Automated Source Code Changes Classification for Effective Code Review and Analysis

Evgeny G. Knyazev

***Abstract*—Software development process is a complex sequence of actions having source code of working system as a result. All project participants should track changes in source code during work process to know what's happening. However to make «manual» code review everyone should have corresponding technical skills and a lot of time to spend.**

This work describes usage of automated source code changes classification aimed to control source code evolution. The method bases on statistical clusterization of change metrics. In this work we show usage of automatic classification of changes to optimize code review and code change control on final development stages. Development process report building is also shown.

***Index Terms*— Programming, Project management, Software metrics, Code Changes Classification**

I. Introduction

The most important software development project's asset is its source code. Almost all modern software projects keep all history of source code changes in special repository of source code versioning system. Unfortunately this information is available only for that project participants, who has been trained for source code analysis, i.e. mostly for developers. While testers, managers and other specialists are also work on project and are also interested in information, retrieved from source code in the form of functionality lists for concrete version, different kinds of reports, etc. Moreover source code change history analysis is hard because of a lot of incoming information. For example, source code versioning system repository of real project contains lots of small, insignificant changes, making analyzer's task complicate.

Automated classification of changes, as an additional tool, helps to increase source code history analyzer's performance. For example usage of automated source code change classification helps to filter out unimportant changes for analyzer. Developer or development team leader can find changes, lead to new functionality and focus on them.

Using automated change classification team leader can automate restriction of particular classes of changes on the defined development stages. For example he can set up automated classification tool to notify him when new functionality was added on the final testing stage, what should be usually prohibited.

This work also provides several use cases of automated source code changes classification for that project participants, who are not directly work with source code. Automated changes classification gives testers an opportunity to get information about changes, in which new functionality was added, bugs fixed in the form of source code or comments, provided with changes by its developer. Project manager can build reports with change distribution on classes.

So usage of automated source code changes classification leads to increase of speed and quality of code review. Also it provides additional mechanisms to control development project state.

Method of automated source code changes classification, described in this work, bases on source code change metrics clasterization using *k-means algorithm by MacQueen [1]*. Classification adequacy proved on the experiment, provided in [1]. Coefficient of agreement *Kappa* [3] there was equal to *0.79*. This value is on the border between significant and excellent agreement rate of expert and automated classification methods.

II. Automated change classification use cases

Automated source code changes classification can be useful for all software development project members. Use cases of automated source code changes classification provided below.

A. Usage of changes classification by developer

Common software developer often faces to the need of reviewing lots of source code changes. It takes place, for example, when he's starting to work on a project with existent development history or just after vacation. In such cases he has to read every change comment carefully and if there is not enough information, than look through change contents. This process can be very time expensive.

Automated change classification will relieve him from need to dive in every change details. It will be enough to choose change classes interesting to him and look through changes, belonging to the specified classes. Figure 1 shows scheme of code changes review with selected change class

Manuscript received April 1, 2008.

E. G. Knyazev is with the Saint-Petersburg State University of Information Technologies, Mechanics and Optics (phone: +7-911-251-7636; fax: +7-812-325-3132; e-mail: evgeny.knyazev@gmail.com).

filter.

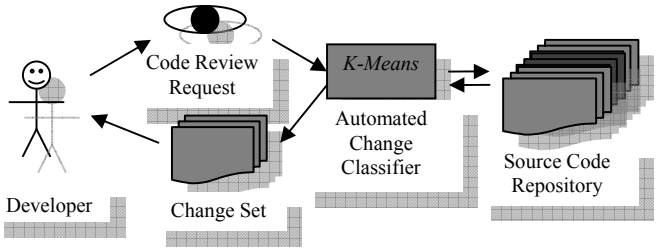

Fig. 1. Code change review with filtering by class.

Automated classification of code changes can help developer to localize errors, inserted in source code in some time period. In this case developer should extract classes of changes, potentially affecting chosen module and find the change, which break it capacity for work.

B. Usage of changes classification by development team leader

Good development team leader performs regular code reviews and control all changes at the current development stage. Code review is a very useful practice. It is source code look-through trying to find errors and style, design and other problems. This practice can help to discover and fix a lot of problems on early development stages, while these fixes don't become very time expensive. Change control on current development stage is to reject changes, potentially able to destabilize system on important development stages. That's why, for example, new feature implementation on the final development phases is inadmissible.

Let's examine code review task. One of common actions to keep code quality high is permanently review changes, made by developers. Average developers team generates a lot of code changes, which can lead to physical inability of team leader to review all changes. Table 1 provides data about number of changes in different projects for the same period of time. In some projects number of changes can be extremely high.

In table 1 you can see change count for period about one month for three projects: GUI for Subversion *TortoiseSVN [4]*, client-server application for fleet monitoring *Navi-Manager* [5], developed by author of this work in *Transas Technologies* company and window system for Linux and Unix *KDE* [6].

TABLE I
SOURCE CODE CHANGE NUMBER BY PROJECT DURING ONE MONTH OF DEVELOPMENT

Project	Tortoise SVN	Navi-Manager	KDE
Time period (~ 1 month) Number of changes	Sept 22, 2007 – Oct 22, 2007 215	Sept 22, 2007 – Oct 22, 2007 72	Sept 17, 2007 – Oct 14, 2007 11841(!)

During review of big number of changes reviewer choose only the most important changes for review basing on the text of change comments. However, choosing changes only by comments, provided by change author, may lead to misses of some important changes with non-clear or inadequate comments. This is the way to miss control of product quality.

Solution of this problem is in usage of automated source code changes classification. Code changes review with usage of additional information about change class gives ability to filter out changes that are not interesting for team leader for more precise study of important changes.

Let's examine task of changes control on current development stage. During its implementation software product has several stages. For example release preparation stage called *stop code* allows any bug fixes. This stage is needed to stabilize product before its release.

When all found bugs were fixed, stage *freeze code* is declared, when only critical bug fixes can occur. For stability control it needs to review each change by at least one team member except change author. This state lasts for installed product version while it is being supported.

Every development stage limits development process with some restrictions. For example during *stop code* and *freeze code* stages developers should fix bugs with no new features implementation.

Automated code changes classification can be used to automate process of control change classes on current development stage. To do this it is enough to provide information about available on current stage change classes and automated classifier will do the rest. Fig. 2 shows scheme of module, controlling change classes on current development stage.

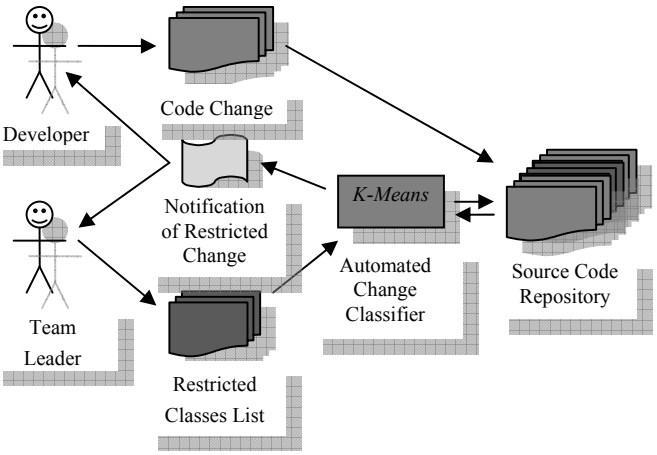

Fig. 2. Restricted changes search with automated change classifier.

C. Usage of changes classification by white-box testing team

In this section we will discuss only white-box testing (for example, unit testing). Usage automated source code changes classification for black-box testing (for example, acceptance testing) is not yet appropriate.

During their work testers communicate to developers to understand project state more clearly. Testers often have not enough information about new functionality and bugs fixed in concrete product version. Sometimes they have only one way

to get precise list of new features and bug fixes in concrete version. This way is to ask developer to look through all code changes starting from time when previous version was shipped to the time of version of interest. Usage of automated source code change classification dramatically decreases time of such request execution by filtering out all change classes except new functionality and bug fix.

D. Usage of changes classification by project manager

Project manager is interested in hi-level development process parameters. Information about what part of changes was made for new functionality implementation, comparing to refactoring and bug fixes will help to measure work effectiveness. Fig. 3 shows change distribution by classes for *Navi-Manager* project during one month of new functionality implementation stage. Looking on fig. 3 one can conclude that *Navi-Manager* project has not enough progress in new features implementation because of main developer forces where focused on bug fixes, not on new features implementation.

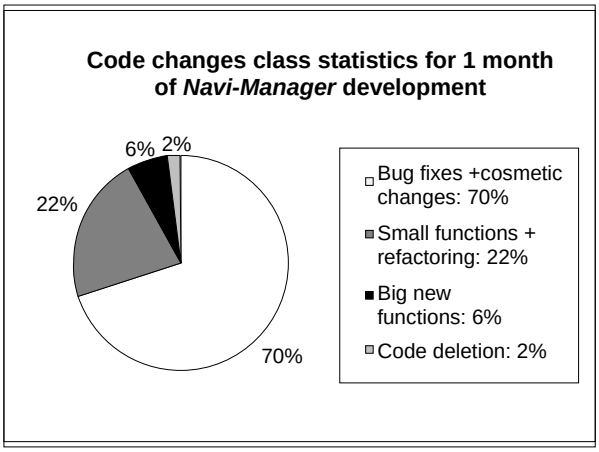

Fig. 3. Code changes distribution by classes.

III. CODE CHANGES CLASSIFICATION

In this work it is suggested to use source code changes classification method, allows to automate separation of semantically different changes basing on values of metrics of source code. For example source code change can belong to one of the following classes: new functionality implementation, refactoring, bug fix, cosmetic change. There are several methods of source code change classification. They can be divided into following groups [7]:

– *informal methods* – such methods as automatic change classification by comment analysis [8][9], refactoring detection method [10];

– *syntax methods* – such methods as heuristic comparison of syntax trees of code versions [11] and version difference analysis with code tags [12].

Automated method of source code change classification [2], described in this work, bases on clasterizaion of metrics values of source code using method *k-means MacQueen* [1]. Result of this method work is set of source code changes divided on predefined number of clusters. Each cluster corresponds to particular class of changes.

A. Change classification task formalization

We define here source code change as mapping δ_r, transforming state of source code S_{r-1} to state S_r. Let $C=\{c_i\}$ is a set of source code change classes, defined by an expert. Expert manually classifies any set of changes, providing for each change δ_r appropriate class c_i. Although this process is hard and time expensive, we'll try to automate it, using expert's knowledge only once at method study stage.

Here we suggest an algorithm of automated changes classification (*expert* – a human expert, *tool* – an automatic tool for changes classification):

1. An expert chooses *learning set of changes* – some subset of full set of changes.
2. A tool performs *change metrics sets calculation*, and *clustering of change metrics sets* of learning set of changes.
3. An expert builds *cluster interpretation* – assigns expert class for every cluster built on learning set of changes.
4. After these steps a tool can automatically classify any set of changes (within the same project, from witch learning set was extracted) by performing step (2) of this algorithm and than using cluster interpretation, built on step (3) to automatically assign every changes from each cluster appropriate class.

Automated classification function I_A, transforms set of source code changes $\{\delta_r\}$ to the set of their classes $\{c_i\}$:

$$I_A : \{\delta_r\} \xrightarrow{I_A} \{c_1, c_2, ..., c_n\}.$$

Here we treat function I_A as composition of clasterization function I_Q and cluster interpretation function I_{IQ}:

$$I_A = I_Q \circ I_{IQ},\ I_Q : \{\delta_r\} \xrightarrow{I_Q} \{q_1, q_2, ..., q_m\},$$

$$I_{IQ} : \{q_1, q_2, ..., q_m\} \xrightarrow{I_{IQ}} \{c_1, c_2, ..., c_n\},$$

where $Q=\{q_j\}$ is a set of clusters q_j.

Clustering function I_Q transforms set of changes $\{\delta_r\}$ to the set of clusters $\{q_j\}$. Cluster interpretation function I_{IQ} transforms set of clusters $\{q_j\}$ to the set of change classes $\{c_i\}$. This is two-step process: first split changes by clusters, than interpret all changes from each cluster as a change class.

TABLE 2
SOURCE CODE CHANGE METRICS USED FOR CHANGE CLUSTERING

Metric Symbol	Metric Name	Change Metric Effect and Description
eLOC	Effective Lines of Code	Number of lines of code without empty lines and comments
CC	Cyclomatic Complexity	Number of linearly independent execution paths [13]
CS	Classes / Structures	Number of classes or structures

Clustering function can be built using *MacQueen k-means algorithm* [1]. Clustering algorithm groups code changes in similarity clusters. Similarity here is proximity between change metric sets. Clustering algorithm treats each change metrics vector $<M`\delta_r>_n = <M_1`\delta_r, M_2`\delta_r ... M_n`\delta_r>$ as point in

n-dimensional space and splits these points on predefined number of clusters basing on proximity between points.

Metric $M`$ of the change δ_r may be defined as difference between source code metric M values of changed code S_{r+1} and original code S_r:

$$M`\delta_r = MS_{r+1} - MS_r.$$

In this work we used set of change metrics, based on three source code metrics defined in table 2.

B. *Clusters interpretation*

Choosing learning set of changes, *k–means* clusterization method parameter *number of clusters* and building cluster interpretation function I_{IQ} are expert tasks in current research. Interpretation function can be built by choosing several changes from each cluster and expert classification of these changes to make a conclusion about what class of changes represents this cluster. This task is significantly less time-expensive than original classification task because there are not so many classes on practice to be extracted.

During interpretation function building expert analyses some source code changes of each cluster and changes comments. As a result expert defines class c_i, which is appropriate for analyzed cluster q_j. When there's no way to classify one cluster unambiguously than it needs to choose other set of metrics and/or set of classes.

After function I_{IQ} has been built for some set of changes, classification of other changes of the same project can be performed in automatic mode without an expert.

IV. Conclusion

Described method can be used by participants of almost any software development project. A tool was developed to support automation of code changes classification. On the moment of the publication it supports only one version control system *Subversion* and programming languages *C++, C#*. This tool allows calculating changes metrics based on *cyclomatic complexity* [13], *effective lines of code, and common number of classes or structures*.

Code changes classification experiment was set up in [2], and *Kohen's* agreement rate [3] between human expert and automated classifier was measured: $\kappa=0.79$. This value shows agreement strength of changes classification method based on metrics clustering and human expert between significant and excellent.

Described method faces with mixed changes problem, consisting of several changes with different nature. Change classification method not always can correctly classify such changes. But probably complex non-clear changes should be avoided in good development process. When exist, these changes make code review and other work with history harder.

Problem of mixed changes separation during clustering is to be solved in future research. Other problem left for future research is clustering stability problem in long-term analysis.

Changes classification method, based on change metrics have several advantages, comparing to other change classification methods:

- *Objectiveness*: Analysis is performed on source code itself, not on change comments, as, for example in automatic change classification by comment analysis [8], [9].
- *Ability of tuning:* Different sets of metrics can be chosen for classification automation, depending on target classification [2]. Other methods allow change classification only by given set of classes, although in some of their [8],[9],[11] not very big amount of additional work can be performed to add new classes of changes.
- *Adaptivity:* Resulting number of clusters and expert classification of learning changes set passed to clustering method. With this data project specific classes of changes can be extracted thanks to data-mining techniques (*McQueen clustering*) used in suggested in this article method. Other methods of changes classification, described here, possibly except [8],[9], cannot be simply adapted to specific project.
- *Formality:* Change classification bases on formal statistic methods, while some informal methods, bases on heuristics of special words usage (as "bug", "fix", "refactor", etc) in comments [8],[9] or metrics values changes by specific rules [10].

References

[1] J. B. MacQueen: "Some Methods for classification and Analysis of Multivariate Observations", Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press, 1967, vol.1 pp. 281-297

[2] E. G. Knyazev, D. G. Shopyrin, "Automated classification of source code changes by means of multidimensional statistical analysis", *Information Technologies*, to be published in Russian.

[3] J. Cohen, "A Coefficient of Agreement for Nominal Scales", *Educational and Psychological Measurement*, 1960, pp. 37–46.

[4] TortoiseSVN, A Subversion client, implemented as a windows shell extension. Available: http://tortoisesvn.tigris.org

[5] Navi-Manager Vessel Monitoring System. Available: http://www.transas.com/products/shorebased/manager/ http://www.transas.ru/products/shorebased/fleet/navi-manager/

[6] KDE. A powerful Free Software graphical desktop environment for Linux and Unix workstations. Available: http://www.kde.org

[7] H. Kagdi, M. Collard, J. Maletic, "Towards a Taxonomy of Approaches for Mining of Source Code Repositories", *ACM SIGSOFT Software Engineering Notes. Proceedings of the 2005 International Workshop on Mining Software Repositories MSR '05*, St. Louis, Missouri. 2005, pp. 1–5.

[8] A. E. Hassan, R. C. Holt, "Source Control Change Messages: How Are They Used And What Do They Mean?", 2004. Available: http://www.ece.uvic.ca/~ahmed/home/pubs/CVSSurvey.pdf

[9] A. Mockus, L. G. Votta, "Identifying reasons for software change using historic databases", *Proceedings of the International Conference on Software Maintenance (ICSM)*, San Jose, California. 2000, pp. 120–130.

[10] S. Demeyer, S. Ducasse, O. Nierstrasz, "Finding refactorings via change metrics", *Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '00)*, 2000, pp. 166–178.

[11] S. Raghavan, R. Rohana, A. Podgurski, V. Augustine, "Dex: A Semantic-Graph Differencing Tool for Studying Changes in Large Code Bases", *Proceedings of 20th IEEE International Conference on Software Maintenance (ICSM'04)*, Chicago, Illinois. 2004, pp. 188–197.

[12] J. I. Maletic, M. L. Collard, “Supporting Source Code Difference Analysis”, *Proceedings of IEEE International Conference on Software Maintenance (ICSM'04)*. Chicago, Illinois. 2004, pp. 210–219.
[13] T. J. McCabe, “A Complexity Measure”, IEEE Trans SE-2. 1976. №4.