Developer Dynamics and Syntactic Quality of Commit Messages in OSS Projects

Kuljit Kaur Chahal, Munish Saini

▶ To cite this version:

Kuljit Kaur Chahal, Munish Saini. Developer Dynamics and Syntactic Quality of Commit Messages in OSS Projects. 14th IFIP International Conference on Open Source Systems (OSS), Jun 2018, Athens, Greece. pp.61-76, 10.1007/978-3-319-92375-8_6 . hal-01875491

HAL Id: hal-01875491
https://inria.hal.science/hal-01875491
Submitted on 17 Sep 2018

Developer Dynamics and Syntactic Quality of Commit Messages in OSS Projects

Kuljit Kaur Chahal, Munish Saini
Department of Computer Science
Guru Nanak Dev University Amritsar India
kuljitchahal@yahoo.com

Abstract. Community dynamics play an important role in the Open Source Software (OSS) development paradigm. Researchers have extensively studied the human aspects of the OSS paradigm from the point of view of community formation to community evolution. A few studies relate community dynamics with OSS product attributes such as code quality. However, the impact of community dynamics on non-code contributions such as commits has not been explored. In this paper, the aim is to analyze the impact of community dynamics on syntactic quality of commit messages of an OSS project. We first propose and validate a commit message quality model, and then use that model to analyze the OSS projects. Empirical analysis of seven OSS projects available in the Git repository shows that a small group of contributors active at the same time in a project leads to high syntactic quality contributions. These observations may prove useful to developers as well as project managers who need quantifiable techniques for monitoring the OSS projects.

CCS CONCEPTS

• **CCS → Software and its engineering → Software notations and tools → Software configuration management and version control systems**

Keywords. Open Source Software (OSS); Software Evolution; Source Code Management; Commit activity; Commit Message Quality.

1 INTRODUCTION

In Open Source Software (OSS) development is presumed to entail collaborative participation of geographically distributed developers for creating a successful project. A source code management system such as Git records and manages contributions of participants in the project repository. Majority of the participants are volunteers. Major motivations for participants include developing and improving skills, getting recognition for the skills, and building a reputation which in turn helps them in furthering their commercial endeavors [8]. Sans any organization control, they can join or leave an OSS project as per their own convenience. Unlike commercial software in which committed employees contribute regularly, an OSS project generally depends upon contributions from self-motivated individuals who belong to different geographical locations, and diverse cultures and backgrounds. More recently, corporate backed OSS projects are also emerging on the landscape. Large corporates such as IBM, HP support OSS development with their own resources (i.e. their paid workforce contributes) in various projects that they use in their own products. Whatever the mode of participation, we can say that OSS community plays an important role in the OSS development paradigm.

However, this is not static but a very dynamic community. There are no fixed roles. A member can contribute to an OSS project in a number of ways depending upon his skill set (as a developer, tester, or documenter). Not only this, he /she can fluidly shift from being an end user to a developer (for example a tool user can work on improving the tool). Due to the volunteer nature of participation, there could be many lean periods in a project's activity when participants are busy in their regular life activities, for example working on a full-time job on weekdays, enjoying vacations, or not contributing due to inexplicable reasons. It will be interesting to investigate community dynamics and its impacts on OSS development processes.

In this paper, we are interested in understanding the impact of community dynamics on the quality of contributions committed to a project's repository.

A commit, a software change that involves a source code or other type of contribution such as documentation, is a fundamental component of an OSS development process. In the recent past, commit analysis has been a topic of active research to understand the software development processes of OSS projects [8, 12, and 13]. For example commit activity (measured as the number of commits in a unit of time) of an OSS project is correlated with successfulness of the project [13]. A developer with high commit frequency is more productive. A project with high commit frequency is healthy as it gets regular contributions. There is not much research on this topic to define commit quality or to use commit quality information to characterize OSS projects or developers' contribution practices. A few studies have analyzed commits of software projects from quality perspective [1], though there is a lot of work focusing on evaluating the code quality of OSS projects. In this paper, we propose to answer the following question:

What is the impact of community dynamics on commit message (syntactic) quality in the context of OSS projects?

A good quality commit contains a well-crafted message with all the necessary details (meta-data) to effectively convey the change to current or future developers [5]. It not only makes the changes contributed by others easy to understand but also helps in recalling one's own changes contributed in the past. A good commit message should follow a simple and consistent style for specifying commit meta-data and content.

With respect to commit message quality in OSS projects, we propose a commit message syntactic quality model in section 3. Following this model, each commit message can be assigned a commit score. In this study, we are focusing only on the syntactic view of commit messages. The syntactic analysis shows "writing styles of the contributors" i.e. whether they follow the rules of the syntax while describing their commits. We analyzed 202,561 commit messages of seven OSS projects to understand the way committers commit changes in a Source Code Management (SCM) system specifically Git.

Paper organization: The rest of the paper is organized as follows. Next section presents the related work. Rest of the sections present the commit quality model, the data collection steps, and the results in that order. In the end, a section mentions limitations of the study followed by conclusions and future work.

2 RELATED WORK

The success of an OSS project suggestively depends upon the type of community support available to the project as social aspects significantly determine evolution [16]. However, in the context of the OSS paradigm, the community itself is dynamic in nature. This section discusses the prior work related to community dynamics and its impact on OSS processes. A few of the works related to commit quality and commit analysis are also mentioned.

Some studies in the past focused only on static community structure to understand the demographic diversity of community members [14], gender differences [15], and role of the core members [22]. While [4] studied community dynamics using social network analysis to understand the changes that happen to a community over a period of time. Changes in the community structure are presented with the help of temporal visualization and quantitative analysis.

Bird *et al.* [7] analyzed two large OSS projects, Firefox and Eclipse, to investigate the impact of distributed locations of the developers on the quality of the code they contributed. They found that quality of components (measured using the number of defects) developed by distributed teams was bad in comparison to the quality of components developed by collocated teams.

Ahmed *et al.* [2] relate poor coding practices with growth in the number of developers. The study concludes that code as well as design quality declines as the number of developers increases. Souza and Silva [21] analyze affect/effect of developer sentiment levels (as expressed in commit messages) on build status of Travis, a Continuous Integration server. The results suggest that negative sentiment reduces the chances of a successful build, though the effect is minor.

Santos *et al.* [18] studied the unusualness of commit messages by training n-gram language models on 120,000 commits of OSS projects and used cross-entropy as an indicator of a commit message's "unusualness". Their work focused mainly on finding the unusualness of a commit message, and further correlating it with code quality. Agrawal *et al.* [1] studied the commit quality of five high-performance computing projects and compared the performance of the projects with three low performance computing projects.

Most of the works in the research literature on commit analysis of OSS projects deal with identifying commit size distribution [3], commit frequency distribution [13], commit characterization [17, 23], and contributor's commit activity distribution [8]. Chełkowski *et al.* [8] analyzed commit contributions of Apache contributors to highlight inequalities among open source contributors' in producing content in the OSS paradigm which is often described as collaborative.

Lack of literature on the subject and the broad nature of practitioner recommendations suggest a need for a research study regarding the quality analysis of the commit messages recorded in a source code management system. We followed a Multi-vocal Literature Review (MLR) approach [8]**.** In this study, our focus is on measuring commit message quality syntactically by using 11 syntactical metrics by introducing a novel approach to calculate commit quality. Moreover, we focused on finding if there is any relation between community evolution and the commit message quality of the OSS projects. The commit message quality is correlated with the number of contributors to understand the impact of community dynamics on development processes of the OSS projects.

3 THE PROPOSED MODEL AND ITS VALIDATION

A set of measures, to calculate the syntactic quality of commit messages recorded in SCM system of the OSS projects, are devised after consulting a bulk of literature (published [1], or available online [9]) to understand "how to write a good commit message". The online search to this topic "how to write a good commit message" or "good commit logs" yielded a large number of results. We followed a double cross-check approach to select the commit quality metrics. Both the authors analyzed the top 33 web links (beyond this the content was repeated) individually, and noted all the rules and identified the possible list of attributes that can act as commit quality measures. At the last, we combined the rules and the lists of attributes that were identified by both the authors. This double cross-check approach avoided any rule or attribute to get skipped from the analysis. After consulting the literature, common rules indicating a good quality commit message are identified as shown in Table 1:

Table 1. Rules for writing a good commit

1.	Title (subject line) of commit message should be short (between 50-72 characters).
2.	Subject line should end with a dot.
3.	Capitalize the subject line i.e. first character of the subject line should be capital.
4.	Use imperative mood in the subject line for example use words like fix, add, update in place of fixing, adding, and updating etc.
5.	Subject line should be concise and limit the number of "and", "or".
6.	Subject line should not include details such as bug number, file name, ticket number, and any other external references.
7.	Subject line and body must be separated by a blank line.
8.	Body of a commit message must have multiline description. It should be well explanatory detailing why and what is changed.
9.	Body of a commit message should not contain lots of bullets, hyphens, or asterisks.
10.	Commit should have one logical change.

By considering all these rules, we devised 11 commit quality measures (see Table 2). Out of which, seven are for subject line (title), and the rest four are for the body (multiline description) of a commit message. After evaluating the count and values for the commit messages, the proposed approach assigns scores to each measure on a scale of 1 to 5 (as shown in Table 2).

Next step was to see whether the proposed rules and the corresponding metric definitions sound reasonable from practitioners' point of view. We chose the survey based method to get inputs from practitioners. In response to a survey request, 20 developers volunteered to participate in the survey. Most of the participants were graduates, with a total of 16 graduate and 4 undergraduate degree holders. Their industrial experience varied from 5 to 7 years in software projects based on Java/C#. In the survey, the participants were provided a sample of commits and were asked to upvote a rule if they agree, downvote a rule if they don't agree, or post a neutral response if it does not matter to them while reading a commit message. They also reported their votes on metric definitions.

As a result of this survey (see Table 3), we concluded that the rules and the metrics to evaluate the commits were useful and reasonable.

Table 2. Commit Message Syntactic Quality Measures

Commit Quality Measures	**Commit Score**					
	1	**2**	**3**	**4**	**5**	**unit**
Length of Title	=0 or >72	1-10	11-30	31-50	51-72	number of characters
Title ends with dots	No	Yes				y→1, n→0
Title first character capital	No	Yes				y→1, n→0
Count number of “and” “or” in Title	>6	5-6	3-4	1-2	0	count
Count number of “file name” in title	>6	5-6	3-4	1-2	0	count
Count number of external references in title	>6	5-6	3-4	1-2	0	count
Imperative mode in title	No	Yes				y→1, n→0
Commit body existence	No	Yes				y→1, n→0
Count number of “file name” in body	0	10>	6-10	3-5	1-2	count
Count number of external references in body	0	10>	6-10	3-5	1-2	count
Count number of paragraph in body	0	10>	5-10	3-4	1-2	count

Then we calculated the total score for each commit message. We cannot use these scores of individual measures as such to calculate the total score of message quality of a commit as different commit measures have different scales (few commit measures have values on scale 1-5, whereas other have values on the scale 1-2). Therefore, we first normalized the commit scores of individual measures to a common scale [0, 1].The normalization of commit measures is done by using the following formula [20]:

$$\text{Normalized commit message score}=\text{ActualScore}/\text{MaxScore} \qquad (1)$$

Table 3: Survey Responses

	Rule	Upvotes	Downvotes	Neutral
1.	Title (subject line) of commit message should be short (between 50-72 characters).	15	4	1
2.	Subject line should end with a dot.	10	7	3
3.	Capitalize the subject line i.e. first character of the subject line should be capital.	8	6	6
4.	Use imperative mood in the subject line for example use words like fix, add, update in place of fixing, adding, and updating etc.	20	-	-
5.	Subject line should be concise and limit the number of “and”, “or”.	18	-	2
6.	Subject line should not include details such as bug number, file name, ticket number, and any other external references.	17	-	3
7.	Subject line and body must be separated by a blank line.	12	2	6
8.	Body of a commit message must have multiline description. It should be well explanatory detailing why and what is changed.	19	-	1
9.	Body of a commit message should not contain lots of bullets, hyphens, or asterisks.	18	-	2
10.	Commit should have one logical change.	20	-	-

$$\text{Total commit score} = \sum_{1}^{11} WCS\_CommitMeasure\_i \tag{2}$$

In order to further validate the results of the commit message quality score, the results of the proposed model for a sample of 100 commits messages (50 with commit messages as per the rules and 50 otherwise) were compared with the assessment results made available by the same survey participants. The results show that 84% of the commit messages were correctly judged by the proposed model. Specifically, for commit messages with good quality, about 88% of messages were correctly judged, and about 80% of messages with poor quality were correctly judged by the proposed model. This shows that the proposed model is effective.

4 DATA COLLECTION

Seven OSS projects were selected on the basis of their popularity, age, size, number of people involved, and availability of the project repository in Git (an open source distributed version control system).

PostGreSQL is an object-relational data base management system. glibc is a GNU C library used in the GNU/Linux systems. Eclipse-CDT is an industrial strength IDE for developing C/C++ programs and plug-in tools. GnuCash is a double entry accounting software for personal and small enterprises. WordPress is web publishing software. Firebug is a web browser extension for Mozilla Firefox for debugging and performance analysis of web pages rendered in the browser. Rhino is a JavaScript engine. It is an open source application of JavaScript. It is regularly implanted into Java applications to give scripting to end users.

Development repositories of the OSS projects are obtained from the Github (www.github.com). A repository is downloaded by making a clone of the original repository onto the local machine by using Gitbash(www.git-scm.com). A script is written in Java to fetch the commit messages and number of contributors for the observation period for all the OSS projects. Table 4 summarizes the statistics of the datasets collected for all the seven projects.

Table 4. Descriptive Statistics of the OSS projects

OSS Projects	Origin Date	Number of Months	Number of Contributors	Commit messages
PostgreSQL	Jul, 1996	239	43	54355
glibc	Feb, 1989	321	410	43313
Eclipse-CDT	Jun, 2002	168	203	28817
GnuCash	Nov, 1997	222	105	21969
WordPress	Apr, 2003	158	73	37333
Firebug	Aug, 2007	181	45	13043
Rhino	Apr, 1999	105	56	3721

5 RESULT AND ANALYSIS

This study explores the commit messages of seven OSS projects to calculate and analyze the commit quality of these commit messages. Further, the commit message quality measures are used to answer the research question specified in Section 1.

In this section, we first analyze the commit message quality of the OSS projects as they evolve over a period of time. But before that, we look at the differences in levels of the commit message quality in these projects. With the help of box plots, Fig. 1 shows the variation in the commit message quality across the projects. *GnuCash* and *WordPress* have the best median values (>0.80) for the commit message score. Next is *Firebug* with median commit message score 0.75. The remaining four projects (which includes very popular projects *PostgreSQL* and *Eclipse-CDT*) have median commit message scores less than 0.70.

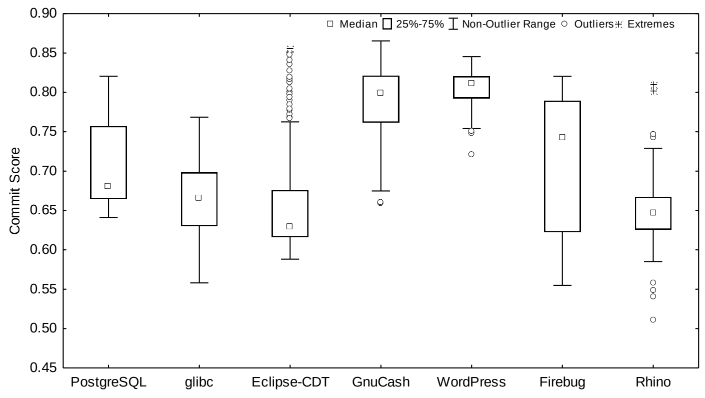


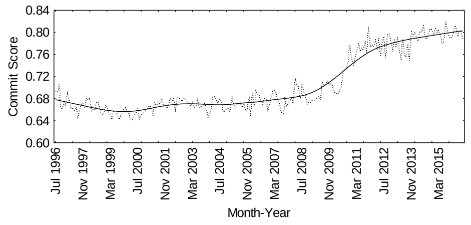
Fig. 1: Variation in commit quality of the OSS projects

Eclipse-CDT has many outliers towards the upper side of the box plot otherwise, the median commit score is the minimum in comparison to other projects.

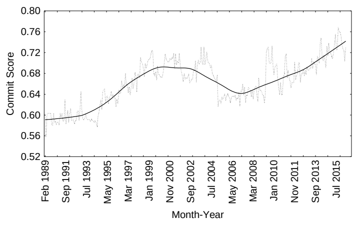
Next, we analyzed the commit message quality evolution in these projects. Intuitively, commit message quality should improve over the period of time as a project matures as core team of experienced developers is supposed to vet commits submitted by less experienced developers . We can see in Fig. 2a, 2c (on next page) that commit message quality improves after staying stable for a long period of time. In fig. 2g, it stays stable throughout. We note from Table 5 that commit message score of these projects follows an increasing trend when analyzed using linear regression [19]. In case of the other three projects (see Fig. 2d-2f), it drops down and shows a decreasing trend when analyzed using linear regression. We believe that this behavior may be due to developer dynamics. A detailed explanation of this behavior of the OSS projects is due until discussion of the next section. The in-between variation (e.g. decrease in March 2008 for the *glibc* project) in commit message score of these projects needs further analysis to understand the factors responsible for such behavior. In the next section, we revisit this type of behavior to find the factors that may have affected the commit message score. From the above observations, we can conclude that the commit message quality does not always improve as a software project matures. Some of the projects in this case study point to slender periods when commit message quality goes down.

Table 5. Trend in commit scores of the OSS projects

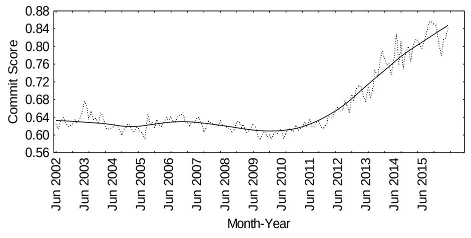
OSS Project	Regression Equation	Trend
PostgreSQL	y=0.62+0.00065x	Increasing
glibc	y=0.61+0.00029x	Increasing
Eclipse-CDT	y=0.57+0.0010x	Increasing
GnuCash	y=0.79-0.0001x	Decreasing
WordPress	y=0.82-0.0003x	Decreasing
Firebug	y=0.83-0.002x	Decreasing
Rhino	y=0.641+8.67E-5x	Increasing



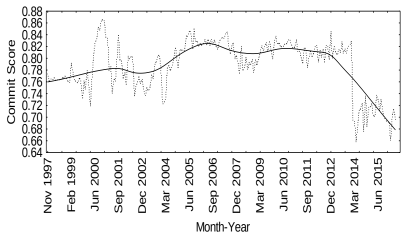
(a)



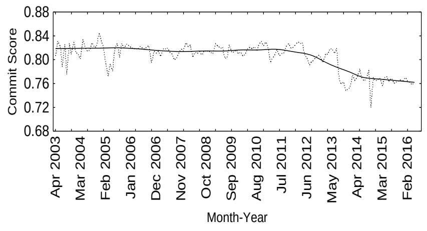
(b)



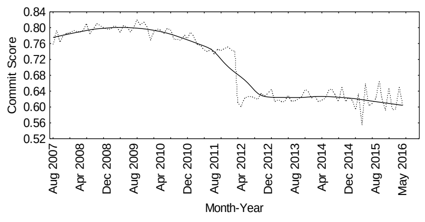
(c)



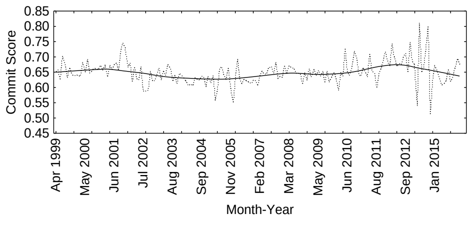
(d)



(e)



(f)



(g)

Fig. 2. Average commit message score for *a)* PostgreSQL, *b)* glibc, *c)* Eclipse-CDT, *d)* GnuCash, *e)* WordPress, *f)* Firebug, *g)* Rhino

5.1 Does the number of contributors affect the commit message syntactic quality?

In OSS projects, developer community plays an important role. Contributors contribute by writing new code and documentation, and also make changes to fix bugs, or to improve the overall quality of a software project. Therefore, it is important to see the relation of contributors' participation in OSS projects with their commit message quality. We want to examine whether an increase in the number of contributors coincides with the increase in the commit message quality of the OSS projects. Intuitively, better commit message quality can be expected as multiple contributors should share the responsibility, and contribute to the project in a better way. Single contributors or small teams may not be able to produce good contributions when they are bogged down by the work pressure.

To begin with, we analyzed the variation in the number of contributors across the OSS projects.

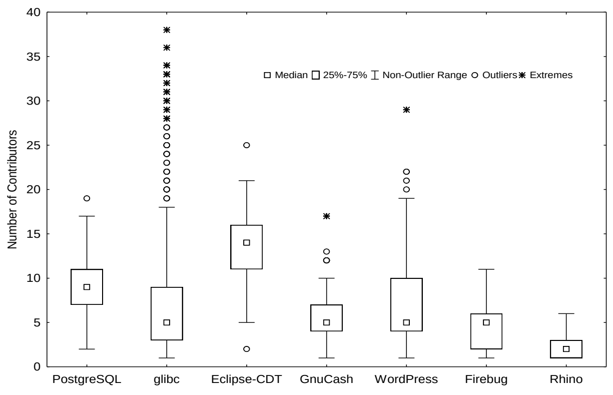


Fig. 3. Variation in the number of contributors of the OSS projects

Fig. 3 shows that maximum range is in case of the *glibc* project. *glibc* is a very old project, started in 1989, a project with the longest history. As far as the median values are concerned, *Eclipse-CDT* has the largest team size of 15 contributors in a month. *Eclipse-CDT* enjoys the reputation of a very popular project among the developer community. In all the other cases, team size is less than 10 contributors. Four out of the seven projects i.e. *glibc*, *WordPress*, *Firebug*, and *GnuCash* have a median value of 5 contributors in a month. Among these four projects, three (*WordPress*, *Firebug*, and *GnuCash*) have the best commit score (median > 0.75). Interestingly, commit message quality is worst in the *Eclipse-CDT* project (Fig. 1 box plot for commit score). Whereas small teams in four other projects are better in producing good quality work. *Eclipse* is a mature project, and such projects also tend to allow contributions from peripheral developers. Therefore, a large team may not ensure better work quality if peripheral (may be untrusted) contributors are allowed to submit changes. It could also be due to if core team does not bother to vet such changes.

Fig. 4 (a-g) shows contributor churn of the OSS projects over a period of time. In all the OSS projects except *Rhino*, the number of contributors follows an increasing trend over the period of time (see Table 6). We can observe that after initial few years, developer participation has increased manifold notably in the projects *PostgreSQL*, *glibc*, *Eclipse-CDT*, and *WordPress*. However, for the project like *Eclipse-CDT*, this initial period is very small i.e. only two years, but for *glibc* it is very long. We can see a surge in the number of contributors of *glibc* only after 2009 - the year its code base was migrated from CVS to Git. Similar contributor pattern can be seen for the *GnuCash* project as well whose code base shifted to Git in 2014. Shifting of source code management to the Git repository perhaps reduced the barriers for new contributors to enter. Git uses the fork and pull request model in which a contributor forks the code branch to which it wants to contribute, makes changes to the clone, and then submits it. When accepted, such contributions can be easily merged with the main branch. It suggests that modern tools have improved the process of code contribution.

In case of *Firebug*, the pouring in of contributors stopped around April 2014. An interesting observation is when we relate it to the commit activity of the project; it also dried up around the same time. *Firebug* is an extension of the Mozilla Firefox web browser for debugging and monitoring of the web pages rendered in the browser. Project pages reveal that the *Firebug* project was abandoned during this period of time. People unhappy with this development, as a result, chose Google Chrome over Mozilla Firefox as they had earlier preferred Firefox just because of the *Firebug* plug-in available with it.

Table 6. Trend in number of contributors

OSS Project	Regression Equation	Trend
PostgreSQL	y=5.48+0.031x	Increasing
glibc	y=0.064x-2.05	Increasing
Eclipse-CDT	y=9.25+0.048x	Increasing
GnuCash	y=2.95+0.022x	Increasing
WordPress	y=0.098x-0.15	Increasing
Firebug	y=4.12+0.006x	Increasing
Rhino	y=2.12-0.0008x	Decreasing

5.2 Understanding the Contribution Pattern

In order to explore further the commit message quality of these projects, we decided to analyze the volume and the quality of contribution of the individual contributors in these OSS projects. The basis of this decision was the outcome of the previous studies on the patterns of contribution in OSS projects that the bulk of the activity is due to a relatively small number of contributors [8]. The social structure of OSS projects is more notably known as the onion model [10] in which core members form the innermost layer and peripheral contributors belong to the outer layers. Contributors of OSS projects are put into core and periphery categories where core contributors are supposed to possess better skills and have more authority on the project development over the peripheral contributors.

Therefore in this regard, the first step is to find commit distribution among different contributors of the OSS projects to identify the core group of contributors. Next, we analyze their commit behavior from two perspectives – commitment (i.e. regularity to commit), and the level of skill (i.e. commit message quality), as these are among the factors that determine the status of a contributor in the social structure of an OSS project.

Table 7 presents the total contribution (in %) of the different contributors of the OSS projects. It shows the commit distribution of only top six contributors (as beyond this number the individual contribution drops significantly in this dataset). Contributions of the rest (excluding the top six contributors) are merged under the head ‘others’. *PostgreSQL*, *glibc*, and *Rhino* have approximately 80% contributions from top 6 contributors. *Firebug* has 80% contributions from top three contributors only. For the rest three projects i.e. *Eclipse-CDT*, *GnuCash*, *WordPress*, contributions are more widespread.

Table 7. Contributor wise commits distribution (in %)

OSS Projects	C1	C2	C3	C4	C5	C6	Other
PostgreSQL	34.53	26.52	7.35	3.62	3.33	3.1	21.55
glibc	40.58	24.18	4.72	4.14	3.56	3.36	19.46
Eclipse-CDT	10.11	7.9	6.43	5.64	5.56	5.52	58.84
GnuCash	16.66	14.84	12.65	7.93	7.57	7.57	32.78
WordPress	22.13	7.67	7.4	5.24	4.79	3.82	48.95
Firebug	46.8	19.66	14.37	6.08	2.99	1.29	8.81
Rhino	30.42	20.78	11.82	5.36	4.93	3.35	23.34

We know that, in this data set, *Eclipse-CDT* has the largest (median) number of contributors. The contribution is also quite equally spread among all the contributors of the project as per the data in Table 7. Majority of the *Eclipse-CDT* commits are from non-core (external) contributors. That may be the reason for low commit quality in the project. Though contribution pattern is uniformly spread across different contributors in case of *GnuCash* and *WordPress* projects as well, but they have small team size. At the same time, their commit message quality is good. A small number of contributors are responsible for the commit activity, work distribution is balanced, and commit message quality is also good.

For the commitment or regularity of the commit activity, we tracked their commit activity over the period of time. In Fig. 5 (a - g), a horizontal line represents the period when a contributor is active. If there is no contribution in a month, then there is a gap in the line. We can observe in the figure that some of the lines represent continuous activity indicating a regular activity, whereas in some cases there are gaps indicating irregular activity.

It shows that a few contributors are more regular in commit activities. Only a few of them contribute regularly to a repository. The commit activity of different contributors overlaps at several points. Except *Rhino*, all other projects have at least one contributor with regular commit activity of not less than six years. We can see that when multiple contributors are active in a project at the same time, commit quality is better.

For the *PostgreSQL* project, commit quality improves after 2010 (see Fig. 2a). Multiple committers are active around the same time. In case of the *glibc* project, commit quality improves from 1999 to 2004, and after a dip, again from 2009 onwards (see Fig.2b). Look at the figures for the same time periods, multiple committers are active at the same time. Same is the case for *GnuCash* and *WordPress*. In the *GnuCash* project, commit quality decreases in 2014. At the same time, we can see in the Fig. 5d, the number of active contributors also reduces. There is only one contributor after that time period. In case of the *WordPress* project, commit quality goes down around 2012 (see Fig. 2e). Around the same time period, all the active contributors stop contributing (Fig. 5e).Three new contributors join in, and their commits are perhaps not yet of good quality. *WordPress* project follows a liberal procedure to let people join the project. The *Rhino* project has the least activity in the group of analyzed projects. Committer activeness is also scant for this project.

Therefore, based on the above discussion, we can say that a group of contributors active at the same time in a project leads to high-quality contributions. It may be a consequence of the uniform work distribution among multiple contributors. It could also be due to the availability of peer-support which helps in gaining insights and developing better ideas.

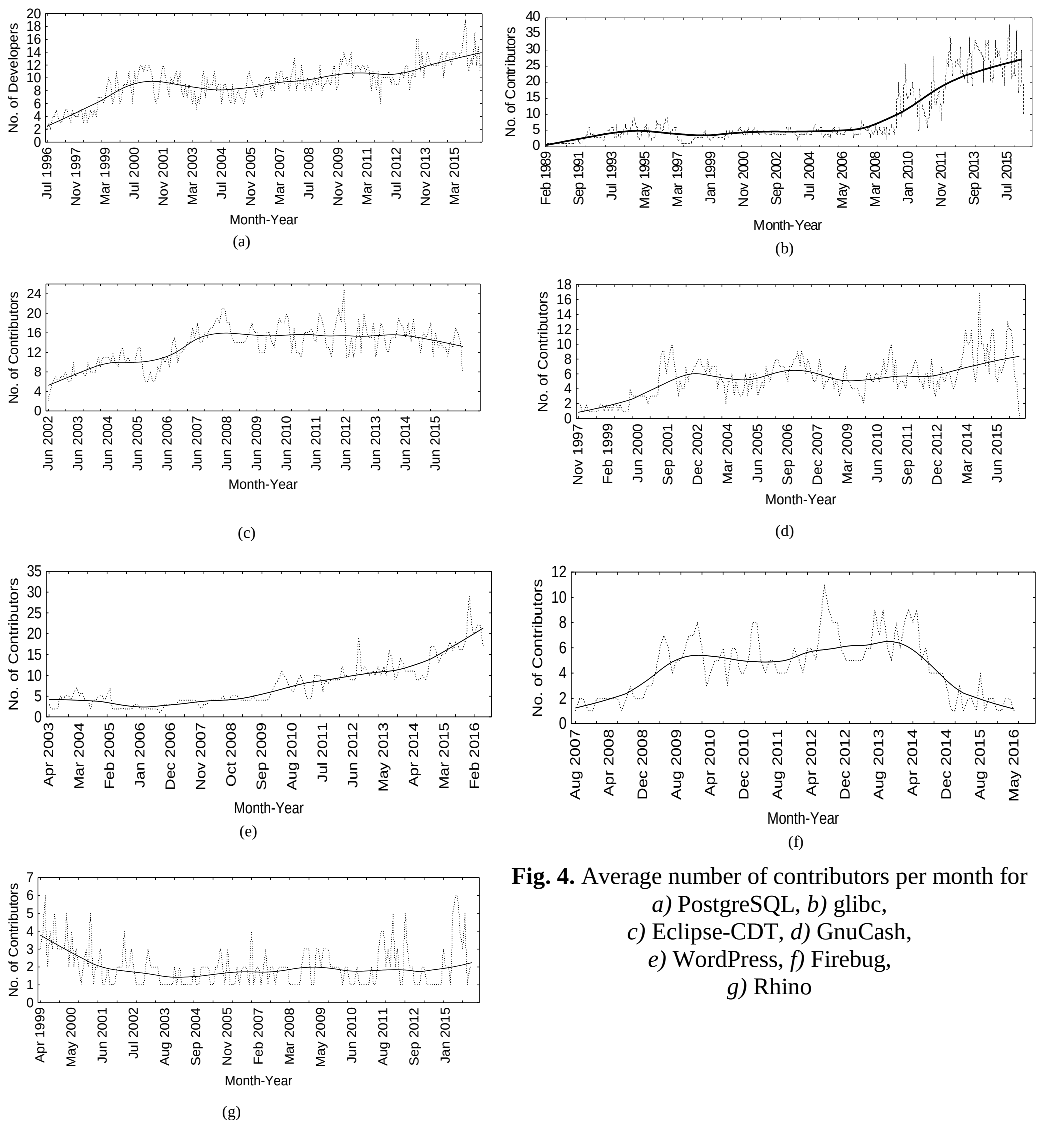
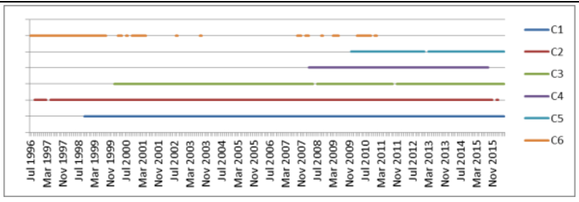


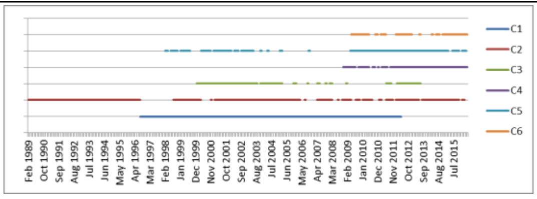
Fig. 4. Average number of contributors per month for *a)* PostgreSQL, *b)* glibc, *c)* Eclipse-CDT, *d)* GnuCash, *e)* WordPress, *f)* Firebug, *g)* Rhino

This study shows that as code contribution practices evolve, commit activity improves. Pull request systems are found to be more efficient for source code management. Previous research also shows that process effectiveness ignites users' interest in an OSS project [11].

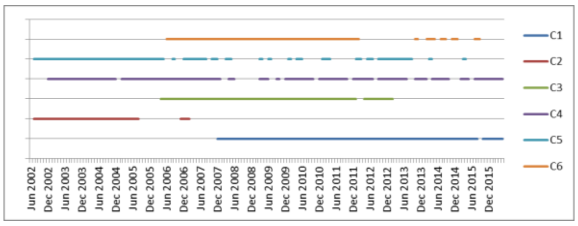
Open source projects have contributors with diverse skill sets. Individuals with better skills are likely more powerful and, are the core contributors. Non-core contributors are individuals who lack knowledge and experience in comparison to the core contributors. External contributors can affect the commit message quality in two different ways. One is when non-core contributors contribute work with mediocre quality. For example, the case of the *Eclipse-CDT* project, which has a uniformly spread contributions from a large number of contributors. Second is when multiple commits are committed as part of a single large commit as is the case of the *PostgreSQL*. In both the cases, commit message quality suffers.



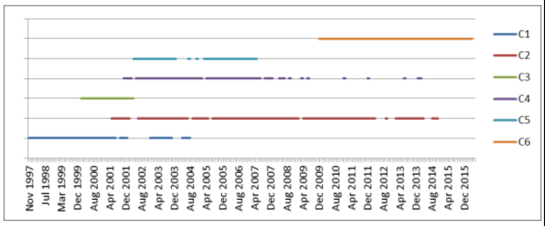
(a)



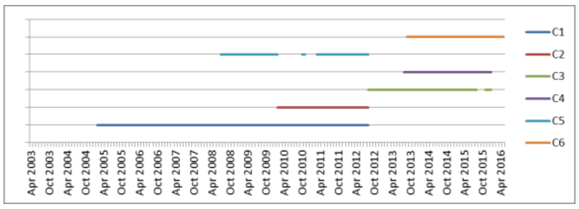
(b)



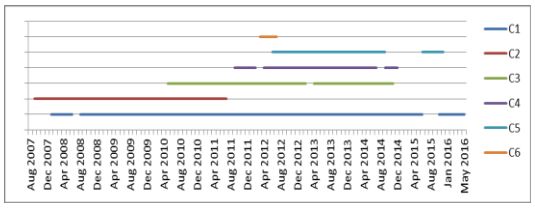
(c)



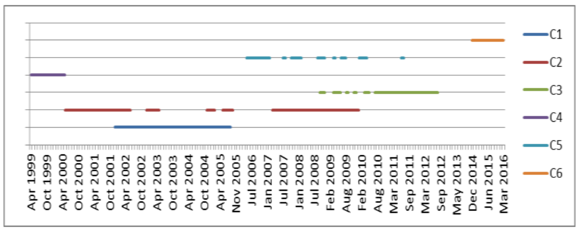
(d)



(e)



(f)



(g)

Fig. 5: Committing behavior of the top 6 contributors of
a) PostgreSQL, *b)* glibc,
c) Eclipse-CDT, *d)* GnuCash,
e) WordPress, *f)* Firebug,
g) Rhino

6 Limitations of the Study

This study considers the commits that are 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.

Selection of the subject systems is biased towards projects with valid Git repositories.

Though we developed objective measures to capture different aspects of a good commit message, but certain features might have got skipped by both the authors.

7 Conclusions and Future Work

In an OSS community, people are not committed to use or contribute to a particular project regularly. Sometimes, the community support flourishes, and sometimes it dwindles. The major objective of this study was to understand the impact of community dynamics on the quality of contributions submitted to a source code management system of an OSS project. A commit message quality model is proposed to evaluate the syntactic quality of commit meta-data submitted by the developers of an OSS project. *GnuCash* and *WordPress* have very high commit quality throughout in comparison to other five projects analyzed in this study. As per our observation, it is due to the balanced load among core developers of these projects who are active during the same time period. Though *Eclipse-CDT* has the same trait as far as the contribution pattern is concerned, but its commit quality is quiet low. We believe contribution from non-core developers is the reason. Furthermore, choice of source code management for repository management also matters a lot in attracting contributors. We found that as projects (e.g. *glibc)* shifted from traditional SCM systems to modern SCM such as Git, the code contribution process improved. We aim to extend the work further to see the semantic quality of commits. Another proposal is to see the commit message quality of different types of commits such as corrective v/s non-corrective. Future work should also investigate the relevance of commit message quality with quality of the code contributed as part of commits.

References

1. Kapil Agrawal, Sadika Amreen, and Audris Mockus. 2015. *Commit quality in five high performance computing projects*. In Proceedings of the 2015 International Workshop on Software Engineering for High Performance Computing in Science, IEEE Press, pp. 24-29.
2. Iftekhar Ahmed, Soroush Ghorashi, and Carlos Jensen. 2014. *An Exploration of Code Quality in FOSS Projects*. OSS 2014, IFIP (International Federation for Information Processing), AICT 427, Corral, L.et al. (Eds.), pp. 181–190. Springer, Berlin, Heidelberg.
3. Oliver Arafat and Dirk Riehle. 2009. *The Commit Size Distribution of Open Source Software*. In Proc. HICSS'09 (Hawaii, USA, January 5-8, 2009). IEEE Computer Society Press, New York, NY, 2009, 1-8.
4. Amir Azarbakht and Carlos Jensen. 2014. *Drawing the Big Picture: Temporal Visualization of Dynamic Collaboration Graphs of OSS Software Forks*. OSS 2014, IFIP (IFIP International Federation for Information Processing) AICT 427, Corral L., *et al.* (Eds.).
5. Chris Beams. 2016. How to write a git commit message. http://chris.beams.io/posts/git-commit/[retrieved on 26 March 2016.
6. Evangelia Berdou. 2011. Organization in Open Source Communities: At the Crossroads of the Gift and Market Economies. Routledge.

7. Christian Bird and Nachiappan Nagappan . 2012. Who? Where? What? Examining Distributed Development in Two Large Open Source Projects. Proceedings of the 9th IEEE Working Conference on Mining Software Repositories, 237–246.
8. Tadeusz Chełkowski1, Peter Gloor, and Dariusz Jemielniak. 2016. Inequalities in Open Source Software Development: Analysis of Contributor's Commits in Apache Software Foundation Projects. PloS one 11, 4 (April, 2016).
9. D.J.Marcolesco.(retrieved on 28 July 2016).Writing good commit messages. https://github.com/erlang/otp/wiki/Writing-good-commit-messages
10. Paul A. David and Francesco Rullani. 2008. Dynamics of innovation in an "open source" collaboration environment: lurking, laboring, and launching FLOSS projects on SourceForge. Industrial and Corporate Change, 17(4) :647-710.
11. Amir Hossein Ghapanchi, Aybüke Aurum, and Farhad Daneshgar. 2012. The impact of process effectiveness on user interest in contributing to the open source software projects. Journal of software, 7(1): 212-219.
12. Jesus M. Gonzalez-Barahona, Gregorio Robles, Israel Herraiz, and Felipe Ortega. 2014. Studying the laws of software evolution in a long lived FLOSS project. Journal of Software: Evolution and Process, 26(7):589-612.
13. Carsten Kolassa, Dirk Riehle, and Michel Salim. 2013. The Empirical Commit Frequency Distribution of Open Source Projects. In: Proceedings of the 2013 joint International Symposium on Wikis and Open Collaboration, OpenSym'13, ACM.
14. Jérôme Kunegis, Sergej Sizov, Felix Schwagereit, and Damien Fay. 2012. Diversity dynamics in online networks. In: Proc. of the 23rd ACM Conf. on Hypertext and Social Media, USA.
15. Victor Kuechler, Claire Gilbertson, and Carlos Jensen. 2012. Gender Differences in Early Free and Open Source Software Joining Process. In: Hammouda, I., Lundell, B., Mikkonen, T., Scacchi, W. (eds.) OSS 2012. IFIP AICT, vol. 378, pp. 78–93. Springer, Heidelberg.
16. Tom Mens and Mathieu Goeminne .2011. Analysing the evolution of social aspects of open source software ecosystems. Eds: Jansen, Bosch, Ahmed, and Campell Proceedings of the Workshop on Software Ecosystems (IWSECO 2011).
17. Munish Saini and Kuljit Kaur. 2016. Change Profile Analysis of Open Source Software Systems to Understand their Evolutionary Behavior. Frontiers of Computer Science, Springer.
18. Eddie Santos and Abram Hindle. 2016. Judging a commit by its cover: correlating commit message entropy with build status on travis-CI. In Proceedings of the 13th International Conference on Mining Software Repositories (MSR '16). ACM, New York, NY, USA, 504-507.
19. G. Seber and Alan Lee. 2012. Linear regression analysis. Vol. 936. John Wiley & Sons.
20. Winters R. Scott. Score Normalization as a Fair Grading Practice. http://www.ericdigests.org/2003-4/score-normilization.html [retrieved on 20 July 2016].
21. Rodrigo Souza and Bruno Silva. 2017. Sentiment Analysis of Travis CI Builds. 2017. 14th International Conference on Mining Software Repositories.
22. M. R. Martinez Torres, S. L. Toral, M. Perales, and F. Barrero. 2011. Analysis of the Core Team Role in Open Source Communities. In: 2011 Int. Conf. on Complex, Intelligent and Software Intensive Systems (CISIS), pp. 109–114. IEEE.
23. Stanislav Levin and Amiram Yehudai. 2017. Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes. Pceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering Pages 97-106, Toronto, Canada — November 08 - 08, 2017