International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 7, Issue 6, June 2017)

An Evaluation of Popular Code Mining Frameworks through Severity Based Defect Rule

K Venkata Ramana[1], Dr K Venugopala Rao[2]

[1]*Assistant Professor, Department of Computer Science & Engineering, VNR Vignana Jyothi Institute of Engineering and Technology, Hyderabad, Telangana, India.*

[2]*Professor, Department of Computer Science & Engineering, G. Narayanamma Institute of Technology and Science, Hyderabad, Telangana, India.*

***Abstract*— Traditional testing methods for detection of the bugs are reaching to a point where an additional testing phase is being considered as an additional load on the software code development life cycle. The recent progresses of research to reduce the life cycle of software development made multiple notable contributions. The major contributions are consisting of inclusion of defects predictions into the development environment to enable the developers and researchers to predict detects and avoid the mistakes during the development phase. The result of this approach is significant in reducing the defect correction duration and improved the software source code standard. However, the scope of improvement is still persists in resent advancements as the detection and measure of the defects are overrated or underrated sometimes. Hence, this work demonstrates a novel severity based defect tracking system and compare the performance with highly popular detect tracking frameworks.**

***Keywords*— Severity, Defect Detection, PMD, Check Style, Checker Framework**

I. INTRODUCTION

The inconsistency in the software source code is mainly caused by the lines of codes those are unreachable during a normal course of the execution. The inconsistent code or the unreachable codes are often found to be one of the major causes of the defects during the production phase [1]. During the static analysis of the source code, it is important to detect the control paths where the regular execution will never be addressed. Henceforth multiple research attempts are made to define the feasibility of detecting the feasible paths for program executions. The results of those research attempts are the tools and frameworksto detect paths in the source code which are inconsistent [2] [3] [4] [5]. Observations made in this work on the detection of the bugs in the Linux kernel [6], in the source code of Eclipse [7] and also in the Tomcat server [8].

Nevertheless, the detected inconsistent codes are sometimes overrated, meaning all the detected inconsistent codes may not lead to detects. Henceforth, the detection of those reported detects may cause inconsistency in defect detection and need to be reported as positive. Thus it is the demand of recent research to identify the unreachable codes, which will cause detects. In order to achieve the goal, this work introduces the severity based measure for detecting the defects based on the unreachable code.

The rest of the work is furnished as in Section-II the severity based defect detection method is discussed, in Section-III the outcomes of the parallel research is explained, in the Section-IV the comparative study results are been analysed and in the Section-V this work presents the conclusion.

II. SEVERITY BASED DEFECT DETECTION

The detection of the defects proposed by various researchers is over rated due to the missing factor of severity. The detection of the over rated defects can slow down the development life cycle and result in zero improvement in the software development process. The defect severity measure will ensure the reduction in false positive reports, thus results into a massive reduction in development time.

The researchers do not categorize a majority of the defects, as the final issues caused by the defects are unpredictable. However, this work presents the severity measures based on the outcome of PMD [10].

The details of the severity descriptions are discussed in this section [Table-1]. The RuleSet description contains the standard Java code rules. The source code to be tested in this framework will be kept in the severity level where the source code violates the maximum rules.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 7, Issue 6, June 2017)

Table–1
Severity Based RuleSet Violations & Detections

RuleSet Type	RuleSet Description	Severity Category
Basic JSP Rules	• NoLongScripts • NoScriptlets • NoInlineStyleInformation • NoClassAttribute • NoJspForward • IframeMissingSrcAttribute • NoHtmlComments • DuplicateJspImports • JspEncoding • NoInlineScript	Blocker / Checker Violation
Basic Development / Best Practices Rules	• EmptyCatchBlock • EmptyIfStmt • EmptyWhileStmt • EmptyTryBlock • EmptyFinallyBlock • EmptySwitchStatements • JumbledIncrementer • ForLoopShouldBeWhileLoop • UnnecessaryConversionTemporary • OverrideBothEqualsAndHashcode • DoubleCheckedLocking • ReturnFromFinallyBlock • EmptySynchronizedBlock • UnnecessaryReturn • EmptyStaticInitializer • UnconditionalIfStatement • EmptyStatementNotInLoop • BooleanInstantiation • UnnecessaryFinalModifier • CollapsibleIfStatements • UselessOverridingMethod • ClassCastExceptionWithToArray • AvoidDecimalLiteralsInBigDecimalConstructor • UselessOperationOnImmutable • MisplacedNullCheck • UnusedNullCheckInEquals • AvoidThreadGroup • BrokenNullCheck • BigIntegerInstantiation • AvoidUsingOctalValues • AvoidUsingHardCodedIP • CheckResultSet • AvoidMultipleUnaryOperators • EmptyInitializer • DontCallThreadRun	Warning Violation / Critical Violation
Source Code Size / KLOC Rules	• NPathComplexity • ExcessiveMethodLength • ExcessiveParameterList • ExcessiveClassLength • CyclomaticComplexity • ExcessivePublicCount • TooManyFields • NcssMethodCount • NcssTypeCount • NcssConstructorCount • TooManyMethods	Warning Violation / Urgent Violation
Source Code Conflict Rules	• UnnecessaryConstructor • NullAssignment • OnlyOneReturn • UnusedModifier • AssignmentInOperand • AtLeastOneConstructor • DontImportSun • SuspiciousOctalEscape • CallSuperInConstructor • UnnecessaryParentheses	Important Violation

	• DefaultPackage • BooleanInversion • DataflowAnomalyAnalysis • AvoidFinalLocalVariable • AvoidUsingShortType • AvoidUsingVolatile • AvoidUsingNativeCode • AvoidAccessibilityAlteration • DoNotCallGarbageCollectionExplicitly • AvoidLiteralsInIfCondition • UseConcurrentHashMap	
Source Code Architecture / Design Rules	• UseSingleton • SimplifyBooleanReturns • SimplifyBooleanExpressions • SwitchStmtsShouldHaveDefault • AvoidDeeplyNestedIfStmts • AvoidReassigningParameters • SwitchDensity • ConstructorCallsOverridableMethod • AccessorClassGeneration • FinalFieldCouldBeStatic • CloseResource • NonStaticInitializer • DefaultLabelNotLastInSwitchStmt • NonCaseLabelInSwitchStatement • OptimizableToArrayCall • BadComparison • EqualsNull • ConfusingTernary • InstantiationToGetClass • IdempotentOperations • SimpleDateFormatNeedsLocale • ImmutableField • UseLocaleWithCaseConversions • AvoidProtectedFieldInFinalClass • AssignmentToNonFinalStatic • MissingStaticMethodInNonInstantiatableClass • AvoidSynchronizedAtMethodLevel • MissingBreakInSwitch • UseNotifyAllInsteadOfNotify • AvoidInstanceofChecksInCatchClause • AbstractClassWithoutAbstractMethod • SimplifyConditional • CompareObjectsWithEquals • PositionLiteralsFirstInComparisons • UnnecessaryLocalBeforeReturn • NonThreadSafeSingleton • UncommentedEmptyMethod • UncommentedEmptyConstructor • AvoidConstantsInterface • UnsynchronizedStaticDateFormatter • PreserveStackTrace • UseCollectionIsEmpty • ClassWithOnlyPrivateConstructorsShouldBeFinal • EmptyMethodInAbstractClassShouldBeAbstract • SingularField • ReturnEmptyArrayRatherThanNull • AbstractClassWithoutAnyMethod • TooFewBranchesForASwitchStatement	Critical Violation
Source Code Logging Rules (Only the Apache Standards)	• UseCorrectExceptionLogging • ProperLogger • GuardDebugLogging • MoreThanOneLogger • LoggerIsNotStaticFinal • SystemPrintln • AvoidPrintStackTrace	Warning Violation

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 7, Issue 6, June 2017)

III. Study On The Popular Frameworks

In this section, the work measures the benefits of the popular defect tracking tools and frameworks in the light of severity and true measures of detects.

CheckStyle: CheckStyle [9] is a static code evaluator to checkjava codes against the java coding standards. The CheckStyle is originated as one of the most popular technique in 2001 and developed by Oliver Burn. This framework is majorly used by the developer to test java source codes for quality, reusability and reduction factors of the development cost.

One limitation to this framework is to be highlighted and understood here as the checking of the source code is done based on the phenomena of code presentation or code understanding rather than the code logic. Nevertheless, many programmers may have own way of defining the contents to make the source code dynamics stable for multi system integrations.

The CheckStyle deploys a wide variety of modules to check the code and each style can raise the violation notifications as notification, warnings and errors.

PMD: Another notable milestone in the domain of code analysis is PMD. This is a rule driven correctness verification tool for the source code. The framework comes with many pre-defined rule sets and also gives the flexibility to define custom rules, which is very important in order to accomplish the organization standards for each individual.

PMD does not verify the code for functional defects, rather verifies for coding standards [10].

The major reason for the popularity of PMD is to verify the codes for possible bugs like Empty try/catch/finally/switch blocks, unused code variable or unreachable codes. Also the PMD enables the developers to find "Overcomplicated expressions like Unnecessary if statements, for loops that could be while loops, Suboptimal code like Wasteful String/StringBuffer usage, Classes with high Cyclomatic Complexity measurements, Duplicate code like Copied/pasted code can mean copied/pasted bugs and decreases maintainability".

Checker Framework: The Checker framework provides the advantages of adding pluggable type modules to the Java language for "Backward-Compatible way" [11]. The build in error checkers of Java, detects many errors. However, the detections are not enough for making the source code completely error free.

The most popular way of using the Checker Framework is to define new type qualifiers and relevant semantics. Further, the added compiler of the language applies the semantics. Thus, it enables programmers to apply those plugin and detect the errors.

The Checker Framework is not only applicable to programmers, rather also the system designers to verify the source code before deployment. The errors are reported during compilation time.

IV. Results And Discussion

This work evaluates five difference java open source codes to understand the defect detection measures from the three prime frameworks / tools as discussed prior in this work.

Firstly, the PMD framework is used for testing and results are been observed [Table–2].

Table–2:
Severity Based Defect Detections using PMD

Test Source Code Set	Number of Lines of Code	Blocker Violation Detected	Critical Violation	Urgent Violation	Important Violation	Warning Violation	Total Defects Detected
BugInstance.java	2807	3	2	907	4	37	953
OpcodeStack.java	3609	2	34	1240	14	60	1350
SortedBugCollection.java	1395	7	2	503	8	27	547
PluginLoader.java	1608	6	22	753	1	53	835
FindBugs2.java	1220	0	24	407	2	51	484

Thus is it natural to understand that the PMD framework detects the defects based on the severity of the defects and reduces the false positive defect reports.

Secondly, this work evaluates the defect tracking results by the Check Style framework [Table–3].

Table–3:
Defect Detections using Check Style

Test Source Code Set	Number of Lines of Code	Total Defects Detected
BugInstance.java	2807	4
OpcodeStack.java	3609	86
SortedBugCollection.java	1395	3
PluginLoader.java	1608	3
FindBugs2.java	1220	4

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 7, Issue 6, June 2017)

Hence it is natural to understand that the Checker Style does not detect the defects based on severity. Nevertheless the framework reduces the number of defects. However, the ignorance of the severity, cannot always guarantee to avoid the false positive defects.

Lastly, this work evaluates the reports by the Checker Framework [Table–4].

Table – 4:
Defect Detections using Checker Framework

Test Source Code Set	Number of Lines of Code	Total Defects Detected
BugInstance.java	2807	49
OpcodeStack.java	3609	15
SortedBugCollection.java	1395	18
PluginLoader.java	1608	7
FindBugs2.java	1220	11

Thus it is observable that the Checker Framework also does not reply on the severity-based detection, however the numbers of defects detected are moderate. Hence, the detection of the defects cannot be guaranteed to improve the performance of the source code.

This work also analyses the results of these three frameworks or tools in order to understand the number of defects detected [Table–5].

Table–5:
Defect Detections using PMD, Check Style and Checker Framework

Test Source Code Set	Number of Lines of Code	Total Defects Detected		
		PMD	Check Style	Checker Framework
BugInstance.java	2807	953	4	49
OpcodeStack.java	3609	1350	86	15
SortedBugCollection.java	1395	547	3	18
PluginLoader.java	1608	835	3	7
FindBugs2.java	1220	484	4	11

Hence, the results are clear indication that the severity based defect detections are the prime options as the number of defects can be detected and as well as based on the severity the number of defects can also be reduced.

The comparative analysis is also been observed graphically [Figure–1].

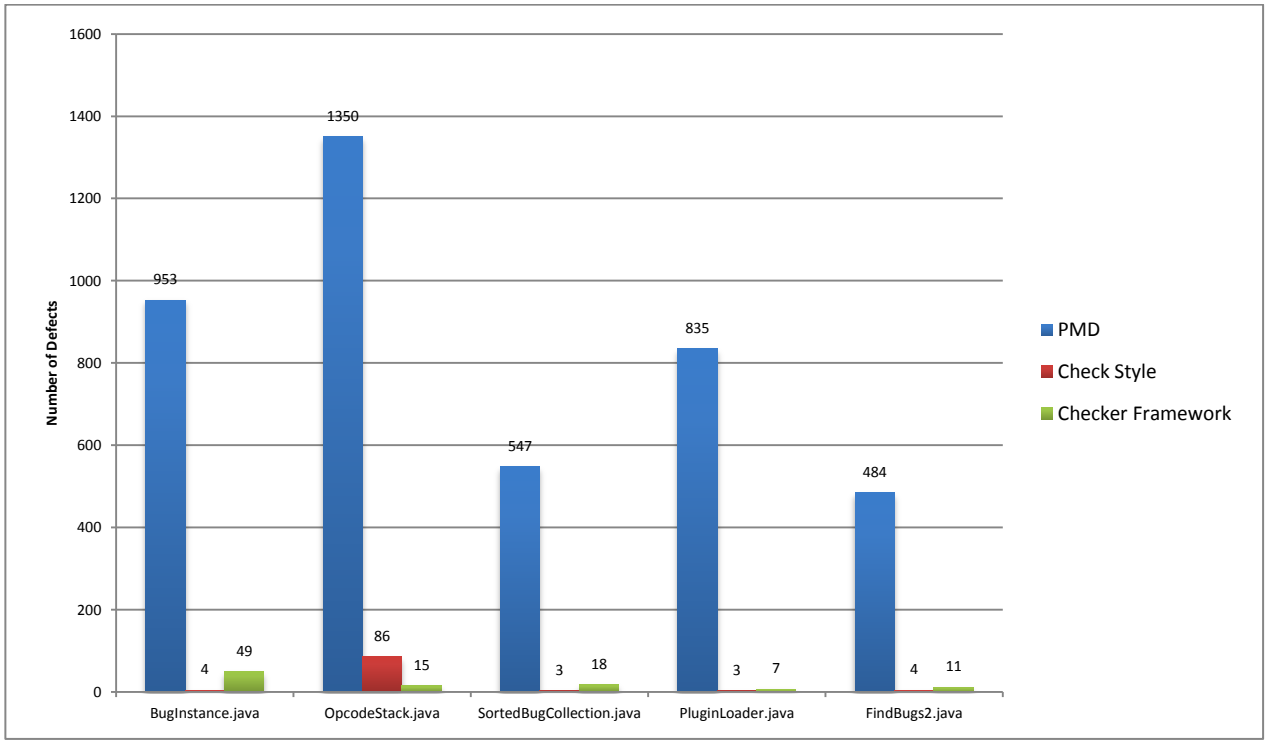

Figure-1: Comparative Analysis of Defect Detection

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 7, Issue 6, June 2017)

V. CONCLUSION

In order to establish the concept of severity based detection of the source code defects, this work analysis five different open source codes using three different defect detection frameworks / tools. The frameworks used in this work for the evaluations are different in nature and uses different approaches to detect the defects. This study proves the improvement and flexibility of using the severity-based measures to detect the defects in order to reduce the development time frame by reducing the false positive reports of defects. This analysis will be helpful and useful for further analysis of the source code defect detection following the same analogy.

REFERENCES

[1] S. Arlt, P. R□ummer, and M. Sch□af. Joogie: From java through jimple to boogie. In SOAP, 2013.

[2] S. Arlt and M. Sch□af. Joogie: Infeasible code detection for java. In CAV, 2012.

[3] M. Janota, R. Grigore, and M. Moskal. Reachability analysis for annotated code. In SAVCBS, 2007.

[4] A. Tomb and C. Flanagan. Detecting inconsistencies via universal reachability analysis. In ISSTA, 2012.

[5] X. Wang, N. Zeldovich, M. F. Kaashoek, and A. Solar-Lezama. Towards optimization-safe systems: analyzing the impact of undetained behavior. In SOSP, 2013.

[6] D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf. Bugs as deviant behavior: A general approach to inferring errors in systems code. In SOSP, 2001.

[7] D. Hovemeyer and W. Pugh. Finding more null pointer bugs, but not too many. In PASTE, 2007.

[8] T. McCarthy, P. R□ummer, and M. Sch□af. Bixie: Finding and understanding inconsistent code. http://www.csl.sri.com/bixie-ws/, 2015.

[9] "Checkstyle Home Page". 2010. Retrieved 2010-11-02.

[10] Rutar, Almazan, Foster (2004), "A Comparison of Bug Finding Tools for Java". ISSRE '04 Proceedings of the 15th International Symposium on Software Reliability Engineering, IEEE

[11] https://checkerframework.org