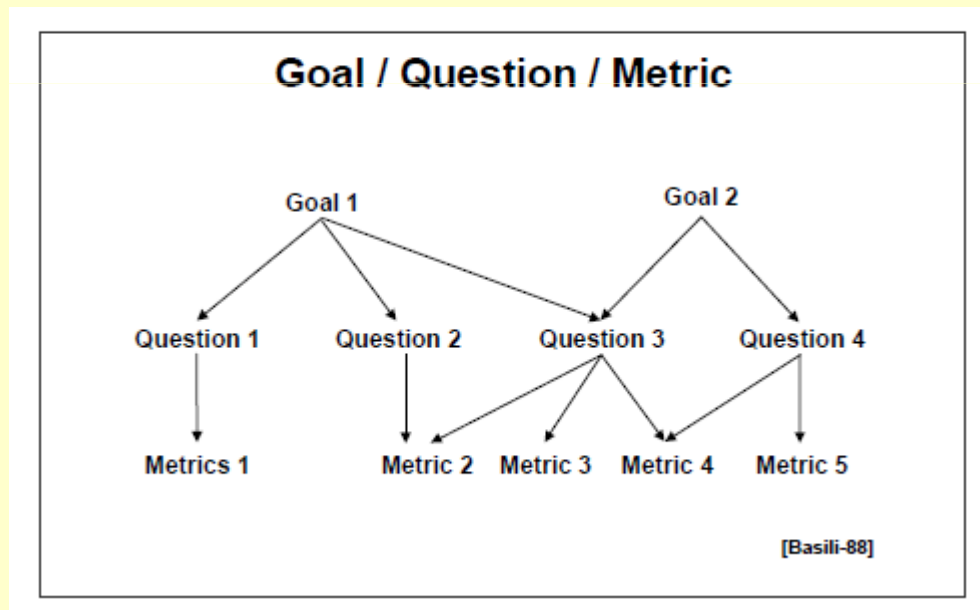
# Software Metrics Come In Many Different Kinds

- **Test Managers/Testers:** The people responsible for performing the verification and validation activities. Interested in finding as many new defects as possible in the time allocated to testing and in obtaining confidence that the software works as specified. These people are also responsible for collecting a significant amount of the required data.
- **Specialists:** Individuals performing specialized functions (e.g., Marketing, Software Quality Assurance, Process Engineering, Software Configuration Management, Audits and Assessments, Customer Technical Assistance). Interested in quantitative information upon which they can base their decisions, finding and recommendations.
- **Clients/Users:** Interested in on-time delivery of high quality software products and in reducing the over-all cost of ownership.

# Software Metrics Come In Many Different Kinds

In this module, we will focus on metrics to measure the quality of code.

Following the *Goal Question Metric Approach* of **Basili, Caldiera and Rombach** we need to link our goals – better code (easier to maintain/extend/reuse/understand/...) to specific metrics



Fortunately, there exist many standard metrics for different programming paradigms:

- OO
- Functional
- Procedural, etc...

**QUESTION: What metrics do you know about?**

## Metrics: background reading

*Software Modeling and Measurement: The Goal/Question/Metric Paradigm*, **V. Basili**, 1992

*Using metrics to evaluate software system maintainability*,  
**Coleman, D.; Ash, D.; Lowther, B.; Oman, P.**, 1992

*A validation of object-oriented design metrics as quality indicators*,  
**Basili, V.R.; Briand, L.C.; Melo, W.L.**, 1996

*Object-oriented metrics that predict maintainability*, **Wei Li and Sallie Henry**, 2003

# Introduction to Metrics Tools for programmers

General tool that you should know how to use and use in your own development:

[Eclipse Plug-In](#) – Java Metrics for static analysis

Important To Know About:

Stand-alone specialist *static analysis* tools, eg, JDepend for OO design metrics

(see <http://clarkware.com/software/JDepend.html>)

*Dynamic Analysis* of java code as it executes, eg, \*J

(see <http://www.sable.mcgill.ca/starj/> ) for event traces

# Eclipse Plug-In– Java Metrics

## Quick Instructions:

- 1) **Make sure you have Eclipse 3.1 (or later) installed,**  
otherwise  
if Eclipse not on your machine then download and install  
latest version from:

<http://www.eclipse.org/downloads/>

if outdated version on your machine then automatically  
update the version from menus:

**help->software updates**

- 2) **Follow initial plugin installation procedures at**  
<http://metrics.sourceforge.net/>

# Eclipse Plug-In– Java Metrics

Generating Metrics for a project:

**Download sample code to be analysed** from, eg:

<http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/Dijkstra.java>

**Store this code in a new directory** (**Dijkstra**, eg)

**Create a new ‘Dijkstra Project’ using Eclipse menus:**

**File->New->Project->From Existing Source** (use **Dijkstra** directory)

**Follow further plugin installation procedures at**

**<http://metrics.sourceforge.net/>**

**until you get metrics readout for Dijkstra.java**

...

# Eclipse Plug-In– Java Metrics

You should see that one of the metrics gives a **warning**:

The screenshot shows the Eclipse IDE interface with the 'Metrics - Dijkstra - Number of Overridden Methods (avg/max per type)' window open. The table below represents the data shown in the Metrics window.

Metric	Total	Mean	Std. D...	Maximum	Resource causing Maxi...	Method
Number of Overridden Methods (avg/max per type)	3	1	1,41	3	/Dijkstra/Dijkstra.java	
Number of Attributes (avg/max per type)	31	10,33	3,3	14	/Dijkstra/Dijkstra.java	
Number of Children (avg/max per type)	0	0	0	0	/Dijkstra/Dijkstra.java	
Number of Classes (avg/max per packageFragment)	3	3	0	3	/Dijkstra	
Method Lines of Code (avg/max per method)	310	13,48	13,13	48	/Dijkstra/Dijkstra.java	input_graph
Number of Methods (avg/max per type)	23	7,67	10,84	23	/Dijkstra/Dijkstra.java	
Nested Block Depth (avg/max per method)		1,87	0,99	4	/Dijkstra/Dijkstra.java	step2
Depth of Inheritance Tree (avg/max per type)		2,33	1,89	5	/Dijkstra/Dijkstra.java	
Number of Packages	1					
Afferent Coupling (avg/max per packageFragment)		0	0	0	/Dijkstra	
Number of Interfaces (avg/max per packageFragment)	0	0	0	0	/Dijkstra	
McCabe Cyclomatic Complexity (avg/max per meth...		3,96	3,22	12	/Dijkstra/Dijkstra.java	step2
Total Lines of Code	390					
Instability (avg/max per packageFragment)		1	0	1	/Dijkstra	
Number of Parameters (avg/max per method)		1,26	1,36	5	/Dijkstra/Dijkstra.java	drawArrow
Lack of Cohesion of Methods (avg/max per type)		0,25	0,36	0,76	/Dijkstra/Dijkstra.java	
Efferent Coupling (avg/max per packageFragment)		0	0	0	/Dijkstra	
Number of Static Methods (avg/max per type)	0	0	0	0	/Dijkstra/Dijkstra.java	
Normalized Distance (avg/max per packageFragmer)		0	0	0	/Dijkstra	
Abstractness (avg/max per packageFragment)		0	0	0	/Dijkstra	
Specialization Index (avg/max per type)		0,22	0,31	0,65	/Dijkstra/Dijkstra.java	
Weighted methods per Class (avg/max per type)	91	30,33	42,9	91	/Dijkstra/Dijkstra.java	

## Eclipse Plug-In– Java Metrics

Using this metric information, which methods in `Dijkstra.java` risk to be too complex for maintenance and extension?

What could one do to change the code in order to manage this complexity?

Your company has internal quality metric that no method should have more than 40 lines of code. Update the Eclipse metric preferences to automatically provide a warning when this value is surpassed ([Window->Preferences->Metric Preferences->Safe Ranges](#)). Re-run the metrics analysis on the `Dijkstra.java` code (you can choose rebuild automatically in the menus so that this is done for you).

What do you see? ...

# Eclipse Plug-In– Java Metrics

You should see that method lines of code metric gives a **warning**:

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left shows the project structure. The main editor displays the code for the `step1()` method in `Dijkstra.java`. The Metrics view at the bottom shows a table of metrics for the `Dijkstra` class.

Metric	Total	Mean	Std. D...	Maximum	Resource causing Maxi...	Method
Number of Overridden Methods (avg/max per type)	3	1	1,41	3	/Dijkstra/Dijkstra.java	
Number of Attributes (avg/max per type)	31	10,33	3,3	14	/Dijkstra/Dijkstra.java	
Number of Children (avg/max per type)	0	0	0	0	/Dijkstra/Dijkstra.java	
Number of Classes (avg/max per packageFragment)	3	3	0	3	/Dijkstra	
Method Lines of Code (avg/max per method)	310	13,48	13,13	48	/Dijkstra/Dijkstra.java	input_graph
Number of Methods (avg/max per type)	23	7,67	10,84	23	/Dijkstra/Dijkstra.java	
Nested Block Depth (avg/max per method)		1,87	0,99	4	/Dijkstra/Dijkstra.java	step2
Depth of Inheritance Tree (avg/max per type)		2,33	1,89	5	/Dijkstra/Dijkstra.java	
Number of Packages	1					
Afferent Coupling (avg/max per packageFragment)		0	0	0	/Dijkstra	
Number of Interfaces (avg/max per packageFragment)	0	0	0	0	/Dijkstra	
McCabe Cyclomatic Complexity (avg/max per metho		3,96	3,22	12	/Dijkstra/Dijkstra.java	step2
Total Lines of Code	390					
Instability (avg/max per packageFragment)		1	0	1	/Dijkstra	
Number of Parameters (avg/max per method)		1,26	1,36	5	/Dijkstra/Dijkstra.java	drawArrow
Lack of Cohesion of Methods (avg/max per type)		0,25	0,36	0,76	/Dijkstra/Dijkstra.java	
Efferent Coupling (avg/max per packageFragment)		0	0	0	/Dijkstra	
Number of Static Methods (avg/max per type)	0	0	0	0	/Dijkstra/Dijkstra.java	
Normalized Distance (avg/max per packageFragmer		0	0	0	/Dijkstra	
Abstractness (avg/max per packageFragment)		0	0	0	/Dijkstra	
Specialization Index (avg/max per type)		0,22	0,31	0,65	/Dijkstra/Dijkstra.java	
Weighted methods per Class (avg/max per type)	91	30,33	42,9	91	/Dijkstra/Dijkstra.java	
Number of Static Attributes (avg/max per type)	0	0	0	0	/Dijkstra/Dijkstra.java	

## Eclipse Plug-In– Java Metrics

Can you see that it is the **input\_graph** method that is causing the warning?

What could one do to change the code in order to manage this complexity risk?

Which lines of code (LOC) is it actually counting?

- Comments?
- WhiteSpace?
- Number of expression separators ‘;’s?

Make changes to the code, rebuild and recheck metrics data to verify that you know which LOC are counted.

# Eclipse Plug-In– Java Metrics

Export the metric data as an xml file:

Use metric view drop down menu **export xml**



to give:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Metrics scope="Dijkstra.java" type="CompilationUnit" date="2007-03-06" xmlns="http://metrics.sourceforge.net/2003/Metrics-First-Flat">
- <Metric id="NORM" description="Number of Overridden Methods">
- <Values per="type" total="3" avg="1" stddev="1,41" max="3">
  <Value name="Dijkstra" source="Dijkstra.java" package="(default package)" value="3" />
  <Value name="Edge" source="Dijkstra.java" package="(default package)" value="0" />
  <Value name="Node" source="Dijkstra.java" package="(default package)" value="0" />
</Values>
</Metric>
...
- <Metric id="NSF" description="Number of Static Attributes">
- <Values per="type" total="0" avg="0" stddev="0" max="0">
  <Value name="Dijkstra" source="Dijkstra.java" package="(default package)" value="0" />
  <Value name="Edge" source="Dijkstra.java" package="(default package)" value="0" />
  <Value name="Node" source="Dijkstra.java" package="(default package)" value="0" />
</Values>
</Metric>
</Metrics>
```

## Eclipse Plug-In– Java Metrics

One can use metrics to compare programs that provide different solutions to the same problem.

For example, in graph algorithms there are numerous solutions (algorithms) to the *minimum spanning tree problem*.

We can find implementations of these in Java and – using the Eclipse metric plug-in – compare their ‘complexity’

**TO DO: Carry out such an analysis for 2 such solutions -**

Prim (<http://www-b2.is.tokushima-u.ac.jp/~ikedasuuri/dijkstra/Prim.shtml>, eg)

Kruskal (<http://www-b2.is.tokushima-u.ac.jp/~ikedasuuri/kruskal/Kruskal.shtml>, eg)