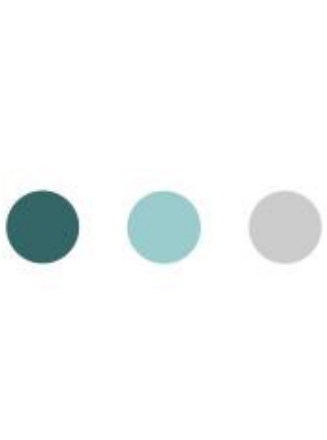


Software Complexity

Ali Fatolahi
TA of SEG4913
University of Ottawa, SITE
Fall 2007

Agenda

- A Brief Discussion
- Some theoretical background
  - Some slides and information by:
    - Alan Williams, SEG3203, Summer 2007, SITE, UOttawa
      - http://www.site.uottawa.ca/~awilliam/seg3203/Process_Metrics.ppt
    - Misbah Islam, SEG3300, Summer 2007, SITE, UOttawa
      - http://www.site.uottawa.ca/~misbah/seg3300b/LecICAdvancedTopicsEstimation.doc
- Practical Insight
  - Some tools
  - An Example

Software Quality Metrics

- Measure
  - quantitative indication of the extent, amount, dimension, capacity or size of some attribute of a product or process
- Measurement
  - act of determining a measure
- Metric (IEEE Std. 610.12)
  - quantitative measure of the degree to which a system, component, or process possesses a given attribute
  - may relate individual measures
- Indicator
  - metric(s) that provide insight into software process, project, or product

Objective

- Use of metrics is vital for:
  - Estimates of development effort:
    - Schedule
    - Cost
    - Size of team
  - Progress tracking
    - Deviations from schedule, cost
    - Determining when corrective action is needed.
- A history of project metrics can help an organization improve its process for future product releases, and improve software quality.

Types of Metrics

- Product Metrics:
  - Measurements about the size / complexity of a product.
  - Help to provide an estimate of development effort.
- Process Metrics
  - Progress tracking.
  - Staffing pattern over the life cycle
  - Productivity
  - Determine the effectiveness of various process stages, and provide targets for improvement.
- Operational Metrics:
  - Collection of data after product release.

Problems of Measurement

- Data collection:
  - How to collect data without impacting productivity?
  - Usually need to collect data via tools, for accuracy, currency.
- Inappropriate focus:
  - Developers target “ideal” metric values instead of focusing on what is appropriate for the situation.
  - Perception that metrics will be used to rank employees.

Product metrics

- Objective: obtain a sense of the size or complexity of a product for the purpose of:
  - Scheduling:
    - How long will it take to build?
    - How much test effort will be needed?
    - When should the product be released?
  - Reliability:
    - How well will it work when released?

Lines of Code

- This is the “classic” metric for the size of a product.
  - Often expressed as KLOC (thousand lines of code)
  - Sometimes counted as (new + modified) lines of code for subsequent releases, when determining development effort.
- Advantages:
  - Easy to measure with tools.
  - Appears to have a correlation with effort.
- Disadvantages:
  - “What is a line of code?”
    - Comments? Declarations? Braces?
  - Not as well correlated with functionality.
  - Complexity of statements in different programming languages.
  - Automated code generation from GUI builders, parser generators, UML, etc.

“Hello, world” in C and COBOL

```c
#include <stdio.h>
int main(void)
{
   printf("Hello
   World");
}
```

- 5 lines of C, versus 17 lines of COBOL

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. HELLOWORLD.
000300
000400*
000500 ENVIRONMENT DIVISION.
000600 CONFIGURATION SECTION.
000700 SOURCE-COMPUTER. RM-COBOL.
000800 OBJECT-COMPUTER. RM-COBOL.
000900
001000 DATA DIVISION.
001100 FILE SECTION.
001200
100000 PROCEDURE DIVISION.
100100
100200 MAIN-LOGIC SECTION.
100300 BEGIN.
100400 DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
100500 DISPLAY "Hello world!" LINE 15 POSITION 10.
100600 STOP RUN.
100700 MAIN-LOGIC-EXIT.
100800 EXIT.
```

Function Point Analysis

- Defined in 1977 by Alan Albrecht of IBM
- Details: http://www.nesma.nl/english/menu/frsfpa.htm
- Attempts to determine a number that increases for a system with:
  - More external interfaces
  - More complex

Function Point Analysis

- Steps to performing function point analysis:
  1. Identify user functions of the system (5 types of functions are specified).
  2. Determine the complexity of each function (low, medium, high).
  3. Use the function point matrix to determine a value for each function in step 1, based on its complexity assigned in step 2.
  4. Rate the system on a scale of 0 to 5 for each of 14 specified system characteristics.
     - Examples: communications, performance,
  5. Combine the values determined in steps 3 and 4 based on specified weightings.

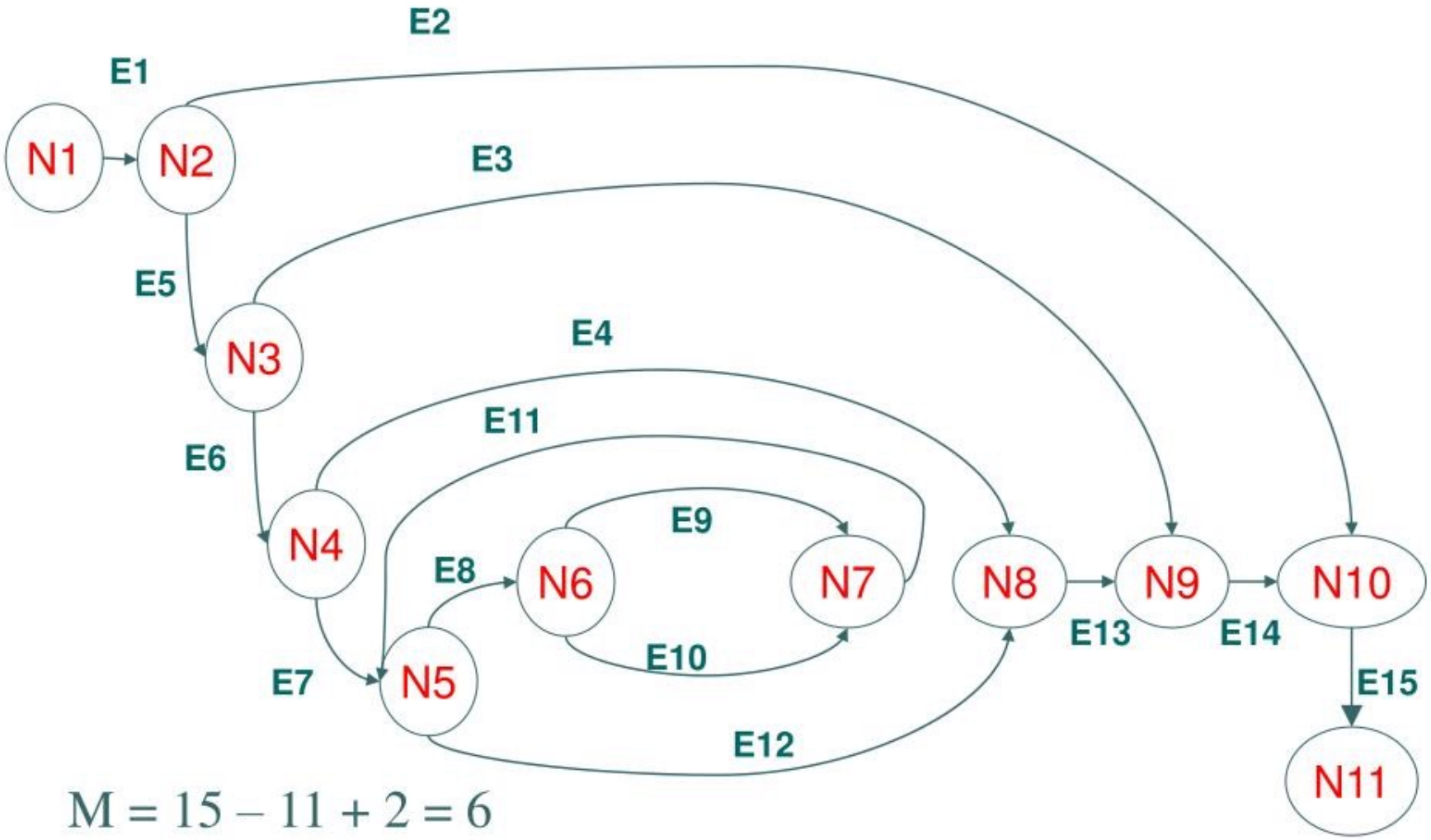
Using Function Points Analysis

- To use function point analysis for estimation, the following information is needed:
  - Standard rate of development per function point.
    - Collected from previous projects.
  - Special modifications (plus or minus) to the standard rate for the project to be estimated;
    - Factors: new staff, new development tools, etc.
  - Include additional factors not based on project size.

McCabe's Cyclomatic Complexity

- Based on a flow graph model of software.
- Measures the number of linearly independent paths through a program.
- Uses graph components:
  - E: Number of edges in flow graph
  - N: Number of nodes in flow graph
- Cyclomatic complexity: $M = E - N + 2$
- Claim: if M > 10 for a module, it is "likely" to be error prone.

Calculating Cyclomatic Complexity



$$M = 15 - 11 + 2 = 6$$

Cyclomatic Complexity

- With a cyclomatic complexity of 6, there are 6 independent paths through the flow graph:
  1. E1-E2-E15
  2. E1-E5-E3-E14-E15
  3. E1-E5-E6-E4-E13-E14-E15
  4. E1-E5-E6-E7-E12-E13-E14-E15
  5. E1-E5-E6-E7-E8-E10-E11-E12-E13-E14-E15
  6. E1-E5-E6-E7-E8-E9-E11-E12-E13-E14-E15

Cyclomatic Complexity

- Advantages:
  - It is very easy to compute as illustrated in the example.
  - Unlike other complex measurements, it can be computed immediately in the development cycle (which makes it agile friendly).
  - It provides a good indicator of the ease of code maintenance.
  - It can help focus testing efforts.
  - It makes it easy to find complex code for formal review.

Subsystem Complexity

- **Two Aspects**
  - **complexity within each component**
  - **structural complexity of the relationships between components**

Subsystem Complexity =

$$\sum_{1}^{n} (\text{Component's Internal Complexity}) + \text{Structural Complexity}$$

$$\text{Structural Complexity} = (1 / n) \sum_{1}^{n} (\text{Fan-In})_j \text{ **2}$$

Software Package Metrics

- From R.C. Martin, *Agile Software Development: Principles, Patterns, and Practices*
- Metrics determined from a package of classes in an object-oriented language
  - Number of classes and interfaces
  - Coupling:
    - Number of other packages that depend on classes in this package (CA)
    - Number of packages depended on by classes in this package (CE).
  - Ratio of abstract classes to total classes (0 to 1)
  - …

Additional OO metrics

- Class dependencies: classes used, classes used by
- Degree of inheritance:
  - Number of subclasses for a class
  - Maximum number of inheritances from root of tree to a class.
- Number of method calls invoked to respond to a message.

Some Tools

- PMD
  - Java Code Quality – Open Source
- CyVis
  - Java Code Complexity Visualizer
- devMetrics
  - Complexity Meausure for .Net
- Reflector
  - .Net Analyzer
- Klockwork
  - Automated Quality Assurance for Java and C/C++

CyVis

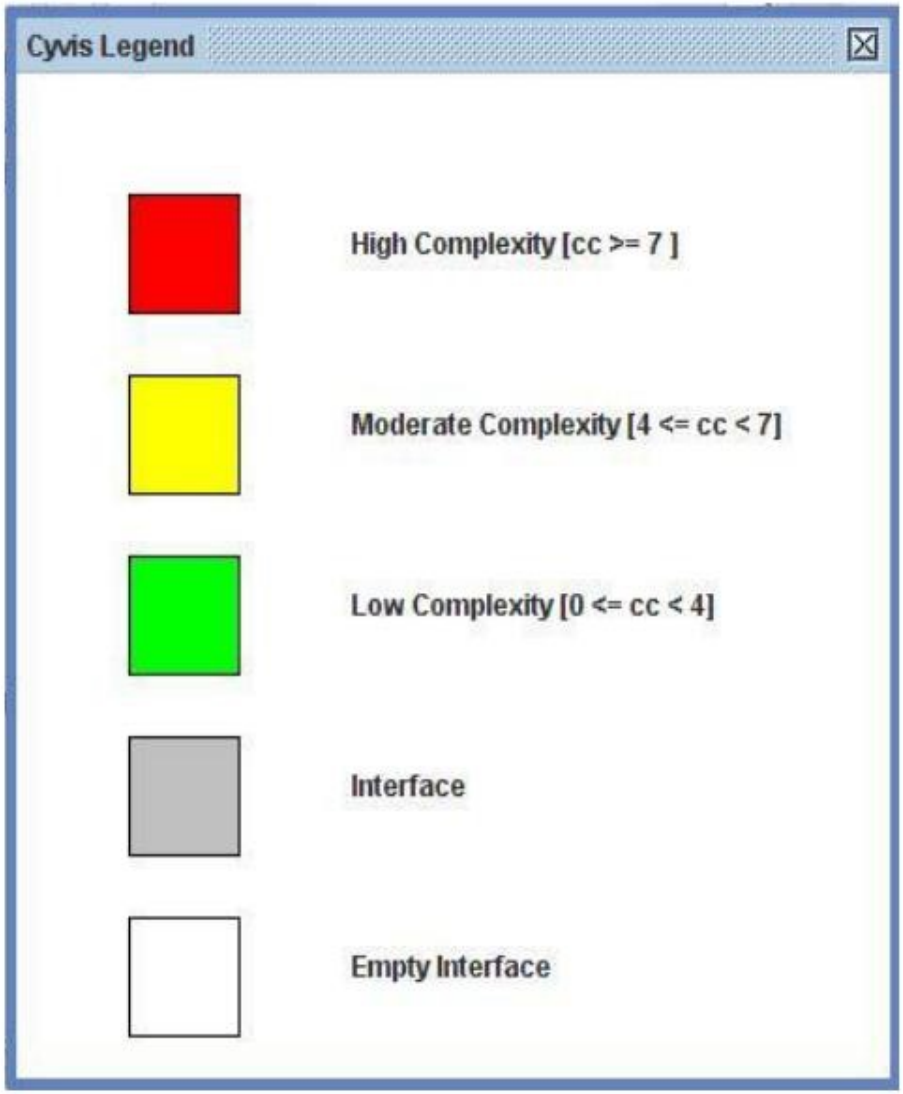
- Available from:
  - http://cyvis.sourceforge.net/
- Collects Metrics from class files
  - no need for the source code
- integrated with Ant builder
- Detailed Metrics at Project, Package and Class Levels.
- Metrics shown in tables and charts.
- Customizable coloured highlighting in charts for better data visualisation.
- HTML & Text Reports generation, or raw metrics export in XML format.

CyVis (Contd.)

- Three modes:
  - Graphical
  - Command Line
  - Ant Task

CyVis (Contd.)

- Sample
  - I ran it on my StateMachineTransformer application
  - A part of my PhD research
  - Using the graphics mode
  - Running on the jar file



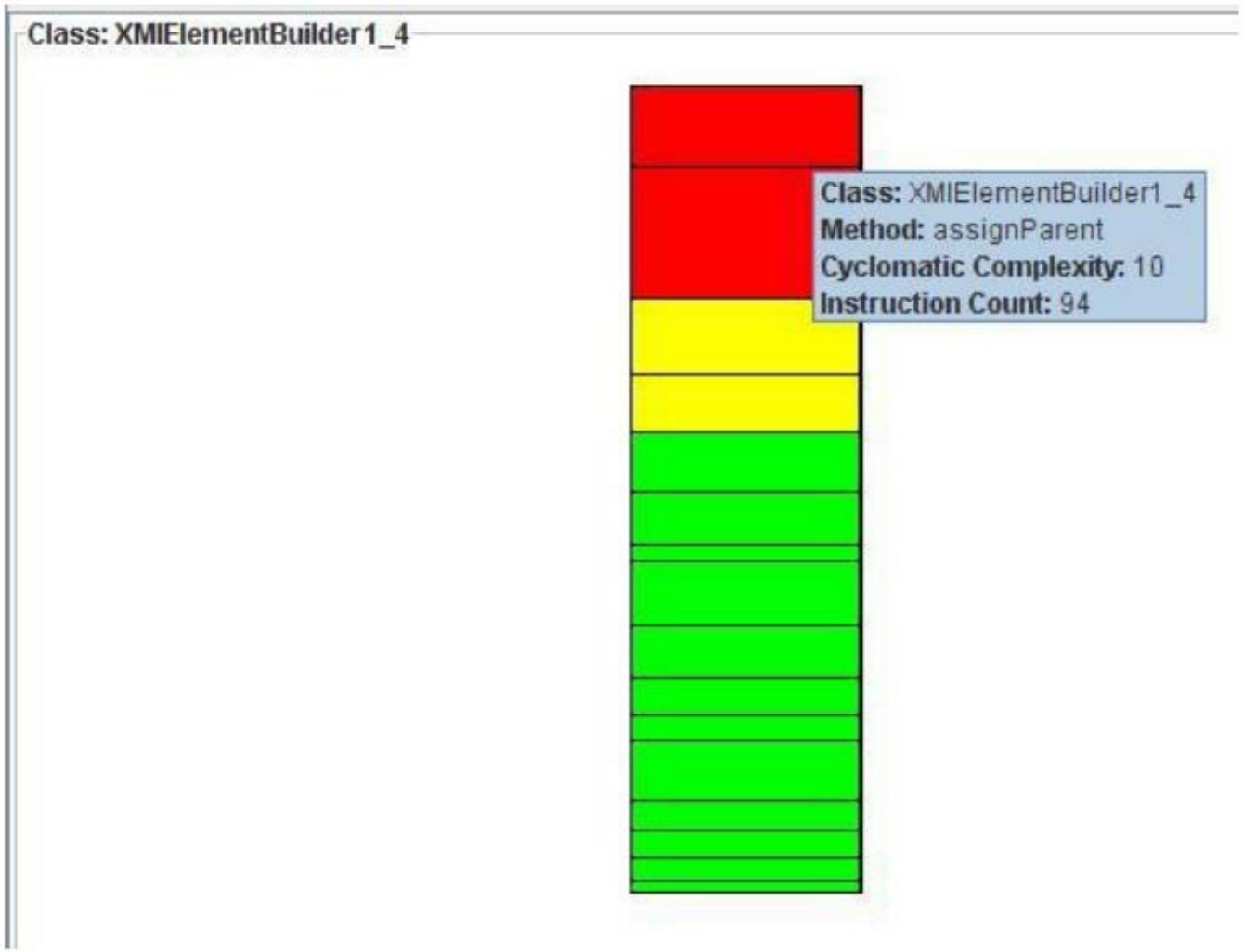
CyVis (Contd.) Sample Results

CyVis (Contd.) Sample Results

Methods in Class: xmi.XMIElementBuilder1_4

Method Name	Cyclomatic Complexity	Length
assignParent	10	94
getStateXMI	8	149
getModelXMI	4	87
getNamedElementXMI	4	68
getClassXMI	3	67
getUseCaseXMI	3	61
assignParent	3	19
getTransitionXMI	2	75
getParameterXMI	2	60
getOperationXMI	2	41
getSignalEventXMI	2	30
getStateMachineXMI	1	68
getCallEventXMI	1	35
getStereotypeXMI	1	31
getPackageXMI	1	27
<init>	1	12

CyVis (Contd.) Sample Results



CyVis (Contd.) Sample

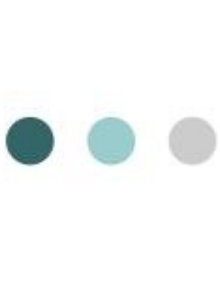
```
if (childType.compareToIgnoreCase("stereotype")==0) {
return ancestor.findChildByName("UML:Namespace.ownedElement");
}

if (childType.compareToIgnoreCase("signalevent")==0) {
return ancestor.findChildByName("UML:Namespace.ownedElement");
}

if (childType.compareToIgnoreCase("state")==0) {
return ancestor.findChildByName("UML:CompositeState");
}

if (childType.compareToIgnoreCase("transition")==0) {
return ancestor.findChildByName("UML:StateMachine.transitions");
}

if (childType.compareToIgnoreCase("statemachine")==0) {
return ancestor.findChildByName("UML:UseCase");
}
```

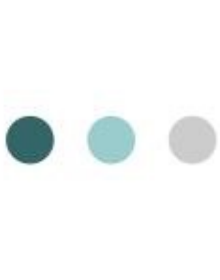


PMD

- PMD
  - Java Code Quality – Open Source
  - Available from
    - http://pmd.sourceforge.net/
- Integrated with:
  - JDeveloper, Eclipse, JEdit, JBuilder, BlueJ CodeGuide, NetBeans/Sun Java Studio Enterprise/Creator, IntelliJ IDEA, TextPad, Maven, Ant, Gel, JCreator, and Emacs.
- Eclipse: http://pmd.sf.net/eclipse

PMD (Contd.)

- Looks for:
  - Possible bugs - empty try/catch/finally/switch statements
  - Dead code - unused local variables, parameters and private methods
  - Suboptimal code - wasteful String/StringBuffer usage
  - Overcomplicated expressions - unnecessary if statements, for loops that could be while loops
  - Duplicate code - copied/pasted code means copied/pasted bugs



PMD (Contd.)

- How it works?
  - checks source code against rules and produces a report.
  - The Report is filled with RuleViolations, and those get printed out in XML or HTML or whatever
- Does not report cyclomatic complexity
- In fact, it didn't report any other problem with my code

devMetrics

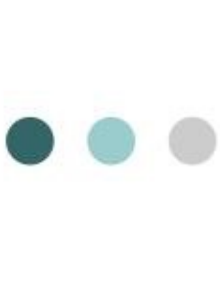
- Comes with a free community edition
- Available from:
  - http://www.anticipatingminds.com/content/products/devMetrics/devMetrics.aspx
- Adds a tools submenu to .Net environment

Klockworks

- Available from:
  - http://www.klocwork.com/
- Not free but available for a 30 days trial
- Supports Java and C/C++
- No need to test cases
  - over 200 different security vulnerabilities and quality defects
- Eclipse
  - Name: KDJ Update Site
    - http://developer.klocwork.com/UpdateSite/win32
    - http://developer.klocwork.com/UpdateSite/linux
    - http://developer.klocwork.com/UpdateSite/solaris

Klockworks (Contd.)

Description	Resource	Location	Severity
UIR.CONSTR: Reading field this.id prior to explicit initialization in constructor	uml/NamedElement.java	59	Error (3)
SV.RANDOM: Use of insecure Random number generator Random	util/RandomGenerator.java	20	Review (9)
SV.RANDOM: Use of insecure Random number generator Random	util/RandomGenerator.java	11	Review (9)
STRCON.LOOP: Using append for string action in a loop	ui/TransitionScreen.java	82	Suggestion (7)
REDUN.NULL: Variable 'operation' used in expression, but it is always 'null' in this context.	mapping/Mapping.java	201	Unexpected ...
NPDS.VAR: Potential NullPointerException by dereferencing variable operations, having its	mapping/Mapping.java	193	Investigate (5)
NPDS.VAR: Potential NullPointerException by dereferencing variable elements, having its v	mapping/Mapping.java	190	Investigate (5)
NPDS.EXPR: Potential NullPointerException by dereferencing result of call to javax.swing.J	ui/TransitionScreen.java	79	Investigate (5)
JD.VNU.SI: Variable 'transition' assigned only once and never read	ui/FlatScreenWizard.java	135	Warning (6)
JD.VNU.SI: Variable 'run' assigned only once and never read	main/Run.java	17	Warning (6)
JD.VNU.SI: Variable 'commandWizard' assigned only once and never read	mapping/Mapping.java	100	Warning (6)
JD.UNCAUGHT: Method 'main.Run.main(String[])' does not catch exceptions '[java.lang.Nu	main/Run.java	18	Warning (6)
JD.UN.PMET: Private method 'findOperation' is unused.	mapping/Mapping.java	185	Warning (6)
JD.RC.EXPR.CHECK: Test expression 'nodeName!=0' is always 'true' because 'nodeName!	ui/TransitionButtonListener.java	18	Suggestion (7)
ECC.EMPTY: Empty catch clause	util/XML.java	52	Investigate (5)
ECC.EMPTY: Empty catch clause	ui/RefineWindow.java	74	Investigate (5)
ECC.EMPTY: Empty catch clause	ui/FlatScreenWizard.java	50	Investigate (5)
CMP.STR: Comparing strings "" and trim().$RET with ==	ui/TransitionButtonListener.java	28	Investigate (5)
CMP.STR: Comparing strings "" and trim().$RET with ==	ui/TransitionButtonListener.java	18	Investigate (5)
CMP.STR: Comparing strings "" and trim().$RET with ==	ui/TransitionButtonListener.java	15	Investigate (5)
CMP.STR: Comparing strings "" and trim().$RET with ==	ui/ModelTree.java	50	Investigate (5)



Finally

- Do not be idealistic
- Do not try to kill complexity
  - Just keep it manageable
- Do not waste your time on complexity more than required
  - But give it some time to report it to make sure you get your mark!
- Use tools please
- Any Question?