Defect Collection and Reporting System for Git based Open Source Software

Ruchika Malhotra[#1], Nakul Pritam[*2], Kanishk Nagpal[#3], Prakarsh Upmanyu[*4]

Department of Computer Engineering, Delhi Technological University
Bawana Road, Shahbad, Daulatpur, Delhi-110042, India

[1]ruchikamalhotra2004@yahoo.com
[2]nakul.pritam@gmail.com
[3]kanishk0001@gmail.com
[4]prakarsh.upmanyu23@gmail.com

***Abstract*— This paper describes Defect Collection and Reporting System which is an automated tool that aids in generating various reports to provide useful information regarding the defects which were present in a given version of a Git Version Control System based Open Source Software and were fixed in the subsequent version. The generated reports contain information such as the total number of defects, class wise and the corresponding values of different OO metrics for each class in the source code. Such kind of defect data can be readily used for defect prediction hypothesis pertaining to Git based Open Source Software. Some potential fields of application could be analysis and validation of the effect of a given metric suite on fault proneness; to predict fault proneness models for Defect Prediction, etc. In addition to the core functionalities and overall functioning of the above stated tool, this paper also gives an overview of some of the functionalities which might be useful for the end user, namely- cloning of Git Repositories, and self-logging data of the tool.**

***Keywords*— Open Source, Defect Collection, Metrics, Git, Cloning, Self-logging.**

I. INTRODUCTION

Today's world is characterized by rapidly evolving technologies and the role of Open Source Software has increased manifold. Open Source Software has truly become the most prominent feature of Open Source Development. As compared to Closed Source or Proprietary Software, Open Source Software is characterized by lower costs, better security, no vendor or copyright issues and higher quality. Also, Open Source Software is developed with a different approach and methodology. Users are generally treated as co-developers and development is accomplished through collaborative effort of programmers. The working product is released much earlier and is usually followed by various versions which offer fewer functionalities but more stability. High Modularization and Dynamic Decision Making are commonly employed. Version Control Systems (such as Git) for Open Source Software ensure much easier and structured maintenance, especially for the changes incurred. Owing to these reasons, Open Source Software is increasingly being adopted by individuals and many organizations across the globe. Perhaps the best example that clearly depicts the capabilities of an Open Source Software is Android OS. It is a Linux-based Operating System which has been designed primarily for touch screen devices, including smart phones and tablets. Unveiled in 2007, Android today has become the most widely used operating system for mobile devices throughout the world, completely outwitting Proprietary Software for mobile devices, such as Symbian OS. Android's source code is available at Google's Git Repository; and a vast developer community caters to the development of applications ("apps") that extend the functionality of devices.

Open source software repositories provide rich data that can be extracted and employed in various empirical studies. Primary focus is on defect and change data which can be obtained from version control repositories of open source software systems. For instance, previous studies have shown that defect data collected from open source projects can be used in research areas such as defect prediction [1]. Some commonly traversed areas of defect prediction include Analysis and Validation of the effect of given metric suite, (such as CKJM and QMOOD) on fault proneness [2]; and evaluating the performance of fault proneness models, such as Artificial Neural Networks [3]. However, an important question here is that what kind of software repositories may be potentially mined for extracting such kind of data. After a rigorous study and analysis, we came across software repositories which employ 'Git' as the Version Control System and these can be appropriate candidates for evaluation and selection so as to be employed in research and empirical studies. The reasons for selecting Git based Software systems are:

- Git is the most popular distributed version control system and the defect and change data can be extracted in relatively easier ways through the change logs maintained by Git.
- A large number of open source software systems are maintained through Git, including Google's Android OS and a vast range of Apache's Software applications.

But at present, most of the empirical data collection process is carried out manually. There exists no mechanism which can employ an automated process to collect the defect data for Git based Open Source Software, and provide information useful enough to be used in research areas mentioned above.

978-1-4799-4674-7/14/$31.00 ©2014 IEEE

Therefore, a system is required which can efficiently collect defect data for a Git-based open source software, which might be used in the above mentioned research areas and other applications.

Such kind of a system is expected to perform the following operations: First, obtain the 'git change logs' for a given software and process them to obtain the defects which were reported in a specific version of that software and were fixed in the subsequent version. Then, the system should extract vital defect information such as unique bug or defect identifier and the defect description, if provided. Next, we require the association of defects to their corresponding source files (java code files, or class files contained in the source code). Then the system should perform the computation of total number of defects fixed corresponding to each class, i.e., the number of defects associated with that class. Finally, the corresponding values of different metrics should be calculated by the system for each source code file for previous version of the open source software.

In this work, we have developed a tool named Defect Collection and Reporting System (here onwards, referred to as DCRS), which incorporates each and every functionality stated above and consequently generates various reports that present the collected data in a more processed, meaningful and useful form.

Rest of the paper is organized as follows: Overall functioning of the DCRS for collection of defect data and report generation is summarized in Section II. Section III describes different types of reports that are generated by the DCRS from the collected defect data. Section IV summarizes some additional functionalities of the DCRS. Demonstration of the DCRS, including data source, preliminary analysis and results, is presented in Section V. Section VI gives an overview of some potential applications of the DCRS. Section VII presents the conclusions and future work.

II. WORKING MECHANISM

The DCRS sequentially carries out a series of well-defined operations to collect and process the Defect Data of Git based Open Source Software and then finally generate various types of reports for the same. These operations are described as follows:

First, the DCRS processes an open source software's source code (obtained from Git repository) for each of the two predetermined consecutive versions, to retrieve what are known as Git Change-Logs, or simply change logs. A Change Log provides information about the changes or modifications that have been incurred from time to time in the software's source code. These changes could have been made for various purposes, such as bug or defect fixing, feature addition, refactoring, enhancements, etc. Each and every change incurred, no matter how big (significant) or small, is recorded in the change logs and thus constitutes an individual change record. An individual change record provides various kinds of information such as-

- The Timestamp of committing (i.e., recording) the incurred change with the Git Repository.
- Unique change identifier
- If the change has been made for defect fixing, a unique defect identifier is provided in most of the cases.
- An optional change description or information.
- List of the changed or modified source code files, along with the LOC changes for each of the changed files.

Source code for both of the versions is necessary for processing because a change log provides the change information since the beginning of time (i.e., when the initial release of the software was made available), but we're interested only in the changes which were incurred during the transition from a given previous version to the subsequent one.

Here, we have applied the 'git whatchanged' command with a 'stat' flag to get the change records along with the LOC changes. Fig. 1 depicts the general format of a change log file (along with an individual change record) for Open Source Software hosted at Git Repository, obtained by applying the above command.

<Change Log Record-1>
<Timestamp>
<Change-Id>
<Defect-Id>*
<Change Description>*
<Changed Source File 1> +++…+---…- **
<Changed Source File 2> +++…+---…- **
.
.
.
<Change Log Record-2>
.
.
.
<Change Log Record-N>

* May or may not be present
** + : 1 LOC Insertion; - : 1 LOC Deletion

Fig. 1 General format of Git change log file

In the next operation, Git change logs are further processed, an individual change record at a time, to obtain Defect-Records (i.e. changes which were incurred for defect-fixing, not for other any other reason such as refactoring, enhancement, etc.). Bug or Defect-Ids and the Defect Description, if any, are retrieved from the Defect-Logs. If there's no Defect-Id but only a change description, then we look for various keywords such as 'bug', 'issue', 'defect', etc. in the provided description, and treat the change record as a defect record. In such a case, a default value is stored for the Defect-Id.

The collected defects (with or without any Defect-Id) are finally mapped to Classes in the source code. We have considered only class files or java source code files and other types of files are ignored. The data collected is consequently employed to generate various reports in .csv format. These reports are described later in the paper. Fig. 2 presents the format of Defect information stored by DCRS.

<Defect ID 1> <Defect ID 2> ... <Defect ID N>

<Defect Description>

<Java Source File 2 Changed For Defect Fixing>
<LOC Inserted, Deleted>

<Java Source File 2 Changed For Defect Fixing >
.
.
.
<Java Source File Changed M For Defect Fixing >

Fig. 2: Defect information Stored by DCRS

The entire DCRS system has been implemented in the Java Programming Language (Java SE 1.7). The data for required Metrics for the class files of previous version for a given open source software has been obtained by embedding an open source software in the DCRS, which is available at http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/metric.html and covers a wide range of metrics [4]-[7]. The procedure for defect collection and reporting is presented in Fig. 3.

III. Reports Generated by the DCRS

From the defect data collected as stated in the above section, the tool generates following primary reports:

A. *Defect Details Report*

This report provides descriptive and statistical information corresponding to each defect: The Defect-Id (if any), description (if any), and the source code file(s) (specifically the Java files) which have been changed for fixing that particular defect. The report also contains data for the LOC changes for each class which was modified in order to fix any reported defect.

The fields which are contained in this report, corresponding to each defect, are:

- Java source code file name or simply the class file name
- Unique defect identifier, if any
- Defect description, if any
- LOC inserted and deleted to fix the defect (modification data was not provided in the change logs).

B. *Defect Count and Metrics Report*

This report provides the total number of defects fixed source-file wise (i.e. total number of times a source-file was modified to fix defects), along with total LOC changes, with the data for the following metrics [4]-[7]:

- CKJM Suite (CBO, NPM, RFC, LCOM, DIT, WMC)
- QMOOD Suite (NPM, DAM, MOA, CAM, MFA)
- Martin's Metrics (Ca, Ce)
- Miscellaneous (LOC, LCOM3, AMC, IC, CBM).

More information regarding these metrics may be obtained from [4]-[7]. The fields which are contained in this report are given below:

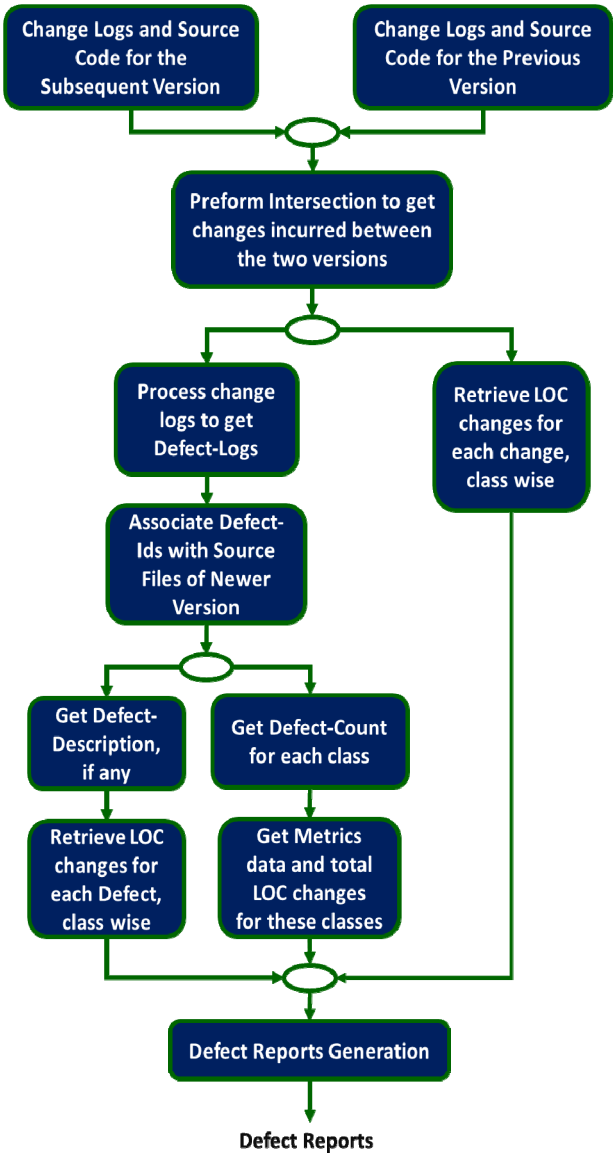

Fig. 3 Flowchart for Defect collection and reporting mechanism

- Java source file name
- Total number of defects mapped to the above class.
- Total LOC inserted and deleted for all of the defects mapped to this class.
- List of metric values, corresponding to the above class.

C. *LOC Changes Report*

This report contains the total number of LOC changes incurred in the source code files, corresponding to each change (irrespective of the purpose of that change, i.e. enhancement, defect-fixing, refactoring, etc.).

The fields which are contained in this report are as follows:

- Java source file name
- Total LOC inserted and deleted for all the changes (including defect fixes, enhancement, etc.) mapped to the above class.

In addition to the above primary reports, the following auxiliary reports are also generated by the DCRS:

D. Newly Added Source Files

This reports contains the list of source code files which were not present in the previous version of the considered open source software, but were introduced or added during the transition to subsequent version. The total number of Defects reported for each class file, and the corresponding LOC changes are also included in the report.

The fields which are contained in this report are as follows:

- Java source file name added in the subsequent version
- Total number of defects for which the above source file has been modified
- Total LOC inserted and deleted for all of the defects mapped to this class file

E. Deleted Source Files

This reports contains the list of source-files which were present in the previous version of an open source software, but have been deleted or removed during the transition to the subsequent version. The total number of Defects reported and corresponding LOC changes are also included, considering the probability of deleting a source file in the latest version after fixing some defects.

The fields which are contained in this report are as follows:

- Java source file name removed from the subsequent version
- Total number of defects for which the above source file has been modified
- Total LOC inserted and deleted for all of the defects mapped to this class file.

F. Consolidated Defect and Change Report

This report is similar to the defect or bug-count report described in B., but the only difference here is that instead of reporting the LOC Changes corresponding to all the defects mapped to a particular class, we report total LOC changes which are incurred in that class. These changes may not only be incurred for issue, defect or bug fixing, but for any other purpose, such as enhancement, refactoring, etc. In other words, this report can be considered as the combination of Bug-Count report and LOC changes report (described in B and C respectively), wherein we report the bug-data from bug-count report, but LOC Changes incurred for each class are reported from the LOC Changes report.

G. Descriptive Statistics Report

This reports gives various statistical measures for each and every metric incorporated in the tool. The various statistical measures for a metric are summarized in Table I.

TABLE I
STATISTICAL MEASURES

Characteristic	Definition
Mean	The arithmetic mean
Median	Middle value when the values for are sorted
Mode	The most frequently occurring value
Harmonic Mean	Reciprocal or inverse of the arithmetic mean
Minimum	The lowest value
Maximum	The highest value
Range	The difference between maximum and minimum values
First Quartile	The value above 25% of the minimum values
Third Quartile	The value above 75% of the minimum values
Interquartile Range	The difference between Third and First Quartiles.
Variance	Arithmetic mean of the square of difference between mean and the individual values
Standard Deviation	Square root of variance
Coefficient of Variation	The ratio of standard deviation to the mean
Variation Ratio	Proportion of values that are not mode
Skewness	It is a measure of the asymmetry of the given values.
Kurtosis	It is a measure of the peakedness of the given values.

IV. Additional Functionalities

Apart from the core functionalities of defect collection and report generation, the DCRS also incorporates some additional features or functionalities which might prove to be useful for the end user. These are grouped into two broad categories-

A. Cloning of Git based Software Repositories

The system provides full support for "cloning" software repositories which employ Git as the Version Control System. "Cloning" implies 'copying and pasting' the entire software repository from a local or remote hosting server to the end-user machine. Software Repositories may be generally classified as [8]:

- *Source Control Repositories,* such as Git and CVS, which record the development track or history of a software project and track all the changes incurred in the project along with the meta-data for each change (timestamp, developer information, etc.).
- *Bug Repositories,* which maintain the history of defect reports and/or feature requests that may be reported by the users and as well as developers of medium to large software. Bugzilla and Jira are popular bug repositories.
- Discussions about various aspects during the lifetime of a software project are maintained in *Archived Repositories* such as Mailing lists, emails, instant messages, etc.
- *Deployment Logs*, that record the execution information of single or multiple deployments of a software application. We can detect execution anomalies by studying the usage patterns across various deployments, and then deviations from these patterns may be recorded
- *Code Repositories*, which archive the source code for a huge number of software projects. Google code and Sourceforge.net are two large source code repositories.

Different types of artefacts may be obtained from a software repository depending upon the type of that repository. For example, source control repositories provide change-logs. Source code may be obtained through code repositories.

In our case, the DCRS supports cloning of Code Repositories which employ Git as the Source Control Repository or Version Control System. The user can thus obtain the source code and as well as Git change logs, which were described earlier. These two artefacts may be then employed for various purposes. For instance, the DCRS employs these artefacts for generating various types of defect reports we have stated earlier. Two crucial areas of application include research related to fault-proneness or defect-prediction, and change proneness. These are discussed later in Section VI.

B. *Self-Logging*

Self-Logging or simply logging may be defined as the process of automatically recording events, data and/or data structures about a tool's execution to provide an audit trail. The recorded information can be employed by developers, testers and support personal for identifying software problems, monitoring live systems and for other purposes such as auditing and post-deployment debugging. Logging process generally involves the transfer of recorded data to monitoring applications and/or writing the recorded information and appropriate messages, if any, to files.

We've also provided the user with the functionality to view the operational logs of the tool. These self-logs are stored as text file(s), indicating the different events, and/or operations which have occurred during the tool's working along with their timestamp. These are ordered by the sequence and hence, the timestamp. The self-log file follows a daily rolling append policy, i.e. logs for a given day are appended to the same file, and a new file is created after every 24 hours. The previous day's file is stored with a name that indicates the time of creation. Java Libraries of LogBack and SLF4J have been employed to implement self-logging in the DCRS. They can be downloaded from http://www.slf4j.org/download.html.

V. Demonstration of the DCRS

In this section, we present our demonstration of the DCRS by operating it on a Git based open source software, and then discuss the results obtained for the same.

A. *Data Source*

For our demonstration of the DCRS, we've selected the open source Android OS with Git as its Version Control System. The two versions of Android OS, namely – Ice Cream Sandwich (v4.0) and Jelly Bean (v4.1) have been considered for data collection. The source code has been obtained from Google's Git Repository (https://android.googlesource.com.

It was noted that the source code for Android was unevenly distributed in as many as 379 application packages. These include the packages for kernel, compilation headers, build libraries, linker code, and the native applications which are included in the Android OS, such as Web browser, Gallery, E- mail, Contacts, Music Player, Calendar, Telephony, etc.

Since we want to consider only the java source code or class files in the source code, we have analysed a few of the available Android application packages in order to determine the percentage of java source code files in every package. It was observed that there were significantly less number of java source code files as compared to other files formats, such as interface or activity layout files, resource files such as media, string values, etc. in each and every application package we had analysed. Table II presents the summary of Java source files contained in a few Android Application Packages for Android v4.0.

TABLE II
DISTRIBUTION OF JAVA FILES IN APPLICATION PACKAGES

Android Source Package	Total Number of Source Files	Number of Java Files	Percentage of Java files (%)
Email	1070	402	37.62
Contacts	1071	331	31.00
Telephony	707	200	28.61
Gallery	87	22	25.29
Bluetooth	207	38	18.47
Calculator	121	13	10.72
Desk Clock	301	31	10.33
Music Player	516	48	9.32
NFC	243	22	9.13
Calendar	398	11	2.81
Total	**4721**	**1154**	**18.66**

Fig. 4 gives the distribution percentage of Java source files for the same data in Android v4.0. However, the Android Application Packages were "cloned" (downloaded) by the DCRS itself from the git repository: https://android.googlsesource.com. This functionality has already been discussed in detail in Section IV.

We now discuss the results obtained for the Android 'Contacts' Application Package.

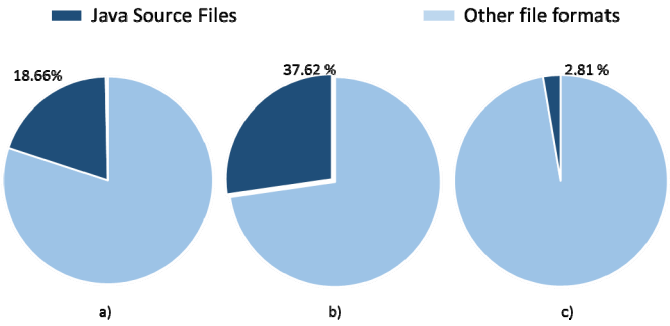

Fig. 4. Distribution of Java Files in Android Application Packages – Mean Percentage in a, Maximum in b and Minimum in c.

B. *Analysis of Results*

We have operated the DCRS on Android's 'Contacts' Application Package (v4.0 and v4.1) to first obtain the change-logs and finally various defect reports as stated in Section III. Fig. 5 presents a sample change record from the change-log of the Contacts application.

```
TIMESTAMP 1354144296

BEGINNING OF THE COMMIT RECORD

Prevent going into single contact mode un-necessarily. On a
tablet, after creating a new contact, the contact list
would enter "single contact mode". This mode is intended
for the case where you may have an account filter set and
you want to see a contact in a different account.  Combined
with a stale contact list, this caused the app to go into
single mode when it was not necessary.
This will fix the most common case where all accounts are
shown and a new contact is added. The new contact will be
displayed in the normal list. This will not fix the case
where an account filter is active and a new contact is
created.  In this case, the new contact is still displayed
in single contact mode.

Also tested was the test cases in:
I233162572e25acb737c9eae787dfc146879a0dc2

Bug: 7621855
Change-Id: I6f323c4003677cec9db63cae347f4532071037b9

ENDING OF THE COMMIT RECORD BODY
src/com/android/contacts/list/ContactBrowseListFragment.jav
a | 9 ++++++++-
1 file changed, 8 insertions(+), 1 deletion(-)
```

Fig. 5. Sample change record in the change-log of Contacts Application

1) Defect Data Analysis: From the Defect count and metrics report, we were able to obtain the data for the total number of defects reported in the classes for v4.0, during the transition from v4.0 to v4.1. Table III presents the distribution of defects in the classes of Contacts application in v4.0. Fig. 6 depicts a diagrammatic representation of the defect distribution in the considered software. Majority of the classes (67.8%) were found to have no defect during the transition from v4.0 to v4.2. 29% of the classes were found to be having a defect count in the range of 1-5. Classes with the defect count of 6 and above were merely 3.2%.

TABLE III
DISTRIBUTION OF DEFECTS IN THE CLASSES

Number of Defects	0	1-5	6-10	11-15	>15
No of classes	224	96	4	4	3

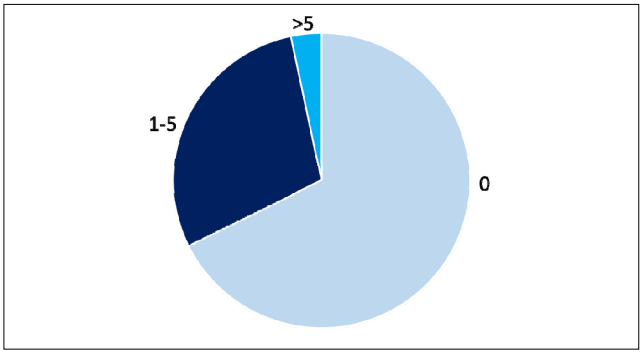

Fig. 6. Distribution of Defects

2) Change Data Analysis: From the consolidated defect and change report, we were able to obtain the data for the total number of changes incurred in the classes for v4.0, during the transition from v4.0 to v4.1. Table IV presents the distribution of changes in the classes of Contacts application in v4.0. Fig. 7 depicts a diagrammatic representation of the change distribution in the considered software. Majority of the classes (64%) were found to have no change incurred during the transition from v4.0 to v4.2. 33% of the classes were found to be having a change count in the range of 1-5. Classes with the total number of incurred changes more than 5 were only 3%.

TABLE IV
DISTRIBUTION OF CHANGES IN THE CLASSES

Number of Changes	0	1-5	6-10	11-15	>15
No of classes	212	109	4	2	4

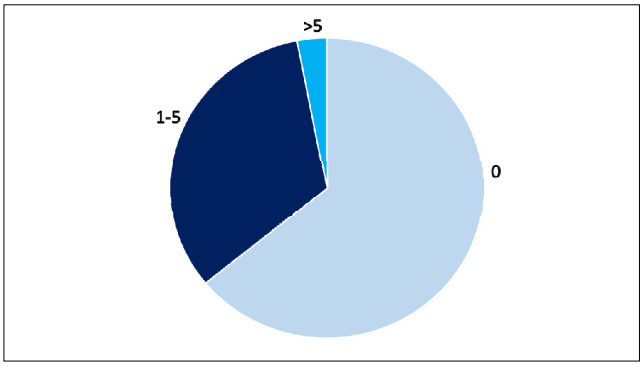

Fig. 7. Distribution of Changes

3) Statistical Comparison: From the analysis of newly added and deleted source code file reports, we can perform a statistical comparison of the two different versions (v4.0 and v4.1) of the Android Contacts application in the terms of, for example, the total number of added java files, total number of deleted java files, etc. Table V summarizes the various parameters we have computed for comparing the two versions of the application under consideration.

TABLE V
PARAMETERS FOR COMPARISON

Parameter	Value
Number of new source code files	31
Number of defects reported in new files	64
Number of deleted source code files	8
Number of defects reported in deleted files	10
Number of files added and then deleted from v4.1	0
Total number of file changes	23

VI. APPLICATION OF THE DATA COLLECTED THROUGH DCRS

In the previous section, we presented a demonstration of the DCRS for a sample Android OS application and analysed the data and results obtained from the various generated reports. On the basis of this discussion, we can state that the data collected from DCRS can be potentially employed in the following applications:

A. Defect Prediction Studies

Fault proneness or defect prediction is a useful technique for predicting the fault prone classes in a given software.

Simply, it can be stated as the method to predict the occurrence and/or number of defects in the software under consideration. For the past many years, defect prediction has been recognized as an important research field so as to organize a software project's testing resources and facilities. For example, consider a scenario wherein we have limited time and/or resources available for software testing procedure and activities. In such a situation, appropriate defect prediction models can aid the testing personnel to focus more attentively on those classes which are highly probable to be defective in the later releases of the software. Various studies have been conducted in the past for predicting effective fault proneness models, and also for validating the impact of OO Metrics on fault proneness ([1]-[3], [9], [10]).

It can be observed form Section V B. that the defect reports generated by the DCRS can be effectively employed for such studies.

B. Change Proneness Studies

Change proneness may be defined as the probability that a given component of the software would change. Along with defect prediction or fault proneness, change proneness is also very crucial and needs to be evaluated accurately. Prediction of change prone classes may aid substantially in maintenance and testing. A class that is highly probable to change in the later releases of a software needs to be tested rigorously, and proper tracking is required for that class while modifying and maintaining the software. Therefore, various studies have also been carried out in the past for predicting effective change proneness models, and to validate the impact of OO Metrics on change proneness [11]-[14].

As we have already described in Section V B., the DCRS reports can also be used in Change analysis and therefore for change proneness studies as well.

C. Statistical Comparison

We have already demonstrated the statistical comparison of the two versions (v4.0 and v4.1) of the Android OS Contacts Application Package in Section V B. Such kind of comparison can also be extended or generalized for a large number of Git based open source software systems and applications. We may also identify some additional parameters for the comparison through defect and change data analysis, such as the total number of defects reported and total number of changes incurred in the previous version of the considered software.

VII. CONCLUSIONS AND FUTURE WORK

Giving a description of the Defect Collection and Reporting System, this paper concludes that the developed system can be potentially employed for the collection of Defect Data pertaining to Open Source Software (which employ Git as the Version Control System) and generating useful reports for the same.

The gathered information can be effectively used for various purposes, including the following applications:

- Defect prediction, change prediction and related research work or studies, including analysis and validation of the effect of given metric suite on fault and/or change proneness (CKJM, QMOOD, etc.) and the evaluation and comparison of various techniques in predicting fault and/or change proneness models, such as statistical and machine learning methods.
- Statistical Comparison of any two versions of a given Open Source Software (which is based on Git version control system), in terms of various parameters, including the number of source files which have been added in the newer version, the number of source files which were present in the previous version but have been deleted in the newer version, and the number of defects which have been reported in the previous version of the software under consideration.

We have planned to carry out similar studies and work which would not be limited to only Git Version Control System based Open Source Software, but also for other widely used open source applications that are made available through various software repositories.

REFERENCES

[1] R. Malhotra, "A Defect Prediction Model for Open Source Software", *Proceedings of the World Congress on Engineering* (2012) Vol II, London, U.K, 2012.

[2] K.K. Aggarwal, Y. Singh, A. Kaur and R. Malhotra, "Empirical Analysis for Investigating the Effect of Object- Oriented Metrics on Fault Proneness: A Replicated Case Study", *Softw. Process Improve. Pract.* , 389-412, 2008.

[3] I. Gondra, "Applying machine learning to software fault-proneness prediction", *Journal of Systems and Software 81*, 186–195, 2008.

[4] B. Handerson and Selleres, "Object-Oriented Metrics, Measures of Complexity", *Pentice Hall,* 1996.

[5] S. Chidamber and C. Kemerer, "A metrics Suite for Object-Oriented Design", *IEEE Trans. Software Engineering*, vol. SE-20, no.6, 476-493, 1994.

[6] J. Bansiya and C. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment", *IEEE Transactions on Software Engineering*, vol. 28, no. 1, 4-17, 2002.

[7] T. McCabe, "A Complexity Measure", *IEEE Transactions on Software Engineering*, vol. 2, no. 4, 1996.

[8] A. Hassan, "The Road Ahead for Mining Software Repositories", *Frontiers of Software Maintenance,* 48-57, 2008.

[9] Y. Kaur and R. Malhotra, "Application of Random Forest in Predicting Fault-Prone Classes", *International Conference on Advanced Computer Theory and Engineering,* 37-43, 2008.

[10] F. Peters, T. Menzies and A. Marcus, "Better Cross Company Defect Prediction", *MSR 2013,* San Francisco, CA, US, 409-418, 2013.

[11] P. Liang and J. Li, "A Change-Oriented Conceptual Framework of Software Configuration Management", IEEE.

[12] R. Malhotra and M. Khanna, "Investigation of relationship between object-oriented metrics and change proneness", *International Journal of Machine Learning and Cyber*netics, Vol. 137, Number 1(January 2010), 2012.

[13] X. Tao, "Software Configuration Management of Change Control Study Based on Baseline", *International Conference on Intelligent Control and Information Processing*, August 13-15, 2010 - Dalian, China.

[14] Z. Yuming, L. Hareton and X. Baowen, "Examining the Potentially Confounding Effect of Class Size on the Associations between Object-Oriented Metrics and Change-Proneness", *IEEE Transactions on Software Engineering*, Vol. 35, No. 5, September/October 2009.