# Automated Classification of Change Messages in Open Source Projects

Ahmed E. Hassan
School of Computing
Queen's University
Kingston, Canada
ahmed@cs.queensu.ca

## ABSTRACT

Source control systems permit developers to attach a free form message to every committed change. The content of these *change messages* can support software maintenance activities. We present an automated approach to classify a change message as either a bug fix, a feature introduction, or a general maintenance change. Researchers can study the evolution of project using our classification. For example, researchers can monitor the rate of bug fixes in a project without having access to bug reporting databases like Bugzilla.

A case study using change messages from several open source projects, shows that our approach produces results similar to a manual classifications performed by professional developers. These findings are similar to ones reported by Mockus and Votta for commercial projects.

## 1. INTRODUCTION

Source control systems such as CVS [8], ClearCase [6], and Perforce [14] are used by large software projects to manage their source code [15]. As a software system evolves, changes to its code are stored in a source control repository. The repository contains detailed information about the development history of a project. The repository stores the creation date of every file, its initial content and a record of every change done to a file. A change record stores the date of the change, the name of the developer who performed the change, the numbers of lines that were changed, the actual lines of code that were added or removed, and a *change message* entered by the developer usually explaining the reasons for the change.

Such *change messages* could be used by researchers to build tools or to study approaches for assisting in the maintenance of long lived software systems. Chen *et al*. presented a case study of a source code searching tool that makes use of these change messages [4]. The tool uses the messages as meta data that is attached to a changed line and which can be searched when developers are looking for the implementation location of specific features.

Mockus and Votta presented a lexical analysis approach to classify a change based on the content of its change message [13]. Such classifications can be used to monitor the evolution of a software project and to gauge its reliability. Using the classification, a manager could, for example, determine the rate of bug fix changes versus feature introduction changes for a project. A high rate indicates that too much effort is spent fixing bugs and may be sign that the quality of the produced code must be improved. A manager can then consider enforcing quality control practices like code reviews and unit tests. These practices could reduce the rate of bug fixes in the future.

The aforementioned approaches demonstrate the value of change messages in understanding and monitoring software projects. Unfortunately, change messages are free form. Their quality and content is not enforced in traditional source control systems. For example, developers do not need to enter any change message and do not need to specify the purpose of a change in the entered message. Mockus and Votta proposed an automatic classifier to classify change messages in a large commercial telephony project [13]. Their results show that 61% of the time, their automatic classifier and the developer, who performed the change, agree on the classification of a change.

With the widespread of open source projects, researchers have used their repositories instead of relying on the repositories from commercial projects which are harder to acquire. Work by Chen *et al*. [4] was conducted on open source systems; whereas work by Mockus and Votta [13] was conducted on commercial telecom projects. We are concerned with the feasibility of applying an approach similar to Mockus and Votta's approach on open source software projects. It is not obvious if open source developers would input sufficient information in a change message for an automated classification approach to work. It may be the case that Mockus and Votta's approach worked well for commercial projects due to the extensive software development processes followed by commercial telecom projects. Recent work cautioned of the quality and reliability of open source change logs [5]. Change logs are usually stored in a single file called the `ChangeLog` file. They provide a high level overview of changes that occurred throughout the lifetime of a software project. Whereas change logs summarize the purpose of changes to the source code and are likely to omit a large number of details, change messages derived from source control repositories record the specifics of every change to the code.


*SAC'08* March 16-20, 2008, Fortaleza, Ceará, Brazil

## 1.1 Organization of Paper

This paper is organized as follows. In Section 2, we discuss the logistics of our case study. In Section 3, we discuss the results of our study of an automated classification approach. Our approach classifies changes by lexically examining the change message. Finally in Section 4, we summarize our findings.

# 2. CASE STUDY

We now introduce the goals of our study and present our study participants.

## 2.1 Study Goals

We want to determine if an automatic classification of a change message would agree with a manual classification performed by industrial developers. For many software projects, source code repositories are the only source of historical records about the project. Bug reports are commonly not archived, making it impossible to perform any analysis on the reliability of many open source projects. To study the reliability of a project, we can use the source code repositories to recover information about the occurrences of bugs (*e.g.*, [11]). We can use a lexical based approach, similar to [13], to classify change records into three types based on the content of the change message. The classification approach would determine if a change was done to fix a bug, to add features, or to perform general maintenance activities such as updating copyright notices or indenting the source code. Mockus and Votta have used such a lexical approach successfully on commercial systems. However, it is not clear if change messages in open source projects would benefit from the same lexical analysis approach.

## 2.2 Study Participants

To perform our study, we picked a small number of participants which are accessible to us so we can easily interview them to clarify their replies if needed. We asked six software developers to participate in the study. The developers worked in companies in the following software domains: security, telecommunication, graphics, and databases. The developers were chosen so they would represent two groups: an intermediate and a senior group. We hoped that this grouping would uncover if there are any noticeable variations in the study that may be attributed to experience or managerial differences between both groups of developers:

- The first group consists of 3 intermediate software developers with at least 7 years of software development with no experience of managing other software developers.
- The second group consists of 3 experienced software developers with at least 9 years of experience developing industrial software systems and who have previously managed or are currently managing other software developers.

Table 1 gives background information about the participants in our study. At the time of the study, the developers worked at a five different companies. All the developers had used source control systems for most of their professional career.

A list of 18 change messages from several open source projects were presented to every developer. Every developer was asked to allocate 10 points to four categories. Three categories represented the possible purpose of a change: bug fixing, feature introduction, and general maintenance. A fourth category was "Not Sure" – Developers were asked to use this category when the change message did not have sufficient information for them to confidently classify the change into one of the other three categories. We limited the number of change messages in the study to 18 messages so that the professional developers would finish the classification in a timely and accurate fashion without interfering with their busy schedules.

| Dev. # | Development Experience (years) | Source Control Experience (years) | Avg. Team Size | Team Lead |
|---|---|---|---|---|
| I1 | 7 | 5 | 5 | No |
| I2 | 7 | 7 | 5 | No |
| I3 | 7 | 7 | 30 | No |
| S1 | 9 | 5 | 5 | Yes |
| S2 | 15 | 12 | 8 | Yes |
| S3 | 9 | 9 | 5 | Yes |

**Table 1:** Characteristics of the Participants of the Study

| Application Name | Application Type | Start Date | Programming Language |
|---|---|---|---|
| NetBSD | OS | March 1993 | C |
| FreeBSD | OS | June 1993 | C |
| OpenBSD | OS | Oct 1995 | C |
| Postgres | DBMS | July 1996 | C |
| KDE | Windowing System | April 1997 | C++ |
| Koffice | Productivity Suite | April 1998 | C++ |

**Table 2:** Summary of the Studied Systems

The 18 change messages were selected from the repositories of six large open source systems (NetBSD, FreeBSD, OpenBSD, Postgres, KDE, Koffice - Table 2 lists details of these projects). Every change message in these repositories is already classified as either a bug fixing, feature introduction or general maintenance change using an automatic classifier described in Section 3. We randomly picked 18 changes from the repository of every project: 6 bug fixing, 6 feature introduction, and 6 general maintenance changes (for a total of 108 changes). We then randomly chose half of these changes (54 changes) and broke them into three disjoint sets of 18 changes. We followed this selection procedure to guarantee that the mathematical analysis performed later does not suffer from any bias due to the type of change messages or their sources. Every set was classified by a member of the intermediate group and a member of the senior group. Each group classified the three sets of changes. No two developers in the same group classified the same set of changes.

# 3. STUDY RESULTS

We developed an automated classifier program that reads every change message and classifies its change record as one of the following three types:

**Bug Fixing change (BF):** These are the changes which are done to fix a bug. Our automatic classifier labels all changes which contain terms such as *bug*, *fix*, or *repair* in the change message as BF change.

**General Maintenance change (GM):** These are changes that are mainly bookkeeping changes and do not reflect the implementation of a particular feature. Examples of such changes are updates to the copyright notice at the top of source files, re-indentation of the source code by means of a code beautifier (pretty-printer). Our automatic classifier labels all changes which contain terms such as *copyright/update*, *pretty/print*, or *indent* in the change message as GM changes.

**Feature Introduction changes (FI):** These are the changes that are done to add or to enhance features. Our automatic classifier labels all changes that are not FR or GM changes as FI changes.

Relative to the Swanson's classical classification of changes [9]:

- BF changes correspond to corrective maintenance which is performed to correct defects.
- FI changes correspond to adaptive and perfective maintenance. Adaptive maintenance corresponds to functional enhancements of a software system. Adaptive maintenance corresponds to non-functional enhancements.
- GM changes do not correspond to any of the change classifications proposed by Swanson. GM changes do not correspond to features instead they are concerned with source code hygiene issues.

Each participating developer was shown the message associated with a change and asked to allocate a total of 10 points to four categories. Three of the categories mirrored the automated classification categories (BF, GM, and FI). A fourth category was "Not Sure" (NS). Developers were asked to use the NS category when the change message did not have sufficient information for them to confidently classify the change into one of the other three categories. For the senior developer group only one out of 54 changes was ranked as NS. For the intermediate developer group, three out of 54 changes were ranked as such. For our analysis, we considered changes classified as NS to be FI changes. The automatic classifier uses FI as the default classification when it cannot determine the type of a change; therefore, we chose to use the same default rule for the manual classification done by developers.

Developers had the option to allocate points between different categories but our automatic classifier only assigns a single category to a change. We chose to classify a manual change based on the highest ranked category, so we can compare manual classifications to the automated ones. When there were ties, we used the following tie breaking priority order: BF, GM, then FI. For example, if a developer allocates 5 points to the BF category and 5 points to the FI category, we would consider this changes to a be an BF change. This tie breaking order was used for only two classified changes. This order rule was followed as it is the same rule followed by the automatic classifier. The automatic classifier tends to be more pessimistic in classifying changes by ensuring that changes that may be a combinations of fault repairing and feature introduction are considered as fault repairing changes to get a more complete count of repaired faults in a software system.

The two groups of developers were given the same 54 change messages to classify. Every developer in a group was given a disjoint set of 18 messages to classify. We then combined the classification by every developer to arrive to a classification for the whole group (Intermediate and Senior classifications). The same 54 change messages were classified using our classifier program. We performed two types of analysis:

- In the first analysis we compared the intermediate group classification to the automatic classifier (Analysis 1A) and the senior group classification to the automatic classifier (Analysis 1B).
- In the second analysis (Analysis 2), we combined the classification done by the senior and intermediate groups to create a common classification. We then compared this common classification to the classification done by the automatic classifier.

We now present the results of the two types of analysis.

## 3.1 Analysis 1A and 1B of Developers' Classifications

| | Automatic Classifier | | | |
|---|---|---|---|---|
| Manual Classifier | GM | BF | FI | Total |
| GM | 15 | 2 | 3 | 20 |
| BF | 4 | 14 | 7 | 25 |
| FI | 0 | 0 | 9 | 9 |
| Total | 19 | 16 | 19 | 54 |

**Table 3:** Classification Results for the Intermediate Developers Group vs. the Classifier Program (Analysis 1A)

Table 3 and 4 summarize the results for analysis 1A and 1B. The last row in both tables shows the distribution of change types as classified by the automatic classifier. The automatic classifier categorized the 54 changes into 19 GM, 16 BF, and 9 FI changes. The last column of both tables shows the results of the manual classification which differs between the two groups of software developers. Table 4 shows that our automatic classifier has classified 16 changes as BF changes. By comparison column 2 of Table 4 shows that the senior developers have classified 15 out of these 16 changes as FI and one of the changes as GM.

| | Automatic Classifier | | | |
|---|---|---|---|---|
| Manual Classifier | GM | BF | FI | Total |
| GM | 15 | 1 | 4 | 20 |
| BF | 3 | 15 | 4 | 22 |
| FI | 1 | 0 | 11 | 12 |
| Total | 19 | 16 | 19 | 54 |

**Table 4:** Classification Results for the Senior Developers Group vs. the Automatic Classifier (Analysis 1B)

The diagonal of both tables lists the number of times the developers and the automatic classifier agreed on their classifications. Summing the diagonal values in both tables shows that:

- For Table 3 the intermediate developers agreed 38 (15 + 14 + 9) times with the automatic classifier. The intermediate group agreed ($\frac{38}{54} = 70$)% of the time with the automatic classifier.
- For Table 4, the senior developers agreed 41 (15 + 15 + 11) times with the automatic classifier. The senior group agreed ($\frac{41}{54} = 76$)% of the time with the automatic classifier.

We calculated Cohen's Kappa ($\kappa$) coefficient for both groups of developers [7]. The Kappa coefficient is a widely adopted technique to measure the degree of agreement between two raters, in our case: the automatic classifier and the developers participating in our experiment. The Kappa for the senior group and the automatic classifier is 0.64. The Kappa for the intermediate group and the automatic classifier is 0.56. According to the Kappa thresholds values proposed by El Emam [10] (see Table 5), the agreement between the automatic classifier and the group of senior developers is *substantial*. The agreement between the automated classification and the group of intermediate developers is high *moderate*. These results are similar to the ones reported by Mockus and Votta who found *moderate* agreement between an automated classification and a manual classification using the El Emam classification. In brief, the results indicate that an automated classification approach is likely to achieve similar classifications to ones done manually by professional software developers.

| Kappa Value | Strength of Agreement |
|---|---|
| $< 0.45$ | Poor |
| $0.45 - 0.62$ | Moderate |
| $0.63 - 0.78$ | Substantial |
| $> 0.78$ | Excellent |

**Table 5:** Kappa Values and Strength of Agreement

## 3.2 Analysis 2 of Developers' Classifications

| | Intermediate Classifier | | | |
|---|---|---|---|---|
| Senior Classifier | GM | BF | FI | Total |
| GM | 17 | 2 | 1 | 20 |
| BF | 2 | 19 | 1 | 22 |
| FI | 1 | 4 | 7 | 12 |
| Total | 20 | 25 | 9 | 54 |

**Table 6:** Classification Results for the Senior Developers Group vs. the Intermediate Developers Group

For the second analysis, we combined the classifications done by both the senior and intermediate developer groups to create a common classification. We removed change messages, which both intermediate and senior developers disagreed in their classification, from the common classification. We felt that since both human classifiers could not agree on the classification of a message, then we should not expect an automatic classifier to determine the correct classification of that message. Table 6 summarizes the classification results for the senior and intermediate developers. Out of 54 change messages, the senior and intermediate developers disagreed on the classification of 11 change messages. The Table indicates an 80% overall agreement between both developer groups and a Kappa of 0.68, corresponding to a *substantial* agreement. A closer look at the degree of agreement between classifiers for each change type reveals that there is an 85% agreement for GM changes, 81% agreement for BF changes, and 68% agreement for FI changes. In short, developers tend to agree more on classifying GM or BF changes, than on classifying FI changes. This is likely due to developers using specific keywords to classify GM and BF messages like "*indent*", "*bug*", or "*fix*".

We used the agreed on classifications to create a common classification for the remaining 43 change messages. We compared the common and the automatic classification (see Table 7). The Kappa for the common classification is 0.71. Using the Kappa thresholds values shown in Table 5, we note that the agreement between the automated classification and the common classification is *substantial*. The table indicates that the automated classification and the common classification agree 81% of the time.

| | Automatic Classifier | | | |
|---|---|---|---|---|
| Manual Classifier | GM | BF | FI | Total |
| GM | 14 | 1 | 2 | 17 |
| BF | 2 | 14 | 3 | 16 |
| FI | 0 | 0 | 7 | 7 |
| Total | 14 | 15 | 12 | 43 |

**Table 7:** Classification Results for the Common Classifications vs. the Automatic Classifier (Analysis 2)

In addition to performing the Kappa analysis on the classifications, we performed a Stuart-Maxwell Test. While Kappa examines the agreement between classifiers, the Stuart-Maxwell Test examines the disagreement between classifiers. In particular, the Stuart-Maxwell Test measures the marginal homogeneity for all classification categories [1, 2, 3, 12]. One reason classifiers disagree is because of different tendencies to use classification categories. For example, a developer may tend to always classify changes as bug fixes. The Stuart-Maxwell Test determines if classifiers have biases towards specific classification categories or if they do not. A small probability value $P$ implies that there is an association between both classifiers and that no bias exists. Table 8 summarizes the results for the Stuart-Maxwell Test for the classification tables. The Stuart-Maxwell Test holds for all classification tables at above 90%. These Stuart-Maxwell Test results show that there is no bias and they agree with the Kappa analysis performed above.

| | Maxwell Test | |
|---|---|---|
| Classification Table | Chi-Squared | $P$ |
| Intermediate vs. Automated (Table 3) | 10.494 | 0.0053 |
| Senior vs. Automated (Table 4) | 6.786 | 0.0336 |
| Common vs. Automated (Table 7) | 5.238 | 0.0729 |

**Table 8:** Results of the Stuart-Maxwell Test

The results of analysis 1A, 1B, and 2 indicate that an automated classification approach for change records for open source projects is likely to produce results that are substantially similar to classifications done manually by professional developers. These results are encouraging as they permit us to automatically recover a historical overview of the bug fixes applied to an open source system. These bug

fix changes could be used, for example, to study the quality of open source systems and to analyze the benefit of adopting different techniques to improve the quality of software systems in general [11, 13].

## 4. CONCLUSION

In this paper, we investigated an artifact of software development that is rarely studied; namely, the change messages attached to every change committed to a source control system. We investigated the possibility of classifying changes automatically into bug fixing, bookkeeping and feature introduction changes. Our results indicate that automated classifications agree over 70% of the time with manual classifications by practitioners. Although our study involved a small number of developers, we believe that their classifications are representative of developers in Industry, since they worked at different companies spanning various domains and they have several years of industrial experience. Nevertheless, it is desirable to investigate that our findings hold using a larger number of participants and for additional projects.

Researchers should investigate techniques and approaches to improve the quality of the change messages and to make them more accessible for developers as they evolve software systems.

## 5. REFERENCES


[1] S. AA. A test for Homogeneity of the Marginal Distributions in a Two-way Classification. *Biometrika*, 42:412–416, 1955.

[2] M. AE. Comparing the Classification of Subjects by Two Independent Judges. *British Journal of Psychiatry*, 116:651–655, 1970.

[3] E. BS. *The Analysis of Contingency Tables*. Chapman and Hall, London, 1977.

[4] A. Chen, E. Chou, J. Wong, A. Y. Yao, Q. Zhang, S. Zhang, and A. Michail. CVSSearch: Searching through source code using CVS comments. In *Proceedings of the 17th International Conference on Software Maintenance*, pages 364–374, Florence, Italy, 2001.

[5] K. Chen, S. R. Schach, L. Yu, J. Offutt, and G. Z. Heller. Open-Source Change Logs. *Empirical Software Engineering*, 9(197):210, 2004.

[6] Rational ClearCase - Product Overview. Available online at `http://www-306.ibm.com/software/awdtools/clearcase/`.

[7] J. Cohen. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurements*, pages 37–46, Dec. 1960.

[8] CVS - Concurrent Versions System. Available online at `http://www.cvshome.org`.

[9] E. B. Swanson. The Dimensions of Maintenance. In *Proceedings of the 2nd International Conference on Software Engineering*, pages 492–497, San Francisco, California, Oct. 1976.

[10] K. E. Emam. Benchmarking Kappa: Interrater Agreement in Software Process Assessments. *Empirical Software Engineering*, 4(2):113–133, Dec. 1999.

[11] A. E. Hassan and R. C. Holt. Studying The Chaos of Code Development. In *Proceedings of the 10th Working Conference on Reverse Engineering*, Victoria, British Columbia, Canada, Nov. 2003.

[12] Tests of Marginal Homogeneity. Available online at `http://ourworld.compuserve.com/homepages/jsuebersax/margin.htm`.

[13] A. Mockus and L. G. Votta. Identifying reasons for software change using historic databases. In *Proceedings of the 16th International Conference on Software Maintenance*, pages 120–130, San Jose, California, Oct. 2000.

[14] Perforce - The Fastest Software Configuration Management System. Available online at `http://www.perforce.com`.

[15] W. F. Tichy. RCS - a system for version control. *Software - Practice and Experience*, 15(7):637–654, 1985.


## APPENDIX

In this appendix we show the questionnaire given to the participants in our study.

The following are 18 actual change messages, which have been randomly picked from several large software projects. For every change, please classify the change message as

**BF:** A bug fix.

**FE:** A feature enhancement/addition

**BK:** A bookkeeping change such as merging of source control branches, updating copyright dates, indentations, spelling corrections, etc.

**NS:** Not sure. The change message does not give enough details to classify the change.

Please allocate 10 points among the 4 classes (BF, FE, BK and NS).
For example, if you feel confident that a change is a bug fix then assign all 10 points to BF. If you feel a change is likely a bug fix and a feature enhancement then you could assign 5 points for BF and 5 points for FE. If you are not sure how to classify the message then assign all 10 points to NS. For example:

0. "fix error in hyperzont.c"

BF. _10_ FE.____ BK.____ NS.____

Here are the change messages that you are to classify.

[*Personalized Generated List of Change Messages for Every Participant*]

### Using the Data from this survey

Can we acknowledge you when we report these results? (yes/no, I would like to remain anonymous) ___