AN EMPIRICAL CHARACTERIZATION OF COMMITS IN SOFTWARE REPOSITORIES

A thesis submitted

to Kent State University in partial

fulfillment of the requirements for the

degree of Master of Science

by

Abdulkareem Alali

May, 2008
Thesis written by

Abdulkareem Alali

M.S., Kent State University, Kent, Ohio, USA 2008

B.S., Yarmouk University, Irbid, Jordan 2002

Approved by

Dr. Jonathan I. Maletic, Advisor

Dr. Robert A. Walker, Chair, Department of Computer Science

John R. D. Stalvey, Dean, College of Arts and Sciences
# TABLE OF CONTENTS

LIST OF FIGURES ................................................................................................................. V

LIST OF TABLES ..................................................................................................................... VII

ACKNOWLEDGEMENTS ......................................................................................................... VIII

CHAPTER 1 INTRODUCTION ................................................................................................. 1

1.1 Research Hypothesis and Questions ............................................................................... 2

1.2 Research Contributions .................................................................................................. 3

1.3 Organization of the Thesis ............................................................................................. 4

CHAPTER 2 RELATED WORK ................................................................................................. 5

CHAPTER 3 OVERVIEW OF APPROACH ............................................................................ 11

3.1 Commits in Software Repositories .................................................................................. 11

3.2 Commit Size Measures .................................................................................................. 13

3.3 Characterizing Commits ................................................................................................ 17

CHAPTER 4 TYPICAL SIZE OF COMMITS ......................................................................... 23

4.1 Correlation between Characteristics ............................................................................. 24

CHAPTER 5 VOCABULARY VERSUS COMMIT SIZE ......................................................... 33

5.1 Approach ....................................................................................................................... 33

5.2 Results .......................................................................................................................... 34
CHAPTER 6 CONCLUSIONS AND FUTURE RESEARCH ........................................ 40

6.1 Main Results ................................................................................................. 40

6.2 Future Research Directions ........................................................................ 41

REFERENCES ..................................................................................................... 42
LIST OF FIGURES

FIGURE 1. A SNIPPET OF KDELIBS SUBVERSION LOG ........................................ 13

FIGURE 2. AN EXAMPLE BOXPLOT. ........................................................................ 16

FIGURE 3. HISTOGRAM FOR THE NUMBER OF COMMITS DISTRIBUTED OVER NUMBER OF FILES THAT WERE CHANGED FOR GCC (54,536 COMMITS OVER 8 YEARS). ................................................................. 19

FIGURE 4. BOXPLOT FOR THE NUMBER OF FILES PER COMMIT FOR GCC. .. 20

FIGURE 5. HISTOGRAM FOR THE NUMBER OF COMMITS DISTRIBUTED OVER NUMBER OF LINES THAT WERE CHANGED FOR GCC (54,536 COMMITS OVER 8 YEARS). ................................................................. 21

FIGURE 6. BOXPLOT FOR THE NUMBER OF LINES PER COMMIT FOR GCC. .... 21

FIGURE 7. HISTOGRAM FOR THE NUMBER OF COMMITS DISTRIBUTED OVER NUMBER OF HUNKS THAT WERE CHANGED FOR GCC (54,536 COMMITS OVER 8 YEARS). ................................................................. 22

FIGURE 8. BOXPLOT FOR THE NUMBER OF HUNKS PER COMMIT FOR GCC. .... 22

FIGURE 9. HISTOGRAM THE COMMONALITY BETWEEN THE METRICS. THIS GIVES THE PERCENTAGE OF COMMITS THAT ARE COMMON BETWEEN METRIC FOR EACH SIZE CATEGORY ......................................................... 26

FIGURE 10. HISTOGRAM THE CORRELATION COEFFICIENTS (R) BETWEEN EACH TWO METRICS OVER EACH SIZE CATEGORY ............................................. 30
FIGURE 11. LINEAR CORRELATION ($R$) OVER REGIONS TO NINE PROJECTS.

........................................................................................................................................32
LIST OF TABLES

TABLE 1. GCC COMMITS CATEGORIZED BY SIZE FOR EACH OF THE THREE CHARACTERISTICS. THE DURATION OF THE COMMITS WAS OVER EIGHT YEARS. ................................................................. 16

TABLE 2. GCC $r$ AND $r^2$ VALUES FOR ALL SIZE METRICS COMBINATIONS. ......................................................................................................................... 28

TABLE 3. NINE OPEN SOURCE SYSTEMS USED IN STUDY. .................... 30

TABLE 4. TOP 25 AVERAGE TERM FREQUENCY OVER ALL NINE SYSTEMS. .............................................................................................................. 34

TABLE 5. FREQUENT TERM SET DISTRIBUTION, FOR ALL NINE SYSTEMS, OVER THE NUMBER OF FILES CHANGED PER COMMIT FOR EACH SIZED CATEGORY.................................................................................. 37

TABLE 6. FREQUENT TERM SET DISTRIBUTION, FOR ALL NINE SYSTEMS, OVER THE NUMBER OF LINES CHANGED PER COMMIT FOR EACH SIZED CATEGORY.................................................................................. 38

TABLE 7. FREQUENT TERM SET DISTRIBUTION, FOR ALL NINE SYSTEMS, OVER THE NUMBER OF HUNKS CHANGED PER COMMIT FOR EACH SIZED CATEGORY.................................................................................. 39
ACKNOWLEDGEMENTS

I want to acknowledge my "thesis family" (discard ordering): SDML (research lab), Wedad Alali (mother), Jonathan I. Maletic (supervisor), Qasem Alali (father), Walid Alali (brother), Gwenn Volkert (committee), Sireen Abu-Khafajah (fiancée), Bacim Alali (brother), Huzefa Kagdi (research partner), Michael Collard (committee), SDML members, CS/Kent State University, and Ruoming Jin (committee).

Also thanks for the support of: Tharwa Alali (sister), Feras Alali (brother), Mysoon Alali (sister), Khaled Al-z'ubi (brother in law), Ismaeel Alali (brother), Aya Alali (sister), Ahmad Abu-Khafajah (father in law), Mohammad Abu-Khafajah (fiancée brother), Shatha Abu-Khafajah (fiancée sister), Tamara Al-z'ubi (niece), Daleh Abu-Khafajah (mother in law), Hilal Nmeir (friend), Emran Matalka (friend), Abdullah Alsharo (friend), Jehad Rababah (friend), Abdulkareem Ghazi (friend), Jamal Al-sakran (friend).

Again, thanks to the closest soul - the love of my life Sireen.

Abdulkareem Alali

March, 2008, Kent, Ohio
CHAPTER 1

Introduction

During software evolution the continuous change of a software system is typically recorded in version control systems such as subversion\(^1\) or CVS\(^2\). This history is a valuable source of data for understanding the evolution process [Kagdi, Collard, Maletic 2007]. 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. The goal is to understand not just a single version of a system, but multiple versions of a system in the context of the system’s evolution.

Commits recorded by version control systems keep track of changes to the software system. Here, we limit our study to source code changes and the other artifacts of software evolution such as changes to external documentation are not examined. These commits (changes) take place any time a developer adds, modifies, or deletes something in the source code.

\(^1\) http://subversion.tigris.org/

\(^2\) http://www.nongnu.org/cvs/
1.1 Research Hypothesis and Questions

This thesis 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*, *CVS*). 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 all commits, for all the systems, are quite small 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. 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 typically in place for projects and often 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.

To address the goal of understanding how source code changes, three size-based characteristics of commits are examined:

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 number of files that are committed together are used for the first characteristic and for the line count we determine this using GNU\(^1\) \textit{diff} [Hunt, Vo, Tichy 1998]. A \textit{hunk} is a continuous group of lines that are changed together. The term hunk is widely used in the context of differencing and change analysis. The number of hunks can be easily determined using \textit{diff}.

### 1.2 Research Contributions

This is the first broad examination of the characteristics of commits in open source software systems. The work is of particular interest to the software engineering community and those conducting empirical studies on software repositories. A broad classification of research termed Mining Software Repositories is particularly impacted by the characteristics of commits. A number of assumptions about how developers commit changes is typically made in these studies and this work directly addresses when and how these assumptions can be made.

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

\(^{1}\) http://www.gnu.org/
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 typically in place for projects and often 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.

1.3 Organization of the Thesis

The Thesis is organized as follows: related work is given in CHAPTER 2. The details of our approach are given in CHAPTER 3. The results of the categorization of the three characteristics and the distribution of vocabulary terms for each category is presented in CHAPTER 4 and CHAPTER 5, respectively. We end with conclusions and future work in CHAPTER 6.
CHAPTER 2

Related Work

The unit of change used in our research was the commit in a Subversion repository. It is a point of time when a system changed. Whether it affects the system in a minor or major way, it produces rich information about software evolution. Researchers have used different definitions of a point of change. German [German 2004] 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 the same period of time (calculated in [German 2004]). Some main observations were that bugMRs contain few files and commentMRs contain a large number of files. Standard MRs contain at least two files and most of them are small with regard to files. Most MRs are composed of files that belong to the same module. The maintenance period has fewer MRs than an improvement period. Most files are modified and owned by one individual developer.

Hindle et. al [Hindle, Godfrey, Holt 2007] studied release artifacts that can be accessed from the software repositories. Their purpose was to find changes patterns that happen around a release. They portioned revisions into 4 classes
(source code, test, build and documentation revisions) and found that MySQL\(^1\) documentation and testing commits occur more before a release than after. While the build commits increased, source code changes dipped around release time. In [Anvik, Hiew, Murphy 2006] authors presented a semi-automated approach to assign issue reports to developers. A machine-learning algorithm is used on bug reports to learn kinds of reports each developer resolves.

Another study by Robles et al [Robles, González-Barahona, Michlmayr, Amor 2006] 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 here covered five stable releases of Debian\(^2\) 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) in all of the releases. Large packages were shown to increase in subsequent releases and more small packages were added. Most used programming language in each release is C\(^3\). Using C decreased in subsequent

\footnotesize
\begin{align*}
\text{1} & \text{ http://www.mysql.com/} \\
\text{2} & \text{ http://www.debian.org/} \\
\text{3} & \text{ http://cm.bell-labs.com/cm/cs/who/dmr/chist.html}
\end{align*}
releases from 76.7% in release 2.0 to 55.8%. The usage percentage of the interpreted languages such as Python and Perl\(^1\) 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 [Dinh-Trong, Bieman 2005] validate five of the seven hypotheses pertaining to successful open-source software development given by Mockus et al. [Mockus, Fielding, Herbsleb 2002] from their empirical studies on Apache\(^2\) and Mozilla\(^3\). Authors examined whether the FreeBSD\(^4\) 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 3 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

\(^1\) http://www.perl.org/

\(^2\) http://www.apache.org/

\(^3\) http://www.mozilla.org/

\(^4\) http://www.freebsd.org/
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 [Purushothaman, Perry 2004; 2005]. 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 [Canfora, Cerulo 2006] 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 [Chen et al. 2001] created a tool (CVSSearch\(^1\)) 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.

Mockus et. al [Mockus, Votta 2000] changes were classified as corrective, adaptive, perfective, and inspections. Inspections were mostly perfective and corrective changes, the number of such changes justified the introduction of a different type of change classification. In any systematic software development environment, code inspections and modifications of code following each inspection are standard procedures. Textual description field proved in their analysis to be essential to identify the reason for a change, and suspect that other properties of the change could be identified using the same field.

Eick et. al [Eick et al. 2001] demonstrate in his analysis for a change management data that code decays; a conceptual model proposed for code decay:

\(^1\) http://cvssearch.sourceforge.net/
a module is decayed if it is harder to change than it should be, measured in terms of effort, interval and quality. More analyses were performed: (1) An increase over time in the number of files touched per change to the code; (2) The decline in modularity of a subsystem of the code, as measured by changes touching multiple modules; (3) Contributions of several factors to fault rates in modules of the code; and (4) That span and size of changes are important predictors of the effort to implement a change.

Perry et. al. [Perry, Siy, Votta 1998 ] presented an empirical investigations to understand the nature of large scale parallel development. The data shows, each day, there is ongoing work on multiple MRs by different developers solving different IMRs belonging to different features within different releases of two similar products aimed at distinct markets. The activities within each of these levels cut across common files. Over the interval of a release, the number of files changed by multiple developers is 50% which, while not concurrent with respect to the MR level, is concurrent with respect to the release.

Kagdi et. al [Kagdi, Collard, Maletic 2007] reported an in-depth survey of Mining Software Repositories (MSR) approaches used under the context of software evolution.
CHAPTER 3

Overview of Approach

This chapter presents an overview of the approach. We start with 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 for the study from the systems studied. Finally, we describe the methods used to categorize the data.

3.1 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 [Kagdi, Maletic 2007]. 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 [Kagdi, Maletic 2006]. 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\(^1\) (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\(^2\)`). 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.

\(^1\) http://kdelibs.com/wiki/index.php/Main_Page

\(^2\) http://pysvn.tigris.org/
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 measure of a commit. This file-level size is also cheap to compute, as only the log-entries need to be examined and 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 changes in these commits?

In an attempt to get a clearer picture, we further process the files to the granularity of line and hunks 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 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 comment with the
license information. Therefore, hunk 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 line 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. Therefore, in our investigation we consider all the three kinds of size measures for a commit and leave the issue open to an empirical investigation. We use the following measures:

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

**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.
Figure 2. An example boxplot.

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

Table 1. gcc commits categorized by size for each of the three characteristics. The duration of the commits was over eight years.

<table>
<thead>
<tr>
<th>Quartiles</th>
<th>Number of Files</th>
<th>Number of Lines</th>
<th>Number of Hunks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Q0 (Min)</strong></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Q1</td>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Q2</td>
<td>2</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>Q3</td>
<td>4</td>
<td>46</td>
<td>8</td>
</tr>
<tr>
<td><strong>Q4 (Max)</strong></td>
<td>4908</td>
<td>203359</td>
<td>8067</td>
</tr>
</tbody>
</table>

**IQR**

- **Boxplot:**
  - **Range**
    - x-small: 1 - 1
    - small: 2 - 4
    - medium: 5 - 7
    - large: 8 - 10
    - x-large: 11 - 4908
  - **Freq**
    - x-small: 4580
    - small: 3710
    - medium: 6980
    - large: 2201
    - x-large: 3675
  - **Ratio**
    - x-small: 8.4%
    - small: 68.0%
    - medium: 12.8%
    - large: 4.0%
    - x-large: 6.7%
  - **Range**
    - x-small: 0 - 5
    - small: 6 - 46
    - medium: 47 - 106
    - large: 107 - 166
    - x-large: 167 - 203359
  - **Freq**
    - x-small: 10844
    - small: 30162
    - medium: 6072
    - large: 2335
    - x-large: 5123
  - **Ratio**
    - x-small: 19.9%
    - small: 55.3%
    - medium: 11.1%
    - large: 4.3%
    - x-large: 9.4%
  - **Range**
    - x-small: 0 - 1
    - small: 2 - 8
    - medium: 9 - 17
    - large: 18 - 26
    - x-large: 27 - 8067
  - **Freq**
    - x-small: 5625
    - small: 3556
    - medium: 5862
    - large: 2224
    - x-large: 5263
  - **Ratio**
    - x-small: 10.3%
    - small: 65.2%
    - medium: 10.7%
    - large: 4.1%
    - x-large: 9.7%
3.3 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 history, and then use a descriptive statistics method to classifying 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 third quartile holds 75% of the data. Therefore, what has 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 $Q_3 (Q_3 - Q_2 + 1.5 \times IQR)$,

6. Mild outliers upward $Q_3 (Q_3 + 1.5 \times IQR - Q_2 + 3 \times IQR)$ 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 [Johnson, Wichern 1998] that is a histogram-like method for displaying data (see Figure 1). The distribution of our data is limited to five regions; we never had a case that has commits 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, number of lines, and 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. The data and quartiles for gcc are given in Table 1 and the corresponding histogram-like and boxplot for files, lines and hunks in Figure 3, Figure 4, Figure 5, Figure 6, Figure 7, and Figure 8 for gcc 54,536 commits over 8 years.

Figure 3 shows a Histogram for the number of commits distributed over number of files that were changed for gcc 54,536 commits over 8 years. Committers group files in one commit wither it has been an added, modified or
deleted. Our unit of change, the commit, has different number of files each commit in gcc that ranges from 1 – 4908 files committed together, but what is happening on a most daily bases of committing practices is 2 (44.2% commit), 3 (12.8% commit), or 4 (11.1% commit) files where 68% of commits has 2 – 4 files. The corresponding Boxplot Figure 4 shows that median equals to $Q_1$ (2 files), which shows that commits are condensed in the left side of the small size region. The mean is to the right of the median that falls among medium size commits, the distribution skewed to the right and most commits small, but there are a few exceptional large ones fall into the large and the extra-large regions. Those exceptional values will affect the mean and pull it to the right. The box plot will look as if the box shifted to the left so that the right tail will be longer, and the median will be closer to the left line of the box in the box plot. Moreover, that applies too to distributions of lines and hunks.

![Histogram](image)

**Figure 3.** Histogram for the number of commits distributed over number of files that were changed for gcc (54,536 commits over 8 years).
Figure 4. Boxplot for the number of files per commit for gcc.

In each commit there are lines that replace old lines, whether they are very new lines or modified lines. Table 1 and Figure 5 says; that more than half of the commits are small and commits in this region have lines ranges between 6 – 45 lines. The boxplot Figure 6 shows that the mean is far to the right of the median falls among large commits. The distribution skewed to the right and most values are small reach to 55.3%, and the very next region here is the extra-small with 19.9% as a 0 – 5 lines while there are few large commits fall into the large and the extra-large commits. Some big numbers of lines in commits pulled the mean to the far right.
Figure 5. Histogram for the number of commits distributed over number of lines that were changed for gcc (54,536 commits over 8 years).

Table 1 and Figure 7 show the distribution of commits over number of hunks in the boxplot module. The boxplot Figure 8 shows a distribution skewed also to the right and most values are small with 65.2%. The mean is on the right of the median and falls among the large commits.

Figure 6. Boxplot for the number of lines per commit for gcc.

As mentioned before, a hunk is an area of change in a file. UNIX diff command between two conductive commits that change same file will show blocks of lines changed together each called a hunk.
Figure 7. Histogram for the number of commits distributed over number of hunks that were changed for *gcc* (54,536 commits over 8 years).

<table>
<thead>
<tr>
<th>Category</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 1 hunk(s)</td>
<td>10.3%</td>
</tr>
<tr>
<td>2 - 8 hunk(s)</td>
<td>65.2%</td>
</tr>
<tr>
<td>9 - 17 hunk(s)</td>
<td>10.7%</td>
</tr>
<tr>
<td>18 - 26 hunk(s)</td>
<td>4.1%</td>
</tr>
<tr>
<td>27 - 8067 hunk(s)</td>
<td>9.7%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>extra-small</td>
</tr>
<tr>
<td>small</td>
</tr>
<tr>
<td>medium</td>
</tr>
<tr>
<td>large</td>
</tr>
<tr>
<td>extra-large</td>
</tr>
</tbody>
</table>

Figure 8. Boxplot for the number of hunks per commit for *gcc*.

Next chapter we present the results of our categorization for all the systems we studied.
CHAPTER 4

Typical Size of Commits

Here, we want to determine in what size range most of the commits fall, i.e., what is the size of a typical commit. Let us again consider the histograms given in Figure 3, Figure 5, and Figure 7 for the file-size, line-size, and hunk-size measures along with the raw data in Table 1. This data is for the GNU gcc 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\(^1\) 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

\(^1\) http://www.gnu.org/copyleft/gpl.html
correlated to one other. That is, if it a commit has a small number of files, does that also indicate that a small number of lines were changed?

### 4.1 Correlation between Characteristics

We now address two related questions concerning gcc and later generalize this to all nine projects. First we see if there is a correlation between any of the two different 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 need to investigate the linear correlation coefficient and the coefficient of determination [Johnson, Wichern 1998].

The linear correlation coefficient (denoted by $r$) [Johnson, Wichern 1998] 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:

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

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 are 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 $\pm 1$ 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.

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 9. Histogram of 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 $r^2$, it is nice to have a picture about the percentages of commonality between the three characteristics over the five categories. Figure 9 is a detailed cross intersection over the three characteristics distributions. 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 around 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 were 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 (extra-small as 0, … , extra-large as 4 for gcc).

- By considering $x$ as number of files and $y$ can be the number of hunks. For a gcc commit with ID# 31155 is a medium size commit as number of files so $x = 2$ and a small size commit under the number of hunks metric so $y = 1$, 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.

We only see a strong correlation between number of hunks $\times$ 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 2. gcc $r$ and $r^2$ values for all size metrics combinations.

<table>
<thead>
<tr>
<th>gcc</th>
<th>$files \times lines$</th>
<th>$files \times hunks$</th>
<th>$hunks \times lines$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$</td>
<td>0.5</td>
<td>0.7</td>
<td>0.8</td>
</tr>
<tr>
<td>$r^2$</td>
<td>0.3</td>
<td>0.4</td>
<td>0.6</td>
</tr>
</tbody>
</table>

This result is hardly surprising. Developers can commit one file with hundreds of lines changed and also commit hundreds of files with one line changed for each file. Thus, there should be little correlation between the 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.
- For 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 10 gives the correlation coefficient for each of the two characteristics projected over each size category. 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 the may very well 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 10. Histogram the correlation coefficients $r$ between each two metrics over each size category.

Table 3. Nine open source systems used in study.

<table>
<thead>
<tr>
<th>System</th>
<th>Source</th>
<th>Duration</th>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>hgcc.gnu.org</td>
<td>8 years</td>
<td>54,536</td>
</tr>
<tr>
<td>Collab</td>
<td><a href="http://www.collab.net">www.collab.net</a></td>
<td>5.7 years</td>
<td>20288</td>
</tr>
<tr>
<td>JEdit</td>
<td><a href="http://www.jedit.org">www.jedit.org</a></td>
<td>6.1 years</td>
<td>2467</td>
</tr>
<tr>
<td>Ruby</td>
<td><a href="http://www.ruby-lang.org/en">www.ruby-lang.org/en</a></td>
<td>9 years</td>
<td>10667</td>
</tr>
<tr>
<td>LinuxBoss</td>
<td>bosslinux.in</td>
<td>7.9 years</td>
<td>3023</td>
</tr>
<tr>
<td>Phpmyadmin</td>
<td><a href="http://www.phpmyadmin.net/home_page/index.php">www.phpmyadmin.net/home_page/index.php</a></td>
<td>6.7 years</td>
<td>6028</td>
</tr>
<tr>
<td>MySql Admin</td>
<td><a href="http://www.mysql.com/products/tools/administrator">www.mysql.com/products/tools/administrator</a></td>
<td>1.3 years</td>
<td>384</td>
</tr>
<tr>
<td>Python</td>
<td><a href="http://www.python.org">www.python.org</a></td>
<td>6 years</td>
<td>20420</td>
</tr>
<tr>
<td>Debian-installer</td>
<td><a href="http://www.debian.org/devel/debian-installer">www.debian.org/devel/debian-installer</a></td>
<td>7.5 years</td>
<td>40425</td>
</tr>
</tbody>
</table>
As in the case study gcc, we apply our procedure to the remaining eight open source projects (open source systems ranging across different application domains, programming languages and sizes (see Table 3)). The correlation over regions is given in Figure 11. 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. 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.
Figure 11. Linear correlation $r$ over regions to nine projects.
CHAPTER 5

Vocabulary versus Commit Size

5.1 Approach

Given the size categorization described previously, we now try to understand what types 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 a Lovins stemmer algorithm [Lovins 1968]. 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 always 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, as in apriori [Agrawal, Srikant 1995]. So we will have a ranked list of frequent terms for each project. Then we cross join those nine lists and take the top most 50 frequent terms.
5.2 Results

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.

<table>
<thead>
<tr>
<th>Term</th>
<th>Average Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>fix</td>
<td>36.05%</td>
</tr>
<tr>
<td>add</td>
<td>18.22%</td>
</tr>
<tr>
<td>test</td>
<td>13.96%</td>
</tr>
<tr>
<td>file</td>
<td>11.85%</td>
</tr>
<tr>
<td>new</td>
<td>11.54%</td>
</tr>
<tr>
<td>support</td>
<td>11.35%</td>
</tr>
<tr>
<td>chang</td>
<td>11.14%</td>
</tr>
<tr>
<td>bug</td>
<td>10.88%</td>
</tr>
<tr>
<td>patch</td>
<td>10.03%</td>
</tr>
<tr>
<td>code</td>
<td>9.41%</td>
</tr>
<tr>
<td>remov</td>
<td>9.38%</td>
</tr>
<tr>
<td>set</td>
<td>8.45%</td>
</tr>
<tr>
<td>work</td>
<td>8.24%</td>
</tr>
<tr>
<td>updat</td>
<td>7.92%</td>
</tr>
<tr>
<td>get</td>
<td>6.75%</td>
</tr>
<tr>
<td>error</td>
<td>5.93%</td>
</tr>
<tr>
<td>build</td>
<td>5.73%</td>
</tr>
<tr>
<td>function</td>
<td>5.57%</td>
</tr>
<tr>
<td>update</td>
<td>5.18%</td>
</tr>
<tr>
<td>typo</td>
<td>4.70%</td>
</tr>
<tr>
<td>call</td>
<td>4.66%</td>
</tr>
<tr>
<td>messag</td>
<td>4.65%</td>
</tr>
<tr>
<td>includ</td>
<td>4.52%</td>
</tr>
<tr>
<td>path</td>
<td>4.52%</td>
</tr>
</tbody>
</table>
We take this list and identify frequent term combinations (as in mining frequent itemsets in apriori) 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 list 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.

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 \{remove, test\} (in that order) are most frequent. The set \{add, bug\} is very frequent in extra-small line changes. For the changes to hunks \{add, bug\}, \{file, fix\}, \{fix, use\}, and \{remove, 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.
Table 5. Frequent term set distribution, for all nine systems, over the number of files changed per commit for each sized category.

<table>
<thead>
<tr>
<th>extra-small</th>
<th>small</th>
<th>medium</th>
<th>large</th>
<th>extra-Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terms</td>
<td>Support</td>
<td>Terms</td>
<td>Support</td>
<td>Terms</td>
</tr>
<tr>
<td>fix, typo</td>
<td>1.18%</td>
<td>add, bug</td>
<td>3.51%</td>
<td>file, fix</td>
</tr>
<tr>
<td>file, fix</td>
<td>0.29%</td>
<td>file, fix</td>
<td>3.06%</td>
<td>add, bug</td>
</tr>
<tr>
<td>fix, test</td>
<td>0.27%</td>
<td>fix, typo</td>
<td>1.99%</td>
<td>remove, test</td>
</tr>
<tr>
<td>remove, test</td>
<td>0.26%</td>
<td>remove, test</td>
<td>1.74%</td>
<td>fix, use</td>
</tr>
<tr>
<td>add, bug</td>
<td>0.25%</td>
<td>fix, test</td>
<td>1.60%</td>
<td>file, remove</td>
</tr>
<tr>
<td>fix, use</td>
<td>0.22%</td>
<td>file, remove</td>
<td>1.34%</td>
<td>fix, set</td>
</tr>
<tr>
<td>bug, fix, work</td>
<td>0.20%</td>
<td>fix, use</td>
<td>1.18%</td>
<td>fix, test</td>
</tr>
<tr>
<td>file, remove</td>
<td>0.14%</td>
<td>fix, set</td>
<td>1.10%</td>
<td>file, update</td>
</tr>
<tr>
<td>doc, update</td>
<td>0.10%</td>
<td>file, update</td>
<td>0.83%</td>
<td>bug, fix, work</td>
</tr>
<tr>
<td>fix, path</td>
<td>0.10%</td>
<td>doc, update</td>
<td>0.80%</td>
<td>doc, update</td>
</tr>
<tr>
<td>fix, set</td>
<td>0.09%</td>
<td>error, message</td>
<td>0.77%</td>
<td>fix, typo</td>
</tr>
<tr>
<td>file, update</td>
<td>0.07%</td>
<td>bug, fix</td>
<td>0.76%</td>
<td>error, message</td>
</tr>
<tr>
<td>error, message</td>
<td>0.05%</td>
<td>bug, fix, work</td>
<td>0.75%</td>
<td>fix, path</td>
</tr>
<tr>
<td>bug, fix</td>
<td>0.04%</td>
<td>fix, path</td>
<td>0.63%</td>
<td>bug, fix</td>
</tr>
<tr>
<td>bug, fix, use</td>
<td>0.01%</td>
<td>bug, fix, use</td>
<td>0.26%</td>
<td>bug, fix, use</td>
</tr>
</tbody>
</table>
Table 6. Frequent term set distribution, for all nine systems, over the number of lines changed per commit for each sized category.

<table>
<thead>
<tr>
<th>Terms</th>
<th>extra-small</th>
<th>Terms</th>
<th>Support</th>
<th>small</th>
<th>Terms</th>
<th>Support</th>
<th>medium</th>
<th>Terms</th>
<th>Support</th>
<th>large</th>
<th>Terms</th>
<th>Support</th>
<th>extra-Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>add, bug</td>
<td>8.19%</td>
<td>file, fix</td>
<td>2.96%</td>
<td>file, fix</td>
<td>6.05%</td>
<td>file, fix</td>
<td>6.28%</td>
<td>file, fix</td>
<td>8.99%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>file, fix</td>
<td>2.81%</td>
<td>fix, typo</td>
<td>2.14%</td>
<td>remove, test</td>
<td>5.33%</td>
<td>add, bug</td>
<td>4.69%</td>
<td>add, bug</td>
<td>6.17%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fix, typo</td>
<td>2.04%</td>
<td>add, bug</td>
<td>1.87%</td>
<td>add, bug</td>
<td>3.92%</td>
<td>fix, use</td>
<td>3.99%</td>
<td>file, update</td>
<td>5.62%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fix, use</td>
<td>1.87%</td>
<td>remove, test</td>
<td>1.87%</td>
<td>fix, set</td>
<td>3.17%</td>
<td>file, update</td>
<td>3.59%</td>
<td>fix, use</td>
<td>5.23%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bug, fix, work</td>
<td>0.86%</td>
<td>fix, test</td>
<td>1.71%</td>
<td>fix, test</td>
<td>2.71%</td>
<td>bug, fix, work</td>
<td>2.80%</td>
<td>file, remove</td>
<td>4.04%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fix, test</td>
<td>0.72%</td>
<td>file, remove</td>
<td>1.52%</td>
<td>file, remove</td>
<td>2.62%</td>
<td>remove, test</td>
<td>2.77%</td>
<td>fix, set</td>
<td>3.75%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove, test</td>
<td>0.54%</td>
<td>fix, set</td>
<td>1.06%</td>
<td>fix, use</td>
<td>2.62%</td>
<td>file, remove</td>
<td>2.76%</td>
<td>remove, test</td>
<td>3.62%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fix, set</td>
<td>0.47%</td>
<td>fix, use</td>
<td>1.04%</td>
<td>file, update</td>
<td>2.30%</td>
<td>doc, update</td>
<td>2.69%</td>
<td>bug, fix, work</td>
<td>3.59%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>file, update</td>
<td>0.43%</td>
<td>bug, fix</td>
<td>0.82%</td>
<td>bug, fix, work</td>
<td>2.22%</td>
<td>fix, test</td>
<td>2.56%</td>
<td>doc, update</td>
<td>3.34%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc, update</td>
<td>0.41%</td>
<td>error, message</td>
<td>0.82%</td>
<td>doc, update</td>
<td>1.89%</td>
<td>fix, set</td>
<td>2.54%</td>
<td>fix, test</td>
<td>3.20%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>file, remove</td>
<td>0.29%</td>
<td>file, update</td>
<td>0.77%</td>
<td>error, message</td>
<td>1.22%</td>
<td>error, message</td>
<td>1.66%</td>
<td>bug, fix</td>
<td>2.39%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fix, path</td>
<td>0.22%</td>
<td>doc, update</td>
<td>0.77%</td>
<td>fix, typo</td>
<td>1.09%</td>
<td>bug, fix</td>
<td>1.56%</td>
<td>error, message</td>
<td>1.54%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bug, fix</td>
<td>0.20%</td>
<td>fix, path</td>
<td>0.73%</td>
<td>bug, fix, use</td>
<td>0.87%</td>
<td>fix, typo</td>
<td>0.96%</td>
<td>bug, fix, use</td>
<td>1.42%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>error, message</td>
<td>0.14%</td>
<td>bug, fix, work</td>
<td>0.70%</td>
<td>fix, path</td>
<td>0.85%</td>
<td>fix, path</td>
<td>0.80%</td>
<td>fix, path</td>
<td>1.23%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bug, fix, use</td>
<td>0.04%</td>
<td>bug, fix, use</td>
<td>0.27%</td>
<td>bug, fix</td>
<td>0.81%</td>
<td>bug, fix, use</td>
<td>0.50%</td>
<td>fix, typo</td>
<td>1.02%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 7. Frequent term set distribution, for all nine systems, over the number of hunks changed per commit for each sized category.

<table>
<thead>
<tr>
<th>Terms</th>
<th>Support</th>
<th>Terms</th>
<th>Support</th>
<th>Terms</th>
<th>Support</th>
<th>Terms</th>
<th>Support</th>
<th>Terms</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>extra-small</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add, bug</td>
<td>8.77%</td>
<td>file, fix</td>
<td>2.97%</td>
<td>file, fix</td>
<td>5.04%</td>
<td>file, fix</td>
<td>7.06%</td>
<td>file, fix</td>
<td>6.63%</td>
</tr>
<tr>
<td>file, fix</td>
<td>3.12%</td>
<td>fix, typo</td>
<td>2.21%</td>
<td>add, bug</td>
<td>3.79%</td>
<td>remove, test</td>
<td>5.53%</td>
<td>fix, use</td>
<td>6.44%</td>
</tr>
<tr>
<td>fix, use</td>
<td>2.15%</td>
<td>add, bug</td>
<td>1.79%</td>
<td>remove, test</td>
<td>3.20%</td>
<td>add, bug</td>
<td>4.72%</td>
<td>file, update</td>
<td>6.44%</td>
</tr>
<tr>
<td>fix, typo</td>
<td>1.39%</td>
<td>remove, test</td>
<td>1.71%</td>
<td>fix, use</td>
<td>3.00%</td>
<td>fix, set</td>
<td>4.48%</td>
<td>add, bug</td>
<td>5.96%</td>
</tr>
<tr>
<td>fix, test</td>
<td>0.75%</td>
<td>fix, test</td>
<td>1.67%</td>
<td>fix, test</td>
<td>2.48%</td>
<td>fix, use</td>
<td>4.25%</td>
<td>fix, set</td>
<td>4.21%</td>
</tr>
<tr>
<td>bug, fix, work</td>
<td>0.65%</td>
<td>file, remove</td>
<td>1.39%</td>
<td>file, remove</td>
<td>2.28%</td>
<td>bug, fix, work</td>
<td>4.03%</td>
<td>file, remove</td>
<td>3.96%</td>
</tr>
<tr>
<td>remove, test</td>
<td>0.64%</td>
<td>fix, use</td>
<td>1.00%</td>
<td>file, update</td>
<td>2.20%</td>
<td>file, remove</td>
<td>3.90%</td>
<td>bug, fix, work</td>
<td>3.50%</td>
</tr>
<tr>
<td>fix, set</td>
<td>0.56%</td>
<td>fix, set</td>
<td>1.16%</td>
<td>bug, fix, work</td>
<td>1.90%</td>
<td>file, update</td>
<td>3.25%</td>
<td>bug, fix</td>
<td>3.49%</td>
</tr>
<tr>
<td>bug, fix</td>
<td>0.42%</td>
<td>bug, fix</td>
<td>1.03%</td>
<td>doc, update</td>
<td>1.78%</td>
<td>doc, update</td>
<td>3.19%</td>
<td>doc, update</td>
<td>3.29%</td>
</tr>
<tr>
<td>file, remove</td>
<td>0.40%</td>
<td>doc, update</td>
<td>0.78%</td>
<td>fix, set</td>
<td>1.70%</td>
<td>fix, test</td>
<td>2.99%</td>
<td>fix, test</td>
<td>3.29%</td>
</tr>
<tr>
<td>file, update</td>
<td>0.37%</td>
<td>file, update</td>
<td>0.74%</td>
<td>bug, fix</td>
<td>1.53%</td>
<td>bug, fix</td>
<td>1.87%</td>
<td>remove, test</td>
<td>3.28%</td>
</tr>
<tr>
<td>doc, update</td>
<td>0.37%</td>
<td>error, message</td>
<td>0.82%</td>
<td>fix, typo</td>
<td>1.42%</td>
<td>error, message</td>
<td>1.61%</td>
<td>bug, fix, use</td>
<td>2.08%</td>
</tr>
<tr>
<td>fix, path</td>
<td>0.22%</td>
<td>fix, path</td>
<td>0.68%</td>
<td>error, message</td>
<td>1.08%</td>
<td>fix, typo</td>
<td>1.14%</td>
<td>error, message</td>
<td>1.72%</td>
</tr>
<tr>
<td>error, message</td>
<td>0.16%</td>
<td>bug, fix, work</td>
<td>0.66%</td>
<td>fix, path</td>
<td>0.90%</td>
<td>bug, fix, use</td>
<td>0.80%</td>
<td>fix, path</td>
<td>1.43%</td>
</tr>
<tr>
<td>bug, fix, use</td>
<td>0.08%</td>
<td>bug, fix, use</td>
<td>0.26%</td>
<td>bug, fix, use</td>
<td>0.78%</td>
<td>fix, path</td>
<td>0.75%</td>
<td>fix, typo</td>
<td>1.33%</td>
</tr>
</tbody>
</table>
CHAPTER 6

Conclusions and Future Research

This work 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. Additionally, the commit messages are examined along with the characteristics.

6.1 Main Results

We presented a study that investigated nine open source systems ranging across different application domains, programming languages, sizes. The goal of our study was to infer the characteristics of a typical commit from years of historical information. The data that was obtained indicate 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 would provide an insight into the purposes of the changes.

6.2 Future Research Directions

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?
REFERENCES


