Mining Change Logs and Release Notes to Understand Software Maintenance and Evolution

Liguo Yu

Computer Science and Informatics
Indiana University South Bend
South Bend, IN 46634, USA
Email: ligyu@iusb.edu

Abstract: Software change logs and release notes are documents released together with new versions of a software product. They contain the description of the changes made to the previous version and the new features introduced in the new version. In this paper, we present a keyword-based approach to mining and analyzing non-source code documents and define a mathematical framework to represent the data. This approach is applied in the study of the change logs of Linux and the release notes of FreeBSD. The results show that the software maintenance process and evolution process share some common properties and the keyword-based text mining technique could be used as a systematic method to study software maintenance and evolution.

Keywords: Software maintenance, software evolution, change logs, release notes, text mining, Linux, FreeBSD

1 Introduction

After a software product is released, it has to change to remain its long term usefulness and compatibility. This is called the maintenance and evolution process. Usually, software maintenance is referred as the process that removes bugs and fixes errors; software evolution is referred as the process that updates a product to reflect the changes of the requirements and the environments [1]. Software maintenance and evolution is inevitable, because (1) it is practically impossible to produce an error-free software product; and (2) the users' requirements (both functional requirements and non-functional requirements) and the software working environments vary with time. Therefore, the software maintenance and evolution process is characterized by the continuous error-fixing and feature incorporating/updating activities [2].

Change Logs and release notes are documents that are distributed together with software products when they are made public for use. The change logs and release notes for continually evolved software products normally contain a summary of changes, enhancements and bugs fixed in this particular version [3]. In closed-source software, the proprietor has well organized internal documents about software changes; the change logs and release notes are intent to provide minimal information to give the readers sufficient information about the software upgrade. For open-source software, change logs and releases notes are the only available formal announcements for software upgrade (Although source code, CVS, and bug tracking systems can be used to extract software upgrade information, they are not considered as a formal announcement of software upgrade) [4]. With the growth of the complexity of software products, open-source developers begin to enforce detailed change logs and release notes. Especially recently, the open-source community has witnessed the growth of the information recorded in open-source change logs and release notes. This information is amenable to modern data mining techniques.

Text mining (also called text data mining) is the process of deriving high quality information from text, in which *high quality* refers to some combination of relevance, novelty, and interestingness [5]. Despite its success in many areas, text mining method is rarely used in software engineering [6]. In this paper, we present a keyword-based approach to mining change logs and release notes to extract

useful software maintenance and evolution information. For each change log and release note, a vocabulary is generated and keywords are identified and associated with bug-fixing activities and feature- incorporating/updating activities in the development of the new version of the software product.

The remainder of the paper is organized as follows. Section 2 describes the keyword-based text mining approach. Section 3 presents our case studies on FreeBSD and Linux. The conclusions and future work are in Section 4.

2 Keyword-Based Text Mining Approach

2.1 Mapping activities to keywords

Software maintenance and evolution include error fixing, feature incorporation, feature upgrading, and other activities, such as code restructuring. In this paper, we borrow the concept of feature from previous work and consider a feature as a requirement of a program that a user can exercise to produce an observable behavior [7] [8] [9]. We consider two types of activities: error-fixing, which belongs to the maintenance activity, and feature-incorporating/upgrading, which is identified as evolution activity. Other activities are considered irrelevant and are ignored in this study. The research method presented here is based on two types of observations.

- **Observation 1:** The error-fixing activities and the feature-incorporating/updating activities of a new version of a software product are recorded in the change logs and/or release notes.
- **Observation 2:** The information contained in the text of change logs and release notes of a software product is represented with some keywords.

In the two types of observations, Observation 2 has been widely accepted in the text mining community, i.e., keyword mining has become a standard text mining technique [10]; Observation 1 is supported by the evidences reported by previous studies. For example, in [6], Baysal and Malton found that the non-source code documents contain similar amount of contents of source code changes in software maintenance and evolution, which indicates that non-source code documents, such as email archives, release notes, and change logs, might accurately record the maintenance and evolution activity of a software product. In [3], Chen et al. found that the content in change logs is relatively accurate in representing the software changes. Based on these reported observations, we make the following two assumptions.

First, we assume that the error-fixing activities and the feature-incorporating/updating activities of a new version of a software product are represented with some keywords in change logs and/or release notes. The keywords that are related with software maintenance (error-fixing) activities are called *maintenance keywords*; the keywords that are related with software evolution (feature-incorporating/updating) activities are called *evolution keywords*. Because one activity can be mapped to more than one keyword and one keyword can be mapped to more than one activity, the mapping between activities to keywords has a many to many relation. To simply the problem, we combine the same types of activities and the same types of keywords and assign them one to one relation, as depicted in Figure 1. Second, we assume that the number of keywords is proportional to the number of activities. Accordingly, in the remaining of this paper, we use the number of keywords to represent the number of activities.

Figure 1. The mapping of software maintenance and evolution activities to keywords in change logs and release notes.

2.2 Data extraction

The data extraction contains three phases, non-keyword preparation, keyword extraction, and keyword separation. In Phase 1 (non-keyword preparation), we scan the first and the last available change log (release note), the words that are common to these two documents constitute the *non-keyword dictionary*. Because these words appear in two change logs (release notes) that have the longest time span, they are unlikely relate to the same software maintenance or evolution activity. Most likely, these common words are documents related instead of activity related.

In Phase 2, we extract maintenance and evolution keywords from change logs or release notes. For each change log or release note, the operation contains four steps.

1. The text file is scanned and the capital letters are transferred to lower case letters;
2. Each word is recorded and common English words that are articles, numbers, prepositions, etc are removed;
3. The words are scanned again and non-keywords (words from *non-keyword dictionary*) are removed;
4. The words are sorted and duplicates are removed.

In Phase 3, maintenance keywords and evolution keywords are separated. In this paper, we identify two types of maintenance keyword, English word and non-English word. The English words include *fix*, *bug*, *error*, *failure*, *correct*, etc, which are related with the maintenance activity. The non-English words are those words that can not be found in an English dictionary but are related with source code classes, functions, variables, etc., such as *cpu_save*, *fs_nls*, and *xtont*. These words are assumed to be related with specific maintenance activity and are accordingly considered as maintenance keywords. The remaining English words are considered as evolution keywords, such as *security*, *improve*, *increase*, and so on.

2.3 Data representation

The maintenance keywords and evolution keywords are analyzed based on the following definitions, some of which are inspired by work presented in [6].

- **Definition 1** The maintenance and evolution of a product is represented as a set of versions of product $P = \{v_1, v_2, \dots, v_n\}$, where v_i ($1 \leq i \leq n$) is i th release of product P , $n=|P|$ is the number of release of the product.
- **Definition 2** The release notes (change logs) of a software product, denoted by $R = \{r_1, r_2, \dots, r_n\}$, is a set of documents distributed with the software product P , where r_i ($1 \leq i \leq n$) is the release note (change log) of version v_i .
- **Definition 3a** The maintenance keywords of release note (change log) r_i ($1 \leq i \leq n$) is represented as a set of words M_i ($1 \leq i \leq n$).
- **Definition 3b** The evolution keywords of release note (change log) r_i ($1 \leq i \leq n$) is represented as a set of words E_i ($1 \leq i \leq n$).

- **Definition 4a** The new maintenance keywords introduced in release note (change log) r_i ($1 \leq i \leq n$) are represented as $N(m)_i = M_i - \bigcup_{k=1}^{i-1} M_k$. The number of new maintenance keywords in release note (change log) r_i is $|N(m)_i|$.
- **Definition 4b** The new evolution keywords in release note (change log) r_i ($1 \leq i \leq n$) are represented as $N(e)_i = E_i - \bigcup_{k=1}^{i-1} E_k$. The number of new evolution keywords in release note (change log) r_i is $|N(e)_i|$.
- **Definition 5a** The recurring maintenance keywords in release note (change log) r_i ($1 \leq i \leq n$) are represented as $O(m)_i = M_i \cap \bigcup_{k=1}^{i-1} M_k$. The number of recurring maintenance keywords in release note (change log) r_i is $|O(m)_i|$.
- **Definition 5b** The recurring evolution keywords in release note (change log) r_i ($1 \leq i \leq n$) are represented as $O(e)_i = E_i \cap \bigcup_{k=1}^{i-1} E_k$. The number of recurring evolution keywords in release note (change log) r_i is $|O(e)_i|$.
- **Definition 6a** For a maintenance keyword a , the number of sets of M_i ($1 \leq i \leq n$) that contains a is called the number of appearances of a .
- **Definition 6b** For an evolution keyword β , the number of sets of E_i ($1 \leq i \leq n$) that contains β is called the number of appearances of β .
- **Definition 7a** For a maintenance keyword a , if $a \in M_i$ and $a \in M_j$ and $a \notin \bigcup_{k=1}^{i-1} M_k$ and $a \notin \bigcup_{k=j+1}^n M_k$ ($1 \leq i \leq j \leq n$), $D_a = j - i + 1$ is called the version distance of a .
- **Definition 7b** For an evolution keyword β , if $\beta \in E_i$ and $\beta \in E_j$ and $\beta \notin \bigcup_{k=1}^{i-1} E_k$ and $\beta \notin \bigcup_{k=j+1}^n E_k$ ($1 \leq i \leq j \leq n$), $D_\beta = j - i + 1$ is called version distance of β .

Figure 2 shows the relationship of version number, release note/change log number, and keyword sets, in which, the keywords in keyword set M represent maintenance activities and keywords in keyword set E represent evolution activities.

Figure 2. The relationship of version number, release note/change log number, and keyword sets.

Figure 3 illustrates definitions 4 and 5. The new maintenance keywords identified in release note (change log) r_i represent the new maintenance activities carried out in version v_i ; the number of new maintenance keywords $|N(m)_i|$ represents the number of new maintenance activities carried out in version v_i . The new evolution keywords identified in release note (change log) r_i represent the new

evolution activities carried out in version v_i ; the number of new evolution keywords $|N(e)_i|$ represents the number of new evolution activities carried out in version v_i .

The recurring maintenance keywords identified in release note (change log) r_i represent the recurring maintenance activities carried out in version v_i ; the number of recurring maintenance keywords $|O(m)_i|$ represents the number of recurring maintenance activities carried out in version v_i . The recurring evolution keywords identified in release note (change log) r_i represent the recurring evolution activities carried out in version v_i ; the number of recurring evolution keywords $|O(e)_i|$ represents the number of recurring evolution activities carried out in version v_i .

Figure 4 illustrates the count of the appearance of a maintenance keyword and an evolution keyword. The appearance of a maintenance keyword a in M_i indicates that the maintenance activity mapped with keyword a is performed in version v_i ; the appearance of an evolution keyword β in E_i indicates that the evolution activity mapped with keyword β is performed in version v_i .

Figure 5 illustrates the definitions of version distances. For a maintenance activity mapped with a maintenance keyword a , version distance D_a represents the duration of the maintenance activity. For an evolution activity mapped with an evolution keyword β , version distance D_β represents the duration of the evolution activity.

Figure 3. The illustrations of Definitions 4 and 5.

Figure 4. The number of appearances of (a): maintenance keyword α ; and (b) evolution keyword β .

Figure 5. The version distance of (a): the maintenance keyword α ; and (b) the evolution keyword β .

3 Case Studies of FreeBSD and Linux

FreeBSD [11] and Linux [12] are two open-source operating systems. The first version of FreeBSD was released in 1993 and the first version of Linux was released in 1991. After the first releases, both FreeBSD and Linux have been updated and many versions have been released. Compared to their initial versions, the size of the source code of current versions of both FreeBSD and Linux have increased tremendously. For example, From version 1.0.0 to 2.6.22, the kernel size of Linux has increased about 40-50 times. During their maintenance and evolution processes, many bugs have been removed and many features have been incorporated and improved. FreeBSD and Linux follow different documentation regulations: FreeBSD has release notes describing new features introduced in the new version; Linux has change logs describing the maintenance activities. Accordingly, in the case studies presented in this paper, we use release notes of FreeBSD to study its evolution process and we use change logs of Linux to study its maintenance process.

Version 1.1 of FreeBSD was released in May 1994 and version 6.2 of FreeBSD was released in January 2007. We study all the formal releases of FreeBSD between (including) version 1.1 and version 6.2, which contain 43 release notes. Linux begins to enforce change logs since version 2.4.1, which was released in January 2001 and the current version 2.6.22 was released in July 2007. We study all the releases of Linux between (including) version 2.4.1 and version 2.6.5, which contain 115 change logs. Figure 6 illustrates the versions we studied.

Figure 6. The versions of FreeBSD and Linux studied in this research.

Figure 7a shows the number of total evolution keywords and the number of new evolution keywords in each release note of FreeBSD. Figure 7b shows the number of total maintenance keywords and the number of new maintenance keywords in each change log Linux. Although these numbers varies with versions, in generally, they follow an increasing trend.

Figure 7. The number of (a) evolution keywords in the release notes of FreeBSD; (b) maintenance keywords in the change logs of Linux.

Each evolution activity can be considered as either introducing new features or updating existing features, which are represented with new evolution keywords and repeating (recurring) evolution keywords respectively; each maintenance activity can be considered as either working on a new maintenance issue or working on a pre-existing maintenance issue, which are represented with new maintenance keywords and repeating (recurring) maintenance keywords respectively. Figure 8a shows the scatter plot of the number of new evolution keywords and the number of repeating evolution keywords in the release notes of FreeBSD. Figure 8b shows the scatter plot of the number of new maintenance keywords and the number of repeating maintenance keywords in the change log of Linux.

Figure 8. The relations between the number of new keywords and the number of repeating keywords: (a) evolution keywords in the release notes of FreeBSD; and (b) maintenance keywords in the change logs of Linux.

To study the relation between the number of new keywords and the number repeating keywords, we tested the Spearman's rank [13] correlation between them. The results are in Table 1. A Spearman's rank correlation could have a value in the range of $[-1, 1]$. A value of -1 indicates a perfect negative relationship, i.e., new keywords and repeating keywords are clearly separate in different versions; a value of 1 indicates a perfect positive relationship, i.e., new keywords is strongly correlated with repeated keywords; a value of 0 indicates of no relationship between two variables. In both two tests, the correlation is positive and the significance is at or above the 0.05 level. Therefore,

we conclude, (1) in each revision of FreeBSD, the number of new evolution activities (represented with new evolution keywords) is linearly correlated with the number of repeating evolution activities (represented with repeating evolution keywords); (2) in each revision of Linux, the number of new maintenance activities (represented with new maintenance keywords) is linearly correlated with the number of repeating maintenance activities (represented with repeating maintenance keywords).

Table 1. The correlation between the number of new keywords and the number of repeating keywords.

Software	Category	Data pairs	Correlation	Significance
FreeBSD	Evolution	43	0.344	0.05
Linux	Maintenance	115	0.872	0.01

As discussed in Section 2.1, one keyword can be mapped to one evolution or maintenance activity. The number of times a keyword appears in different release notes or change logs represents the number of times the same maintenance or evolution activity is performed in different versions. Figure 9a shows the distribution of the evolution keywords with respect to the number of times they appear in different versions of FreeBSD release notes; Figure 9b shows the distribution of the maintenance keywords with respect to the number of times they appear in different versions of Linux change logs.

Figure 9. The distribution of keywords with respect to number of appearances: (a) the evolution keywords in FreeBSD release notes; and (b) the maintenance keywords in Linux change logs.

Version distance indicates the time span between the first time one maintenance/evolution activity is opened and the last time the maintenance/evolution activity is reopened and finally closed. Figure 10a shows the distribution of the evolution keywords with respect to version distance in FreeBSD release notes; Figure 10b shows the distribution of the maintenance keywords with respect to version distance in Linux change Logs.

Figure 10. The distribution of keyword with respect to version distance: (a) the evolution keywords in FreeBSD release notes; and (b) the maintenance keywords in Linux change logs.

Figure 9 and Figure 10 show that most activities (evolution/maintenance) are finalized in one revision. The number of reopened activities (evolution/maintenance) decreases as the number of appearances and version distance increase.

Software maintenance and evolution are usually considered as two processes fulfilling different objectives, in which maintenance process is related with error-fixing activities and evolution process is related with features-incorporating/improving activities. In this study, we found that these two processes share some common properties, as demonstrated through mining the corresponding keywords in release notes and change logs.

Keywords that have greater number of appearances and larger version distance are related with complicated development (maintenance or evolution) activities. A frequently reopened and long lasting maintenance activity might indicate a hard removable bug, which not only consumes developers' effort but also degrades the product quality. A frequently reopened and long lasting evolution activity might involve incorporating a complicated feature that needs to be implemented and tested many times. Applying keyword-based text mining techniques on changes and releases notes can also help identify these activities and more attentions can be given to finalize the solution sooner.

4 Conclusions and Future Work

In this paper, we presented a keyword-based approach to mining non-source code documents and defined a mathematical framework to represent and interpret the data. This approach was applied in the study of the change logs of Linux and the release notes of FreeBSD. The results showed that the software maintenance process and evolution process share some common properties.

1. The new maintenance/evolution activities are not clearly separated from repeating maintenance/evolution activities. Instead, the number of new maintenance/evolution activities is linearly correlated with the number of repeating maintenance/evolution activities in each release;
2. The majority maintenance/evolution activities are finished in one release. The number of reopened activities (evolution/maintenance) decreases as the number of appearances and version distance increase.

This is our initial results of applying text mining techniques on release notes and change logs to understand software maintenance and evolution. In the future, we will extend the mathematical framework presented in this paper and mine other software documents. Specifically, our future work will focus on three areas.

1. Set up a maintenance activity dictionary and an evolution activity dictionary to help identify different types of keywords.

2. Mine keywords in source code and non-source code documents to verify the accuracy and the completeness of using non-source code document, such as release notes and change logs, to retrieve maintenance and evolution information.
3. Mine Concurrent Versions Systems (CVS) and extract activity keywords of different software products to study the co-evolution of similar software products. For example, the maintenance and evolution activities of FreeBSD [14] and OpenBSD [15] might be correlated; mining the same keywords appearing in their CVS archives might help understand their co-evolutions.

5 References

1. I. Sommerville, *Software Engineering*, 8th Edition, Addison Wesley, 2006.
2. C. F. Kemerer and S. Slaughter, "An Empirical Approach to Studying Software Evolution," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, 1999, pp. 493–505.
3. K. Chen, S. R. Schach, L. Yu, G. Z. Heller and J. Offutt, "Open-Source Changelogs," *Empirical Software Engineering*, vol. 9, no. 3, 2004, pp. 197–210.
4. L. Yu, "Indirectly Predicting the Maintenance Effort of Open-Source Software," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 18, no. 5, 2006, pp. 311–332.
5. R. Feldman and J. Sanger, *The Text Mining Handbook*, Cambridge University Press, 2006.
6. O. Baysal and A. J. Malton, "Correlating Social Interactions to Release History During Software Evolution," *Proceedings of the 4th International Workshop on Mining Software Repositories*, Minneapolis, Minnesota, May 2005, pp. 7.
7. R. Koschke and D. Simon, "Locating Features in Source Code," *IEEE Transactions on Software Engineering*, vol. 29, no. 3, 2003, pp. 210–224.
8. N. Wilde and M.C. Scully, "Software Reconnaissance: Mapping Program Features to Code," *Journal of Software Maintenance: Research and Practice*, vol. 7, no. 1, 1995, pp. 49–62.
9. G. Antoniol and Y. Gael, "Feature Identification: An Epidemiological Metaphor," *IEEE Transactions on Software Engineering*, vol. 32, no. 9, 2006, pp. 627–641.
10. M. Ohba and K. Gondow, "Toward Mining Concept Keywords from Identifiers in Large Software Projects," *Proceedings of the 2nd International Workshop on Mining Software Repositories*, St. Louis, Missouri, May 2005, pp. 1–5.
11. The FreeBSD Project, <http://www.freebsd.org/>
12. The Linux Kernel Archive, <http://www.kernel.org/>
13. B. Nolan, *Data Analysis, an Introduction*, Polity Press, Cambridge MA, 1994.
14. FreeBSD CVS Repositories, <http://www.freebsd.org/cgi/cvsweb.cgi/>
15. OpenBSD CVS Repositories, <http://www.openbsd.org/cgi-bin/cvsweb/>