Try White Box Testing in Our Project

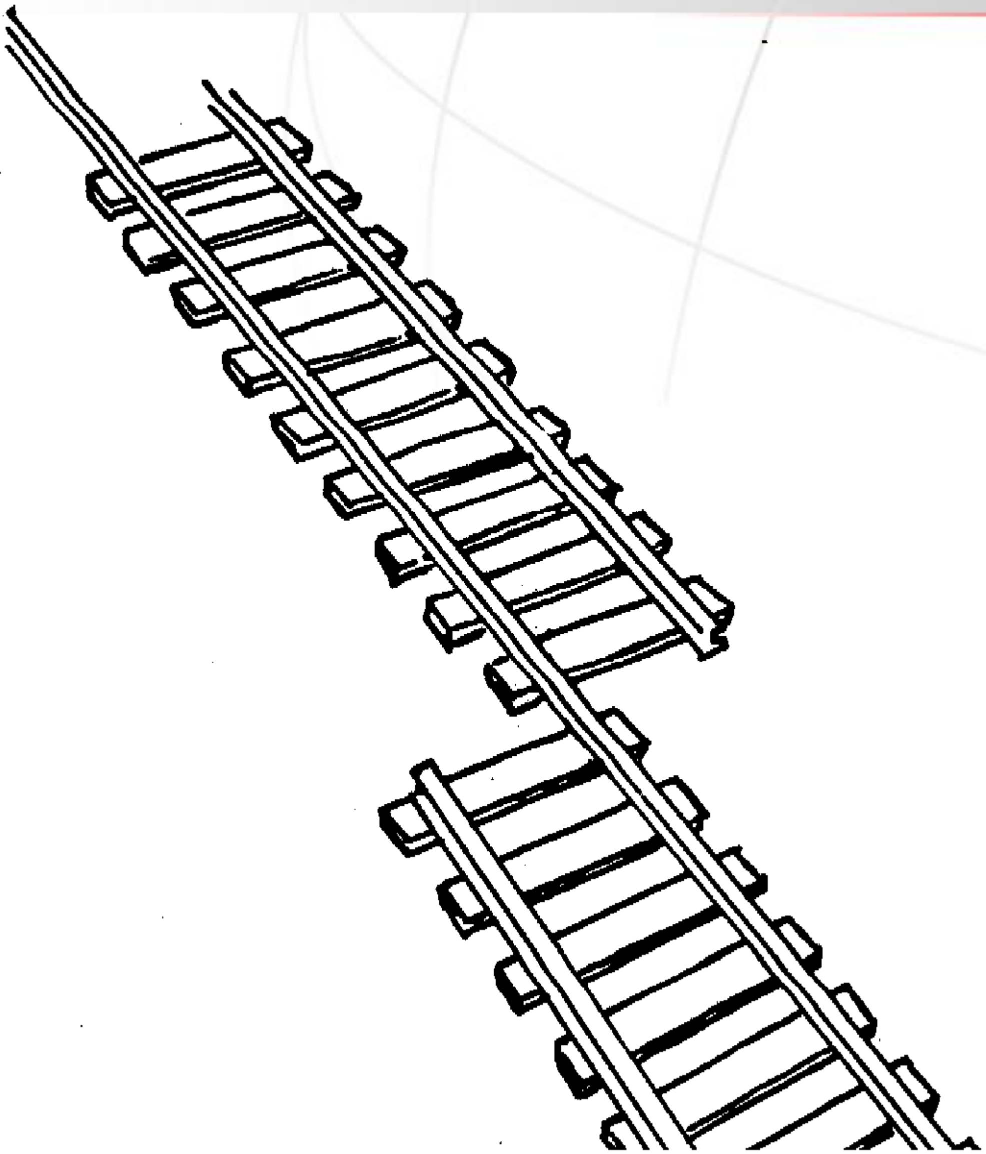
Forrest
GMS Team

Nov. 2009

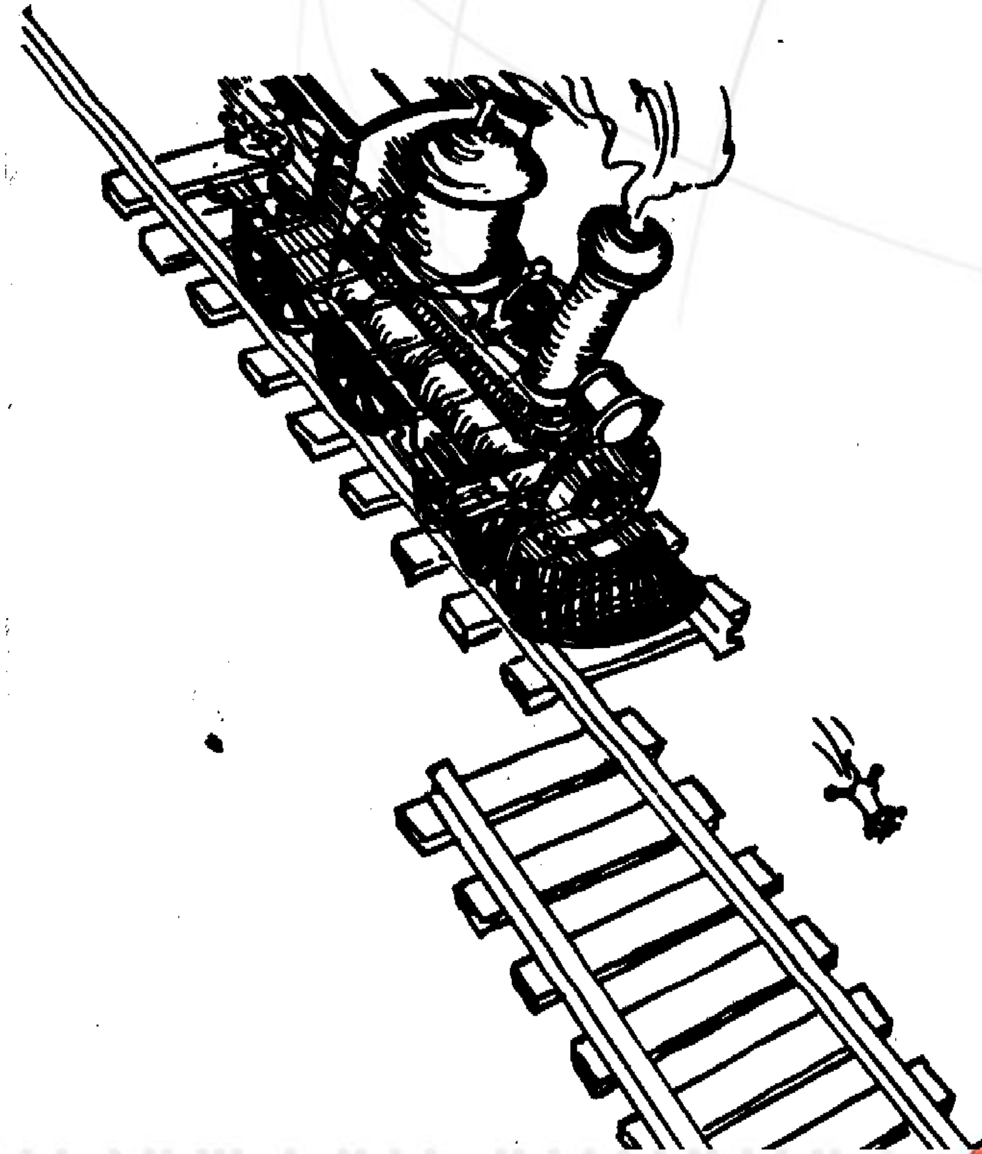
Agenda

- White Box
- Static Code Review
- FindBugs
- FindBugs upon HBase0.20.0
- WhiteBox practice in our project
- Coverage Test
- Discussion

What is this?

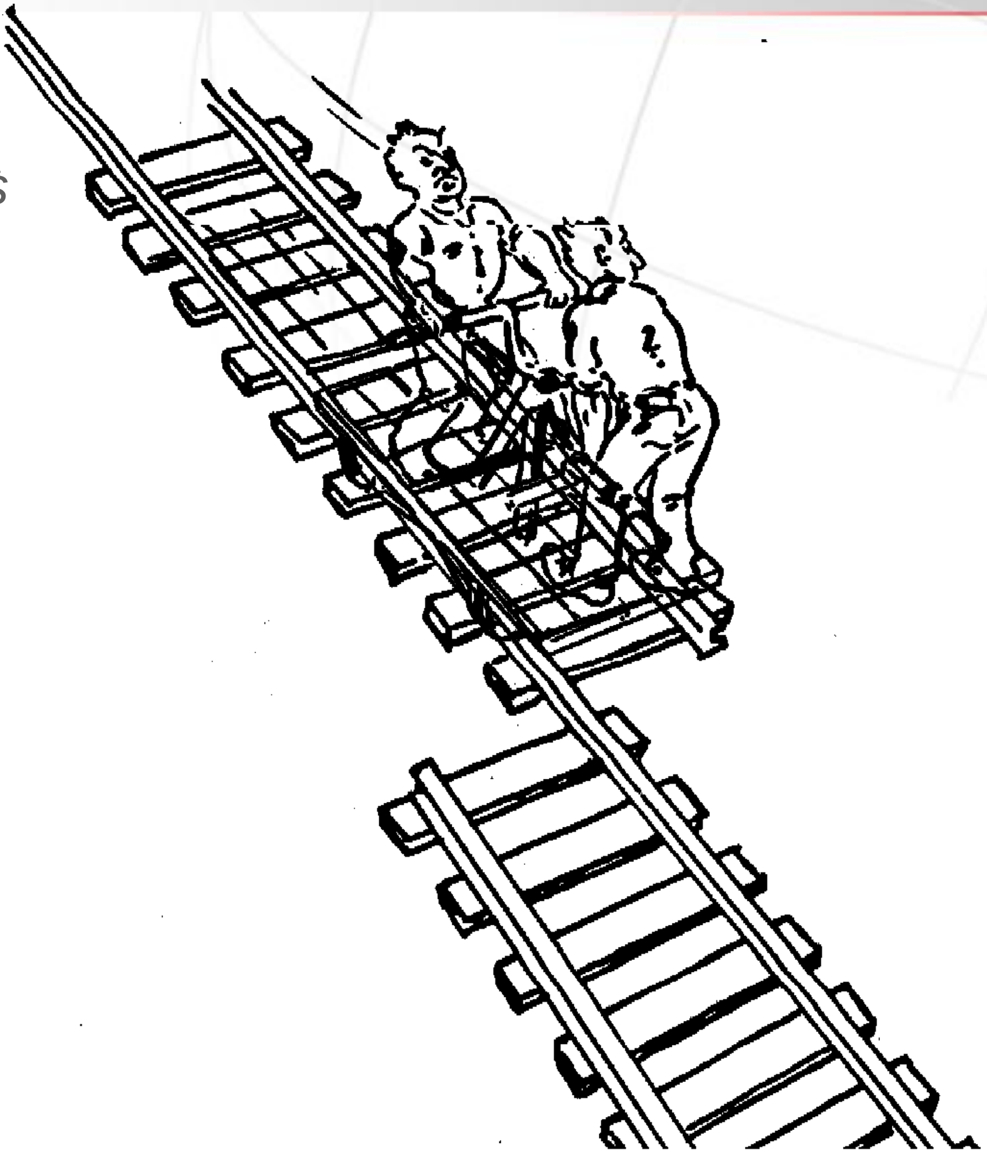


How can we prevent this?



Testing!

"*Given enough eyeballs, all bugs are shallow.*" -- Linus's Law according to Eric S. Raymond



Approach of boxes

- **Black box testing**
- Black box testing treats the software as a "black box"—without any knowledge of internal implementation.

- **White box testing**
- White box testing is when the tester has access to the internal data structures and algorithms including the code that implement these.

- Detailed Introduction of White Box Testing will not be discussed here. ☹
- See ETP AT1001 Testing methodology and ETP AT1013 White box and grey box testing

White Box Testing in development

- White box testing can be applied at all levels of system development

  —unit, integration, and system.

- Manual: Inspection
- Automatic: Code Analysis

when you have code like:

```
// Eclipse 3.0.1
if (in == null)
  try {
     in.close();
  } catch (IOException e) {}
```

Easy to confuse == and !=

```
// JBoss 4.0.0RC1
if (header != null ||
header.length > 0) {
    ...
}
```

Easy to confuse && with ||

This type of error (and less obvious bugs) occur in production mode more frequently than you might expect

Securing Your Web World

Code Inspection

- Lots of stupid errors exist in production code and can be found using simple techniques
  - but successfully using this in the software development process can be a challenge

- Code inspection is an error-detection technique that employs code reading and uses error checklists

Myers G. *The Art of Software Testing*. John Wiley & Sons, 1979.

TREND MICRO

Automatic static code reviews

- Inspect the code automatically for possible defects
- performed without actually executing programs
- performed by an automated tool, with human analysis being called program comprehension
- Defect types: single threaded correctness, multithreaded correctness, performance issue, style etc.

Static Code Analysis Tools : Open Source Project

- FindBugs - works on bytecode, uses BCEL. Source code uses templates, nifty stuff!
- PMD scans Java source code and looks for potential problems
- Checkstyle - Very detailed, supports both Maven and Ant. Uses ANTLR.
- DoctorJ - Uses JavaCC. Checks Javadoc, syntax and calculates metrics.
- ESC/Java - Finds null dereference errors, array bounds errors, type cast errors, and race conditions. Uses Java Modeling Language annotations.
- Hammurapi - Uses ANTLR, excellent documentation, lots of rules
- Jamit - bytecode analyzer, nice graphs
- JCSC - Does a variety of coding standard checks, uses JavaCC and the GNU Regexp package.
- Jikes - More than a compiler; now it reports code warnings too
- JLint - Written in C++. Uses data flow analysis and a lock graph to do lots of synchronization checks. Operates on class files, not source code.
- JPathFinder - A verification VM written by NASA; supports a subset of the Java packages
- JWiz - Research project, checks some neat stuff, like if you create a Button without adding an ActionListener to it. Neat.

Static Code Analysis Tools : Commercial Prodcuts

- AppPerfect - 220 rules, produces PDF/Excel reports, std version is free, professional version $500 and includes auto-fixing problems
- Assent - The usual stuff, seems pretty complete.
- Aubjex - Rules aren't listed online. Appears to have some code modification stuff, which would be cool to have in PMD. $299.
- AzoJavaChecker - Rules aren't listed online so it's hard to tell what they have. Not sure how much it costs since I don't know German.
- CodePro AnalytiX - $1,299, Eclipse plug-in, extensive audit rules, JUnit test generation/editing, code coverage and analysis
- Enerjy Java Code Analyser - 200 rules, lots of IDE plugins
- Flaw Detector - In beta, does control/data flow analysis to detect NullPointerExceptions
- JStyle - $995, nice folks, lots of metrics and rules
- JTest - Very nice with tons of features, but also very expensive and requires a running X server (or Xvfb) to run on Linux. They charge $500 to move a license from one machine to another.
- Lint4J - Lock graph, DFA, and type analysis, many EJB checks

FindBugs

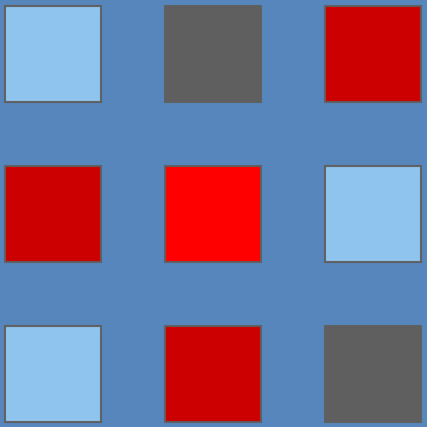
- FindBugs looks for bugs in Java programs.
- It is based on the concept of *bug patterns*.
- A bug pattern is a code idiom that is often an error.
- FindBugs uses *static analysis* to inspect Java bytecode for occurrences of bug patterns.
- FindBugs can find bugs by simply inspecting a program's code: executing the program is not necessary.

Findbugs

INPUT

- Set of ".class" files containing byte-code
- Configurations

PROCESSING



Detectors

OUTPUT

- Bug Pattern Code
- Source Line Number
- Descriptive Message

FindBugs Bug Descriptions

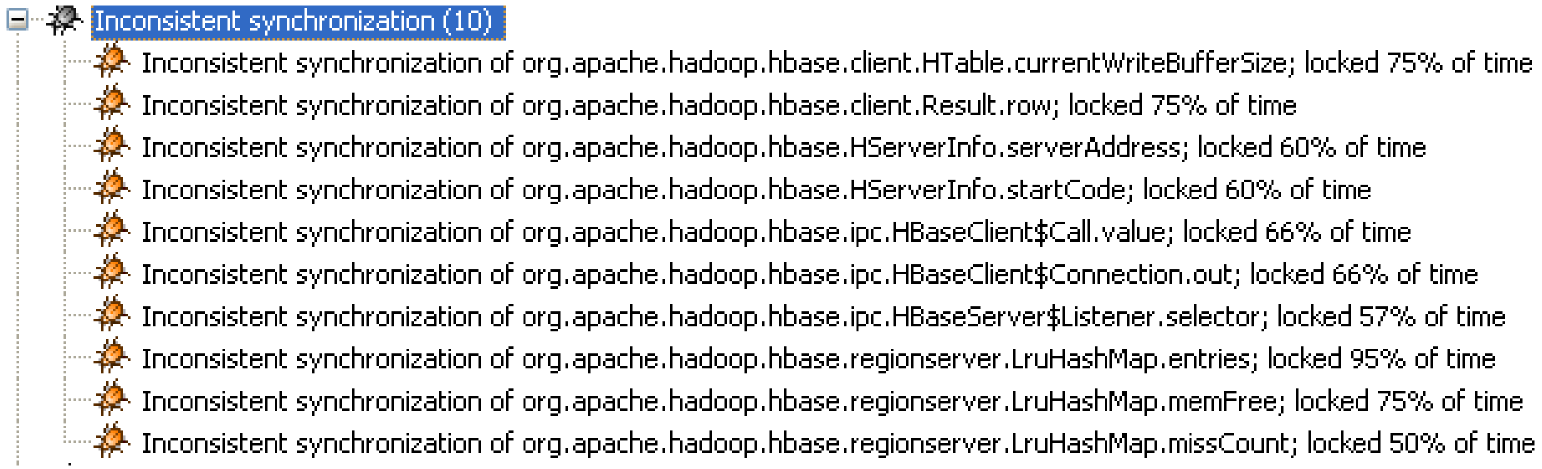
Current Version (version 1.3.9.) include following categories:

- Bad practice
- Correctness
- Malicious code vulnerability
- Multithreaded correctness
- Performance
- Dodgy

To see details

Potential Bugs Discovered in HBase 0.20.0

- **Bug:** Inconsistent synchronization of org.apache.hadoop.hbase.client.HTable.currentWriteBufferSize; locked 75% of time.
- The fields of this class appear to be accessed inconsistently with respect to synchronization.
- A typical bug matching this bug pattern is forgetting to synchronize one of the methods in a class that is intended to be thread-safe.



```
Inconsistent synchronization (10)
    Inconsistent synchronization of org.apache.hadoop.hbase.client.HTable.currentWriteBufferSize; locked 75% of time
    Inconsistent synchronization of org.apache.hadoop.hbase.client.Result.row; locked 75% of time
    Inconsistent synchronization of org.apache.hadoop.hbase.HServerInfo.serverAddress; locked 60% of time
    Inconsistent synchronization of org.apache.hadoop.hbase.HServerInfo.startCode; locked 60% of time
    Inconsistent synchronization of org.apache.hadoop.hbase.ipc.HBaseClient$Call.value; locked 66% of time
    Inconsistent synchronization of org.apache.hadoop.hbase.ipc.HBaseClient$Connection.out; locked 66% of time
    Inconsistent synchronization of org.apache.hadoop.hbase.ipc.HBaseServer$Listener.selector; locked 57% of time
    Inconsistent synchronization of org.apache.hadoop.hbase.regionserver.LruHashMap.entries; locked 95% of time
    Inconsistent synchronization of org.apache.hadoop.hbase.regionserver.LruHashMap.memFree; locked 75% of time
    Inconsistent synchronization of org.apache.hadoop.hbase.regionserver.LruHashMap.missCount; locked 50% of time
```

Potential Bugs Discovered in HBase 0.20.0

- **Bug:** Long is incompatible with expected argument type String
  Pattern id: GC_UNRELATED_TYPES, **type:** GC, **category:** CORRECTNESS
- This call to a generic collection method contains an argument with an incompatible class from that of the collection's parameter (i.e., the type of the argument is neither a supertype nor a subtype of the corresponding generic type argument). Therefore, it is unlikely that the collection contains any objects that are equal to the method argument used here. Most likely, the wrong value is being passed to the method.

Potential Bugs Discovered in HBase 0.20.0

- **Bug:** Call to equals() comparing different types
  Pattern id: EC_UNRELATED_TYPES, **type:** EC, **category:** CORRECTNESS

  This method calls equals(Object) on two references of different class types with no common subclasses. According to the contract of equals(), objects of different classes should always compare as unequal; therefore, according to the contract defined by java.lang.Object.equals(Object), the result of this comparison will always be false at runtime.

- Problem patterns found in HBase 0.20.0 by Findbugs

  Check all the problems reported by Findbugs, find those are real problems, and make some simple analysis and comments.

- Full report of problems found in HBase 0.20.0 by Findbugs

And others found in HBase deserve our attentation

1. Call to equals() comparing different types
2. Class defines compareTo(...) and uses Object.equals()
3. Class defines equals() and uses Object.hashCode()
4. Constructor invokes Thread.start()
5. Dead store to local variable
6. equals method always returns false
7. Inefficient use of keySet iterator instead of entrySet iterator
8. instanceof will always return false
9. integral division result cast to double or float
10. Invocation of hashCode on an array
11. Invocation of toString on an array
12. Method call passes null for nonnull parameter

And others found in HBase deserve our attentation(cont..)

13. Method calls Thread.sleep() with a lock held
14. Method concatenates strings using + in a loop
15. Method does not release lock on all exception paths
16. Method invokes inefficient Boolean constructor; use Boolean.valueOf(...) instead
17. Method invokes inefficient Number constructor; use static valueOf instead
18. Method invokes System.exit(...)
19. Method may fail to close stream
20. Method might ignore exception
21. Method names should start with a lower case letter
22. Method with Boolean return type returns explicit null
23. Naked notify

And others found in HBase deserve our attentation(cont..)

24. Non-transient non-serializable instance field in serializable class
25. Nullcheck of value previously dereferenced
26. Possible null pointer dereference
27. Private method is never called
28. Random object created and used only once
29. Redundant nullcheck of value known to be non-null
30. Result of integer multiplication cast to long
31. Should be a static inner class
32. Synchronization performed on util.concurrent instance
33. Test for floating point equality
34. TestCase defines setUp that doesn't call super.setUp()

And others found in HBase deserve our attentation(cont..)

35. TestCase defines tearDown that doesn't call super.tearDown()
36. Unconditional wait
37. Unread field
38. Unread field: should this field be static?
39. Unsigned right shift cast to short/byte
40. Unusual equals method
41. Vacuous comparison of integer value

And more ….

FindBugs' Recommendation

- First, review the **correctness** warnings. We feel confident that developers would want to fix most of the high and medium priority correctness warnings we report. Once you've reviewed those, you might want to look at some of the other categories.
- In other categories, such as **Bad practice** and **Dodgy** code, we accept more false positives. You might decide that a pattern bug pattern isn't relevant for your code base, and even for the bug patterns relevant to your code base, perhaps only a minority will reflect problems serious enough to convince you to change your code.

Static analysis VS **Functional tests**

- Functional tests apply only to specific parts of the code.
- Functional tests test the most important parts of the code under specific conditions
- Static analysis tests almost all code
- Static analysis pushes up average code quality

Advantages of Static Analysis Tools

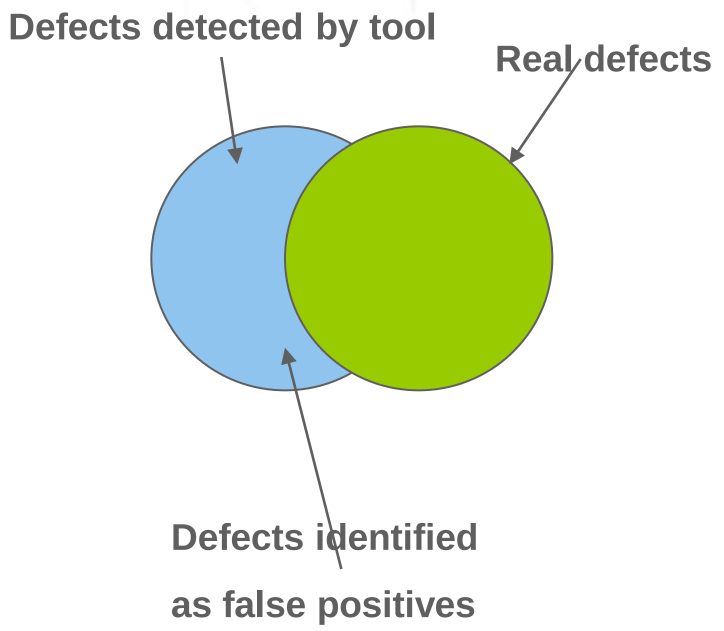
- Spots bug patterns without running the code
- Automates the process of looking through code for defects
- Can speed up code inspection process
- Can make the process of finding bugs more effective; objectivity
- At a gross level can give an indicator of quality

Disadvantages of Running Static Analysis Tools

- False positives
- Complains over minor infringements
- Prioritising is an issue

- *Developers need to be motivated* to read the reports

- Bugs patterns can be at times obscure



Combining Static Analysis Tools with Code Inspection

- Static analysis tools can complement code inspection by speeding up the process
- The results of the tools have to be examined in order to determine if the bugs reported are false positives and if bugs are missed
- Conclusion: These approaches can complement each other

Integrate the tools into our project

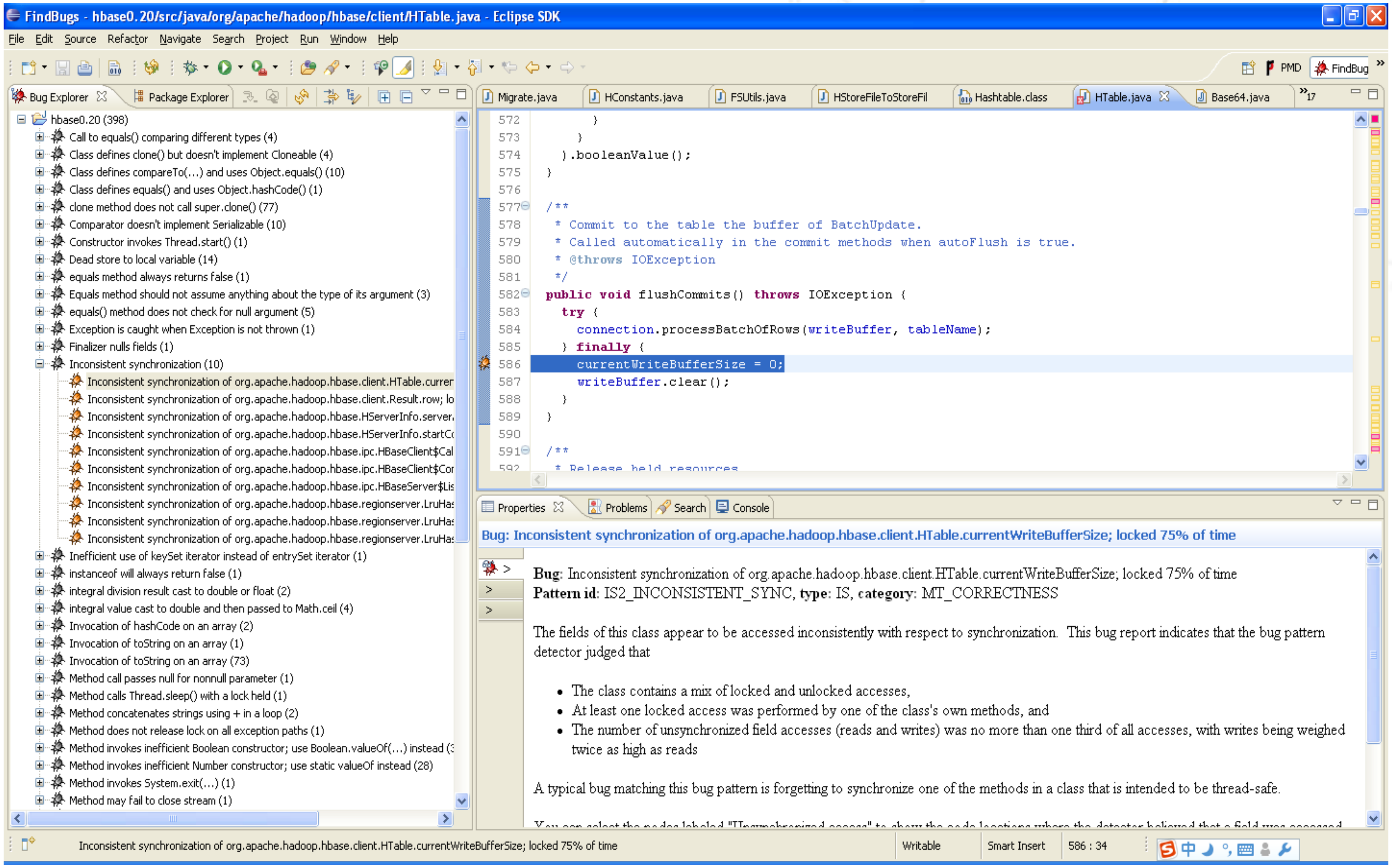
- Eclipse plugin

  Perform static code check in developing env.
- Eclipse update sites
- FindBugs: http://findbugs.cs.umd.edu/eclipse/
- PMD: http://pmd.sf.net/eclipse

- Ant

  Require adding such task into build script for Ant, which is a popular Java build and deployment tool used in our project. The build script can automatically run static code analysis for each build.

FindBugs™ Eclipse plugin



FindBugs - hbase0.20/src/java/org/apache/hadoop/hbase/client/HTable.java - Eclipse SDK

hbase0.20 (398)
- Call to equals() comparing different types (4)
- Class defines clone() but doesn't implement Cloneable (4)
- Class defines compareTo(...) and uses Object.equals() (10)
- Class defines equals() and uses Object.hashCode() (1)
- clone method does not call super.clone() (77)
- Comparator doesn't implement Serializable (10)
- Constructor invokes Thread.start() (1)
- Dead store to local variable (14)
- equals method always returns false (1)
- Equals method should not assume anything about the type of its argument (3)
- equals() method does not check for null argument (5)
- Exception is caught when Exception is not thrown (1)
- Finalizer nulls fields (1)
- Inconsistent synchronization (10)
  - Inconsistent synchronization of org.apache.hadoop.hbase.client.HTable.currer
  - Inconsistent synchronization of org.apache.hadoop.hbase.client.Result.row; lo
  - Inconsistent synchronization of org.apache.hadoop.hbase.HServerInfo.server.
  - Inconsistent synchronization of org.apache.hadoop.hbase.HServerInfo.startC
  - Inconsistent synchronization of org.apache.hadoop.hbase.ipc.HBaseClient$Cal
  - Inconsistent synchronization of org.apache.hadoop.hbase.ipc.HBaseClient$Cor
  - Inconsistent synchronization of org.apache.hadoop.hbase.ipc.HBaseServer$Lis
  - Inconsistent synchronization of org.apache.hadoop.hbase.regionserver.LruHas
  - Inconsistent synchronization of org.apache.hadoop.hbase.regionserver.LruHas
  - Inconsistent synchronization of org.apache.hadoop.hbase.regionserver.LruHas
- Inefficient use of keySet iterator instead of entrySet iterator (1)
- instanceof will always return false (1)
- integral division result cast to double or float (2)
- integral value cast to double and then passed to Math.ceil (4)
- Invocation of hashCode on an array (2)
- Invocation of toString on an array (1)
- Invocation of toString on an array (73)
- Method call passes null for nonnull parameter (1)
- Method calls Thread.sleep() with a lock held (1)
- Method concatenates strings using + in a loop (2)
- Method does not release lock on all exception paths (1)
- Method invokes inefficient Boolean constructor; use Boolean.valueOf(...) instead (
- Method invokes inefficient Number constructor; use static valueOf instead (28)
- Method invokes System.exit(...) (1)
- Method may fail to close stream (1)

```
        }
      }
    }).booleanValue();
  }

  /**
   * Commit to the table the buffer of BatchUpdate.
   * Called automatically in the commit methods when autoFlush is true.
   * @throws IOException
   */
  public void flushCommits() throws IOException {
    try {
      connection.processBatchOfRows(writeBuffer, tableName);
    } finally {
      currentWriteBufferSize = 0;
      writeBuffer.clear();
    }
  }

  /**
   * Release held resources
```

Bug: Inconsistent synchronization of org.apache.hadoop.hbase.client.HTable.currentWriteBufferSize; locked 75% of time

Bug: Inconsistent synchronization of org.apache.hadoop.hbase.client.HTable.currentWriteBufferSize; locked 75% of time
Pattern id: IS2_INCONSISTENT_SYNC, **type**: IS, **category**: MT_CORRECTNESS

The fields of this class appear to be accessed inconsistently with respect to synchronization. This bug report indicates that the bug pattern detector judged that

- The class contains a mix of locked and unlocked accesses,
- At least one locked access was performed by one of the class's own methods, and
- The number of unsynchronized field accesses (reads and writes) was no more than one third of all accesses, with writes being weighed twice as high as reads

A typical bug matching this bug pattern is forgetting to synchronize one of the methods in a class that is intended to be thread-safe.

Inconsistent synchronization of org.apache.hadoop.hbase.client.HTable.currentWriteBufferSize; locked 75% of time | Writable | Smart Insert | 586 : 34

Using the FindBugs™ Ant task

- **1. Installing the Ant task**

  copy *$FINDBUGS_HOME*/lib/findbugs-ant.jar into the lib subdirectory of your Ant installation.

- **2. Modifying build.xml**

```
<taskdef name="findbugs" classname="edu.umd.cs.findbugs.anttask.FindBugsTask"/>

 <property name="findbugs.home" value="E:\WorkDoc\WhiteBox\findbugs-1.3.9" />

 <property name="jvmargs" value="-server -Xss1m -Xmx800m -Duser.language=en -
Duser.region=EN -Dfindbugs.home=${findbugs.home}" />

  <target name="findbugs" depends="jar">

    <findbugs
home="${findbugs.home}“ output="xml“   jvmargs="${jvmargs}“ outputFile="HelloFindBugs.xml" >
                    <sourcePath path="${basedir}/src" />

                  <class location="${basedir}/build/hbase-0.20.0.jar" />

    </findbugs>

   </target>
```

Details config

In Our Project

- Suggest adding the static code analysis report checking to developing process.
- There might not be anything wrong with this code, but it is worth reviewing.
- Need a negotiation between QA and RD. ☺

Discussion Topics & Answers

- Will we use the static code analysis tools?

  Yes -Harry, Roland
- How can we adapt it into our project?

  Try to use it, and find a practical way. -Harry
- Who will be responsible for checking the code analysis report? The role of QA and RD in this process?

  QA -Roland
- Which one? Findbugs or PMD?

  Try FindBugs first -Roland
- And other questions?

  No -All

WhiteBox Testing in GMS - Group Code Review

- A meeting required all the RD and QA .
- Introduce the detail design and implementation of each module (package and class in java) by RD
- Introduce the main technology used
- The difficulty in implementation.
- Briefly review the code.
- Actually a process of study than a review.

WhiteBox Testing in GMS - Pair Code Review

RD	QA	Module
All RD	Ping	com.trendmicro.spn.gms com.trendmicro.spn.gms.util com.trendmicro.util
Ken	Thomas	com.trendmicro.spn.gms.datatype com.trendmicro.spn.gms.global com.trendmicro.spn.gms.message com.trendmicro.spn.gms.rest
Harry	Roland	com.trendmicro.spn.gms.dblayer
Sandy	Forrest	com.trendmicro.spn.gms.job
John	Sean	com.trendmicro.spn.gms.sqlqueryparse

WhiteBox Testing in GMS – an implementation for feature test

- Sometimes read or debug code when to trace the root cause of bug

- Read or debug code when it is hard to construct the testing environment.

Discussions

- Any suggestion
- Experience sharing.

- "I don't think anybody tests enough of anything"
- Untested code is full of bugs. The fewer tests you have, the more undiscovered bugs lurk in your code.

Code coverage

- **Code coverage** is a measure used in software testing. It describes the degree to which the source code of a program has been tested.

- Test engineers can look at code coverage test results to help them devise test cases and input or configuration sets that will increase the code coverage over vital functions.

code coverage tools

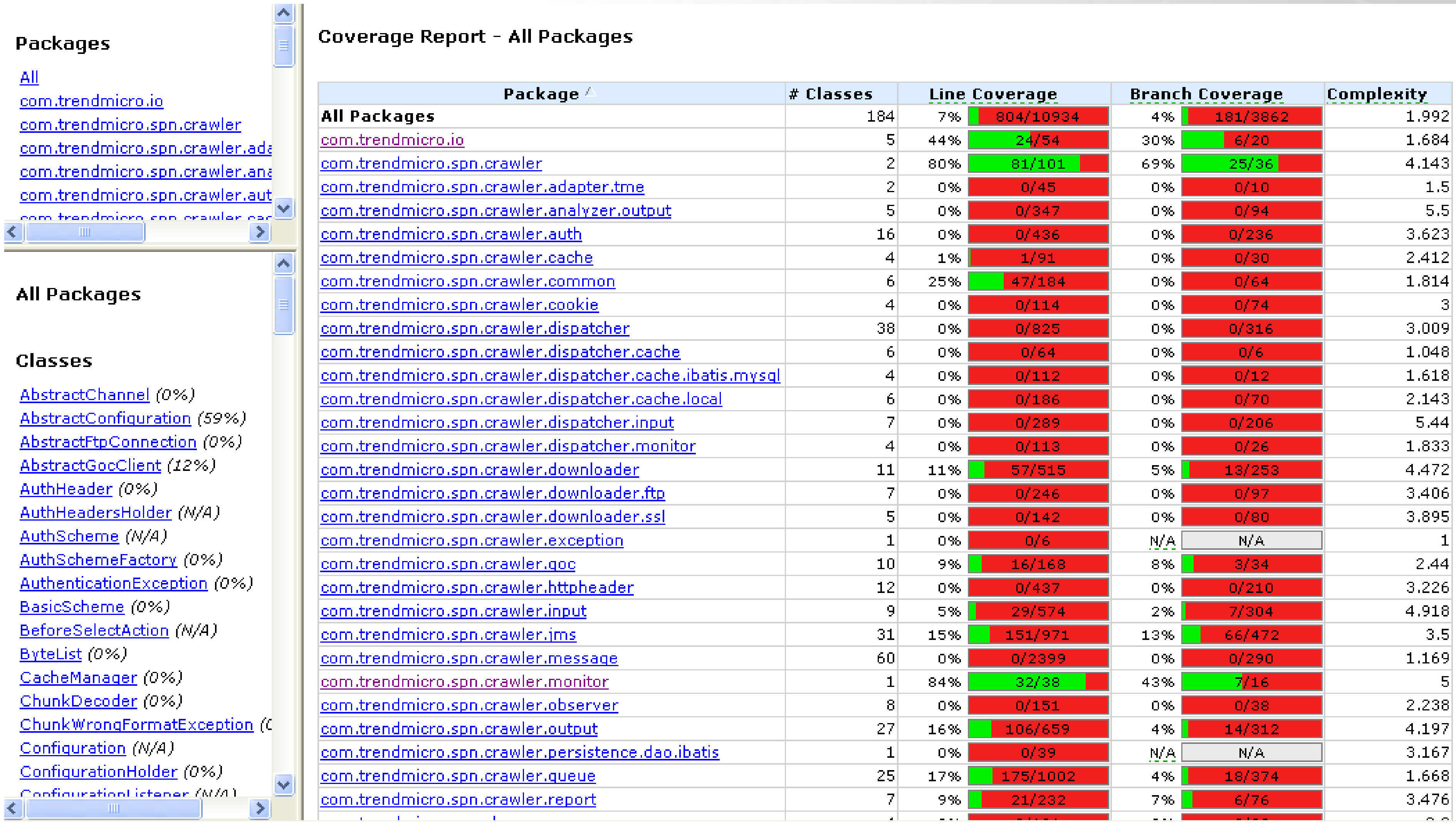
- Atlassian Clover - a Java code coverage and test visualization tool which also displays coverage per-test. Free for open source projects
- BullseyeCoverage - C and C++ code coverage tool
- Cobertura - a free Java tool that calculates the percentage of code accessed by tests
- CodeCover - Java / Cobol code coverage tool for use by shell in win / linux, apache ant script, or as Eclipse-Plugin (including boolean analyzer and correlation matrix), XML Reports
- Devel::Cover - Code coverage metrics for Perl
- EMMA - a free Java code coverage tool
- gcov - Code coverage test for GCC compiler
- gcov-kernel - gcov support for the Linux kernel
- ggcov - GTK+ GUI for gcov
- IBM OLIVER (CICS interactive test/debug), application program test/debugging and code coverage tool for IBM CICS
- Insure++ - a coverage of source code of application tested with functional tests.
- iSYSTEM winIDEA - measures coverage on a wide variety of embedded processors. It works by recoding execution directly on hardware, without instrumenting code or modifying the program and in real-time.
- LDRA Testbed measures statement coverage, branch/decision coverage, LCSAJ Coverage, procedure/function call coverage, branch condition coverage, branch condition combination coverage and modified condition decision coverage (MC/DC) for DO-178B Level A.
- Jtest, C++test, .Test - calculate percentage of code covered by tests.

Cobertura

- Can be executed from ant or from the command line.
- Instruments Java bytecode after it has been compiled.
- Can generate reports in HTML or XML.
- Shows the percentage of lines and branches covered for each class, each package, and for the overall project.
- Shows the code complexity of each class, and the average code complexity for each package, and for the overall product.
- Can sort HTML results by class name, percent of lines covered, percent of branches covered, etc. And can sort in ascending or decending order.

Cobertura Report In Package

Securing Your Web World



Packages

All
com.trendmicro.io
com.trendmicro.spn.crawler
com.trendmicro.spn.crawler.ada
com.trendmicro.spn.crawler.ana
com.trendmicro.spn.crawler.aut

All Packages

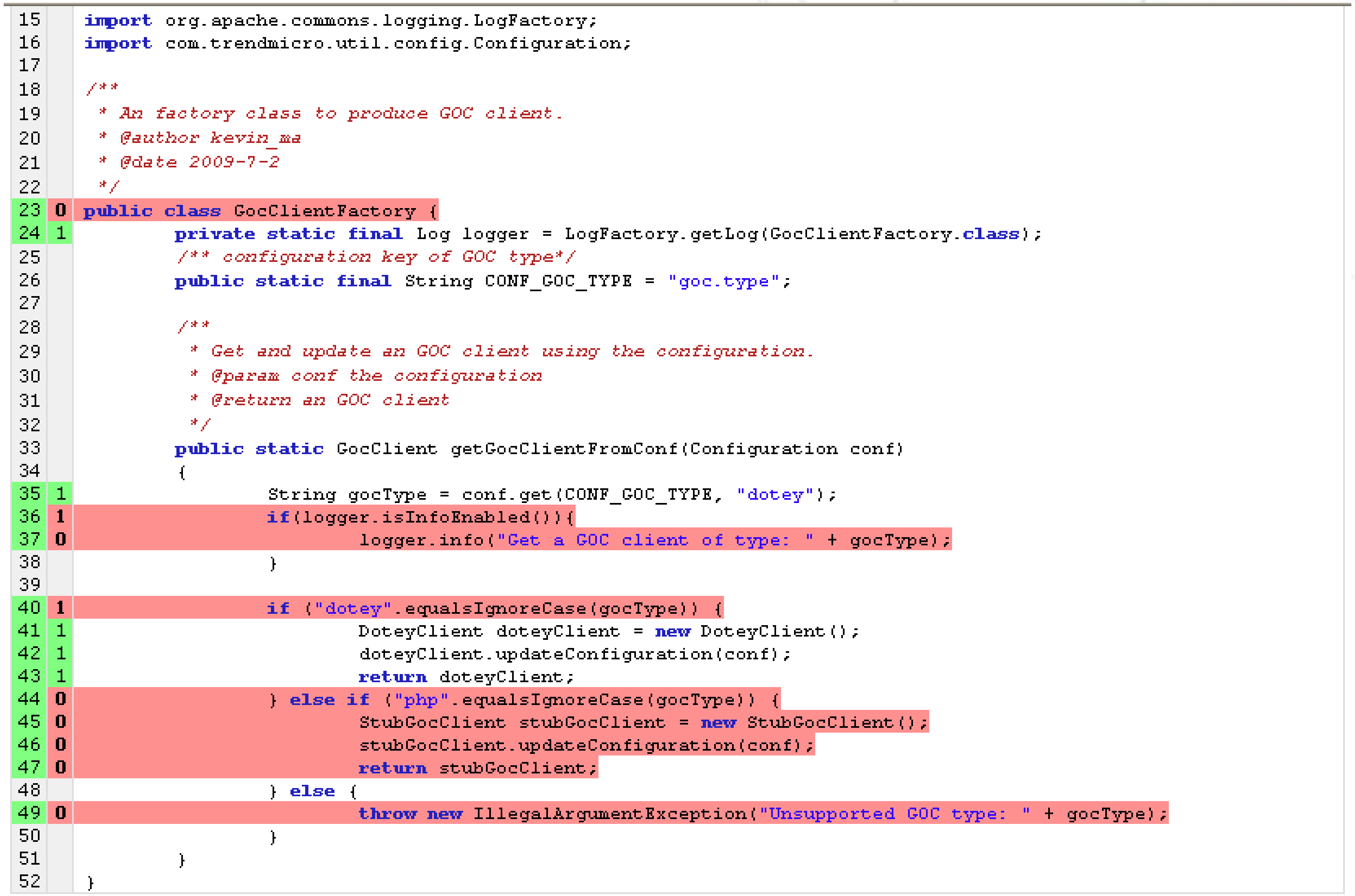
Classes

AbstractChannel *(0%)*
AbstractConfiguration *(59%)*
AbstractFtpConnection *(0%)*
AbstractGocClient *(12%)*
AuthHeader *(0%)*
AuthHeadersHolder *(N/A)*
AuthScheme *(N/A)*
AuthSchemeFactory *(0%)*
AuthenticationException *(0%)*
BasicScheme *(0%)*
BeforeSelectAction *(N/A)*
ByteList *(0%)*
CacheManager *(0%)*
ChunkDecoder *(0%)*
ChunkWrongFormatException *(*
Configuration *(N/A)*
ConfigurationHolder *(0%)*

Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	184	7%	804/10934	4%	181/3862	1.992
com.trendmicro.io	5	44%	24/54	30%	6/20	1.684
com.trendmicro.spn.crawler	2	80%	81/101	69%	25/36	4.143
com.trendmicro.spn.crawler.adapter.tme	2	0%	0/45	0%	0/10	1.5
com.trendmicro.spn.crawler.analyzer.output	5	0%	0/347	0%	0/94	5.5
com.trendmicro.spn.crawler.auth	16	0%	0/436	0%	0/236	3.623
com.trendmicro.spn.crawler.cache	4	1%	1/91	0%	0/30	2.412
com.trendmicro.spn.crawler.common	6	25%	47/184	0%	0/64	1.814
com.trendmicro.spn.crawler.cookie	4	0%	0/114	0%	0/74	3
com.trendmicro.spn.crawler.dispatcher	38	0%	0/825	0%	0/316	3.009
com.trendmicro.spn.crawler.dispatcher.cache	6	0%	0/64	0%	0/6	1.048
com.trendmicro.spn.crawler.dispatcher.cache.ibatis.mysql	4	0%	0/112	0%	0/12	1.618
com.trendmicro.spn.crawler.dispatcher.cache.local	6	0%	0/186	0%	0/70	2.143
com.trendmicro.spn.crawler.dispatcher.input	7	0%	0/289	0%	0/206	5.44
com.trendmicro.spn.crawler.dispatcher.monitor	4	0%	0/113	0%	0/26	1.833
com.trendmicro.spn.crawler.downloader	11	11%	57/515	5%	13/253	4.472
com.trendmicro.spn.crawler.downloader.ftp	7	0%	0/246	0%	0/97	3.406
com.trendmicro.spn.crawler.downloader.ssl	5	0%	0/142	0%	0/80	3.895
com.trendmicro.spn.crawler.exception	1	0%	0/6	N/A	N/A	1
com.trendmicro.spn.crawler.goc	10	9%	16/168	8%	3/34	2.44
com.trendmicro.spn.crawler.httpheader	12	0%	0/437	0%	0/210	3.226
com.trendmicro.spn.crawler.input	9	5%	29/574	2%	7/304	4.918
com.trendmicro.spn.crawler.jms	31	15%	151/971	13%	66/472	3.5
com.trendmicro.spn.crawler.message	60	0%	0/2399	0%	0/290	1.169
com.trendmicro.spn.crawler.monitor	1	84%	32/38	43%	7/16	5
com.trendmicro.spn.crawler.observer	8	0%	0/151	0%	0/38	2.238
com.trendmicro.spn.crawler.output	27	16%	106/659	4%	14/312	4.197
com.trendmicro.spn.crawler.persistence.dao.ibatis	1	0%	0/39	N/A	N/A	3.167
com.trendmicro.spn.crawler.queue	25	17%	175/1002	4%	18/374	1.668
com.trendmicro.spn.crawler.report	7	9%	21/232	7%	6/76	3.476

Cobertura Report In Class



```
15     import org.apache.commons.logging.LogFactory;
16     import com.trendmicro.util.config.Configuration;
17
18     /**
19      * An factory class to produce GOC client.
20      * @author kevin_ma
21      * @date 2009-7-2
22      */
23  0  public class GocClientFactory {
24  1          private static final Log logger = LogFactory.getLog(GocClientFactory.class);
25             /** configuration key of GOC type*/
26             public static final String CONF_GOC_TYPE = "goc.type";
27
28             /**
29              * Get and update an GOC client using the configuration.
30              * @param conf the configuration
31              * @return an GOC client
32              */
33             public static GocClient getGocClientFromConf(Configuration conf)
34             {
35  1                  String gocType = conf.get(CONF_GOC_TYPE, "dotey");
36  1                  if(logger.isInfoEnabled()){
37  0                          logger.info("Get a GOC client of type: " + gocType);
38                     }
39
40  1                  if ("dotey".equalsIgnoreCase(gocType)) {
41  1                          DoteyClient doteyClient = new DoteyClient();
42  1                          doteyClient.updateConfiguration(conf);
43  1                          return doteyClient;
44  0                  } else if ("php".equalsIgnoreCase(gocType)) {
45  0                          StubGocClient stubGocClient = new StubGocClient();
46  0                          stubGocClient.updateConfiguration(conf);
47  0                          return stubGocClient;
48                     } else {
49  0                          throw new IllegalArgumentException("Unsupported GOC type: " + gocType);
50                     }
51             }
52     }
```

Discussions

- How can we adopt this coverage tool in our product
- Experience sharing

- One of my article in SPN wiki: Do coverage for our test cases

Open-source or Noncommercial for static code analysis

- **NET (C#, VB.NET and all .NET compatible languages)**
- FxCop — Free static analysis for Microsoft .NET programs that compile to CIL. Standalone and integrated in some Microsoft Visual Studio editions. From Microsoft.
- StyleCop — Analyzes C# source code to enforce a set of style and consistency rules. It can be run from inside of Microsoft Visual Studio or integrated into an MSBuild project. Free download from Microsoft
- **C**
- Sparse — A tool designed to find faults in the Linux kernel.
- Splint — An open source evolved version of Lint (C language).
- — A tool designed to find most common type of programming errors without generating too much output.
- BLAST (Berkeley Lazy Abstraction Software verification Tool) — a software model checker for C programs based on lazy abstraction.
- Frama-C — A static analysis framework for C.
- **C++**
- Cppcheck — can find memory leaks, buffer overruns and many other common errors.
- **Objective-C**
- Clang — the free Clang project includes a static analyzer. As of version 3.2, this analyzer is included in Xcode.[1]
- **Perl**
- Perl::Critic — module and program to help find deviations from commonly accepted best practices

Commercial products for static code analysis

- **NET**
- Products covering multiple .NET languages.
- ReSharper — Add-on for Visual Studio 2003/2005 from the creators of IntelliJ IDEA, which also provides static code analysis for C#.
- NDepend — Simplifies managing a complex .NET code base by analyzing code dependencies, by defining design rules, by doing impact analysis, and by comparing different versions of the code (all .NET languages supported)
- CodeIt.Right — combines Static Code Analysis and automatic Refactoring to best practices which allows automatically correct code errors and violations. Supports both C# and VB.NET.
- Gendarme — extensible rule-based tool to find problems in .NET applications and libraries, particularly those that contain code in ECMA CIL format.
- **C/C++**
- Abraxas Software CodeCheck — programmable static analysis and style checker for C and C++ code.
- — Run-time error analyzer for C
- Green Hills Software DoubleCheck — static analysis for C and C++ code.
- — A static analysis tool for C and C++ programs
- LDRA Testbed — A software analysis and testing tool suite for C & C++.
- Microsoft — The "Analyze Tool" included with Microsoft Visual Studio Team Editions.
- Microsoft (PFD) — An extension to PREfast to allow better analysis of Windows device drivers.
- Microsoft (SDV) — Performs detailed code path analysis for Windows device drivers.
- — The Program Analyzer Generator.
- PC-Lint — A software analysis tool for C & C++.
- QA-C (and QA-C++) — deep static analysis of C for quality assurance and guideline enforcement.
- Red Lizard's Goanna — Static analysis for C/C++ in Eclipse and Visual Studio.
- — analyzes C, C++ code to detect 64-bit portability issues.
- CppDepend — Simplifies managing a complex C++ code

Resources

- 10 Code Coverage Tools for C & C++
- Easy code review tools
- 10+ free tools for static code analysis
- http://www.ibm.com/developerworks/cn/java/j-findbug1/
- http://findbugs.sourceforge.net/
- http://pmd.sourceforge.net/

Thank you !!!

PMD

PMD scans Java source code and looks for potential problems like:

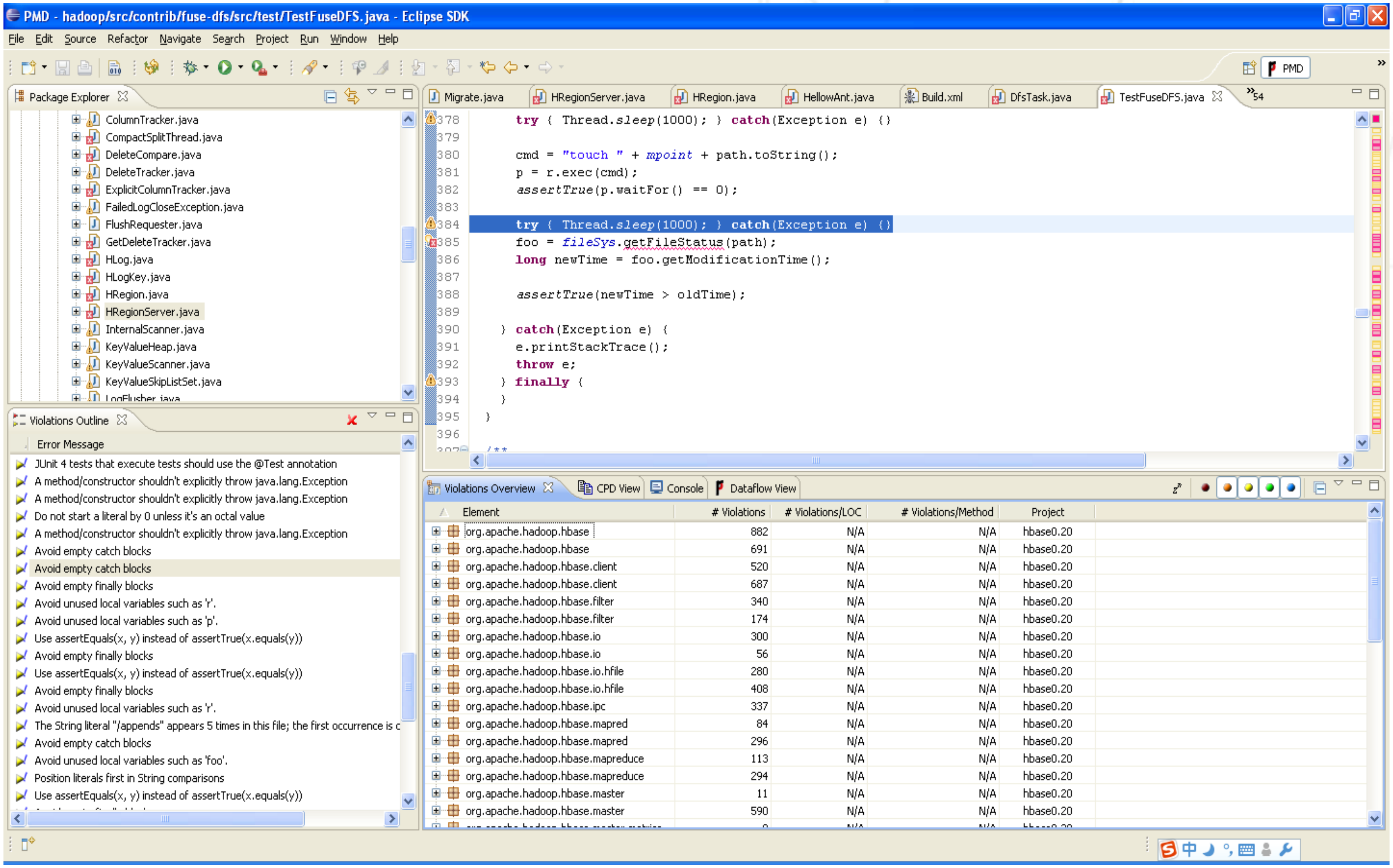
- Possible bugs - empty try/catch/finally/switch statements
- Dead code - unused local variables, parameters and private methods
- Suboptimal code - wasteful String/StringBuffer usage
- Overcomplicated expressions - unnecessary if statements, for loops that could be while loops
- Duplicate code - copied/pasted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

- PMD is integrated with JDeveloper, Eclipse, JEdit, JBuilder, BlueJ, CodeGuide, NetBeans/Sun Java Studio Enterprise/Creator, IntelliJ IDEA, TextPad, Maven, Ant, Gel, JCreator, and Emacs

The rules of PMD

- Android
- Basic
- **Braces**
- Code Size
- Clone
- Controversial
- Coupling
- Design
- Finalizers
- Import Statements
- J2EE
- Javabeans
- JUnit Tests
- Logging (Java)
- Logging (Jakarta)
- Migrating
- Naming
- Optimizations
- Strict Exceptions
- Strings
- Sun Security
- Unused Code
- Java Server Pages
- Java Server Faces

PMD Eclipse plugin


PMD - hadoop/src/contrib/fuse-dfs/src/test/TestFuseDFS.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
PMD
Package Explorer
Migrate.java HRegionServer.java HRegion.java HellowAnt.java Build.xml DfsTask.java TestFuseDFS.java »54
Violations Outline
Violations Overview CPD View Console Dataflow View

Package Explorer:

- ColumnTracker.java
- CompactSplitThread.java
- DeleteCompare.java
- DeleteTracker.java
- ExplicitColumnTracker.java
- FailedLogCloseException.java
- FlushRequester.java
- GetDeleteTracker.java
- HLog.java
- HLogKey.java
- HRegion.java
- HRegionServer.java
- InternalScanner.java
- KeyValueHeap.java
- KeyValueScanner.java
- KeyValueSkipListSet.java

```
        try { Thread.sleep(1000); } catch(Exception e) {}

        cmd = "touch " + mpoint + path.toString();
        p = r.exec(cmd);
        assertTrue(p.waitFor() == 0);

        try { Thread.sleep(1000); } catch(Exception e) {}
        foo = fileSys.getFileStatus(path);
        long newTime = foo.getModificationTime();

        assertTrue(newTime > oldTime);

      } catch(Exception e) {
        e.printStackTrace();
        throw e;
      } finally {
      }
    }
```

Error Message

- JUnit 4 tests that execute tests should use the @Test annotation
- A method/constructor shouldn't explicitly throw java.lang.Exception
- A method/constructor shouldn't explicitly throw java.lang.Exception
- Do not start a literal by 0 unless it's an octal value
- A method/constructor shouldn't explicitly throw java.lang.Exception
- Avoid empty catch blocks
- Avoid empty catch blocks
- Avoid empty finally blocks
- Avoid unused local variables such as 'r'.
- Avoid unused local variables such as 'p'.
- Use assertEquals(x, y) instead of assertTrue(x.equals(y))
- Avoid empty finally blocks
- Use assertEquals(x, y) instead of assertTrue(x.equals(y))
- Avoid empty finally blocks
- Avoid unused local variables such as 'r'.
- The String literal "/appends" appears 5 times in this file; the first occurrence is c
- Avoid empty catch blocks
- Avoid unused local variables such as 'foo'.
- Position literals first in String comparisons
- Use assertEquals(x, y) instead of assertTrue(x.equals(y))

Element	# Violations	# Violations/LOC	# Violations/Method	Project
org.apache.hadoop.hbase	882	N/A	N/A	hbase0.20
org.apache.hadoop.hbase	691	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.client	520	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.client	687	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.filter	340	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.filter	174	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.io	300	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.io	56	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.io.hfile	280	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.io.hfile	408	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.ipc	337	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.mapred	84	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.mapred	296	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.mapreduce	113	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.mapreduce	294	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.master	11	N/A	N/A	hbase0.20
org.apache.hadoop.hbase.master	590	N/A	N/A	hbase0.20

Using the PMD™ Ant task

- **1. Installing the Ant task**

  copy *$PMD_HOME*/lib/*.jar into the lib subdirectory of your Ant installation.

- **2. Modifying build.xml**

```
<target name="pmd">
 <taskdef name="pmd" classname="net.sourceforge.pmd.ant.PMDTask"/>
 <pmd shortFilenames="true">
  <ruleset>rulesets/favorites.xml</ruleset>
  <ruleset>basic</ruleset>
  <formatter type="html" toFile="pmd_report.html" linkPrefix="http://pmd.sourceforge.net/xref/"/>
  <fileset dir="D:\hadoop\pmd_hbase-0.20-rc\src\">
  </fileset>
 </pmd>
</target>
```

Details config