ChangeScribe: A Tool for Automatically Generating Commit Messages

Mario Linares-Vásquez[1],Luis Fernando Cortés-Coy[2], Jairo Aponte[2], Denys Poshyvanyk[1]
[1]The College of William and Mary, Williamsburg, VA, USA
[2] Universidad Nacional de Colombia, Bogotá, Colombia
mlinarev@cs.wm.edu, lfcortesco@unal.edu.co, jhapontem@unal.edu.co, denys@cs.wm.edu

***Abstract*—During software maintenances tasks, commit messages are an important source of information, knowledge, and documentation that developers rely upon. However, the number and nature of daily activities and interruptions can influence the quality of resulting commit messages. This formal demonstration paper presents *ChangeScribe*, a tool for automatically generating commit messages. *ChangeScribe* is available at http://www.cs.wm.edu/semeru/changescribe (Eclipse plugin, instructions, demos and the source code).**

***Index Terms*—Commit message, summarization, code changes**

I. Introduction

During software development process, changes to software artifacts are hosted in control version systems (CVS), and those changes can be partially documented by using commit messages (*a.k.a.*, commit notes or commit comments). The intended purpose behind commit messages is to describe the changes and help encoding rationale behind those changes. These descriptions can be used later by developers to understand and validate changes, locate and re(assign) bug reports, and trace changes to other artifacts. In general, commit messages are an important source of information, knowledge, and documentation that developers rely upon while addressing software maintenance tasks [9], [4], [8].

However, mostly because of the number and nature of daily activities by software developers [12], [16], [4], commit messages can be non-informative (*e.g.*, "initial commit", "last commit before lunch") or practically empty. Another possible explanation for the lack of descriptive/useful commit messages is the consideration that details about the changes and changed code units generated with line-based differencing tools are enough for understanding the change. According to Buse and Weimer [1], raw diffs are not always enough as a summary for some of the *what* questions about the change, because raw diffs only report textual differences between two versions of the files, which is often long and confusing, and does not provide developers with answers to many high-level questions. Therefore, line-based diffs do not provide enough context to understand the *why* behind the changes.

In this paper we present *ChangeScribe*, a tool aimed at assisting developers when committing changes, by automatically generating commit messages. *ChangeScribe* implements the summarization-based approach, which was presented and evaluated by developers in our previous work [3]. *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. The outcome is a commit message that provides an overview of the changes and classifies and describes in detail each of the changes; the message describes the $what$ of a change and provides context about the why using natural language. *ChangeScribe* also allows to control the length of the message by using an elegant impact set-based heuristic.

II. Related Work

ChangeScribe is mainly related to tools for augmenting the context of source code changes. Those tools are described as in the following.

Semantic Diff [11] detects differences between two versions of a procedure, and then summarizes the semantic differences by using program analysis techniques. Other approaches that improve line-based differencing tools are *LDiff* [2] and *iDiff* [17]. Parnin *et al*. [18] proposed an approach for analyzing differences between program versions at bytecode statement level; for describing the changes, type information and fully qualified source code locations of the changes (in the source entity and the entities impacted by the change) are presented. *ChangeScribe* also relies on line-based differencing, however it augments the context of the changes with a natural language description that includes the commit stereotype, change descriptions, and impact set.

DeltaDoc [1] automatically generates textual descriptions of source code changes using symbolic execution and summarization. However, when the change-set is very large (*i.e.* many files or methods), it describes each method separately ignoring possible dependencies of those methods. Rastkar and Murphy [19] proposed a multi-document summarization technique for describing the motivation behind a change. As compared to the approaches above, the commit messages generated by *ChangeScribe* contain more information on the *what* about the changes including information on dependencies and do not require using artifacts of multiple types.

The closest tool to *ChangeScribe* is *ARENA* [13]. It generates a textual description of the changes generated between two releases of a system. The purpose of *ARENA* is to generate detailed release notes that include information such as fixed/open bugs, licensing changes, and changes in the dependencies. Both *ChangeScribe* and *ARENA* use code summarization techniques, however the target audiences are different. Therefore, the information and the structure of the description are different in both cases. Meanwhile *ARENA*'s purpose is to generate release notes, which can be long and

very detailed, *ChangeScribe*'s is to include in the message more context of the changes by using commit stereotypes, impact sets, and specific templates.

The code context of source code changes can be also augmented using visualizing tools. For instance, *Commit 2.0* [4] augments commit logs with a visual context of the changes. *Commit 2.0* provides a visualization of the changes at different granularity levels, and allows developers to annotate the visualization. *ChangeScribe* only generates a textual description, however, a visualization like the one in *Commit 2.0* can be integrated into our tool.

III. CHANGESCRIBE

ChangeScribe is an Eclipse plugin that analyzes two versions of a system, and generates a textual description of the changes. In particular, *ChangeScribe* is integrated with the commit action of Eclipse, in such a way that automatically generates a commit message. The purpose of *ChangeScribe* is not to replace developers when writing commit messages, the purpose is to help developers to write descriptive commit messages. Therefore, the messages generated by *ChangeScribe* can be edited by developer before committing the code, and the length of the message can be tailored by using an impact set-based heuristic. In the following, we describe the features provided by *ChangeScribe* and plans for the future work.

A. Describing Source-Code Changes

ChangeScribe is integrated into the Eclipse IDE, and its functionality can be invoked via contextual menu or the menu bar. Current version of *ChangeScribe* only supports Java projects hosted in Git repositories. For the Git-based push and push-and-commit operations, and for extracting the change-set between two adjacent version (*i.e.*, HEAD version of the system in the Git repository and current version in the local workspace), *ChangeScribe* uses the *JGit*[1] Java Library. For each element of the change set, *ChangeScribe* identifies the change type (*i.e.*, addition, deletion or modification) and the renamed files. If a .java file is updated, *ChangeScribe* uses the *Change distiller* tool [7] to identify fine-grained code changes.

Both, changes types from the change-set and fine-grained changes, are used to generate the two parts of the commit message: general description, and detailed description. The former characterizes the change-set with a general overview of the commit. It has (i) a phrase describing whether it is an initial commit, (ii) a phrase describing commit's intent, (iii) a phrase indicating class renaming operations, (iv) a sentence listing the new modules, (v) a sentence indicating whether the commit includes changes to properties or internationalization files. Sentences (i) and (iii)-(v) are generated with *ChangeScribe* templates, and the commit's intent in sentence (ii) is based on the commit stereotypes proposed by Dragan *et al.* [5]. Because the commit stereotype identification relies on method stereotypes [6], *ChangeScribe* uses the *JStereoCode* tool [14].

[1]Implementation of Git SCM in Java. http://wiki.eclipse.org/JGit/

The second part of the message describes the changes made to each Java file, and the changes are organized according to packages. Based on the change type, if it was an addition or deletion, *ChangeScribe* describes the class' goal and its relationships with other objects. Moreover, if an existing file is modified, *ChangeScribe* describes the changes for each inserted, modified and deleted code snippet. *ChangeScribe* generates descriptive phrases for all changes at class/method/statement level. For instance for added/removed classes, *ChangeScribe* describes the class responsibility based on the approach by Hill *et al.* [10], and for describing classes signature our tool uses the class stereotypes defined by Moreno *et al.* [15]. For modified classes, *ChangeScribe* generates descriptions with the information provided by *Change Distiller* and the sentence templates proposed in our previous work [3]. For more information about the templates and commit message generation we refer the interested reader to [3].

When the commit message is generated it is displayed in the main window as presented in Figure 1-b. The main window includes: an editable text field with the commit message ①; the commit stereotype signature ②, which depicts the distribution of method stereotypes in the commit[2]; an iconized-button group ③ for showing the online help, refreshing the commit message, and (un)selecting all the files in the commit ④; the list of modified files ④ allows for individual selection and has a file name-based filtering; and finally, a button group ⑤ for committing-and-pushing/committing the code, and closing the window. The following snippet shows part of a message generated for a commit (http://goo.gl/IV6aWm) of Apache Solar at GitHub:

> *This is a state update modifier commit: this change set is composed only of mutator methods, and these methods provide changes related to updates of an object's state. This change set is mainly composed of:*
> *1. Changes to package org.apache.solr.common.cloud:*
> *1.1. Modifications to ClusterState.java:*
> *1.1.1. Remove an unused functionality to get shard*

Although the real commit message is "SOLR-2592: realtime-get support", *ChangeScribe*'s is more descriptive and provided augmented information that helps to understand the rationale behind the change (*i.e.*, the `getShard` method at `ClusterState.java` was removed because the method was dead code).

Because stereotypes and their semantic (*e.g., state update modifier*) may be unknown for developers, *ChangeScribe*'s main window includes an online help that describes both method and commit stereotypes (Figure 1-c).

B. Impact Set-Based Filtering

Large commits lead to large descriptions. In fact, findings in our previous work [3] suggest that some developers find large commit messages superfluous, because giving a detailed

[2]The signature includes tooltips over each color bar explaining the corresponding method stereotype

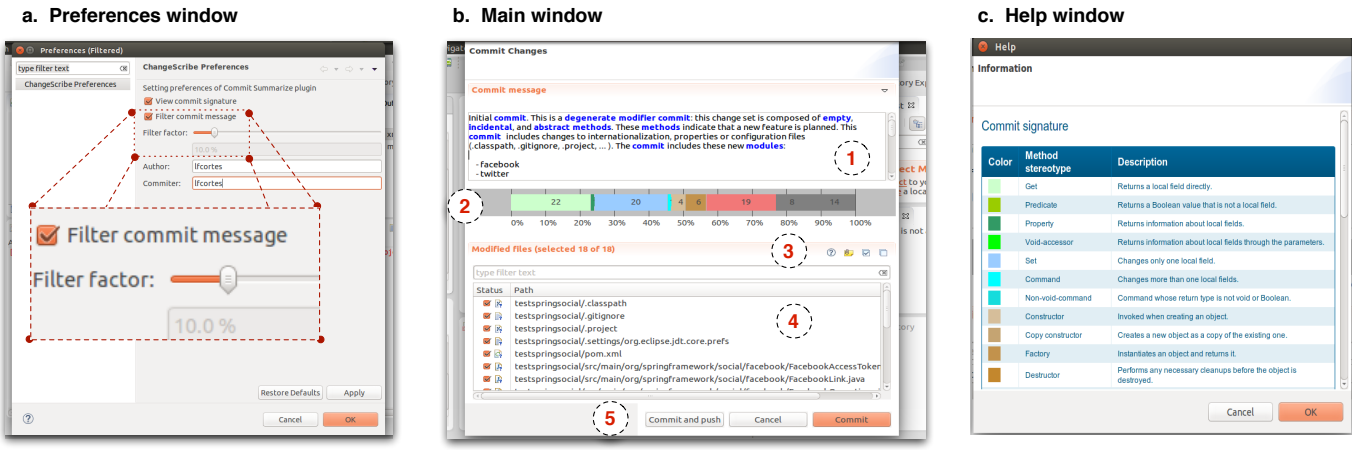

Figure 1. The three windows in the *ChangeScribe* Plugin: Preferences window (a) allows developer to set variables such as the impact threshold, and the author name; Main window (b) is displayed when a developer executes the "Commit" action in the Eclipse GUI; The help window (c) lists stereotypes (method and commit) and their descriptions.

description for each diff-based change does not contribute to understanding the rationale of a change-set. One option for controlling the length of a description is to truncate it by a number of words or characters; the truncated description is often augmented with ellipsis "..." at the end to indicate clipped text. However, truncating descriptions can break the semantic and syntax of the sentences/paragraphs in the description, and defining a gold set of the appropriate number of characters/words/lines is a daunting task.

To deal with the issues of truncating large descriptions, *ChangeScribe* uses an elegant heuristic based on impact analysis. The intuition behind the heuristic is the following: change-sets have representative classes, and by representative we mean classes with changes that have high impact on the change-set; therefore, representative classes contribute more to the description of the change-set and are more related to the rationale behind the commit. If change descriptions focus only on classes with high impact set, detailed descriptions of non-interesting classes can be removed to reduce description length. In summary, the idea is to include in the description only the descriptions of classes with high impact, and the threshold for deciding between high or low impact is provided by the code owner, i.e., the developer in charge of the commit should be able to set the threshold that distinguish representative and non-representative classes.

The impact of a class C_i in the change-set S is computed as the relative number of methods in the difference set $S - C_i$ impacted by any change in C_i. For instance, the impact value of new class C_i is the number of external methods calling a method/attribute in C_i over the total of methods in the change set; if C_i is deleted, the impact value is the number of methods modified because of C_i deletion, over the total of methods in S; or if there is any change in C_i that generates modifications in the rest of classes, the impact value is the number of methods modified in $S - C_i$ due to changes in C_i.

For each class C_i in the change set S, *ChangeScribe* computes the impact value, then, the detailed description of a class is included in the commit message if its impact-value is greater than or equal to the impact threshold defined by the software developer. The threshold is set (by demand) in the Preferences Window (See Figure 1-a).

C. Availability

More information about *ChangeScribe* can be found on our webpage[3], which contains (i) videos demonstrating its main features, (ii) link for downloading the eclipse plugin, (iii) link for downloading and Eclipse bundle with *ChangeScribe* and source code [4], (iv) architecture description, and (v) examples of commit messages for several open source applications.

D. Usage Example

The underlying approach used by *ChangeScribe* was evaluated previously [3] by 23 students and developers in an study with 50 commits of six Open Source projects (Elastic search, Retrofit, Spring social, JFreeChart, Apache Felix, Apache Solr). *ChangeScribe* is able to describe initial commits and non-initial commits, and generates messages with important information such as file renames, impact set of a change, new modules added to the system, removal of unused functionality, among others. For instance, this is an example for the first commit (http://goo.gl/5Igx1s) of Spring Social:

> *Initial commit. This is a degenerate modifier commit: this change set is composed of empty, incidental, and abstract methods. These methods indicate that a new feature is planned. This commit includes changes to internationalization, properties or configuration files (.classpath, .gitignore, .project, ...). The commit includes these new modules:*
> *- facebook*
> *- twitter* [...]

The real message is "initial commit", but *ChangeScribe*'s includes the commit stereotype and mentions the modules included in the initial commit.

Regarding impact sets, *ChangeScribe* detects when a change at method level (*i.e.*, method addition) triggers changes in

[3]http://www.cs.wm.edu/semeru/changescribe
[4]https://github.com/SEMERU-WM/ChangeScribe

other classes/methods. For instance, *ChangeScribe*'s message for a commit in JFreeChart repo (http://goo.gl/StXeJS) warns that new method in `LineUtitiles.java` triggered changes in the `RingPlot` class:

> *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.jfree.chart:*
> *1.1. Modifications to TestUtilities.java:*
> *1.1.1. Add javadoc at serialised(Object) method*
> *2. Changes to package org.jfree.chart.util:*
> *2.1. Modifications to LineUtilities.java:*
> *2.1.1. Add a functionality to extend line*
> *The added/removed methods triggered changes to RingPlot class*

ChangeScribe also describes the purpose of new classes. For example, *ChangeScribe*'s message for a commit to Retrofit is the following (http://goo.gl/mmbxzC):

> *This is a large modifier commit: this is a commit with many methods and combines multiple roles. This commit includes changes to internationalization, properties or configuration files (pom.xml). This change set is mainly composed of:*
> *1. Changes to package retrofit.converter:*
> *1.1. Add a Converter implementation for simple XML converter. It allows to: Instantiate simple XML converter with serializer; Process simple XML converter simple XML converter from body; Convert simple XML converter to body*
> *Referenced by: SimpleXMLConverterTest class*

The original message is "Add a SimpleXML converter", and *ChangeScribe*'s includes details such as the class purpose (*e.g.*, *It allows to ...*), implementation details (the class is an implementation of the *Converter* interface), and the classes referencing the new class (impact set).

E. Future Work

Current implementation of *ChangeScribe* only works with Git-based repositories, however, we will extend the plugin to work also with Subversion. *ChangeScribe* works as a plugin running on top of Eclipse, which is useful for developers. However, automatic generation of messages for large number of commits, for example when Mining Software repositories (MSR), can benefit researchers. Therefore, future work will be devoted to implement (i) a command line version and (ii) Application Programming Interface (API), which can be used for large scale studies related to MSR, program comprehension, evolution and maintenance. We want to improve the quality of the detailed descriptions by defining more templates, and detecting refactorings (the refactoring description will be part of the general description). Finally, *ChangeScribe* does not link automatically commits to issue/bug reports in a tracking system, thus, a following version will augment the commit message with information from the bug tracking system(s).

IV. Conclusion

We introduced *ChangeScribe*, a tool that implements the approach for generating commit messages via summarization of source changes, presented in our previous work [3]. The evaluation in [3] indicates that ChangeScribe can be useful as an online assistant to aid developers in writing commit messages or to automatically generate commit messages when they do not exist or their quality is low. Therefore *ChangeScribe* can assists developers when committing changes to a repository, by generating an overview of the changes and classifying/describing in detail each of the changes made by a developer in the source code. *ChangeScribe* can be also used as a tool for (re)documenting history of a system between adjacent versions, or between non-adjacent versions; this scenario is useful for evolution/maintenance tasks when no documentation is available or the quality of the commit messages is low.

V. Acknowledgements

This work is supported in part by the NSF CCF-1253837 and CCF-1218129 grants. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

References

[1] R. Buse and W. Weimer. Automatically documenting program changes. In *ASE'10*, pages 33–42, 2010.
[2] G. Canfora, L. Cerulo, and M. D. Penta. Ldiff: An enhanced line differencing tool. In *ICSE'09*, pages 595 –598, 2009.
[3] L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk. On automatically generating commit messages via summarization of source code changes. In *SCAM'14*, pages 275–284, 2014.
[4] M. D'Ambros, M. Lanza, and R. Robbes. Commit 2.0. In *Workshop on Web 2.0 for Software Engineering (Web2SE '10)*, pages 14–19, 2010.
[5] N. Dragan, M. Collard, M. Hammad, and J. Maletic. Using stereotypes to help characterize commits. In *ICSM'11*, pages 520–523, 2011.
[6] N. Dragan, M. Collard, and J. Maletic. Reverse engineering method stereotypes. In *ICSM'06*, pages 24–34, 2006.
[7] B. Fluri, M. Wursch, M. Pinzger, and H. Gall. Change distilling:tree differencing for fine-grained source code change extraction. *IEEE Transactions on Software Engineering*, 33(11):725 –743, 2007.
[8] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse. How developers drive software evolution. In *IWPSE 2005*, pages 113–122, 2005.
[9] A. Hassan. The road ahead for mining software repositories. In *Frontiers of Software Maintenance (FoSM'08)*, pages 48–57, 2008.
[10] E. Hill, L. Pollock, and K. Vijay-Shanker. Automatically capturing source code context of nl-queries for software maintenance and reuse. In *ICSE'09*, pages 232–242, 2009.
[11] D. Jackson and D. Ladd. Semantic diff: A tool for summarizing the effects of modifications. In *ICSM'94*, pages 243–252, 1994.
[12] W. Maalej and H. Happel. From work to word: How do software developers describe their work? In *MSR'09*, pages 121–130, 2009.
[13] L. Moreno, G. Bavota, M. D. Penta, R. Oliveto, A. Marcus, and G. Canfora. Automatic generation of release notes. In *FSE'14*, 2014.
[14] L. Moreno and A. Marcus. Jstereocode: automatically identifying method and class stereotypes in java code. In *ASE'12*, pages 358–361.
[15] L. Moreno, A. Marcus, L. Pollock, and K. Vijay-Shanker. Jsummarizer: An automatic generator of natural language summaries for java classes. *ICPC'13 - formal tool demonstration*, pages 230–232, 2013.
[16] G. Murphy. Attacking information overload in software development. In *VL/HCC'09*, page 4, 2009.
[17] H. A. Nguyen, T. T. Nguyen, H. V. Nguyen, and T. N. Nguyen. iDiff: Interaction-based program differencing tool. In *ASE'11*, pages 575–575.
[18] C. Parnin and C. Gorg. Improving change descriptions with change contexts. In *MSR'08*, pages 51–60, 2008.
[19] S. Rastkar and G. C. Murphy. Why did this code change? In *ICSE'13*, pages 1193–1196, 2013.