Kataribe: A Hosting Service of Historage Repositories

Kenji Fujiwara, Hideaki Hata, Erina Makihara, Yusuke Fujihara, Naoki Nakayama, Hajimu Iida, and Kenichi Matsumoto
Graduate School of Information Science
Nara Institute of Science of Technology, Nara, Japan
{kenji-f, hata, makihara.erina.lx0, yusuke-fuj, nakayama.naoki.my7, matsumoto}@is.naist.jp, iida@itc.naist.jp

ABSTRACT

In the research of Mining Software Repositories, code repository is one of the core source since it contains the product of software development. Code repository stores the versions of files, and makes it possible to browse the histories of files, such as modification dates, authors, messages, etc. Although such rich information of file histories is easily available, extracting the histories of methods, which are elements of source code files, is not easy from general code repositories. To tackle this difficulty, we have developed `Historage`, a fine-grained version control system. Historage repository is a Git repository which is built upon original Git repository. Therefore, similar mining techniques for general Git repositories are applicable to Historage repositories. `Kataribe` is a hosting service of Historage repositories, which enables researchers and developers to browse method histories on the web and clone Historage repositories to local. The Kataribe project aims to maintain and expand the datasets and features.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Version control*; D.2.9 [**Software Engineering**]: Management—*software configuration management*; H3.5 [**Information Storage and Retrieval**]: Online Information Services—*web-based services*

General Terms

Management

Keywords

fine-grained histories, Git, Historage, Kataribe

1. INTRODUCTION

Code repository of version control system is one of the important source for research of Mining Software Repositories (MSR), since they store rich information of products and real software development/maintenance activities. Source code histories, such as the times of modifications, the number of fixing bugs, the authors of changes, etc., are easily collected from version repositories. In addition, integrating the data of code repositories with other software repositories like bug repositories and mail archives enables us to analyze bug fixing, communications among developers, and so on. There are active research areas that require version repositories, such as bug prediction [7, 9, 12, 13], code clone analyses [4, 10, 17], evolution studies [14, 16, 19], and assessment of activities [2, 3, 11].

Since empirical studies rely largely on data, furthering rich data should deepen the MSR research. Analysis of the histories of methods or functions, which is called as *fine-grained histories*, is one of the expected further research. However, it is not easy to collect fine-grained entity histories from usual code repositories. Several tools have been proposed and used in research. *BEAGLE* is a framework incorporating subtools from evolution metrics, software visualization and relational databases [5, 18]. At the point of fine-grained histories, it performs *origin analysis* to identify types of changes including renaming, moving, splitting, and merging. However, it only targets selected release revisions for applying *origin analysis*. *C-REX* [6] and *J-REX* [15] are evolutionary extractors. It records fine-grained entity changes over the development periods. Although *C-REX* and *J-REX* target entire revisions, it cannot identify renaming. *Kenyon* is designed to facilitate software evolution research [1]. It supports CVS, Subversion, and ClearCase SCM systems and conducts preprocessing tasks for fine-grained change analysis. Although it stores entire revisions, types of changes are limited to adding, deleting, and modifying. *APFEL* collects fine-grained changes in relational databases [21]. It investigates fine-grained changes at the token level. Though revisions are stored entirely, renaming is not identified.

Previous tools have limitations, that is, selective (not entire) versions and/or limited rename identification, which both are provided by normal code repositories for files. We have addressed these problems and proposed `Historage`, a fine-grained version repositories [8]. The key idea of Historage is using Git, one of the version control systems, as a storage of fine-grained entity histories. Since Historage repositories are also Git repositories, method histories can be obtained by similar procedure used for file histories. In addition, entire histories are stored, and renames are identified similarly to file-level code repositories.

MSR'14, May 31 – June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2863-0/14/05...$15.00
http://dx.doi.org/10.1145/2597073.2597125

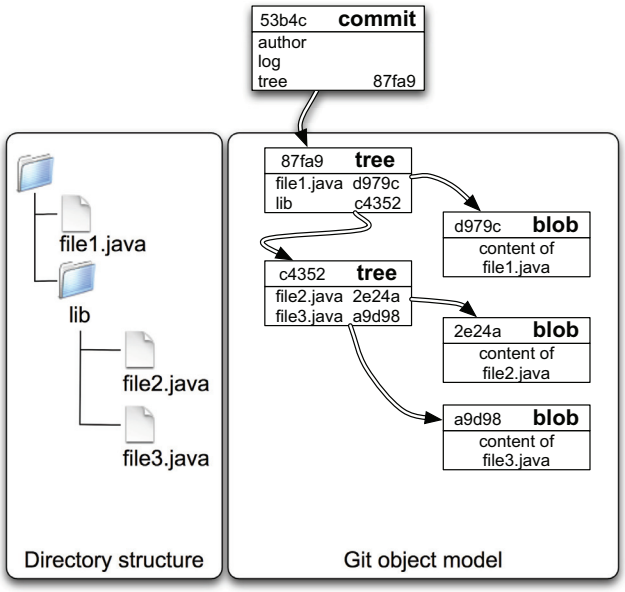


Figure 1: How a snapshot is stored in Git.

For the MSR community, we set up the Historage hosting service called `Kataribe`, which enables researchers and developers to browse fine-grained histories on the web and have their own Historage repositories in their local environment by cloning repositories. We plan to proceed the Kataribe project for maintaining Historage repositories and expanding the volume and the features of the web service.

2. HISTORAGE: FINE-GRAINED VERSION CONTROL SYSTEM

2.1 Git as Storage

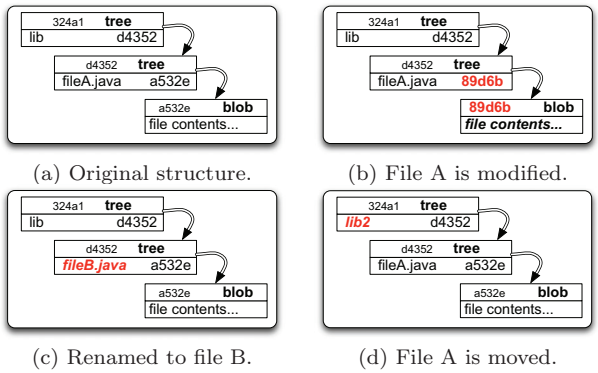
Historage is built on top of Git. Git is a content-addressable file system which controls file contents, directory structures, file histories, commit logs, etc., by managing Git objects. Each object is stored in Git object database and is compressed and named by the SHA-1 value of its contents.

Snapshot Representation.

Figure 1 shows how directory structure of a snapshot is stored and managed with Git object model. The left side of Figure 1 represents sample directory structure at the time of a commit and the right side shows a Git object model that reflect the directory structure. Each blob object, which represents a file, is referred by a tree object. A tree object, which represents a directory, refers blobs and trees. The top tree object is referred by a commit object, which contains the author and log of the commit. As shown in Git object model in Figure 1, each object is identified with SHA-1 value.

Change Identification.

Git stores snapshots separately, but does not store changes. If contents of files are unchanged, it will keep the same SHA-1 identifiers. With Figure 2, we explain how Git identifies the kind of the changes. Figure 2 (a) shows an original directory structure in Git object model. It shows that `fileA.java` exists in the directory named `lib`. The content



(a) Original structure. (b) File A is modified.

(c) Renamed to file B. (d) File A is moved.

Figure 2: How changes are identified in Git.

of `fileA.java` is stored in a *blob*, which is named by SHA-1 value: `a5352e`. The `lib` directory is represented as *tree* named as `d4352`, and the name of the directory is stored in the `324a1` *tree*.

If the `fileA.java` is modified, the Git object model changes to Figure 2 (b). Since the file content is changed, the corresponding *blob* is also changed. Figure 2 (c) represents the rename of the file. This can be identified because same *blob* SHA-1 value is linked to different file name, `fileB.java`. The Figure 2 (d) represented a directory structure after moving the `fileA.java`. This can be detected because the directory, which has different name `lib2`, contains the `fileA.java`.

When paths of the files are changed, it is often with the case that the contents of the files are also modified. Even in such case, Git can detect relationships between changes if the file contents are similar enough. This is performed by checking that the amount of deletion of original content and insertion of new content is larger than a threshold, which is set to 50% of the size of smaller files (original or modified). Therefore, if deletion or insertion is less than 50%, two files in parent and child commits are detected as moving or renaming. In addition, the threshold value can be changed.

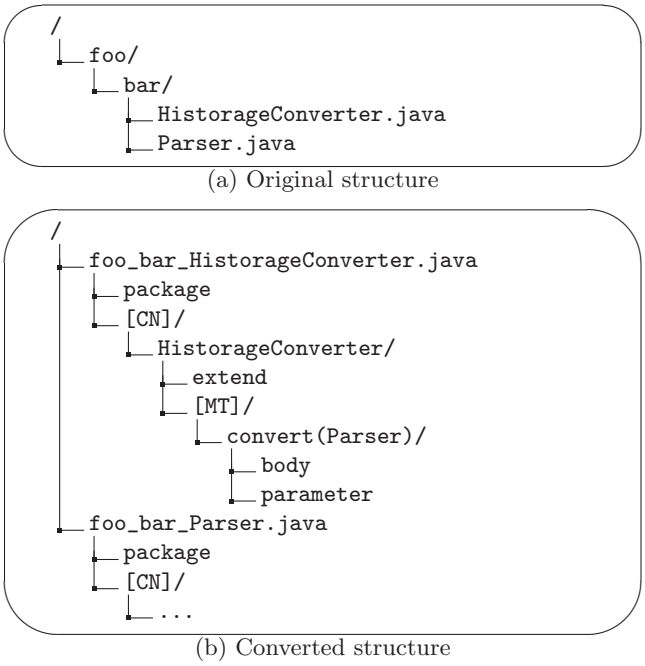
2.2 Git Extension for Historage

As seen before, Git stores snapshots separately, and the histories of files can be derived by finding matches between snapshots. Historage is constructed based on this Git architecture, that is, each fine-grained entity is additionally stored in snapshots. Methods are extracted from source code files, and stored as files.

3. KENJA: HISTORAGE BUILDING TOOL

We have developed a tool, Kenja[1], for building a Historage repository from an existing file-level Git repository. Kenja builds directory structure for the Historage based on the result of syntactic analysis of all source code files in each commit. The Historage repository made by Kenja does not store original files to save the building time. However, each commit of the converted Historage repository has a link to the original commit, which makes it possible to recover the original file histories. We provide the link by using `git notes`, which is a feature of Git. Currently, Kenja only supports building Historage for Java. A Historage repository stores information of packages, classes, method and fields.

[1] `http://github.com/niyaton/kenja`



(a) Original structure

(b) Converted structure

Figure 3: Directory structure converted by Kenja

Figure 3 (b) shows an example of directory structure converted by Kenja from the original structure shown in Figure 3 (a). Kenja creates directories that correspond to original Java file under the root of the converted repository. Names of these directories are determined from the paths of original Java files. Each top directory in the converted repository contains syntactic information of original Java file. For example, information of the classes is stored under the `[CN]` directory. If a class extends another class, the corresponding directory will include a file named as extend which contains the name of the extended class. For each directory corresponding with the class, information of the methods is stored under the `[MT]` directory, information of the constructor of the class is stored under the `[CS]` directory and information of the fields is stored under the `[FE]` directory.

As an example of mining Historage repository converted by Kenja, researcher can trace the changes of `convert` method in the `HistorageConverter` class by investigating the changes of the file `body` under the `foo_bar_HistorageConverter.java/[CN]/HistorageConverter/[MT]/convert(Parser)`. The time when the `convert` method was changed can be known by simply executing following Git command.

```
git log --name-status -M <path of body file>
```

4. KATARIBE: HISTORAGE HOSTING SERVICE

We present Kataribe as a hosting service of Historage repositories. Kataribe uses Gitlab, which is a well-known OSS for hosting Git repositories. Users can get Historage repositories from Kataribe without registration. Registration at Kataribe enables users to browse Historage repositories on the web. Gitlab enables users to see logs and graphical statics of repositories.

Features of Kataribe include importing existing Git repositories that are provided on Git hosting services such as Github, and also constructing Historage repositories incrementally. Since a Historage repository created by Kenja is separated from the original repository, users of Kataribe can continue their development regardless of their Historage repository. When developer pushes her/his commits from the original repository, Kataribe automatically converts pushed commits into the Historage repository. These features allow researchers to get latest fine-grained histories when they want to start their new research.

5. CURRENT DATA

Currently, we have prepared Historage repositories based on the work by Zhang et al. [20]. They provide the list of projects on Sourceforge.net for further research. Whole available Historage repositories can be seen in the following URL:

`http://kataribe.naist.jp/public`

6. CONCLUSIONS AND FUTURE WORK

As new data and tools become more easily available to public, researchers can widen and deepen their studies with them. The Kataribe project intends to share our fine-grained version control system, Historage. Since Historage is constructed on top of Git, method histories can be collected with Git commands. All Historage repositories are provided in Kataribe, a GitHub like hosting service. This enables users to clone Historage repositories to local and browse method histories on the web.

As future work, we plan to add Historage repositories for researchers. Furthermore, we have implemented a visualization feature, which summarizes semantics of changes in the commit by analyzing the commit of Historage repository. This feature will be available at the same location proposed in this paper.

— *Fine-Grained Histories (for all)*

7. ACKNOWLEDGMENTS

This study has been supported by JSPS KAKENHI Grant Number 25880015. The authors thank Chan Kar Long for valuable comments that helped further improve this paper.

8. REFERENCES

[1] J. Bevan, E. J. Whitehead, Jr., S. Kim, and M. Godfrey. Facilitating software evolution research with kenyon. In *Proc. of 5th Joint Meeting of the European Softw. Eng. Conf. and the ACM SIGSOFT Symp. on the Found. of Softw. Eng.*, ESEC/FSE-13, pages 177–186, New York, NY, USA, 2005. ACM.

[2] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy. Does distributed development affect software quality? an empirical case study of windows vista. In *Proc. of 31st Int. Conf. on Softw. Eng.*, ICSE '09, pages 518–528, Washington, DC, USA, 2009. IEEE Computer Society.

[3] C. Bird and T. Zimmermann. Assessing the value of branches with what-if analysis. In *Proc. of 20th ACM SIGSOFT Int. Symp. on Found. of Softw. Eng.*, FSE '12, pages 45:1–45:11, New York, NY, USA, 2012. ACM.

[4] N. Göde and R. Koschke. Frequency and risks of changes to clones. In *Proc. of 33rd Int. Conf. on Softw. Eng.*, ICSE '11, pages 311–320, New York, NY, USA, 2011. ACM.

[5] M. W. Godfrey and L. Zou. Using origin analysis to detect merging and splitting of source code entities. *IEEE Trans. Softw. Eng.*, 31:166–181, February 2005.

[6] A. E. Hassan and R. C. Holt. C-REX: An evolutionary code extractor for C. CSER meeting, Montreal, Canada, 2004.

[7] A. E. Hassan and R. C. Holt. The top ten list: Dynamic fault prediction. In *Proc. of 21st IEEE Int. Conf. on Softw. Maintenance*, ICSM '05, pages 263–272, Washington, DC, USA, 2005. IEEE Computer Society.

[8] H. Hata, O. Mizuno, and T. Kikuno. Historage: Fine-grained version control system for Java. In *Proc. of 3rd Joint Int. and Annual ERCIM Workshops on Principles of Softw. Evolution and Softw. Evolution Workshops*, IWPSE-EVOL '11, pages 96–100, New York, NY, USA, 2011. ACM.

[9] H. Hata, O. Mizuno, and T. Kikuno. Bug prediction based on fine-grained module histories. In *Proc. of 34th Int. Conf. on Softw. Eng.*, ICSE '12, pages 200–210, Piscataway, NJ, USA, 2012. IEEE Press.

[10] M. Kim, V. Sazawal, D. Notkin, and G. Murphy. An empirical study of code clone genealogies. In *Proc. of 5th Joint Meeting of the European Softw. Eng. Conf. and the ACM SIGSOFT Symp. on the Found. of Softw. Eng.*, ESEC/FSE-13, pages 187–196, New York, NY, USA, 2005. ACM.

[11] M. Kim, T. Zimmermann, and N. Nagappan. A field study of refactoring challenges and benefits. In *Proc. of 20th ACM SIGSOFT Int. Symp. on Found. of Softw. Eng.*, FSE '12, pages 50:1–50:11, New York, NY, USA, 2012. ACM.

[12] S. Kim, E. J. Whitehead, Jr., and Y. Zhang. Classifying software changes: Clean or buggy? *IEEE Trans. Softw. Eng.*, 34:181–196, March 2008.

[13] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Proc. of 27th Int. Conf. on Softw. Eng.*, ICSE '05, pages 284–292, New York, NY, USA, 2005. ACM.

[14] L. S. Pinto, S. Sinha, and A. Orso. Understanding myths and realities of test-suite evolution. In *Proc. of 20th ACM SIGSOFT Int. Symp. on Found. of Softw. Eng.*, FSE '12, pages 33:1–33:11, New York, NY, USA, 2012. ACM.

[15] W. Shang, Z. M. Jiang, B. Adams, and A. E. Hassan. Mapreduce as a general framework to support research in mining software repositories (msr). In *Proc. of 6th IEEE Work. Conf. on Mining Softw. Repositories*, MSR '09, pages 21–30, Washington, DC, USA, 2009. IEEE Computer Society.

[16] R. Sindhgatta, N. C. Narendra, and B. Sengupta. Software evolution in agile development: a case study. In *Proc. of ACM Int. Conf. Companion on Object Oriented Programming Syst. Languages and Appl. Companion*, SPLASH '10, pages 105–114, New York, NY, USA, 2010. ACM.

[17] S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta. An empirical study on the maintenance of source code clones. *Empirical Softw. Eng.*, 15(1):1–34, Feb. 2010.

[18] Q. Tu and M. W. Godfrey. An integrated approach for studying architectural evolution. In *Proc. of 10th Int. Workshop on Program Comprehension*, IWPC '02, pages 127–136, Washington, DC, USA, 2002. IEEE Computer Society.

[19] G. Xie, J. Chen, and I. Neamtiu. Towards a better understanding of software evolution: An empirical study on open source software. In *Proc. of 25th IEEE Int. Conf. on Softw. Maintenance*, ICSM '09, pages 51–60, 2009.

[20] F. Zhang, A. Mockus, Y. Zou, F. Khomh, and A. Hassan. How does context affect the distribution of software maintainability metrics? In *Proc. of 29th IEEE Int. Conf. on Softw. Maintenance*, pages 350–359, Sept 2013.

[21] T. Zimmermann. Fine-grained processing of CVS archives with APFEL. In *Proc. of OOPSLA Workshop on Eclipse Technol. eXchange*, eclipse '06, pages 16–20, New York, NY, USA, 2006. ACM.