IJETST- Vol.||04||Issue||02||Pages 4974-4980||February||ISSN 2348-9480 2017

Open access Journal International Journal of Emerging Trends in Science and Technology
IC Value: 76.89 (Index Copernicus) Impact Factor: 4.219 DOI: https://dx.doi.org/10.18535/ijetst/v4i2.03

Categorizing and Analyzing the Impact of Bugs in Open Source Software

Authors
Manpreet Kaur[1], Hardeep Singh[2]
[1]Department of Computer Science and Engineering, Guru Nanak Dev University, Amritsar, India
Email: *manpreet.buttar.csb@gmail.com*
[2]Department of Computer Science and Engineering, Guru Nanak Dev University, Amritsar, India
Email: *hardeep_gndu@rediffmail.com*

ABSTRACT

As open source software is gaining popularity, it becomes necessary to do the modifications in the code. Modifications could be enhancing the functionalities, or bug fixing. Therefore in this paper, we have used Find Bugs plugin in the Eclipse environment to categorize the different types of bugs in Open Source Software to study the bug dynamics. We have used various versions of JFreechart software to track and analyse different types of bugs. JFreechart is open source software. After this CodePro AnalytiX plugin is used in Eclipse environment to calculate the complexity metrics. Complexity is calculated on each version of JFreechart to study the reasons for increase or decrease in the number of bugs in each version. The results have shown that increase or decrease in number of bugs is closely related to average Cyclomatic complexity.

Keywords: *Open source software, categorization of bugs, License, Lines of Code, Efferent Couplings, Average Cyclomatic complexity.*

INTRODUCTION

Open Source means something which can be changed as its design is available publicly. Open source software is software in which code is accessible for alteration or improvement by different users. Code which is also called Source code is available to the developers for enhancing the program by inserting new and improved functionalities to it or repairing the errors that are not producing correct outputs. Open source software, products, or projects are those which include open exchange, mutual contribution, quick prototyping, transparency, and community expansion. Therefore it is software in which copyright holder gives the code to other users who want to use the code, or copy it, change it, or share it. But users have to agree to the conditions of a license while using open source projects. Moreover these open source licenses support group effort and contribution as they permit users to do alterations to the code and include those modifications into their projects. Mainly open source licenses confirm that anybody who modifies and shares the code with other users should also distribute program's code free of cost. In case, if they do not providing the code free of cost, they might be infringing the conditions of open source licenses. According to Open Source Initiative, "open source doesn't mean that code is available." It is a way with which anybody must be capable to change the code to fulfil his/her requirements, and no one could stop others from doing this. Moreover there is general misunderstanding about open source that developers can charge fee for open source projects for producing them. However, as the majority of open source licenses involve providing their code while selling these software to other users, and several open source project developers charges money for software facilities and support instead of charging fee for the software itself and they find it more profitable. In this way, software is accessible free of cost and they earn profits by serving others to install, exploit, and troubleshoot it. Bugs are the errors that occur in the software that leads to incorrect outputs. Therefore in case of open source software, as the code is

available and there is high probability that many programmers can access the code and it is sometimes possible that inexperienced users make changes and may submit the buggy patches to bug repositories. Therefore in case of open source software, huge number of bugs can lead to more serious errors that hinder the functionality of open source software. Bugs can also lead to more risky issues which might be very difficult for developers to handle.

Categorization of Bugs in Open Source Software

We have categorized and analyzed 41 different types of bugs across 37 different versions of JFreechart software. But for paper point of view we are explaining 20 different types of bugs which are given in table 1:

Table 1. Classification of Bugs present in different versions of JFreechart Software

BUG TYPE	DESCRIPTION
Scariest	Highest ranked bugs, more vulnerable. These are kind of logic errors which produce unexpected output. Ranking (1-4).
Scary	High ranked bugs and they are semantic type of errors. Comparatively less dangerous than scariest and Ranking within the range of (5-9).
Troubling	Bugs are problematic in software but can be resolved as they are either syntactic or semantic errors; ranking given is (10-14).
Of Concern	These bugs are less dangerous and can be resolved easily. These are syntax or semantic types of errors. The ranking given to them is (15-20).
Normal Confidence	Used to find warnings with a particular bug confidence. The value property must be an integer value: **1 for high-confidence warnings, 2 for normal-confidence warnings, or 3 for low-confidence warnings.**
High Confidence	Used to find warnings with a particular bug confidence. The value property must be an integer value: **1 for high-confidence warnings, 2 for normal-confidence warnings, or 3 for low-confidence warnings.**
Call to equals() comparing different types	Calls equals(Object) on two references of separate class types and examination recommends that they are two objects of separate classes at runtime. Besides this, analysis of the equals methods that will be called recommends that either call always return false, or the equals method is not be symmetric (which is a property required by the contract for equals in class Object).[**Rank**: Scariest (1), **confidence**: High **Pattern**: EC_UNRELATED_TYPES **Type**: EC, **Category**: CORRECTNESS]
Self assignment of field	Method consists of a self assignment of a field; like int x; public void foo() { x = x; }.These assignments are useless, and may specify a logic error.[**Rank**: Scariest (1), **confidence**: High **Pattern**: SA_FIELD_SELF_ASSIGNMENT **Type**: SA, **Category**: CORRECTNESS]
Uninitialized read of field in constructor	Such constructor examines a field whose value is not assigned yet. This generally happens when programmer by mistake uses the field instead of one of the constructor's arguments. [**Rank**: Scariest(1), **confidence**: High **Pattern**: UR_UNINIT_READ **Type**: UR, **Category**: CORRECTNESS]
Doomed test for equality to NaN	Code confirms whether floating point value is equal to the particular Not A Number value (e.g., if (x == Double.NaN)). But due to unique meaning of NaN, no value is equal to Nan, including NaN. Thus, x == Double.NaN is always calculated as false. For verifying whether a value present in x is the particular Not A Number value, apply Double.isNaN(x) (or Float.isNaN(x) if x is floating point value). **Rank**: Scary (6), **confidence**: High **Pattern**: FE_TEST_IF_EQUAL_TO_NOT_A_NUMBER **Type**: FE, **Category**: CORRECTNESS]
Impossible cast	Throws a ClassCastException. FindBugs tool follows type information from instanceof checks, and also utilizes more accurate information about the kinds of values returned from methods and loaded from fields and therefore uses this information to decide that a cast will throw an exception at execution time.[**Rank**: Scary (9), **confidence**: High **Pattern**:BC_IMPOSSIBLE_CAST **Type**: BC, **Category**: CORRECTNESS]
Method call passes null for non-null parameter	Method call passes a null value for a non-null method parameter. This either means that the variable is taken as a variable that must be non-null all the time, or investigation has shown that it will be dereferenced all the time. [**Rank**: Scary (8), **confidence**: Normal **Pattern**:NP_NULL_PARAM_DEREF **Type**: NP, **Category**: CORRECTNESS]
Call to static DateFormat	DateFormats are not secure for multithreaded purpose. The detector finds a call to a parameter of DateFormat which is achieved by a static field. [**Rank**: Scary (8), **confidence**: Normal **Pattern**:STCAL_INVOKE_ON_STATIC_DATE_FORMAT_INSTANCE **Type**: STCAL, **Category**: MT_CORRECTNESS]
Read of unwritten field	Dereferencing a field which does not ever have non-null value written to it. Dereferencing the field value will produce a null pointer exception. [**Rank**: Scary (8), **confidence**: Normal **Pattern**: NP_UNWRITTEN_FIELD **Type**: NP, **Category**: CORRECTNESS]
Class defines equals() and uses Object.hashCode()	Class overrides the method equals(Object), but does not override hashCode(), and inherits the implementation of hashCode() from java.lang.Object and it returns the unique hash code, an arbitrary value assigned to the object by the VM). Therefore, the class is violating the invariant that equal objects must have equal hashcodes. [**Rank**: Troubling (14), **confidence**: High **Pattern**:HE_EQUALS_USE_HASHCODE **Type**: HE, **Category**: BAD_PRACTICE]
Method might ignore exception	Method may overlook an exception. Basically, exceptions must be handled or they must not be included inside the method.[**Rank**: Troubling (14), **confidence**: High **Pattern**: DE_MIGHT_IGNORE **Type**: DE, **Category**: BAD_PRACTICE]
Call to static Calendar	Calendars are inherently dangerous for multithreaded usage. The detector has found a call to an instance of Calendar which has been achieved through a static field which is itself doubtful. [**Rank**: Troubling (14), **confidence**: Normal **Pattern**: STCAL_INVOKE_ON_STATIC_CALENDAR_INSTANCE **Type**: STCAL, **Category**: MT_CORRECTNESS (Multithreaded correctness)]
Unwritten field	Field is not written ever. Every reads of this field will return the default value. [**Rank**: Troubling (12), **confidence**: Normal **Pattern**:UWF_UNWRITTEN_FIELD **Type**: UwF, **Category**: CORRECTNESS]

Dead store to local variable	Assigns a value to a local variable, but the value is not read or used in any succeeding instruction. This generally point towards an error, as the value calculated is never used. [**Rank**: Of Concern (15), **confidence**: High **Pattern**: DLS_DEAD_LOCAL_STORE **Type**: DLS, **Category**: STYLE (Dodgy code)]
Test for floating point equality	Compares two floating point values for parity. As floating point computation may include rounding of digits, computed values of float and double may be inaccurate. Therefore, values that need accurate precision, like financial values, use a fixed-precision type like BigDecimal and values that do not need precision, use comparing for equality within some range, like: if (Math.abs(x - y) < .0000001). [**Rank**: Of Concern (15), **confidence**: High **Pattern**: FE_FLOATING_POINT_EQUALITY **Type**: FE, **Category**: STYLE (Dodgy code)]

RELATED WORK

Sascha Just et al. 2008 [1] have conducted a survey on three major bug tacking systems namely APACHE, MOZILLA, AND ECLIPSE in order to find the information requirements and problem faced by developers in bug reporting system. N. Jalbert et al. 2008 [2] have proposed a method that automatically categorizes redundant bug reports when they arrive to save developer time. Thomas Zimmermann et al. 2009 [3] have addressed the problem of inadequately designed bug tracking systems in which information about bugs filtered out after numerous iterations of messages between user and developers. G. Abaee et al. 2010 [4] have compared the features as well as limitations of four bug tracking tools. V.B Singh et al. 2011[5] have done the comparative study of various bug tracking tools. A. Raza et al. 2012 [6] have proposed the model to establish the relationship between usability bugs in Open source projects and online public conferences. Akhilesh Babu Kolluri et al. 2012 [7] have presented a model which is efficient in tracking the bugs by collecting the important information from users and useful in resolving the bugs immediately. S Lal et al. 2012 [8] have presented which provides the comparison between different kinds of bug reports on the basis of metrics like statistics on close-time, number of comments, entropy among reporters, entropy across comp-onent, continuity and debugging efficiency performance characteristics. Sean Banerjee et al. 2012 [9] have presented the methodology named Factor LCS that uses common sequence matching for finding the duplicate bug reports. Swati Sen et al. 2013 [10] have suggested that in open source development process bug tracking system are most significant for tracking the bugs.

EXPERIMENTAL SETUP

Figure 1 shows the flowchart for proposed methodology which is explained below:

Step1: Initially the different versions of JFreechart softwareare downloaded from Sourceforge.net. JFreechart is open source software.

Step2: In second step Eclipse tool is used and JFreechart versions are imported in it.

Step3: In this step FindBugs plugin is downloaded in the Eclipse environment for tracking the bugs in various versions of JFreechart software.

Step4: In this step FindBugs plugin is used for tracking the bugs on every version of JFreechart software.

Step5: In this step bugs tracked with the help of FindBugs plugin are evaluated to study the different types of bugs present in JFreechart software.

Step6: In this step Codepro AnalytiX plugin is downloaded in Eclipse enviroment to evaluate different complexity metrics on each version of JFreechart software.

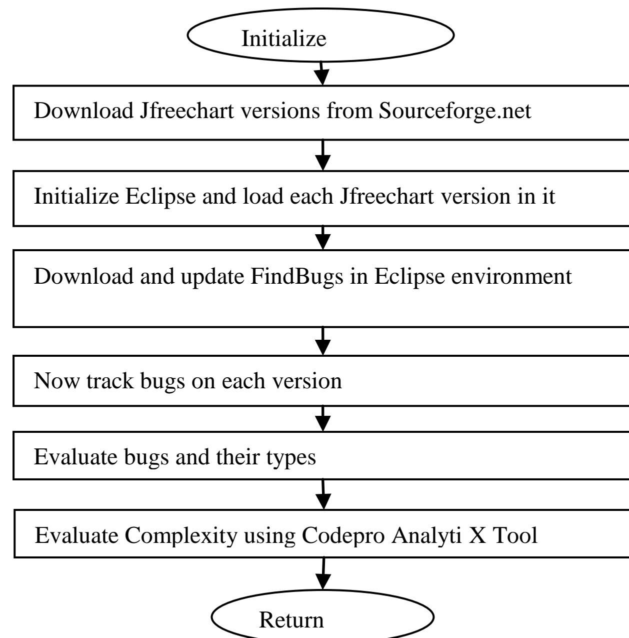

Figure 1. Flowchart for Proposed Methodology

RESULTS AND DISCUSSION

Correlation Analysis

Table 2 shows the Pearson Correlation Analysis. We have used Pearson Correlation coefficient for measuring the correlation between multiple variables like total bugs, Scariest, Scary, Troubling, Of Concern type of bugs, loc, number of methods, number of constructors and Efferent coupling. Strong correlation is shown by values close to 1 and -1 in which 1 indicates perfect correlation and -1 indicates inverse correlation while values close to 0 show no correlation. Pearson correlation is significant at level 0.01. Initially, total bugs are showing weakest correlation with Scariest type of bugs as total bugs are having correlation value 1 whereas Scariest has correlation value -.528. Therefore total bugs have inverse relation with Scariest. As total bugs are increasing number of Scariest type of bugs are decreasing. Moreover Scariest types of bugs are logical type of errors which produces unexpected output and are most difficult to locate and fix. Therefore developers try to keep the low false positive rate of such kinds of errors and thus their number remain small as these errors are not frequently occurring errors and therefore they are less contributing to total bugs. Total bugs are having strongest relation with efferent coupling. Efferent coupling is an indicator of package dependency on external packages. Therefore more the class is coupled with other classes, there will be a more risk that number of bugs will increase as whenever the code changes in one class, we have to change the code in other classes also and if changes are not done properly number of bugs will increase.

Secondly, Scariest types of bugs are showing weakest correlation with Efferent coupling as Scariest types of bugs are having value 1 and efferent coupling is having value -.680. As a result Scariest types of bugs are having inverse relation with efferent coupling. As the value of Scariest types of bugs are less contributing in total bugs because these errors are very small in number, therefore efferent coupling value is increasing due to inverse relation with Scariest types of bugs. Scariest types of bugs (with value 1) are showing strongest correlation with average Cyclomatic Complexity (with value .018). This is because if logical types of errors are increasing, they definitely increase the overall Cyclomatic complexity.

Scary types of bugs have weakest correlation with Scariest types of bugs because Scary are semantic types of errors which occur due to improper use of program statements and they comparatively easy to locate as whenever these errors occur error message will be shown whereas Scariest are logical types of errors which get executed without any errors but produce unexpected outcomes. So they have weakest correlation with Scary types of bugs. Scary types of bugs are showing strongest correlation with total bugs because these are semantic errors which are frequently occurring errors like string not declared in scope or a word is not declared in scope.

Troubling bugs are showing weakest correlation with Scariest bugs as troubling are Compile time errors or syntax errors and semantic errors whereas Scariest are logical errors. Troubling is showing strongest correlation with total bugs because these are frequently occurring errors like equals check for incompatible operand or missing the rules of programming language like missing semicolon etc.

Of Concern is showing weakest correlation with Scariest types of bugs as Of Concern is basically semantic types of errors. Of Concern is showing strongest correlation with Scary bugs because both are semantic types of errors.

Lines of code have weakest correlation with Scariest types of bugs because Scariest are small in number and correcting these bugs do not have significant effect on Lines of Code. Lines of code have perfect 1-1 relation with number of methods as number of methods are increasing, LOC will also increase.

Number of Methods is showing weakest correlation with Scariest types of bugs because Scariest are very small in number and they have inverse relation with number of methods therefore as the number of methods are increasing the value of Scariest types of bugs is decreasing. This is because increase in number of methods improves the readability of code

and decreases the logic errors. Number of methods is showing perfect 1-1 relation with LOC as number of methods are increasing, LOC will automatically increase.

Number of Constructors has weakest correlation with Scariest types of bugs because they have similar behavior as methods and improve the readability of code and decrease the logic errors. Number of Constructors has strongest correlation with LOC as number of constructors are increasing, LOC will also increase.

Efferent Coupling is having weakest correlation with Scariest types of bugs because Scariest types of bugs are less contributing in total bugs because these errors are very small in number, therefore efferent coupling value is increasing due to inverse relation with Scariest types of bugs. Efferent Coupling is showing strongest correlation with total bugs because more the class is coupled with other classes, there will be a more risk that number of bugs will increase as whenever the code changes in one class, we have to change the code in other classes also and if changes are not done properly number of bugs will increase.

Average Cyclomatic Complexity is showing weakest correlation with Scariest types of bugs as these errors are small in number increase in these types of errors do not have significance on overall complexity of code. Average Cylcomatic Complexity is showing strongest correlation with Of Concern types of bugs because syntax and semantic types of errors are more frequently occurring errors and increase in syntax and semantic errors increase the overall complexity of code.

Table 2. Pearson Correlation Table

Correlations											
		Total Bugs	Scariest	Scary	Troubling	Of Concern	Lines of Code	No of Methods	Number of Constructors	Efferent Coupling	Average Cyclomatic Complexity
Total Bugs	Pearson Correlation	1	-.528**	.927**	.979**	.759**	.815**	.819**	.773**	.935**	.208
	Sig. (2-tailed)		.001	.000	.000	.000	.000	.000	.000	.000	.216
	N	37	37	37	37	37	37	37	37	37	37
Scariest	Pearson Correlation	-.528**	1	-.515**	-.521**	-.274	-.618**	-.623**	-.583**	-.680**	.018
	Sig. (2-tailed)	.001		.001	.001	.101	.000	.000	.000	.000	.917
	N	37	37	37	37	37	37	37	37	37	37
Scary	Pearson Correlation	.927**	-.515**	1	.834**	.808**	.794**	.802**	.702**	.869**	.089
	Sig. (2-tailed)	.000	.001		.000	.000	.000	.000	.000	.000	.599
	N	37	37	37	37	37	37	37	37	37	37
Troubling	Pearson Correlation	.979**	-.521**	.834**	1	.668**	.783**	.784**	.771**	.920**	.245
	Sig. (2-tailed)	.000	.001	.000		.000	.000	.000	.000	.000	.144
	N	37	37	37	37	37	37	37	37	37	37
Of Concern	Pearson Correlation	.759**	-.274	.808**	.668**	1	.612**	.618**	.519**	.662**	.321
	Sig. (2-tailed)	.000	.101	.000	.000		.000	.000	.001	.000	.053
	N	37	37	37	37	37	37	37	37	37	37
Lines of Code	Pearson Correlation	.815**	-.618**	.794**	.783**	.612**	1	1.000**	.978**	.883**	.207
	Sig. (2-tailed)	.000	.000	.000	.000	.000		.000	.000	.000	.218
	N	37	37	37	37	37	37	37	37	37	37
No of Methods	Pearson Correlation	.819**	-.623**	.802**	.784**	.618**	1.000**	1	.973**	.885**	.190
	Sig. (2-tailed)	.000	.000	.000	.000	.000	.000		.000	.000	.259
	N	37	37	37	37	37	37	37	37	37	37
Number of Constructors	Pearson Correlation	.773**	-.583**	.702**	.771**	.519**	.978**	.973**	1	.840**	.248
	Sig. (2-tailed)	.000	.000	.000	.000	.001	.000	.000		.000	.139
	N	37	37	37	37	37	37	37	37	37	37
Efferent Coupling	Pearson Correlation	.935**	-.680**	.869**	.920**	.662**	.883**	.885**	.840**	1	.247
	Sig. (2-tailed)	.000	.000	.000	.000	.000	.000	.000	.000		.141
	N	37	37	37	37	37	37	37	37	37	37
Average Cyclomatic Complexity	Pearson Correlation	.208	.018	.089	.245	.321	.207	.190	.248	.247	1
	Sig. (2-tailed)	.216	.917	.599	.144	.053	.218	.259	.139	.141	
	N	37	37	37	37	37	37	37	37	37	37
**.Correlation is significant at 0.01 level (2-tailed)											

DISCUSSION

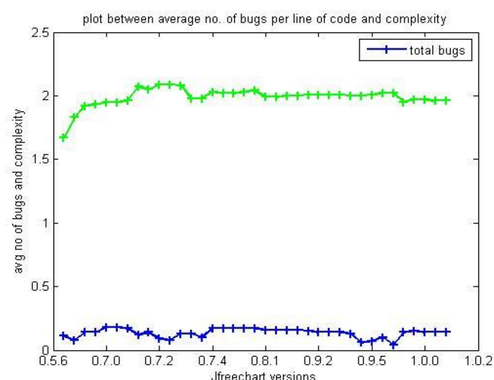

Figure 2. Correlation between Average number of bugs and Average Cyclomatic Complexity

The graph shown in Figure 2 is a plot between average number of bugs per lines of code and the average cyclomatic complexity. The green line shows the values for average Cyclomatic complexity and blue line shows average number of bugs per lines of code. Initially as the numbers of versions are increasing, the complexity is also increasing as user is demanding better functionality in new versions. Therefore with the increase in complexity, numbers of bugs are decreasing because previous bugs are removed from earlier versions and better functionality is provided to newer versions. As from version 0.7.0 to version 0.7.1 complexity and number of bugs remain the same. But from version 0.7.2 to 0.7.4 and version 0.9.5 to 0.9.6 number of bugs are decreasing as well as complexity is also decreasing because sometime new version is demanded, therefore reduction is done in the number of bugs as well as complexity because there may some non-usable functions with high complexity and more number of bugs present in earlier versions and they are removed to decrease the complexity as well as number of bugs.

CONCLUSIONS

In this paper we have tracked and analyzed different types of bugs present in various versions of JFreechart We have used Find Bugs plugin in Eclipse environment while categorizing the bugs. In order to study the increasing and decreasing nature of bugs in JFreechart software, complexity is calculated using CodePro AnalytiX plugin in Eclipse environment. The empirical analysis shows that variation in the number of bugs is closely related to average Cyclomatic complexity. So in future, we can consider more number of versions of JFreechart software or can take other open source software for tracking and analyzing more types of bugs. Moreover besides Cyclomatic complexity we can also consider more parameters for finding the relation with number of bugs.

REFERENCES

1. Just, Sascha, Rahul Premraj, and Thomas Zimmermann. "Towards the next generation of bug tracking systems." In Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on, pp. 82-85. IEEE, 2008.
2. Jalbert, Nicholas, and Westley Weimer. "Automated duplicate detection for bug tracking systems." In Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on, pp. 52-61. IEEE, 2008.
3. Zimmermann, Thomas, Rahul Premraj, Jonathan Sillito, and Silvia Breu. "Impro-ving bug tracking systems." In Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on, pp. 247-250. IEEE, 2009.
4. Abaee, Golnoosh, and D. S. Guru. "Enhancement of Bug Tracking Tools; the Debugger." In Software Technology and Engineering (ICSTE), 2010 2nd Internat-ional Conference on, vol. 1, pp. V1-165. IEEE, 2010.
5. Singh, V. B., and Krishna Kumar Chaturvedi. "Bug tracking and reliability assessment system (btras)." International Journal of Software Engineering and Its Applications 5, no. 4 (2011): 1-14.
6. Raza, Arif, Luiz Fernando Capretz, and Faheem Ahmed. "Usability bugs in open-source software and online forums." IET software 6, no. 3 (2012): 226-230.

7. Akhilesh Babu Kolluri, K. Tameezuddin, Kalpana Gudikandula “Effective Bug Tracking Systems: Theories and Implementation” IOSR Journal of Computer Engineering (IOSRJCE) ISSN: 22780661 Volume 4, Issue 6(Sep-Oct. 2012), PP 31-36.
8. Lal, Sangeeta, and Ashish Sureka. "Comparison of seven bug report types: A case-study of google chrome browser project." In Software Engineering Conference (APSEC), 2012 19th Asia-Pacific, vol. 1, pp. 517-526. IEEE, 2012.
9. Banerjee, Sean, Bojan Cukic, and Donald Adjeroh. "Automated duplicate bug report classification using subsequence matching." In High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on, pp. 74-81. IEEE, 2012.
10. Swati Sen, Anita Ganpati “Proposed Framework for Bug Tracking System in OSS Domain” Volume 3, Issue 8, August 2013ISSN: 2277 128XInternational Journal of Advanced Research in Computer Science and Software Engineering.