International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2014): 5.611

Automatic Generation of Commit Messages using Natural Language Processing

M Vishalakshi[1], Dr. V. Krishnapriya[2]

[1] Research Scholar (MPhil CS), Department of Computer Science, Sri Ramakrishna College of Arts and Science for Women, Coimbatore

[2] Head of the Department, Dept. of Computer Science, Sri Ramakrishna College of Arts and Science for Women, Coimbatore

Abstract: ***Software development requires Version Control System to manage and manipulate the changes made to source code. When a change is done in the file, related information is updated as commit message. Most of the time, commit messages are empty or very short. Accurate and complete commit messages summarizing the software changes are important for tracking the development and maintenance activities of a project. This paper presents an approach of Natural Language Processing for generating automatic commit messages, based on code changes included in a changeset and simultaneously integrated to software usage library to read the document files of the software. Commit messages are found useful, and to present an initial model of output for natural-language commit messages using verb phrases and their associated direct objects.***

Keywords: Version Control, Software, Revision History, Commit Message, changeset, Natural Language Processing..

1. Introduction

In earlier days, software application was developed in various languages like Visual Basic, FORTRAN, COBOL, Dotnet, Java, etc. Now, the trend is changed and many open source projects became common. Generally, open source projects are developed in many languages and we should concentrate more to acquire quality of the software application.

In market, there are various tools available for revision maintenance, automatic commit messages and tracking revision history of the source code. Metrics taken from Version Control Systems like Subversion, Clearcase, GIT collect and save the changes as revisions and maintain the history of the files.

Despite developers are allowed to enter commit message when a change is done, most of the time, it is not complete and do not have all the relevant change notes. If the commit messages have complete information, then it would help other developers to understand and validate the changes. There were approaches to create automatic Commit messages for a change set using Visualization, Code Summarization, Line based differencing, and multi document summarization approaches.

In software development, team work and collaboration plays vital role, as every team member will work on part of the software application. Hence, it's very important to have awareness of what other team member has worked on. But, it's practically difficult to get that information, if it's not tracked or stored in a system. If the version control system collects all these information properly and store it, then the developer can work efficiently, this in turn saves time and improve productivity of the entire team.

A successful software engineering company nowadays is often spread over multiple locations or has an offshore software production. The teams have to work across borders as well as the differences of cultures and face the challenges of distributed software development. In this situation communication and collaboration are of utmost importance [1-2]. The design of a well defined API, for example using contracts as discussed in the paper by Nordio et al [3], becomes essential. The effect of distribution on software development has been researched from different angles [4-5]. Espinosa et al [4] looked at the impact of time zones on the performance during software development. During the DOSE [6-7] university course, Nordio et al [3] studied the effect of time and cultural differences on the communication within the teams. Possible tactical approaches to face global software development are discussed by Carmel et al [2].

2. Version Control System

Version control systems (VCS) are used in almost any software project with multiple team members. Teamwork requires sharing of files. In Software Engineering, VCS are the approved solution for managing text files and releases. During the last few years distributed VCS like Git or Mercurial became increasingly more popular. With the ability to work independently of a server and a centralized repository, the projects gain flexibility when branching and merging. Whether you choose a centralized or a distributed system, version control is a time-consuming, non-trivial activity. The cycle of sharing content takes multiple operations, commit, pull, push, merge and resolve. This paper proposes a solution to reduce the time overhead introduced by the standard version control systems. The version control activities are simplified and automatized while conflicts are avoided and resolved using change awareness. However using the change awareness the developers will be implicitly in the loop about the changes on the other tasks.

Each VCS has its own terminology.
Some of the terms used in this report are adopted from the distributed version control system Git [8]. Other notions are specific to simplified version control.

Repository: A repository is the folder of the VCS containing its Meta data and, if the repository is not bare, the working directory. The Meta data consists of all needed version control information like the different revisions, the branches and the configuration.

Working directory: The working directory contains the checked-out version of the files. The files can be modified without using the VCS. They also will be influenced by certain version control operations like a rollback.

Staged changes: The VCS is unaware of modifications made in the working directory without its help. By staging the changes, they are added to the index of the VCS and they will be automatically included in the next revision.

Uncommitted changes: The modifications that differ from the head revision of the working directory are denoted as uncommitted. This equals the staged as well as the unstaged changes.

Revision history: The commits leading to the head revision make up the commit history. A revision may have more than one parent if it's a result of a merger of multiple revisions. The revision history can therefore be more complex than a linear list of revisions.

Head revision: The head revision is the last commit in the revision history. Usually this will also be the most recent commit, unless the head is moved by going to a different revision.

Branch: A repository can have multiple branches. Each branch has its own head revision. When the repository is created, it already has one branch denoted as the default or master branch.

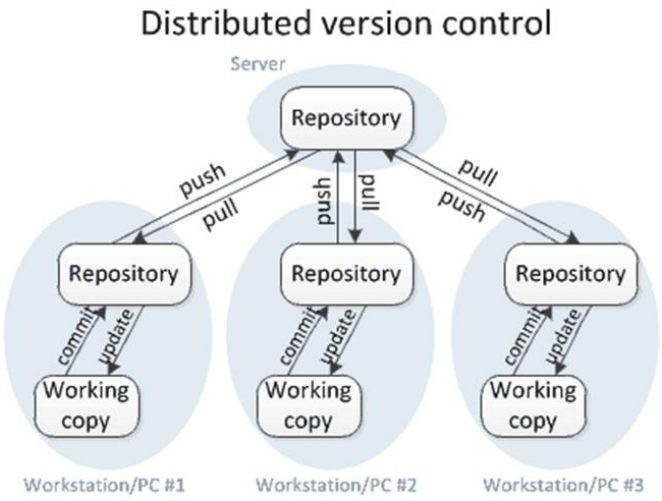
Current branch: The current branch is the branch checked out in the working directory.

Commit: By committing a new revision is created in the repository with either all the modifications or only the staged changes depending on the options used.

Share changes: Sharing refers to exchanging and merging revisions with the main repository. Depending on the circumstances, sharing changes includes a previous commit. The term is represented by a chain of operations in Git and can also be compared to the notion of "update" used by centralized VCS like SVN.

3. Software Commit Messages

In version control systems, set of files and directories are maintained in repository and the changes done to files in the repository are updated with commit operation. Hence, the change information is tracked with commit messages. This is helpful to developer team and project manager for making decisions and project maintenance.



Lets discuss about the importance of the commit message. Main objective of the Commit messages is to provide information about *What* and *Why* of the changes done. *What* refers to the implementation of the change and *Why* refers the context for the change. Line based differencing tools provide details of the change, but not the context. A tool called ChangeScribe is developed to address the issue above. This tool generates an automatic message with the code changes and their impact set.

A. Describing and augmenting the Context of code changes

- There are tools like Semantic Diff tool and Line based Differencing tools for analyzing the source code version at bytecode level. ChangeScribe is also line based differencing tool, it augments the context of the changes with a natural language description that includes the commit stereotype, change description and change set. [2]
- DeltaDoc is another tool which used Symbolic execution and summarization techniques to populate text message for all the changes done.
- Visualizing tools are used to augment the code context of the source code. This provides visualization of the changes at different granularity levels.

B. Natural Language Descriptions of Software Artifacts

During Software maintenance, it's very essential to have the complete details of a changeset. Summarizing the changes done in methods and classes using different information retrieval techniques and natural language processing.
Summaries should contain all the salient features of the source code change and the sentences are generated with structural and natural language information. Some techniques for Java source code have below 3 elements:
1) General description of the object in a class
2) Class stereotype information with responsibilities
3) Class behavior description

Other techniques already available are summarizing java code fragments using machine learning technique and summary of Java methods with local and contextual information. ChangeScribe uses code summarization techniques based on NLP[2]

C. Empirical Studies on Characterizing Commit Messages
Various studies are conducted for characterizing commit messages. Most of the time, the messages are descriptive to small extent and contain few words (1 to 15 words). Some other studies do not reveal the characteristics of commit messages. But, they are used to categorize based on change type.

4. Models of the File Content

The file content is modelled by three classes as shown in the figure 6.1. The file specified by its path is represented by a CombinedFileModel instance containing a map of lines. Each line is a LineModel object with a randomly generated integer as a unique identifier. Using this identifier the line points to its predecessor and its successor. A line has multiple line version arranged in a tree. Each line version represented by the LineVersionModel class has a content and a list of tags for which this content applies. The CombinedFileModel is, as its name says, a combination of all versions of the file. It assembles the file content of each repository and each branch and differentiates between whether the content was committed or not.

The model is persisted by saving the serialized object in the project folder. The CombinedFileModel is initialized when the file is first opened in the web-based IDE or if it is altered in a commit from an external repository. The initialization is based on the fact that when initialing the file contents are all the same for all versions of the file. As the model is persisted the challenge is no longer the initialization, but to keep it synchronized to the content in the version-controlled repositories.

ChangeScribe generates sentences for class signatures (a.k.a., class declaration) using the class stereotypes proposed , following template is used to generate sentences for class signatures:

<change type> <class stereotype> <represented object>. It allows: <methods description>

Table 4.1. EXAMPLE OF ChangeScribe's COMMIT MESSAGE LISTING IMPACT SET DETAILS (CLASSES IMPACTED BY A METHOD ADDITION/DELETION)
BUG - FEATURE: <type-ID>

This is a small modifier commit that does not change the system significantly. This change set is mainly composed of:
1. Changes to package org.springframework.social.oauth2:
1.1. Modifications to AccessGrant.java:
1.1.1. Add a constructor method
The added/removed methods triggered changes to OAuth2ProviderSignInAccount class
2. Changes to package org. spring framework. social. web.signin:
2.1. Modifications to OAuth2ProviderSignInAccount.java:
2.1.1. Modify arguments list when calling connect method at connect (Serializable) method
These techniques are called Natural Language Program Analysis (NLPA), and work by combining knowledge of the structure of the English language with knowledge about the structure of source code, in order to meaningfully extract information from the code to aid in the process of software maintenance

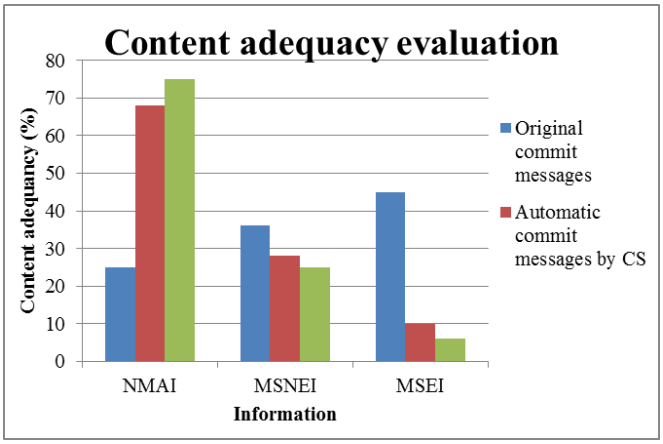
5. Experimental Analysis

As the first step of the analysis, one of the authors evaluated the content adequacy of the commit messages created by the participants in order to determine whether each respondent understood the shown changes. It is worth noting that the evaluator was quite familiar with each changeset included in the study, and thus, he was competent to judge this property of these commit messages. The result of this evaluation showed that 10% of the commit messages generated by the participants (12 commit messages out of the 119) did not contain correct information, and therefore, indicated a poor understanding of the changes done.

As mentioned above, the participants were asked to evaluate both the commit messages generated by ChangeScribe and the commit messages written by the original developers. The properties evaluated were: content adequacy, conciseness, and expressiveness. Content adequacy judges whether the contains all important information about the changes done. Conciseness assesses whether a commit message is clear and succinct or, in other words, if it does not contain superfluous and unneeded information. Expressiveness evaluates if a commit message is easy to read and if the way it is presented facilitates understanding of the changes done

A.Content Adequacy
Consider this property as the most important one since commit messages that contain all essential information about the changes done may ease a number of maintenance tasks. The results show that only in 13% of the cases proposed approach generated commit messages that missed essential information. Conversely, the original commit messages miss essential information in 40% of the cases (Table 4.1). In general, this result indicates that the approach achieves a significant improvement in terms of relevant information needed to properly explain the changes done by the committee, and thus, its use might substantially alleviate a well-known maintenance issue.

Figure 5.1. Content adequacy evaluation of the original, automatic commit messages by ChangeScribe and automatic commit messages by Natural Language Processing with Software Model (NLP-SM)



On the other hand, the Figure 5.1 results show that proposed Natural Language Processing with Software Model (NLP-SM) approach is able to generate a commit message that includes all essential information of the changes done in 75% of the cases, while the messages written by the developers only reach this degree of completeness in 35% of the cases. From this point of view, the improvement achieved by ChangeScribe is also significant. In terms of statistical significance of the results, the difference is significant (p-value = 1:543E-08) between the content adequacy rankings of the original messages and the messages by ChangeScribe; and the magnitude of the difference is large (d = -0:9386784).

Table 5.1: Content adequacy evaluation of the original automatic commit messages by ChangeScribe and automatic commit messages by Natural Language Processing with Software Model (NLP-SM)

Response	Original commit messages (% ratings)	Automatic commit messages (% ratings)	
		Change Scribe(CS)	Natural Language Processing with Software Model (NLP-SM)
	Content adequacy (%)		
Not missing any information (NMAI)	25	68	75
Missing some no essential information (MSNEI)	36	28	25
Missing some essential information (MSEI)	45	10	6

6. Conclusion

When making changes to software, developers spend more time trying to understand code rather than implementing changes. Critical to assisting developers in understanding code is human-written documentation. Unfortunately, in many contexts the documentation is not as good as it could be. Therefore, if linguistic information can be extracted from code and presented as documentation, it provides support for developers when in cases where documentation is lacking. This paper presents an approach Natural Language Processing (NLP) for generating automatic commit messages based on the code changes included in a change set and simultaneously NLP is integrated to software usage library to read the document files of the software. ChangeScribe extracts and analyzes the differences between two versions of the source code, and also, performs a commit characterization based on the stereotypes of methods modified, added and removed.

Practically, its difficult for a developer to concentrate on changes done and update it in the commit message, due to urgencies and making multiple changes at a time. Hence, automatic generation of change information, during commit, makes the developer comfortable and not worry about the tracking of changes made by him. This in turn, improves productivity and team collaboration. Also, the effectiveness and conciseness will be measured in the future scope.

7. Acknowledgment

For all the efforts behind this paper work, I first & foremost would like to express our sincere gratitude to the staff of Department of Computer Science, for the extended help and suggestions at every stage of this paper. It is with a great sense of gratitude, that we acknowledge the support, on time suggestion and highly indebted to our guide Head of the Department Dr. V. Krishna Priya. Finally I pay sincere thanks to all those who indirectly and directly helped us towards the successful completion of this paper.

References

[1] M. Nordio, H.-C. Estler, B. Meyer, J. Tschannen, C. Ghezzi, and E. D. Nitto. How do distribution and time zones affect software development? Conference on Global Software Engineering (ICGSE 2011). IEEE, 2011.
[2] E. Carmel and R. Agarwal. Tactical approaches for alleviating distance in global software development. IEEE Softw., 18:22–29, March 2001.
[3] M. Nordio, R. Mitin, B. Meyer, C. Ghezzi, E. D. Nitto, and G. Tamburrelli. The role of contracts in distributed development. In Proceedings of Software Engineering Approaches for Offshore and Outsourced Development, 2009.
[4] J. A. Espinosa, N. Nan, and E. Carmel. Do gradations of time zone separation make a difference in performance? A first laboratory study. In Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE 2007), pages 12–22. IEEE, Aug. 2007.
[5] H.-C. Estler, M. Nordio, C. A. Furia, B. Meyer, and J. Schneider. Agile vs. structured distributed software development: A case study. In Proceedings of the 7th International Conference on Global Software Engineering. IEEE, 2012.
[6] M. Nordio, C. Ghezzi, B. Meyer, E. D. Nitto, G. Tamburrelli, J. Tschannen, N. Aguirre, and V. Kulkarni. Teaching software engineering using globally distributed projects: the DOSE course. In Collaborative Teaching of Globally Distributed Software Development -

Community Building Workshop (CTGDSD), New York, USA, 2011. ACM.
[7] M. Nordio, R. Mitin, and B. Meyer. Advanced hands-on training for distributed and outsourced software engineering. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE), pages 555–558. IEEE, 2010.
[8] Wikipedia - Git (Software). http://en.wikipedia.org/wiki/Git_ (software), Sept. 2012.
[9] Cortes-Coy, L.F. ; Linares-Vasquez, M. ; Aponte, J. ; Poshyvanyk, D. "On Automatically Generating Commit Messages via Summarization of Source Code Changes ", Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on 2014 , Page(s): 275 – 284
[10] de Moura, M.H.D. ; do Nascimento, H.A.D. ; Rosa, T.C. "Extracting New Metrics from Version Control System for the Comparison of Software Developers" Software Engineering (SBES), 2014 Brazilian Symposium, 2014 , Page(s): 41 – 50
[11] Pradeep Singh, K. D. Chaudhary, Shrish Verma, "An Investigation of the Relationships between Software Metrics and Defects" International Journal of Computer Applications (0975 – 8887) Volume 28– No.8, August 2011

Author Profile

Vishalakshi Muthukumar received the MCA degree from Indira Gandhi National Open University, New Delhi. She is a student of MPhil, (Computer Science), Sri Ramakrishna College of Arts and Science for Women, Coimbatore.