# What's a Typical Commit? A Characterization of Open Source Software Repositories

Abdulkareem Alali, Huzefa Kagdi, Jonathan I. Maletic
*Department of Computer Science*
*Kent State University*
*Kent Ohio 44242*
*{aalali, hkagdi, jmaletic}@cs.kent.edu*

## Abstract

*The research examines the version histories of nine open source software systems to uncover trends and characteristics of how developers commit source code to version control systems (e.g., subversion). The goal is to characterize what a typical or normal commit looks like with respect to the number of files, number of lines, and number of hunks committed together. The results of these three characteristics are presented and the commits are categorized from extra small to extra large. The findings show that approximately 75% of commits are quite small for the systems examined along all three characteristics. Additionally, the commit messages are examined along with the characteristics. The most common words are extracted from the commit messages and correlated with the size categories of the commits. It is observed that sized categories can be indicative of the types of maintenance activities being performed.*

## 1. Introduction

During software evolution the continuous change of a software system is typically recorded in version control systems such as *subversion* or *CVS*. This history is a valuable source of data for understanding the evolution process [8]. In the work presented here we aim to better understand how developers typically commit changes to these repositories. Surprisingly, there has been little empirical work characterizing what a typical or normal commit looks like. We feel that a better understanding of this process will allow us to predict the characteristics of commits for given maintenance tasks. So we are trying to understand not just a single version of a system but multiple versions of a system in the context of evolution.

Commits record changes to the software system. Here, we limit our study to source code changes. Changes take place any time a developer adds, modifies, or deletes something in the source code. To address our goal of understanding how source code changes, we examine three size-based characteristics of commits:

1. Number of files being added, modified, or deleted together in a commit
2. Number of lines being added, modified, or deleted in a commit
3. Number of hunks being added modified or deleted in a commit

The files that are committed together are used for the first characteristic and the line count is determined using the Unix tool *diff*. A *hunk* is a continuous group of lines that are changed along with contextual unchanged lines. The number of hunks is also determined using *diff*.

The study presented here examines nine open source projects, each with years of evolution history. After collecting the data on all these systems, we categorize each characteristic into five intervals from extra-small to extra-large, present the distributions for each system, and give the trends over all systems. Moreover, we examine if there is a relationship between any of the three characteristics.

Delving a bit deeper we examine the commits in the context of their corresponding log (or commit) message. When a change is committed developers add a message describing the change. Standards for commit message content are sometimes in place for projects and they may use a preset vocabulary. We calculated the distribution of vocabulary terms over the commits and present the most often used terms over each size category of commits. While no solid conclusions can be drawn in this coarse view of the data, it is apparent that a more investigation is needed.

The paper is organized as follows. The details of our approach are given in section 2. The results of the categorization performed for the three characteristics and the distribution of vocabulary terms for each category are presented in sections 3 and 4, respectively. Related work is given in 5 and we end with conclusions and future work.

## 2. Approach

We now give a brief background on version control systems and the commit process to better understand the type of data we are collecting. Then we describe how we collected the data necessary from the systems studied. Finally, we describe the methods used to categorize the data.

### 2.1. Commits in Software Repositories

Source code repositories store metadata such as user-IDs, timestamps, and commit comments in addition to the source code artifacts and their differences across versions. This metadata explains the why, who, and when dimensions of a source code change. Modern source-control systems, such as *Subversion*, preserve the grouping of several changes in multiple files to a single change-set as performed by a committer. Version-number assignment and metadata are associated at the change-set level and recorded as a log entry.

Figure 1 shows a log entry from the *Subversion* repository of *kdelibs* (a part of *KDE* repository). A log entry corresponds to a single *commit* operation. This commit log information can be readily obtained by using the command–line client *svn log* and a number of APIs (e.g., *pysvn*). *Subversion*'s log entries include the dimensions *author*, *date*, and *paths* involved in a change-set. In this case, the changes in the files *khtml_part.cpp* and *loader.h* are committed together by the developer *kling* on the date/time *2005-07-25T17:46:20.434104Z*. The *revision* number *438663* is assigned to the entire change-set (and not to each file that is changed as is in the case with some version-control systems such as CVS). Additionally, a text message describing the change entered by the developer is also recorded. Note that the order in which the files appear in the log entry is not necessarily the order in which they were changed.

```
<?xml version="1.0" encoding="utf-8"?>
<log>
  <log entry revision="438663">
    <author>kling</author>
    <date>2005-07-25T17:46:20.434104Z</date>
    <paths>
      <path action="M">khtml_part.cpp</path>
      <path action="M">loader.h</path>
    </paths>
    <msg>
       Do pixmap notifications when
       running ad filters.
    </msg>
  </log entry>
</log>
```

**Figure 1. A Snippet of kdelibs Subversion Log**

### 2.2. Commit Size Measures

It is evident from the previous subsection that it is fairly straightforward to determine the files involved in a commit. So the number of files could be used as the macro-level size measure of a commit. This file-level size is also cheap to compute, as only the log-entries need to be examined and does not need any processing of the files for their content changes. However, a file-based measure may not necessarily be a true reflection of the extent of changes. Furthermore, such a measure may not be useful in decisively comparing two different commits. For example, consider a commit with five files and another commit with ten files. A file-based measure would indicate that the commit with a larger number of files is 'larger'. However, a fair question to ask is, is this result really the representative of the size of the changes in these commits?

In an attempt to get a zoomed-in picture, we further process the files to the granularity of line and hunk differences. Here, we use the *GNU diff* utility to compute both line and hunk differences. Here, the term hunk has the same meaning as in the output of a unified *diff* format. A hunk basically represents the ranges of lines that contain the changes in the two versions of the same file. The number of such ranges for a file approximately depends on the contiguous changed and unchanged lines. Therefore, we believe that hunks are an indicator of the different regions that are affected by changes. A large number of hunk changes in a file could represent changes that ripple throughout. On the other hand, a large number of lines forming only a single hunk could represent a highly localized, well-understood change, or something as trivial as a change in a header comment with the license information. Therefore, hunks could effectively address some of the 'overestimation' that might result from a raw, micro-level line measures.

In our previous example, if the commit with five files had a total of 50 lines changed and the commit with ten files had a total of 10 lines changed, obviously the results of a line-based measure would be the exact opposite of that of a file-based measure. Another perspective to consider is the spread of the line changes in a file. This is exactly what is provided by a hunk-based measure. Once again, in our example, if the commit with five files and 50 lines were restricted to five hunks, and the commit with ten files and 10 lines were spread across ten hunks, we again have a different result. Clearly, none of the three measures alone is clearly decisive with an ironclad logic. Therefore, in our investigation we consider all the three kinds of size measures (size-base characteristics) for a commit and leave the issue open to an empirical investigation.

*File-Size:* the total number of files that are added, deleted, and/or modified in a commit.

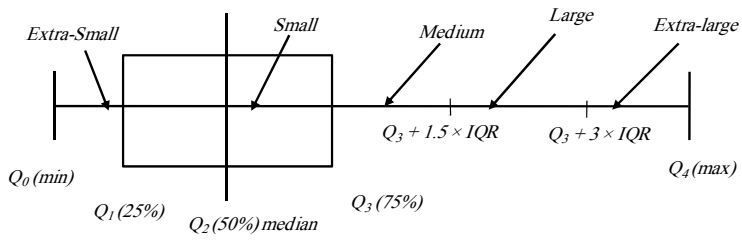


**Figure 2. A box plot showing the Inter Quartile Range (IQR) regions used to categorize commits into five categories.**

*Line-Size:* the total number of lines that are added, deleted, and/or modified of all the files in a commit.

*Hunk-Size*: the total number of hunks with line changes, i.e., added and/or deleted, in all the files in a commit.

Now that we described our size measures of interest, we discuss next our method of characterizing commits.

### 2.3. Characterizing Commits

To categorize commits into different categories, looking at a single commit in isolation is of little use in the absence of predefined classification criteria. Here, we first take a holistic view of all the commits in a given duration of the history of a software system, and then use a descriptive statistics method to classify them into different categories. For each commit, the three size measures are computed. We use the values of these measures as the data points for classification.

We categorize the size data through their 5-Point summaries: 1) the minimum observation $Q_0$; 2) the lower quartile $Q_1$; 3) the median $Q_2$ 4) the upper quartile $Q_3$; and 5) the maximum observation $Q_4$. The observation $Q_1$ is the upper edge for the smallest 25% of the data; the median $Q_2$ is up to 50% and the third quartile holds 75% of the data. Therefore, it has the 50% of data that surrounds the median. The Inter Quartile Range (*IQR*) is the data points covered in the range of quartiles $Q_3$ and $Q_1$, i.e., $Q_3 - Q_1$. We use the 5-point summary to distribute the data over seven regions:

1. Extreme outliers downward $Q_1$ ($Q_0 - Q_1$-$3\times IQR$),
2. Mild outliers downward $Q_1$ ($Q_1$-$3\times IQR - Q_1$-$1.5\times IQR$),
3. Non-outliers downward $Q_1$ ($Q_1$-$1.5\times IQR - Q_1$),
4. In the box ($Q_1 - Q_3$),
5. Non-outliers upward $Q_3$ ($Q_3 - Q_3 + 1.5\times IQR$),
6. Mild outliers upward $Q_3$ ($Q_3 + 1.5\times IQR - Q_3 + 3\times IQR$) and
7. Extreme outliers upward $Q_3$ ($Q_3 + 3\times IQR - Q_4$),

This categorization can be displayed using a Boxplot, a box-and-whisker plot [7] that is a histogram-like method for displaying data.

The distribution of our data is divided into five regions. In the systems examined here, we did not find a case that had a commit fall in the extreme-outlier or the mild-outlier regions beneath $Q_1$. Since our characteristics are measurement of size, we assigned size units names to our regions. In the case of our data, the number of files, the number of lines, and the number of hunks per commit can each be distributed over these five regions. We classify the size of each commit based on these regions as extra-small, small, medium, large, and extra-large as shown in Figure 2.

**Table 1. Nine open source systems used in study.**

| System | Duration | Versions |
|---|---|---|
| *gcc* | 8 years | 54536 |
| *Collab* | 5.7 years | 20288 |
| *JEdit* | 6.1 years | 2467 |
| *Ruby* | 9 years | 10667 |
| *LinuxBoss* | 7.9 years | 3023 |
| *Phpmyadmin* | 6.7 years | 6028 |
| *MySql-Administrator* | 1.3 years | 384 |
| *Python* | 6 years | 20420 |
| *Debian-installer.* | 7.5 years | 40425 |

The data and quartiles for *gcc* are given in Table 2 and the corresponding boxplot for files in Figure 3.

We now present the results of our categorization for all the systems we studied.

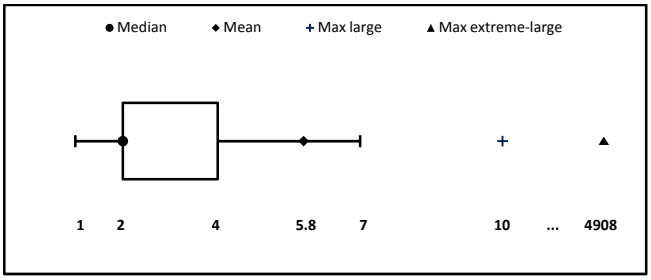


**Figure 3. Boxplot for the number of files per commit for *gcc*.**

## 3. Typical Size of Commits

Here, we want to determine the size range in which most of the commits fall, i.e., what is the size of a typical commit. Let us consider the histogram given in Figure 4 for the file-size, line-size, and hunk-size measures along with the raw data in Table 2. This data

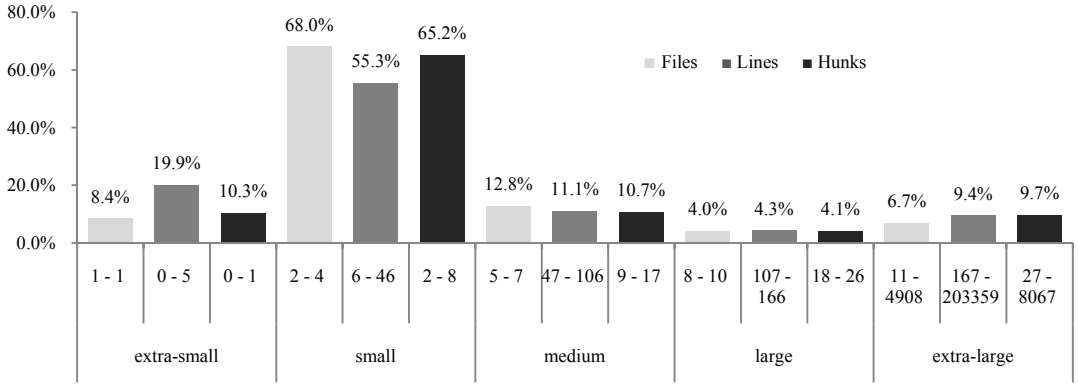


**Figure 4. Histogram for the number of commits distributed over number of files, lines, and hunks that were changed for *gcc* (54,536 commits over 8 years).**

is for the *GNU gcc* system over 50 thousand commits performed during its eight years of history. Our first observation is the right skew to the distribution. That is most of the commits are in the small or extra small categories.

In all three cases (files, lines, and hunks) approximately 75% of the commits are small or extra-small. However, larger commits do happen with a non-trivial frequency. The largest commits are often those that touch every file (e.g., updating the GPL license in every file's header comment) or add/modify a large file.

Given this trend of most commits being fairly small across all three characteristics, we now investigate if any of these characteristics are statistically correlated to one other. That is, if a commit has a small number of files, does that also indicate that a small number of lines were changed?

## 3.1. Correlation between Characteristics

We now address two related questions concerning *gcc* and later extend this to all the nine projects. First, we see if there is a correlation between any of the two different size measures. That is, if there are a large number of lines added, is there also a large number of files changed, or a large number of hunks changed? Second we also examine if there is a correlation between any two measures for an individual size category. The second issue gives us a fine-grained look at the data from various different categories (e.g., extra-small and small). To answer these questions, we investigate the linear correlation coefficient and the coefficient of determination [7].

The linear correlation coefficient (denoted by $r$) [7] is a quantity between the values -1 and +1 and measures the strength and the direction of a linear relationship between two variables $x$ and $y$, the formula for computing $r$ is, where $n$ is the number of pairs of data:

**Table 2. *gcc* commits categorized by size for each of the three characteristics. The duration of the commits was over eight years.**

| **Quartiles** | **Number Of Files** | | | **Number Of New Lines** | | | **Number Of Hunks** | | |
|---|---|---|---|---|---|---|---|---|---|
| ***$Q_0$ (Min)*** | 1 | | | 0 | | | 0 | | |
| ***Q1*** | 2 | | | 6 | | | 2 | | |
| ***$Q_2$ Median*** | 2 | | | 14 | | | 3 | | |
| ***$Q_3$*** | 4 | | | 46 | | | 8 | | |
| ***$Q_4$ (Max)*** | 4908 | | | 203359 | | | 8067 | | |
| ***IQR*** | 2 | | | 40 | | | 6 | | |
| **Boxplot** | **Range** | **Freq.** | **Ratio** | **Range** | **Freq.** | **Ratio** | **Range** | **Freq.** | **Ratio** |
| **x-Small** | 1 - 1 | 4580 | 8.4% | 0 - 5 | 10844 | 19.9% | 0 - 1 | 5625 | 10.3% |
| **Small** | 2 - 4 | 37100 | 68.0% | 6 - 46 | 30162 | 55.3% | 2 - 8 | 35562 | 65.2% |
| **Medium** | 5 - 7 | 6980 | 12.8% | 47 - 106 | 6072 | 11.1% | 9 - 17 | 5862 | 10.7% |
| **Large** | 8 - 10 | 2201 | 4.0% | 107 - 166 | 2335 | 4.3% | 18 - 26 | 2224 | 4.1% |
| **x-Large** | 11 - 4908 | 3675 | 6.7% | 167 - 203359 | 5123 | 9.4% | 27 - 8067 | 5263 | 9.7% |

$$r = \frac{n\sum xy - (\sum x)(\sum y)}{\sqrt{n(\sum x^2) - (\sum x)^2}\ \sqrt{n(\sum y^2) - (\sum y)^2}}$$

The value of $r$ is such that $-1 < r < +1$. The + and - signs are used for positive linear correlations and negative linear correlations, respectively. Positive and negative correlation is determined as follows. If $x$ and $y$ have a strong positive linear correlation, $r$ is close to +1, while a strong negative linear correlation, $r$ is close to -1. An $r$ of exactly +1 or -1 indicates a perfect positive or negative fit, respectively. Positive values indicate that as values for $x$ increases, values for y also increase and negative values are the reverse. A perfect correlation of ± 1 occurs only when the data points all lie exactly on a straight line.

If there is no linear correlation or a weak linear correlation, $r$ is close to 0. A value near zero means that there is a random or nonlinear relationship between the two variables. A correlation greater than 0.8 is generally described as strong, whereas a correlation less than 0.5 is generally described as weak. By convention we accept a *p-value* of 0.05 or less as being statistically significant. For all of our correlation values presented here we got statistically significant results with 95% confidence level and have a *p-value* less than 0.05.

The coefficient of determination, $r^2$, is a measure that allows us to determine how certain one can be in making predictions from a certain model/graph. The coefficient of determination is such that $0 < r^2 < 1$, and denotes the strength of the linear association between $x$ and $y$. It represents the percent of the data that is the closest to the line of best fit. For example, if $r = 0.922$, then $r^2 = 0.850$, which means that 85% of the total variation in $y$ can be explained by the linear relationship between $x$ and $y$. The other 15% of the total variation in $y$ remains unexplained.

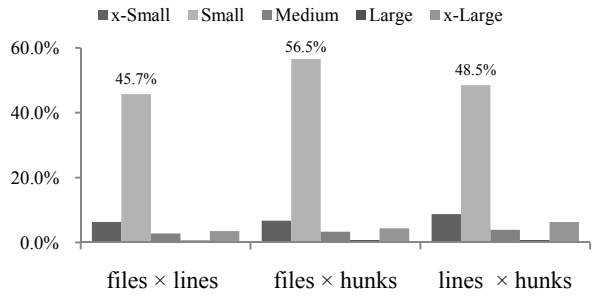


**Figure 5. Histogram the commonality between the metrics. This gives the percentage of commits that are common between metric for each size category.**

Before examining the values or $r$ and $r^2$, let us present a picture of the percentages of commonality between the three characteristics over the five categories. Figure 5 gives the cross intersection distributions over the three characteristics. We see three groups of columns. The first is the cross intersection of files × lines, next is files × hunks, and last is lines × hunks. The cross intersection between each characteristic are similar across the size categories. That is, when a commit has a small number of files changed, it also has a small number of lines changed with the highest frequency compared to the total. In computing this distribution we simply count the commits that are the same (i.e., have equal commit ID) that have the same size category for both characteristics and divide that by the total number of *gcc* commits. So we see approximately 50% commonality between the small size commits, whereas all the other sizes have commonality with less than 10%.

Now, we examine the correlations between characteristics. First, we address the question: if there are a large number of lines added was there also a large number of files changed or a large number of hunks changed? To answer this question we need to calculate the correlation over regions. That can be calculated as follows:

- We take any two characteristics and for each commit we consider the first characteristics value as $x$ and the other as $y$, the characteristics value is the commit size, the commit size can be represent by numbers (extreme-small as 0, …, extreme-large as 4 for *gcc*).
- Consider $x$ as number of files and $y$ can be the number of hunks. The *gcc* commit with ID# 31155 is a medium size commit as number of files $x = 2$ and a small size commit under the number of hunks metric $y = 1$, we repeat this for the left *gcc* commits.
- Then calculate $r$ and $r^2$ for $x$ and $y$.
- We repeat that for the two other combinations.

The values of $r$ and $r^2$ for *gcc* are given in Table 3. We only see a strong correlation between number of hunks × number of lines but only 60% of commits can be explained in a linear relationship (based on $r^2$) while 40% cannot. The remainders show no strong correlation and have low $r^2$ values.

**Table 3. *gcc* $r$ and $r^2$ values for all size metrics combinations.**

| *gcc* | files × lines | files × hunks | hunks × lines |
|---|---|---|---|
| $r$ | 0.5 | 0.7 | 0.8 |
| $r^2$ | 0.3 | 0.4 | 0.6 |

This result is hardly surprising. Commits can have the same number of line changes but very different

number of file changes. For example a change can consist of one file with 100 lines changed or a change can be 100 files with one line changed for each file. Thus, there should be little correlation between these two characteristics. The two measures that do demonstrate the most correlation are hunks and lines. This makes sense given that hunks are a group of continuous lines with changes. Therefore, as the number of hunks increases so does the number of lines.

Now let us look at the data in a bit more detail and address the next question: Is there a correlation between any two measures for a specific a size category? That can be calculated as follows:

- We take any two characteristics and then pick any size category
- The first characteristic from the chosen category is *x* and the other is *y*
- Then calculate *r* and $r^2$ for *x* and *y*.
- Then we do this for the remaining size category and assign new *x* and *y* until we finish all sizes.
- Then we repeat that for the other combinations.

Figure 6 gives the correlation coefficient for each of the two characteristics projected over each size category for *gcc*. This more detailed examination reconfirms the previous findings. So when taken separately, no size category combination has a strong correlation and most have a weak correlation at best.

Again, this is not surprising given the characteristics being measured. Moreover, we view this as a very important and positive result. Since these characteristics of commits are not correlated they can be used in conjunction to identify particular practices and/or characteristics. A simple example of this is a very large number of files committed yet only a small number of lines changed. This is indicative of a global change to a header file or the like. While we have not investigated this aspect further it is of great interest for future work.

Of course, we have only presented the finding for *gcc*. We will now discuss the correlations over all nine projects.

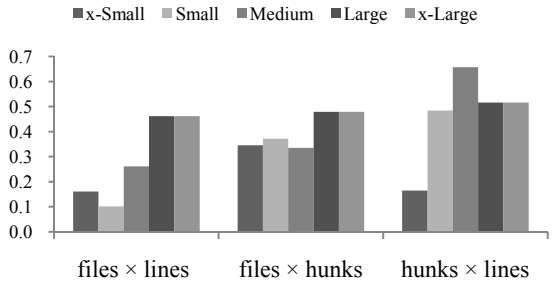


**Figure 6. Histogram the correlation coefficients (*r*) between each two measures over each size category for *gcc*.**

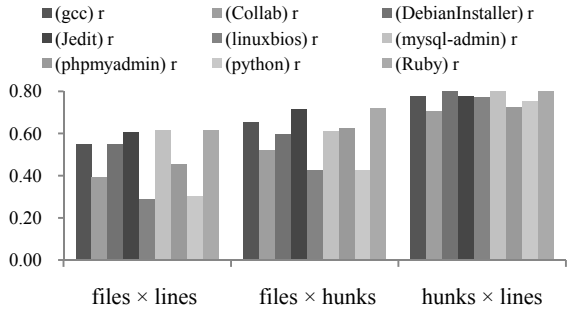


**Figure 7. Histogram the correlation coefficients (*r*) between each two size measures to nine projects.**

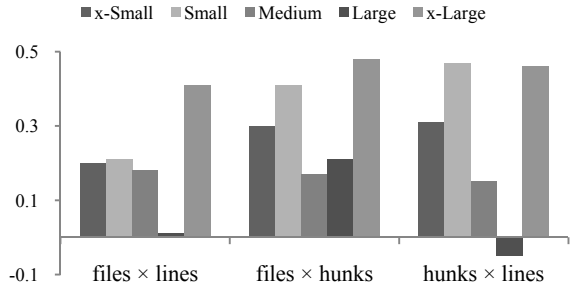


**Figure 8. Histogram the correlation coefficients (*r*) between each two size measures over each size category to nine projects.**

As in the case study *gcc*, we apply our procedure to the remaining eight open source projects. The correlation over size-based characteristics is given in Figure 7. We have three groups of columns, one for each cross combination files × lines, files × hunks, and hunks × lines. For each cross we have nine bars, each presents the value of *r* (how much the crossing measures correlated). For example, the cross hunks × lines for *Ruby* shows a strong correlation and has an $r^2$ of 0.7. This means that 70% of the commits can be explained in a linear relationship while 30% cannot. However, the same trend for *gcc* seems to hold for all nine systems. Additionally, when we did the analysis on each individual size categories the trends were very similar to what we saw for *gcc* (see Figure 8). That is, there is little correlation among the three characteristics we measured of commits.

Next, we further examine the idea of a typical commit but now with respect to the vocabulary used in the log messages for each size category.

## 4. Vocabulary versus Commit Size

Given the size categorization described previously, we now try to understand what types of changes usually (or most commonly happen) for each size category. To do this we build a vocabulary of the most frequent words used in commits' log messages over the nine projects. As a threshold we consider the top most

frequent terms. Then we find the top most frequent terms-sets for each size over the nine projects together over each size category.

For each project, we collected the log messages and eliminated stop words using the Lovins stemmer algorithm [9]. This algorithm removes suffixes from words. It uses a predefined list of about 250 different suffixes, and removes the longest suffix attached to the word. The stem after the removed suffix is at least three-characters long. We take the stemmed words and count the frequencies of each word by considering each commit as a customer basket [1]. The result is a ranked list of frequent terms for each project. Then we cross join those nine lists and take the top most 50 frequent terms.

Table 4 has a list of the top 25 terms used across all nine projects. We see that terms such as fix, add, test, and file are prevalent, as to be expected.

**Table 4. Top 25 average term frequency over all nine systems.**

| Term | Average Rank |
|---|---|
| fix | 36.05% |
| add | 18.22% |
| test | 13.96% |
| file | 11.85% |
| new | 11.54% |
| support | 11.35% |
| chang | 11.14% |
| bug | 10.88% |
| patch | 10.03% |
| code | 9.41% |
| remov | 9.38% |
| set | 8.45% |
| work | 8.24% |
| updat | 7.92% |
| get | 6.75% |
| error | 5.93% |
| build | 5.73% |
| function | 5.57% |
| typo | 4.70% |
| call | 4.66% |
| messag | 4.65% |
| includ | 4.52% |
| path | 4.52% |
| need | 4.49% |

We take this list and identify frequent term combinations (as in mining frequent itemsets) considering each commit in a project as our customer basket. Here we are looking for two or more terms per set. We take each characteristic separately. For each size category we use a regular expression search to look for term combinations and count the occurrences of each set in the commits. This results in a list of term combinations and we generate these ranked frequent term sets for the rest of the projects

The results are given in Table 5, Table 6, and Table 7 with ranked lists of frequent term combinations (sets) and the their support. From the tables we see a number of interesting trends. In particular, we see that even over different projects there is commonality of terms and the frequency of their usage. This gives some credence to the idea of using a set of predefined terms for use in commit messages.

For the file measure {file, fix}, {fix, use}, and {file, update} are the most frequent sets on average. In the extra large category {file, fix} is a very frequently occurring set.

For line changes {file, fix}, {add, bug}, {fix, use}, and {remov, test} (in that order) are most frequent. The set {add, bug} is very frequent in xtra-small line changes.

For the changes to hunks {add, bug}, {file, fix}, {fix, use}, and {remov, test} (in that order) are the most frequent sets. Again, {add, bug} is very prevalent for extra-small changes.

The finer granularity of using the line measure versus the file measure seems to result in larger support values for term sets when using lines as your characteristic in the extra-small category.

More investigation is necessary to see if particular types of changes correspond to size categories over the different measures. However, there is at least some evidence of this phenomenon.

## 5. Related Work

The unit of change studied here is the commit in a *subversion* repository. Whether it affects the system in minor or major way, it saves rich information about the evolution of the system. Researchers are particularly interested in different definitions of a point of change. German [6] studied the characteristics of a Modification request (MR). A group of files is considered to be in the same MR if they have the same creator, same log, and over a given period of time (calculated in [6]). Some main observations were that bugMRs contain few files and commentMRs contain a large number of files. Standard MRs contains at least two files and most of them are small. Most MRs are composed of files that belong to the same module. The maintenance period has a fewer MRs than an improvement period. Most files are modified owned by one individual developer.

Hindle et. al [14] studied release artifacts that can be accessed from the software repositories. Their purpose was to find release patterns that happen around a release. They portioned revisions into four classes (source code, test, build, and documentation revisions) and found that MySQL documentation and testing commits occur more before a release than after. While

the build commits increased, source code changes dipped around release time. In [2] authors presented a semi-automated approach to assign issue reports to developers. A machine-learning algorithm is used on bug reports to learn the kinds of reports each developer resolves.

Robles et al [13] studied the evolution of a software distribution. The author characterized the number of packages, lines of code, use of programming languages, and sizes of packages/files with regards to the evolution of a software distribution. The work there covered five stable releases of Debian within seven-year duration. They observed that overall size of Debian doubled every two years. Fewer large packages (over 100 KLOC) than small packages (1 KLOC to 50 KLOC) were in all of the releases. Large packages were shown to increase in subsequent releases and more small packages were added. The most used programming language in each release is C. Using C decreased in subsequent releases from 76.7% in release 2.0 to 55.8%. The usage percentage of the interpreted languages such as Python and Perl shows a sharp growth. File sizes of programs written in the procedural and structural languages are larger than those written in the object-oriented languages.

Dinh-Trong and Bieman [5] validate five of the seven hypotheses pertaining to successful open-source software development given by Mockus [10] in their empirical studies on Apache and Mozilla. Authors examined whether the FreeBSD project supported the seven hypotheses proposed by Mockus et al. the data gathered were enough to evaluate hypotheses H1, H2, H3, H5, and H6. The result ends is up supporting three hypotheses and the others need more revision: the supported hypotheses were: H3: In successful open source developments, a group larger by an order of magnitude than the core will repair defects, and a yet larger group (by another order of magnitude) will report problems. H5: Defect density in open source releases will generally be lower than commercial code that has only been feature tested, that is, received a comparable level of testing. H6: In successful open source developments, the developers will also be users of the software.

Understanding the impact of small changes (one-line changes) with regards to faults, the relationship between different types of changes (i.e., add, delete, and modify), the reason for the change (i.e., corrective, adaptive, and perfective), and dependencies between changes were studied by Purushothaman et. al [11, 12]. They report these observations: Approximately 10% of changes were one-line changes. About 50% of changes involved at most 10 LOC, and about 95% of changes involved at most 50 LOC. The perfective category consisted of approximately 2.5% one-line additions and approximately 10% of other types of one-line changes. Most changes were found to be adaptive and contained addition of code. About 40% of changes introduced for fixing defects introduced at least one more defect. Only 4% of the one-line changes caused a defect. The chances of a one-line addition and modification causing a defect are approximately 2% and 5%, respectively. The chance of a defect occurring for a change that involved more than 500 LOC is about 50%. It remained inconclusive whether deletions of less than 10 LOC cause a defect.

Canfora et. al [3] presented an impact analysis of a change request and a fine grained analysis method of software repositories is used to index code at different levels of granularity, such as lines of code and source files, with free text contained in software repositories. The method employs information retrieval algorithms to link the change request description and code entities impacted by similar past change requests.

Chen et. al [4] created a tool (CVSSearch) that searches for fragments of source code by using CVS comments. CVSSearch allows one to better search the most recent version of the code by looking at previous versions to better understand the current version.

For more detailed background on Mining Software Repositories approaches see Kagdi's [8] survey of MSR approaches used under the context of software evolution.

## 6. Conclusion

We presented a study that investigated nine open source systems ranging across different application domains, programming languages, and sizes. The goal of the study was to infer the characteristics of a typical commit from years of historical information. The data that was obtained indicates that a large amount of commits are of very small sizes with respect to file (2-4), line (approximately less than 50), and hunk (approximately less than 8) measures. The statistical co-relationship analysis shows no relationship between file and line size measures, however does show a substantial co-relationship between the hunk and line measures. This shows that hunk is an equivalent indicator line changes. Furthermore, we analyzed vocabulary associated with the commits. The vocabulary data shows interesting sets of terms used across different categories of commits.

One observation is that the terms that suggest bug related changes are associated with fairly small-sized commits. We feel that this is an interesting set of data that will provide an insight into the purposes of the changes. In future, we plan to repeat the study with a different categorization technique (IQR was used here), and include changes in syntactic entities (e.g., class and methods/functions) for size-based measures of

commits. That is, how does a typical commit look like from a syntactic perspective?

## 7. References


[1] Agrawal, R. and Srikant, R., "Mining Sequential Patterns", in Proceedings of 11th International Conference on Data Engineering, Taipei, Taiwan, March 1995.

[2] Anvik, J., Hiew, L., and Murphy, G. C., "Who Should Fix This Bug?" in Proceedings of 28th International Conference on Software Engineering (ICSE'06), Shanghai, China May 20-28 2006, pp. 361-370.

[3] Canfora, G. and Cerulo, L., "Fine Grained Indexing of Software Repositories to Support Impact Analysis", in Proceedings of 3rd International Workshop on Mining Software Repositories (MSR'06), Shanghai, China May 22-23 2006, pp. 105-111.

[4] Chen, A., Chou, E., Wong, J., Yao, A. Y., Zhang, Q., Zhang, S., and Michail, A., "CVSSearch: Searching through Source Code using CVS Comments", in Proceedings of 17th IEEE International Conference on Software Maintenance ( ICSM'01) Florence, Italy, November, 6-10 2001, pp. 364-373.

[5] Dinh-Trong, T. T. and Bieman, J. M., "The FreeBSD Project: A Replication Case Study of Open Source Development", IEEE Transactions on Software Engineering, vol. 31, no. 6, 2005, pp. 481-494.

[6] German, D. M., "Mining CVS Repositories, the SoftChange Experience", in Proceedings of 1st International Workshop on Mining Software Repositories (MSR'04), Edinburph, Scotland, 2004, pp. 17-21.

[7] Johnson, R. A. and Wichern, D. W., Applied Multivariate Statistical Analysis, 4th ed., Prentice Hall, 1998.

[8] Kagdi, H., Collard, M. L., and Maletic , J. I., "A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution", Journal of Software Maintenance and Evolution: Research and Practice, vol. 19, no. 2, March/April 2007, pp. 77-131.

[9] Lovins, J. B., "Development of a stemming algorithm", Mechanical Translation and Computational Linguistics, vol. 11, 1968, pp. 22-31.

[10] Mockus, A., Fielding, T., and Herbsleb, D., "Two Case Studies of Open Source Software Development: Apache and Mozilla", ACM Transactions on Software Engineering and Methodology vol. 11, no. 3, July 2002, pp. 309-346

[11] Purushothaman, R. and Perry, D. E., "Towards Understanding the Rhetoric of Small Changes", in Proceedings of 1st International Workshop on Mining Software Repositories (MSR'04), Edinburgh, Scotland, UK, May, 25 2004, pp. 90-94.

[12] Purushothaman, R. and Perry, D. E., "Toward Understanding the Rhetoric of Small Source Code Changes", IEEE Transactions on Software Engineering, vol. 31, no. 6, 2005, pp. 511-526.

[13] Robles, G., González-Barahona, J. M., Michlmayr, M., and Amor, J. J., "Mining Large Software Compilations Over Time: Another Perspective of Software Evolution", in Proceedings of 3rd International Workshop on Mining Software Repositories (MSR'06), Shanghai, China May 22-23 2006, pp. 3-9.

[14] Hindle, A., Godfrey, M. W., and Holt, R. C., "Release Pattern Discovery via Partitioning: Methodology and Case Study", in ACM Special Interest Group on Software Engineering Washington, DC, USA, 2007, pp. 19-27.


**Table 5. Frequent term set distribution, for all nine systems, over the number of files changed per commit for each sized category.**

| x-Small | | Small | | Medium | | Large | | x-Large | |
|---|---|---|---|---|---|---|---|---|---|
| **Terms** | **Support** | **Terms** | **Support** | **Terms** | **Support** | **Terms** | **Support** | **Terms** | **Support** |
| fix, typo | 1.18% | add, bug | 3.51% | file, fix | 6.12% | file, fix | 8.07% | file, fix | 10.18% |
| file, fix | 0.29% | file, fix | 3.06% | add, bug | 4.22% | add, bug | 5.07% | fix, use | 8.35% |
| fix, test | 0.27% | fix, typo | 1.99% | remov, test | 4.09% | fix, use | 4.16% | file, updat | 6.83% |
| remov, test | 0.26% | remov, test | 1.74% | fix, use | 2.62% | file, updat | 4.04% | add, bug | 6.75% |
| add, bug | 0.25% | fix, test | 1.60% | file, remov | 2.56% | bug, fix, work | 3.30% | bug, fix, work | 5.77% |
| fix, use | 0.22% | file, remov | 1.34% | fix, set | 2.48% | remov, test | 3.22% | file, remov | 4.30% |
| bug, fix, work | 0.20% | fix, use | 1.18% | fix, test | 2.23% | fix, set | 2.83% | remov, test | 4.28% |
| file, remov | 0.14% | fix, set | 1.10% | file, updat | 1.90% | doc, updat | 2.75% | doc, updat | 4.10% |
| doc, updat | 0.10% | file, updat | 0.83% | bug, fix, work | 1.86% | file, remov | 2.56% | fix, set | 3.81% |
| fix, path | 0.10% | doc, updat | 0.80% | doc, updat | 1.39% | fix, test | 2.40% | bug, fix | 3.47% |
| fix, set | 0.09% | error, messag | 0.77% | fix, typo | 1.11% | error, messag | 1.56% | fix, test | 3.35% |
| file, updat | 0.07% | bug, fix | 0.76% | error, messag | 1.10% | fix, typo | 1.22% | error, messag | 2.00% |
| error, messag | 0.05% | bug, fix, work | 0.75% | fix, path | 0.95% | bug, fix | 1.10% | bug, fix, use | 1.97% |
| bug, fix | 0.04% | fix, path | 0.63% | bug, fix | 0.94% | fix, path | 0.89% | fix, typo | 1.51% |
| bug, fix, use | 0.01% | bug, fix, use | 0.26% | bug, fix, use | 0.71% | bug, fix, use | 0.57% | fix, path | 1.30% |

**Table 6. Frequent term set distribution, for all nine systems, over the number of lines changed per commit for each sized category.**

| x-Small | | Small | | Medium | | Large | | x-Large | |
|---|---|---|---|---|---|---|---|---|---|
| **Terms** | **Support** | **Terms** | **Support** | **Terms** | **Support** | **Terms** | **Support** | **Terms** | **Support** |
| add, bug | 8.19% | file, fix | 2.96% | file, fix | 6.05% | file, fix | 6.28% | file, fix | 8.99% |
| file, fix | 2.81% | fix, typo | 2.14% | remov, test | 5.33% | add, bug | 4.69% | add, bug | 6.17% |
| fix, typo | 2.04% | add, bug | 1.87% | add, bug | 3.92% | fix, use | 3.99% | file, updat | 5.62% |
| fix, use | 1.87% | remov, test | 1.87% | fix, set | 3.17% | file, updat | 3.59% | fix, use | 5.23% |
| bug, fix, work | 0.86% | fix, test | 1.71% | fix, test | 2.71% | bug, fix, work | 2.80% | file, remov | 4.04% |
| fix, test | 0.72% | file, remov | 1.52% | file, remov | 2.62% | remov, test | 2.77% | fix, set | 3.75% |
| remov, test | 0.54% | fix, set | 1.06% | fix, use | 2.62% | file, remov | 2.76% | remov, test | 3.62% |
| fix, set | 0.47% | fix, use | 1.04% | file, updat | 2.30% | doc, updat | 2.69% | bug, fix, work | 3.59% |
| file, updat | 0.43% | bug, fix | 0.82% | bug, fix, work | 2.22% | fix, test | 2.56% | doc, updat | 3.34% |
| doc, updat | 0.41% | error, messag | 0.82% | doc, updat | 1.89% | fix, set | 2.54% | fix, test | 3.20% |
| file, remov | 0.29% | file, updat | 0.77% | error, messag | 1.22% | error, messag | 1.66% | bug, fix | 2.39% |
| fix, path | 0.22% | doc, updat | 0.77% | fix, typo | 1.09% | bug, fix | 1.56% | error, messag | 1.54% |
| bug, fix | 0.20% | fix, path | 0.73% | bug, fix, use | 0.87% | fix, typo | 0.96% | bug, fix, use | 1.42% |
| error, messag | 0.14% | bug, fix, work | 0.70% | fix, path | 0.85% | fix, path | 0.80% | fix, path | 1.23% |
| bug, fix, use | 0.04% | bug, fix, use | 0.27% | bug, fix | 0.81% | bug, fix, use | 0.50% | fix, typo | 1.02% |

**Table 7. Frequent term set distribution, for all nine systems, over the number of hunks changed per commit for each sized category.**

| x-Small | | Small | | Medium | | Large | | x-Large | |
|---|---|---|---|---|---|---|---|---|---|
| **Terms** | **Support** | **Terms** | **Support** | **Terms** | **Support** | **Terms** | **Support** | **Terms** | **Support** |
| add, bug | 8.77% | file, fix | 2.97% | file, fix | 5.04% | file, fix | 7.06% | file, fix | 6.63% |
| file, fix | 3.12% | fix, typo | 2.21% | add, bug | 3.79% | remov, test | 5.53% | fix, use | 6.44% |
| fix, use | 2.15% | add, bug | 1.79% | remov, test | 3.20% | add, bug | 4.72% | file, updat | 6.44% |
| fix, typo | 1.39% | remov, test | 1.71% | fix, use | 3.00% | fix, set | 4.48% | add, bug | 5.96% |
| fix, test | 0.75% | fix, test | 1.67% | fix, test | 2.48% | fix, use | 4.25% | fix, set | 4.21% |
| bug, fix, work | 0.65% | file, remov | 1.39% | file, remov | 2.28% | bug, fix, work | 4.03% | file, remov | 3.96% |
| remov, test | 0.64% | fix, use | 1.00% | file, updat | 2.20% | file, remov | 3.90% | bug, fix, work | 3.50% |
| fix, set | 0.56% | fix, set | 1.16% | bug, fix, work | 1.90% | file, updat | 3.25% | bug, fix | 3.49% |
| bug, fix | 0.42% | bug, fix | 1.03% | doc, updat | 1.78% | doc, updat | 3.19% | doc, updat | 3.29% |
| file, remov | 0.40% | doc, updat | 0.78% | fix, set | 1.70% | fix, test | 2.99% | fix, test | 3.29% |
| file, updat | 0.37% | file, updat | 0.74% | bug, fix | 1.53% | bug, fix | 1.87% | remov, test | 3.28% |
| doc, updat | 0.37% | error, messag | 0.82% | fix, typo | 1.42% | error, messag | 1.61% | bug, fix, use | 2.08% |
| fix, path | 0.22% | fix, path | 0.68% | error, messag | 1.08% | fix, typo | 1.14% | error, messag | 1.72% |
| error, messag | 0.16% | bug, fix, work | 0.66% | fix, path | 0.90% | bug, fix, use | 0.80% | fix, path | 1.43% |
| bug, fix, use | 0.08% | bug, fix, use | 0.26% | bug, fix, use | 0.78% | fix, path | 0.75% | fix, typo | 1.33% |