

Towards Automatic Generation of Short Summaries of Commits

Siyuan Jiang and Collin McMillan
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
Email: {sjiang1, cmc}@nd.edu

Abstract—Committing to a version control system means submitting a software change to the system. Each commit can have a message to describe the submission. Several approaches have been proposed to automatically generate the content of such messages. However, the quality of the automatically generated messages falls far short of what humans write. In studying the differences between auto-generated and human-written messages, we found that 82% of the human-written messages have only one sentence, while the automatically generated messages often have multiple lines. Furthermore, we found that the commit messages often begin with a verb followed by a direct object. This finding inspired us to use a “verb+object” format in this paper to generate short commit summaries. We split the approach into two parts: verb generation and object generation. As our first try, we trained a classifier to classify a diff to a verb. We are seeking feedback from the community before we continue to work on generating direct objects for the commits.

I. INTRODUCTION

A commit is the action of software developers submitting a software change to a version control system. Commits can have commit messages, which are often written by developers to describe the changes. Commit messages are important because developers use them to review, validate, and understand the commits, but commit messages sometimes are non-informative or even empty [1].

To address this problem, automatic commit message generation techniques have been proposed. They often use program analysis and differencing techniques to generate summaries of changes [1]–[4]. These summaries are much shorter than the diff files (generated by differencing tools), but the summaries still tend to have multiple lines. Other techniques generate commit messages from other project documents. For example, Rastkar and Murphy proposed to generate the commit messages from user stories [5]. The summaries generated by these techniques are useful, but what is still missing is one-sentence summaries which convey the key ideas of commits.

The idea of generating one-sentence summaries is based on our exploratory data analysis on the two million commit messages that we present in this paper. We used natural language processing (NLP) techniques to analyze the text of the commit messages and found that the majority of the commit messages are only one sentence long, and **nearly half of the commit messages begin with a verb followed by a direct object**. This finding inspired us to design a method for generating commit summaries that are similar

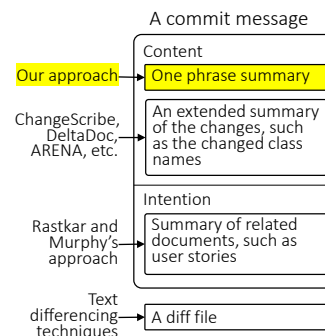


Fig. 1. Existing techniques for commit message generation compared to our approach

to what developers write: “verb + object”. These one-phrase summaries can be the leading sentences or the topics of the summaries generated by the existing techniques (Figure 1).

We divided our work into three parts: the exploratory data analysis, the verb generation and the direct-object generation. We summarized the exploratory data analysis in the previous paragraph and describe it in Section IV. For the verb generation, we trained a Naive Bayes classifier to identify verbs based on diff files that should be the key verbs in the commit messages. We also conducted a preliminary evaluation of the verb generation in Section V.

In this ERA paper, we present several open questions to the community that we hope will guide our future work, and in particular the generation of direct objects for the verbs.

Our contributions include:

- Using NLP techniques to analyze the commit messages, which enable us to analyze a large set of the messages (which we release in our online appendix)
- Discovery of a common phrase structure that is used by software developers to write commit messages, and a program that automatically extracts such phrase structure
- A proposal that aims to generate one-sentence commit messages that convey the key ideas of commits

In the rest of this paper, we will present a motivational example, the related work, the exploratory data analysis, the verb generation technique, and the future work.

Online Appendix We put our scripts and results on our online appendix: <http://nd.edu/~sjiang1/commitact>

<pre> Diff file --- a/trunk/src/core/com/lowagie/text/pdf/PdfStamperImp.java +++ b/trunk/src/core/com/lowagie/text/pdf/PdfStamperImp.java @@ -215,10 +215,9 @@ producer = Document.getVersion(); } else if (producer.indexOf(Document.getProduct()) == -1) { - StringBuffer buf = new StringBuffer(Document.getVersion()); - buf.append(" (originally created with: "); - buf.append(producer); - buf.append(')'); + StringBuffer buf = new StringBuffer(producer); + buf.append("; modified using "); + buf.append(Document.getVersion()); producer = buf.toString(); } // XMP </pre>
<pre> DeltaDoc When calling PdfStamperImp close () If indexOf (Document . getProduct ()) != -1 set producer = producer + "; modified using " + Document . getVersion () Instead of set producer = Document . getVersion () + "(originally created with : " + producer + ")" </pre>
<p>Our approach “change the producer info”</p>

Fig. 2. The diff file, the commit message generated by DeltaDoc [2], and the commit message that our approach aims to generate for Commit r3909 in iText (Section II)

II. EXAMPLE

In this section, we borrow the example of Commit r3909 in iText from the paper of DeltaDoc [2]. The diff file of r3909 and the summary generated by DeltaDoc are shown in Figure 2.

The size of the generated document is about half of the diff file, but it is still difficult to get the general idea at the first glance. Similarly, Changescribe [4] also generates messages that are several lines long. What is missing is a leading sentence that summarizes all the changes in a commit. Now consider the commit message that the developer wrote: “Changing the producer info.” This phrase contains the action of the commit, “change”, and what is the object of the action, “the producer info”. The developer can skim this phrase and understand what was changed in the commit.

Currently, our approach generates “change” for r3909, and in the future, we will have an approach to generate “the producer info”. The combination of the two approaches is going to generate phrases like “change the producer info”.

III. RELATED WORK

Our project has two parts: exploratory data analysis and commit message generation. Based on the two parts, we separate the related work into three categories: empirical studies about commit messages, empirical studies about diff files, and techniques that generate commit messages.

A. Empirical Studies about Commit Messages

Several empirical studies about commits messages have been conducted for commit classification and commit message generation [2], [3], [6]–[8]. For example, Moreno et al. [3]

manually inspected the existing release notes before they designed an approach to generate release notes automatically. Buse and Weimer [2] conducted a similar manual inspection for automatic commit message generation. Like these previous studies, our exploratory data analysis aims to gain insights for our approach of generating commit messages.

Different from the previous studies, we used natural language processing (NLP) techniques, which help us to mine information from the existing commit messages automatically and confirm hypotheses on a large data set. Besides manual inspection, the previous studies also computed the sizes of commit messages and analyzed the messages as bags of words [7], [8]. In contrast, we are able to conduct grammar analysis on the commit messages. The grammar analysis led to a key finding that shaped our approach.

B. Empirical Studies about Commit Changes

There are many empirical studies about the changes in commits [6]–[9]. For example, Fluri et al. studied change types based on their syntax differencing technique [9]. Currently, we have not conducted an empirical study on the commit changes, but we plan to study the content of the diff files in the future. Instead of looking for change types, we will study whether there are overlapped words in the commit messages and their diff files and where we can locate the overlapped words in the diff files.

C. Commit Message Generation Techniques

A common way to generate commit messages is summarizing code changes of a commit [2]–[4]. Many techniques use syntax differences to present code changes [2]–[4], [9]. Different from the existing techniques, we use diff files (generated by git diff command) in our approach. Diff files are textual differences and easy to obtain. On the other hand, syntax differencing requires code parsing. Additionally, syntax differencing includes only code changes, while diff files contain other changes, such as comment and makefile changes. While the two differencing types have their own advantages and disadvantages, we chose to use the diff files as our first try, because they are easier to obtain.

To include context information in a commit message, several approaches consider the information outside the text or code changes of a commit [3], [5], [10]. While we agree that the context information is an important part of a commit message, our approach is currently focusing on summarizing text changes into a short sentence to increase readability and interpretability of a commit message.

Our approach to generate a verb for a commit is similar to the approach taken by Le et al. to link issue reports to commits [10]. Le et al. conducted textual similarity analysis between commit messages and issue reports where they used term frequency-inverse document frequency (tf-idf) to represent commit messages and issue reports. We also used tf-idf, but tf-idf is used to represent the diff files instead of the commit messages.

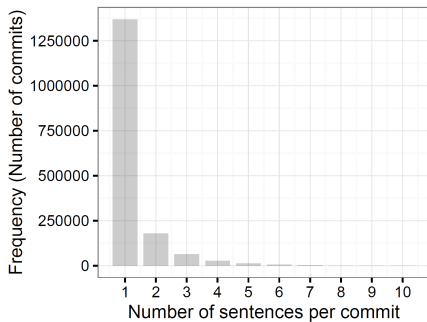


Fig. 3. Histogram of number of sentences in the commit messages

IV. EXPLORATORY DATA ANALYSIS

We conducted an exploratory data analysis that is similar to the analysis done by Hattori and Lanza [7]. Hattori and Lanza found that most commits include few files and very few commits have hundreds of files. Likewise, we found that most commit messages have few sentences and few commit messages have more than ten sentences.

The Data Set First, we obtained 967 commits from the work by Mauczka et al [11]. Second, we obtained all the commits from the top 1,000 popular Java projects in Github (due to space limit, we put the details on our online appendix, Section I). Then, we filtered the commit messages that are empty or have non-English letters. In the end, we obtained 2,027,734 commits.

Removing Special Commits We excluded the rollback and merge commits from our analysis. Version control systems often provide automatic commit messages for rollbacks and merges, such as, “merge commits X and Y”. In the two million commits, we removed nearly 400k rollbacks and merges by checking whether the commit messages are begun with “merge” or “rollback”.

Number of the Sentences In the remaining 1.6 million commit messages, we counted the number of the sentences in each commit message by using Stanford CoreNLP [12]. The majority of the commit messages have few sentences. 82% of the commit messages have only one sentence. Only 0.2% of the commit messages have more than ten sentences. Figure 3 shows the histogram of the number of the sentences in the commit messages (excluding the messages have more than ten sentences due to space limit).

Grammar Analysis on the Commit Messages We took two steps in the grammar analysis. First, we manually read 12 randomly-sampled commit messages from the commits we obtained from Mauczka et al [11]. In this step, we formed the hypothesis that “verb + object” is a common phrase structure in the commit messages. Second, to confirm the hypothesis, we used Stanford CoreNLP [12] to detect the verbs and their direct objects in the first sentences of the commit messages. In the 1.6 million messages, we found 763,826 messages (which is 47% of the 1.6 million messages) where the first sentences are begun with a verb and its direct object.

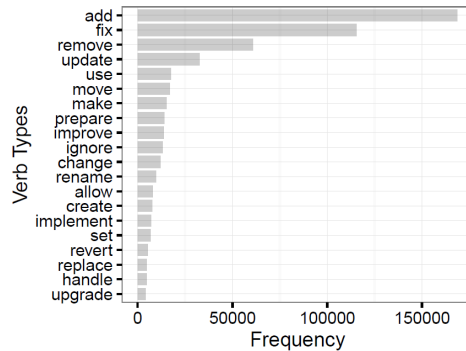


Fig. 4. Histogram of the verbs in commit messages

TABLE I
THE VERB GROUPS

Id	Verb types	Id	Verb types	Id	Verb types
1	add, create, make, implement	6	move, change	12	allow
2	fix	7	prepare	13	set
3	remove	8	improve	14	revert
4	update, upgrade	9	ignore	15	replace
5	use	10	handle		
		11	rename		

V. CLASSIFYING DIFFS INTO VERB GROUPS

In this section, our goal is to generate a verb from a commit. We used diff files (i.e., textual differences) to represent the changes of commits because diff files can be easily obtained by git diff command. Then we treated the problem of verb generation as a multiclass classification problem—classifying a diff file into one of the verb groups, where a verb group is a group of verbs that have similar meanings. As the first step, we define our verb groups in the following section.

A. Verb Groups

When we analyzed phrase structures of the commit messages (Section IV), we retrieved for each commit a verb from the commit message. There are 763k verbs in total. We transformed the verbs into their lemmas and we called each distinct lemma a verb type. There are 4962 verb types in the 763k verbs. Figure 4 shows the histogram of the 20 most frequent verb types. Alali et al. [6] has reported a list of frequent words in commit messages, which overlap with our frequent verb types.

From all the verb types, we considered only the 20 most frequent word types, which cover 70% of the commit messages (537k commit messages). We grouped similar word types by using a word embedding tool¹, which uses word2vec method [13]. Finally we manually inspected the grouped verbs and added “implement” to the group of “add”. There are 15 verb groups in total, which are shown in Table I. The first, third, and fifth columns list the ids of the verb groups.

Labeling To label each diff file, we used the verb that we extracted from the commit message, and we labeled the diff file with the verb group id that includes the verb. The verb

¹http://bionlp-www.utu.fi/wv_demo/ On this webpage, we looked up the 20 nearest words for each verb type. Two verb types are grouped together if one verb type is in the other verb type’s 20 nearest word list.

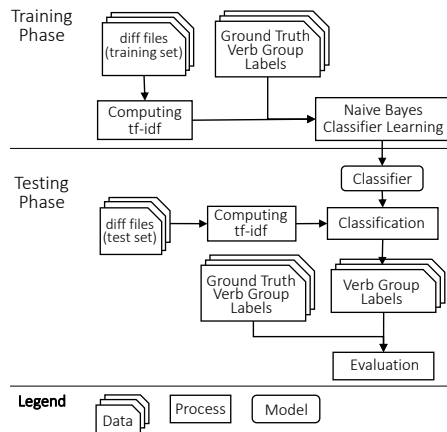


Fig. 5. The overall approach

groups only include the 20 most frequent verb types, in this study, we excluded the diffs that have other verbs. In total, we have 537k labeled diff files.

B. The Data Set

We removed the diff files that are larger than 1MB due to space limit. We also removed the diff files that have non-ascii codes. In the end, we have 509k labeled diff files. We randomly selected 3k diff files as the test set and the rest of the diff files are used for training.

C. Overall Approach

The overall approach is shown in Figure 5. We chose a Naive Bayes classifier to classify the diff files into the verb groups. Before we train the classifier on the diff files, we computed tf-idf (term frequency-inverse document frequency) for every word type (i.e., distinct word) as the features of the diff files. Tf-idf is a common textual feature that evaluates the importance of a word type by two factors: 1) the number of times the word type occurs in a diff file divided by the total number of words in the diff file, and 2) the number of times the word type occurs in all the diff files [10].

D. Evaluation

The overall accuracy is 39%; the precision is 43%; and the recall is 39%. The classifier works best for verb groups 1 and 9. The precision for verb group 1 is 38% and the recall is 100%; the precision for verb group 9 is 100% and the recall is 41%. Although we trained the classifier with 15 verb groups, the classifier classified the test set into five verb groups and was not able to detect any of the other ten verb groups. We plan to improve our training approach by 1) trying other machine learning techniques, such as random forests [10]; 2) using SMOTE [14] to address the problem of the unbalanced data set (most of the diffs are labeled with verb group 1).

VI. DISCUSSION AND FUTURE WORK

In the process of this project, we have formed several potential research questions to be discussed in the conference.

We hope the conversions at the conference will help in directing us towards answering these questions.

RQ1 What techniques are appropriate for generating direct objects for the commits? We observed that the direct objects often occur in the diff files. So one of our options is to use extractive summarization techniques to extract the “direct objects” from the diff files.

RQ2 What machine learning models and features suit verb-generation task better? To improve our verb-generation approach, we can try other classification methods, such as decision trees. Feature-wise, diff files follow a certain format and we can create some features to represent the characteristics of a diff file, for example, the number of “+” in a diff file.

RQ3 To what extent are the short summaries useful? Although we think the short summaries are useful based on our experience, we need to conduct a study to confirm our hypothesis. Our current assumption is that the short summaries help developers understand a commit more quickly.

ACKNOWLEDGMENT

This work was partially supported by the NSF CCF-1452959 and CNS-1510329 grants, and the Office of Naval Research grant N000141410037. Any opinions, findings, and conclusions expressed herein are the authors’ and do not necessarily reflect those of the sponsors.

REFERENCES

- [1] M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk, “Changscribe: A tool for automatically generating commit messages,” in *ICSE ’15*, vol. 2, May 2015, pp. 709–712.
- [2] R. P. Buse and W. R. Weimer, “Automatically documenting program changes,” in *ASE ’10*, 2010, pp. 33–42.
- [3] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora, “Automatic generation of release notes,” in *Proceedings of the 2014 FSE*, pp. 484–495.
- [4] M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk, “Changscribe: A tool for automatically generating commit messages,” in *2015 IEEE/ACM 37th IEEE ICSE*, vol. 2, pp. 709–712.
- [5] S. Rastkar and G. C. Murphy, “Why did this code change?” in *Proceedings of the 2013 ICSE*, ser. ICSE ’13, 2013, pp. 1193–1196.
- [6] A. Alali, H. Kagdi, and J. I. Maletic, “What’s a typical commit? a characterization of open source software repositories,” in *2008 16th IEEE Intl. Conf. on Program Comprehension*, pp. 182–191.
- [7] L. P. Hattori and M. Lanza, “On the nature of commits,” in *2008 23rd ASE - Workshops*, pp. 63–71.
- [8] A. Hindle, D. M. German, M. W. Godfrey, and R. C. Holt, “Automatic classification of large changes into maintenance categories,” in *2009 IEEE 17th ICPC*, pp. 30–39.
- [9] B. Fluri, M. Wuersch, M. Pinzger, and H. Gall, “Change distilling: tree differencing for fine-grained source code change extraction,” *IEEE TSE*, vol. 33, no. 11, pp. 725–743, 2007.
- [10] T. D. B. Le, M. Linares-Vasquez, D. Lo, and D. Poshyvanyk, “Relinker: Automated linking of issue reports and commits leveraging rich contextual information,” in *2015 IEEE 23rd ICPC*, pp. 36–47.
- [11] A. Mauczka, F. Brosch, C. Schanes, and T. Grechenig, “Dataset of developer-labeled commit messages,” ser. MSR ’15, pp. 490–493.
- [12] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, “The Stanford CoreNLP natural language processing toolkit,” in *ACL System Demonstrations*, 2014, pp. 55–60.
- [13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *NIPS*, 2013, pp. 3111–3119.
- [14] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: synthetic minority over-sampling technique,” *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.