Introduction to Static Analysis

Dependable Software Laboratory

Static Analysis

- Static analysis is the process of examining source code prior to compilation
  - Without executing

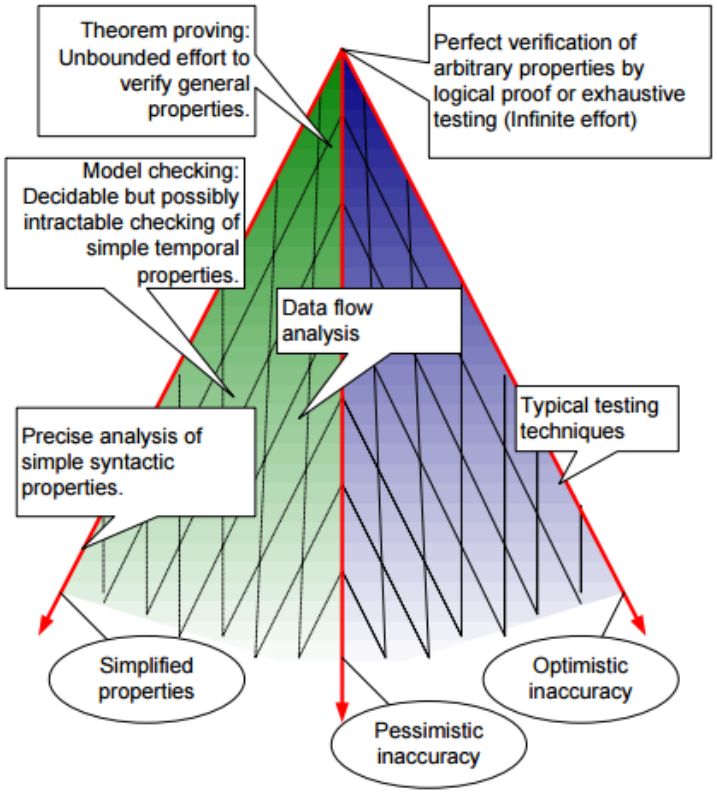
- Static analysis can diagnose for:
  - **Quality aspects** such as maintainability, reliability, understandability and complexity
  - **Testing** issues
  - **Coding standard** compliance issues
  - Best programming practices and **unsafe programming** constructs and **coding defects**

Static Analysis

- Analyze the program without executing it
  - Doesn't depend on having good test cases or even any test cases
  - Generally, doesn't know what your software is *supposed to do*
  - Looks for violations of reasonable programming
  - Not a replacement for testing
  - Very good at finding problems on **untested paths**
  - But many defects can't be found with static analysis
  - False alarm occurs
    - Generally 30%

A position of the Static Analysis in Verification

- Verification Trade-off Dimensions



- Optimistic inaccuracy
  - We may accept some programs that do not possess the property.
  - It may not detect all violations.
  - Example: Testing
- Pessimistic inaccuracy
  - It is not guaranteed to accept a program even if the program does possess the property being analyzed, because of false alarms.
  - Example: Automated program analysis
- Simplified properties
  - It reduces the degree of freedom by simplifying the property to check.
  - Example: Model Checking

Static Analysis

- Static analysis can be divided 3 levels

- **Level 1: syntax checking of the source code**
  - Rule checking, coding style checking

- **Level 2: quality analysis with translated source code to CFG/DFG form**
  - Sematic analysis, complexity analysis

- **Level 3: static analysis, analyzing critical errors which can be issued during execution**
  - Divided by zero, NULL pointer, Etc.

Level 1

- Rule checking, coding style checking
- Syntax checking by IDE (e.g. *eclipse, visual studio*) is a kind of static analysis
- Several kinds of rules
  - Simple rule checking
    - *E.g. Brace location, tab, Etc.*
  - Safe coding rule checking
    - T. A., R.W. WITTY, "SAFE PROGRAMMING," 1978
      - Safe specification and programming(coding) is the simplest way to improve software reliability
      - Proposing several rules for safe software (safe programming)
      - *E.g. infinite loop checking with counter, protecting buffer overflow code*
- It is useful for coding style checking when working as a team
  - Readability
  - Maintainability

```
public void test(){


}

public void test1()
{

}
```

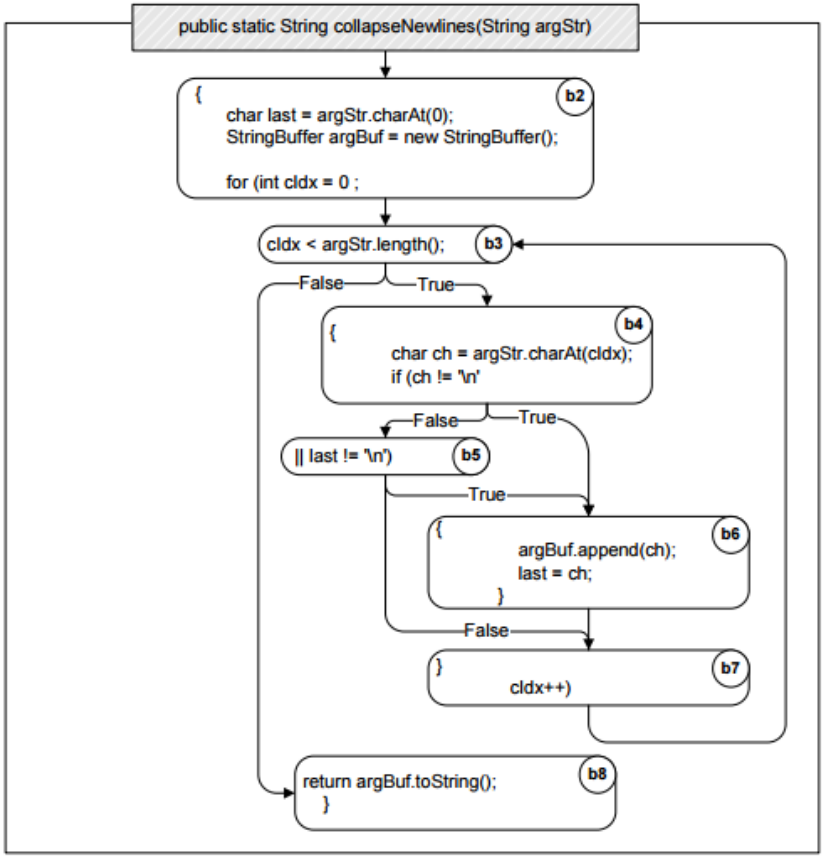
Level 2

- Kinds of complexity, coverage, depend metrics
  - Using CFG (Control Flow Graph), DFD (Data Flow Graph), Etc.

```
public static String collapseNewlines(String argStr)
  {
     char last = argStr.charAt(0);
     StringBuffer argBuf = new StringBuffer();

     for (int cIdx = 0 ; cIdx < argStr.length(); cIdx++)
     {
        char ch = argStr.charAt(cIdx);
        if (ch != '\n' || last != '\n')
        {
           argBuf.append(ch);
           last = ch;
        }
     }

     return argBuf.toString();
  }
```

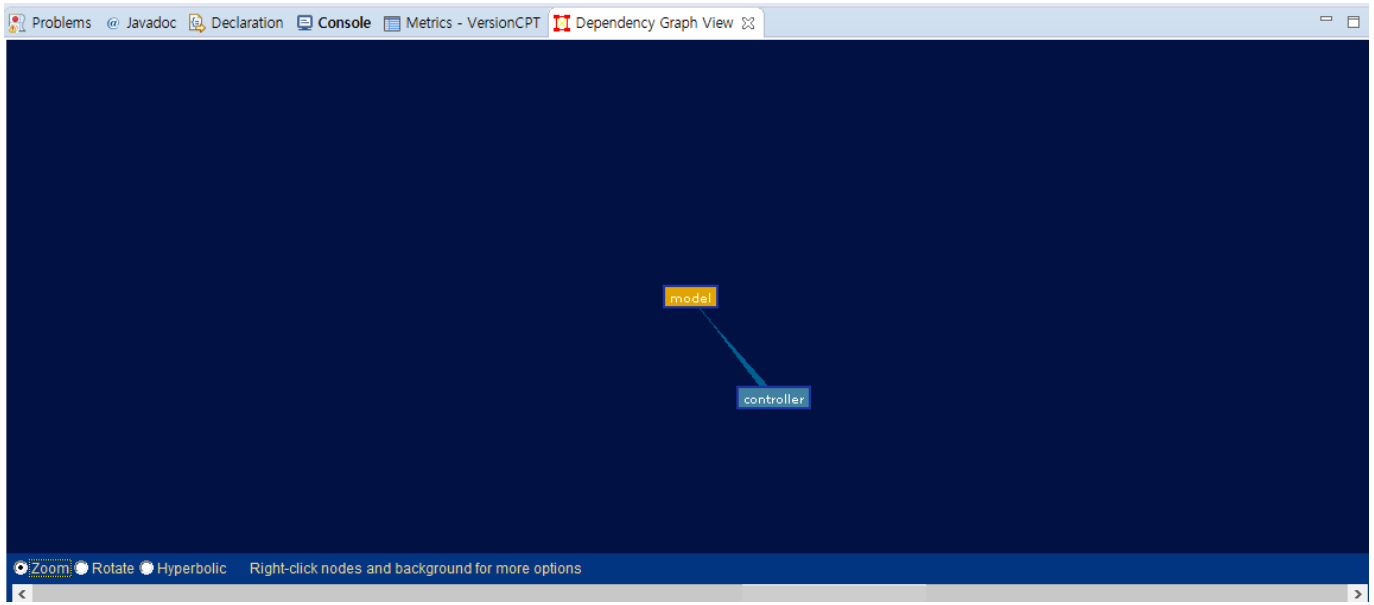


Level 2

- Source code information and dependency graph
  - *E.g.* Cyclomatic Complexity

Problems | Javadoc | Declaration | Console | Metrics - src - McCabe Cyclomatic Complexity (avg/max per method) | Dependency Graph View

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per method)		1.7	2.407	23	/VersionCPT/src/controller/ServerThread.java	analyzeMSG
Number of Parameters (avg/max per method)		0.655	0.878	5	/VersionCPT/src/controller/AdminVer.java	chkAuth
Nested Block Depth (avg/max per method)		1.355	0.746	5	/VersionCPT/src/controller/AdminVer.java	chkAuth
Afferent Coupling (avg/max per packageFragment)		1.5	1.5	3	/VersionCPT/src/model	
Efferent Coupling (avg/max per packageFragment)		1.5	1.5	3	/VersionCPT/src/controller	
Instability (avg/max per packageFragment)		0.5	0.5	1	/VersionCPT/src/controller	
Abstractness (avg/max per packageFragment)		0	0	0	/VersionCPT/src/controller	
Normalized Distance (avg/max per packageFragment)		0.5	0.5	1	/VersionCPT/src/model	
Depth of Inheritance Tree (avg/max per type)		1.091	0.287	2	/VersionCPT/src/controller/ServerThread.java	
Weighted methods per Class (avg/max per type)	187	17	7.828	28	/VersionCPT/src/controller/ServerThread.java	
Number of Children (avg/max per type)	0	0	0	0	/VersionCPT/src/controller/AdminProject.java	
Number of Overridden Methods (avg/max per type)	1	0.091	0.287	1	/VersionCPT/src/controller/ServerThread.java	
Lack of Cohesion of Methods (avg/max per type)		0.596	0.261	0.894	/VersionCPT/src/model/Version.java	
Number of Attributes (avg/max per type)	54	4.909	2.678	11	/VersionCPT/src/model/Choice.java	
Number of Static Attributes (avg/max per type)	0	0	0	0	/VersionCPT/src/controller/AdminProject.java	
Number of Methods (avg/max per type)	108	9.818	6.422	24	/VersionCPT/src/model/Version.java	
Number of Static Methods (avg/max per type)	2	0.182	0.575	2	/VersionCPT/src/controller/MainServer.java	
Specialization Index (avg/max per type)		0.061	0.192	0.667	/VersionCPT/src/controller/ServerThread.java	
Number of Classes (avg/max per packageFragment)	11	5.5	0.5	6	/VersionCPT/src/controller	
Number of Interfaces (avg/max per packageFragment)	0	0	0	0	/VersionCPT/src/controller	
Number of Packages	2					
Total Lines of Code	917					
Method Lines of Code (avg/max per method)	549	4.991	12.482	119	/VersionCPT/src/controller/ServerThread.java	analyzeMSG

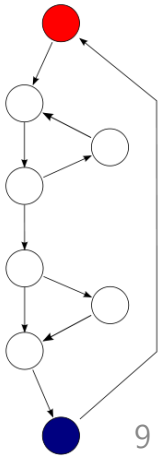


Cyclomatic Complexity

- Cyclomatic complexity is a software metric (measurement), used to indicate the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code. It was developed by Thomas J. McCabe, Sr. in 1976.

- Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. Cyclomatic complexity may also be applied to individual functions, modules, methods or classes within a program.

- One testing strategy, called basis path testing by McCabe who first proposed it, is to test each linearly independent path through the program; in this case, the number of test cases will equal the cyclomatic complexity of the program.



Cyclomatic Complexity

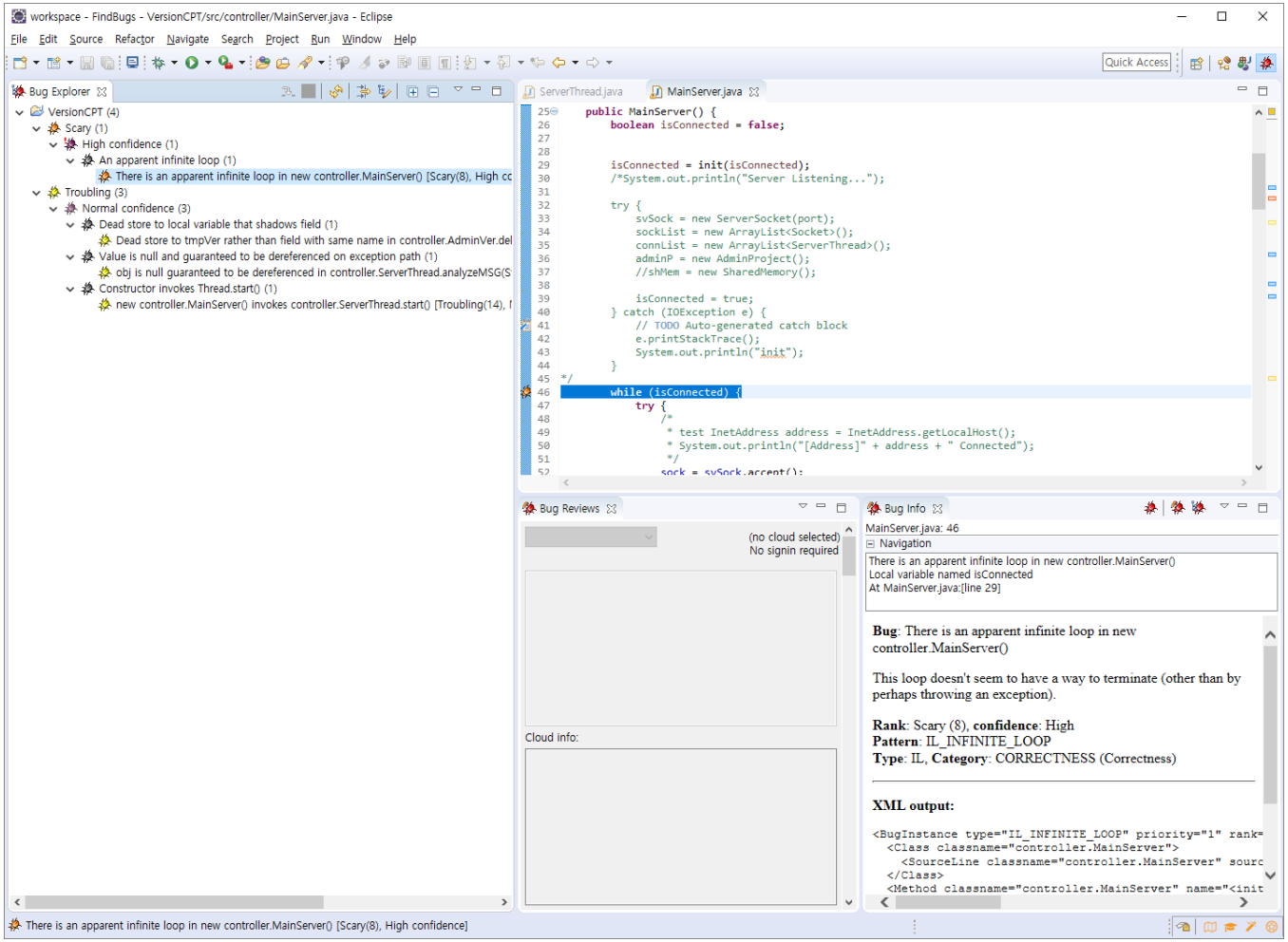
- **Lower is better.** A McCabe complexity under 5 is good, from 5-10 is OK, and over 10 is too complex. A high flow complexity may be a symptom of a function which does too much or has low cohesion (does to many different things). But don't take these numbers too seriously -- you may have comprehensible control flow despite high numbers. For example, one large *switch* statement can be clear to understand, but can dramatically increase the count.

- **23 is too high**

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per method)		1.7	2.407	23	/VersionCPT/src/controller/ServerThread.java	analyzeMSG
src		1.7	2.407	23	/VersionCPT/src/controller/ServerThread.java	analyzeMSG
controller		2.614	3.582	23	/VersionCPT/src/controller/ServerThread.java	analyzeMSG
ServerThread.java		9.333	9.741	23	/VersionCPT/src/controller/ServerThread.java	analyzeMSG
ServerThread		9.333	9.741	23	/VersionCPT/src/controller/ServerThread.java	analyzeMSG
analyzeMSG	23					

Level 3

- Static analysis
  - analyzing critical errors which can be issued during execution
  - Without execution(compile)



Automated Static Analysis

- There are several tools for static analysis of source codes
  - Commercial
    - Powerful tool is too expensive
  - Open source
    - Several open source tools exist also

Tools

Level 1 : Rule Checking , Coding Style Checking - 코드의 syntax 를 분석

checkstyle(java)	: 코딩 표준 준수 검사 도구
StyleCop(C#)	: 코딩 표준 준수 검사 도구
N'SIQ CppStyle(c/c++)	: NHN 의 표준 코딩 스타일 체크 도구
cpplint(c++)	: 구글의 코딩 스타일 가이드를 위한 코딩 규칙 검사 도구
PMD(java)	: Dead code, 빈 조건문, 반복되는 코드 검사 등을 지원
sonar(c/c++, java 등)	: 코딩 규칙, 중복 검사 등을 지원 (다양한 기능이 있는 플랫폼)
QA-C(c)	: MISRA C 를 포함한 코딩 표준과 코딩 룰 준수 검사 도구

Tools

Level 2: OOO analysis - 코드를 CFG/DFG 등의 형태로 변환하여 각종 품질 관련 분석을 수행

eclipsemetrics : 소스코드 복잡도 분석 도구 in Eclipse, 그래프로 결과 확인

javancss(java) : 소스코드 복잡도 분석 도구

cobertura(java) : 소스코드 복잡도 분석 및 커버리지 측정

MALPAS(c, ada, PLM) : 분석을 위해 directed graphs 와 regular algebra 를 사용하여 프로그램을 표현. 자동화 도구를 사용해 프로그램의 구조를 묘사하는 것이 가능. Code 가 specification 을 만족하는 것을 보여주기 위한 formal proof 지원. Nuclear, aerospace, defence 분야의 safety critical application 의 correctness 를 보여주기 위해 사용되고 있다.

ECLAIR(c/c++) : formal methods-based static analysis

BLAST(c/c++) : lazy abstraction 기반 C 프로그램 model checker

Sotoarc/Sotograph(c/c++, java) : 구조를 시각화하여 의존성과 참조 문제 등을 확인하는 도구

Tools

Level 3 : static analysis - 코드에 대한 정적분석을 통해 실제 execution 時발행할 수 있는 중요한 오류(Divide by Zero, Null Pointer 등)를 예측

(대부분의 도구가 비슷한 정적 분석을 수행한다.)

Findbugs(java) : 소스코드에 문제가 될 수 있는 부분을 찾는 정적 분석 도구.
(예로 integer 를 나누기 할 경우 double 이나 float 로 cast 되는 부분을 알려준다.)

cppcheck(c/c++) : regular expression 분석을 통해 소스코드의 문제점을 분석

Fluctuat(c, ada) : 부동소수점에 관한 검사에 특화된 분석 도구

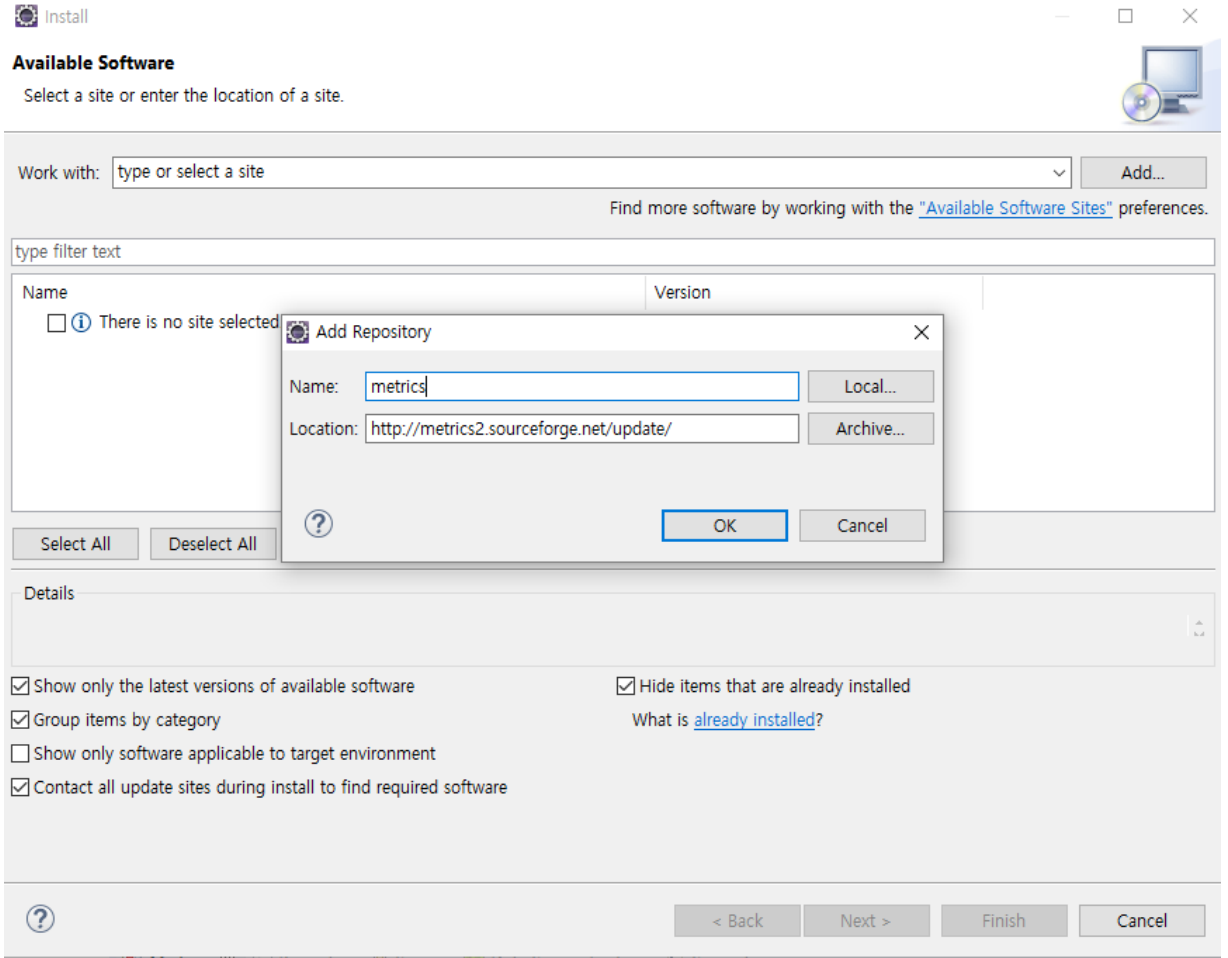
DevPartner(java, .NET) : 통합도구(code coverage, dead lock detection, code quality 등)

Coverity(c/c++,c#,java) : 정적 분석 도구 리더 중 하나로 open source에서 제공하는 기본적인 기능을 대부분 가지고 있으며, coverage 까지 확인 가능한 도구

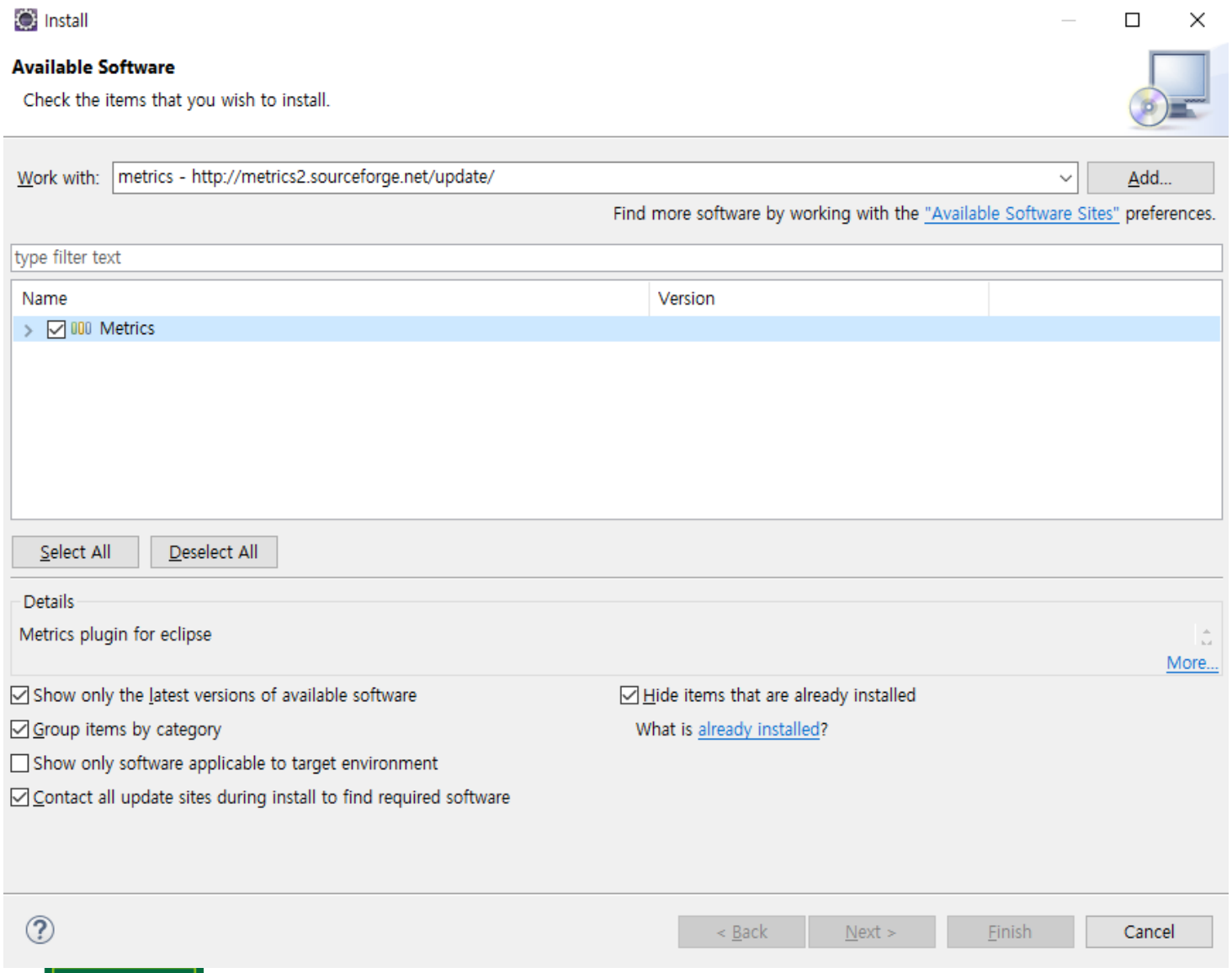
Eclipse Metrics Plugin

- Level 2

- Install
  - Help -> Install New Software -> Add
    -> input the location (http://metrics2.sourceforge.net/update/)

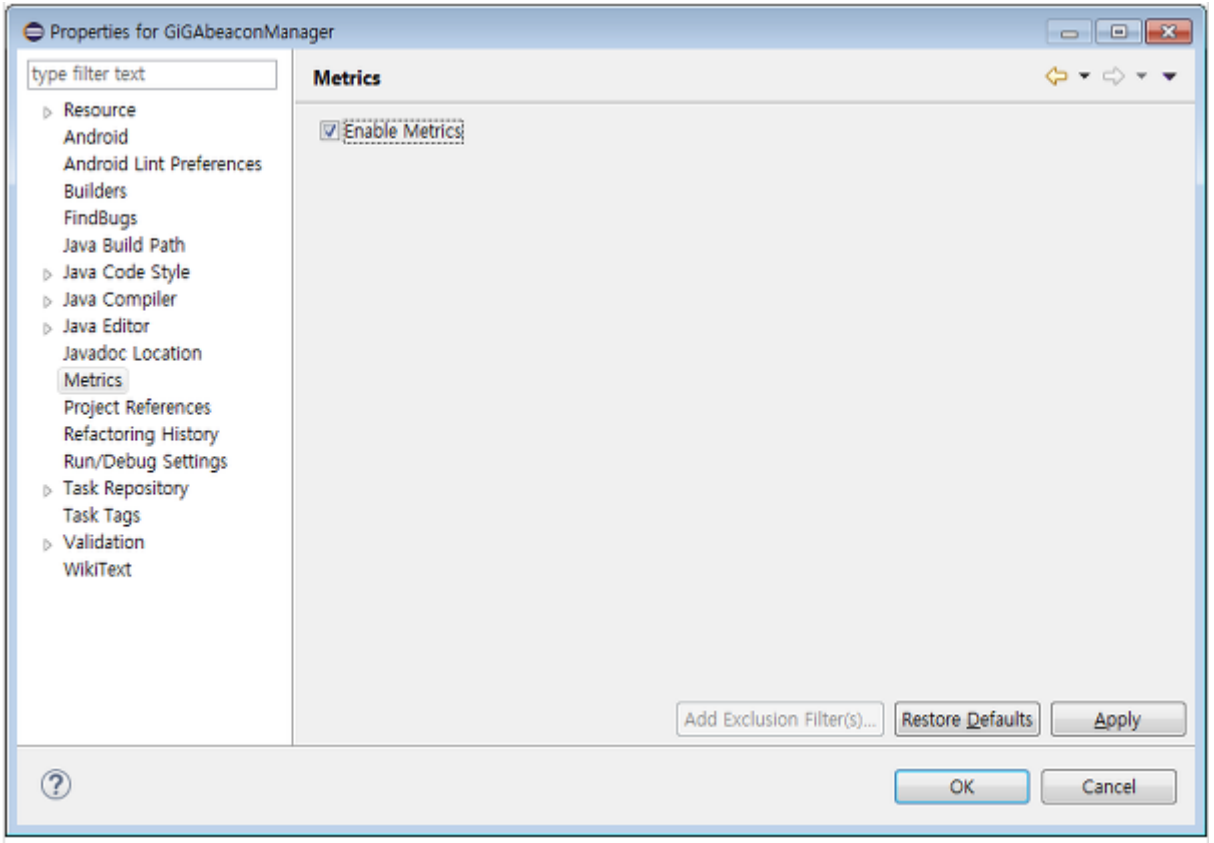


Eclipse Metrics Plugin



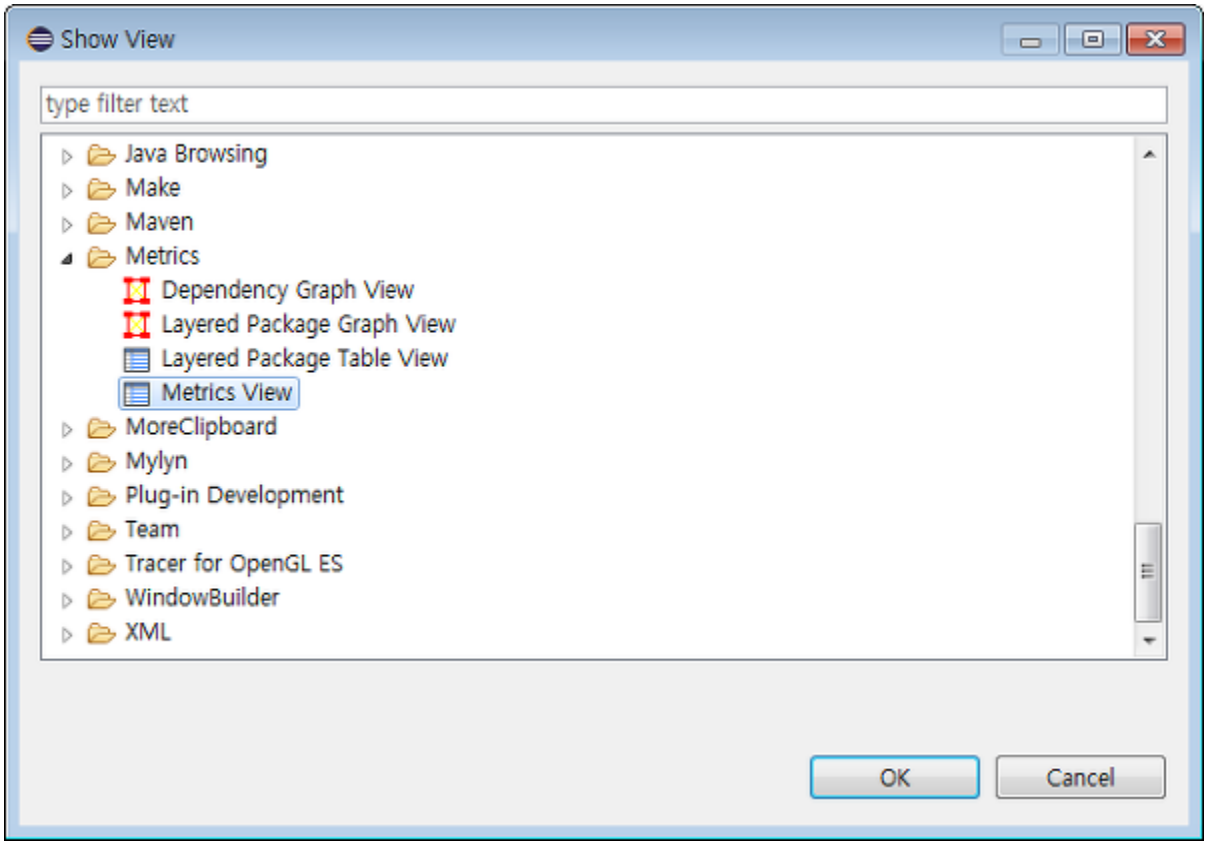
Eclipse Metrics Plugin

- 각 project -> properties -> Metrics -> enable

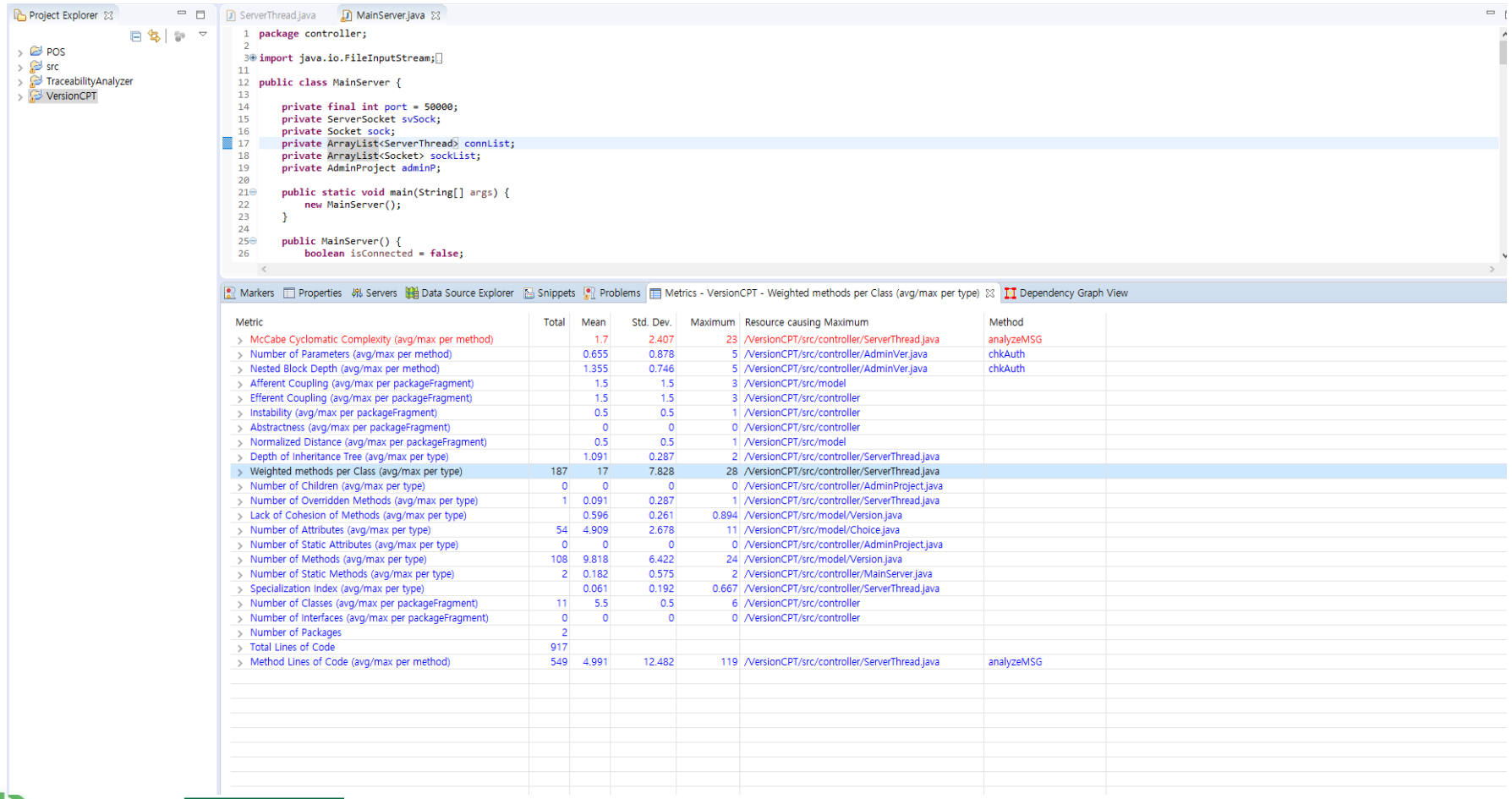


Eclipse Metrics Plugin

- Window -> Show View -> Metrics -> Metrics View



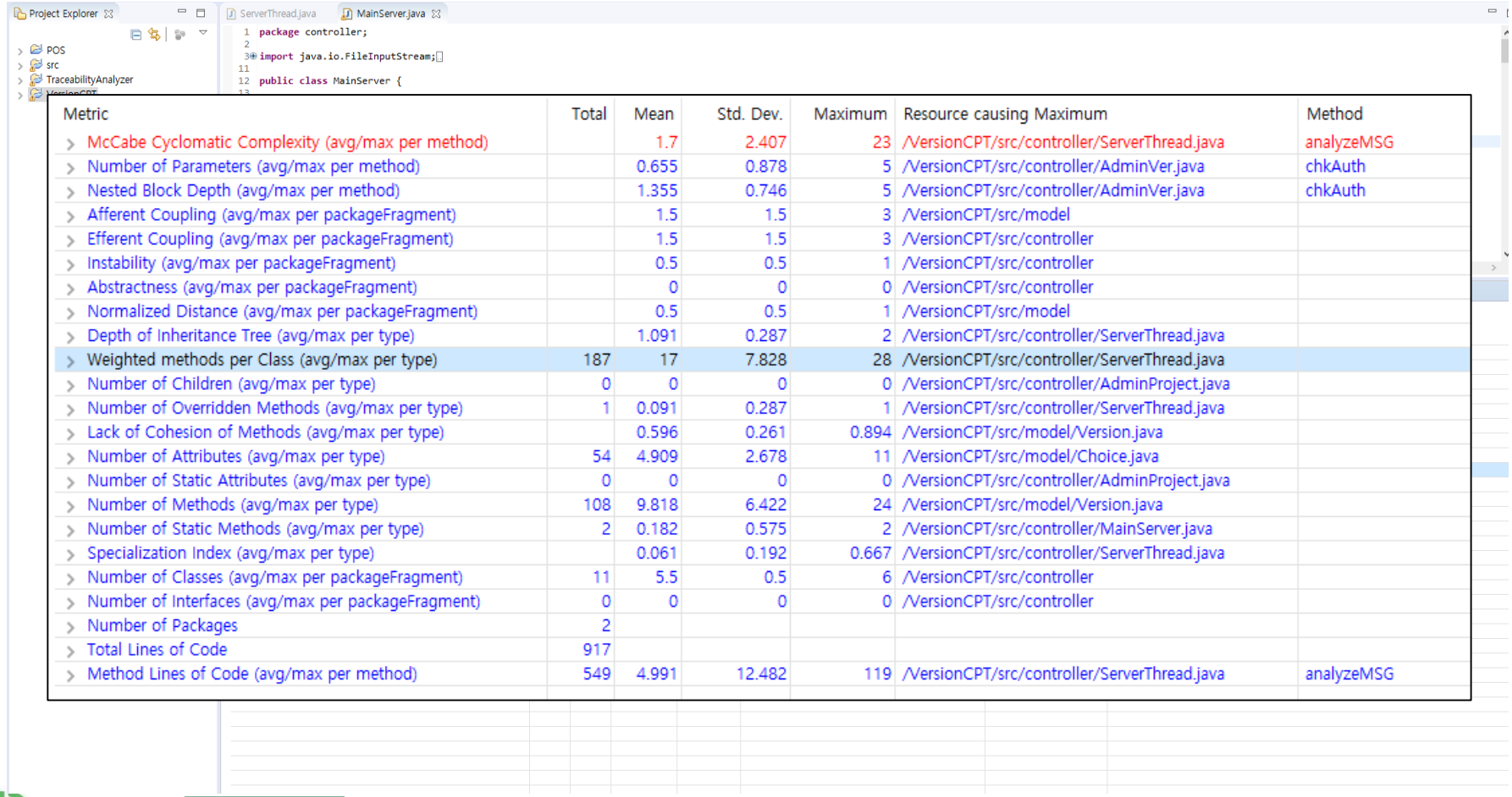
Eclipse Metrics Plugin



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per method)		1.7	2.407	23	/VersionCPT/src/controller/ServerThread.java	analyzeMSG
Number of Parameters (avg/max per method)		0.655	0.878	5	/VersionCPT/src/controller/AdminVer.java	chkAuth
Nested Block Depth (avg/max per method)		1.355	0.746	5	/VersionCPT/src/controller/AdminVer.java	chkAuth
Afferent Coupling (avg/max per packageFragment)		1.5	1.5	3	/VersionCPT/src/model	
Efferent Coupling (avg/max per packageFragment)		1.5	1.5	3	/VersionCPT/src/controller	
Instability (avg/max per packageFragment)		0.5	0.5	1	/VersionCPT/src/controller	
Abstractness (avg/max per packageFragment)		0	0	0	/VersionCPT/src/controller	
Normalized Distance (avg/max per packageFragment)		0.5	0.5	1	/VersionCPT/src/model	
Depth of Inheritance Tree (avg/max per type)		1.091	0.287	2	/VersionCPT/src/controller/ServerThread.java	
Weighted methods per Class (avg/max per type)	187	17	7.828	28	/VersionCPT/src/controller/ServerThread.java	
Number of Children (avg/max per type)	0	0	0	0	/VersionCPT/src/controller/AdminProject.java	
Number of Overridden Methods (avg/max per type)	1	0.091	0.287	1	/VersionCPT/src/controller/ServerThread.java	
Lack of Cohesion of Methods (avg/max per type)		0.596	0.261	0.894	/VersionCPT/src/model/Version.java	
Number of Attributes (avg/max per type)	54	4.909	2.678	11	/VersionCPT/src/model/Choice.java	
Number of Static Attributes (avg/max per type)	0	0	0	0	/VersionCPT/src/controller/AdminProject.java	
Number of Methods (avg/max per type)	108	9.818	6.422	24	/VersionCPT/src/model/Version.java	
Number of Static Methods (avg/max per type)	2	0.182	0.575	2	/VersionCPT/src/controller/MainServer.java	
Specialization Index (avg/max per type)		0.061	0.192	0.667	/VersionCPT/src/controller/ServerThread.java	
Number of Classes (avg/max per packageFragment)	11	5.5	0.5	6	/VersionCPT/src/controller	
Number of Interfaces (avg/max per packageFragment)	0	0	0	0	/VersionCPT/src/controller	
Number of Packages	2					
Total Lines of Code	917					
Method Lines of Code (avg/max per method)	549	4.991	12.482	119	/VersionCPT/src/controller/ServerThread.java	analyzeMSG

Eclipse Metrics Plugin

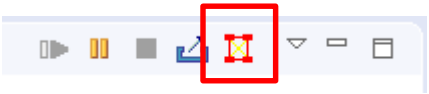
- Complexity, code line, 상속 관계 등

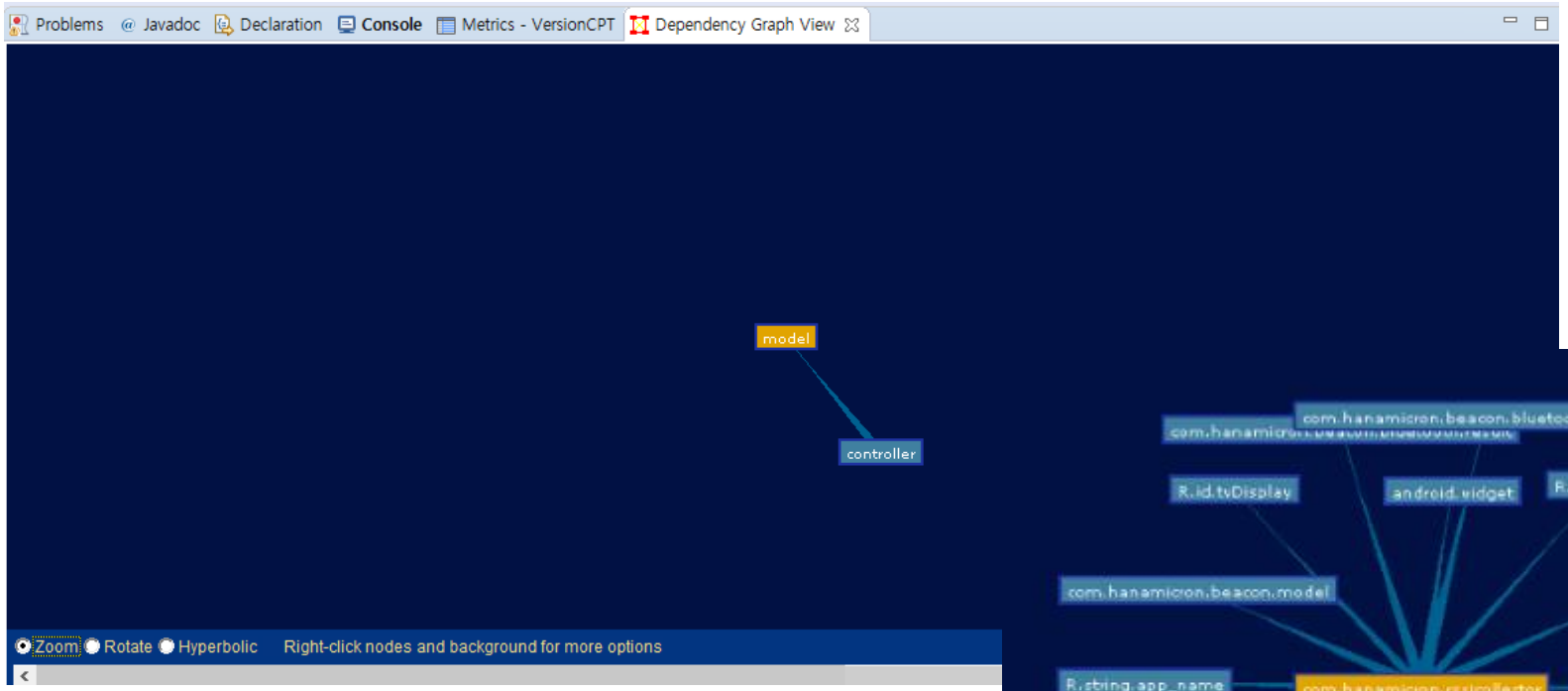


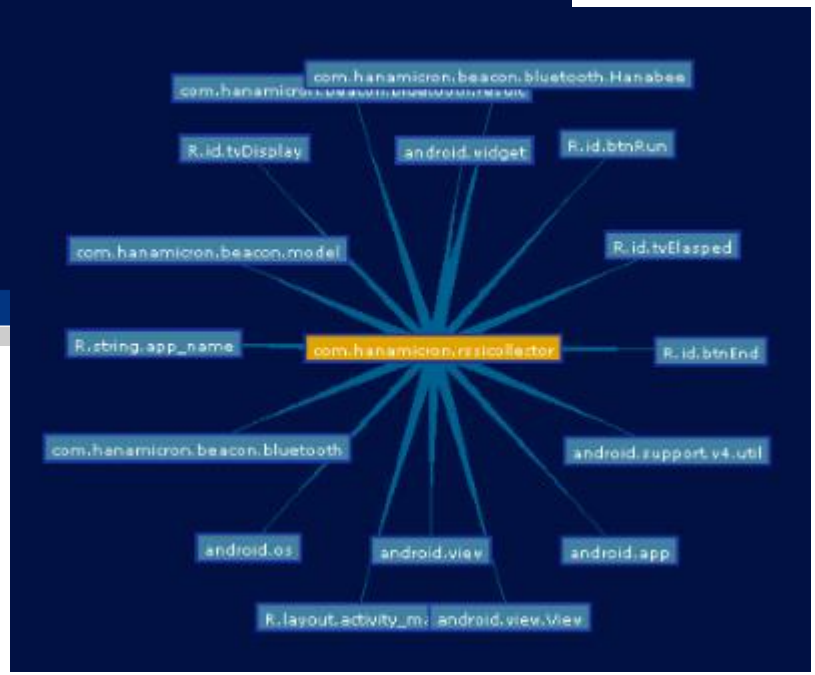
Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per method)		1.7	2.407	23	/VersionCPT/src/controller/ServerThread.java	analyzeMSG
Number of Parameters (avg/max per method)		0.655	0.878	5	/VersionCPT/src/controller/AdminVer.java	chkAuth
Nested Block Depth (avg/max per method)		1.355	0.746	5	/VersionCPT/src/controller/AdminVer.java	chkAuth
Afferent Coupling (avg/max per packageFragment)		1.5	1.5	3	/VersionCPT/src/model	
Efferent Coupling (avg/max per packageFragment)		1.5	1.5	3	/VersionCPT/src/controller	
Instability (avg/max per packageFragment)		0.5	0.5	1	/VersionCPT/src/controller	
Abstractness (avg/max per packageFragment)		0	0	0	/VersionCPT/src/controller	
Normalized Distance (avg/max per packageFragment)		0.5	0.5	1	/VersionCPT/src/model	
Depth of Inheritance Tree (avg/max per type)		1.091	0.287	2	/VersionCPT/src/controller/ServerThread.java	
Weighted methods per Class (avg/max per type)	187	17	7.828	28	/VersionCPT/src/controller/ServerThread.java	
Number of Children (avg/max per type)	0	0	0	0	/VersionCPT/src/controller/AdminProject.java	
Number of Overridden Methods (avg/max per type)	1	0.091	0.287	1	/VersionCPT/src/controller/ServerThread.java	
Lack of Cohesion of Methods (avg/max per type)		0.596	0.261	0.894	/VersionCPT/src/model/Version.java	
Number of Attributes (avg/max per type)	54	4.909	2.678	11	/VersionCPT/src/model/Choice.java	
Number of Static Attributes (avg/max per type)	0	0	0	0	/VersionCPT/src/controller/AdminProject.java	
Number of Methods (avg/max per type)	108	9.818	6.422	24	/VersionCPT/src/model/Version.java	
Number of Static Methods (avg/max per type)	2	0.182	0.575	2	/VersionCPT/src/controller/MainServer.java	
Specialization Index (avg/max per type)		0.061	0.192	0.667	/VersionCPT/src/controller/ServerThread.java	
Number of Classes (avg/max per packageFragment)	11	5.5	0.5	6	/VersionCPT/src/controller	
Number of Interfaces (avg/max per packageFragment)	0	0	0	0	/VersionCPT/src/controller	
Number of Packages	2					
Total Lines of Code	917					
Method Lines of Code (avg/max per method)	549	4.991	12.482	119	/VersionCPT/src/controller/ServerThread.java	analyzeMSG

Eclipse Metrics Plugin

- Dependency Graph







PMD

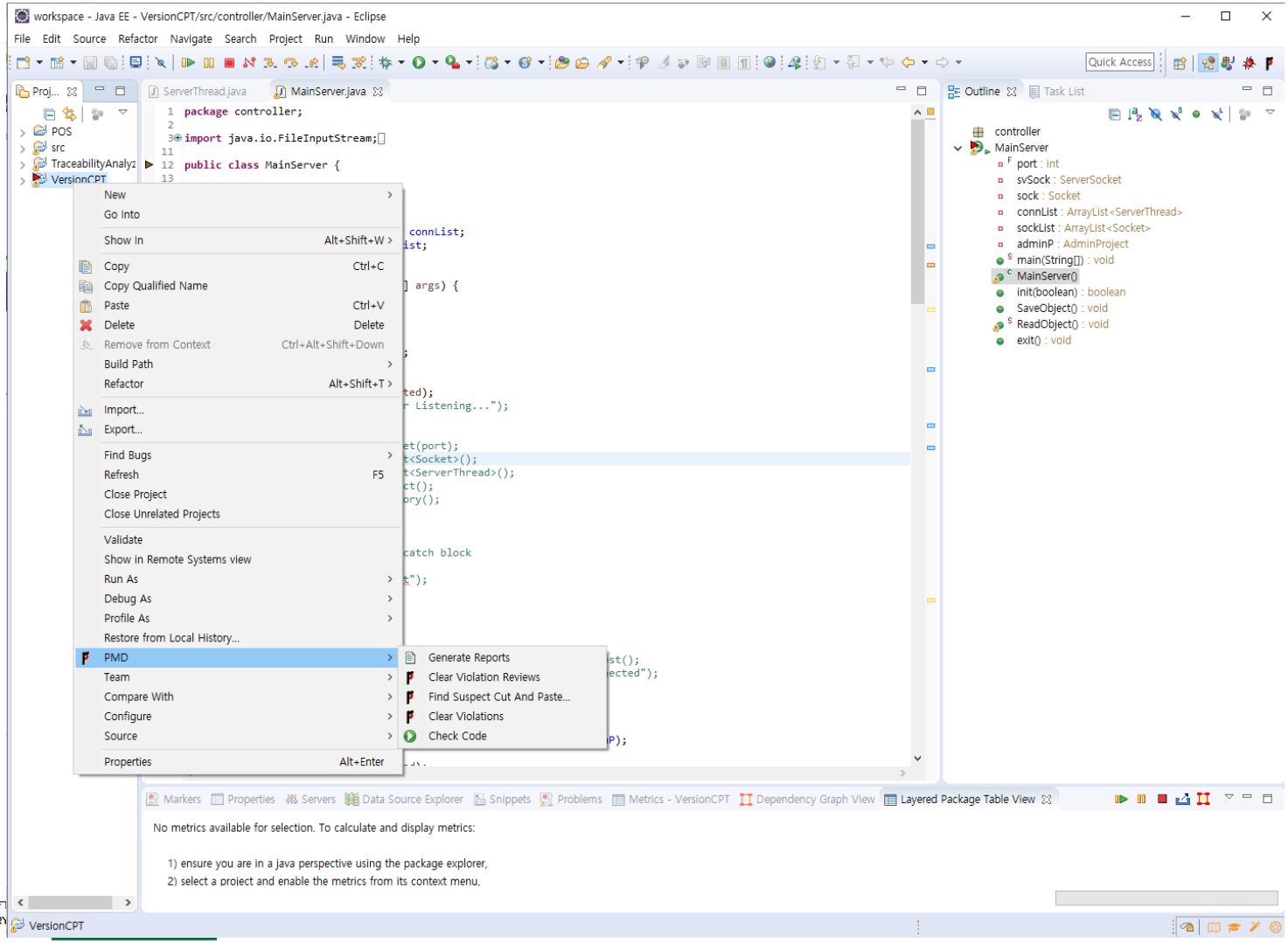
- Level 1+Level 3 (part of)

- PMD is a source code analyzer. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth. It supports Java, JavaScript, Salesforce.com Apex and Visualforce, PLSQL, Apache Velocity, XML, XSL.

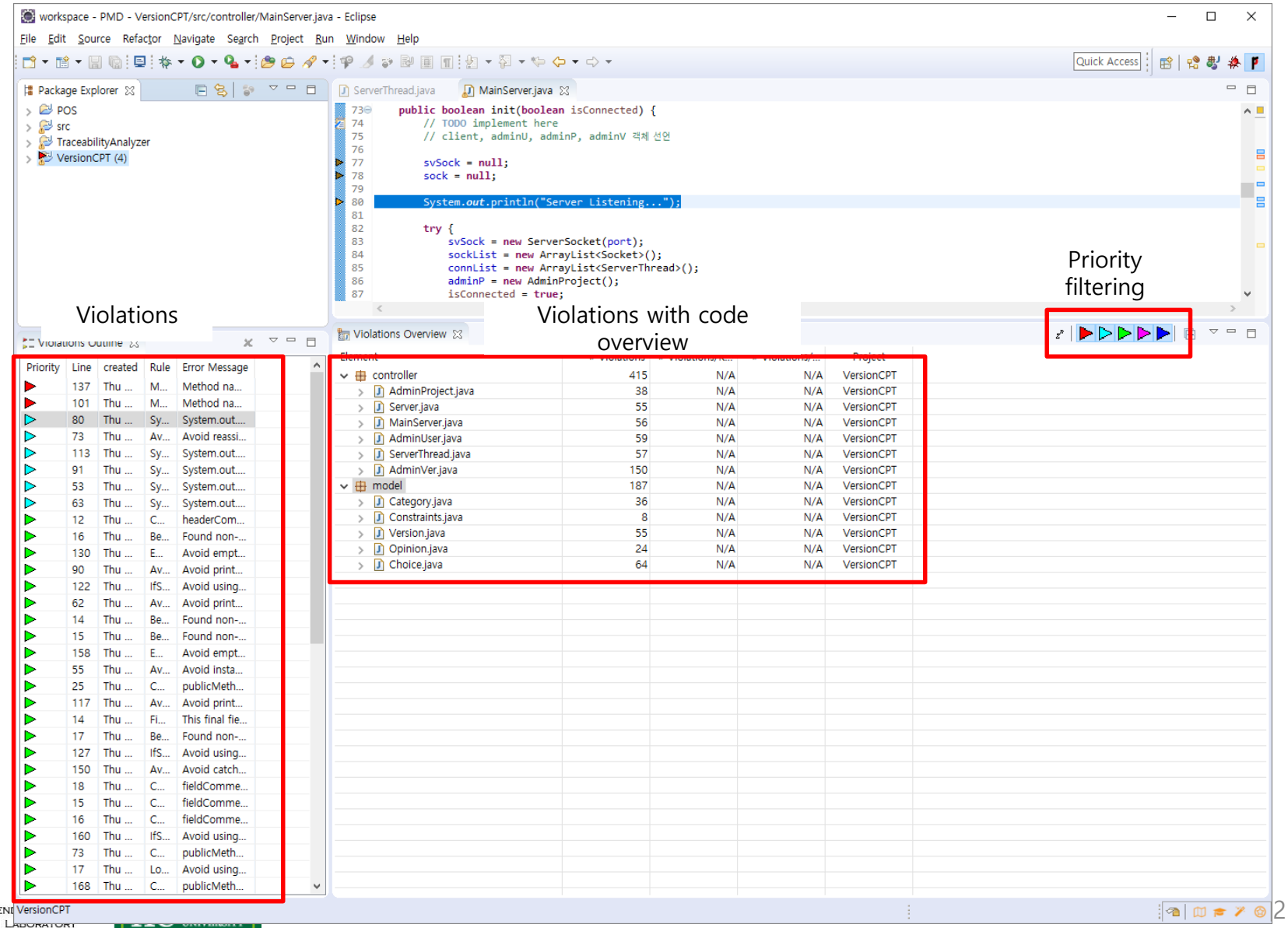
- Additionally it includes CPD, the copy-paste-detector. CPD finds duplicated code in Java, C, C++, C#, Groovy, PHP, Ruby, Fortran, JavaScript, PLSQL, Apache Velocity, Scala, Objective C, Matlab, Python, Go, Swift and Salesforce.com Apex and Visualforce.

PMD

- Install
  - Install new software -> https://dl.bintray.com/pmd/pmd-eclipse-plugin/updates/

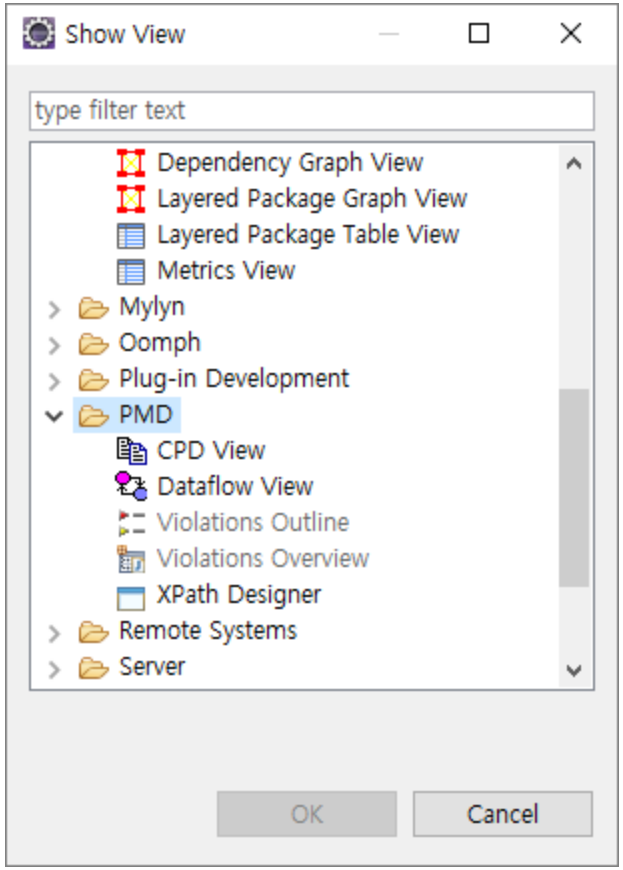


PMD


workspace - PMD - VersionCPT/src/controller/MainServer.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Package Explorer
POS
src
TraceabilityAnalyzer
VersionCPT (4)
ServerThread.java
MainServer.java
public boolean init(boolean isConnected) {
// TODO implement here
// client, adminU, adminP, adminV 객체 선언
svSock = null;
sock = null;
System.out.println("Server Listening...");
try {
svSock = new ServerSocket(port);
sockList = new ArrayList<Socket>();
connList = new ArrayList<ServerThread>();
adminP = new AdminProject();
isConnected = true;
Violations
Violations with code overview
Priority filtering
Violations Outline
Priority Line created Rule Error Message
Violations Overview
Element Violations Violations/K... Violations/... Project
controller 415 N/A N/A VersionCPT
AdminProject.java 38 N/A N/A VersionCPT
Server.java 55 N/A N/A VersionCPT
MainServer.java 56 N/A N/A VersionCPT
AdminUser.java 59 N/A N/A VersionCPT
ServerThread.java 57 N/A N/A VersionCPT
AdminVer.java 150 N/A N/A VersionCPT
model 187 N/A N/A VersionCPT
Category.java 36 N/A N/A VersionCPT
Constraints.java 8 N/A N/A VersionCPT
Version.java 55 N/A N/A VersionCPT
Opinion.java 24 N/A N/A VersionCPT
Choice.java 64 N/A N/A VersionCPT
VersionCPT

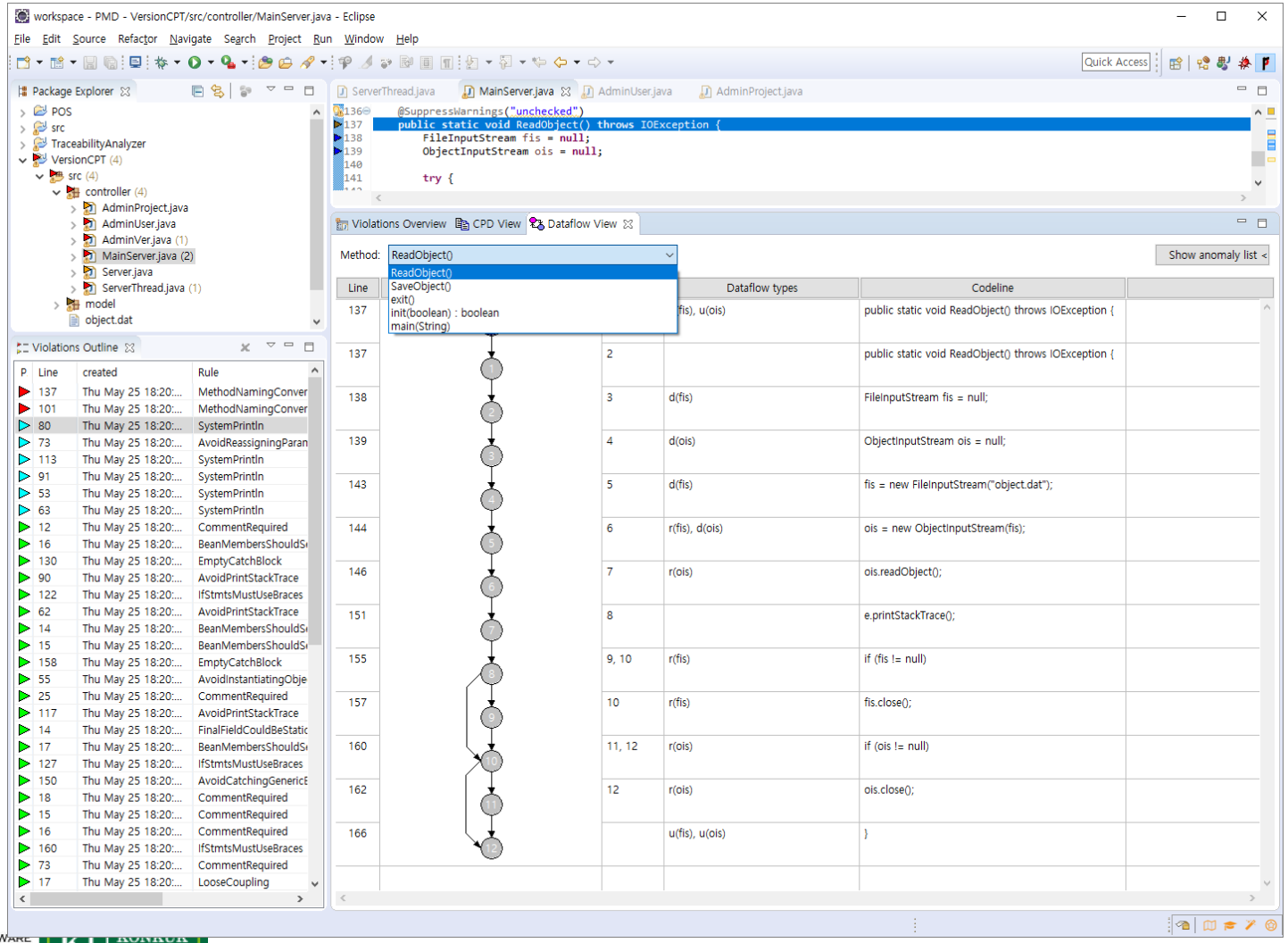
PMD

- 이외에도 dataflow, CPD (Finding duplicated code) 가 가능



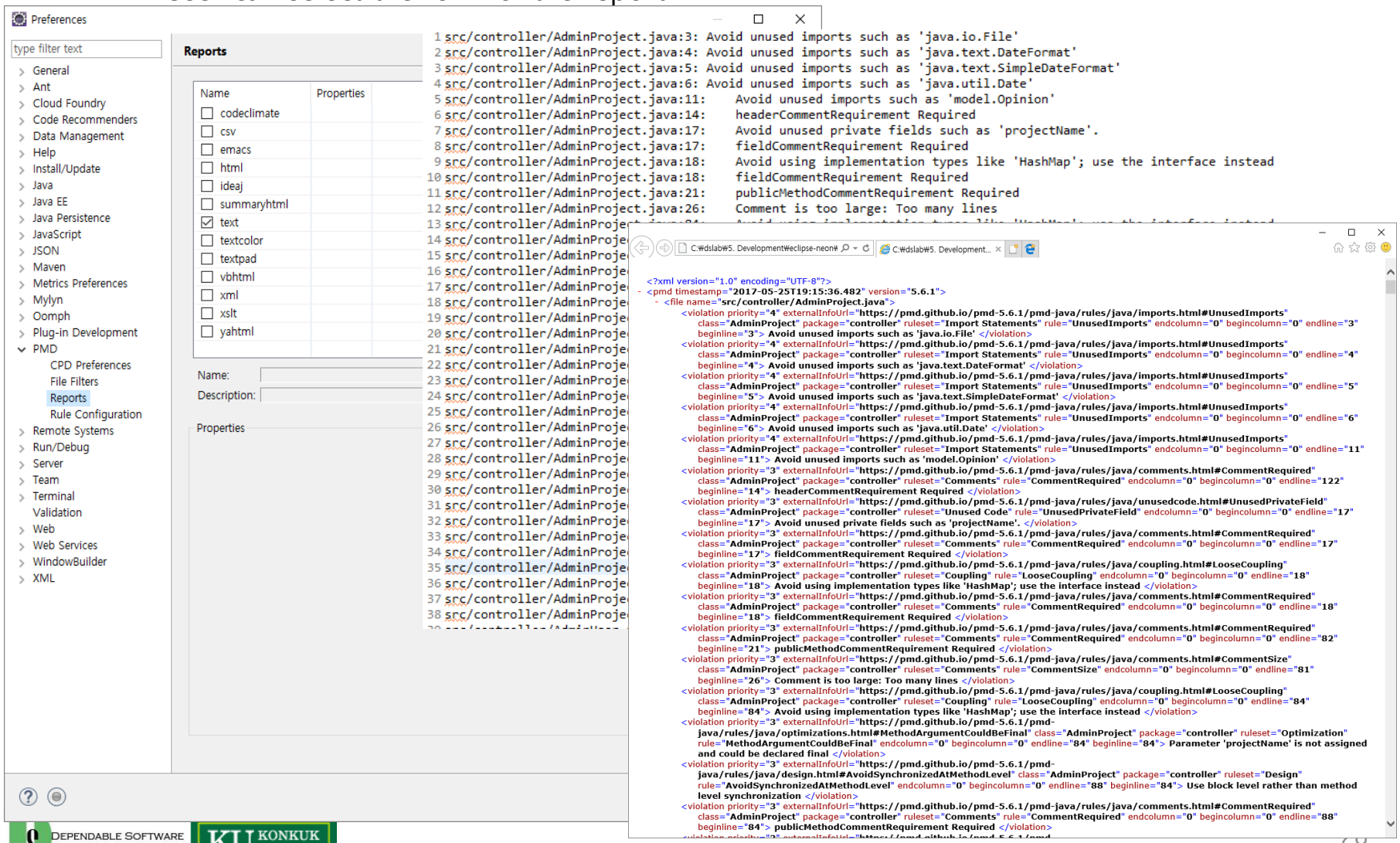
PMD

- Method 별 data flow



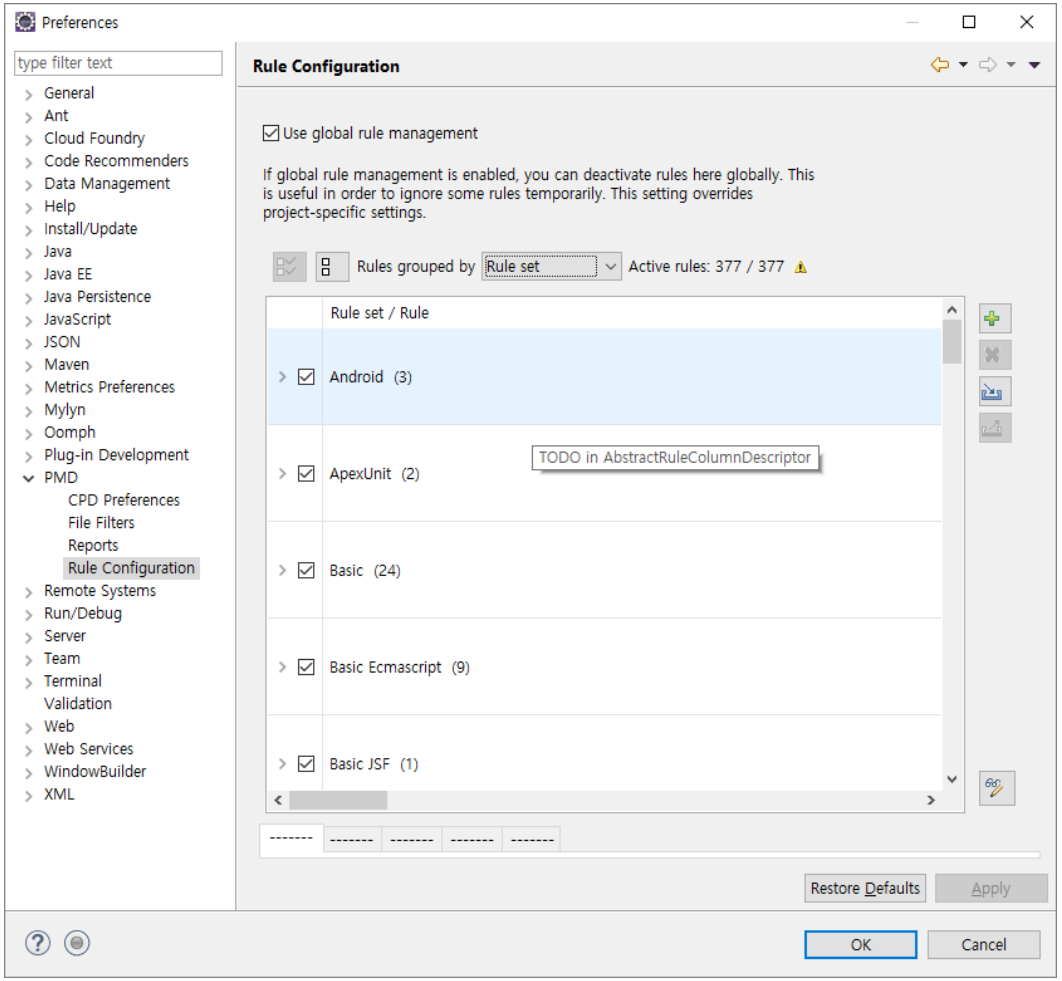
PMD

- Report example
  - User can select the form of the report



PMD

- Window -> Preference -> PMD -> Rule Configuration
  - 사용할 rule set 설정
  - 새로운 rule 추가 가능



Checkstyle

- Level 1

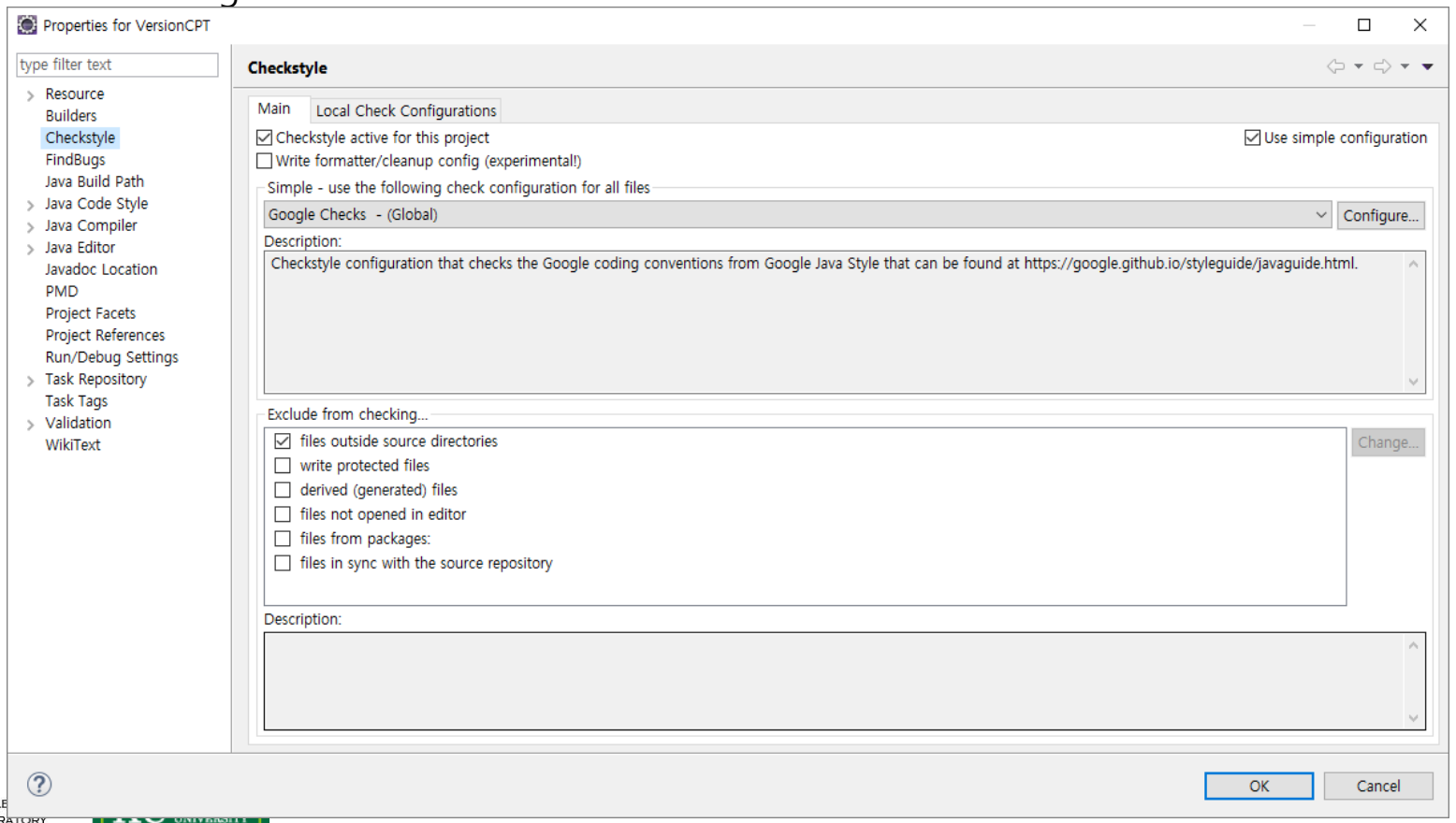
- Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard.

- Checkstyle is highly configurable and can be made to support almost any coding standard. An example configuration files are supplied supporting the Sun Code Conventions, Google Java Style.

Checkstyle

- Install
  - Install New Software -> http://eclipse-cs.sf.net/update/

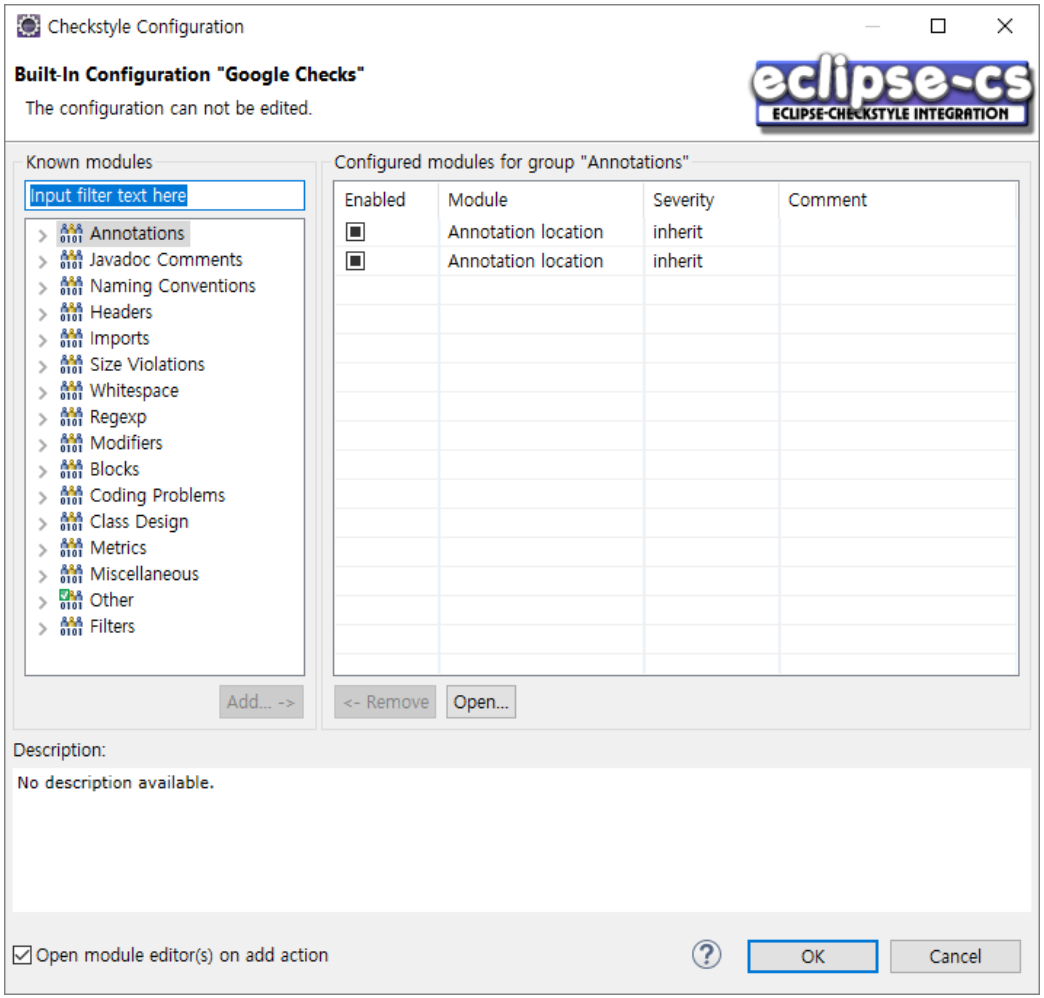
- Properties
  - Activation
  - Rule configuration



Checkstyle

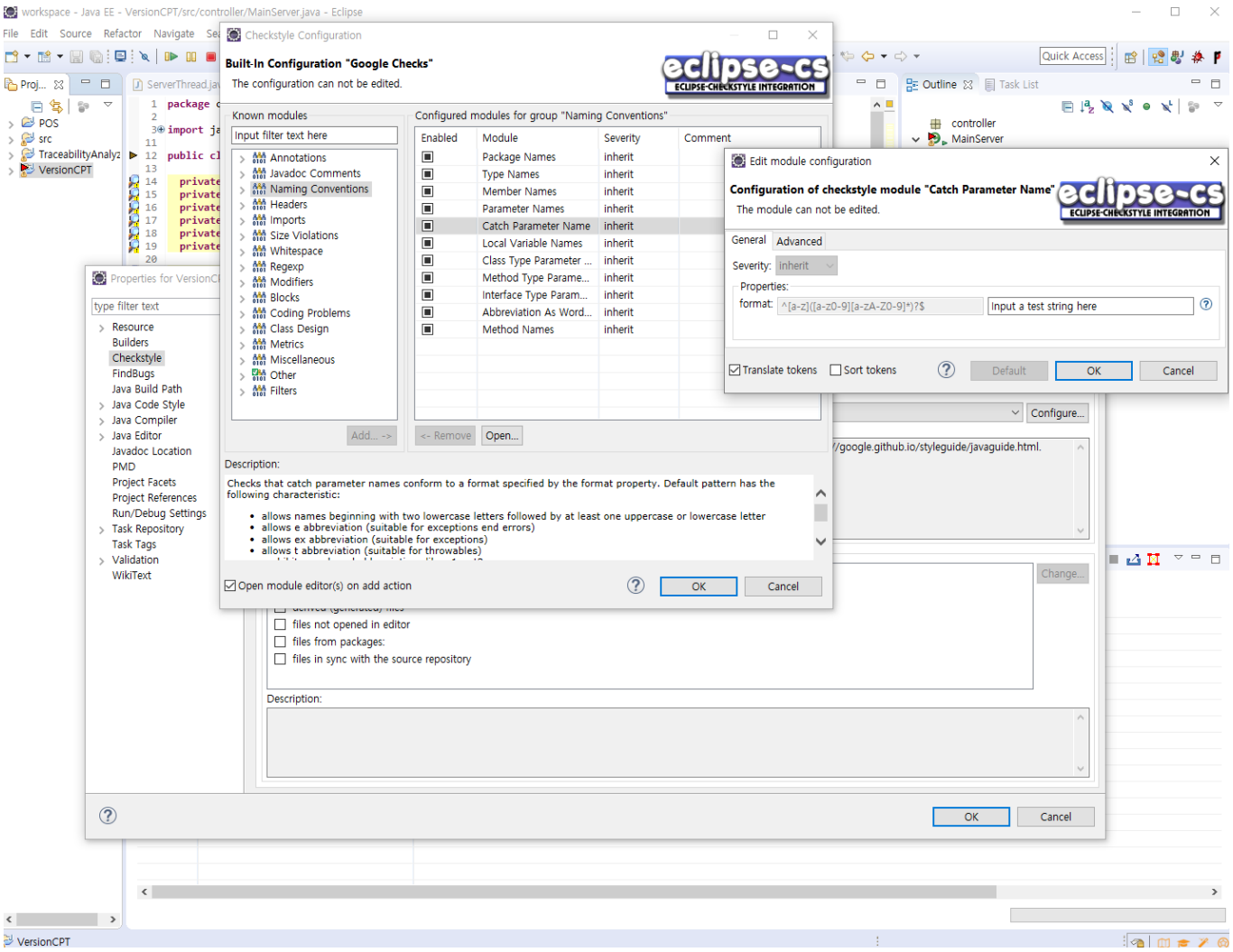
- Rule configuration
  - 기본적으로 google, sun의 coding style 제공
  - 여러 style을 선택 가능

- **User define rule also available**



Checkstyle

- Naming Convention example



Checkstyle

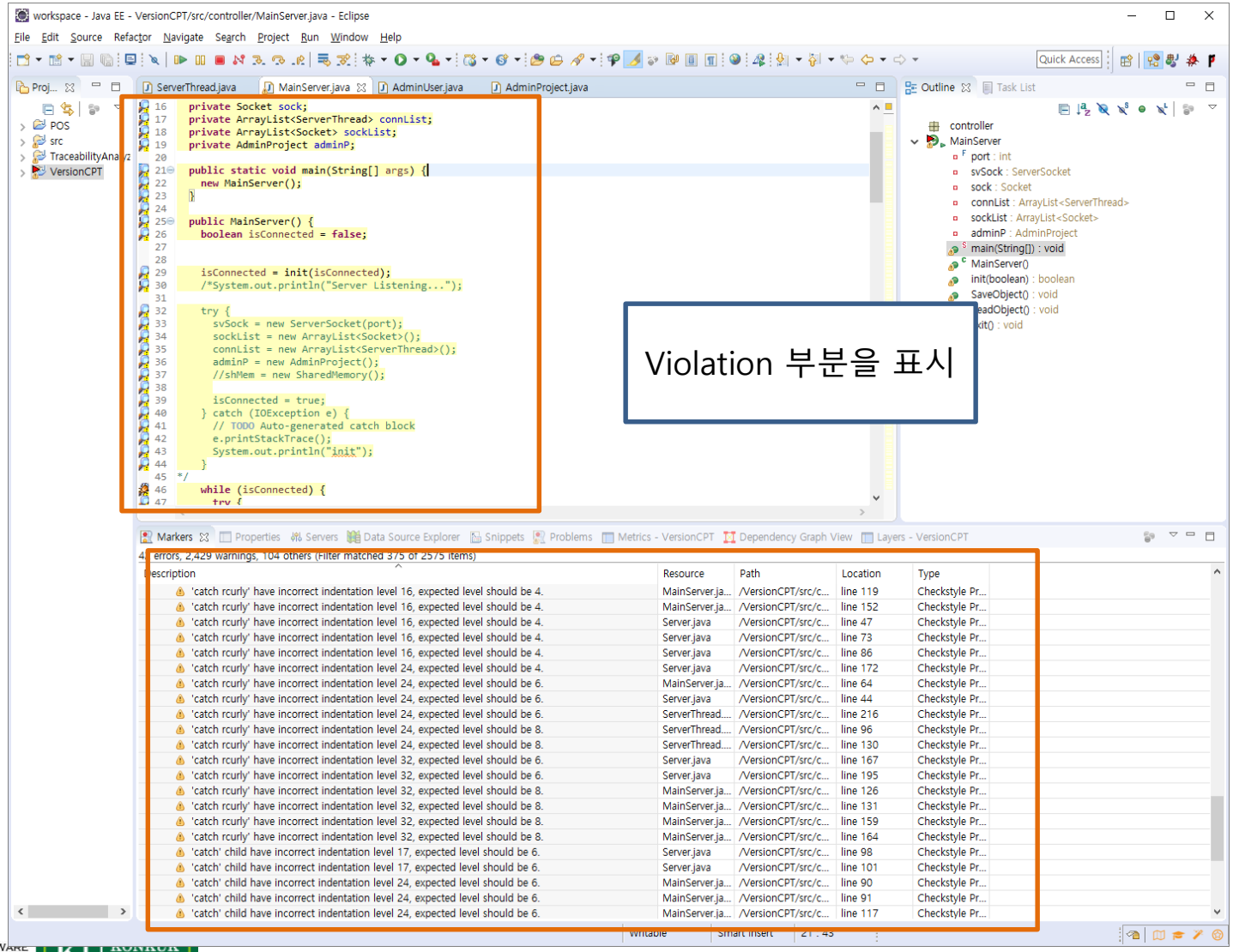
- Other rule example

항목	원인	회피방법
JavadocPackage	모든 method, class에는 help가 존재해야 한다.	시간상 힘들고, 관리되지 않는 주석은 더욱 큰혼란을 가지고 온다. method의 이름 규칙으로 대신하기로 한다.
NewlineAtEndOfFile	java code의 가장 마지막 줄은 빈공백열로 마쳐져야 한다.	마지막 line에는 항시빈공백을 넣는다.
Translation	Properties file을 이용한 경우, 국가별 번역이 모두 존재해야 한다.	국가별 번역 파일을 따로 만들거나 default 문자열만을 이용한다.
FileLength	java file의 length는2000 line을 넘지 않도록 작성한다.	2000 line이 넘어가는경우, 설계상의 문제가있기 때문에 class를 재정의한다.
FileTabCharacter	java file 내부에 tab 문자열이 있으면 안된다.	tab을 모두 space로 치환해서 사용하도록 한다.
RegexpSingleline	1 line에는 한개의method만이 존재해야 한다.	1 line에 대한 설정을 명확하게 해서 사용하도록 한다.

- http://netframework.tistory.com/363

Checkstyle

- Activate example



FindBugs

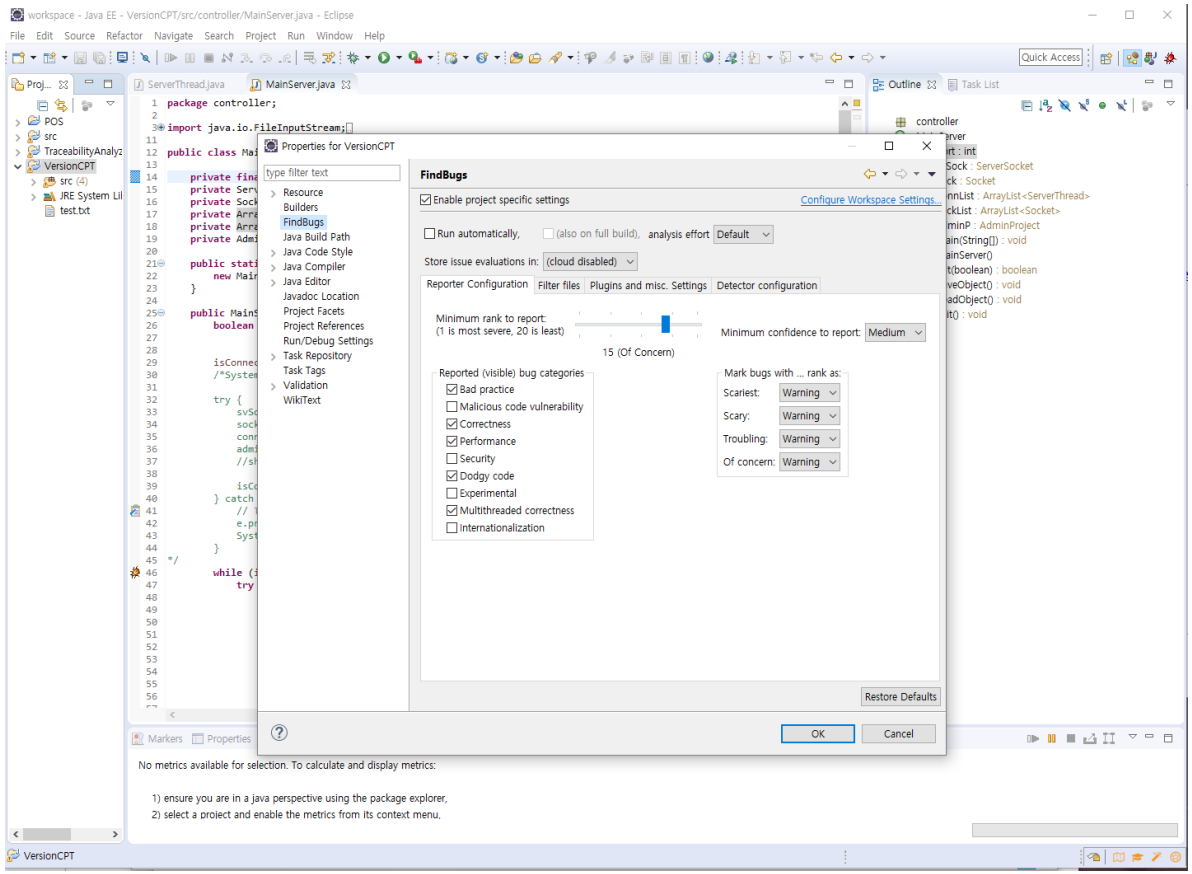
- Level 3

- It is an open source program which looks for bugs in Java code
  - Operates on Java bytecode, rather than source code
  - Source code also available

FindBugs

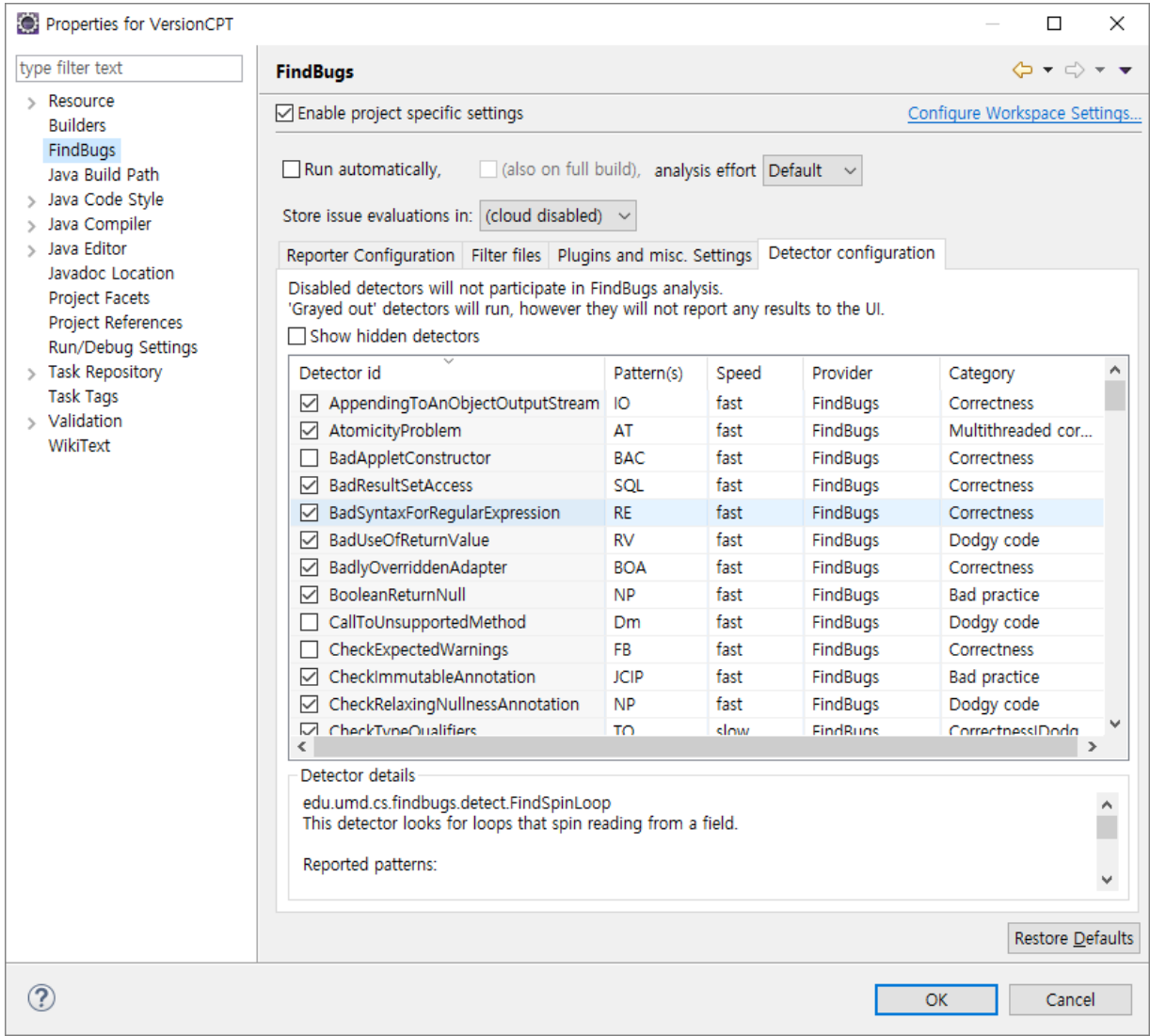
- Install
  - Install new software -> http://findbugs.cs.umd.edu/eclipse

- project -> properties -> Findbugs -> enable



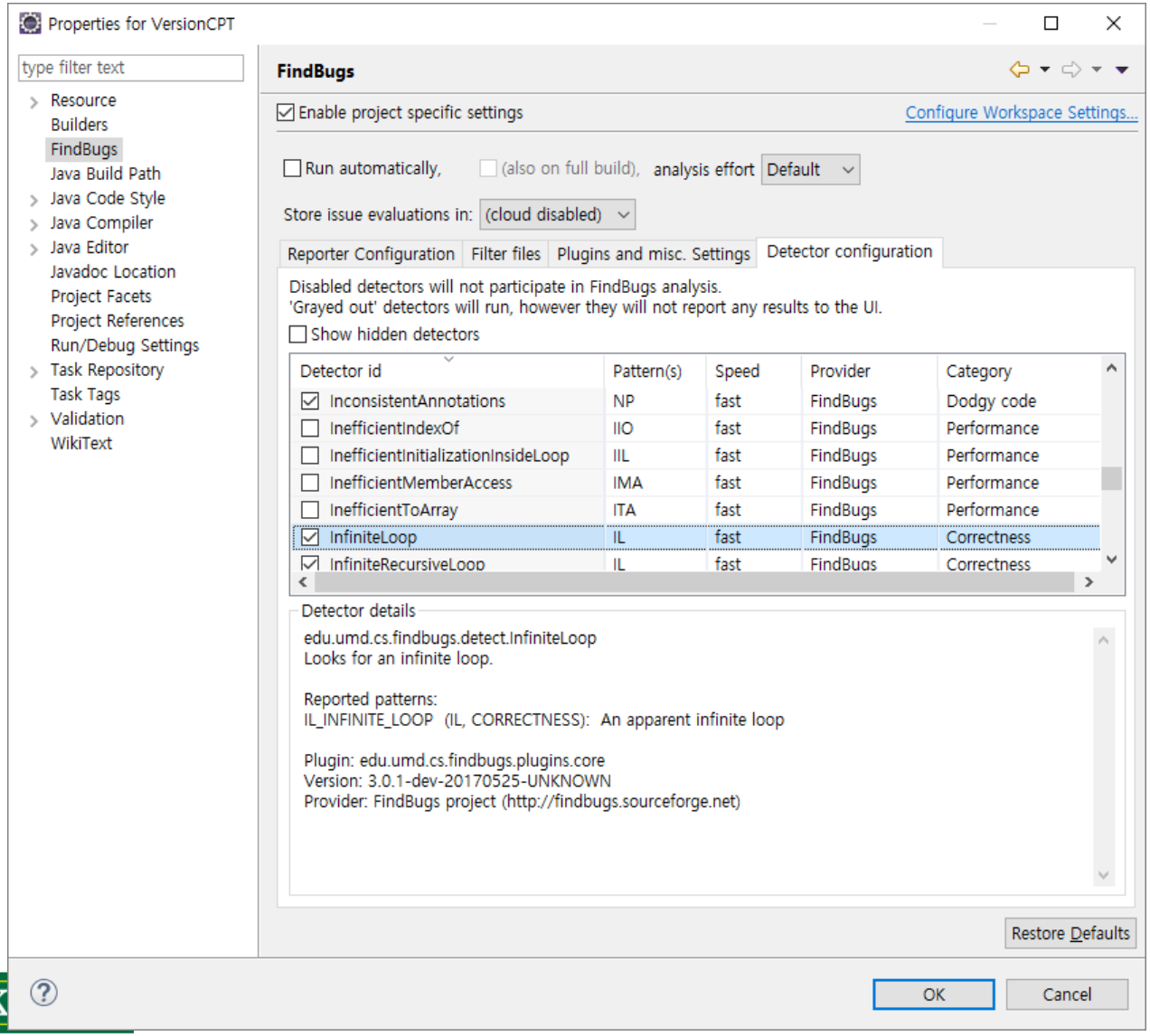
FindBugs

- Detector configuration
  - Searching **rule setting**
    - Enable/disable



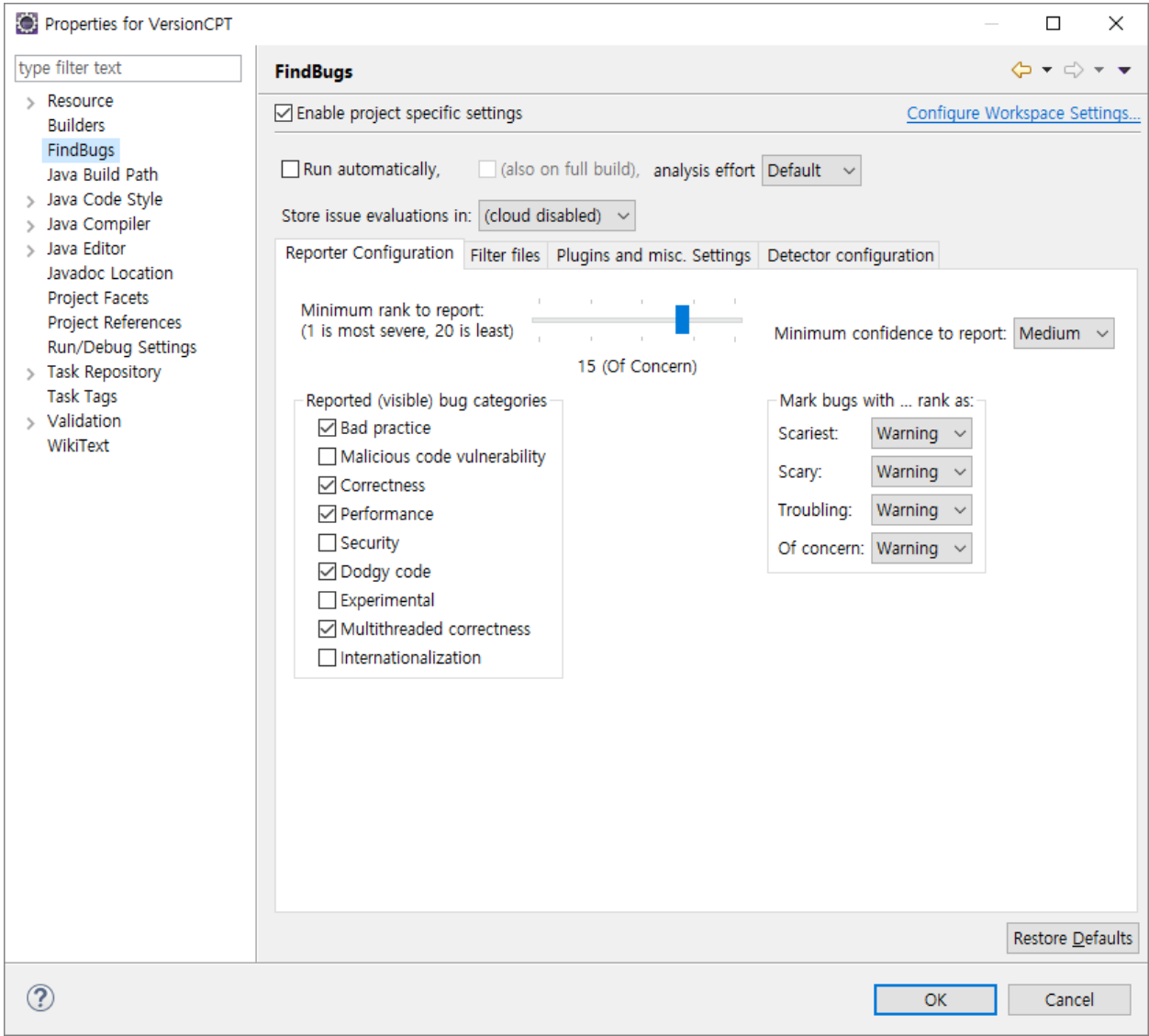
FindBugs

- Example - InfiniteLoop



FindBugs

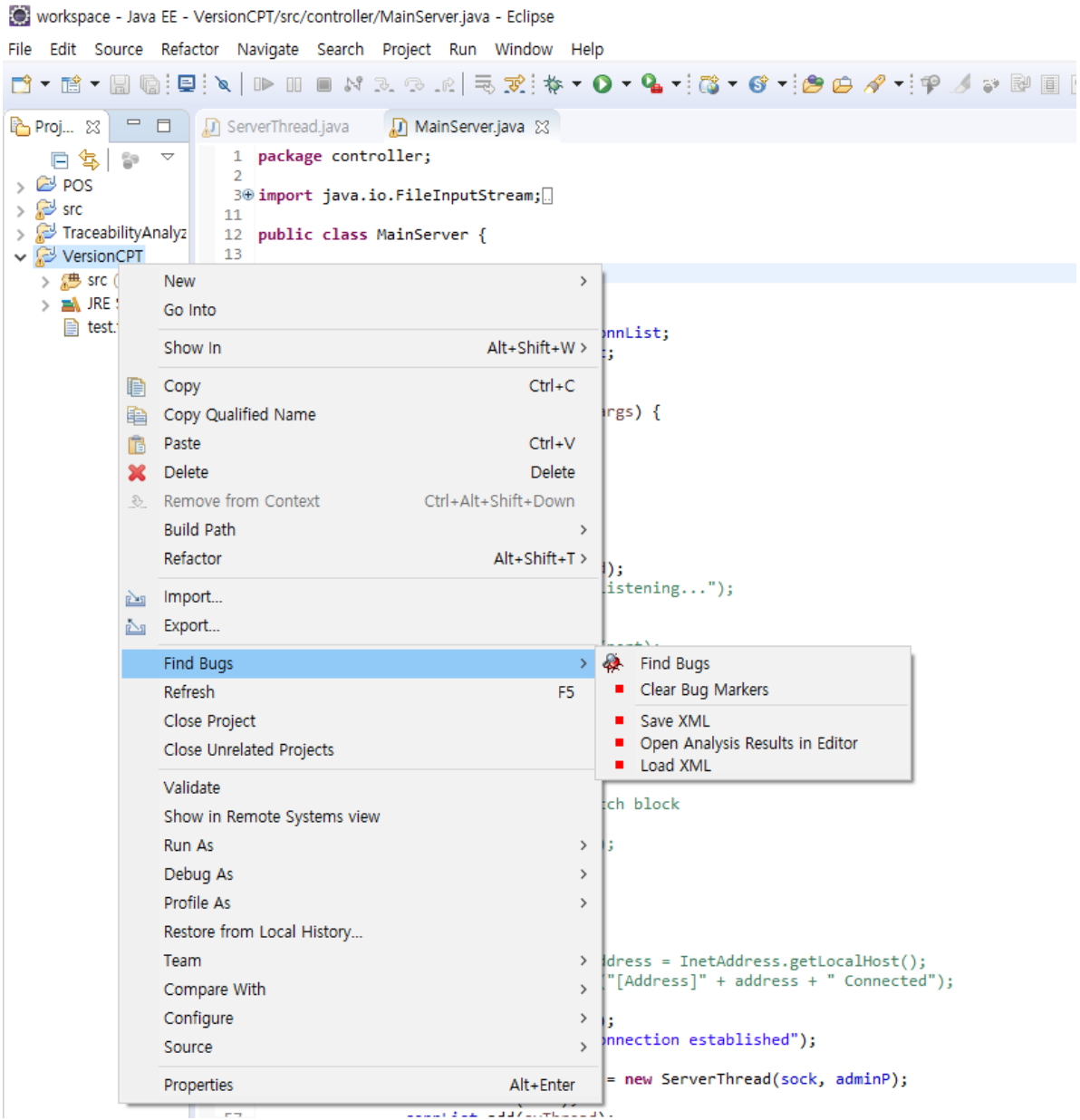
- Report setting
  - Report로 생성할 항목들 설정 등



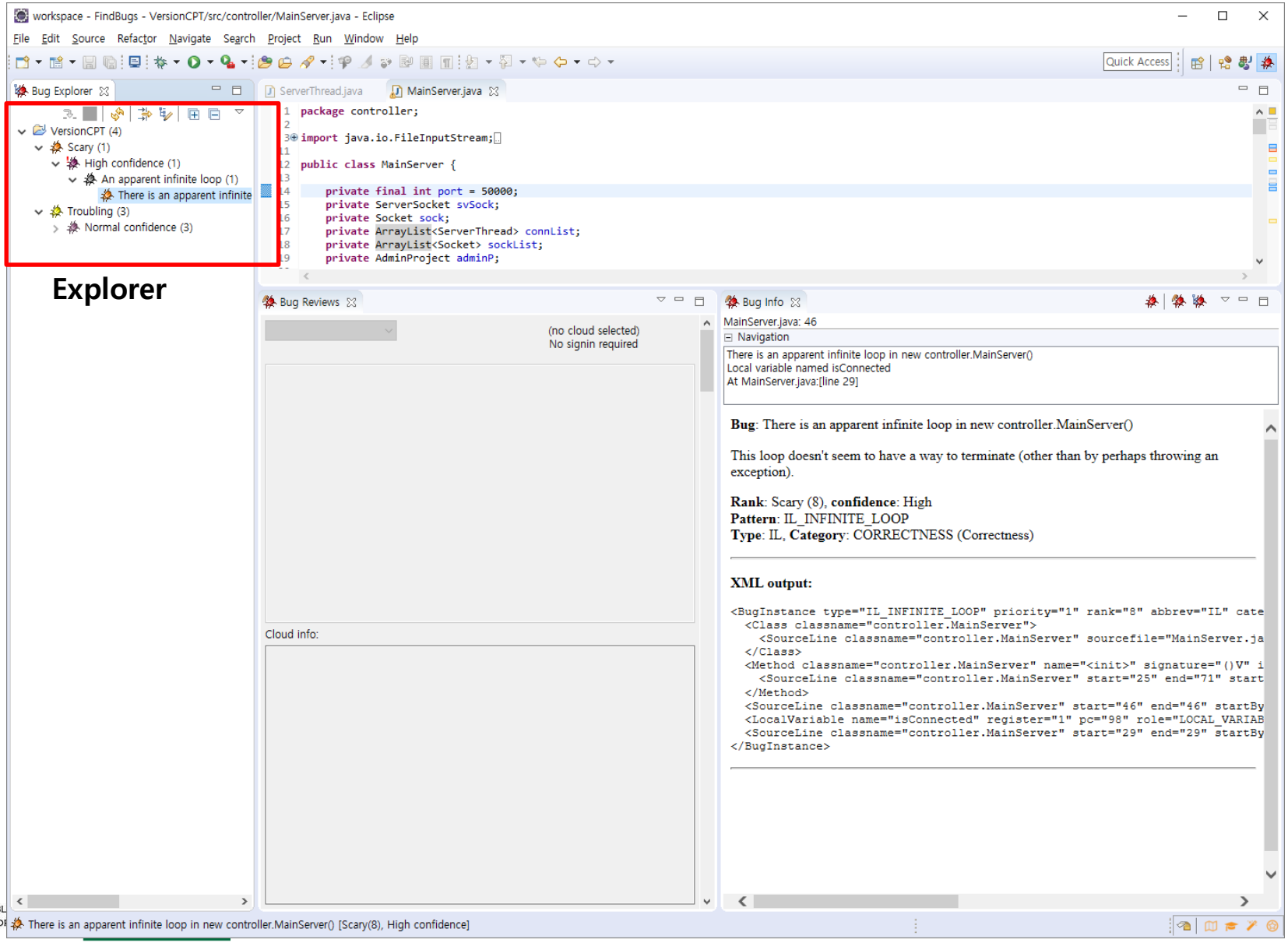
FindBugs

- Execution
  - Find Bugs click

- XML generation is possible

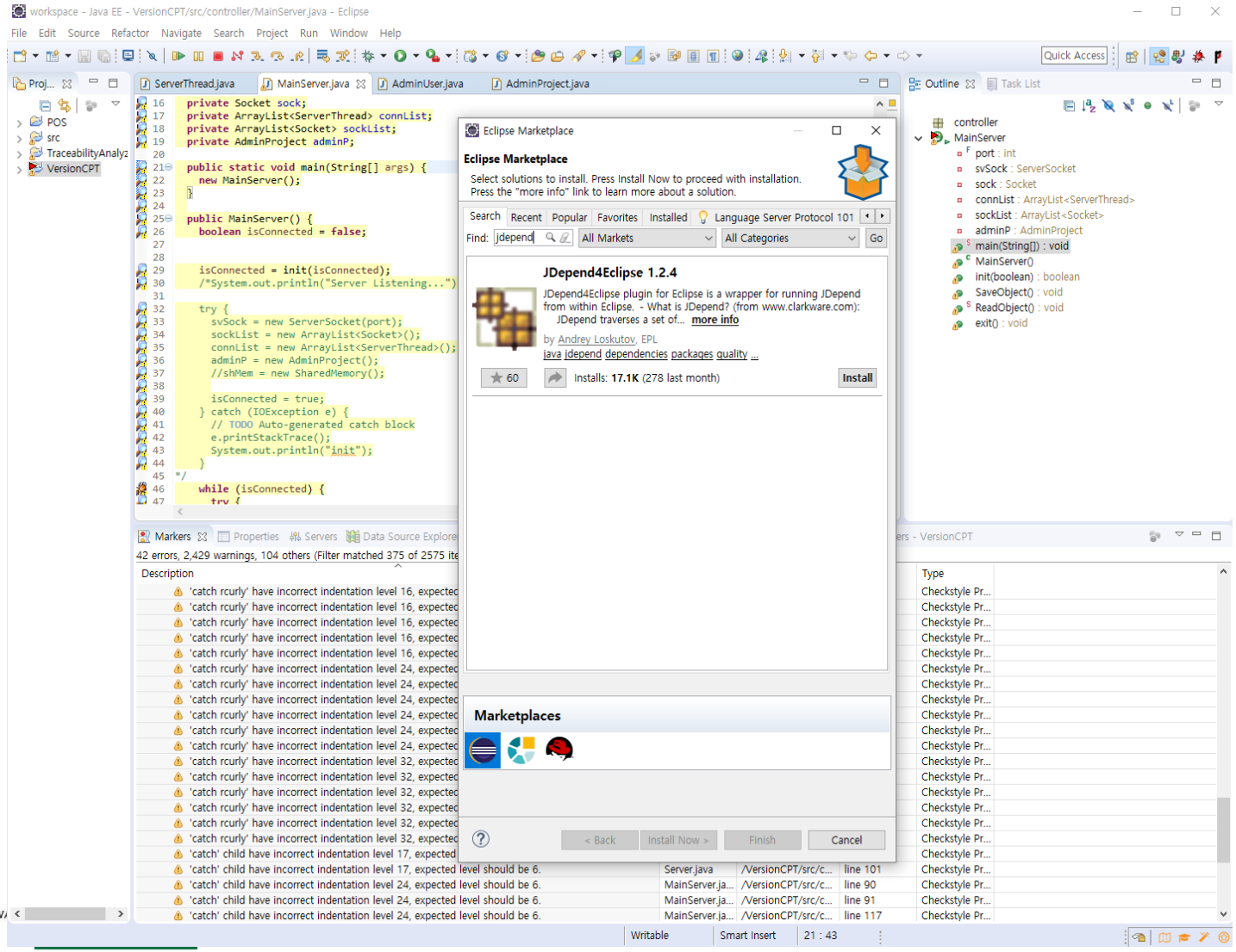


FindBugs


workspace - FindBugs - VersionCPT/src/controller/MainServer.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Bug Explorer
VersionCPT (4)
Scary (1)
High confidence (1)
An apparent infinite loop (1)
There is an apparent infinite
Troubling (3)
Normal confidence (3)
ServerThread.java
MainServer.java
package controller;
import java.io.FileInputStream;
public class MainServer {
private final int port = 50000;
private ServerSocket svSock;
private Socket sock;
private ArrayList<ServerThread> connList;
private ArrayList<Socket> sockList;
private AdminProject adminP;
Explorer
Bug Reviews
(no cloud selected)
No signin required
Cloud info:
Bug Info
MainServer.java: 46
Navigation
There is an apparent infinite loop in new controller.MainServer()
Local variable named isConnected
At MainServer.java:[line 29]
Bug: There is an apparent infinite loop in new controller.MainServer()
This loop doesn't seem to have a way to terminate (other than by perhaps throwing an exception).
Rank: Scary (8), confidence: High
Pattern: IL_INFINITE_LOOP
Type: IL, Category: CORRECTNESS (Correctness)
XML output:
<BugInstance type="IL_INFINITE_LOOP" priority="1" rank="8" abbrev="IL" cate
<Class classname="controller.MainServer">
<SourceLine classname="controller.MainServer" sourcefile="MainServer.ja
</Class>
<Method classname="controller.MainServer" name="<init>" signature="()V" i
<SourceLine classname="controller.MainServer" start="25" end="71" start
</Method>
<SourceLine classname="controller.MainServer" start="46" end="46" startBy
<LocalVariable name="isConnected" register="1" pc="98" role="LOCAL_VARIAB
<SourceLine classname="controller.MainServer" start="29" end="29" startBy
</BugInstance>
There is an apparent infinite loop in new controller.MainServer() [Scary(8), High confidence]

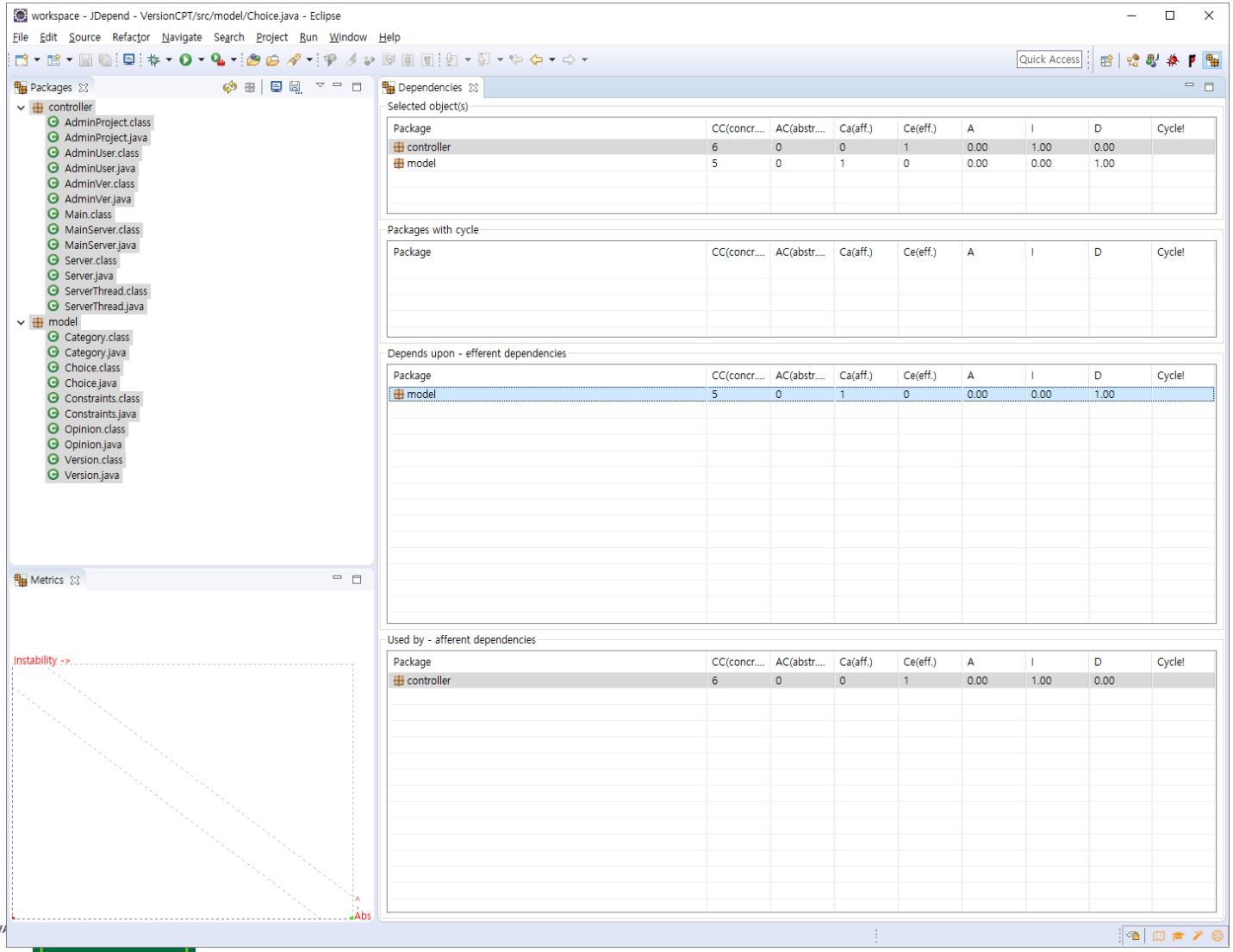
JDepend

- Level 2
  - Help -> Eclipse Marketplace -> jdepend 입력후 검색



JDepend

- Source code가 있는 폴더 선택 후 JDepend run



JDepend

- CC :: Concrete Class 인터페이스나 추상 클래스가 아닌 Concrete Class 의 수를 나타냄
- AC :: Abstract Class 추상 클래스나 인터페이스의 수를 나타내며 확장성의 척도가 됨
- Ca :: Afferent Couplings 현재 패키지의 클래스에 의존하고 있는 패키지의 수를 나타내며 책임의 척도가 됨
- Ce :: Efferent Couplings 현재 패키지의 클래스들이 의존하고 있는 패키지의 수를 나타내며 독립성의 척도가 됨
- A :: Abstractness (A = AC/CC+AC) 추상화 정도를 나타내며, 0 은 구체적인 패키지를, 1 은 추상적인 패키지를 나타냄
- I :: Instability (I = Ce(Ce+Ca)) 변화에 대한 안정성을 나타내며 0 부터 1 사이의 값을 가짐, 0 은 외부 변화에도 끄떡 없는 패키지이며 1 은 작은 변화에도 쉽게 흔들릴 수 있는 패키지를 나타냄
- D :: Distance to Main Sequence Main Sequence 로부터의 거리를 나타내며, 0 은 Main Sequence 와 완전 가깝고 1 은 완전 먼 상태임, Main Sequence란 이상적인 패키지로 완전 추상적이면서 안정적이거나 완전 구체적이면서 불안정한 패키지를 나타냄
- Cycle :: Package dependency cycles 패키지들 상호 간에 의존성을 가지고 있을 때 발생함, 안 좋은 상황이기 때문에 경고 아이콘으로 보여짐

SonarQube

- 이것도 한번 사용해 보세요

발표 – Static Analysis

- 각자 서로 team의 source code를 대상으로 static analysis를 수행, 결과 발표
  - 3개의도구 선택 (복잡도 or 의존성 분석 도구 1개 반드시 포함)
  - 설명한 도구를 포함해 많은 도구들 중 자유롭게 선택
  - 분석 결과 중 critical 한 부분들에 대한 분석발표

1	4, 3
2	4, 1
3	1, 2
4	2, 3
5	8, 7
6	8, 5
7	5, 6
8	6, 7