Volume 8, No. 5, May-June 2017
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
RESEARCH PAPER
Available Online at www.ijarcs.info

A Novel Automatic Source Code Defects Detection Framework and Evaluation on Popular Java Open Source APIs

K Venkata Ramana
Assistant Professor
Department of Computer Science & Engineering
VNR Vignana Jyothi Institute of Engineering and Technology
Hyderabad, Telangana, India.

Dr K Venugopala Rao
Professor
Department of Computer Science & Engineering
G.Narayanamma Institute of Technology and Science
Hyderabad, Telangana, India.

Abstract: The unmatched growth in the automation and application of software code segments for automation is the main reactive reason for improvements in industrial, education, and healthcare and security sectors. The deployed code segments or the complete application used for the purpose is developed extensively with ample amount of features. The number of lines of code and number of man-hours deployed to build the applications are gigantic. In addition to that, the testing of the applications is the added cost for the development cycle. However, in spite of the best practice efforts, the applications can fail in real-time due to undetected errors resulting in fault and failure. Hence, the demand of the modern code development industry to the current research trend is to automate the testing process and derive a framework for enhanced defects detection. This work proposes a novel code defect detection technique to deep scan the code and report all possible bugs and defects and errors. To justify the thoughts, the framework tests the most popular java open source APIs and demonstrates the results. Another novel outcome of this work is to build a generic defect metric for all classes of source code.

Keywords: Coding Standards, Bad Practices, Null Path, Locale Errors, Reference Errors, Parallelism Error, Performance, Security, Suspected Error

I. INTRODUCTION

The ease of static software code mining techniques and tools for detection of defects are useful and productive in the current age of research and studies. The frameworks and tools are effective in order to improve the automatic defect detections by integrating into development environment for building high quality and highly reliable codes.

Nevertheless, the mining tools and frameworks are often reported for highlighting overly huge amount of errors and warnings including the false positive errors and detects. Henceforth the developers need to indulge a massive amount of time in solving and reporting the false positive and true positive errors apart from the code development life cycle time frame. This makes the development life cycle to be tedious and large in terms of time frame and completion time efficiency resulting into poor development practice and defeating the complete purpose of the automatic detection of defects.

The tools and the frameworks are expected to involve deep scan for the code defect detection considering the ignorance factors for false positive alerts. Henceforth, multiple research attempts are been made to incorporate priority based defect detection. Those attempts made a significant reduction in the number of alerts generated by the tools and frameworks, nevertheless the effectiveness of the priority rules are debatable.

Significantly, the source code development industry has progress into multiple vertical of automation and resulting into various coding standards. The coding standards and made to

ensure the reliability of those applications complying the requirements of the specified domains. Henceforth, a standard priority based defect detection technique is always debatable.

Thus, the need for a novel metric for minimized and robust fault detection is the demand from the current research. Likewise, the metric should result into an automated framework for detection of detects.

This work proposes and validates a novel metric for detection of defects and evaluates the performance by producing a novel automation framework.

The rest of the paper is organized as, in Section II the current state of art is been demonstrated, in Section III the proposed defect metric is formulated, in Section IV the framework for the automation is demonstrated, in Section V the results are been discussed and in Section VI the work concludes.

II. OUTCOMES FROM THE PARALLEL RESEARCHES

Software inspection tools have been studied widely. Zitser et al. evaluated several open source static analysers with respect to their ability to find known exploitable buffer over-flows in open source code [1] [2]. Engler et al. evaluate the warnings of their defect detection technique [3].

It was examined the results of applying five tools [4], specifically Bandera, ESC/Java 2, FindBugs, JLint and PMT, to a variety of Java programs over different checking tasks. Thereby it was possible to crosscheck their bug reports and warnings. In experimental results it was showed that none of the tools can fully replace one-another, and indeed the tools often find non-overlapping bugs (mostly warnings are distinct). There is also no correlation of warning counts between pairs of tools. Therefore, it was proposed a meta-tool, which combines the output of the tools together, looking for particular lines of code, methods, and classes that many tools warn about. Thus it enables to precisely identify false positives and false negatives. Summarizing,

this meta-tool automatically combine and correlate their output. It was concluded that the main difficulty in using the tools is simply the quantity of output (mostly because of false positives). Wagner et al. compared results of defect detection tools with those of code reviews and software testing [5]. Their main finding was that bug detection tools mostly find different types defects than testing, but find a subset of the types found by code reviews [6]. Warning types detected by a tool are analysed more thoroughly than in code reviews [7].

Heckman et al. proposed a benchmark and procedures for the evaluation of software inspection prioritization and classification techniques, focused at Java programs [8].

In recent years, many solutions have been proposed to reduce the number of inspected violations and, instead, emphasis on the most relevant ones, according to some criterion [9] [10].

The classical approach most automated inspection tools use for prioritizing and filtering results is to classify the results based on several levels (statically). Such levels are associated with the type of defects detected; they are obvious of the actual code that is being analysed and of the location or frequency of a given defect. Therefore, the ordering and filtering that can be achieved while using this technique is rather crude.

Kremenek and Engler [11], proposed Z-ranking, a statistical approach to reduce the number of false positives due to inaccurate static analysis [12]. With that technique it is possible to rank the output of static analyses tools so that more important warnings will tend to be ranked more highly. Z Ranking is intended to rank the output of a particular bug checker. They prioritize checks (warning categories) using the frequency of check results [13] [14].

III. A NOVEL DEFECT DETECTION METRIC

The generic metric for defect detection for all category of source codes are subjected to debate as the majority of the best practices are generated and recommended based on the domain to which the source code to be administrated. Thus the need for the novel multi domain applicable metric is the need for the recent research.

This work formulates a novel metric for defect detection and considers the best practices from majority of the domains [Table – I].

TABLE I: NOVEL DEFECT DETECTION METRIC

Serial Number	Parameter Name	Parameter Description	Severity Measure		
			Informational	Moderate	Severe
1	Domain Specific Coding Standards (CS)	Coding mistakes, which violate the organization standards for internal quality.	Violation of language or application preferences	Violation of source code naming conventions	Violation of financial or class hierarchy
2	Bad Practices (BPAS)	Logical mistakes in the source code analysed to be identified as defect during production	Logical / Programming symbolic mistakes	Assignment or allocation errors	Serializability problem
3	Suspected Errors (SE)	The mistakes in source code, that leads to the ambiguity in the source code and the coding branching during the execution may lead to defects	Unassigned value errors	Un-reachable code errors	Faulty exception handling
4	Locale Errors (LE)	Generic mistakes of loading the library for location language pack in order to manipulate the UI and units	Misplaced or misspelled language pack location or file name	-	-
5	Reference Errors (RefE)	The errors in the source code, which leads to the exposure of the class members to other class members	Object reference errors	Parent class reference error	Visibility parameter errors
6	Parallelism Error (PE)	Generally and popularly known as the thread error due to the ill coding of the source code and unable to handle the mutating semaphore	Thread unlock errors	Repeatable read errors	Information unreachable errors
7	Performance Errors (PerE)	In spite of the highest graded hardware resources, the source code fails to utilize the available resources due to inappropriate use of variables classes and storage class or programming framework	Parallel technique errors	Storage class errors	Algorithm technique errors
8	Security Errors (SeqE)	Errors and mistakes causing the leak of information due to mishandling or misuse of the	-	-	Pointers or access modifiers errors

		appropriate data structures			
9	Null Path Errors (NPLE)	The logical errors caused in the code to have some branches which are basically considered as always false conditions or sometimes the branches considered as un reachable code	-	-	-

Based on the proposed metric a framework for automatic detection of code defects is also been build. This work explains the novel framework in the next section.

IV. A NOVEL AUTOMATED FRAMEWORK FOR DEFECT DETECTION

The need for the automated application to evaluates the sources codes and generates the compliance record or score based on the proposed metric is the highest recommendation in order to prove the benefits and applicability of the metric. Hence, this work proposes a Java based application framework for evaluating the performance of the sources codes based on the proposed metric. The architecture of the application framework is discussed in this work as well [Fig – 1].

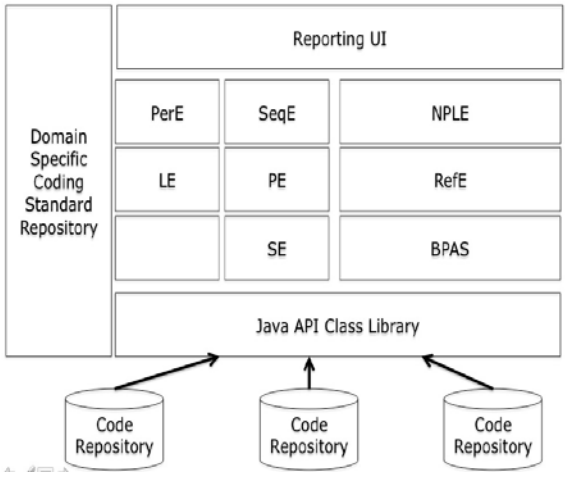

Figure – 1: Architecture of the Proposed Automated Defect Detection Framework

Domain Specific Coding Standards (CS): Coding mistakes, which violate the organization standards for internal quality.
Bad Practices (BPAS): Logical mistakes in the source code analysed to be identified as defect during production.
Suspected Errors (SE): The mistakes in source code, that leads to the ambiguity in the source code and the coding branching during the execution may lead to defects.
Locale Errors (LE): Generic mistakes of loading the library for location language pack in order to manipulate the UI and units.
Reference Errors (RefE): The errors in the source code, which leads to the exposure of the class members to other class members.
Parallelism Error (PE): Generally and popularly known as the thread error due to the ill coding of the source code and unable to handle the mutating semaphore.
Performance Errors (PerE): In spite of the highest graded hardware resources, the source code fails to utilize the available resources due to inappropriate use of variables classes and storage class or programming framework.
Security Errors (SeqE): Errors and mistakes causing the leak of information due to mishandling or misuse of the appropriate data structures.

Null Path Errors (NPLE): The logical errors caused in the code to have some branches that are basically considered as always-false conditions or sometimes the branches considered as un-reachable code.
In the next section the work evaluates all the proposed claims and establishes all improvements, validating the points over popular open source Java APIs.

V. RESULTS AND DISCUSSIONS

The standard applications are built up on the standard java open source code APIs. The proposed application is tasted on the same API in order to validate the applicability of the metric parameters and applicability of the developed framework by this application.

The descriptions of the tested APIs are mentioned here:

- **ant.jar**: Apache Ant is a software tool for automating software build processes, which originated from the Apache Tomcat project in early 2000. It was a replacement for the Unix make build tool, and was created due to a number of problems with the unix make. It is similar to Make but is implemented using the Java language, requires the Java platform, and is best suited to building Java projects. [15]
- **AppleJavaExtensions.jar**: AppleJavaExtensions.jar is a type of JAR file associated with Developer DVD Series developed by Apple Computer Inc. for the Windows Operating System [16].
- **ASM-DEBUG-ALL-5.0.2.JAR**: ASM is an all-purpose Java bytecode manipulation and analysis framework. It can be used to modify existing classes or dynamically generate classes, directly in binary form. Provided common transformations and analysis algorithms allow easily assemble custom complex transformations and code analysis tools [17].
- **BCEL-6.0-SNAPSHOT.JAR**: The BCEL API abstracts from the concrete circumstances of the Java Virtual Machine and how to read and write binary Java class files. The API mainly consists of three parts. Firstly, A package that contains classes that describe "static" constraints of class files, i.e., reflects the class file format and is not intended for byte code modifications. The classes may be used to read and write class files from or to a file. This is useful especially for analysing Java classes without having the source files at hand. Secondly, a package to dynamically generate or modify JavaClass or Method objects. It may be used to insert analysis code, to strip unnecessary information from class files, or to

implement the code generator back-end of a Java compiler. Lastly, various code examples and utilities like a class file viewer, a tool to convert class files into HTML, and a converter from class files to the Jasmin assembly language [18].

- **COMMONS-LANG-2.6.JAR**: The standard Java libraries fail to provide enough methods for manipulation of its core classes. Apache Commons Lang provides these extra methods. Lang provides a host of helper utilities for the java.lang API, notably String manipulation methods, basic numerical methods, object reflection, concurrency, creation and serialization and System properties. Additionally it contains basic enhancements to java.util.Date and a series of utilities dedicated to help with building methods, such as hashCode, toString and equals [19].
- **DOM4J-1.6.1.JAR**: dom4j is an open source Java library for working with XML, XPath and XSLT. It is compatible with DOM, SAX and JAXPstandards.The library is distributed under a BSD-style license [20].

The results produced by this work are been analysed [Table– II].
The results are also visually analysed [Figure–2] and observed that the every category of the applications are detectable through the proposed framework.

TABLE II: DETECTION OF DEFECTS IN THE SOURCE CODE

Java Source Code Name	Metric Parameters									
	Number of Classes	Coding Standards	Bad Practices	Null Path	Locale Errors	Reference Errors	Parallelism Error	Performance	Security	Suspected Error
ant.jar	1528	13	167	7	128	55	60	112	2	252
AppleJavaExtensions.jar	174	0	1	0	0	0	0	0	0	0
asm-debug-all-5.0.2.jar	321	0	26	0	8	6	0	37	0	74
bcel-6.0-SNAPSHOT.jar	904	1	28	0	8	93	0	16	0	65
commons-lang-2.6.jar	1300	4	35	0	4	10	0	2	0	72
dom4j-1.6.1.jar	420	4	68	3	10	10	3	8	0	119

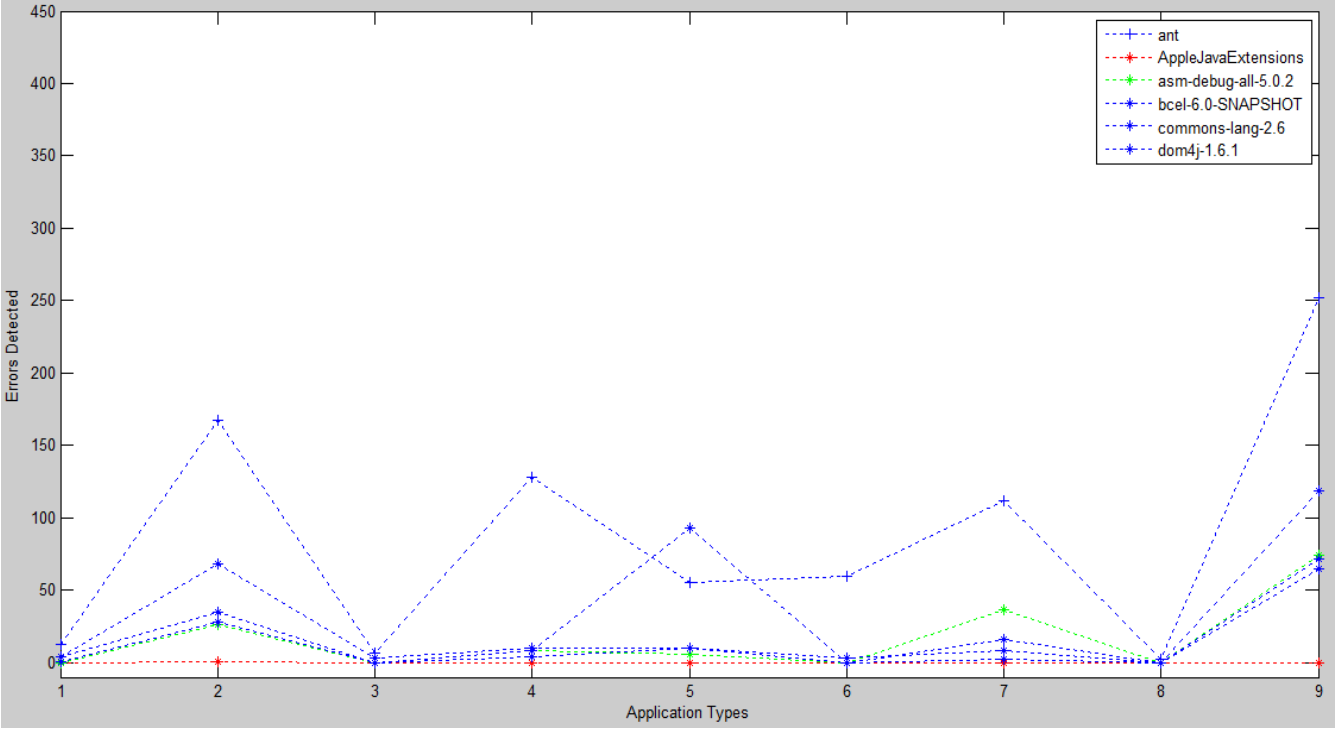

Figure – 2: Graphical Analysis of the Source Code Errors

VI. CONCLUSION

The continuous demand for domain specific defect detection framework and denied by the research groups

considering the ambiguity of possibilities by introducing multiple frameworks. Henceforth the needs for a novel metric for inter domain application source code evaluation and to automate the process for the same metric could not be ignored. Hence, this work establishes the toughs for the novel metric empowered by the standards for each domain and establish the automated tool for defect detection. This work also results into a novel technique of domain specific source code defect technique which will lay down the path for further enhancement in this new dimension of source code mining.

REFERENCES

[1] Tim A. Wagner, Vance Maverick, Susan L. Graham, and Michael A. Har- rison. Accurate static estimators for program optimization. In PLDI '94: Proceedings of the ACM SIGPLAN 1994 conference on Programming lan- guage design and implementation, pages 85–96, New York, NY, USA, 1994. ACM.

[2] Cathrin Weiss, Rahul Premraj, Thomas Zimmermann, and Andreas Zeller. How long will it take to fix this bug? In MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories, page 1, Washington, DC, USA, 2007. IEEE Computer Society.

[3] MishaZitser, Richard Lippmann, and Tim Leek. Testing static analysis tools using exploitable buffer overflows from open source code. In SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering, pages 97–106, New York, NY, USA, 2004. ACM.

[4] Dawson R. Engler, David Yu Chen, and Andy Chou. Bugs as inconsistent behavior: A general approach to inferring errors in systems code. In SOSP, pages 57–72, 2001

[5] Nick Rutar, Christian B. Almazan, and Jeffrey S. Foster. A comparison of bug finding tools for java. In ISSRE '04: Proceedings of the 15th Inter- national Symposium on Software Reliability Engineering, pages 245–256, Washington, DC, USA, 2004. IEEE Computer Society.

[6] Stefan Wagner , Jan Ju¨rjens , Claudia Koller , Peter Trischberger, and Tech - nischeUniversita¨tMünchen Comparing bug finding tools with reviews and tests. In In Proc. 17th International Conference on Testing of Communicat- ing Systems (TestCom'05), volume 3502 of LNCS, pages 40–55. Springer, 2005.

[7] Youfeng Wu and James R. Larus. Static branch frequency and program profile analysis. In MICRO 27: Proceedings of the 27th annual international symposium on Microarchitecture, pages 1–11, New York, NY, USA, 1994. ACM.

[8] Tim A. Wagner, Vance Maverick, Susan L. Graham, and Michael A. Har- rison. Accurate static estimators for program optimization. In PLDI '94: Proceedings of the ACM SIGPLAN 1994 conference on Programming lan- guage design and implementation, pages 85–96, New York, NY, USA, 1994. ACM.

[9] Sarah Heckman and Laurie Williams. On establishing a benchmark for eval- uating static analysis alert prioritization and classification techniques. In ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, pages 41–50, New York, NY, USA, 2008. ACM.

[10] Ted Kremenek, Ken Ashcraft, Junfeng Yang, and Dawson Engler. Correla- tion exploitation in error ranking. In SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering, pages 83–93, New York, NY, USA, 2004. ACM.

[11] Ted Kremenek and Dawson Engler. Z-ranking: using statistical analysis to counter the impact of static analysis approximations. In SAS'03: Proceed- ings of the 10th international conference on Static analysis, pages 295–315, Berlin, Heidelberg, 2003. Springer-Verlag.

[12] Ted Kremenek and Dawson Engler. Z-ranking: using statistical analysis to counter the impact of static analysis approximations. In SAS'03: Proceed- ings of the 10th international conference on Static analysis, pages 295–315, Berlin, Heidelberg, 2003. Springer-Verlag.

[13] Sunghun Kim, Kai Pan, and E. E. James Whitehead, Jr. Memories of bug fixes. In SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering, pages 35– 45, New York, NY, USA, 2006. ACM.

[14] Sunghun Kim, Thomas Zimmermann, Kai Pan, and E. James Jr. White- head. Automatic identification of bug-introducing changes. In ASE '06: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, pages 81–90, Washington, DC, USA, 2006. IEEE Computer Society.

[15] Suereth, Joshua; Farwell, Matthew (2015). SBT in Action: The simple Scala build tool. Manning Publications. ISBN 9781617291272.

[16] https://developer.apple.com/programs

[17] Eric Bruneton, ASM 4.0: A Java bytecode engineering library, 2011

[18] https://commons.apache.org/

[19] http://commons.apache.org/proper/commons-lang/

[20] Generated by Apache Maven Doxia (2010-04-05). "dom4j - Introduction". Retrieved 2012-08-29.

About the Authors:

K Venkata Ramana was born in Guntur, Andhra Pradesh, in 1978. He received MTech in Computer Science & Engineering from Jawaharlal Nehru Technological University, Hyderabad in 2010 and currently pursuing Ph.D in Computer Science & Engineering from Jawaharlal Nehru Technological University Hyderabad, with 15 years of teaching experience.

He is working as an Assistant Professor in the Department of Computer Science & Engineering, VNR Vignana Jyothi Institute of Engineering and Technology, Hyderabad, India. Earlier, he worked as an Associate Professor in the department of CSE, Bhoj Reddy Engineering College for Women, Hyderabad, India, from 2006-2016. He started his profession in 2001 as an Assistant Professor in the Department of Computer Applications, St.Johns Institute of Science & Technology, Hyderabad, India.

His research areas are Data Mining, Machine Learning, Software Engineering, with an emphasis on mining software engineering data and software verification.

K Venugopala Rao was born in Vijayawada, Andhra Pradesh, in 1963. He received the B.Tech degree in Electronics and Communication Engineering from Jawaharlal Nehru Technological University, Hyderabad in 1985, the M.Tech degree in Computer Science and Engineering from the Jawaharlal Nehru Technological University, Hyderabad in 1997 and Ph.D degree in the area of Computer Science & Engineering from Osmania University, Hyderabad in 2008.

From 2006, he is working as a Professor, Department of Computer Science and Engineering at G.Narayanamma institute of technology and science (GNITS) Shaikpet, Hyderabad. From 2002 to 2006, he was an Associate Professor, in the Department of Computer Science and Engineering at G.Narayanamma Institute of Technology & Science Shaikpet, Hyderabad. From 1999t to 2002, he was an Associate Professor in the Department of Computer Science and Engineering at Koneru Lakshmaiah College of Engineering, Greenfields, Vadeesvaram, Guntur. From 1997 to 99, he was an Assistant Professor in the Department of Computer Science and Engineering at VR Siddhartha Engineering College, Vijayawada. From 1992 to 1995, he was an Assistant Professor, Department of Computer Science & Engineering at JNTUCE, Hyderabad (1992-1995). From 1989 to 1992, he was a Technical Officer in CSG Group at ECIL Hyderabad. From 1988 to 1989 he was a Quality control engineer at Ashok Leyland, Hyderabad (1988-1989). From 1985 to 1988, he started working as an Engineer (Maintenance) at Radiant Cables Ltd, Hyderabad (1985-1988).

His research interests include Network Security, data mining, statistical methods and their applications to software engineering.