Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Java software verification tools: evaluation and recommended methodology

F. Painchaud
R. Carbone
DRDC Valcartier

Defence R&D Canada – Valcartier
Technical Memorandum
DRDC Valcartier TM 2005-226
February 2007

Canada

Java software verification tools: evaluation and recommended methodology

F. Painchaud

R. Carbone
DRDC Valcartier

Defence Research and Development Canada – Valcartier

Technical Memorandum

DRDC Valcartier TM 2005-226

February 2007

Author

Frédéric Painchaud, M.Sc.

Approved by

Yves van Chestein, M.Sc., ing.f.
Head/Information and Knowledge Management Section

Approved for release by

Christian Carrier
Chief Scientist

Abstract

Free and Open Source Software (FOSS) is now being adopted worldwide in many different organizations, such as governments, agencies, and companies, which want to reduce their software acquisitional cost, decrease their economic loss at the national level caused by commercial software imports, and develop national information technology expertise by means of access to source code. However, this current trend makes these organizations face a new challenge in software maturity, quality, and security assessment. Indeed, how can they be assured that the FOSS they are selecting among thousands of different projects is mature and secure enough? Hopefully, the availability of source code enables not only manual peer reviewing but also automatic and mechanical verification with a plethora of software tools.

This document categorizes, lists, and describes many of these automated software verification tools for Java. Java has been selected as the programming language because an increasing percentage of FOSS is now developed with this modern and more easily verifiable language. Moreover, this document details the best tools and gives a verification methodology in order to efficiently assess overall Java software quality, which is a pressing need for concerned managers.

Résumé

Les logiciels libres sont maintenant utilisés dans le monde entier par beaucoup d'organismes, tels que des gouvernements, des agences et des compagnies, qui veulent réduire leurs coûts d'acquisition de logiciels, diminuer leurs pertes économiques au niveau national causées par l'importation de logiciels commerciaux et développer une expertise nationale en technologies de l'information en étant en mesure d'accéder au code source. Cependant, cette tendance impose à ces organismes un nouveau défi en matière d'évaluation de la maturité, de la qualité et de la sécurité des logiciels. En effet, comment peuvent-ils être assurés que les logiciels libres qu'ils choisissent parmi des milliers de projets différents sont assez mûrs et sécuritaires ? Heureusement, la disponibilité du code source permet non seulement de le passer manuellement en revue mais également de le vérifier automatiquement et mécaniquement à l'aide d'une pléiade d'outils logiciels.

Ce document classe par catégories, énumère et décrit plusieurs de ces outils automatiques de vérification de logiciels pour Java. Le langage de programmation Java a été sélectionné puisqu'il y a actuellement une augmentation du pourcentage de logiciels libres développés à l'aide de ce langage moderne et plus facilement vérifiable.

De plus, ce document décrit les meilleurs outils et donne une méthodologie de vérification afin d'évaluer efficacement la qualité globale des logiciels Java, un besoin criant pour les gestionnaires actuels.

Executive summary

During the past five years, Free and Open Source Software (FOSS) has emerged as a concrete and credible technological opportunity. It has been determined in [1] that the Government of Canada could benefit from this opportunity by "an improved diversity in software supplies (custom code vs. FOSS vs. Commercial-off-the-shelf (COTS)), augmented security by source code auditing (and enhancement), and higher compliance with open standards and specifications that contribute to system interoperability".

However, the integration of any new piece of software into an organization is not made without risk, no matter where the software comes from (FOSS or COTS). Good planning and appropriate quality assessment is the key to smooth and consistent integration. It is often difficult to assess the quality of COTS software since the source code is generally not available. With the source code in hand, FOSS quality is although more amenable to detailed analysis. However, this is not easy to do because manually reviewing tens of thousands lines of code is counter-productive, if not humanly impossible. Therefore, automatic tools are needed to support this tedious task. The good news is that several of them exist and many are described and commented upon in this document.

This document focuses on tools for the Java programming language because it is now widely accepted and many open source projects are developed in it. Moreover, it is shown in this document that Java has many advantages over C++ which makes it a better language to use in order to develop secure and dependable applications. Java is simply easier to verify mechanically. In fact, much of the basic verifications are already performed by the compiler and the virtual machine, that is, by the platform itself!

All the tools described in this document have been classified in different groups of capabilities: supporting tests, detecting pitfalls, enforcing coding standards, calculating metrics, and decompiling. From all the tools available today and listed in this document, a selection of 26 has been made for evaluation purposes. The winners of this evaluation are AppPerfect DevSuite, a commercial code analyzer, PMD, an open source and free alternative code analyzer, and Simian, a commercial similarity analyzer. These tools make quick software quality assessment possible and in fact, their evaluation inspired the definition of a simple, comparative methodology to quickly assess software quality with the help of automatic tools. This six-step methodology can be summarized as follows (please refer to Section 7 for more details):

Step 0: Seek trusted sources and certified products,

Step 1: Search for signs of maturity and quality on the application website,

Step 2: Use FindBugs and PMD,

Step 3: Use PMD's Copy and Paste Detector,

Step 4: Use AppPerfect DevSuite (optional), and

Step 5: Use Simian Similarity Analyzer (optional).

F. Painchaud, R. Carbone; 2007; Java software verification tools: evaluation and recommended methodology; DRDC Valcartier TM 2005-226; Defence Research and Development Canada – Valcartier.

Sommaire

Au cours des cinq dernières années, les logiciels libres sont devenus une filière technologique concrète et crédible. On a déterminé dans [1] que le Gouvernement du Canada pourrait bénéficier de cette opportunité grâce à “une diversité améliorée dans ses acquisitions logicielles (code maison vs. logiciels libres vs. logiciels commerciaux), une sécurité accrue par l’évaluation (et l’amélioration) du code source et une meilleure conformité aux standards et spécifications ouverts qui contribuent à l’interopérabilité des systèmes”.

Cependant, l’intégration de tout nouveau logiciel dans une organisation n’est pas sans risque, et ce peu importe d’où provient ce logiciel (logiciel libre ou commercial). Une bonne planification et une évaluation adéquate de la qualité est la clé pour une intégration douce et cohérante. Néanmoins, il est habituellement difficile d’évaluer la qualité des logiciels commerciaux puisque leur code source est généralement non disponible. En ayant le code source, la qualité des logiciels libres est plus facilement analysable. Cependant, ce n’est pas nécessairement facile à faire parce que la révision manuelle de dizaines de milliers de lignes de code est contre-productive, voire humainement impossible. Ainsi, des outils automatiques sont nécessaires pour supporter cette tâche ardue. La bonne nouvelle est qu’une multitude d’outils existent et que plusieurs d’entre eux sont décrits et commentés dans ce document.

Ce document se concentre sur les outils pour le langage de programmation Java parce que celui-ci est maintenant largement accepté et utilisé dans plusieurs projets de logiciels libres. De plus, il est démontré dans ce document que Java a plusieurs avantages sur C++, ce qui fait de lui un meilleur langage pour développer des applications sécuritaires et fiables. Java est donc plus facile à vérifier mécaniquement. En fait, la plupart des vérifications sont déjà effectuées par le compilateur et la machine virtuelle, c’est-à-dire par la plate-forme elle-même !

Tous les outils décrits dans ce document ont été classés selon différentes catégories : support aux tests, détection d’erreurs courantes, renforcement de standards de programmation, calculs de métriques et décompilation. De tous les outils disponibles aujourd’hui et listés dans ce document, 26 ont été sélectionnés à des fins d’évaluation. Les gagnants de cette évaluation sont AppPerfect DevSuite, un analyseur de code commercial, PMD, un analyseur de code alternatif libre et gratuit, et Simian, un analyseur de similitudes commercial. Ces outils permettent l’évaluation rapide de la qualité logicielle et, en fait, leur évaluation a inspiré la définition d’une méthodologie simple et comparative pour évaluer rapidement la qualité logicielle à l’aide d’outils automatiques. Cette méthodologie en six étapes peut se résumer comme suit (voir la section 7 pour plus de détails) :

Étape 0 : Privilégier les sources de confiance et les produits certifiés,

Étape 1 : Rechercher des signes de maturité et de qualité sur le site internet de l'application,

Étape 2 : Utiliser FindBugs et PMD,

Étape 3 : Utiliser Copy and Paste Detector de PMD,

Étape 4 : Utiliser AppPerfect DevSuite (optionnelle) et

Étape 5 : Utiliser Simian Similarity Analyzer (optionnelle).

F. Painchaud, R. Carbone; 2007; Java software verification tools: evaluation and recommended methodology; DRDC Valcartier TM 2005-226; Recherche et développement pour la défense Canada – Valcartier.

Table of contents

List of figures

List of tables

Acknowledgments

We are very thankful to David Demers from Defence Research and Development Canada – Valcartier, Louis Bastarache and Denis Poussart (retired) from Laval University, John Davidson and Capt Karine Pellerin from Directorate Distributed Computing Engineering and Integration 3-5, and Chris Augi and Stéphanie Dion from Communications Security Establishment for kindly reviewing the preliminary version of this report.

This page intentionally left blank.

1 Introduction

During the past five years, Free and Open Source Software (FOSS) has become not only omnipresent on the World Wide Web (WWW) but also, for hundreds of cases, mature enough to be considered in production environments [1]. Indeed, it is now being adopted worldwide in many governments, agencies, and companies as replacements for commercial-off-the-shelf (COTS) software that, of course, has a higher acquisitional cost.

However, these institutions are currently facing an important issue: how can they be assured that the FOSS they are selecting and then using everyday is mature enough and secure? In other words, how can FOSS maturity, quality, and security be assessed? Of course, with FOSS, it is theoretically possible to manually review the entire source code of the application since it is readily available. This is usually called *peer reviewing*. Source code can also be thoroughly tested in order to find bugs or security holes. In practice, however, manually scanning through tens of thousands lines of code is costly and counter-productive, if not impossible. Moreover, thorough testing is not always possible for project managers. Tools that can automate and support this task are therefore needed.

Since Java is now widely accepted as a programming language, many FOSS applications are written in it. Therefore, tools for automatically assessing Java code are of great interest. This document begins with Section 2 giving a few clarifications about improvements that have been implemented in Java compared to C++. Section 3 defines the difference between *safety* and *security* and includes a brief characterization of the relationship between defects that are detected by the tools listed in this document and potential safety and security vulnerabilities. Sections 4 and 5 categorize and describe many Java verification tools currently available. This list is not meant to be exhaustive; a selection has been made in order to only keep the best ones. More interestingly, Section 6 presents the results of a practical evaluation that has been performed for some of these tools. Finally, Section 7 gives a simple, comparative methodology for project managers who want to quickly assess Java FOSS maturity and quality.

2 Java Is Not C++

Because the syntax of the Java programming language has some similarity to that of C++, many C++ programmers believe that Java is just a slow version of C++. They cannot be more wrong. First of all, through years of constant development since 1995, and mostly since Java 1.4, the performance of the Java virtual machine has dramatically improved [2]. But more importantly, the semantics and the philosophy

of these two languages are very different. Table 1, based on [3], illustrates this statement.

	Java	**C++**
Pre-processor	No	Yes
Clearly-defined primitive types	Yes	No
Null-terminated strings	No	Yes
Pointer arithmetic	No	Yes
Possible uninitialized variables utilization	No	Yes
Automatic array bounds checking	Yes	No
Inheritance	Single	Multiple
Operator overloading	No	Yes
Native multi-threading	Yes	No
Generics	Yes (since Java 1.5)	Yes (templates)
Garbage collection	Yes	No
Integrated static verifier	Yes	No
Integrated dynamic monitor	Yes	No

Table 1: *Java Improvements Over C++*

For the purpose of this document, the two most interesting features of the Java language are the fact that it does not allow pointer arithmetic and that it provides built-in garbage collection. Indeed, the former ensures that Java is type-safe, that is, every value or datum manipulated by a Java program is handled by the proper variables, operators, and primitives. For example, an integer cannot be transformed into a pointer and used to reference any given address in memory. Such practices can be used in C++ to exploit buffer overflows, for example. The second interesting feature, called garbage collection, ensures that memory is correctly managed by Java programs, without the explicit intervention of the programmer, who has to write error-prone memory management code in C++.

In order to keep this document free of superfluous technicalities, the bytecode verifier and the security manager (monitor), two advantageous components of the Java virtual machine, are not detailed, even though they contribute quite efficiently to Java programs' safety and security. However, it can be mentioned that Java has better-defined primitive types, whose sizes are specified by the language and not by the implementation, in which *boolean* is not actually an *integer*, *char* is Unicode and not simply ISO-8859-1 (facilitating internationalization), and in which the *null* reference is defined in the language instead of being the integer 0. Moreover, no uninitialized variables can be used in a Java program because the compiler detects such dangerous practices. Finally, array bounds checking is automatically performed in Java and strings are not null-terminated, which eliminates the risk of buffer overflows [4].

3 Safety vs Security and the Relationship with Detected Defects

Before briefly characterizing the relationship between defects detected by Java validation and verification tools and potential vulnerabilities, it is mandatory to define what is intended by *safety* defects and *security* defects.

Safety defects are low-level defects by nature. Indeed, they either affect the control flow, the data flow, or the memory space of the problematic program. For instance, an illegal jump instruction branching outside a method's body or into a data fragment of a program constitutes a control flow safety defect. An example of a memory safety defect would be an instruction that dereferences a reference that is not pointing at an allocated object. Therefore, safety defects are usually easier to automatically detect and correct and they must be corrected in order to ensure security because this type of defects can be used as vulnerabilities to circumvent established security protection.

Security defects are higher-level than safety defects. They are therefore much more difficult to detect automatically since they consist in defects perturbing the application's user authentication, data integrity, data confidentiality, availability, transaction non-repudiation, etc. For instance, a bad encryption algorithm can alienate data confidentiality. In practice, however, safety defects can lead to security vulnerabilities. For example, in C or C++, buffer overflows can cause vulnerabilities that can be exploited to get root privileges on a computer and break security. In Java, safety has been taken very seriously and therefore, buffer overflows, for instance, are impossible because arrays and strings are represented by classes that perform array bounds checking. There are still a few pitfalls in the language, however, so Java validation and verification tools are useful despite the platform's ability to avoid almost all common problems found in C and C++ code.

Since security defects can hardly be detected automatically with the help of tools, the Java validation and verification tools that are listed in this document mostly detect safety defects. Some of these safety defects, however, can lead to potential vulnerabilities, conducting most of the time to denial of service attacks. For example, the presence of a *return* instruction in a *finally* block could hide a thrown exception and lead to incorrect behavior that makes the program crash. Fortunately, these types of defects are usually hard to exploit and have moderate consequences in most scenarios.

4 List of Provided Capabilities

The tools described in this document are listed in Table 2 on page 51 and sorted by capabilities, which are explained in the following subsections. The capabilities are presented in order of decreasing importance, starting from the most important.

4.1 Supporting Tests

This is the most important capability because it represents the most interesting tool feature among the ones investigated. Indeed, tools in this category are capable of:

- supporting Java project test cycles in many ways by automatically generating unit test cases,
- automatically running the code on these test cases, producing detailed thrown exception and error reports,
- scanning for different code blocks that are similar to encourage code reuse and maintainability and to drive debugging,
- statically analyzing Java code to detect hard-to-find errors such as thread synchronization problems,
- and model-checking Java code in order to prove that it respects user-defined properties.

Therefore, these tools are the first ones Java developers should know about and have in their toolbox. Project managers should at least be able to recognize their names and understand their capabilities.

4.2 Detecting Pitfalls

Tools in this category are capable of detecting, and sometimes automatically correcting, Java coding pitfalls. These pitfalls are not necessarily errors but have a strong potential of being so. Such pitfalls are sometimes called *bug patterns*. For example, these tools can detect that the case statements in a switch expression do not end with a break instruction. This could be correct in some circumstances but it is usually an omission. Another example is missing default constructors, which could lead to improper object initialization. These tools can be useful to both programmers and project managers.

4.3 Enforcing Coding Standards

This capability contains tools that can analyze Java source code, check if it conforms to some given rules, and report on this conformity. The rules usually describe known coding standards or conventions, like the ones defined by Sun Microsystems Inc. They can also enforce coding style conventions in order to make sure that the look and feel of the code produced by many different developers is the same. Once again, these tools are useful for everyone involved in projects, from managers to programmers. Indeed, with these tools, managers know that the code will respect some standards and developers can better understand the code written by others.

4.4 Calculating Metrics

The tools in this category simply calculate many different metrics associated with Java source code. For instance, there are the total number of code lines, the average number of lines per class, the average number of lines per method, the cyclomatic complexity of each method, etc. They can also highlight values that are above or below certain thresholds. These tools are better suited for managers, because they can use them to get a global, overall picture of the structure and quality of the code developed in their projects.

4.5 Decompiling

Tools in this last, and least important, capability are decompilers. They can be useful to generate, from the bytecode, the source code of a project where it is not available and then use other tools in order to better understand it or perform more in-depth tests. These tools are presented in this document for general information only and will not be covered in details.

4.6 Miscellaneous

This category represents tools that have an extra utility that cannot be classified in another category due to their originality. See the tool descriptions in Section 5 for more details.

5 Tool Descriptions

This list of tools has been compiled by Richard Carbone.

5.1 AppPerfect DevSuite – Code Analyzer

http://www.appperfect.com/products/devsuite/index.html

Status: The program is still under active development and is currently at version 5.1. While it cannot be determined exactly when AppPerfect Code Analyzer was started, the company AppPerfect Corporation was founded in 2002.

Description: AppPerfect Code Analyzer performs two vital functions for the developer: it is both a source code analyzer and a rule enforcer. You can use AppPerfect Code Analyzer to scan through your Java or Java Server Page (JSP) source code to find a variety of both easy- and hard-to-find errors, including logic errors. You can also use AppPerfect Code Analyzer to go through your code and find where your coding practices are inconsistent or violate its built-in coding rules. The program is extensible and allows you to add your own rules. Both the Standard and Professional versions can easily integrate with either Eclipse or NetBeans. It has an easy to use GUI and works under both Win32 and Linux.

Buzz: AppPerfect Code Analyzer is available only when you download AppPerfect DevSuite. The professional version of AppPerfect DevSuite is commercial, while the Standard Edition is free (but limited) and can be used without charge. AppPerfect Code Analyzer is an ideal tool for reviewing peer source code, ideal in large development environments because it is powerful and yet simple enough to use for even the Java programming novice. The program comes bundled with over 150 built-in rules that apply Java best practices knowledge available from various expert sources. This tool is definitely well worth the small price tag associated with it. AppPerfect DevSuite was designed to work with your web servers and Java servers such as IBM WebSphere, Apache Tomcat, JBoss, and BEA WebLogic. It supports Java 1.3 and 1.4.x. Unlike the Standard Edition, the Professional version can integrate with JBuilder, IntelliJ, JDeveloper, and IBM WAS.

License: Commercial and proprietary, however, the Standard Edition is available for unlimited free use. The Professional version is $499 US.

5.2 Jtest

http://www.parasoft.com/jsp/products/home.jsp?product=Jtest&itemId=13

Status: It cannot be determined when the program was made available commercially or when it first appeared in the marketplace. However, Parasoft was

founded in 1997. It also cannot be determined who the developers and maintainers of the program are at Parasoft. Jtest is currently active and at version 5.1.

Description: Jtest is a commercial static analysis tool for Java. Using a pre-built database, it can fully analyze your Java source code to find common error-prone coding practices. It can find more than just security holes in your software. It is able to parse through your source code and will then alert you for all of the problems that it finds. It is able to determine if it is missing classes or library files, or alert you to better ways of doing things. It is also a fully functional static analysis tool capable of automating Java unit testing. It supports many different types of checking and can handle any sized project. The average developer will also appreciate its ability to detect conformity errors to ensure that the source code conforms to well-established coding standards.

Buzz: It has an intuitive user interface and a good help file. One of its major disadvantages is the cost, which is currently pegged at about $3,500 US. Jtest can test for all kinds of different bugs and potential problems or conflicts with your source code by using a database comprising about 400 rules. Each ruleset can be turned on or off via the configuration windows. Probably the most interesting features to those performing vulnerability assessments are Security, Potential Bugs, Threads & Synchronization, Unused Code, Initialization, Possible Bugs, and Garbage Collection. However, it offers a lot more features than just this. Jtest also has the ability to integrate into a team development environment via CVS and sports support for IBM's Java IDE Eclipse. It is overall a very impressive program. However, this program is a static analysis tool only. It is not capable of dynamically watching your Java programs play out. This is unfortunate as it is the only feature missing to make it a fully well rounded program. Nonetheless, it deserves a rating of three stars[1].

License: Commercial and proprietary license. Certain portions of Jtest are open source code and portions are used under the Apache license, GNU GPL license, and CPL license.

5.3 ESC/Java 2

http://www.cs.kun.nl/sos/research/escjava/index.html
http://research.compaq.com/SRC/esc/

Status: Originally developed under the flagship computer company Compaq, the project finds itself still alive today, under the stewardship of HP. The project

[1] Remember that a one-star rating means that the tool's capabilities were judged insufficient for proper application assessment, two-star means acceptable and three-star, excellent.

started in 1999 and continues to this day. However, the original developers of the project are unknown, as are the contributors. The concept was first put forward in a paper at Compaq by David L. Detlefs, K. Rustan, M. Leino, Greg Nelson, and James B. Saxe that is titled "Extended Static Checking" SRC Research Paper 159, that outlines the concepts of ESC/Java. It is currently at version 2.0a7.

Description: ESC/Java 2 is the natural continuation of the original project founded and funded by Compaq. It is a highly advanced static analysis tool for your Java source code that comes bundled in a precompiled package that can run on multiple platforms. ESC/Java 2 is a highly evolved Java static analysis that is able to verify your code against the Java Modeling Language (JML) specification. It helps you to find many common run-time errors in your code prior to execution. Although it will not find all errors, it is powerful and robust.

Buzz: Because many tools do not use the JML specification, they cannot necessarily predict the behaviour of Java programs. The JML specification is an initiative to specify the behaviour and detailed design of Java programs by adding specially designed annotations to the source code. The use of JML thus makes debugging and static analysis easier and does not necessarily rely on a database like Jtest and other programs. However, ESC/Java 2's use of JML is limited. Currently, the ESC/Java 2 project has made many efforts to be compatible with JML; however, it finds its errors through a more complex system. ESC/Java 2 works in two different ways. The first is a stand-alone analysis tool, which verifies your source code. The second adds a superset of features to this, which, through a self-documenting "pragma" that you must add to different parts of the code during the static analysis, enables you to have the static analyzer stop stating the same error. For example, if a variable is supposed to be set to null, the static analyzer will report this as an error. However, using the "pragma", the program no longer states this error and moves on to the next problem. This is a very powerful option to use when debugging large projects. ESC/Java 2 includes an annotation language with which programmers can express design decisions using lightweight specifications. Moreover, ESC/Java 2 is able to verify the integrity of the code using a theorem prover through a highly complex and convoluted process. It uses the Simplify theorem prover that has its own language for proving theorems. Using a mathematical construct, ESC/Java 2 is able to verify if the code is of high integrity or not. This is excellent because it does not rely on databases and reduces the number of false positives.

License: Open source; however the licensing scheme is unknown.

5.4 Jlint

http://www.artho.com/jlint/
http://jlint.sourceforge.net/

Status: This project is still under active development. It was originally developed by Konstantin Knizhnik. It was then added upon by Cyrille Artho who added more options and extensions to the project. While it cannot be determined exactly when the project was originally started, the last version written by Knizhnik was in 1998. The current version by Artho is at version 3.0, which he picked up after version 1.11 from Knizhnik. However, Artho no longer maintains it. It is now a SourceForge project and is maintained by Raphael Ackermann, David Hovemayer, and Strider.

Description: Jlint is a Java static analysis program that checks through your source code and find bugs, inconsistencies, and synchronization errors (through the use of a data-flow analysis mechanism and a building lock graph mechanism). Jlint is composed of two parts: the first is AntiC, that is a syntax verifier that applies to C, C++, Objective C, and Java. The second part is the Jlint checker itself that extracts information from Java class files.

Buzz: The Jlint checker performs its work directly on Java class files because it protects the checker from Java source code extensions. AntiC works best on raw source code with no syntax errors. Although it can detect some syntax errors, it might crash on certain errors. It can detect bugs in tokens, operator priorities, and body statement analysis such as incorrect or faulty assumptions made on loop structures, if-then structures, else branching, or suspicious use of switch. Jlint can find errors with synchronization, inheritance, and data-flow including null variable assignment. However, through the use of global data-flow analysis, Jlint can approximate the value of variables and statements, and detect null assignments to variables. Through its lock dependency graph, Jlint can detect possible race conditions that occur when threads concurrently access the same variables. It can also detect deadlocks during multi-threaded program execution and help you to catch synchronization errors that the Java compiler cannot warn you about. Jlint has an option to enable it to save a history about an analysis and redisplay it at a later time. Consider using AntiC first to check for syntactical errors and then use Jlint to verify your compiled Java code. In addition, as a last note, by using debugging information, Jlint can associate error messages with appropriate source code files.

License: GNU GPL.

5.5 Wasp

http://www.waspsoft.com/

Status: Wasp is a commercial program that was developed at the Institute of Informatics Systems in Russia. Ports of the program were developed under the grant "Program Understanding". WaspSoft, the company which developed the program is owned by AcademSoft, another Russian company. Development appears to have started in 2000. The current release, the light version for Windows is at version 3.0. Version 3.2 is available for Linux.

Description: Wasp is an analysis tool that statically detects many different runtime errors. It is able to do this through the use of its own powerful data-flow analysis. It can statically analyze Java source code and it can also read and understand project files and then parse through the defined Java files from within the project file. It is a command line tool that runs under Windows and Linux. It is simple to use and easy to understand and its error messages are intuitive for an intermediate programmer.

Buzz: While the tool should have been made more platform-independent, at least it does work on one flavour of the UNIX operating system (Linux). Wasp is able to list the bugs and potential bugs it finds in three major categories: absolute errors, conditional errors, and warnings. It is able to present useful information on variables and their use throughout the source code. Because Wasp performs data-flow analysis, it can however require considerable amounts of memory. This is one reason why it would have been better to port Wasp to other platforms where huge multi-memory architectures are more readily available. Do not be surprised if, for a medium sized project, between 100 to 200 MB of physical memory are required. It will stop analyzing code when no more physical memory is available. Wasp comes complete with a garbage collector and saves both completed and partial analyses to files. However, because data-flow analysis can generate huge amounts of information on large programs, using the method call graph option may help you to understand where difficulties occurred in the analysis. The analysis phase consists of three major parts: the first is the translation of the source code by the Java front-end included with Wasp; the second is the data-flow analysis, and the third is the error analysis. It finds essentially the same types of errors as other static analysis tools.

License: All versions are commercial and proprietary. Light versions can be used on a continual basis but have a limitation of analyzing up to 2000 lines of code.

5.6 Bandera

http://bandera.projects.cis.ksu.edu/

Status: This project is being headed by the Laboratory for Specification, Analysis, and Transformation Software (SAnToS Laboratory) in the CIS department at Kansas State University. There are a variety of individuals who both head and develop the project. However, it cannot be determined who is the founding member of the project. The faculty members of the project are James Corbett, Matt Dwyer, and John Hatcliff. The project dates back to at least 1999 because we can determine that former students worked on the project in 1999. The software is currently at version 1.0 but for this evaluation, we used version 0.3.

Description: Bandera should be considered a toolset for extracting models from Java source code and then performing model checking on the extracted models. It builds a finite-state model for automated verification using a technique called program slicing. In short, Bandera is a tool to be used to check the correctness of the properties of a Java program.

Buzz: Finite-state modeling (FSM) can only approximate the behaviour of a given piece of code. FSM is extremely complicated, and most of the time, the development of the Bandera project has been involved in how to perform FSM and then how to apply formal methods to verify the extracted model. Creating the initial model is done through a convoluted process of static analysis, data-flow analysis, and program slicing. FSM is a process often used to understand the behaviour of multi-threaded applications, although it applies equally well to single-threaded applications. A static analysis of the source code is able to refine the dependencies of the class functions and locations that are required to be known by the program slicer. There are limitations to how well FSM can correctly model a program. Against the small tests we performed, we were overall pleased with the results obtained; however, the programs were not more than a few hundred lines of code at the most. How well the Bandera model checker stands up to modeling and verifying the correctness of a program with millions or even hundreds of thousands of lines is questionable. Bandera uses the Soot compiler framework. This framework is used to convert Java source code into an internal representation known as Jimple. Jimple is a three-address code version of bytecode that includes all of the necessary concurrency operations found in multi-threaded applications. The Jimple is then converted back to a Java-like language suitable for internal compilation that is used to produce the FSM model. Bandera also uses its own internal language, the Bandera Specification Language, or BSL, that is a source-level specification language for model checking Java programs. Bandera is an ex-

tremely complex program to work with and requires an advanced knowledge of Java, programming language design, and modeling. This is not a program to be given to someone not knowledgeable in these fields. The learning curve is sharp and somewhat painful. Bandera sports no good useful documentation for someone who lacks the concepts that should be put into simple words. While this program should have been designed for use by a wide-range of IT personnel, instead, it finds itself as a complex, convoluted, and badly explained program. While it is a model checker and works well overall, it is not a tool that will find itself used extensively by developers around the globe. However, we must also realize that this is a research tool and thus is not especially designed to be user-friendly. It appears as if Bandera can use more than one model checker, but for these tests, we used Spin, which uses Promela as its language. However, we did not try to spend too much time trying to understand its internal workings as this is a subject worthy of a student's Master's thesis.

License: GNU GPL.

5.7 Excelsior FlawDetector

http://www.excelsior-usa.com/fd.html

Status: This program is currently at version 1.01 and is developed by Excelsior, LLC. The headquarters for the organization is in Novosibirsk, Russia, with an office in Germany. The company appears to have been around since 1999, however, their product Excelsior appears to be a relatively new product, and version 1.0 was released in May 2004. It cannot be determined who actually developed the software.

Description: While many other tools are able to perform static analysis on the Java source code itself, FlawDetector is able to perform static analysis on both bytecode and JAR files, including the ability to statically analyze third-party libraries. This is a powerful feature in the case where source code is no longer available or retrievable. It commonly scans for null pointer exceptions, arithmetic exceptions, out of bounds and array problems, class cast exceptions, and negative array sizes. However, it also finds many common Java errors such as method override conflicts, references to absent fields and methods, incompatible classes, as well as class verification and instantiation errors. It has built-in plugins for the Eclipse and JBuilder IDE's. However, the post-analysis evaluation of the error is up to the programmer to understand since the error reports are not very explicit.

Buzz: It works only on Win32, but at least it does support both Eclipse and JBuilder. It requires Sun JRE 1.3 or higher in order to work correctly. FlawDetector uses an innovative technique that determines the strength of the analysis to be done based on the actual system configuration of the host system. Therefore, if you have a very high-end Win32 system with gigabytes of RAM, you can safely increase the strength of the analysis without a severe penalty in performance. However, realistically, this feature would have been better supported on large multi-processor RISC-based platforms that commonly have more than 16 processors and many gigabytes of RAM. Because the program works only on Win32 platforms, you will be left with the limitations of the PC architecture. The program can also be run from the command line. It also sports a batch directive allowing it to process many files one after the other in batch form. While it is a powerful product, expect that it will be very slow, even for small analyses. This is why we stand by our claim that the applications should also be ported to the UNIX platform. While the application can decode bytecode, it works best when the source code is also available for the interpretation of the errors.

License: Commercial and proprietary. However, the software can be downloaded for evaluation and relies on the end-user to respect the 30-day trial period. There is no license manager for this application.

5.8 InsectJ

http://insectj.sourceforge.net/index.html

Status: This project is still under active development. This is the open source, generic version of a similar program written by the two authors when they were at the Georgia Institute of Technology. The authors are Alex Orso and Anil Chawla. While it cannot be determined when the authors decided to write a generic open source version of the program, it has been registered with SourceForge since May 2003. The program was originally written by Anil Chawla under the supervision of Alex Orso and Mary Jean Harrold at Georgia Tech under the Aristotle Software Engineering Research Group.

Description: The tool was originally written in the hopes of developing a tool that could assess the adequacy of the testing procedures to be used to verify the code of your project. The project aims to accomplish broad goals, the first being a tool that collects in-depth coverage information about a Java program. The second is to provide the necessary optimization techniques to reduce the tasks performed by the system. Supporting Java's object-oriented nature requires gathering coverage information such as thrown and caught exceptions,

and identifying a mechanism for dynamic object invocation. Thus, through the aid of a dynamic and static analyzer the tool is able to provide extensive coverage of a Java program.

Buzz: In order to accomplish all of this, the program uses the Java program bytecode. It therefore makes use of the BCEL library in order to manipulate the bytecode. The tool then inserts a series of probes back into the bytecode. At runtime, these probes then report back the information collected. This is done when the manipulated bytecode is run with a set of monitor classes which gather and record the information collected by the probes. This information is then mapped into a flow-control graph. Using a technique known as virtual-call resolution it can be possible in some instances to reduce the number of probes that must be inserted into the code by upwards of 90%. Therefore, the program can help developers and debuggers find problems in their code which can only become apparent at runtime. However, this tool should be considered as alpha code and is not very stable. While it will run under Linux and UNIX in general, it is possible to run it under Windows if the source code is recompiled.

License: GNU GPL.

5.9 SimScan

http://www.blue-edge.bg/download.html

Status: It is not possible to determine when this project started. It does still appear to be under active development, and the offered version for download is currently at version 1.0. The project was developed at Blue Edge Bulgaria, a consulting company in Bulgaria.

Description: SimScan is an in-house product of the company which was written for finding similar or duplicated code fragments within or across Java development projects.

Buzz: Each version comes with a GUI plugin for IDEA, JBuilder, and Eclipse, as well as a command line interface. It is a very effective product. It should be considered for any project where too much code generation or lack of code reuse is an issue. Even if you think your developers are efficiently using code reuse, you should still try out this product to confirm your assessments. Code reuse tends to yield faster and more efficient programs. Creating and using generic reusable classes as opposed to the “copy-paste-modify” paradigm will result in fewer opportunities for “misused class reuse”. This program requires

large amounts of physical memory. It is a stable program and should be considered ready for production environments.

License: Commercial and proprietary.

5.10 Simian Similarity Analyzer

http://www.redhillconsulting.com.au/products/simian/index.html

Status: Developed at RedHill Consulting Pty. Ltd, it is still under active development. RedHill is an Australian-based company. The software was developed by the company's founder and lead developer, Simon Harris.

Description: Simian Similarity Analyzer is a tool that finds and identifies duplicate code in many various programming languages, which include Java, C, C++, COBOL, Ruby, JSP, ASP, HTML, XML, and Visual Basic. The distribution includes a JAR file and Win32 executable so you can be up and running in minutes on the platform of your choice. It can intregate with IntelliJ, Eclipse, Maven, and has a CheckStyle plugin. It also sports a command line interface. Its output is very simple to understand and can be used by novices and experts alike.

Buzz: Simian runs natively under Microsoft .NET. It can also run under any JVM 1.4 or higher. You can use the tool during the development and build phase of your project, or as a re-factoring guide. This tool can help save much time, effort, energy, and money in debugging your code. It may give you a false positive from time to time, but that is rare. Newer versions will include support for ADA and Python, as well as support for multi-threaded analysis that will yield large gains in performance. Simian should work on any Java-enabled system.

License: Free for non-commercial and open source use. Fee based for commercial use. Evaluation for 15 days for those considering commercial use.

5.11 JVerify

http://www.mmsindia.com/JVerify.html

Status: Developed by the founder of Man Machine Systems, K. Rangaraajan, this product started development in 1997 and it appears to have stopped development in 2002. Man Machine Systems is a company in India. JVerify is currently at version 1.1.

Description: JVerify is a tool to help you in performing invasive testing of your Java classes. JVerify is both a set of executable classes and a collection of API's. Thus, you can use it out of the box, or you can create your own programs that incorporate its features. JVerify uses a scripting language called JMScript, a language similar to the Java programming language, which makes it possible to test a given class for conditions and assertions not normally available in other debugging tools. Invasive testing allows for a more systematic method of control and observation during the testing of a class. For instance, JMScript allows the tester/developer to write specialized scripts that can directly manipulate (read/write) private fields in Java classes. JMScript instantiates Java objects and sends messages to them in a very Java-like manner. JVerify is a tool that is based on a FSM (finite state model/machine) and as such, has certain unique advantages over other testing tools. The main advantage is that a FSM-enabled tool can test for more potential state conditions than other dynamic tools. However, its main disadvantage is that all too often, there are so many potential conditions that it would take an enormous amount of time to resolve all of them. Thus, JVerify is uniquely equipped to help reduce the number of conditions to a manageable amount.

Buzz: It is generally considered more difficult to detect bugs in object-oriented programs rather than non-object oriented programs simply because of the abstraction of the information layer inherent in object-oriented programs. Thus, the author of the tool proposes an API/class toolset that can penetrate through the information hiding barriers and write a driver that allows normally inaccessible parts of the program/class to become accessible, and thus testable. The author of the tool claims that using invasive testing offers bug detection capabilities not available with other methods. Thus, even private elements of a class become accessible. However, the invasive testing model uses an approach that tests almost all machine states of the host system. This very often can lead to an almost infinite number of potential states. However, the ability to access private elements can significantly reduce the processing time. Unfortunately, the learning curve for this type of debugging method is rather steep. It requires learning a new programming language, JMScript, which in itself is complex. Assertions made can only be checked if they came directly from JMScript. While this is an interesting method, it is useful only in testing some of your classes. This method is not yet ready to be used in cases where you would want to test large projects or a large number of classes in a short period of time. Nonetheless, it is a promising development in the testing of Java programs. Unfortunately, however, it is only supported under Win32. Finally, its reporting mechanism needs some substantial work.

License: Commercial and proprietary. A 15-day evaluation period is available for anyone to try it out.

5.12 Hyades

http://www.eclipse.org/hyades

Status: The project is still under active development and it is currently at stable version 3.0.2 and alpha version 3.3.0. The Hyades project is being actively developed by the Eclipse project. While it cannot be determined exactly when the Hyades project was started, the Eclipse project was started in December 2002 when IBM, Rational, and several other companies got together to form the Eclipse project. The Hyades project was soon born out of the Eclipse project with the need for an automated software-testing environment.

Description: The Hyades project attempts to provide an open source solution for your automated quality assurance needs. You can use Hyades for automated testing, tracing, and monitoring of your software projects and systems. By integrating Hyades into the Eclipse project, it becomes possible to have a tight level of integration between your development processes and your software testing to provide a higher level of efficiency in your software development lifecycle. As time goes by, the software is continuously being worked on and you can expect more functionality and more stability. Hyades offers an extensible framework for your software testing needs. Other features that set it apart from the competition are a unified model and user workflow, as well as a unified set of API's.

Buzz: You will find that Hyades actually requires quite a few things before you can get it to run, namely Eclipse 3.0.1 or greater, Java 1.4 or higher, the Eclipse Modelling Framework SDK 2.0.1, and finally, the XML Schema Infoset Model (XSD) SDK 2.0.1. Luckily, these products are not difficult to get or to install on your platform. Especially if you will be working in Linux, Solaris, or Win32, you will not find it too difficult to setup. Hyades, unfortunately, comes with several add-on modules in order to give it all the functionality you may need. There is the runtime, the testing component, the data collection engine (compiled for your platform), logging implementation, and a translation log plugin. Once Hyades is installed, you will find that it is a powerful tool that can be used for most of the work you will ever need to do in a software quality control environment. Hyades is a very stable Java software-testing suite. However, Hyades can only work alongside Eclipse and thus cannot be used in a different development environment.

License: Common Public License 1.0.

5.13 Lint4J

http://www.jutils.com/

Status: Currently at version 0.7, the project began in October 2002. It cannot be readily determined who the author(s) of the program is(are). The project is still under active development.

Description: Lint4J is a multi-platform static analysis tool for Java. It helps the developer to track down such issues as locking and threading problems. It can also find performance-based issues in your code, find scalability problems with the code, as well as check for complex technologies such as Java serialization. The tool was designed to test out and find bugs with third-party software coming either from the commercial or open source world. It helps software developers to find difficult-to-detect defects in their source code.

Buzz: The author of the tool has used it to find bugs and defects in all kinds of commercial and commercial-quality open source products such as VisiBroker for Java, WebObjects, OpenEJB, OpenORB, and JBoss. It supports a wide variety of features for bug detection which are too numerous to enumerate here. Of potential interest to developers are its performance tests, Java language constructs testing, detection of debugging and immature code, memory leaks and architectural problems, as well as synchronization and serialization problems. Because the project is still young and appears to be moving along at a healthy pace, we can be optimistic and expect some great things from this project. However, it needs a little more maturity to be used in production environments.

License: Proprietary; however, it can be used without purchase.

5.14 JCover

http://www.mmsindia.com/JCover.html

Status: The program appears to be under development. It is developed and sold by Man Machine Systems, a company in India which was founded in 1997. It cannot be ascertained when the program first became available. JCover is currently at version 2.1.

Description: JCover, or Java Code Coverage Analyzer, is an advanced Java source code analysis tool. What sets it apart from the other tools is that rather than searching specifically for programming and logic errors, as well as bad coding practices, it focuses on finding coverage issues. JCover allows you to gather

test coverage measures of both Java source code and compiled classes. Using JCover's data analysis feature you can see how well the tool is able to pick up errors over multiple runs, and see how its coverage improves with each run. Charts can be exported to HTML, XML, and CSV. It also sports a rich API so you can hook your own programs directly into JCover. It also features an intuitive and easy-to-use interface.

Buzz: It works only under Win32 and this is unfortunate as much of the Java world is running and developing on other platforms. It works with Java 1.2, 1.3, and 1.4. A very useful feature is that it supports batch operations for large amounts of source analysis. It sports several different kinds of reporting charts with simple and easy to understand statistics about your source code and the analyses performed. Coverage analysis allows the Java developer to focus on problematic areas of the code. After several runs, it can help the developer zero in on those hard-to-spot areas of code which require more in-depth analysis and testing. The popular coverage techniques discussed in the open literature are: Statement, Condition, Decision, Multiple conditions, Loops, Statement and Branch, as well as Call-Pair, Path, and Data-Flow. However, JCover only supports Statement, Branch, Method, and Class coverage. JCover also sports some interesting statistical and metric evaluations. It also supports coverage of EJB (Enterprise Java Beans). You can select portions of code to exclude from analysis and it can even compare data from multiple test runs.

License: Commercial and proprietary. It is available for $495 US.

5.15 JSynTest

http://www.mmsindia.com/JSynTest.html

Status: The program appears to be under development. It is developed and sold by Man Machine Systems, a company in India which was founded in 1997. It cannot be ascertained when the program first became available. JSynTest is currently at version 1.0.

Description: JSynTest is a syntax-testing tool. Syntax-testing, which is also commonly referred to as grammar-based testing, is a different kind of source code testing. In short, a syntax-testing tool is able to automatically generate test data from a formal specification. While this is not a tool to be used lightly by the novice or even the intermediate developer, it can help you in testing out source code through its auto-generation of test data which is ideal for testing out new programming languages, compilers, obfuscators, scripting and shell languages, as well as menu-driven software.

Buzz: Its interface is simple to use. Moreover, unlike many open source programs that claim to be able to support the functionality of syntax/grammar-testing, JSynTest does deliver on its promise. However, be realistic and do not expect the tool to help you find a variety of errors in your source code. Rather, expect it to be able to verify if your source code can properly handle its input data. It is multi-platform and runs on Win32, Solaris, and Linux. Completely written in Java, it will possibly work on other platforms as well. It requires JDK 1.2.2 and up. The auto-generation of test data is performed by an embedded Java program that implements the generation logic and as such, this module can be embedded into your applications in order to implement test data generation directly within your own programs.

License: Commercial and proprietary. It is available for $395 US.

5.16 JUnit

http://sourceforge.net/projects/junit/
http://junit.org/

Status: This project no longer appears to be under active development since February 2004. The last release was made available in September 2002. It was being developed by Mike Clark, Erich Gamma, Erik Meade, Kent Beck, and Vladimir R. Bossicard. The project was registered with SourceForge in 2000. It is currently at version 3.8.1.

Description: JUnit is an open source unit testing framework for Java which you can use to write and run tests on your Java source code. A nice feature is that the tests can often be repeated from one source code to another. In short, JUnit is a tool which allows you to write a testing framework designed around your Java programs that you want to test.

Buzz: JUnit was written because, generally, most programs lack automated testing features. Once developers are finished writing their code, they must still prove that it works given a series of test data. Using a debugger is a great idea when you want to observe certain variables, their value, and their interaction in the system. However, a debugger is not the way to go when you have many test cases to test out. Therefore, a framework is necessary. This is what JUnit attempts to accomplish. Because most developers are too busy writing code with respect to deadlines, most code never gets the opportunity to be rigorously tested. JUnit can help to make this possible. Using a composite feature, it makes it possible to run many tests on the same class at the same time. It is a good program with lots of potential, but this is not a tool for novice or

intermediate programmers because in order to properly extract the maximum benefit from the program, you should have an advanced knowledge of Java so that you can write the appropriate tests for your code.

License: IBM Common Public License 1.0.

5.17 JStyle

http://www.mmsindia.com/jstyle.html

Status: Developed by the founder of Man Machine Systems, the author, K. Rangaraajan, started development in 1997 and it appears to have stopped development in 2003. Man Machine Systems is a company in India. JStyle is currently at version 5.0.

Description: JStyle is a powerful static analysis tool that can both perform metrics calculations on your source code as well as test it for important and common errors in Java source code. It is a sister tool to JVerify and can be used in conjunction with JVerify, or standalone.

Buzz: It can handle very large projects and is fast. It supports more than 100 coding guidelines and lets you add your own rules using JMScript. JStyle searches for many different types of syntactical/lexical errors such as class naming conventions, class member naming conventions, class section ordering, bad code reusability, unsafe exception handling, design deficiencies, and many more. It can also search and give metrics information for LCOM, Weighted Methods Complexity, Halstead measures, Cyclomatic Number, RFC, Class Fan-in/Fan-Out, Depth of Inheritance, and many more. It is also possible to incorporate VBScript to write your own scripts. It has powerful and flexible reporting and commenting options. It offers many different types of charting options for the metrics information. It can integrate into JBuilder. JStyle can also run in batch mode. Unlike its sister tool, JVerify, JStyle is ready to be used in production environments. It is simple to use and easy to understand. The only complex portion is writing your own scripts in either JMScript or VBScript.

License: Commercial and proprietary. A 15-day evaluation period is available for anyone to try it out.

5.18 JCSC

http://jcsc.sourceforge.net/

Status: This project is still under active development. The project itself was started in 1999. The project has been registered with SourceForge in 2002. The project is currently at version 0.97. While there are several contributors to the effort, the main developer and author is Ralph Jocham.

Description: JCSC is a static analysis tool that was inspired by Lint. JCSC is highly configurable and flexible. You can add you own rules very easily. It is multi-platform and currently supports more than 100 rules. The rules are stored in an XML format. Thus, if you know XML, you can create and add your own rules. It also supports Javadoc checking and can also report some interesting metrics on your source code.

Buzz: Note that the command line version of JCSC cannot scan folders recursively. For this option, you must use the JCSC Ant task. To function, JCSC requires J2SE 1.3 or higher. Thus, it will work under UNIX, Linux, Mac OS X, and Win32. JCSC now also has an IntelliJ IDEA plugin. Checks are broken into two major groups, general checks and naming checks. The default checks adhere to the Sun Conventions for the Java Programming Language.

License: GNU GPL.

5.19 PMD

http://pmd.sourceforge.net/
http://www.ociweb.com/jnb/jnbJun2004.html

Status: This project is still under active development and it was sponsored by the project Cougaar (Cognitive Agent Architecture - http://cougaar.org/index.php?referrer=pmd), a project funded by DARPA as a framework for developing distributed agent-based applications. Started in 2002, the project currently boasts nine developers. The maintainers of the project are David Craine, David Dixon-Peugh, and Tom Copeland. It is currently at version 1.9.

Description: PMD is another open source static analysis tool for Java source code that is multi-platform and is very expandable and flexible. It is based on a framework that enables it to better understand the source code so that it can find and diagnose potential problems more efficiently. It currently comes bundled with 80 rulesets that are broken down into 14 categories. These categories can be broken down into four broad groups: 1) useless code; 2) style rules; 3) best practices, and 4) likely defects.

Buzz: While PMD works well from the command line, it is suggested instead to use one of its IDE plugins that are available for Eclipse, Emacs, IDEA,

NetBeans, JBuilder, jEdit, JDeveloper, and Gel, because the command line method is not very user-friendly. In fact, the program's documentation certainly needs working on. However, it is powerful and it allows the user to create their own rulesets for finding errors and bugs. PMD requires that the rules be written in a XML format. The binary format of PMD should work on any system with a Java-compliant JRE.

License: BSD.

5.20 JiveLint

http://www.bysoft.se/sureshot/javalint/

Status: This project is a commercial application that is developed by Sureshot. Sureshot is a division of DATA AB, a privately held Swedish company. Sureshot was founded in 2000 and its program is currently at version 1.22.

Description: JiveLint is a Windows-based command line tool for performing static analysis of your Java source code. It reports on potential bugs and weaknesses and its manual describes how they can be improved. It also verifies that the source code follows a set of coding conventions. The main goals of JiveLint are to help the programmer maintain readability and enforce naming and coding conventions, help produce high-quality code by pointing out to the developer many different dangerous constructs in coding, and finally, provide a manual with in-depth information.

Buzz: While this program is not as advanced as other programs (like Jtest, for instance), it is an easy-to-use program that can be used by the novice and the advanced Java programmer alike. A nice feature it provides is in formatting and indentation, a feature that many other programs overlook but that is just as important as finding security vulnerabilities because readable code is maintainable code. It also helps the programmer to catch many common mistakes that are not always obvious after simply reading over the code for errors. Its manual also provides meaningful information on the errors it finds and how to remedy them.

License: Commercial and proprietary, but it can be used for up to 15 days before you have to buy it.

5.21 AzoJavaChecker

http://www.andiz.de/azosystems/en/index.html

Status: Developed by Arguntum, this commercial program is no longer under active development. Development appears to have started in 1999 and the version 2.0 was released in 2000. The program is currently at version 2.3, but it cannot be determined when this version was developed or released. In addition, it cannot be determined who specifically wrote the software.

Description: AzoJavaChecker is a static analysis tool that compares your Java source code against a set of included rules. It checks for unconventional identifier names, detects code that is rather difficult to understand (based on the rule-set), and calculates some metrics about your source code. It is a very fast and efficient parser. It sports an easy-to-use GUI and save its reports in HTML and its tables in CSV.

Buzz: AzoJavaChecker does not check for syntax correctness nor does it verify that the source code can be compiled. Generally, it does not deal with these kinds of errors and when it detects a syntax error, it will stop processing and give an output message stating that it can go no further. Therefore, do not use this program to check for syntactical soundness in your code. The program does not follow any codified set of rules such as the Sun Convention or other specifications. There are versions for Linux and Solaris available; however, we cannot find them. Its rule database is limited and you are likely to be disappointed by this tool. It does not work with Java 1.4.x. It does appear however to work with version 1.3.x.

License: Commercial and proprietary.

5.22 Aubjex

http://www.alajava.com/aubjex/products.htm

Status: Alajava LLC, formerly known as Aubjex LLC, was founded by Jonathan Colt and Don Gilmore. While it cannot be determined when the company was actually founded or when the first version of Aubjex appeared in the marketplace, it is currently at version 2.0. It is still under active development.

Description: Aubjex is a static analysis tool and parser with partial support for dynamic analysis. The project is written entirely in Java, and it can run on just about any platform. First, Aubjex will parse your source code into a database, and then browse the code using advanced XML-based scripts. Once the data is in the database, XML-scripts will do much of the work. Aubjex is a useful program to examine all of the interconnections in your code. Aubjex also includes a code browser for which you can write your own XML scripts, or use the almost 400 scripts pre-bundled with Aubjex.

Buzz: Aubjex runs on any system with a JVM 1.3 or higher, and because it is written entirely in Java, it is platform-independent. It uses only a small subset of the Java 1.3 JDK. It is one of the few tools available on the market that uses XML-based scripts. XML enables Aubjex to remain platform-independent without using platform-dependent scripts. However, not all of the functionality of Aubjex is considered production-ready. For example, its refactoring editor should be considered a prototype at best. However, it uses an advanced IDE. Its code browser very closely resembles a file system browser that will help you in navigating through your code. Scripts will actually perform the work of static and dynamic analysis, as well as source code obfuscation. The obfuscated code can then be recompiled with a standard Java compiler. The scripts for dynamic analysis are not free and must be purchased separately. However, the static analysis scripts support a plethora of features such as finding dead code, erroneous dependencies, etc. Nevertheless, its demo seems hard to get since we could not download it (file not found).

License: Commercial and proprietary.

5.23 QStudio Pro

http://www.qa-systems.com/products/qstudioforjava/
http://qjpro.sourceforge.net/

Status: This program is still under active development. It was originally available only as a commercial product, however, the developer, QA-Systems, has open sourced its professional version. The Enterprise version is still for purchase only. QA-Systems started working on the QStudio Java Pro version in 1999, and in 2001, the Enterprise version became available. Although it is not directly mentioned when the company open sourced its QStudio Pro edition, it has been registered with SourceForge since February 2004. Therefore, we can assume that it has been open sourced since early 2004. The open source version is currently at version 2.1.

Description: QStudio is a multi-platform static analysis tool for Java source code. It performs a full code review and then tests it to ascertain whether it follows the ISO 9126 Java standard. ISO 9126 is a software quality framework for design and implementation. Using a database of about 200 rules, it is able to scan through the code and determine many hard-to-find errors and potential errors, as well as security vulnerabilities in your code.

Buzz: There are several important modules available from the project site: an Eclipse/WebSphere plugin, a JBuilder plugin, and a JDeveloper plugin. A

nice feature that QStudio supports that others do not is its ID, Line, Pos, Attr, Impact information line that displays all on one useful line the ID number of the error, the line number in the source code, its position in the code, its attribute type, and the impact it may cause. QStudio is a very powerful Java static analysis tool that finds errors and potential bugs similar to those found using Jtest. However, QStudio's bug database is not as large or as refined as that of Jtest. QStudio's emphasis is on quality control and it allows users to add their own rules using PMD (also covered in this document).

License: The Enterprise edition is commercial and proprietary; however, it is offered freely to those of the Java developer community for a one-year trial, but without source code. The Professional edition is GNU GPL and the source code is available via CVS.

5.24 ASSENT

http://www.tcs.com/0_products/assent/assent_rules.htm#java

Status: While this program is not a publicly licensed one, it does borrow from the open source world and is an on-going commercial application. It is developed by Assent, a subsidiary of TATA Consultancy Services, both of which are companies based in India. The program appears to date back to 1999, and it is currently at version 2.6. An evaluation version is available for download.

Description: ASSENT is an enforcement tool that automatically checks for the conformity of a Java program to the standards and guidelines in place and enforces them. It performs strict and comprehensive checking of a program for its conformity to the Java programming standards in order to detect bugs and other potential errors. It performs static analysis by using a flow analysis technique. It supports over 250 coding standards that come from the following specifications: Java, JSP (Java Servlet Pages), and EJB (Enterprise Java Beans). It generates reports in HTML and Excel, and it can create reports such as call trees, control structure reports, and non-conformance reports.

Buzz: It is available under UNIX for Linux, AIX, and HP-UX, and also for Win32. It requires JRE 1.4.2 or higher to run on these platforms. ASSENT ranges in its rule-checking complexity from simple lexical rules to very complex semantic rules. It will start by parsing and linking the source code, and then perform an extensive analysis on the source code by using a data-flow analysis technique. Then it checks the source code for its conformance to the rules it is equipped with, and picks out the non-conformities in the source code. ASSENT is equipped with a wide-ranging variety of rules that apply to many

different portions of the various Java specifications that form the basis of its rule-set. It is possible to create your own rule-sets through an easy-to-use GUI.

License: Free evaluation version available for all supported platforms; however, license is commercial and proprietary. It also makes use of other non-disclosed GNU GPL and GNU LGPL licenses.

5.25 JavaChecker

http://sourceforge.net/projects/javachecker/

Status: This project is no longer under active development since November 2002. Roni Hursti is the author, developer, and maintainer of the project. The first public release was also in November 2002. It appears as if all effort had been concentrated on producing a publicly releasable project. The project advanced two more times in the same month but never went any further. It is currently at version 1.2 and all development has ceased.

Description: JavaChecker is another static analysis tool for Java. Specifically, it searches for certain types of hard-to-spot errors such as missing default constructors, use of incorrect operators for two's power, incorrect variable use or missing variables, unused variables, and incorrect explicit import-clauses.

Buzz: While JavaChecker is nowhere near a powerful static analyzer, it still does offer the ability to find small, hard-to-spot errors or potential problems with your code. It is freely available and runs on any platform where a Java runtime engine is available.

License: GNU GPL.

5.26 FindBugs

http://findbugs.sourceforge.net/

Status: This project is still under active development. While it cannot be determined exactly when the project was started, it is currently at version 0.8.5 and was released in October 2004. However, the first publicly available version appears to be 0.5.0. It was originally developed by Bill Pugh and it is currently being maintained by Bill Pugh and David Hovemeyer, as well as a team of volunteers.

Description: FindBugs is another Java bug-finding program. It works on the bytecode of a Java application and makes use of the Byte Code Engineering Library (BCEL) and Dom4j in order to decompile the bytecode, analyze it, and perform its XML manipulations. The application was designed around the premise that a bug pattern is a code idiom that can often be an error. These kinds of idioms can arise for a variety of reasons, which generally are: difficult to grasp and implement portions of the Java language, misunderstood and incorrectly used API methods, typos, Boolean errors and other such common mistakes, and code modification during maintenance, patching, and upgrading. You can save the results of an analysis in XML format. It comes complete with a plugin for Eclipse. The plugin should work on any platform where Eclipse is supported. FindBugs also uses filters that can be used to include or exclude bug reports for particular classes and methods. The filters are also written in XML.

Buzz: It is downloadable in both source code and compiled format; however, if you wish to build it you will need Ant. FindBugs works under various UNIX flavours and Win32. In future releases, you can expect FindBugs to support generating reports in both HTML and XML. FindBugs is a powerful open source tool. It is powerful and innovative enough in its approach to be recommended as a definite tool to keep in your arsenal. However, note FindBugs is a memory hog and requires a fast CPU. One last comment is that it is a little limited in its flexibility for configuring what to scan for.

License: GNU LGPL.

5.27 JavaWizard

http://csdl.ics.hawaii.edu/Tools/JWiz/JWiz.html

Status: This project is no longer under active development. It was developed at the Department of Information and Computer Sciences at the University of Hawaii. It was written by Jennifer Geis in collaboration with the NSF and Digital Equipment Corp. The last release of this tool was in 1999. It cannot be ascertained when this project started.

Description: JavaWizard is a static analysis tool whose aim is to find run-time errors in Java source code that is syntactically correct. It is an automated source code checker and requires virtually no user interaction. It is a command line and GUI-based tool.

Buzz: It also sports an Emacs interface so that you can test your source code from within Emacs. It should be cross-platform compatible, and has more than 50

different types of Java programming errors in its database. Unfortunately, it does have a high incidence of reporting false positives. More effort will have to be made in adding better heuristic algorithms in order to reduce the rate of false positives. The source code is provided for those wishing to modify it or improve it.

License: Unknown.

5.28 Hammurapi

http://www.hammurapi.org/content/home.html

Status: Currently at version 3.3.0, it has been available since version 2.1.beta-1. While it cannot be determined when the project actually started, it has been registered with SourceForge since June 2004. However, it appears to be several years old, perhaps dating back to 2002 or 2001. There are currently five members developing the project, and they are dunai2004, Jochen Skulj, Johannes, Pavel Vlasov, and zorror8080.

Description: Hammurapi is a Java source code revision tool. It starts by scanning your Java source code files and verifies their adherence to the coding standards. It can produce reports in either HTML or XML. The program can be executed either as a stand-alone program or as an Ant task. It can revise source code from version 1.4 of Java and down. It is a platform-independent analysis tool that can be run under Win32 and various UNIX flavours. It can also be run under the Eclipse IDE. Hammurapi only works on source code repositories. It introduces two interesting features: the Inspector and Violation. Currently, it comes bundled with 114 inspectors. It can perform source code documentation, and provide some different metrics information on your source code. Its reporting facilities are powerful and easy to understand.

Buzz: Hammurapi supports a wide variety of command-line options enabling you to better customize its functionality. Many advanced programmers may actually enjoy configuring Hammurapi via Ant build files. This enables you to configure it in powerful ways for reuse in other projects and other source verification analyses. However, it uses lots of memory, and the more source code there is to analyze, the more it requires for efficient analysis. Nevertheless, you can write Waivers which are ways of telling Hammurapi that some of its error findings are in fact not errors or violations. These are written in XML. You can also modify and write your own filters and language element filters because they all have an XML-based format. You can also write your own annotations. Its reports are complete and it can find a wide variety of errors, as

well as document them and even suggest in most cases potential fixes for the errors. This is a must tool for anyone programming in Java and who requires an excellent and easy-to-use tool. This is a three-star program.

License: GNU GPL.

5.29 JavaPureCheck

http://java.sun.com/products/archive/100percent/4.1.1/index.html

http://java.sun.com/products/archive/100percent/4.1.1/100PercentPureJavaCookbook-4_1_1.pdf

Status: This project is no longer under active development. It was developed and sponsored by Sun Microsystems. The last version appears to have been completed in 2000. The latest version is 4.1.1, which contains no new features versus version 4.1. It cannot be determined when the project actually started or who actually developed it or contributed to it. It is free but not open source.

Description: JavaPureCheck is a static analysis tool made available directly from Sun Microsystems to test and inspect a Java program for portability problems and issues. It was developed as a part of the Pure Java Check Certification Program. It uses a two-stage process to verify your Java source code for issues or potential bugs that might prevent you from porting it readily to other Java-ready platforms.

Buzz: The first stage involves verifying individual classes and checking them on a class-by-class basis for their Java purity. The second stage involves checks for Java impurities that could arise through the interaction of different Java classes. The program uses a data-flow analysis mechanism to detect when platform-specific strings are incorrectly used. Developed with JDK 1.1, it is compatible with all newer versions of JDK. It works on any Java-ready platform and is intuitive and easy-to-use. It can also save your analysis to file. Before you start checking your code, you should first read the "100% Pure Java Cookbook" which is available from http://java.sun.com/products/archive.

License: Sun Microsystems Binary Code License Agreement.

5.30 CheckStyle

http://CheckStyle.sourceforge.net/

Status: This project is still under active development and it is currently at version 3.4. The project appears to have been developed by Oliver Burn. There are currently three other developers working on the project: Lars Kuhne, Oleg Sukhodolsky, and Rick Giles. While it cannot be determined exactly when the project started, it has been registered with SourceForge since June 2001.

Description: CheckStyle is a tool to help developers who write Java programs adhere to a coding standard. CheckStyle automates the process of checking source code and determining how well it adheres to defined settings and parameters. This makes it a well-suited program to use for large projects where all source code should adhere to the same standards. CheckStyle is also capable of using your own personally-written checks.

Buzz: CheckStyle is highly configurable and extensible. It can check for: Javadoc comments and interface definitions, naming conventions headers, imports, size violations, whitespace, modifiers, blocks, coding problems such as array initialization and inline conditionals, class design, duplicate code, and metrics checks that are used to restrict the number of special symbols and characters in your source code. Miscellaneous checks can be used for recursive checking or pattern checking/searching. The final type of check is used for checking J2EE coding problems. When you write your own checks for CheckStyle, it needs Antlr (http://www.antlr.org) to parse them. You can write your own rules by hand or use the graphical SDK to help you out. Checks are encoded using an XML template. It is also possible to write your own listeners and filters, each of which has a role to play in the reporting and handling of errors. In short, the listeners are used to tell the checker how to respond to events and situations, and the filters are used to audit events which the checker then reports through its listeners. In general, CheckStyle is an excellent program to use in order to verify that your source code follows a certain logic and structure. However, the documentation is vague, especially regarding notes for using and writing your own checks, listeners, and filters. There are also many plugins available to connect CheckStyle to your favourite IDE.

License: GNU LGPL.

5.31 JMetric

http://www.it.swin.edu.au/projects/jmetric/products/jmetric/

Status: This project is no longer under active development. It was started in 1998 and it was stopped in 2000. It was maintained by Andrew Cain but was originally also developed by Rajesh Vasa. It is not possible to determine the current

version of JMetric. The project also had Adon and Maksim Lin as contributors. The last modification to the project appears to be May 2000. It is open source.

Description: JMetric is a Java source code metrics analysis tool. Once a Java source code file has been opened, it then performs a quick broad lexical analysis of the source code in order to better break it down into its constituent parts. In short, JMetric performs a statistical analysis of your source code and gives you meaningful information that can be broken down into two major categories: size metrics and quality metrics.

Buzz: This is a unique tool for analysing Java source code. JMetric performs automated metric collection according to project, package, class, method, and variable. It offers charting and tables with limited range. It sports a project tree view that is very handy in order to get a better grasp on the structure of the project. It also offers the ability to print its reports and export them to text files. Some of the types of metrics it collects are number of classes, methods, and variables; lines of code and statement counts; number of classes and class depth; inheritance related metrics; lack of cohesion of methods; variable use, and cyclomatic complexity. Of important note, however, JMetric works only with Java 1.1 (with Swing 1.1) and Java 1.2.

License: GNU GPL.

5.32 Jad

http://www.kpdus.com/jad.html

Status: This project is still under active development and is currently at version 1.5.8e. Up until version 1.5.6, it was available freely for both commercial and non-commercial use alike. Development was done by Pavel Kouznetsov.

Description: Written entirely in C++, it is a fast and highly ported Java class file decompiler. While the source code is not actually available, the author has taken it upon himself to compile his code for the majority of UNIX platforms, for both RISC and CISC architectures alike. There is also a version available for Win32.

Buzz: A very popular Java decompiler, it is currently used as the backend decompiler to many other front-end applications such as FrontEnd Plus, Decafe Pro, FrontJad, JavaDecompiler, JASM (the current Java decompiler for the NetBeans project), as well as many others. It is a very highly effective decompiler

and is able to accomplish a bit more in terms of its accuracy over the Homebrew Decompiler. It is a command line tool, and as such, it supports some very powerful features. One such feature is the -dead option that allows the decompiler to remove dead code that can often arise when the Java source code was compiled using optimizations. Like any other Java decompiler, if the code was obfuscated, it is likely to work with a bit more difficulties. Jad works best with programs compiled with the Sun Java compiler. It is also possible to extract Java source code from an executable (*.exe) by using the Java Split utility available from the Decafe team. Overall, it is a fast and powerful program. The program was originally designed with the intentions of being able to recover lost source code from compiled form. Like any other decompiler, it may or may not violate copyrights laws.

License: Proprietary; but free for non-commercial use.

5.33 JODE

http://jode.sourceforge.net/

Status: This project is still under active development after an interruption in releases of almost 3 years. It is currently at version 1.1.2-pre1; however, for this evaluation, we used version 1.1.1 which dates back to August 2001. The author of the program is Jochen Hoenicke who has been with the project since the beginning. The project appears to have started in 1998.

Description: JODE is both a command line driven and GUI-based Java decompiler. It can decompile both Jar and Java class files and restore them into a human readable format that generally very closely resembles the original Java source code. JODE should be considered production quality software. It also comes bundled with an optimizer and obfuscator that you can run on your class files.

Buzz: The process of decompilation is never perfect: a decompiler cannot recreate source code comments and the names of all the local variables unless debugging was turned on during the compilation process. Also, because there is often more than one way to write the same code, JODE will generally write out the easiest way to express a piece of bytecode. JODE is able to decompile class files via the use of a systematic flow analysis mechanism. It offers type deduction that can often accurately guess the type of local variables. It can also decrypt strings even if they were encrypted through the use of certain obfuscators. However, some JDK 1.3 synthetic access functions are not properly understood during decompilation. Moreover, if not all the dependent classes

are found, JODE may exit with an error. JODE may also have a hard time with complex expressions but it should still succeed in decompiling them; however, the resulting code may look cryptic. Finally, the options of using an optimizer on your class files and an obfuscator may be an interesting option for many.

License: GNU GPL.

5.34 Homebrew Decompiler

http://www.pdr.cx/projects/hbd/

Status: While it has not been developed in about one year (since 2003), the project does still appear to be active. It was developed by Pete Ryland. The project appears to have started in 1994 and carried on until some time in 2003. The project is currently at version 0.2.3.

Description: The Homebrew Decompiler is a fast, compact Java decompiler written in C++. It is portable and can run on any platform with a GCC compiler. Its source code is easy to understand and can be implemented into your own projects – one can consult it to understand more about how bytecode works.

Buzz: A fast and easy to use decompiler. Documentation is not required for the program because it is straightforward and easy-to-use. Specifiy the name of class file and its output can be piped into a file; the resulting output is a Java source code that very closely resembles the original. After having performed some preliminary tests, it has been ascertained that, overall, the decompiled source code can be easily recompiled with little to no modification. The decompiled source code is very similar in structure and syntax to the original source code. Of course, it does not correctly pick up the variable names. Nonetheless, it is a brilliant little program that you can use in your own projects. Unfortunately, as good as this little program is, it cannot replace what more developed and evolved commercial Java decompilers can accomplish.

License: GNU GPL.

5.35 Javadoc

http://java.sun.com/j2se/javadoc/index.jsp

Status: This tool is still under active development and has been included with the Sun Java JDK since Java 1.2. The current version is at 1.5, which was released with JDK 1.5 during the second half of 2004.

Description: While Javadoc is in itself not an analysis tool in the common sense of the word, it is nonetheless vital. Javadoc is a Java source code documentation tool available with all modern versions of Sun JDK (Java Development Kit). Javadoc parses Java source code and creates one or more HTML files defining the overall documentation and structure of the source code. More specifically, it pulls out the comments entered in by the programmer. The comments are found in two types, general comments entered in by the programmer and tags that are embedded within comments that document the uses of the various features of the Java programming languages. There are different types of tags and they can be placed into the following groups: author information, version information, data return information, serialized data information, deprecated information, exception class information, parameter name descriptions, and hyperlinked information to other comments within the source code. Finally, Javadoc can also help to represent the overall structure and relationships between objects, classes, data, and other objects represented by the Java programming language. If it is not already available, it is important to generate the Javadoc documentation of applications in order to assess their maturity and quality.

Buzz: Adding comments in Java is the same as in C or C++ and you can easily recognize comments with their distinctive style beginning and ending in either // or /* */. Javadoc is simple enough to be used by both novices and professional developers. It runs on all platforms where Sun JDK runs, and many other non-Sun JDK's also include their own similar Java documentation tool. Using the Javadoc tool, you will easily be able to understand the purpose of a given Java application by studying its structure, form, layout, objects, classes, other data objects, and how they interact with one another.

License: Sun Microsystems Binary Code License Agreement.

5.36 DoctorJ

http://doctorj.sourceforge.net/index.html

Status: While still under active development, this project is currently at version 5.0.0. It was written by Jeff Pace. It cannot be determined exactly when the project was started; it has been registered with SourceForge since 2001.

Description: DoctorJ picks up where Javadoc leaves off. It offers many new options and functionalities not currently available in Javadoc. DoctorJ is able to compare the Javadoc documentation against the actual source code and find and fix many things that Javadoc does not. It can find misspelled words; it

can find missing, disordered, and misspelled parameter and execution names; and it can find invalid, disordered, missing expected arguments, invalid arguments, and missing descriptions of Javadoc tags. It can also find undocumented classes, methods, fields, and parameters.

Buzz: Written entirely in Java, it should be used after the Javadoc documentation has been built using the Javadoc utility. Many programmers rely on the Javadoc documentation to give them an overview and feel for what the underlying code it represents is supposed to do. However, this often is not the case. Javadoc, as good as it is, does miss items. Because DoctorJ is written in Java, it will run on just about any platform. However, DoctorJ is not a static analysis tool. It really is meant to be used as a tool to correct your Javadoc comments. The best feature of this tool is that it is easy to use and does not require that you have an advanced knowledge of Java in order to be able to extract the maximum benefit and effect of it.

License: GNU LGPL.

5.37 Jex

http://www.cs.ubc.ca/~mrobilla/jex/

Status: This project is no longer under active development. Development began in 1998 when the authors, Martin Robillard and Gail Murphy were at the University of British Columbia in the Department of Computer Science. The last update of the program appears to be 2002. It is open source and currently at version 1.2.1.

Description: Jex is a static analysis tool, although it varies somewhat from the traditional model of static analysis tools in that it does not perform straightforward lexical or parsing analysis. Instead, it is a tool that provides information about all kinds of exceptions that could potentially be raised in a Java program. Jex works on classes, and as its output, gives a description of all exceptions that could be raised by a given method. It is also capable of finding unchecked exceptions.

Buzz: Jex is by no means a simple program. It is not easy to understand nor is it easy to use. You will have to tweak it because it does not work "out of the box". However, it is one of the few tools currently available that is able to find and detail the potential exceptions that a Java program can produce. Nevertheless, you may find Jex a tiresome product to work with because you are required to create a Jex configuration file that will probably vary from analysis to analysis. To complicate matters even further, Jex uses its own Jex stub files

to understand the classes. In addition, if the stubs do not exist, then they must be created. While this tool is a good initiative, it is a long way off from being a useful tool found in production environments. For the adventurous, however, this tool may be of interest. Perhaps what is most interesting about this tool is how it is able to find potential exceptions in a Java program.

License: GNU GPL.

6 Evaluated Tools

From all the tools that have been described in Section 5, a selection of the more promising ones has been made for evaluation purposes. This selection was based on the information given on each of the respective product website and on the anticipated relevance to Java applications assessment. Each of the 26 selected commercial and free tools was tested on a Java open source application consisting of 767 source files totalizing approximately 70 kilo-lines of code[2][3]. All the tests were conducted consistently by the same person (Frédéric Painchaud) over a period of approximately one month. They essentially consisted of exercising the tools on the chosen Java application in order to assess the tools' ease-of-use, robustness, and the pertinence of the verification reports and results. At this point, it is emphasized that these tests are not totally rigorous and are not intended to establish a consensus on the quality of these products. They were more targeted towards the assessment of the tools' general capacity to be used in the context of evaluating Java application maturity, quality, and security. It is with this clarification in mind that Table 3 on page 52, giving the overall results of the selected tools evaluation, must be interpreted.

For the purpose of rating the evaluated tools, a one-star, two-star, and three-star scale is used. A one-star rating means that the tool's capabilities were judged insufficient for proper application assessment, two-star means acceptable and three-star, excellent.

The following provides comments on the eleven top-rated (three-star) tools.

[2]To avoid any form of prejudice, the application name will stay confidential.

[3]This application is judged to be typical because it has been developed by many different individuals over a few years and necessitates both computations and I/O. Therefore, the choice of another application would not drastically change the final results of the evaluation.

6.1 Commercial Tools

6.1.1 AppPerfect DevSuite – Code Analyzer

This commercial software suite of development and quality assessment tools is probably the most comprehensive and complete that is currently available. Indeed, it is divided into 5 components: Code Analyzer, Unit Tester, Profiler, Functional Tester, and Load Tester.

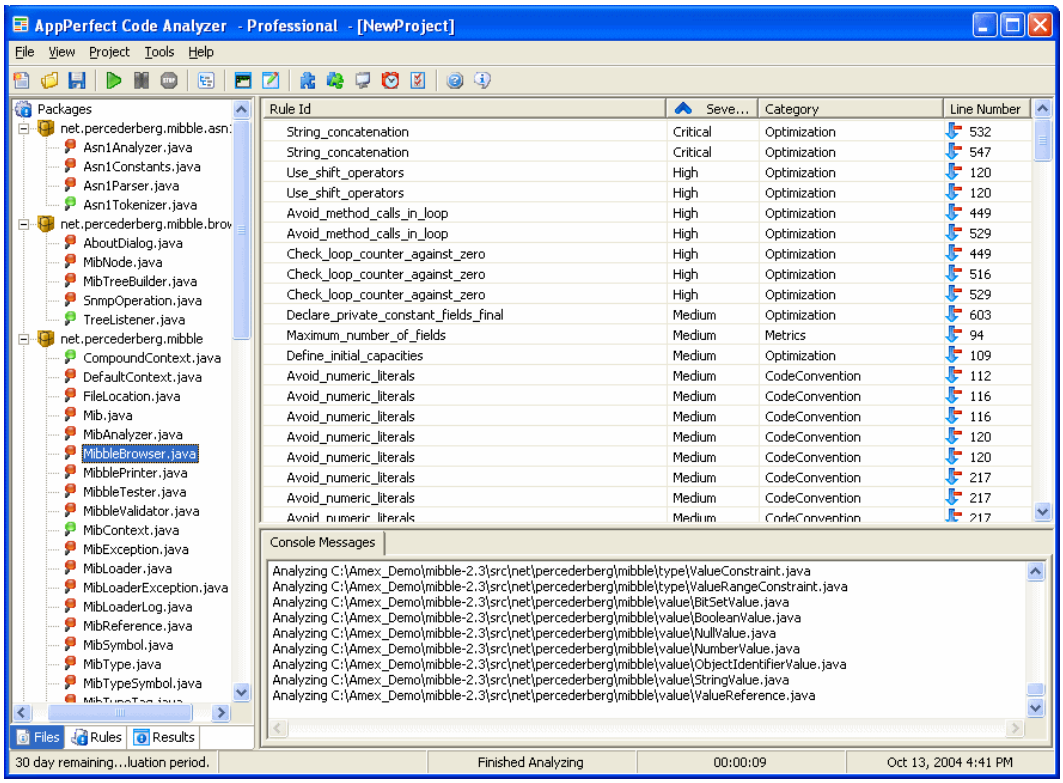

Figure 1: *AppPerfect DevSuite – Code Analyzer*

Its Code Analyzer component (Figure 1) is not only graphically appealing but also very fast, exhaustive, and robust. Its user has access to approximatively 200 rules that code should respect. Moreover, it is extensible and thus, the user can define his/her own rules. In addition, the tool can automatically modify the tested source code so that the rules are respected. For recurrent analyses (during overnight builds, for instance), it can be configured to send the results to the tester by email. This is not only very useful and practical for the developer but also for the manager who wants to quickly assess projects' quality. Due to its conviviality, any person familiar with the Java language and development tools in general can use it in the context of Java applications' maturity, quality, and security evaluation.

The other components are as powerful as Code Analyzer but less appropriate for quick application evaluation. Indeed, they are better suited when directly included into the development cycle because they facilitate testing, which is somewhat cumbersome when performed once the development is finished. Unit Tester is an automated unit test case generator and executor comparable to Jtest (see Subsection 6.1.2) but a lot less expensive. Profiler is a general-purpose, conventional Java profiler. Functional Tester is a support tool for functional testing. Program functionalities can be defined and their testing can be manually monitored within this component. Finally, Load Tester is an automated stress tester for Java applications.

With all these functionalities, AppPerfect DevSuite is a must-have for any serious Java developer. Moreover, it is also an excellent tool to evaluate Java applications.

6.1.2 Jtest

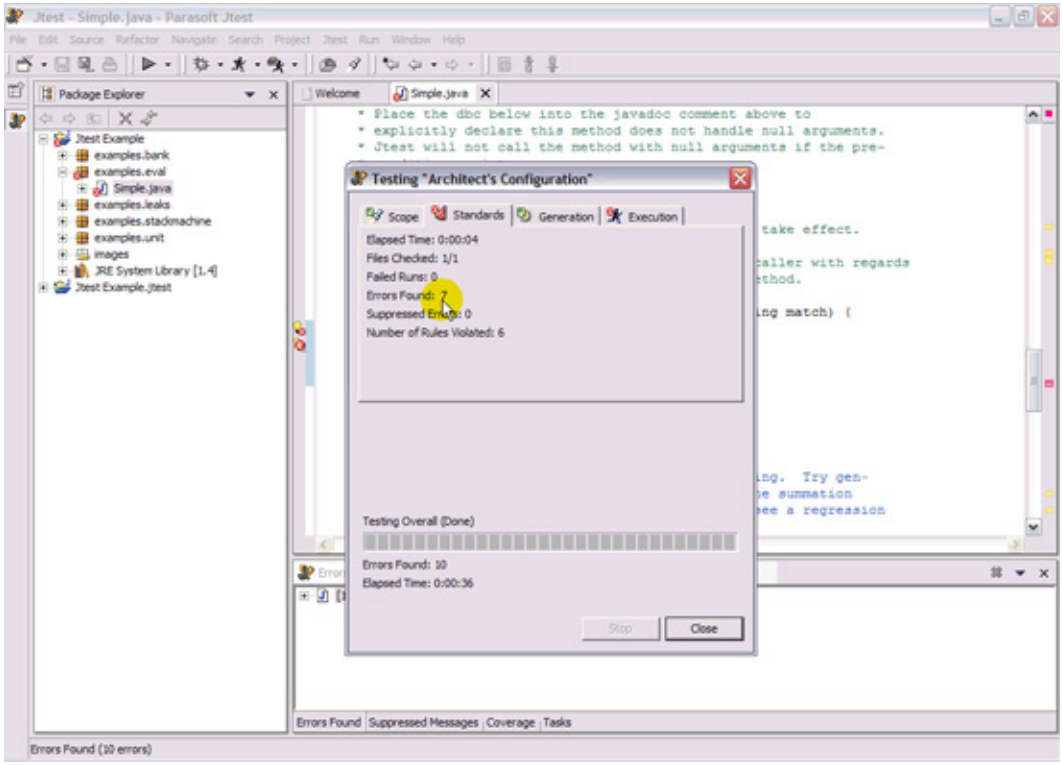

Figure 2: *Jtest*

Jtest (Figure 2) is another commercial product for automated unit test case generation and execution. It can also scan Java source code in order to determine if it respects a user-defined set of predefined rules. It is therefore comparable to AppPerfect DevSuite, while being a little less complete. It is very robust but a little

bit slow. With Jtest's higher price tag, AppPerfect DevSuite becomes a very strong competitor. Of course, Jtest performs extremely well and should not be replaced if one has already acquired it (it has been rated three stars after all) but if one is thinking about buying this kind of software for the first time, AppPerfect DevSuite is recommended. However, the existence of this product should be known so that when its usage is claimed on Java application websites, it becomes a clear sign of maturity and seriousness.

6.1.3 JStyle

This product scans Java source code in order to verify if it respects a set of rules. Being comparable to, but a little bit less impressive than, AppPerfect DevSuite and Jtest, it is mentioned because it performs quite well – being fast, robust, and having a nice graphical interface with the possibility to generate HTML reports – and should be known for the same reason as Jtest.

6.1.4 Simian Similarity Analyzer

Simian Similarity Analyzer is a very fast and robust similarity finder. It is relatively simple to use after reading its user manual. It scans Java source files (and also other kinds of text documents) in order to find parts of code that are similar. Of course, there is no direct link between similarities and bugs. However, from the developer point of view, when a bug is found at some place in the code, it should be verified if the same bug is not lurking somewhere else. When code is copied and pasted at a few different places because similar functionalities are needed in different methods, a tool like Simian Similarity Analyzer is very useful to find those sites and correct the bug wherever it may roam.

From the manager point of view, such a tool becomes even more interesting. Indeed, copied and pasted or similar code is usually a sign of bad design (or no design at all) because one of the main goals of design is to factorize the functionalities into well-defined, well-circumscribed modules. Therefore, using such tool, one can easily assess the quality of Java applications by simply verifying the quantity of similar parts of source code. Of course, even the best projects do not have zero similar part. Modules, classes, and methods cannot always be 100% generic. However, fewer similarities is certainly a good sign of maturity and quality.

6.2 Free Tools

6.2.1 PMD

PMD is an impressive tool for enforcing rules, similar to AppPerfect DevSuite and many others, but this one is very fast and developed with completeness and extensibility in mind. The rules that come with this tool are very useful and have been defined by a community of Java developers. Moreover, a simple language is provided in order to define new rules, depending on user requirements. Finally, being totally free, this tool is a must-have in order to quickly evaluate Java applications' maturity, quality, and security.

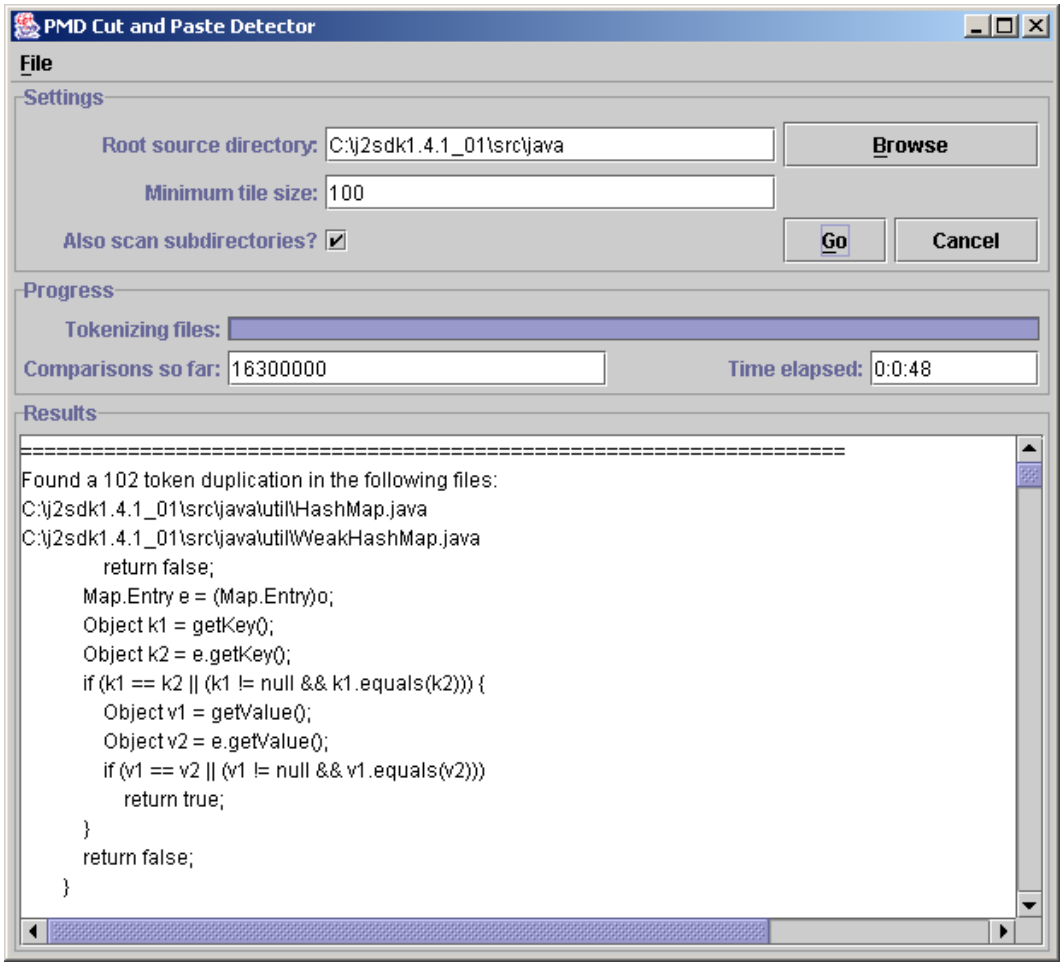

Figure 3: *Copy and Paste Detector (CPD)*

PMD also includes a component called Copy and Paste Detector (CPD) (Figure 3). This tool is also very effective, powerful, and fast. It detects copied and pasted parts of text documents like other similarity analyzers but this time, the parts are

not similar but identical. Its interface is intuitive so almost anyone can use it.

6.2.2 FindBugs

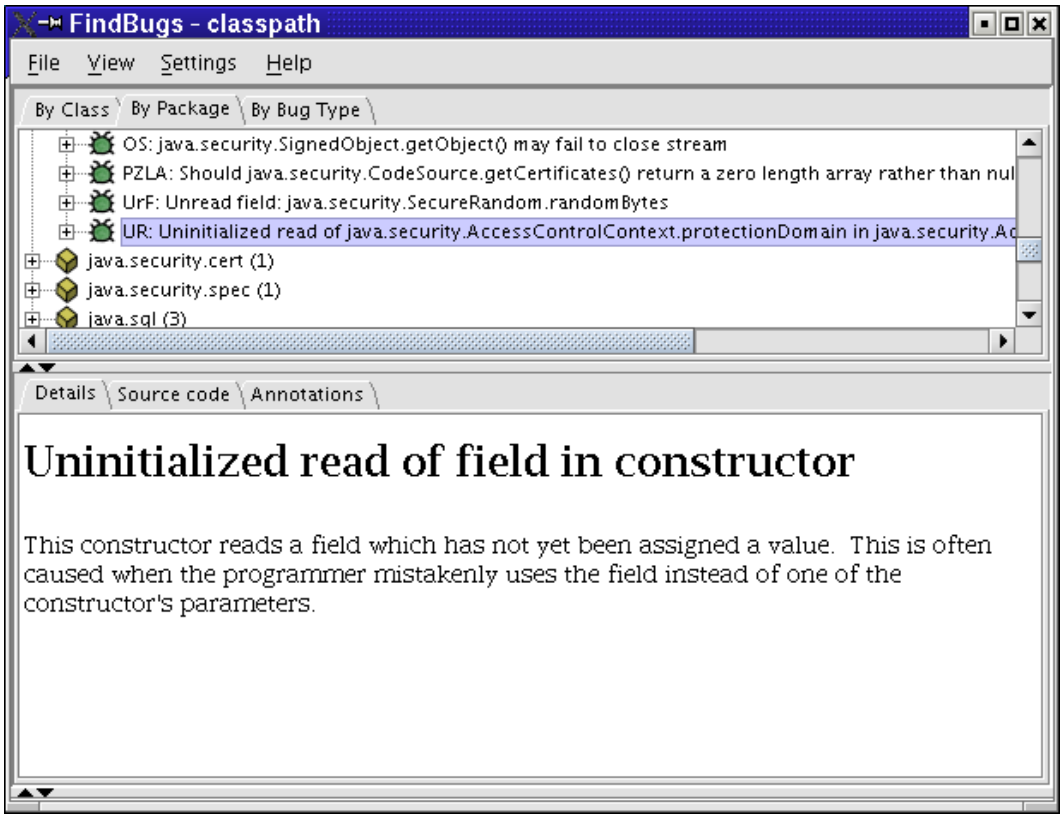

Figure 4: *FindBugs*

FindBugs (Figure 4) is a rule-enforcing tool similar to PMD that is very simple to use with its intuitive interface – yet it is also very effective and powerful. The set of hardcoded rules that it enforces translates to interesting bug patterns (potential bugs or problems caused by recurring patterns of source code) that can be automatically detected. Once again, this tool is broadly used and is a must-have.

6.2.3 Jlint

Jlint is another tool to enforce a predefined set of rules in Java source code. It is fast and simple to use so it is useful to both developers and managers in order to assess applications' overall quality. Its major advantage over the competition is that it is capable of analyzing thread synchronization in order to detect very subtle bugs like race conditions and deadlocks. Obviously, it is not infallible and thus, it can

miss some bugs, but it gives a good idea to its user about how well the application is designed and developed.

6.2.4 Hammurapi

Yet another rule-enforcing tool, Hammurapi gives precise results. It is, however, complex to use and counterintuitive. It is intended for developers rather than managers, and the former will find it useful as a second tool to enforce rules in Java source code. Managers should at least know of its existence because its effectiveness makes its usage a sign of maturity and seriousness in quality code development.

6.2.5 SimScan

The free tool called SimScan is a similarity analyzer like the commercial tool called Simian Similarity Analyzer. It is very well developed and is quite effective. It can integrate into Eclipse, JBuilder, and IDEA IDEs. However, on big projects like the one used to evaluate all these tools, it is relatively slow, slower than Simian Similarity Analyzer, for obtaining approximately the same results. It also takes a lot of memory (400 Megs for the application tested). Nevertheless, the user can abort the analysis and still consult the partial results.

6.2.6 ESC/Java 2

ESC/Java 2 is an extremely powerful static analyzer for Java, setting it apart from the competition. It analyzes the semantics of Java programs and statically (without executing the program) detects problems like possible null dereferences, potential array index overflows, and so on. However, it requires manual annotation of the source code with pre-conditions, post-conditions, and invariants. Therefore, it is not well-suited for already-implemented Java applications' evaluation, where the tester would have a complete, and unknown, source code to manually annotate. Nevertheless, projects that claim to use this tool are certainly willing to produce high quality code.

6.2.7 JavaPureCheck

JavaPureCheck was developed by Sun Microsystems Inc. during the days of the 100% Pure Java program which consisted in certifying Java applications for being 100% portable. Therefore, this tool is partially outdated (and since its development is abandoned, it will become more and more outdated as time goes on) but it can still be useful for projects developed with Java Sun JDK 1.2 and under. Indeed, it

is far from being the most interesting tool to assess applications quality but it still gives very precise details on common errors like hardcoded paths and delimiters. Therefore, this tool can be used in conjunction with others to obtain further insights in an application's internals. A good idea would be to take the rules it enforces and translate them into another tool like PMD. It should be noted, however, that this tool gives many false positives. Maybe that is the main reason why it has been abandoned.

7 Suggested Methodology for Assessing FOSS Quality

From the evaluation of all the Java code assessment tools described in this document, a simple methodology has been derived, aiming at quickly evaluating Java applications' maturity, quality, and security. This methodology's primary goal is *not* to obtain irrevocable evidences that some software is mature, high quality, and secure. Indeed, such diagnoses can only be given after a long and costly study of the code by making expert, thorough manual reviews and by using extensive testing. Rather, this methodology is *comparative* in nature and is useful to project managers who want to choose a particular Java application among three or four that are already considered adequate with respect to the project's requirements. By using this approach, project managers can easily get an approximation of the applications' quality, and therefore base their final choice on more stable grounds than only the reputation of the product or the contents of its website.

7.1 Step 0: Seek trusted sources and certified products

This preliminary step is numbered zero because it is not really in the methodology. Indeed, it rather gives advice that is useful when considering FOSS products. The advice is:

1. *Clarify and document the specific requirements of your architecture*. This is necessary in order to ascertain that the selected FOSS products will fulfill your project requirements.

2. *Acquire software from trusted sources whenever possible*. Many trusted sources such as National Security Agency, Red Hat, or SUN Microsystems Inc. exist and a few of them are listed in [5].

3. *Give preference to certified software or software that is in the process of being*

certified. In this respect, the Common Criteria are probably the best international standards currently available.

7.2 Step 1: Search for signs of maturity and quality on the application website

The real first step of this simple methodology consists of searching for a few signs of maturity and quality on the application website. Even though they sometimes imply quality, the appearance of the images and contents of the website and its layout are usually *not* something on which the final choice of software products should be based. Instead, in addition to making sure that their current project requirements are going to be fulfilled by using a given product, project managers should look for the following signs of maturity and quality on application websites:

1. *The last modification to the project has happened recently*. This ensures that the project is still active and that support will be easier to obtain if necessary.

2. *There is a bug database, it is not empty, and the bugs it contains are corrected by the project owners*. This is the first sign that the project owners are at least concerned by code quality.

3. *One of the tools mentioned in this document is claimed to be used during development*. This means that project owners are serious about quality and security.

4. *In the package downloaded from the project's website, test classes are included (usually in a directory called "tests")*. This not only tells that the project owners test their software but also that project managers (or their developers) can also test it immediately or after making some changes.

Of course, reality is not always so simple and sometimes, quality projects will not show these signs. Therefore, the final choice cannot be based solely on these four simple rules. At least a few of the tools presented in this document must be used in order to get more insight into the application code.

7.3 Step 2: Use FindBugs and PMD

Being totally free and very effective, FindBugs and PMD are the first tools to use to get more insight into the code of the candidate projects. FindBugs is recommended for people who are less familiar with computers, command line tools, and software development in general because it has an intuitive graphical user interface. PMD is a command line tool without a graphical interface but it is easy to use for people

familiar with such programs. Moreover, it has more predefined rules than FindBugs so it tends to give better results.

As a rule of thumb, the more violations FindBugs and PMD return, the poorer the quality of the code. Of course, these violations should at least be quickly read in order to filter any false positives or inconsequent violations. As a manner of confirming these affirmations, the results obtained for the Java open source application used throughout the tests were compared to the results obtained for another Java open source software widely considered to be very stable and robust[4]. There were almost no violations for the latter while there were hundreds for the former. In fact, the former finally revealed to be a prototype implementation, which could not be implied from its website's contents but could be reasonably assumed based on the tools' results.

7.4 Step 3: Use PMD's Copy and Paste Detector (CPD)

This next step involves using CPD (which is part of PMD) in order to find parts of code that have been copied and pasted at different places. Once again, as a rule of thumb, the more identical parts are found by the tool, the poorer the design of the application. Moreover, many copied and pasted parts usually means poor maintainability, which means that the software quality will probably decrease over time.

7.5 Step 4: Use AppPerfect DevSuite (optional)

To go one step further into Java application assessment, one has to spend some money and buy AppPerfect DevSuite. For the price, this is the best commercial tool available that is capable of enforcing rules in Java source code. Moreover, it can be used to perform tests if project managers are willing to be even more certain about application quality. Most of the time, however, FindBugs and PMD are sufficient and therefore, this step is somewhat optional.

7.6 Step 5: Use Simian Similarity Analyzer (optional)

The final (also optional) step is to buy Simian Similarity Analyzer to complement PMD's CPD and find not only identical parts of code but also similar ones.

[4]Once again, to avoid any form of prejudice, this other application's name is not given.

After completing this suite of tests, project managers have high-level insight into the applications' code and are therefore better-informed and can make a better choice regarding which Java application they should adopt for their project.

8 Conclusion

This document demonstrates that many tools exist to statically analyze Java source code in order to enforce a set of rules, to detect bug patterns, or to find similarities (an indicator of difficult-to-maintain software). Moreover, many of these tools are not only free but also very effective and useful. In summary, Java is now a mature programming language that has a number of major advantages over C++ and benefits from a large community that develops tools to give even better support to quality assurance. By using these tools, one can develop a simple, comparative methodology to quickly assess software quality in the context of helping project managers choose the right product among a handful of candidates that fulfill their requirements. Moreover, these tools can be used to build more robust and high quality software, which is of great concern these days because modern society increasingly depends on correct computer operation.

References

1. Charpentier, Robert and Carbone, Richard (2004). Free and Open Source Software – Overview and Preliminary Guidelines for the Government of Canada. (External Client Report ECR 2004-232). Defence Research and Development Canada. iii, v, 1

2. Inc, Sun Microsystems (2005). Java Performance. World Wide Web. 1

3. Anonymous (2005). Ada 95, Java, and C++. World Wide Web. 2

4. Cornelius, Barry (2005). Java versus C++. World Wide Web. 2

5. Demers, David, Charpentier, Robert, and Carbone, Richard (2005). Free and Open Source Software in Military Computing. In *Tenth International Command and Control Research and Technology Symposium*, McLean, Virginia. 44

List of acronyms

COTS	Commercial-off-the-shelf
FOSS	Free and Open Source Software
WWW	World Wide Web

Glossary

Dynamic analysis	A set of techniques to analyze software during execution.
Formal methods	A set of hardware and software analysis methods that are based on mathematical formalism, symbolism, and logic.
Java Security Manager	A component of the Java Virtual Machine that is responsible for ensuring high level security.
Java Virtual Machine	The main component of the Java Platform that is responsible for safely and securely executing Java programs.
Model checking	A technique used to verify that a property holds on every possible state of a hardware or software system.
Static analysis	A set of techniques to analyze software prior to execution.
Validation	A process that is used to ensure that the design requirements are met in a product being developed.
Verification	A process that is used to ensure that a product being developed meets some predefined quality standards.

Distribution list

Internal Distribution
DRDC Valcartier TM 2005-226

1 - Director General
3 - Document Library
1 - Frédéric Painchaud (author)

External Distribution
DRDC Valcartier TM 2005-226

1 - Directorate R & D – Knowledge and Information Management (PDF file)

Tools	Capabilities					
	Supporting Tests	**Detecting Pitfalls**	**Enforcing Coding Standards**	**Calculating Metrics**	**Decompiling**	**Miscellaneous**
AppPerfect DevSuite – Code Analyzer	X	X				
Jtest	X	X				
ESC/Java 2	X	X				
Jlint	X	X				
Wasp	X	X				
Bandera	X	X				
Excelsior FlawDetector	X	X				
InsectJ	X					X
SimScan	X					X
Simian Similarity Analyzer	X					X
JVerify	X					X
Hyades	X			X		
Lint4J	X					
JCover	X					
JSynTest	X					
JUnit	X					
JStyle		X	X	X		
JCSC		X	X	X		
PMD		X	X			
JiveLint		X	X			
AzoJavaChecker		X		X		
Aubjex		X				X
QStudio Pro		X				
ASSENT		X				
JavaChecker		X				
FindBugs		X				
JavaWizard		X				
Hammurapi			X	X		
JavaPureCheck			X			
CheckStyle			X			
JMetric				X		
Jad					X	
JODE					X	
Homebrew Decompiler					X	
Javadoc						X
DoctorJ						X
Jex						X

Table 2: *Tools by Capabilities*

Tools	Results
AppPerfect DevSuite – Code Analyzer	★★★
Jtest	★★★
ESC/Java 2	★★★
Jlint	★★★
Wasp	★
Excelsior FlawDetector	★★
InsectJ	N/A
SimScan	★★★
Simian Similarity Analyzer	★★★
JVerify	N/A
Lint4J	★★
JCover	N/A
JSynTest	N/A
JStyle	★★★
JCSC	★
PMD	★★★
JiveLint	★★
AzoJavaChecker	★★
Aubjex	N/A
QStudio Pro	★★
ASSENT	N/A
JavaChecker	★★
FindBugs	★★★
JavaWizard	★
Hammurapi	★★★
JavaPureCheck	★★★
CheckStyle	★★
JMetric	★
Jex	★

Legend

N/A: Tests were planned for this tool but either it could not be downloaded because it was not available on the website or it was deemed inappropriate for application assessment because it must be included in the development cycle and cannot be used once the application development is completed.

★: This tool's capabilities were judged *insufficient* for proper application assessment.

★★: This tool's capabilities were judged *acceptable* for proper application assessment.

★★★: This tool's capabilities were judged *excellent* for proper application assessment.

Table 3: *Evaluated Tools Results*

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence Research and Development Canada – Valcartier
2459 Pie-XI Blvd North, Québec, QC, Canada, G3J 1X5

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable).

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title).

Java software verification tools: evaluation and recommended methodology

4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)

Painchaud, F. ; Carbone, R.

5. DATE OF PUBLICATION (month and year of publication of document)

February 2007

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc).

66

6b. NO. OF REFS (total cited in document)

5

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered).

Technical Memorandum

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include address).

Defence Research and Development Canada – Valcartier
2459 Pie-XI Blvd North, Québec, QC, Canada, G3J 1X5

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Specify whether project or grant).

15BP01

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written).

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique.)

DRDC Valcartier TM 2005-226

10b. OTHER DOCUMENT NOs. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution beyond the audience specified in (11) is possible, a wider announcement audience may be selected).

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

Free and Open Source Software (FOSS) is now being adopted worldwide in many different organizations, such as governments, agencies, and companies, which want to reduce their software acquisitional cost, decrease their economic loss at the national level caused by commercial software imports, and develop national information technology expertise by means of access to source code. However, this current trend makes these organizations face a new challenge in software maturity, quality, and security assessment. Indeed, how can they be assured that the FOSS they are selecting among thousands of different projects is mature and secure enough? Hopefully, the availability of source code enables not only manual peer reviewing but also automatic and mechanical verification with a plethora of software tools.

This document categorizes, lists, and describes many of these automated software verification tools for Java. Java has been selected as the programming language because an increasing percentage of FOSS is now developed with this modern and more easily verifiable language. Moreover, this document details the best tools and gives a verification methodology in order to efficiently assess overall Java software quality, which is a pressing need for concerned managers.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title).

Software Assurance
Software Reliability and Security
Software Validation and Verification
C2IS Certification
Static Analysis
Dynamic Analysis
Java
Java Security

Defence R&D Canada

Canada's Leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

WWW.drdc-rddc.gc.ca