Analyzing the Change Profiles of Software Systems using their Change Logs

Munish Saini[(1),] Kuljit Kaur [(2)]

(1) Department of Computer Science and Engineering.Guru Nanak Dev University (India)
E-mail: munish_1_saini@yahoo.co.in
(2) Department of Computer Science and Engineering.Guru Nanak Dev University (India)
E-mail: kuljitchahal@yahoo.com

ABSTRACT

This paper analyzes the change history of various software systems for understanding their evolutionary behavior with respect to the type of changes performed over a period of time. The main objectives of this research work are: (a) What types of changes are most likely to occur in a software system during its evolution? (b) Is there any pattern in the type of changes performed over time in a system? An automated keyword based categorization technique is applied to the textual description of commit records of the software systems to categorize change activities into various types such as: Adaptive, Corrective, Perfective, Enhancement, and Preventive. The study finds that corrective changes are the maximum and preventive changes are the least in the software systems analyzed here.

Keywords: GIT, Software Maintenance, Software Evolution, Open Source Software.

1- INTRODUCTION

Software evolution refers to the phenomenon of software change and growth. The environment in which software has to work changes over time and the software itself must adapt to this changing environment. Lehman's first law [1] of software evolution stipulates that useful real-world software systems (i.e., E-type) must undergo change in response to various types of requirements such as corrective, adaptive, perfective, and preventive[2-4]. Kemerer and Slaughter [5] point out that there are several patterns that characterize the maintenance (evolution [6]) of a software system. A relatively high volume of corrective changes takes place immediately after the system becomes operational. It stabilizes as the system matures, but increases when the system ages (due to system entropy). However, functionality enhancement related changes are done on a fairly regular basis. We could not find out any research work which supports/refuse these claims. So it becomes a topic of study in this research paper.

Previous works [7], [8] on change types and frequency of their occurrence are based on surveys. An empirical study [9], reported in 2003, finds significant

differences in the distributions of various change types as against the results reported by the survey based studies. Among the reasons cited for the difference in results include 'change in the nature of software development'. It may be interesting to observe the results after a gap of another ten years given the emergence of new development approaches, such as incremental and iterative development [10]. Moreover, these papers do not consider patterns of change in the change types during the software evolution as talked about in [5] and being studied in this research work. The study analyses the data collected from open source software systems as it may clarify the doubts regarding lack of systematic maintenance in case of such systems [11].

This research aims to study the following:

1) What types of changes are most likely to occur in a software system during its evolution?
2) Is there any pattern in the type of changes performed over time in a system?

Section 2 of this paper discusses the related work. Section 3 specifies the data collection procedure. The change categorization approach is specified in Section 4. Section 5 presents the used methodology along with validation of the automated change categorization approach. The experimental results and its statistical analysis are presented in Section 6. The discussion of the obtained results is specified in Section 7. Section 8 concludes the paper and last section gives the future directions for the work.

2- LITERARTURE REVIEW

Lientz, Swanson, and Tompkins [7] carried out a survey on the relative effort put in adaptive, corrective, and perfective types of maintenance. The study stated that 17.4% of maintenance effort was categorized as corrective in nature; 18.2% as adaptive; 60.3% as perfective; and 4.1% was categorized as 'other'. Nosek and Palvia [8] repeated the same experiment and obtained similar results. However, Schach *et al*. [9] conducted an empirical study in 2003 and found a significant difference in the results from the previous survey based studies.

Lee and Jefferson [12] found the distribution of maintenance effort of a web based Java application similar to the one reported by Basili et al. [13] for software, developed using a different programming paradigm (FORTRAN and Ada). Sousa *et al.* [14] studied 37 large organizations situated in Portugal using a survey based approach. They concluded that, on average, organizations spend 48.6% of the maintenance effort on adaptive maintenance; 36.2% on corrective; only 1.7% on preventive; and 13.5% on perfective maintenance. Yip and Lam [15] conducted a survey of the state of software maintenance in Hong Kong. The results of the study indicate that enhancement related work is the largest among all the maintenance categories (39.7%) followed by corrective (15.7%).

Unlike the studies which model software maintenance by taking a snapshot of software at one point of time, next paragraphs mention the work which investigates the changes in maintenance types covering software life time or a portion of a life time.

Abran and Nguyenkim [16] did a trend analysis of maintenance workload distribution for a time period of two years in a Canadian financial institute. The trend analysis showed cyclic fluctuations in almost all types of changes with perfective changes decreasing sharply towards the later phases of the time period.

Gefen and Schneberger [17] examined a state-of the art Information System for 29 months. They could identify three distinct periods. An initial phase shows an upsurge in corrective modifications (stabilizing phase), followed by the addition of new functions to the existing applications (improvement phase), and then by the addition of new applications (expansion phase).

Another study investigates the changes of maintenance requests during the lifetime of a large software application examined over a 67 month period [18]. They identify four distinct stages. User support types of maintenance requests dominate the first stage, corrective changes dominate the second stage, and enhancement type of changes dominates the third stage. In the last stage, all these types of change requests diminish and the organization starts looking for a replacement of the software.

A few studies have been conducted which attempt to construct descriptive and predictive models for maintenance releases. Goulao *et al.* [19] used the time series forecasting model to predict software maintenance and evolution request in an open source software. They used an *ARIMA Model* [20] for prediction of software evolution trends and seasonal patterns. Kemerer and Slaughter [5] proposed an empirical approach to study software evolution. In this paper a longitudinal research is performed by using sequence analysis. This study allows understanding of how software maintenance activities and costs change over time and gives information about current maintenance and development practices. *Kemerer and Slaughter* [21] discussed the software evolution drivers and dynamics. They conducted the research in three phases. In the first phase, various patterns of software evolution is investigated and assessed whether they are consistent with the laws of software evolution [22]. In the second phase, various drivers of software maintenance are identified and in the last phase the modeling and prediction of software objectives are presented.

In the context of this work, software change classification may also be of interest to the readers. Swanson [23] gave the first and the most commonly used classification. *Mockus and Votta* [2] suggested an automated method of classifying software changes based on their textual descriptions. In this paper they provided a classification of changes, on the basis of specified keywords in the textual abstracts of changes, into three primary categories (adaptive,

corrective, and enhanced) and introduced another category (inspection maintenance). Hassan [24] extended the work of Mockus and Votta [2] to classify change messages. They presented an automated classification of change messages in open source software projects, and classified the change messages into mainly three types as a bug fix, a feature introduction, or a general maintenance change. Kim *et.al* [25] used a machine learning classifier to determine the type of change and classify it as either buggy or clean change. Lehnert et al. [26] discussed the change classifications, but they restrict to fine grained changes only. Chapin *et.al* [27] discussed about the types of software evolution and software maintenance. They discuss the classification of software maintenance activities for practitioners, managers, and researchers. They used a clustering method to combine the various activities into clusters.

The study given in [2] is of our interest. Various issues which are not discussed or presented in the paper are considered as the base of our research work. The issues which we found are as follows:- Change Classification types are limited, Number of keywords specified are less, some common keywords are not included and they don't specify the strong reason for their elimination, frequency count of various change categories and their subcategories is not shown.

Most of these studies do not consider the frequency of different of types of changes; rather they measure effort spent in making changes corresponding to the different change types. It has been observed that corrective changes demand less than half of the effort of non-corrective changes (When change set size is controlled) [28]. So effort distribution cannot be used as an indicator for change distribution. This study is first of its kind as we could not find any research work which does a trend analysis of change distribution in open source software. In this research work the change history data of open source software projects are first classified into particular change categories and then analyzed both graphically and statistically to find the existing common pattern in the change activities.

3- DATA COLLECTION

We selected six open source software projects for performing the analysis. These projects are selected on the basis that all of them have high number of commits, authors and have active development, In this data set, PostgreSQL and GnuCash (16-17 years) are the software systems with long history. With respect to this, Twitter-MySQL and PHP-src can be considered with medium history (13-14 years), and Apache-Tomcat and Wordpress with comparatively short history (8-10 years).

The repositories of the open source software projects are obtained from GIT [29] or GIT Hub [30]. These repositories are downloaded to make clone of the original repository onto the local machine using *GIT Bash*. The version wise change history of projects obtained from these repositories by using *log* –

oneline command and stored in *.txt files*. The repository information along with the descriptive statistics of the projects is provided in Table 1.

Table 1 Descriptive statistics of the six open source software projects

Software Projects	**Origin Date**	**Data Collection Date**	**Number of year**	**Number of Authors**	**Total Commits Analyzed**	*avg, min, max* **Commits each year**
Postgre-SQL	7/9/1996	5/21/2013	17	37	43644	2567, 963, 3120
WordPress	4/1/2003	5/21/2013	10	46	24249	2424, 669, 4596
Twitter MySQL	8/1/2000	5/21/2013	13	1270	71527	5502, 47, 13401
Php-Src	4/7/1999	5/20/2013	14	372	83654	5975, 1887, 8541
Apache Tomcat6.0	10/20/2005	4/30/2013	8	19	5216	652, 50, 1023
GnuCash	11/1/1997	5/21/2013	16	34	19762	1235, 222, 3019

The basic objective of collecting this data is that we want to

A) Categorize each commit record into various change categories.

B) Study the change pattern

4- CHANGE CLASSIFICATION APPROACH

4-1 CHANGE CLASSIFICATION CATEGORIES

Majority of the research studies do not agree on a uniform categorization of change types. We consulted *Swanson* [23], *IEEE* [31] [32], and *ISO\IEC 14764* [33] for deciding the type of change categories; and identified *Adaptive, Corrective, Perfective, Enhancement, and Preventive* as the main change categories. The *Adaptive* category consists all of those activities which are performed due to the changing environment such as adjusting of code, feature etc. The Corrective change consist all corrective type of activities such as fixing, identification and isolation of the the bugs. Corective changes are the most common type of changes which are performed at the most in all evolving software. All the rearrangement, maintenance activities like code beautification is classified as the *Perfective* changes*. Preventive* type of changes consist all those activities which are performed with respect to the future mainte-

nance such as re-implementation, revision, re-installation of certain features. The main difference between the perfective and preventive changes is in the scope of the maintenance where perfective is limited with current maintenance where as the preventive is more concerned about the future maintenance activities. The *Enhancement* category was added so as to detect the new functionality additions and separate them from changes done to improve the existing software. The motivation for this is the change in the development process followed nowadays (agile software development) from the times when the change classification was proposed (traditional process models). Even the concept of software evolution has become more prevalent over the period of time in comparison to the age old concept of software maintenance.

4-2 SPECIFICATION OF KEYWORDS

The change history of the six open source software projects is manually analyzed by both the authors individually to find the list of significant keywords (*keywords in accordance with the definition of specified change category*). The keyword which is in accordance with a particular change type definition is placed in that change categories. For example *fixes* and its similar terms like *fixed, fixing* specifies a corrective action, hence placed in corrective change category. Similarly keyword like *adjust*, *allocates* indicates the adaptive action is performed. Finally, the individual keywords list of each author is consulted by both the author together to find any discrepancy in categorization of particular keywords in a specified change category or any keyword which is missed by one but identified by another author. This double-manual evaluation approach eliminates the chances of missing keyword or placing a keyword in wrong change category. Further, for generalization of the list of keywords we used the WordNet [34] which allows the grouping of keywords based on the meaning of the keywords such as keywords like *fix, fixed, fixing, fixes* are grouped as one keyword such as *fixes*. This reduces the number of keywords in the specified list. At last the concept of tf-idf [35] is also used to eliminat all those words which are irrelevant in the text. The generalized list of identified keywords and their specified categories is shown at the end of paper in Table11 of Annexure 1. We assumed that this list of keywords is enough and efficient to analyze the history and trends of the open source software projects.

4-3 FILTERING OF COMMIT HISTORY

The collected data is very much unstructured. It is found that at many commits entries in the change log no commit message is written, or committer of the change had written some unusual text. So the collected version wise commit history of the all the open source software require filtering process to remove all those commits entries. In this research work we had used manual and automated approach for filtering of the commit history. For automated filterining, a code is written in *Java* which itself look and eliminates the *blank commit* (commit at which no comment is mentioned) and the *unusual commit* (commit

message with no meaning, or do not contain listed keyword) from commit history of all six projects. Further, filtered version wise commit history of all the open source software projects is evaluated and cross checked manually by main author of this paper to filter out any remaining *blank commit* or *unusual commit*, which are missed out in automated filterining process. So at the end, we got filtered version wise commit history which does not have any *blank* or *unusual commit.*

4-4 CRITERIA FOR ASSIGNING CHANGE CATEGORY TO COMMIT RECORD

The assigning of the change category and counting the particular change category frequency is done automatically by the program written in *Java*. Each commit record in the commit history is looked to find out its possible change category. The specified criteria which are followed for assigning the change category consist of the following:

a) Look for the all specified keywords in a commit record and measures the frequencies of each keyword.
b) Look for change category in which these keywords lies.
c) The keyword with highest frequency will decide the change category. For example, if in a commit record keyword *adjusted* has highest count, then it means this commit belongs to the Adaptive change category.
d) In case, if the two keywords from different change category have the same occurrence frequency, then the keyword which is occurring first in the commit record is considered as the decisive.
e) The commit record which does not contain the specified keywords is skipped.
f) The commit record which does not have any commit message is also not considered in this classification.

5- METHODOLOGY

The version wise change history of all the open source software projects is evaluated separately to categorize each commit record (*textual description*) into a specified change category. This categorization is performed automatically by the program which is written in *Java*. The program satisfies and considers all the classification criteria specified before in this paper and gives output as the count of all the classification categories (*Adaptive, Corrective, Perfective, Enhancement, and Preventive*) for a particular version of the open source software.

5-1 VALIDATION OF AUTOMATED CHANGE CLASSIFICATION APPROACH

For validation of the automated classification approach we used the same approach as used by the *Mokus and Votta* [2]. We consulted four experts (*2 open source software developers, 1 Professor, 1 Ph.D researcher*) in the field of open source software evolution. We asked them to manually classify each commit record in the change log into a particular change category by using the same categorization criteria as used for the automated classification.We used the Cohen's Kappa (K) to evaluate the agreement between the manual and the automated change classification. In Table 2,3,4,5, the manual (*by different experts*) and the automated classification of commits of first version of *GNU Cash v1.3* is shown. The rows and columns of these tables specify the manual and automated categorization data respectively.

Table 2 Comparison of Expert 1 Classification with Automated Classification of GNU Cash version v1.3

	Automated Classification				
Expert 1 Classification	Adaptive	Corrective	Perfective	Enhancement	Preventive
Adaptive	70	0	30	0	0
Corrective	20	576	13	161	8
Perfective	0	0	100	0	0
Enhancement	0	0	0	300	0
Preventive	0	0	10	0	6

Table 3 Comparison of Expert 2 Classification with Automated Classification of GNU Cash version v1.3

	Automated Classification				
Expert 2 Classification	Adaptive	Corrective	Perfective	Enhancement	Preventive
Adaptive	70	0	21	0	0
Corrective	20	576	33	161	0
Perfective	0	0	87	0	0
Enhancement	0	0	12	300	4
Preventive	0	0		0	10

Table 4 Comparison of Expert 3 Classification with Automated Classification of GNU Cash version v1.3

	Automated Classification				
Expert 3 Classification	Adaptive	Corrective	Perfective	Enhancement	Preventive
Adaptive	70	0	21	0	0
Corrective	18	573	33	160	1
Perfective	2	2	86	0	4

Enhancement	0	1	12	300	3
Preventive	0	0	1	1	6

Table 5 Comparison of Expert 4 Classification with Automated Classification of GNU Cash version v1.3

	Automated Classification				
Expert 4 Classification	Adaptive	Corrective	Perfective	Enhancement	Preventive
Adaptive	70	0	21	0	0
Corrective	18	573	33	160	1
Perfective	2	2	87	0	0
Enhancement	0	1	12	301	3
Preventive	0	0	0	0	10

The calculated Kappa coefficient (shown in Table 6) for all the projects, lies between [0.5-0.8]. It indicates the existence of a ***moderate agreement*** between the automated and manual classification. This high value of the Kappa coefficient validates the fact that automated classification is valid and effectively categorizes the commits into different change categories.

Table 6 Kappa Coefficient for the six software project's (first version only)

	Kappa Coefficient			
open source software	**Auto/Expert1**	**Auto/Expert2**	**Auto/Expert3**	**Auto/Expert4**
PostgreSQL (version PG95-1)	0.59	0.6	0.57	0.59
WordPress (version 1.5)	0.68	0.71	0.73	0.79
Twitter MySql (version mysql3.23x)	0.58	0.57	0.56	0.60
Php-Src (version php4.0)	0.5	0.52	0.58	0.55
Apache Tomcat 6.0 (version 6.0.0)	0.69	0.7	0.68	0.71
GnuCash (version v1.3)	0.70	0.69	0.72	0.69

6- RESULTS AND ANALYSIS

In this section the results obtained by analyzing the commit records of different versions of open source software systems are presented. The results are shown in three steps. In step one, change distributions are analyzed. The data are presented using pie-charts. In the second step, change pattern study of different software is presented. The change pattern in the evolution of open source software has been presented using various tools such as bubble chart, and line chart. Bubble chart gives a strong visual appeal. Line chart has been used to simplify the picture as much as possible. In the third section, the change categorization data is statistically analyzed, to find the variation (fluctuation) and the change trend in the different change categories of the open source software projects.

6-1 CHANGE DISTRIBUTION ANALYSIS

This section analyses the distribution of different change categories by taking into consideration the whole change log of the software as a single unit of analysis. The pie-charts (see Fig. 1) indicate that the change distributions are different across most of the software systems. Corrective changes have the maximum share in the total number of changes in all the data sets. Preventive changes have the minimum share in all the other change logs except for Twitter MySQL and Apache Tomcat. Code restructuring has been taken up more frequently in these two programs.

Postgre SQL resembles GnuCash in change distribution (see Table 7). They both have a fare share of perfective and enhancement related changes, and comparatively lesser number of corrective changes. These distributions are comparable with that of Mockus and Votta [2], and Schach *et al.*'s [9] studies as they both focus on frequency of maintenance types and not on maintenance effort. Interestingly for the first three systems (PostgreSQL, GnuCash, and Twitter MySQL), changes in adaptive, perfective, and enhancement categories (when combined) are more than 45% (as in [2]), and corrective changes are close to 34%. But for the other three systems (PHP-src, Tomcat, and WordPress), changes when combined in the three categories are less than 45% (as in [9]), and corrective changes are close to 50%. Here we consider the changes of primary types only.

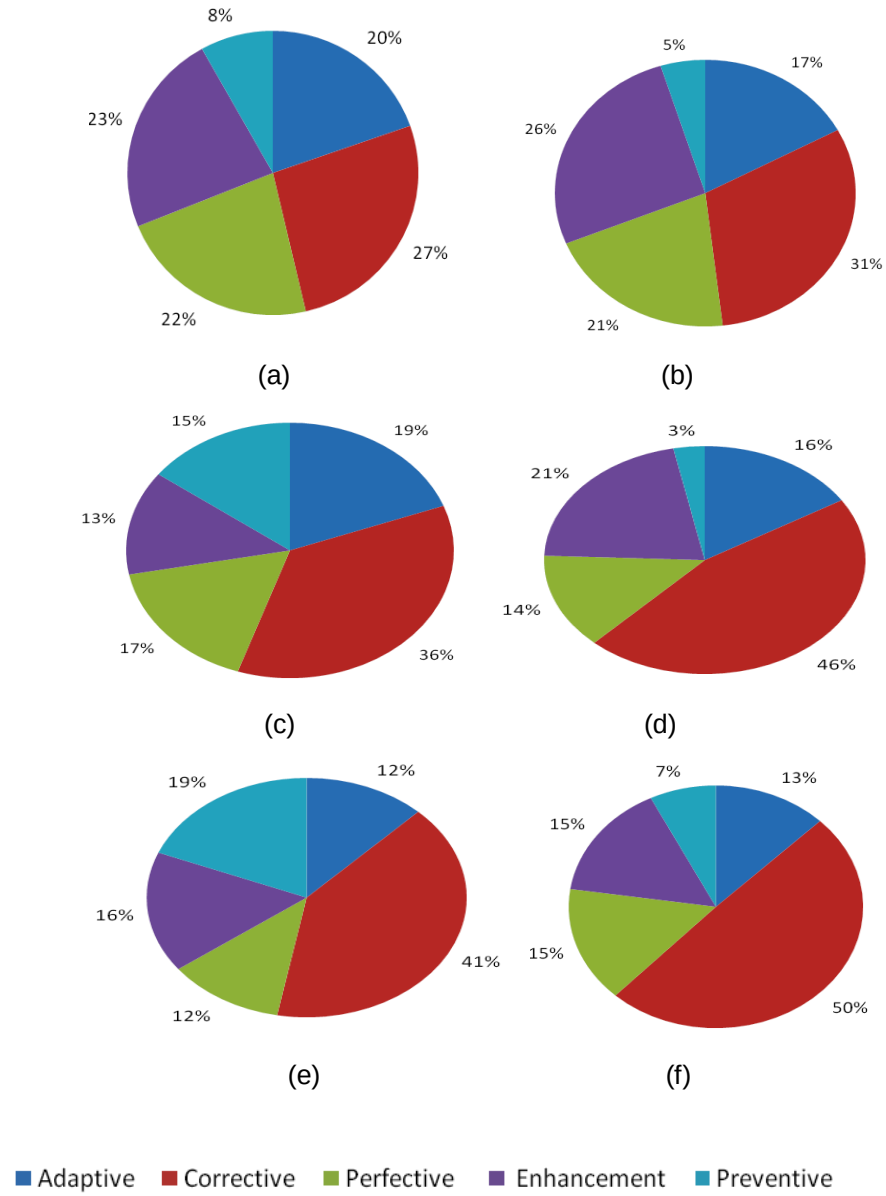


Figure 1 Pie charts representing the percentage count of each change category in (a) PostgreSQL (17 yrs) (b) GnuCash (16 yrs) (c) Twitter MySQL (13 yrs) (d) Php-Src (14 yrs) (e) Apache Tomcat (8 yrs) (f) Wordpress (10yrs)

Table 7 Change Distribution in Change Logs of various software Systems

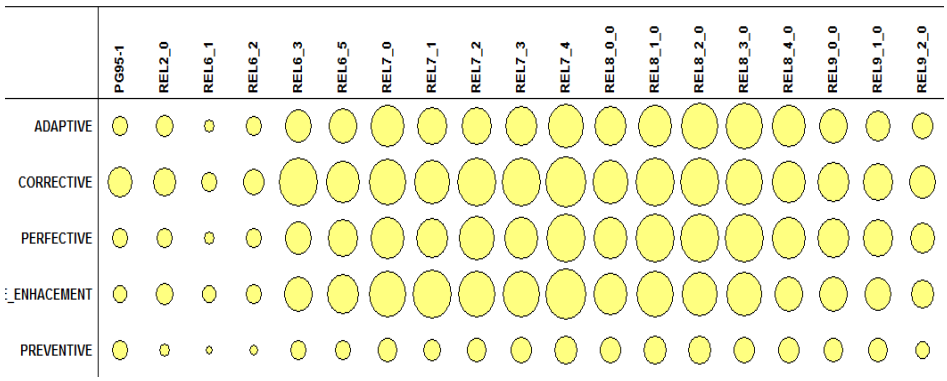
Type of Change	Postgre-SQL	GnuCash	Twitter MySQL	Php-Src	Apache Tomcat	Word-Press	Schach *et al.*[9]	Mockus *et al.*[2]
Corrective	26.70%	31.00%	35.00%	45.40%	41.30%	49.50%	50%	34%
Adaptive	19.60%	17.10%	19.30%	16.00%	12.20%	12.50%	2-4%	-
Preventive	8.10%	4.80%	14.80%	3.10%	19.40%	7.30%	-	4%
Perfective	22.50%	20.70%	16.80%	13.40%	11.90%	15.40%	36-39%	45% (also includes adaptive changes)
Enhance-ment	23.00%	26.40%	13.20%	22.10%	16%	15.30%		

6-2 CHANGE PATTERN ANALYSIS

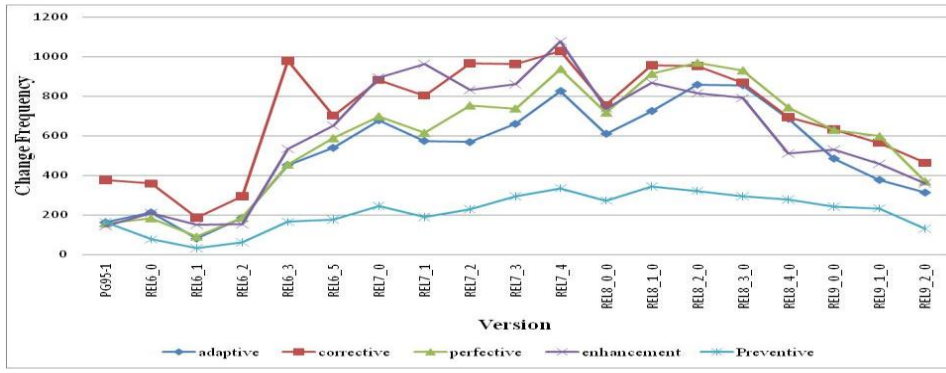
PostgreSQL:

Figure 1 (a) makes it clear that the corrective changes have the maximum percentage and preventive has the least percentage in PostgreSQL. It specifies that most of the change activities in PostgreSQL are corrective in nature followed by enhancement and perfective changes. The bubble chart in Figure 2(a) specifies the frequency of each change category in different versions. The size of bubble depicts the frequency count of the change performed. It is found that at the start of PostgreSQL evolution, change activity is very slow and corrective changes are the most dominant among all the change types. After the 4^{th} version, there is rapid increase in the activity of all the change categories. The corrective changes are surpassed by enhancement and perfective changes at some points, while least activity is found in preventive.

The peaks and troughs in the change pattern become more clearly visible in Figure2 (b). It specifies the trend of changes, and gives the comparison between frequency counts of different change categories over a period of time. As per the change pattern, corrective changes are the most prominent in the beginning followed by enhancement changes in the middle release. Towards the last releases of this data set, the perfective changes become the most prominent. The preventive change category has very less activity and remains at the bottom of the graph throughout. In the change pattern plot, it is found that there is not much fluctuation in the change activity. It starts from a lean period, remains high for a number of releases, and seems to be heading towards a lean period again.



(a)

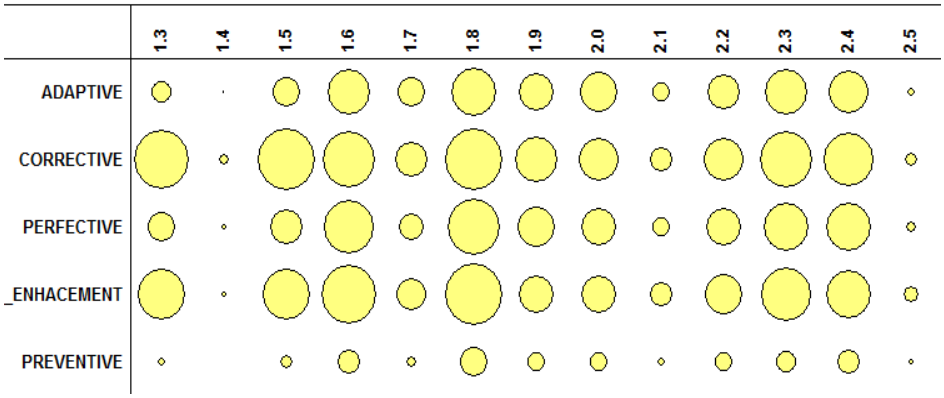


(b)

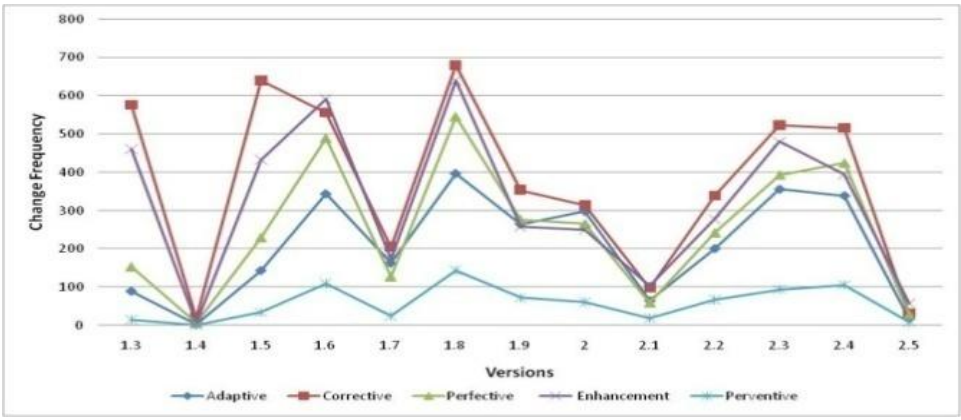
Figure 2 (a) Bubble chart, (b) Line Chart representing the frequency count of each change category in PostgreSQL

GnuCash:

There is a lot of fluctuation in the change activity of this software system (Figures 3(a) and 3(b)). Sudden change is observed at release 1.4, and 1.7. Otherwise the change is gradual. Except for preventive changes, all types of changes follow the same pattern. Most of the times, they all increase (may be in different proportions) at the same time and also decrease at the same time. Corrective changes are dominant throughout. Preventive changes are the least in the whole change history.



(a).

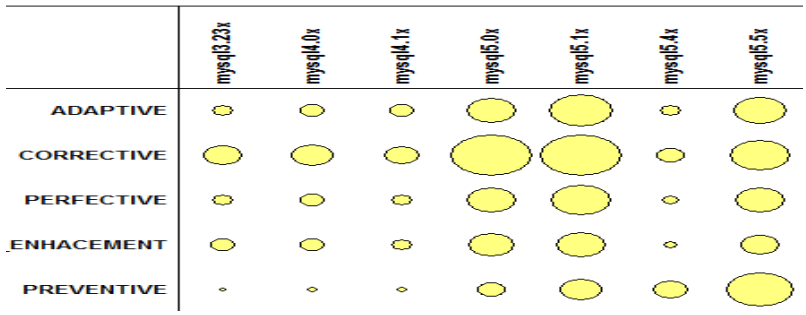


(b)

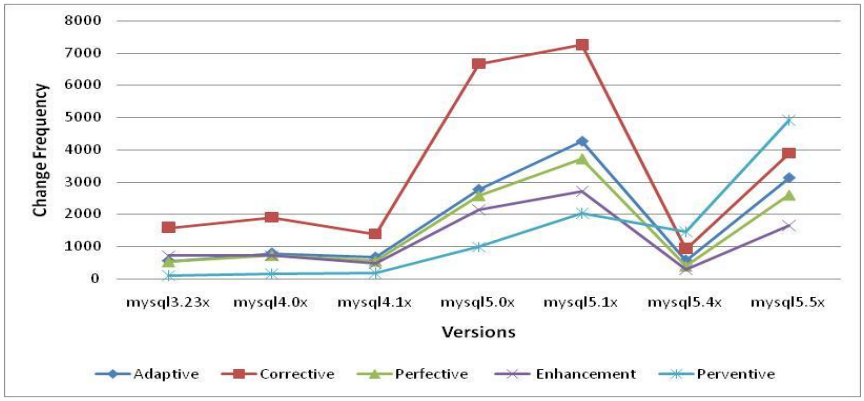
Figure 3 (a) Bubble chart, and (b) Line Chart representing the frequency count of each change category in all the versions of GnuCash

Twitter MySQL:

The change activity starts from a lean period with a major fluctuation after that (Figures 4(a) and 4(b)). Corrective changes are dominant, but not in all the releases. It is important to watch the increase in preventive changes, which has been least in other software systems analyzed in this study. As in GnuCash, all the changes increase and decrease together.



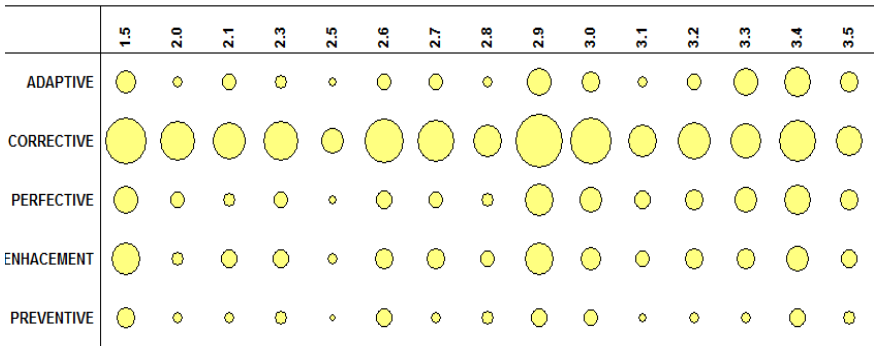
(a)



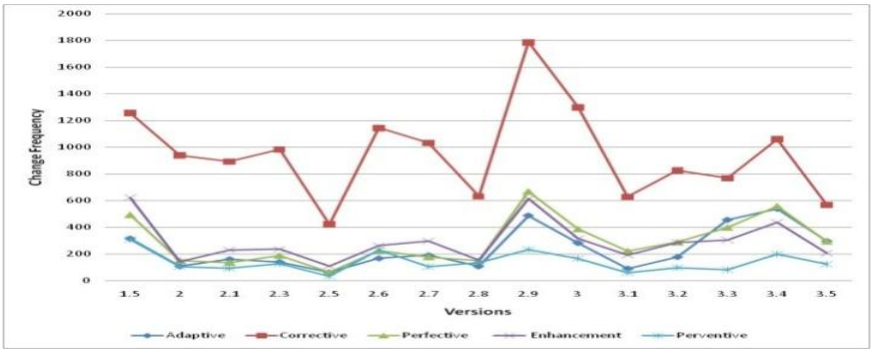
(b)

Figure 4 (a) Bubble chart, (b) Line Chart representing the frequency count of each change category of Twitter MySQL

WordPress:
The preventive changes are the least performed changes in all the versions and they remain at the bottom of the graph (Figures 5(a) and 5(b)). However, none of the other systems show the trend for corrective changes as in this software system. Corrective changes are the most dominant in all the versions and it is clearly seen from the given plot. Another trend of similar increase/decrease in all the changes is observed in this case too.



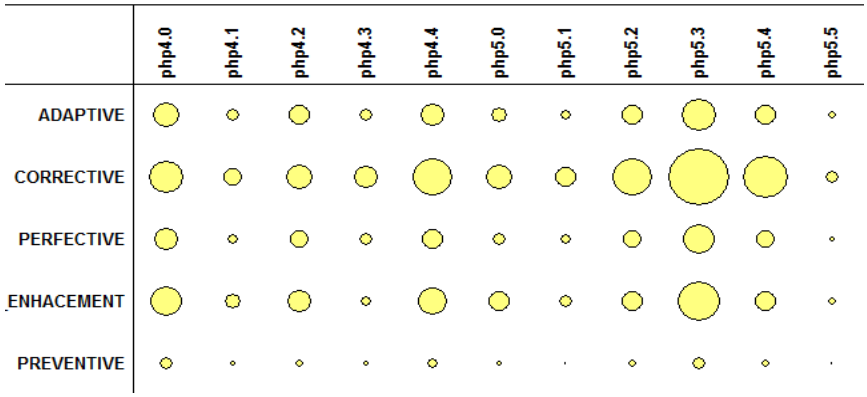
(a)



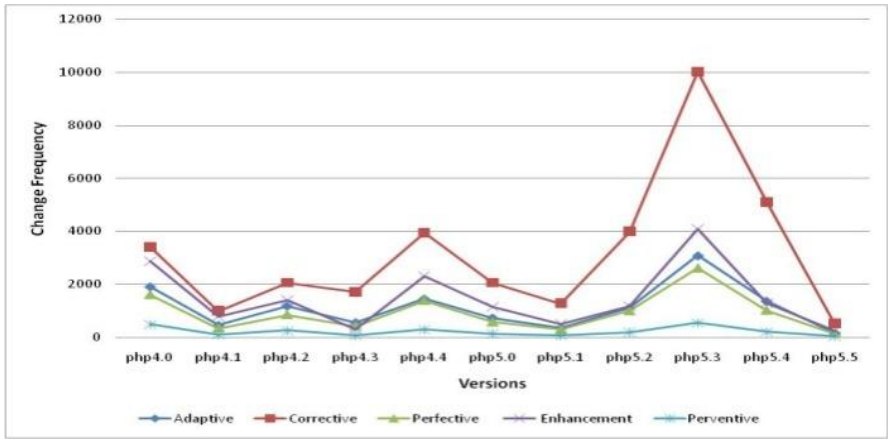
(b)

Figure 5 (a) Bubble chart, (b) Line Chart representing the frequency count of each change category of WordPress

Php-Src:
Changes, except preventive, are very regular in all the initial seven releases (Figures 6(a) and 6(b)). Later on, corrective changes shoot up. The corrective changes have the highest frequency followed by enhancement changes, where as preventive has the least. The adaptive and perfective change category curves overlaps throughout. After php5.1 release there is rapid continuous increase in the growth of each curve which remains for 2-3 versions followed by a decline in the curve.



(a)

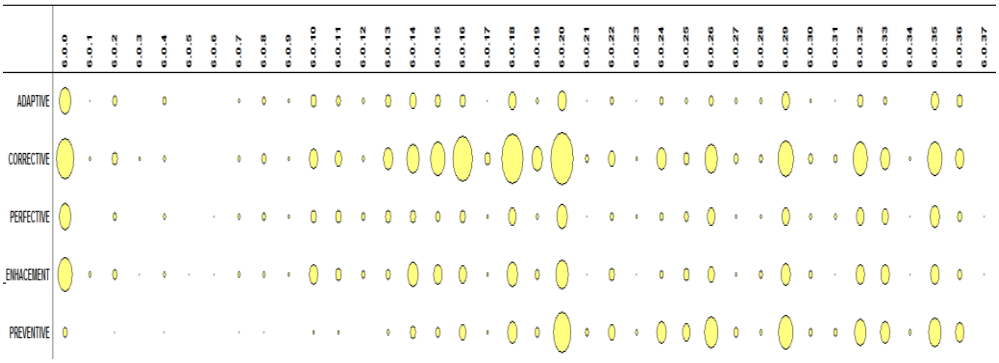


(b)

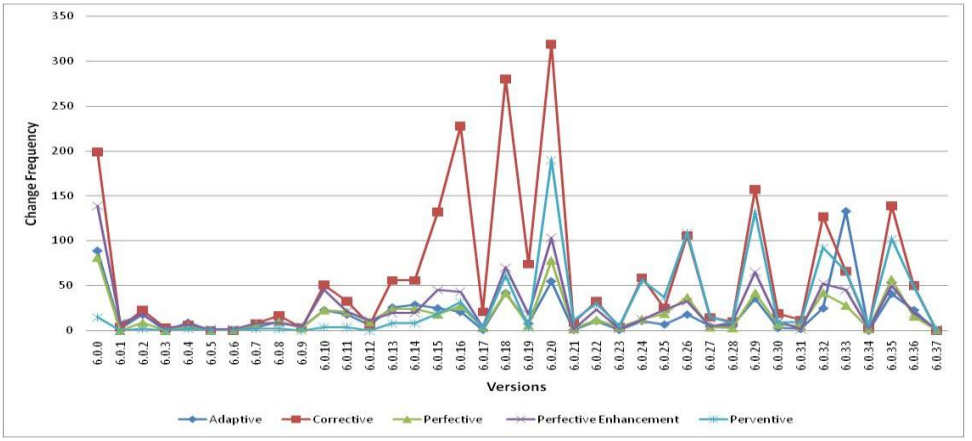
Figure 6 (a) Bubble chart, (b) Line Chart representing the frequency count of each change category of Php-Src

Apache Tomcat6.0:

In case of Apache Tomcat, the time span between various releases is very uneven and short. There are many releases which are released with a gap of one day or even on the same day. Due to that, very few or no change activity is present at many of the places in the bubble chart (Figures 7(a) and 7(b)). Initially, all types of changes are significant only to decrease in the next release. Not much activity is observed after that. Corrective changes gradually increase towards the middle of the change history study period. Preventive changes also follow the corrective changes after release 6.0.20. Here adaptive changes are very less.



(a)



(b)

Figure 7 (a) Bubble chart, (b) Line Chart representing the frequency count of each change category of Apache Tomcat

6-3 STATISTICAL ANALYSIS

6-3-1 ANALYSIS OF VARIANCE

The variation in the change categorization data of all software is statistically tested by using the *variance*. The existence of variance (shown in Table 8) indicates the variation in the data. It is found that the variance of Corrective change category in all software is higher than all other change types. Further, change categorization data is evaluated to find whether the different change categories have same variance or not (i.e having *same or different* change count). Although the Table 8 indicates the values of variance are not same but this (*homogeneity of variance*) is cross validated by using the Levene test.

The obtained *p- values* for the open source software projects are shown in Table 9. The calculated *p-value* of all the software is lower than *0.05*. Hence the null hypothesis that all change categories have homogenous (equal) variance is rejected for all the software projects.

Table 8 Calculated variance for each change category

Software	Adaptive	Corrective	Enhancement	Perfective	Preventive
PostgreSQL	57937.257	70266.8187	89149.427	78811.6959	8655.7836
GnuCash	18605.423	52767.3077	40631.192	30508.4103	2076.3077
Twitter MySQL	2351135.8	6915811.667	876609.95	1836115.667	2941111
Php-Src	709253.67	7174111.491	1398825.2	522288.6909	29373.891
Apache Tomcat	701.2802	6606.2425	912.6408	420.6693	1923.8073
WordPress	23097.543	118341.2667	24113.543	30187.4095	5578.0952

Table 9 Levene-test for testing the homogeneity of variance

Software	p-value
PostgreSQL	0.0024381
GnuCash	0.000797
Twitter MySQL	0.0157528
Php-Src	0.0143949
Apache Tomcat	0.0001702
WordPress	0.0029837

6-3-2 TREND ANALYSIS

The test for variance indicates that the change count value of a particular change category does not remain constant and vary from version to version. The variation in the data also indicates the existence of possible trend (*increasing/decreasing*) with which the change activities are performed. The linear regression analysis is performed on the change categorization data of different software to find out possible trends. The slope of the regression equation is the indicator of trend for analysed data. The positive and negative sign of the coefficient imply the increasing and decreasing trend respectively.Table 10 shows the statistically significant trends (having *significant t-statistics*).

Table 10 Trend analysis

		Regression Equation	Slope (b)/ or trend	Overall Trend
PostgreSQL	Adaptive	y = 302.5614 + 21.4860 x	21.486	Increasing
	Corrective	y = 554.5439 + 15.0982 x	15.0982	Increasing
	Enhancement	y = 424.4561 + 18.2807 x	18.2807	Increasing
	Perfective	y = 290.5614 + 30.3281 x	30.3281	Increasing
	Preventive	y = 116.8772 + 9.6807 x	9.6807	Increasing
GNUCash	Adaptive	y = 153.6154 + 7.5385 x	7.5385	Increasing
	Corrective	y = 468.8846 + -13.6758 x	-13.6758	**Decreasing**
	Enhancement	y = 385.4615 + -9.7473 x	-9.7473	**Decreasing**
	Perfective	y = 216.6538 + 4.6099 x	4.6099	Increasing
	Preventive	y = 37.8846 + 2.8516 x	2.8516	Increasing
Twitter MySQL	Adaptive	y = 258.2857 + 389.2143 x	389.2143	Increasing
	Corrective	y = 1819.4286 + 386.8929 x	386.8929	Increasing
	Enhancement	y = 633.0000 + 151.6071 x	151.6071	Increasing
	Perfective	y = 332.4286 + 311.8929	311.8929	Increasing
	Preventive	y = -1304.2857 + 674.3214 x	674.3214	Increasing
Php-Src	**Adaptive**	y = 1092.7455 + 5.3000 x	5.3	Increasing
	Corrective	y = 1674.0727 + 251.8364 x	251.8364	Increasing
	Enhancement	y = 1646.5455 + -28.6364 x	-28.6364	**Decreasing**
	Perfective	y = 902.4545 + 6.2727 x	6.2727	Increasing
	Preventive	y = 266.4545 + -8.0909 x	-8.0909	**Decreasing**
Apache Tomcat	**Adaptive**	y = 13.8634 + 0.2499 x	0.2499	Increasing
	Corrective	y = 48.9943 + 0.6656 x	0.6656	Increasing
	Enhancement	y = 25.9701 + -0.05920 x	-0.0592	**Decreasing**
	Perfective	y = 14.7269 + 0.1719 x	0.1719	Increasing
	Preventive	y = -4.3073 + 1.6959 x	1.6959	Increasing
WordPress	**Adaptive**	y = 106.6571 + 16.7179 x	16.7179	Increasing
	Corrective	y = 1053.0476 + -13.0643 x	-13.0643	**Decreasing**
	Enhancement	y = 291.4286 + 0.3714 x	0.3714	Increasing
	Perfective	y = 178.7810 + 14.4607 x	14.4607	Increasing
	Preventive	y = 163.6762 + -2.9179 x	-2.9179	**Decreasing**

It is found that in the six open source software projects, the *Adaptive* and *Perfective* changes have increasing trend, means in all versions of these software the *Adaptive* and *Perfective* change are performed with increasing trend. For *Corrective, Enhancement, and Preventive* changes we found the mixed trend. The *Corrective, Preventive changes* have shown decreasing trend for two software (*GnuCash, WordPress*), (*Php-Src, WordPress*) respectively but has shown increasing trend in all other software. Similarly the *Enhancement changes* have shown mixed trend of increasing for three software (*PostgreSQL, Twitter MySQL, WordPress*) and decreasing trend other three software (*GnuCash, Apache Tomcat, Php-Src).*

7- DISCUSSION

The change distribution, in sixopen source software projects are analyzed by classifying the commit activities into various change types. The classification of the commit record (*textual description*) is done by using the keyword based classification technique which follows the specified selection criteria. The whole process of categorization is an automated process accomplished by a program written in *Java*. The validation of the automated categorization method is done by finding the agreement between manual and automated categorization using the Cohen-Kappa test.The high value of the agreement indicates that the automated method is very much effective and efficient in performing the categorization of the commit records into various change categories.

The change distribution and change pattern analysis results have shown that the *corrective changes* are most often performed, where as the *preventive changes* have the minimum share in all change logs except for *Twitter MySQL* and *Apache Tomcat* as the Code restructuring has been taken up in these projects. In almost all the software projects the *Enhancement activity* is second among the change activities. Different charts (*line, bubble and pie chart*) give a very good indication of the change activity and its trends. It is observed that change activity in the projects follows the up and down trend. It does not remain constant. This variation in the data is tested statistically by finding the variance for each change category and then performing the Levene-test to find Homogeneity of the variance of different change categories. The output of the Levene-test has shown that the variance of all change categories is not *Homogeneous.* The overall *increasing/decreasing* trend in the various change category of all open source software is computed statistically by performing the regression analysis. The results of the trend analysis have shown that Adaptive and *Perfective* changes have increasing trend for all six open source software, where as the mixed trend (of *increasing and decreasing*) is found in other change categories.

8- CONCLUSION

In this research work, a keyword based categorization technique is used to extract the change trends from the unstructured data of software change logs. The commit history of six open source software projects is obtained and analyzed for finding the change trends occurring during software evolution. In this paper, the commit record of six open source software systems is used to find the changes and their respective types. An analysis of the change distribution indicates that corrective changes are the maximum and preventive changes are the least. Though detailed comparison is not possible with the results of the existing studies *(because of differences in the way change categories are defined*), but broadly the change distributions of the systems studied here match with the data reported by two different studies. As far as the trend in change pattern is concerned, we find that adaptive and perfective changes have increasing trend for all six open source software, where as the mixed trend (of *increasing and decreasing*) is found in all other change categories. It is also found that corrective changes are most often performed in all the six projects, where as the preventive have the minimum share in all change logs. If change activity increases/decreases, it increases/decreases in all the change types in most of the cases. Change bursts occur randomly. Another observation is that change activity is less in the beginning, followed by increasing and decreasing trend.

9- FUTURE WORK

In the future, we plan to investigate the characteristics (*other than their change logs e.g. information from the project web site*) of the software systems so as to understand the different patterns of their change. For example, it is observed that, preventive changes are very less throughout in some of the change logs but they follow an upwards trend in other cases. Fine-grained analysis of changes using keyword frequency is another direction for further research.

ACKNOWLEDGEMENT

This research work is done under the UGC sanctioned research project entitled "*Tracking open source evolution for the characterization of its evolutionary behavior*". We acknowledge the UGC for giving the grant for doing the research work.

REFERENCES

[1] M. Lehman, "On Understanding Laws, Evolution and Conservation in the Large Program Life Cycle," Journal of systems and Software, 1(3), pp. 213-221, 1980.

[2] M. Audris, L.Votta, "Identifying Reasons for Software Changes using His-

toric Databases," International Conference on software Maintenance, San Jose, CA, pp.120-130, 2000.

[3] R. Purushothaman, D.Perry, "Toward Understanding the Rhetoric of Small Source Code Changes," IEEE Transactions on Software Engineering, 31(6), pp: 511-526, 2005.

[4] L. Briand, V. Basili, "A Classification Procedure for the Effective Management of Changes during the Maintenance Process," Proceedings of International Conference on Software Maintenance, Orlando, FL, USA, pp: 328-336, 1992.

[5] C. F. Kemerer, S. A. Slaughter, "Determinants of Software Maintenance Profiles: An Empirical Investigation," Journal of Software Maintenance: Research and Practice, Vol. 9(2), pp. 235–251, 1997.

[6] K. H Bennett, V.T Rajlich, "Software Maintenance and Evolution: A Roadmap," 22nd International Conference on Software Engineering, IEEE Press, Limerick, pp. 73–78, 2000.

[7] B. P Lientz, E. B. Swanson, and G. E Tompkins, "Characteristics of Application Software Maintenance," Communication of the ACM, 21(6), pp. 466-471, 1978.

[8] J. T Nosek, P. Palvia, "Software Maintenance Management: changes in the last decade," Journal of Software Maintenance: Research and Practice, 2(3), pp. 157-174, 1990.

[9] S. R. Schach, B. Jin, D. R. Wright, G. Z Heller, and J. Offutt, "Determining the Distribution of Maintenance Categories: Survey versus Measurement," Empirical Software Engineering, 8(4), pp: 351-365, 2003.

[10] P. Mohagheghi, R. Conradi, "An empirical study of software change: origin, acceptance rate, and functionality vs. quality attributes," International Symposium on Empirical Software Engineering, IEEE Computer Society Press pp. 7-16, 2004.

[11] I. Stamelos, L. Angelis, A. Oikonomou, and G. L Bleris, "Code quality analysis in open source software development," Information Systems Journal, 12(1), pp. 43-60, 2002.

[12] M. G. Lee, T. L Jefferson, "An empirical study of software maintenance of a web-based java application," 21st IEEE International Conference on Software Maintenance, IEEE Computer Society Press, pp. 571-576, 2005.

[13] V. Basili et.al, "Understanding and Predicting the Process of Software Maintenance Releases," 18th International Conference Software Engi-

neering, IEEE CS Press, Berlin, pp. 464–474. 1996.

[14] M. Sousa et al., “A Survey on the Software Maintenance Process,” International Conference on Software Maintenance, IEEE CS Press, Bethesda, MD, pp. 265–274, 1998.

[15] S. Yip, and T. Lam, “A software maintenance survey,” 1st Asia-Pacific Software Engineering Conference, Tokoyo, pp. 70–79, 1994.

[16] A. Abran, H. Nguyenkim, “Analysis of maintenance work categories through measurement,” International Conference on Software Maintenance, Sorrento, pp. 104-113, 1991.

[17] D. Gefen, S. L. Schneberger, “The Non- Homogeneous Maintenance Periods: A Case Study of Software Modifications”, IEEE Conference on Software Maintenance, Monterey CA, 1996.

[18] E. Burch, H. J. Kungs, “Modeling software maintenance requests: a case study,” International Conference on Software Maintenance, IEEE Computer Society Press, Bari, Italy, pp. 40-47, 1997.

[19] M. Goulao, N. Fonte, M. Wermelinger, and F. B. Abreu, “Software Evolution Prediction Using Seasonal Time Analysis: A Comparative Study,” 16th European Conference on Software Maintenance and Reengineering (CSMR), Szeged, pp 213-222, 2012.

[20] S. Anderson, A. Auquier, W. Hauck, D. Oakes, W. Vandaele, and H. Weisberg, “Statistical methods for comparative studies: techniques for bias reduction”, John Wiley and Sons, 1980.

[21] C. F Kemerer, S. Slaughter, “An Empirical Approach to Studying Software Evolution,” IEEE Transaction on Software Engineering, 25(4), pp 493-509, 1999.

[22] L. Belady, M. Lehman, “A Model of Large Program Development,” IBM systems Journal 15 (1), pp: 225-252, 1976.

[23] E. B. Swanson, “The dimensions of maintenance,” 2nd International Conference on Software Engineering, IEEE Computer Society Press, pp. 492–497, 1976.

[24] A. Hassan, “Automated Classification of Change Messages in Open Source Projects,” ACM Symposium on Applied Computing, pp 837-841, 2008.

[25] S. Kim et.al. “Classifying software Changes: Clean or Buggy,” IEEE Transactions on Software Engineering, 34(2), pp 181-196, 2008.

[26] S. Lehnert, M. Riebisch, “A taxonomy of change types and its application in software evolution,” 19th International Conference and Workshops on Engineering of Computer Based Systems (ECBS), IEEE Computer Society Press, Novi Sad, Serbia ,pp. 98-107, 2012.

[27] N. Chapin, E. Joanne, K. Khan, J.F Ramil, and W. Tan, “Types of Software Evolution and Software Maintenance,” Journal of Software Maintenance and Evolution: Research and Practice, John Wiley & Sons, Ltd, Volume 13, pp 3-30; 2001.

[28] H.C Benestad, B. Anda, and E. Arisholm, “Understanding cost drivers of software evolution: a quantitative and qualitative investigation of change effort in two evolving software systems,” Empirical Software Engineering, 15(2), pp. 166-203, 2010.

[29] http://git-scm.com/ (Last accessed on 5/21/2013)

[30] https://github.com (Last accessed on 5/21/2013)

[31] IEEE.IEEE Standard Glossary of Software Engineering Terminology. Institute of Electrical and Electronics Engineers: New York NY, pp. 83, 1990.

[32] IEEE.IEEE Standard for Software Maintenance (IEEE Std 1219–1998). Institute for Electrical and Electronic Engineers: New York NY, pp. 47, 1998.

[33] ISO/IEC. Software Engineering—Software Maintenance, ISO/IEC FDIS 14764:1999(E). International Standards Organization: Geneva, Switzerland, pp: 38, 1999.

[34] http://wordnet.princeton.edu/wordnet/ (Last accessed on 7/10/2013)

[35] http://en.wikipedia.org/wiki/Tf%E2%80%93idf (Last accessed on 12/26/2013)

ANNEXURE-1

Table 11 List of specified keywords for different change categories

Corrective	BAND_AID ,BUG_FIX, BUG_FIXES, BUG_FIXED, BUGFIX ,BUGFIXES ,BUGFIXED ,BUG_FIXING,BUGFIXING,BUMP,CHECKING,CHECKS,CLEANUP_COMMENT ,CLEANUP_COMMENTS ,COMMENT_CLEANUP,CLEAN ,CLEAN_UP ,CLEANED_OUT ,CLEANING ,CLEANOUT ,CLEANS ,CLEANUP ,CLEANUPS ,CLEARED ,CLEARING ,CLEARS ,CORRECT ,CORRECTED ,CORRECTING ,CORRECTS,DETECT ,FIX ,FIXED ,FIXES ,FIXES_A_COUPLE_OF_MINOR_BUGS ,FIXING ,FIXUPS ,FLUSH ,PATCH ,PATCHED ,PATCHES ,PATCHING
Adaptive	ABORTED ,ABORTS ,ACCEPT ,ADJUST ,ADJUSTED ,ADJUSTING ,ADJUSTMENT ,ADJUSTMENTS ,ALLOCATE ,ALLOCATING,ALLOCATED,ALLOCATES ,ALLOW,ALLOWS ,ALLOWED ,ALLOWING ,ALTER ,ALTERED ,ALTERS,ALTERING ,ALTERATION,BIND ,BLOCK ,BLOCKING,BLOCKS,BLOCKED,COMMIT ,COMMITING ,COMMITS ,COMMITTING ,COMMITTED,COMPARED ,COMPARES,COMPARING ,COMPATABILITY ,COMPATIBILITY ,COMPATIBILTY ,COMPATIBLE ,COMPATIBLITY ,COMPRESS ,COMPRESSED ,COMPRESSION ,CONVERSION ,CONVERSIONS ,CONVERT ,CONVERTS ,CONVERTED ,CONVERTING ,CHANGE ,CHNAGES,CHANGED ,CHANGING ,DEACTIVATED ,DEATIVATES,DEATIVATE,DEACTIVATING ,DEACTIVATION ,DEALLOCATE ,DEALLOCATES ,DEALLOCATED ,DEALLOCATION ,DECOMPRESS ,DECOMPRESSED ,DECOMPRESSING ,DECOMPRESSION ,DECONSTRUCT ,DECOUPLE ,DECRYPT ,DEFINES ,DEFINING ,DEFINED ,DEGRADE ,DEGRADES ,DEGRADING ,DEGRADED,DEIMPLEMENTED ,DISABLE ,DISABLED ,DISABLING ,DISALLOW ,DISALLOWED ,DISALLOWING ,DISALLOWS ,DISCARD ,DISCARDED ,DISCARDING ,DISCARDS ,DISCONNECT ,DOWNGRADE ,DROPS ,DROPPED ,DROPPING ,DUMPED ,ELIMINATE ,ELIMINATED ,ELIMINATES ,ELIMINATING ,ELIMINATION ,ENABLE ,ENABLED ,ENABLES ,ENABLING ,EXCLUDE ,EXCLUDED ,EXCLUDES ,EXCLUSION ,FREED ,FREES ,FREEZE ,FREEZES,FREEZED ,FREEZING ,GET_RID ,IGNORE ,IGNORED ,IGNORES ,IGNORING ,IMPLEMENT ,IMPLEMENTATION ,IMPLEMENTATIONS ,IMPLEMENTED ,IMPLEMENTS ,INITIALIZATION ,INITIALIZE ,INITIALIZED ,INSTALLED,INSTALLING,MAINTAINING ,RELOAD ,RELOADED ,RELOADING ,RELOADS ,RELOCATABLE ,RELOCATE ,RELOCATED ,REVERSE ,REVERSED ,REVERT ,REVERTED ,REVERTING ,REVERTS ,RESCAN ,RESCANNING ,RESCANS,RESCVANNED,RESETS,RESET ,RESETTING ,RESOLVE ,RESOLVED ,RESOLVES ,RESOLVING ,RESTARTING ,RESTORED ,RESTORES ,RESTORING ,RESTRICT ,SET ,SILENCE ,STOPPING ,STOPS ,STOPPED ,SUPPRESS ,SUPPRESSED ,SUPRESS ,SUPRESSING ,SYNC ,SYNCHRONIZE ,SYNCRONIZE ,SYNCRONIZE D,SYNCHRONIZED,TERMINATE ,TERMINATED ,TERMINATING ,TERMINATION ,TOLERATE ,TOLERATED,TOLERATING,TOLERATES ,TRIM

	,TRUNCATE ,TRUNCATED ,TRUNCATING ,UNIFY ,WRAPPED
Perfective	ARRANGE ,AGGREGATE ,AGGREGATES ,BACK_OUT ,BEAUTIFICATION,CODE_BEAUTIFICATION,CREATE ,CREATES,CREATED ,CREATING ,DELETE ,DELETED ,DELETING ,DELETIONS ,DESTROY ,DECREASE ,DECREASED ,DECREASES ,DECREASING ,DECREMENT ,DECREMENTED ,DECREMENTING ,DECREMENTS ,ENCRYPTED ,ENFORCE ,ENFORCED ,EXTEND ,EXTENDED ,EXTENDING ,EXTENSION ,EXTRACTED ,EXTRACTING ,GENERATE ,GENERATED ,GENERATES ,GENERATING ,GROUPED ,INSERT ,INSERTED ,INSERTING ,INSERTION ,INSERTS ,INTEGRATE ,INTEGRATED ,INTRODUCED ,INTRODUCES ,INVENT ,INVOKE ,INVOKES,INVOKING ,MODIFICATION ,MODIFICATIONS ,MODIFIED ,MODIFY ,MODIFYIED ,MODIFYING ,Move, OPTIMIZATION ,OPTIMIZATIONS ,OPTIMIZE ,OPTIMIZED ,OPTIMIZING ,ORDERING ,ORGANIZE ,PREVENT ,PREVENTING ,PULLING ,QUIET ,RE_INCLUDE ,READJUST ,READJUSTMENT ,REALLOCATED ,REALLOCATION ,REANALYSIS ,REARRANGE ,REARRANGED ,REARRANGEMENT ,REARRANGEMENTS ,REARRANGES ,REARRANGING ,REASSIGN ,REASSIGNED ,REASSIGNING ,RECHECK ,RECONNECT ,RECOVER ,REDEFINE ,REDEFINED ,REDESIGN ,REDO ,REDUCE ,REDUCES ,REDUCING ,REFACTOR ,REFACTORED ,REFACTORING ,REFINE ,REFORMAT ,REJECT ,REJECTING ,REJECTS,REMOVAL ,REMOVE ,REMOVED ,REMOVES ,REMOVING ,RENAME ,RENAMED ,RENAMING ,REORDER ,REORDERING ,REORGANIZE ,REPAIR ,REPAIRED ,REPAIRS ,REPARING ,REPLACE ,REPLACED ,REPLACEMENT ,REPLACES ,REPLACING ,REPRODUCE ,RESTRUCTURE ,RESTRUCTURING ,RESTRUCTURED ,RETRIEVE ,RETRIEVES,RETERIEVED ,RETRIEVING ,REMAKE ,REVOKE ,REWORK ,REWORKED ,REWORKS ,REWRITE ,REWRITER ,REWRITES ,REWRITING ,REWRITTEN ,ROLLBACK ,SIMPLIFICATION ,SIMPLIFICATIONS ,SIMPLIFIED ,SIMPLIFIES ,SIMPLIFY ,SIMPLIFYING ,TRANSFORM ,TRANSFORMATION ,TRANSLATE ,TRANSLATED ,TRANSLATIONS ,IMPROVE ,IMPROVED ,IMPROVEMENT ,IMPROVEMENTS ,IMPROVES ,IMPROVING ,IMPROVMENT ,IMPROVMENTS ,INCREASED ,INCREASES ,INCREASING ,INCREMENTED ,INCREMENTING ,INCREMENTS
Preventive	ANALYSIS ,ANALYZE ,AVOID ,AVOIDS ,AVOIDED,AVOIDING ,COMMENT ,COMMENTED ,COMMENTING ,COMMENTS ,FULL_SUPPORT ,INCLUDE ,INCLUDED ,INCLUDES ,REBUILD ,REBUILT ,RECREATE ,REGENERATE ,REIMPLEMENT ,REIMPLEMENTAION ,REIMPLEMENTS ,REINSERT ,REINSTALLED ,REINTRODUCED ,REINVENTED ,REVIEW ,REVIEWED ,REVISE

	,REVISED ,REUSE ,REUSED ,REVISION,RETAIN ,RETAINED ,RETAINING ,RETHINK ,RE-THINK ,RETHINKING ,PROPOSAL,PROPOSED ,PROPOSING,VOTE,VOTES,VOTEING,VOTED,REVOTE,REVOTES,REVOTED,REVOTING
Enhancement	ADD ,ADDED ,ADDIN ,ADDING ,ADDITION ,ADDITIONS ,ADDS ,ENHANCE ,ENHANCED ,ENHANCEMENT ,ENHANCEMENTS ,ENHANCES ,EXPENDING ,EXPENDED ,UPDATE ,UPDATED ,UPDATES ,UPDATING ,UPGRADE ,UPGRADES ,UPGRADING ,READDED ,READDITION