Front. Comput. Sci., 2018, 12(6): 1105–1124
https://doi.org/10.1007/s11704-016-6301-0

RESEARCH ARTICLE

Change profile analysis of open-source software systems to understand their evolutionary behavior

Munish SAINI (✉), Kuljit Kaur CHAHAL

Department of Computer Science, Guru Nanak Dev University, Amritsar 143005, India

Abstract Source code management systems (such as *git*) record changes to code repositories of Open-Source Software (OSS) projects. The metadata about a change includes a change message to record the intention of the change. Classification of changes,based on change messages, into different change types has been explored in the past to understand the evolution of software systems from the perspective of change size and change density only. However, software evolution analysis based on change classification with a focus on change evolution patterns is still an open research problem. This study examines change messages of 106 OSS projects, as recorded in the *git* repository, to explore their evolutionary patterns with respect to the types of changes performed over time. An automated keyword-based classifier technique is applied to the change messages to categorize the changes into various types (corrective, adaptive, perfective, preventive, and enhancement). Cluster analysis helps to uncover distinct change patterns that each change type follows. We identify three categories of 106 projects for each change type: high activity, moderate activity, and low activity. Evolutionary behavior is different for projects of different categories. The projects with high and moderate activity receive maximum changes during 76–81 months of the project lifetime. The project attributes such as the number of committers, number of files changed, and total number of commits seem to contribute the most to the change activity of the projects. The statistical findings show that the change activity of a project is related to the number of contributors, amount of work done, and total commits of the projects irrespective of the change type. Further, we explored languages and domains of projects to correlate change types with domains and languages of the projects. The statistical analysis indicates that there is no significant and strong relation of change types with domains and languages of the 106 projects.

Keywords software evolution, open-source software (OSS), cluster analysis, change classification

1 Introduction

Software systems undergo several types of changes to remain meaningful to their users [1]. These changes pertain to fixing bugs, adding new features, adding support to manage changes in a software system's hardware, software, and business environment, or cleaning/optimizing code. A source code management (SCM) system records metadata about every change (also known as a commit) that is made to the source code of a software project. An SCM system records a commit along with its attributes such as date, time, contributor id, and a textual description. This textual description identifies the intention of the change, such as fixing a bug or addition of a new functionality. Some studies in the past used this commit description to automatically classify changes into change types, such as corrective, adaptive, perfective, and preventive [2–4]. Several previous studies have examined change logs of software systems to understand their change characteristics, but most of them investigate only the distributions of changes types, change size, and change frequency [5–7]. There is no work, to the best of the authors' knowl-

Received June 7, 2016; accepted November 4, 2016

E-mail: munish_1_saini@yahoo.co.in, kuljitchahal@yahoo.com

edge that explores change evolution keeping in mind the different change types such as corrective, adaptive, perfective, etc. The most recent work in this direction by Kemerer and Slaughter [8] dates back to 1999. They point out that there are several patterns that characterize the maintenance (evolution [9]) of a software system. Kemerer and Slaughter used source code metrics unlike the SCM repository metrics used in this study. Gupta et al. [10]studied the change profiles of reusable software components versus software systems reusing them. However, the dataset of their experiment was limited to proprietary systems only. Although Smith et al. [11] targeted open-source software (OSS) projects, the systems that are also the subject of this study, to understand their long-term evolutionary behavior, they analyzed the transitions in size and complexity metrics, and not the SCM repository metrics used in this study.

OSS system development involves volunteers who work from different time zones. All the stakeholders want to know the driving forces behind the evolution of such systems. There is a need to study the OSS evolution from different points of view to better understand the evolutionary behavior of such systems. The present study explores the long-term evolutionary behavior of OSS systems as per different types of changes such as corrective, adaptive, perfective, enhancement, and preventive. The results of this study can provide insights into the evolution of OSS projects from their change profile point of view. The study of change types and its evolution can help project managers and developers in understanding the behavior and pattern of these change types in software projects, i.e., how changes are occurring, which type of change is the most prominent, how changes evolve during the life cycle of a project, and whether there is any correlation between different change types with domains and languages of OSS projects. All stake holders including project managers, developers, and end users can use information related to the change evolution pattern to understand the post implementation activities of the project (such as enhancements to be performed to old features of the project, the addition of new features, or improvements in its performance). When an end user selects a software project for use, they should be able to see the history of change of the software project. How are the changes handled in the past? The kind of changes a software system encounters can indicate the health of the software project (though this is outside the scope of this paper, but we believe that our work creates a base for further discussion in this direction).

This information is also of interest to the researchers and academics as the OSS development paradigm offers an opportunity to understand the software evolution process with the help of the large datasets available in the public domain. What change patterns does an OSS system follow? Can these change patterns be generalized across the problem domains or the development paradigms? These are still open questions.The cluster analysis results provide the insights regarding the change, evolution pattern along the lifetime of OSS projects. Such as corrective, adaptive, and enhancement changes start decreasing after that period, but perfective and preventive changes keep on increasing. As the evolution starts, high and moderate-activity projects are subjected to more enhancement related changes.

A source code contribution (as recorded by an SCM such as *git*) is also defined as a commit [5]. Thus, in the remainder of the paper, we use the terms commit and change interchangeably. Previously, some researchers (such as Gonzalez-Barahona et al. [12], Koch [13], and Lin et al. [6]) have used the number of commits per month as a metric to analyze the evolution of OSS. Gonzalez-Barahona et al. [12] used it on a large OSS project to validate Lehman's laws. Koch [13] performed a coarse-grained analysis of 8,621 projects to identify their evolutionary behavior with respect to their growth rate. In this work, we have used the same metric to measure levels of activity in the OSS projects and to identify their evolution patterns, but from a different point of view. We measure the number of commits per month for a particular change type to find the level of activity for that change type. For example, the number of corrective commits per month indicates the level of activity for the corrective change type only.

Thus, the purpose of this study is to examine the evolution of OSS projects with respect to different change types. We explore answers to the following research questions.

RQ1 What types of changes are made to a software system immediately after it is put to use?

RQ2 How do corrective changes evolve?

RQ3 After how long do the adaptive/perfective changes start?

RQ4 When do the preventive changes start taking place?

RQ5 How do enhancement changes evolve?

RQ6 Do all types of changes diminish over time?

RQ7 How are the OSS projects' attributes, such as number of committers, related to the type of changes performed?

RQ8 Do change types correlate with domains and languages of the OSS projects?

In the light of the above-stated hypotheses, we anticipate that corrective changes to a software system start soon after users begin using the software. Other types of changes may begin a little later. Corrective changes may decrease after some time. Towards the end of this paper, we examine the contribution of other factors in the evolutionary behavior of software systems. We expect that high activity in a change type happens due to a large number of contributors. At the same time, a large number of contributors should make large amounts of contributions.

Project activity is defined for a particular change type based on the number of commits per month [6] for that change type. The results indicate that the change behavior of OSS projects is different for different types of changes. Some change types start with a high level of activity, and then reduce by half in successive months. Some start with very low activity in the beginning, and then increase. Some maintain a relatively high level of activity throughout and some have very low activity with very small variation.Cluster analysis helps to uncover the distinct patterns for each change type. We identify three categories of change activity, based on the number of commits per month for each change type: high activity, moderate activity, and low activity. OSS projects with high and moderate activity for different change types receive maximum changes during 76–81 months of the project lifetime. Corrective, adaptive, and enhancement types of changes start decreasing after that period, but perfective and preventive changes keep increasing. However, many projects have almost consistent low activity in certain change types throughout the observation period. Project attributes such as the number of committers are found to be related with their change activity.

The remainder of the paper is organized as follows. Section 2 gives details of the change classification. Section 3 presents the related work. Section 4 explains the analysis methodology. Section 5 presents the results of the study and discusses and justifies the interpretations. Threats to validity are presented in Section 6. Conclusions of the paper are presented in Section 7.

2 Change classification

Change is essential for a software system to survive in response to changes in the environment. In a software system, changes are made with different purposes such as correcting defects, adding features, and cleaning up the code. Changes to the software may be categorized into different types based on the intention when making them, namely corrective, adaptive, perfective, and preventive. Corrective changes are related to fixing bugs, adaptive changes refer to adding support for managing changes in the environment, perfective changes attempt to improve a system's performance, and preventive changes are made to improve the future maintainability of a software system.

However, most research studies do not agree on a uniform categorization of change types.Mockus and Votta [3]described corrective, adaptive, and perfective as the primary change types. They added another change type, inspection, to include the changes that require formal code inspections. Schach et al. [14] mentioned only corrective, adaptive, and perfective changes. They placed all other types of changes into the 'others" category. Another study gave totally different names to change types such as user support, repair, and enhancement [15]. Even definitions of change types are different across different studies [10].

We consulted the studies of Swanson [16], IEEE [17], ISO/IEC 14764 [18], and the change logs of several OSS systems for deciding the types of changes. We identified corrective, adaptive, perfective, enhancement, and preventive as the change types (Table 1). The corrective changes relate to bug fixes. The adaptive category consists of all those modifications that are performed due to the changing environment such as adjusting of code and features. All the rearrangement activities, such as code embellishment, are classified as perfective changes. The preventive type of changes consists of all those activities that are performed to make future maintenance easier. The enhancement change type was added to detect the new functionality additions and separate them from changes made 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 time in comparison with the age-old concept of software maintenance [9].

This study considers the definitions for the change types as given in Table 1.

Table 1 Change types and their descriptions

Change type	Description
Corrective	Fixing bugs in a software system
Adaptive	Adapting to changes in hardware, software, and business environment
Perfective	Changing a software system for optimization
Preventive	Restructuring and reengineering a system to make modifications easier in future
Enhancement	Addition of new features to a software system

3 Related work

Analysis of change histories of software systems started with the work published by Lientz et al. [19] in 1978. However, their main goal was to analyze the effort distribution in different types of changes. The study stated that 17.4% of maintenance effort was corrective in nature, 18.2% as adaptive, 60.3% as perfective, and 4.1% was categorized as "other." Nosek and Palvia [20] repeated the same experiment and obtained similar results. Schach et al. [14] conducted an empirical study in 2003 and found a significant difference in the results from the previous survey-based studies.

Lee and Jefferson [21] found the distribution of maintenance effort of a web-based Java application similar to that reported by Basili et al. [22] for software developed using a different programming paradigm (FORTRAN and Ada). Sousa and Moreira [23] 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 [24] 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%).

In the1990s, a few studies explored the trends in effort distribution over time in comparison with the earlier work that only took a snapshot view of the effort distribution.

Abran and Nguyenkim [25] performed a trend analysis of maintenance workload distribution for 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. Gefen and Schneberger [26] 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 accession of new functions to the existing applications (improvement phase), and then by the accession of new applications (expansion phase).

Another study investigated changes in maintenance requests during the lifetime of a large software application examined over a 67-month period [10]. They identified 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.

Kemerer and Slaughter [8] explored the evolution of commercial systems based on detailed change events. They used sequence analysis, a social science approach, for a longitudinal study of 23 systems with 25,000 change events spanning 20 years. The study showed that systems pass through several evolutionary phases. However, all systems do not follow the same evolutionary path.

All these studies date back to a time when commercial systems were the only subjects of study. OSS systems were introduced later.

There are few studies that have explored the change evolution in OSS projects from this point of view. Meqdadi et al. [27] studied trends only in the adaptive changes in three OSS projects. As per their analysis, adaptive changes decrease over time. Most of the change analysis studies in OSS projects investigate only the distributions of changes types, change size, and change frequency [5–7].

In this paper, the evolution of a large set of OSS projects is analyzed to explore trends in different types of changes over time. This work analyzes the evolution of 106 OSS projects to identify trends in different change categories.The trend analysis explicitly indicates the pattern in which a particular change type evolves.

In the context of this work, software change classification may also be of interest to the readers. Swanson [16] gave the first and the most commonly used classification. Mockus and Votta [3] suggested an automated method of classifying software changes based on their textual descriptions. They provided a classification of changes based on specified keywords in the textual abstracts of changes into three primary categories (adaptive, corrective, and enhanced) and introduced another category (inspection maintenance). Hassan [4] extended the work of Mockus and Votta [3] to classify change messages. They presented an automated classification of change messages in OSS projects, and classified the change messages into mainly three types as bug fix, feature introduction, or general maintenance changes. In the present study, we have extended the work of Mockus and Votta [3] by introducing more relevant change types by consulting the studies of Swanson [16], IEEE [17], and ISO/IEC 14764 [18], introducing a greater number of keywords (listed in Table 2). The concept of term frequency–inverse document frequency (tf-idf) [28] was used to eliminate irrelevant words.

Kim et al. [29] used a machine learning classifier to determine the type of change and classify it as either buggy or clean change. Lehnert and Riebisch [30] discussed the change classifications, but restricted their study to fine-grained changes only. Chaplin et al. [31] discussed the types of software evolution and software maintenance. They dis-

Table 2 List of keywords for different change types

Change type	Keywords
Corrective	bandaid, bug, bug fix, bump, check, clean, cleanout, cleanup, clear, correct, detect, fix, fixup, flush, patch
Adaptive	abort, accept, adjust, allocate, allow, alteration, bind, alter, block, commit, compare, compatible, compress, compression, conversion, convert, change, deactivate, deactivation, deallocate, deallocation, decompress, decompression, deconstruct, decouple, decrypt, define, degrade, deimplemented, disable, disallow, discard, disconnect, downgrade, drop, dumped, eliminate, elimination, enable, exclude, exclusion, freed, frees, freeze, get rid, ignore, implement, implementation, initialization, initialize, install, maintaining, reload, relocatable, relocate, reverse, revert, rescan, reset, resolve, restore, restrict, set, silence, stop, suppress, sync, synchronize, terminate, termination, tolerate, trim, truncate, unify, wrapped
Perfective	arrange, aggregate, beautify, back out, beautification, create, delete, code beautification, deletions, destroy, decrease, decrement, encrypt, enforce, extend, extension, extract, generate, group, insert, insertion, integrate, introduce, invent, invoke, modification, modify, move, optimization, optimize, ordering, organize, prevent, pull, quiet, reinclude, readjust, reallocate, reallocation, reanalysis, rearrange, reassign, recheck, reconnect, recover, redefine, redesign, redo, reduce, refactor, refine, reformat, reject, remove, rename, reorder, repair, replace, reproduce, restructure, retrieve, remake, revoke, rework, rewrite, rollback, simplification, simplify, transform, translate, improve, increase, increment
Preventive	avoid, comment, include, proposal, proposed, protect, rebuild, rebuilt, recreate, regenerate, reimplement, reinsert, reinstall, reintroduce, reinvent, retain, rethink, reuse, review, revise, revision, revote, vote
Enhancement	add, addin, addition, enhance, enhancement, update, upgrade, expend, readd, readdition

cussed the classification of software maintenance activities for practitioners, managers, and researchers. They used a clustering method to combine various activities into clusters.

4 Analysis methodology

This section describes the approach, methods, and tools we used in this study. A stepwise description of the whole process (Fig. 1) is as follows.

1) Data collection: identify the SCM repository of the OSS projects and retrieve the commit history of each project.
2) Change classification: analyze commit records of the OSS projects using an automated keyword-based categorization technique to categorize the commits into various types such as: adaptive, corrective, perfective, enhancement, and preventive.
3) Data smoothing: remove noise from the dataset to identify patterns.
4) Cluster analysis: classify the projects to indicate the evolution pattern with respect to the level of activity in a particular change type.

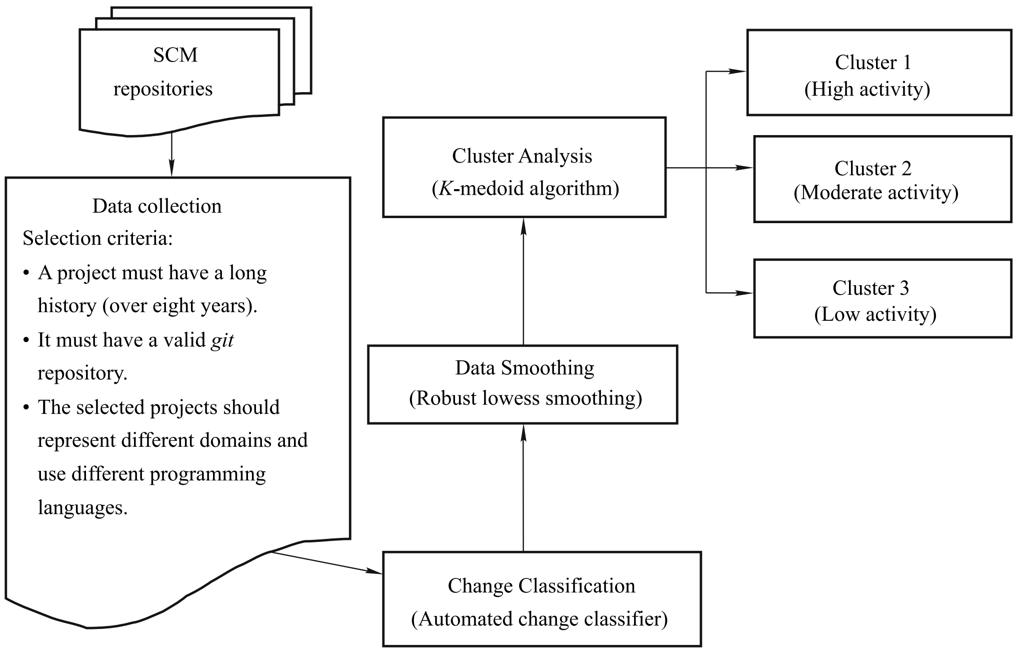
SCM repositories
Data collection
Selection criteria:
• A project must have a long history (over eight years).
• It must have a valid *git* repository.
• The selected projects should represent different domains and use different programming languages.
Change Classification (Automated change classifier)
Data Smoothing (Robust lowess smoothing)
Cluster Analysis (*K*-medoid algorithm)
Cluster 1 (High activity)
Cluster 2 (Moderate activity)
Cluster 3 (Low activity)

Fig. 1 Data analysis methodology

4.1 Data collection

git is a distributed version control system used for software development. It was developed by Linus Torvalds for Linux Kernel with the priorities of speed,data integrity,and support for distributed, non-linear workflows. It is gaining popularity nowadays due to its various features such as cheap local branching, convenient staging areas, and multiple workflows. Unlike other distributed version control systems, *git* provides a complete repository on the client machine with entire history and version-tracking capabilities, independent of network access or a central server.

We selected 106 OSS projects for analysis purpose using the following selection criteria:

- The project must have a long history (over eight years);
- The project must have a valid *git* repository;
- The projects should represent different domains and use different programming languages.

We first cloned official *git* repositories of the OSS projects and used *gitbash* to fetch the change history of each software project. Further, a Javascript was developed to extract commit messages of each of the software projects.

The change history information, extracted here, contains commits that are unstructured and are not inconsistent format.An unstructured and inconsistent commit message is one that does not follow the committing standards. In the change logs, some of the commit messages were blank. In some cases, contributors entered unusual text as shown in the following example:

1. Okay...*Last* commit, now to create a release..
2. At this rate, maybe next year sometime I'll get
3. This done...
4. Slow work...
5. ...Sam",
6. It's coming
7. I'm getting there, slowly :)

Therefore, the collected data required cleaning to remove all those commits entries that contained blank or unusual text.

The cleaning process is semi-automatic. The cleaning process starts with executing the Java code on the change history of 106 OSS projects to remove commit messages that are blank or specified empty in the change log such as

"***** empty log message*****"

In the second step, the change history of the OSS projects was manually analyzed by both authors individually to identify and remove unusual commit messages. Finally, the individually manipulated change history was cross-checked by both authors to find any discrepancy in the removal of unusual commits from the change history. This double manual evaluation approach eliminates the chances of missing any unusual change message.

Complete project related details of the 106 OSS projects such as project name, project start date, total commits (before cleaning), percentage of commits discarded (blank or unusual text commits), total commits classified by automated classifier, number of lines inserted, the number of lines deleted, and total number of files changed is presented in Table 2A of Appendix. In this study, we have classified the projects according to their domain (by consulting the taxonomy in [32]) and the development languages.

4.2 Automated change classifier

The change history of all the OSS projects was manually analyzed by both authors individually to prepare alist of significant keywords (in accordance with the definitions of the change types). For example, *fixes* and similar terms such as *fixed* and *fixing* indicate corrective action. Similarly, keywords such as *adjust* and *allocates* indicate that adaptive action is performed. Finally, the individual keyword lists were cross-checked by both authors to find any discrepancy in categorization of particular keywords in the specified change type or any keywords missed by one but identified by the other. This approach reduced the chances of missing or misclassifying keywords. Further, to generalize the list of keywords, we used the WordNet to group keywords (such as *fix, fixed, fixing, fixes* into one keyword *fix*). This reduced the number of keywords in the list. Finally, the concept of tf-idf [28] was used to eliminate all those keywords that are irrelevant in the context of change classification. tf-idf is a numerical factor that reflects how important a word is to a document. It assigns a weight to a keyword based on its importance in the document. The complete list of keywords is given in Table 2.

The change messages were then analyzed and categorized into change types by using an automated classifier developed in this research study. This automated classifier is implemented in Java. The complete process of classifying the change types is explained in our previous work [33].

The specific criteria for assigning the change type to a commit message are as follows.

- Look for all the specified keywords in a commit record and measure the frequency of each keyword.
- Look for the change type in which these keywords lie.
- The keyword with the highest frequency will decide the change type. For example, if in a commit message, the keyword *adjusted* is the most frequent, then it means this commit belongs to the adaptive change type.
- In case two keywords from different change types have the same occurrence frequency, then the keyword that occurs first in the commit message is considered as decisive.
- A commit record that does not contain the specified keywords is skipped.

The automated classifier classifies the changes into various change types (as adaptive, corrective, perfective, enhancement, and preventive) forthe OSS projects.

For validation of the automated classifier, we used the approach as mentioned in [3,34]. We consulted four experts:two software developers, one professor, and one PhD researcher, working in the field of OSS development. We asked them to manually classify commit records of OSS projects into particular change types by using the same categorization criteria as used for the automated classification. Then Cohen's Kappa test [35] was used to evaluate agreement between the manual and the automated change classification. This test measures the degree of agreement between two raters. In Tables 3–6 the manual (by different experts) and the automated classification of commits of the first version of GNUCashv1.3 is shown. The rows and columns of these tables specify the manual and automated categorization data, respectively. Finally, the Kappa coefficient is calculated by using the manual and automated classification data.

Table 3 Comparison of Expert 1 classification with automated classification of GNUCash version 1.3

Expert 1	Automated classification				
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

The calculated Kappa coefficients for all the projects lie between 0.5 and 0.8. This indicates the presence of substantial *agreement* between the automated and the manual change 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. For more details on the validation process, refer to our previous work [33].

Table 4 Comparison of Expert 2 classification with automated classification of GNUCash version 1.3

Expert 2	Automated classification				
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 5 Comparison of Expert 3 classification with automated classification of GNUCash version 1.3

Expert 3	Automated classification				
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 6 Comparison of Expert 4 classification with automated classification of GNUCash version 1.3

Expert 4	Automated classification				
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

For measuring the classification accuracy of the automated classifier, we used the k-nearest neighbor (KNN) method [34]. The change classified into a particular change type (by the automated classifier) is used as a categorical dependent variable along with the change message as the categorical predictor variable. For the KNN algorithm, we use random sampling 70% (training set) and 30% (testing set), Euclidean distance measure, v-fold cross-validation (assumed $v = 10$). For details on how to choose the value of k and v, we have consulted the literature [34,36]. The computed cross-validation accuracy along with the Kappa coefficient (for predicted and observed values of change types) for the OSS projects is shown in Table 7.

4.3 Data transformation

After classification, the change type data is converted into numerical form by measuring the number of commits per month for each change type. We used the metric number of commits per month (for every change type) to represent this measurement [11]. However when plotted, metric values are found to have frequent fluctuations (Fig. 2 shows corrective changes data for the PostgreSQL OSS project). It is not useful to use this data as such to ascertain any evolutionary patterns. We used the *Robust Lowess* smoothing [37] to smoothen the number of commits (of a particular change type) per month of each project.This converts the dataset (number of commits for the month of a change type) into a functional form.The smoothed curve along with actual data curves for one of the projects (PostgreSQL) for corrective change type is shown in Fig. 2. The dotted curve represents the actual number of commits data for the OSS project, whereas bold curve indicates the smoothed curve.It can be easily seen that finding evolution patterns without a smoothing curve is not possible as the original curve shows frequent spikes with large variation, whereas the bold line indicates the clear pattern.

Table 7 Classification accuracy of the automated classifier

OSS project	Cross-validationaccuracy/%	Kappa coefficient
PostgreSQL	91	0.93
WordPress	84	0.87
GNUCash	86	0.89
php-src	92	0.94
MySQL	89	0.91

4.4 Clustering

Clustering is the process of grouping a set of entities in such a way that entities in a group (cluster) are more similar (related) to each other than entities not in the group.In this study, cluster analysis has been applied to find patterns in the evolution of OSS projects. Clustering employs a variety of approaches that differ by the notion of what constitutes a cluster and how to find a valid set of clusters efficiently. Some of the most popular notions include finding groups with smallest distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions [38]. In this study, we use a distance-based clustering approach to perform the cluster analysis. *K*-means and *K*-medoid [39] are the two most prominent distance-based clustering approaches. *K*-means defines the mean of the data points where as *K*-medoid starts with the most representative point. We use the *K*-medoid algorithm since it provides a more robust estimate of a representative point than the mean as used in the *K*-means algorithm.Moreover, the *K*-medoid algorithm handles extreme values (outliers) better than the *K*-means [40]. *K*-medoid is formally described in the following algorithm.

```
Algorithm  KMedoids($D, K, \mathrm{Dis}$)
  Input: data $D \subseteq \mathscr{X}$; number of clusters $K \in \mathbb{N}$; distance metric Dis: $\mathscr{X} \times \mathscr{X} \to \mathbb{R}$
  Output: $K$ medoids $\mu_1, \mu_2, \ldots, \mu_K \in D$, representing a predictive clustering of $\mathscr{X}$;
1 randomly pick $K$ data points $\mu_1, \mu_2, \ldots, \mu_K \in D$;
2 repeat
3     assign each $\mathbf{x} \in D$ to $\mathrm{argmin}_j \mathrm{Dis}(\mathbf{x}, \mu_j)$;
4     for $j = 1$ to $k$ do
5         $D_j \leftarrow \{\mathbf{x} \in D | \mathbf{x}$ assigned to cluster $j\}$;
6         $\mu_j = \mathrm{argmin}_{\mathbf{x} \in D_j} \sum_{\mathbf{x}' \in D_j} \mathrm{Dis}(\mathbf{x}, \mathbf{x}')$;
7     end
8 until no change in $\mu_1, \mu_2, \ldots, \mu_K$;
9 return $\mu_1, \mu_2, \ldots, \mu_K$;
```

Software projects may follow different evolutionary paths based on their commit activity. As all the projects cannot be presumed to follow a single pattern, we expect to find similar evolution patterns only for projects in the same cluster.

5 Results and analysis

This study explored the evolution patterns of 106 OSS projects to understand their change behavior with respect to different types of changes. The change types are identified as corrective, adaptive, perfective, preventive, and enhancement. Functional forms for every project are created using a number of commits per month metric for each change type [35].

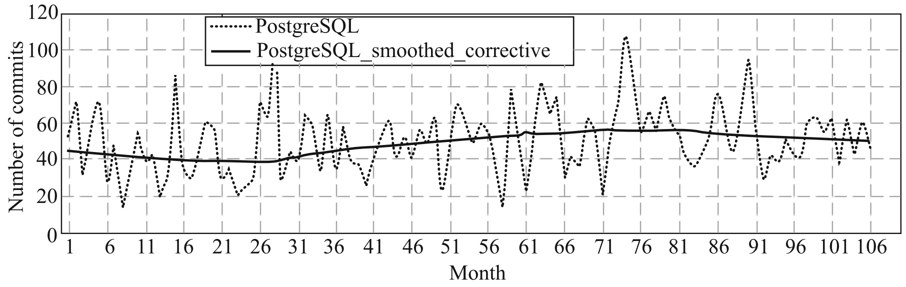

Fig. 2 Smoothed data for corrective change type of an OSS project

We proceed by analyzing the curves to explore the underlying evolution pattern. We begin with a mean curve analysis (in Section 5.1) of each change type. (It provides an overall average evolution trend of each change type). There is a lot of heterogeneity involved in the dataset. The change evolution is quite heterogeneous across different projects. Therefore, we apply the cluster analysis to assort the projects according to the commit activity for each change type (in Section 5.2).

5.1 Mean curve analysis

Figure 3 shows the mean curve (in bold) along with plots of functional forms for all the change categories of all OSS projects considered in this analysis. The mean curve estimates the average number of commits for each month for 106 OSS projects. We choose first 106 months from the life of each OSS project to analyze the evolution of change types in all the OSS projects. The reason for choosing first 106 months is that we want to understand the evolutionary path of each change type: how changes of different type evolve from the start of a project, i.e., which changes are more prominent at the start of a project and how they evolve over time. Mean curves in all the five change categories show an increasing trend (cf. Table 8). They all stabilize after a period of time. We can observe that corrective changes (Fig. 3(a)) stabilize earliest, after approximately 60 months. We use the moving average analysis to find the threshold after which a change type starts to stabilize. Adaptive changes stabilize after nearly 80 months. Perfective changes stabilize even later, after nearly 85 months. Preventive changes show a steady and slow increase. The last increment happens around the 100th month. Enhancement changes stabilize around the 80th month. This shows that different change types follow different patterns.

Table 8 Trend analysis of change types for the observation period

Change type	Regression equation	Trend
Corrective	$y = 12.77 + 0.178x$	Increasing
Adaptive	$y = 4.75 + 0.099x$	Increasing
Perfective	$y = 4.389 + 0.11x$	Increasing
Preventive	$y = 0.76 + 0.021x$	Increasing
Enhancement	$y = 12.26 + 0.088x$	Increasing

The curve plots also show that there is variation in the change evolution of various projects. Observe the pattern of corrective changes in Fig. 3(a). Some projects start with high levels of commit activity, say 100 commits per month, and then reduce by half in successive months. Some start with a very low activity in the beginning, e.g., less than 20 commits/month, and then increase to more than 200 commits in a month. Some projects maintain a relatively high commit activity throughout and some have very low commit activity with very small variation. Other change types, as shown in Fig. 3, also reveal similar patterns. The mean curve does not represent the true picture as it seems to cancel the change dynamics of different projects.

This heterogeneity in the commit activity may not lead to any common understanding of the change patterns. To overcome this, cluster analysis is performed that will group the projects according to the level of activity in different change types. The projects in a group may follow the same evolutionary paths and help in better understanding of the change evolution.

5.2 Curve clustering

There is a lot of variation in the evolution of the change types of OSS projects. Some projects receive a large number of commits in the initial months and the number decreases drastically after that. While others see a reverse trend: a few commits in the initial months and an exponential rise after a few months.

Some projects maintain a high commit activity throughout, while others receive only a few commits in the whole observation period. This heterogeneity for each change type is handled by using the cluster analysis approach. Cluster analysis group projects in such a way that the projects in the same cluster are more related to each other with respect to their level of activity for each change type in comparison with projects in other clusters [41].

We use the K-medoid algorithm [42] to perform the cluster analysis. The K-medoid algorithm outperforms K-means in handling the extreme values. It partitions n observations into k set of clusters, where each observation belongs to a cluster with minimum within-cluster dissimilarity. We use the *Manhattan distance* as the distance measure. The Manhattan distance is based on absolute value distance between two points:

$$\text{Manhattan distance} = \sum_{i=1}^{k} |x_i - y_i|. \tag{1}$$

The exact value of k is evaluated by using the *silhouette* coefficient [42]. The silhouette coefficient of a data point represents how closely it matches the data within its cluster [42]. The average silhouette coefficient of a cluster is the average of silhouette coefficients of all the data points in a cluster. The average silhouette coefficient helps in deciding the appropriate number of clusters (i.e., k). An average silhouette coefficient closer to 1 implies that the data points are in the

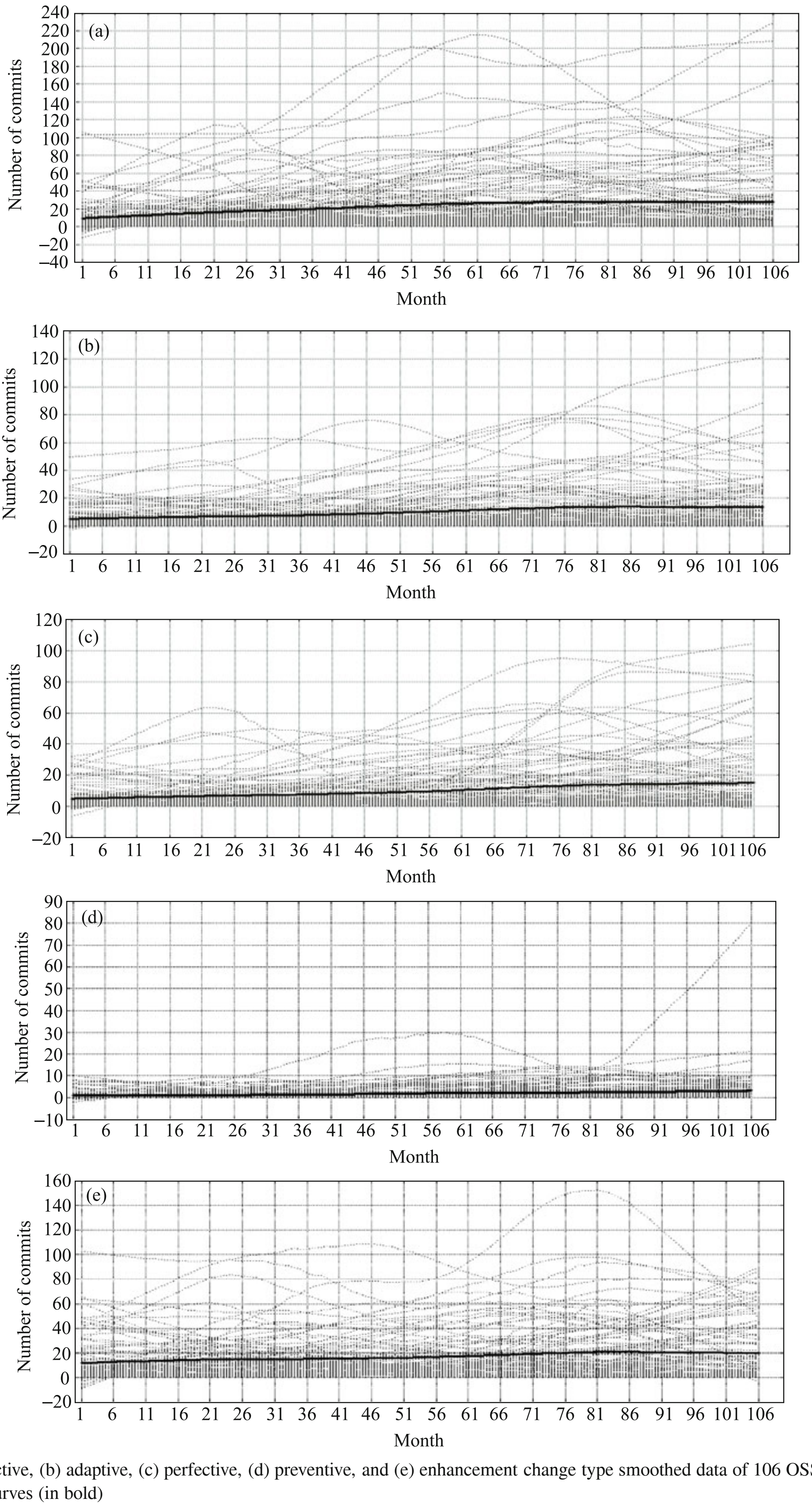

Fig. 3 (a) Corrective, (b) adaptive, (c) perfective, (d) preventive, and (e) enhancement change type smoothed data of 106 OSS projects along with their mean curves (in bold)

appropriate cluster, whereas if the coefficient is close to –1, this signifies that the data points are in the wrong clusters. For more details on how to find the exact number of k (number of clusters), please refer to [42,43].

We start the clustering process by considering only two clusters, then we keep on incrementing the number of clusters by one until we obtain the exact value of k (identified by an average silhouette coefficient that is closer to 1). Table 9 specifies the corrective change along with the average silhouette measure for the respective number of clusters. It points out that, in the corrective change type, the average silhouette coefficient is maximum when we have the number of clusters as 3 ($k = 3$). Similarly, the process is repeated for each change type to find the exact number of clusters.

Table 9 Average silhouette of the OSS projects for corrective change type

Change type	Clusters	Average silhouette coefficient
Corrective	2	0.0514
Corrective	**3**	**0.704**
Corrective	4	0.651
Corrective	5	0.463

Table 10 shows the number of clusters and the number of projects put in different clusters for all five categories of change types. All change types, except preventive, have commit patterns falling into three clusters. Commits related to preventive changes fall into only two clusters.

Table 10 Cluster analysis of the OSS projects

Change type	Clusters	Number of projects in each cluster	
		1	20
Corrective	3	2	39
		3	47
		1	14
Adaptive	3	2	36
		3	56
		1	19
Perfective	3	2	33
		3	54
		1	22
Preventive	2	2	84
		1	17
Enhancement	3	2	54
		3	35

Table 10 also indicates that the numbers of projects in clusters (high, moderate, or low) of different change types are not the same. A project having one change type (for example, corrective change) identified to be in a *high*-activity cluster (cluster consisting of projects with high activity identified by applying cluster analysis) may or may not have all other change types falling in the same cluster of activity. They may be in high, moderate, or low activity depending on the particular change type activity performed in that project. However, a detailed analysis of the projects activity has shown that 52% of the projects in the dataset have all the change types falling in the same cluster (may be high, moderate, or low depending on the change activity). For 12% of the projects, there is no pattern (few change types fall into high activity, while others may be in low or moderate activity and there is as such no pattern in their occurrence for each change type). For the remaining 36% of projects, there exist patterns in the occurrence of all change types into a particular change activity cluster. In the future, we aim to explore this 36% of projects for finding the reason why these projects have resemblance in terms of change activity pattern.

Figure 4 shows the cluster solutions for different change types. We compared the evolution of the projects in different clusters for each change type.

The bold curve in Fig. 4 indicates the overall *mean*, whereas dotted curves represent the cluster means for high-, moderate-, or low-activity projects. In the case of corrective changes (Fig. 4(a)), a comparison of cluster mean curves indicates that projects in cluster 1 start with a high number of commits per month in comparison with projects in the other two clusters. Cluster 3 curves show that monthly corrective commits in this cluster follow almost a flat straight line. Projects in cluster 2 see a gentle increase in the number of corrective commits per month before they stabilize after 81 months. Cluster 1 projects observe maximum increase from 22 corrective commits to 92 corrective commits per month. There is a slight decrease after 76 months in number of corrective commits per month for the projects in this cluster.

In the case of adaptive changes (Fig. 4(b)), projects in cluster 1 observe a sharp increase from 20 adaptive commits in the beginning to 56 commits in the 81st month. After this there is a slight decrease. Projects in cluster 2 show a sublinear increase with maximum number of adaptive commits per month in the 80th month. There is a gentle decrease after that. There is only a small variation in projects in cluster 3 of adaptive commits.

Perfective changes (Fig. 4(c)) in cluster 1 follow a sharp increase in the first 81 months and stabilize after that. Cluster 1 represents projects with a prominent increase in perfective changes for the first 60 months and then again after 81 months, being nearly flat between months 60 and 81. However, projects in cluster 2 see a gentle increase in perfective changes throughout. In this change type as well, there is one group (cluster 3) of projects that do not see any significant variation in perfective changes.

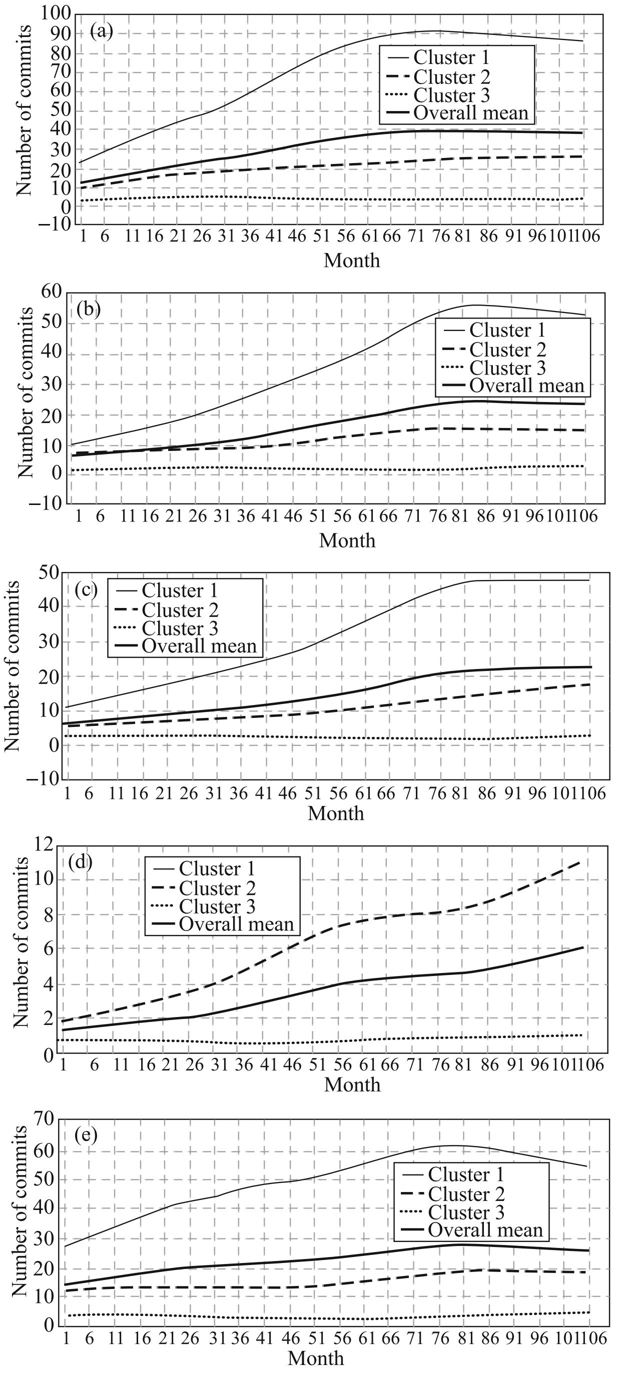

Fig. 4 Cluster mean curves for (a) corrective, (b) adaptive, (c) perfective, (d) preventive, and (e) enhancement change types

Preventive changes (Fig. 4(d)) have only two clusters (evaluated by measuring the average silhouette of the clusters). Cluster 1 represents projects having 2–11 preventive commits per month and keep on increasing throughout the observation period. However, projects in cluster 2 follow almost a flat straight line with very low commit rate (1–2 commits per month).

Enhancement changes (Fig. 4(e)) also have cluster 1 with projects having 27–62 enhancement related commits per month in the initial 80 months and a sharp decline after that. Cluster 2 has projects with the number of enhancement commits per month going up to 18 per month in the first 80 months and stabilizing after that. In this change type as well, there is a cluster (cluster 3) of projects with little variation in the number of enhancement commits per month in the observation period.

In light of the research questions discussed in the introduction, this analysis indicates that there is an activity for all the change types from the beginning of the OSS projects. We may leave aside clusters with low activity in change types, as their activity remains almost constant in the observation period. In the case of high-activity clusters of the change types, the enhancement-related changes are the most dominant to begin with, and not the corrective changes as assumed in the introduction. However, corrective changes increase at a higher rate and achieve their maximum value well before other change types. Adaptive and perfective changes both start at the same level and achieve their maximum values around the same time. Preventive changes are the least frequent, but they keep on increasing throughout the observation period, unlike the other change types. This shows that, in the beginning, there is less pressure to adapt or make a system perfective. Adding more features demands attention in the beginning. Corrective actions also keep on increasing and stabilize before all other change types. At a certain time, adaptation pressure decreases (86 months in this case) but pressure to make the system perfect does not. In fact, three kinds of changes start decreasing at the same time (80–86 months): corrective, adaptive, and enhancement. Perfective and preventive changes do not stop and continue to show an increasing trend.

In moderate-activity clusters, enhancement changes are more than all other types of changes. Corrective changes cross the ten commits per month mark after six months. Adaptive changes cross the same mark after 45 months, and perfective changes even later, after 56 months. Preventive changes cross the same mark very late, after 96 months. In this case, changes in all the categories maintain a stabilized level long into the observation period.

5.3 Understanding the contribution of other factors

This section addresses the research question 7 as mentioned in the introduction. It explores the relationship between the change activity of a project and other project attributes such as the number of committers, total number of commits, and amount of work (in terms of number of files changed).

5.3.1 OSS projects attributes and metrics

In addition to commits, SCM tracks many other attributes of an OSS such as the number of committers, number of files changed, and number of total commits. In this section, we examine the contribution of these attributes in the commit activity of projects in different clusters. In this section, we compare the clusters for each change type based on these three attributes of the OSS projects. We use three metrics to measure these attributes: Committers_Count, Files_Changed, and Total_Commits. These metrics are collected monthly. The Committers_Count metric is the number of committers per month who contributed to the code repository of a project as recorded by the SCM system. It approximates the person effort in months. To understand the amount of work done, we measure the number of files changed per month using Files_Changed metric. Finally, the Total_Commits metric gives the overall picture of the contributions made to a project in a month. In this section, we examine the relationship between commit activity levels of the OSS projects for the change types and other project-related attributes. For every change type, we compare the clusters of projects with different levels of activity on the basis of these project-related attributes.

5.3.2 Comparison of clusters within a change type

Table 11 gives summary statistics for clusters of each change type for different attributes of the projects.

A comparison of Committer_Count (the number of committers per month), across the clusters for each change type shows that number of committers per month is comparatively large for projects with high activity for the corresponding change types, and least for low-activity projects. The result is similar for the other two metrics:Files_Changed (number of files changed per month) and Total_Commits (total commits per month).

This shows that the change activity of a project is related to number of contributors, amount of work performed, and total commits of the projects irrespective of the change type. Large numbers of contributors not only make a large number of commits, but also handle a large number of files. Therefore, the volume of work in these projects is also high. It is not that large numbers of contributors are making small contributions. Whereas in clusters with low activity in change types, it is not only that there are fewer contributors,but the volume of the work that they contribute is also small. Therefore, activity may be less due to a small number of contributors, fewer contributions, and less work performed in a contribution.

Table 11 OSS project attributes and related statistics

Category	Cluster	Statistics	Committers count	Files changed	Total commits
Corrective	1	Mean	18.04	1244.30	69.27
		SE	0.76	45.06	2.21
	2	Mean	6.17	544.32	20.38
		SE	0.24	22.04	0.45
	3	Mean	2.83	252.19	3.97
		SE	0.06	8.74	0.04
Adaptive	1	Mean	21.43	1377.73	36.11
		SE	0.90	66.25	1.57
	2	Mean	7.33	676.71	11.83
		SE	0.30	17.69	0.29
	3	Mean	3.05	253.99	2.42
		SE	0.07	11.88	0.03
Perfective	1	Mean	19.42	1258.56	31.55
		SE	0.79	52.08	1.26
	2	Mean	6.43	670.89	10.72
		SE	0.27	26.40	0.37
	3	Mean	2.84	215.29	2.33
		SE	0.06	7.92	0.03
Preventive	1	Mean	16.28	1120.64	6.29
		SE	0.62	45.25	0.27
	2	Mean	4.48	404.89	0.74
		SE	0.16	10.73	0.01
Enhancement	1	Mean	19.40	1363.54	49.98
		SE	0.73	55.96	0.97
	2	Mean	5.75	475.43	15.45
		SE	0.21	12.19	0.25
	3	Mean	2.69	268.67	3.35
		SE	0.099	17.51	0.061

5.3.3 Comparison of clusters across change types

However, when we compare clusters across the change types (as listed in Table 11), e.g., clusters with high activity for all the change types, the clusters differ with respect to project attributes. We do not see a homogenous trend as discussed in the previous section. For every first cluster, projects with high activity in adaptive changes have a maximum number of contributors followed by perfective, enhancement, corrective, and preventive change types in that order. As far as the amount of work performed is considered (measured by Files_Changed metric), projects with high activity in adaptive changes are followed by enhancement, perfective, corrective, and preventive types in that order. Projects with high activity in enhancement change type have fewer contributors than the projects with high activity in perfective change type, but have more work done (the number of files changed is greater). Projects with high activity for corrective change type lead in the total number of commits they receive. This shows that high activity of corrective changes is easy to manage. Adaptation in such projects requires more resources (i.e., person months).

In clusters with moderate activity, Commiter_Count and Files_Changed are more for adaptive and perfective change types in comparison with corrective and enhancement types of changes.

There is an interesting observation in clusters with low activity across different change types. In the preventive change type, the cluster has the maximum number of contributors as well as the highest number of files changed per month. In the enhancement change type, the cluster has the fewest contributors in comparison with the corresponding clusters of corrective, adaptive, and preventive change types, whereas the number of files changed in this cluster of the enhancement change type is the maximum (as in moderate-activity projects that require greater expertise). This shows that a small group of contributors is making large contributions in the enhancement change type. In the case of the corrective change type, there is a reverse situation. The number of contributors is greater, the number of contributions is greater (total commits), but the number of files changed is lower. This shows that a large number of contributors are making small-sized contributions. It seems that a core group is making enhancement changes, and peripheral contributors are making corrective changes.

5.4 Correlation of change types with domains and languages of OSS projects

This section focuses on the relationship (if any) between change types and the domains of the OSS projects. The OSS projects in this study belong to several different domains. We also explore the association between change types and languages of the OSS projects.

5.4.1 Change types and domains

The OSS projects are manually classified into different domains (as listed in Table 1A of Appendix) by consulting the taxonomy of software types to facilitate search and evidence-based engineering by Forward and Lethbridge [32]. Table 12 shows the domains and number of OSS projects belonging to that domain. To find the correlation between change types with domains of the OSS projects we use the chi-squared test [44]. The strength of the relation is measured by calculating Cramer's V statistic hat is based on Pearson's chi-squared statistic. The strength measures are only used in cases where chi-squared or likelihood ratio significance values (p-value) are lower than 0.05 (i.e., level of significance).

Table 12 OSS project domains

Domain	Number of projects
Accessibility	2
Administrator software and tools	1
Artistic creativity/Eentertainment and education	2
CAD/CAE tools	1
Communication and information	2
Compilers	1
Database	3
Development	7
Development environment	19
Device/Peripheral drivers	1
Diagnostic/Process viewer/Activity monitor	3
Entertainment and education	11
Graphics packages/Rendering engines	7
IM servers	2
Information management and decision support systems	3
Information resources	5
Interoperability infrastructures	1
Kernels/Distributions	4
Load balancers	1
Modeling/Case tools	1
Networking/Communications	4
Personal management	2
Productivity and creativity	3
Reporting	1
Security	2
Server	1
Simulation software	1
Software testing tools	1
Storage	1
Strategic and operations analysis	2
UI support software	1
Web applications/services	7
Web/FTP/Content servers	3

For performing the evaluation, we assume the null hypothesis that there exists no relation between the change types and domain of 106 OSS projects. In other words, we assume that each change type activity (classified as high, moderate, or low activity by using cluster analysis) has no relation with the domain of a OSS project. We start the analysis by consulting all the assumptions (simple random sample, sample size, expected cell count, etc.) of the chi-squared test [44]. All the assumptions are found to hold on our dataset except the "expected cell count" assumption (cell count should not be less than 5) that was violated as we have domains whose actual count is less than 5. Therefore, to overcome this we referred to the literature [44] that indicates that the Yates correction (applied on the dataset) or other statistic measures (such as the likelihood ratio) should be considered to evaluate the relationship.

We start the statistical analysis by considering all the OSS projects that have domain count less than 5. The estimated significant values (p-valued) obtained after applying the chi-squared test on the change types and all domains are listed in Table 13. The "expected cell count" assumption is violated here as some domains have a count less than 5. Therefore, in this case, we look for a likelihood ratio significance value (p-value) for each change type. In the corrective change type, the likelihood ratio ($p = 0.048$) is less than the level of significance (0.05). This causes the rejection of the null hypothesis and suggests the existence of a relation between corrective changes of an OSS project and its domain. The strength of the relation is measured with Cramer's V statistics by referring to the level of relationships interpretation chart. This indicates the presence of a weak relationship between corrective changes and domains of the OSS projects. In all other change types,the likelihood ratio p-value is greater than the level of significance. It specifies that no other change types have any significant relation with the domain of the OSS projects.

We refined the dataset to consider only those domains in which the number of projects is greater than 5. Table 14 demonstrates that the Pearson chi-squared value and likelihood ratio value for each change type is greater than the level of significance (0.05). This shows the acceptance of the null hypothesis (i.e., there is no relationship between change types and domains).

To find the significant relation of change types with domains of the OSS projects, we select he projects having the domain"Development environment" and "Entertainment and education" from our dataset as they have a higher count for the number of projects with this domain (Table 12).

Table 15 lists the estimation of the chi-squaredtests.It specifies that the Pearson chi-squared value and likelihood ratio values are greater than the level of significance (0.05). It causes the acceptance of the null hypothesis (i.e., there is no relationship between change types and domains).

The statistical evaluation for finding the relation between domains and change types of the 106 OSS projects suggested that there exists no strong relation between the change types with the domains of the OSS projects.

5.4.2 Change types and project languages

The collected data of 106 OSS projects have diversity in terms of languages used for the development of projects. Table 16 shows the languages and the number of projects that are developed using these languages. Most of the projects are developed using C, C++, PHP, and Python. To evaluate the correlation of change types with the languages of OSS projects, we assume the null hypothesis that there exists no relation between the change types and languages of the OSS projects. In other words, we assume that each change type activity (classified as high, moderate, or low activity by using cluster analysis) has no relation with the languages used for the development of the OSS projects.

Table 13 Correlate change type with domain

		Corrective	Adaptive	Perfective	Preventive	Enhancement
Chi-squared test	Pearson chi-squared	0.164	0.14	0.316	0.288	0.188
	Likelihood ratio	**0.048**	0.051	0.076	0.192	0.096
Symmetric measures	Cramer's V	**0.164**	0.14	0.316	0.288	0.188

Table 14 Correlate change type with domain (count is not less than 5)

		Corrective	Adaptive	Perfective	Preventive	Enhancement
Chi-squared	Pearson chi-squared	0.468	0.218	0.452	0.883	0.555
test	Likelihood ratio	0.355	0.103	0.357	0.752	0.435
Symmetric measures	Cramer's V	0.468	0.218	0.452	0.883	0.555

Table 15 Correlate change type with domain ("Development environment" and "Entertainment and education")

		Corrective	Adaptive	Perfective	Preventive	Enhancement
Chi-square	Pearson chi-squared	0.552	0.216	0.269	0.85	0.6
test	Likelihood ratio	0.555	0.21	0.259	0.85	0.603
Symmetric measures	Cramer's V	0.552	0.216	0.269	0.85	0.6

Table 16 Development languages of the OSS projects

Language	Number of OSS projects
C	42
C++	26
Haxe	1
Java	6
Perl	6
PHP	10
Python	10
Ruby	2
Scheme	1
ShellScript	1
SQL	1

First, we consider the whole dataset that includes languages whose frequency count is less than 5. The calculation of correlation using the chi-squared test is given in Table 17. The perfective change type likelihood ratio *p*-value is less than 0.05. This means that the null hypothesis is rejected in the case of the perfective change type. There exists a relation between perfective change type and languages of the OSS projects. The Cramer's V value indicates the presence of a very weak relation between them. All other changes have higher *p*-value than the level of significance (0.05). Therefore, we conclude that in all other change types, there exists no relationship between change type and language of the OSS projects.

Next, we prune the dataset and consider only four languages (C, C++, PHP, and Python) projects for evaluating the correlation. Table 18 indicates that all change types (except preventive) show no correlation with languages used for the development of the projects as Pearson chi-squared and likelihood ratio *p*-values are larger than the level of significance value(0.05). The statistical analysis of languages with change types of OSS projects do not show any strong relation.

5.5 Analysis summary

This study provides key information about the evolutionary processes that the OSS projects follow. One thing is clear that there is no uniform evolutionary pattern. However, projects with the same activity level (measured using the number of commits per month) could be grouped and studied together. Observations of the analysis in the previous sections are summarized as follows.

RQ1 What types of changes are made to a software system immediately after it is put to use?
The OSS projects undergo all types of changes after they are put to use.

RQ2 How do corrective changes evolve?
Corrective changes increase at a high rate for projects with high activity in this category, at a moderate rate in projects with medium activity, and are atmost constant for the projects with low activity.After 81 months, corrective changes start decreasing for projects with high activity in this category, but remain stable after that for projects with moderate activity.

RQ3 After how long do the adaptive/perfective changes start?
Adaptive and perfective change also starts around the same time as corrective changes, but the commit rate (23 commits/month) for corrective changes is significantly higher than that of adaptive and perfective changes (10 commits/month).

RQ4 When do the preventive changes start taking place?
Preventive commits are very insignificant in the beginning (2 commits/month). They increase at a sub-linear rate throughout the observation period. For projects with low activity in this category, a change in commits is almost negligible.

Table 17 Correlate change type with the languages of the OSS projects

		Corrective	Adaptive	Perfective	Preventive	Enhancement
Chi-squared	Pearson chi-squared	0.725	0.301	0.142	0.419	0.336
test	Likelihood ratio	0.579	0.082	**0.032**	0.176	0.158
Symmetric	Phi	0.725	0.301	0.142	0.419	0.336
measures	Cramer's V	0.725	0.301	**0.142**	0.419	0.336

Table 18 Correlate change type with the languages (C/C++/Python/PHP) of the OSS projects

		Corrective	Adaptive	Perfective	Preventive	Enhancement
Chi-squared	Pearson chi-squared	0.737	0.456	0.197	0.121	0.153
test	Likelihood ratio	0.773	0.208	0.089	0.048	0.086
Symmetric	Phi	0.737	0.456	0.197	0.121	0.153
measures	Cramer's V	0.737	0.456	0.197	0.121	0.153

RQ5 How do enhancement changes evolve?
Enhancement changes dominate in the initial period, even more than the corrective changes. They also become stable, for projects with high activity in this category, around the same time as corrective changes (81 months), and start decreasing after that. For moderate-activity projects, enhancement changes become stable after that time.

RQ6 Do all types of changes diminish over time?
Only corrective, adaptive, and enhancement-related changes decrease after a period of time (81 months) for high-activity projects in these categories. Perfective changes become stable towards the end. However, preventive changes keep on increasing.
In moderate activity projects, all change types follow a sub-linear increase. Corrective, adaptive, and enhancement changes stabilize, but perfective and preventive activities continue.
In projects with low activity in these change types, there is hardly any change.

RQ7 How are the OSS projects' attributes, such as number of committers, related to the type of changes performed?
Change activity of a project is directly related to the number of contributors, amount of work done, and total commits of the projects irrespective of the change type. Large numbers of contributors not only make a large number of commits, but also handle a large number of files. Therefore, the volume of work in projects with a large number of contributors is also high.
However, when we compare clusters across the change types (listed in Table 11), e.g., clusters with high activity for all the change types, the clusters differ with respect to project attributes. Projects with high activity in adaptive changes have a maximum number of contributors, whereas projects with high activity in corrective changes have a maximum number of monthly commits.

RQ8 Do change types correlate with domains and languages of the OSS projects?
The results indicate a weak relationship between corrective change type and domains of the projects for domains in which the number of projects (of the sample) are less than 5. However, other change types are not related to any specific domains. When domains, with more than five projects are considered, change types activity and domains donot correlate. Similarly, no correlation exists between change types and the languages of the OSS projects.

5.6 Cluster validation using pairwise comparison of clusters

Cluster validation is important to identify that the clustering algorithm should not have created natural clusters, as every clustering algorithm finds clusters in a dataset. We use pairwise comparisons between the cluster averages to validate the clustering. Table 19 lists p-values for the *pairwise comparisons* between the cluster averages. We use the *pairwise comparison method* [45] to find any significant difference between different clusters for all the OSS projects. p-values are found to be significant at 10%. A high p-value is chosen as it is an exploratory study. As the p-values in all cases are less than 0.010, this shows that there are significant differences between different clusters for commit activity of all the OSS projects.

Table 19 Pairwise comparison of projects in different clusters (p-values significant at 10%)

Change type	Cluster	Committers count	Files changed	Total commits
	1 vs. 2	0.00	0.00	0.00
Corrective	2 vs. 3	0.00	0.00	0.00
	1 vs. 3	0.00	0.00	0.00
	1 vs. 2	0.00	0.00	0.00
Adaptive	2 vs. 3	0.00	0.00	0.00
	1 vs. 3	0.00	0.00	0.00
	1 vs. 2	0.00	0.00	0.00
Perfective	2 vs. 3	0.00	0.00	0.00
	1 vs. 3	0.00	0.00	0.00
	1 vs. 2	0.00	0.00	0.00
Enhancement	2 vs. 3	0.00	0.00	0.00
	1 vs. 3	0.00	0.00	0.00

6 Threats to validity

This section discusses the threats to validity of the study.

Construct validity threats concern the relationship between theory and observation. These threats can be mainly because we assumed that all the commits were posted in the revision control tool *git*. Any changes performed in the source code but not logged through the tool may not have become part of the study. The number of keywords used to categorize the change activity into a particular change type may not be enough to classify all the changes.

Internal validity concerns the selection of subject systems and the analysis methods. This study uses a *month* as the unit of measure for tracking the types of change activities. In the future, we would like to use more natural and insightful partition based on major/minor versions of the OSS project for analyzing the change activity of OSS projects. Subject systems were selected from public repositories, but the selection is biased towards projects with valid *git* repositories.

External validity concerns the generalization of the findings. The OSS projects are chosen from different domains, languages, and with different levels of commit activity, so the study addresses this challenge as far as OSS projects are concerned. Results also need to be validated for proprietary projects.

Reliability validity concerns the possibility of replication of the study. The subject systems are available in the public domain. We have attempted to put all the necessary details of the experiment process into this paper.

7 Conclusions

Change evolution analysis helps in understanding the software evolution process. This study aimed to explore the change behavior, from the point of view of different change types, of 106 OSS projects for a long period of time (approximately nine years) to understand common patterns, if any, of their evolution. Change messages are used to categorize changes into various change types such as corrective, adaptive, perfective, preventive, and enhancement. Change activity is measured using the number of commits per month for a particular change type. Cluster analysis of all the change types gives broadly three categories of change activity: high activity, moderate activity, and low activity. Changes in different categories evolve differently. In high-activity clusters, pressure to add more features dominates at the beginning followed by corrective actions. Corrective changes stabilize before all other change types. Adaptive and perfective changes stabilizes lowly. It may be that developer teams focus on (problem-specific) features of projects first, and the adaptive and perfective changes are implemented later to gain a competitive edge. Preventive work is also present and follows an upward trend throughout. In moderate-activity projects, enhancement changes are again dominant in the beginning and stabilize later (after 81 months). Corrective changes increase at a moderate rate and surpass the enhancement changes towards the end of the observation period. Interestingly, adaptive and perfective changes both start at the same level. Thus, the pressure to adapt and make the systems perfective also mounts at a later stage. The study explores the contribution of other project attributes in the change activity. Results indicate that the projects with higher change activity have large volumes of change work contributed by a large number of contributors in comparison with moderate or low activity. A comparison of project attributes across different change types shows that adaptive changes engage the highest number of committers with the maximum number of files changed in the process. Perfective changes consume the least resources. The study also explores the correlation of change type activity with domains and languages of the OSS projects. The statistical results obtained from the analysis show that for 106 OSS projects we do not find the existence of any strong or relevant relationship among them.

This study considers only average values for the metrics measuring the project attributes. In future, the aim is to track changes in project attributes as a project evolves. We are working on collecting project attributes monthly and then analyzing their evolution along with the change types to understand the similarity of trends between the project attributes and the changes. Furthermore, change evolution can be studied along with the quality (bugs/defects) of projects.

Acknowledgements This research work was performed under a UGC sanctioned research project.We acknowledge the UGC for providing the grant to perform the research work.

Appendix

Table 1A provides the description (project name, domain, language, and license) of the OSS projects. Table 2A provides the project related details of the 106 OSS projects such as project name, project start date, total commits (before cleaning), percentage of commits discarded (blank or unusual text commits), total commits classified by automated classifier, number of lines inserted, the number of lines deleted, and total number of files changed. Interested readers please refer to the summplementary file on the journal's website for more information.

References

1. Lehman M M. Programs, life cycles and laws of software evolution. Proceedings of the IEEE, 1980, 68(9): 1060–1076
2. Hindle A, Godfrey M, Holt R C. Mining recurrent activities: fourier analysis of change events. In: Proceedings of the 31st International Conference on Software Engineering-Companion. 2009, 295–298
3. Mockus A, Votta L G. Identifying reasons for software changes using historic databases. In: Proceedings of International Conference on Software Maintenance. 2000, 120–130
4. Hassan A. Automated classification of change messages in open source projects. ACM Symposium on Applied Computing. 2008, 837–841
5. Kolassa C, Riehle D, Salim M. The empirical commit frequency distribution of open source projects. In: Proceedings of ACM Joint International Symposium on Wikis and Open Collaboration. 2013

6. Lin S H, Ma Y T, Chen J X. Empirical evidence on developer's commit activity for open-source software projects. In: Proceedings of the 25th International Conference on Software Engineering and Knowledge Engineering. 2013, 455–460
7. Tiwari P, Li W, Alomainy R, Wei B Y. An empirical study of different types of changes in the eclipse project. The Open Software Engineering Journal, 2013, 7: 24–37
8. Kemerer C F, Slaughter S A. An empirical approach to studying software evolution. IEEE Transactions on Software Engineering, 1999, 25(4): 493–509
9. Bennett K H. Software maintenance and evolution: a roadmap. In: Proceedings of the 22nd International Conference on Software Engineering. 2000, 73–78
10. Gupta A, Conradi R, Shull F, Cruzes D, Ackermann C, Rønneberg H, Landre E. Experience report on the effect of software development characteristics on change distribution. In: Proceedings of the 9th International Conference on Product Focused Software Process Improvement. 2008, 158–173
11. Smith N, Capiluppi A, Ramil J F. A study of open source software evolution data using qualitative simulation. Software Process: Improvement and Practice, 2005, 10(3): 287–300
12. Gonzalez-Barahona J, Robles G, Herriaz I, Ortega F. Studying the laws of software evolution in a long-lived FLOSS project. Journal of Software: Evolution and Process, 2014, 26(7): 589–612
13. Koch S. Evolution of open source software systems–a large-scale investigation. In: Proceedings of the 1st International Conference on Open Source Systems. 2005, 148–153
14. Schach S R, Jin B, Wright D R, Heller G Z, Offutt J. Determining the distribution of maintenance categories: survey versus measurement. Empirical Software Engineering, 2003, 8(4): 351–365
15. Burch E, Kungs H J. Modeling software maintenance requests: acase study. In: Proceedings of the International Conference on Software Maintenance. 1997, 40–47
16. Swanson B. The dimensions of maintenance. In: Proceedings of the 2nd International Conference on Software Engineering. 1976, 492–497
17. IEEE. Standard for Software Maintenance (IEEE Std 1219–1998). New York: Institute for Electrical and Electronic Engineers, 1998
18. ISO/IEC FDIS 14764:1999(E). Software Engineering—Software Maintenance. Geneva: International Standards Organization, 1999
19. Lientz B P, Swanson E B, Tompkins G E. Characteristics of application software maintenance. Communication of the ACM, 1978, 21(6): 466–471
20. Nosek J, Palvia T P. Software maintenance management: changes in the last decade. Journal of Software Maintenance: Research and Practice, 1990, 2(3): 157–174
21. Lee M G, Jefferson T L. An empirical study of software maintenance of a Web-based Java application. In: Proceedings of the 21st IEEE International Conference on Software Maintenance. 2005, 571–576
22. Basili V, Briand L C, Condon S, Kim Y M, Melo W L,Valettt J D. Understanding and predicting the process of software maintenance releases. In: Proceedings of the 18th International Conference on Software Engineering. 1996, 464–474
23. Sousa M J C, Moreira H M. A Survey on the software maintenance process. In: Proceedings of IEEE International Conference on Software Maintenance. 1998, 265–274
24. Yip S W L, Lam T. A software maintenance survey. In: Proceedings of the 1st Asia-Pacific Software Engineering Conference. 1994, 70–79
25. Abran A, Nguyenkim H. Analysis of maintenance work categories through measurement. In: Proceedings of IEEE Conference on Software Maintenance. 1991, 104–113.
26. Gefen D, Schneberger S L. The non-homogeneous maintenance periods: a case study of software modifications. In: Proceedings of IEEE Conference on Software Maintenance. 1996, 134–141
27. Meqdadi O, Alhindawi N, Collard M L, Maletic J I. Towards understanding large-scale adaptive changes from version histories. In: Proceedings of the 29th IEEE International Conference on Software Maintenance. 2013, 416-419
28. Blei D M, Ng A Y, Jordan M I. Latent dirichlet allocation. Journal of Machine Learning Research. 2003, 3: 993–1022
29. Kim S, Whitehead E J, Zhang Y. Classifying software changes: clean or buggy. IEEE Transactions on Software Engineering, 2008, 34(2): 181–196
30. Lehnert S, Riebisch M. A taxonomy of change types and its application in software evolution. In: Proceedings of the 19th International Conference and Workshops on Engineering of Computer Based Systems. 2012, 98–107
31. Chaplin N, Hale J E, Khan K M, Ramil J F, Tan W G. Types of software evolution and software maintenance. Journal of Software Maintenance and Evolution: Research and Practice, 2001, 13(1): 3–30
32. Forward A, Lethbridge T C. A taxonomy of software types to facilitate search and evidence-based software engineering. In: Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds. 2008, 14
33. Saini M, Kaur K. Analyzing the change profiles of software systems using their change logs. International Journal of Software Engineering-Egypt, 2014, 7(2): 39–66
34. Larose D T. K-nearest neighbor algorithm. Discovering Knowledge in Data: An Introduction to Data Mining, 2005, 90–106
35. Cohen J. A coefficient of agreement for nominal scales. Educational and Psychological Measurements, 1960, 20(1): 37–46
36. Kohavi R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of International Joint Conference on Artificial Intelligence. 1995, 1137–1145
37. Cleveland W S. LOWESS: a program for smoothing scatterplots by robust locally weighted regression. The American Statistician, 1981, 35(1): 54
38. Massart D L, Smeyers-Verbeke A J, Capron A X, Schlesier K B. Visual presentation of data by means of box plots. LC-GC Europe, 2005, 18(4): 2–5
39. Ramsay J O, Silverman B W. Applied Functional Data Analysis: Methods and Case Studies. New York: Springer-Verlag, 2002
40. Cuesta-Albertos J A, Gordaliza A, Matrán C. Trimmed *k*-means: an attempt to robustify quantizers. The Annals of Statistics, 1997, 25(2): 553–576
41. Han J, Kamber M. Data Mining: Concepts and Techniques. San Francisco: Morgan Kaufmann, 2000
42. Kothari R, Pitts D. On finding the number of clusters. Pattern Recognition Letters, 1999, 20(4): 405–416
43. Kohavi R. A study of cross-validation and bootstrap for accuracy es-

timation and model selection. In: Proceedings of International Joint Conference on Artificial Intelligence. 1995, 1137–1145

44. Moore D S. Chi-square tests. Purdue University, 1976

45. Bolstad B M, Irizarry R A, Åstrand M, Speed T P. A comparison of normalization methods for high density oligonucleotide array data based on variance and bias. Bioinformatics. 2003, 19(2): 185–193

Munish Saini is a PhD student in the Department of Computer Science, Guru Nanak Dev University, India. He received his B. Tech degree in computer science and engineering from Sant Baba Bhag Singh Institute of Engineering and Technology, India and M.Tech in computer science and engineering from Dr. B. R. Ambedkar National Institute of Technology, India. His research interests are in data mining, open-source software, and software engineering.

Kuljit Kaur Chahal is an assistant professor in the Department of Computer Science, Guru Nanak Dev University, India. She received her PhD degree in computer science from Guru Nanak Dev University, India. Her research interests are in distributed computing, Web services security, and open-source software.