A Hosting Service of Multi-Language Historage Repositories

Kyohei Uemura*, Yusuke Saito*, Shin Fujiwara*, Daiki Tanaka*, Kenji Fujiwara†, Hajimu Iida*, Kenichi Matsumoto*
* Nara Institute of Science and Technology
Ikoma, Nara 630–0192
Email: {uemura.kyohei.ub9@is, saito.yusuke.sl9@is, fujiwara.shin.fe5@is,tanaka.daiki.sx4@is
hata@is, iida@itc, matumoto@is }.naist.jp
† National Institute of Technology, Toyota College
Toyota, Aichi 471–8525
Email: fujiwara@toyota-ct.ac.jp

***Abstract*—In the research of Mining Software Repositories, source code repositories are one of the core sources since it contains the product and the process of software development. A source code repository stores the versions of files and makes it possible to browse the histories of files, such as modification dates, authors, messages, so on. Although such rich information of file histories is easily available, extracting the histories of methods/functions, 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 an original Git repository. Therefore, similar mining techniques for general Git repositories are applicable to Historage repositories. We also have developed `Kataribe`, a hosting service of Historage repositories, which contains hundreds of Historage repositories constructed from repositories in GitHub, which are written in C#, Java, Python and Ruby. The list of all Historage and original repositories are available at http://kataribe.naist.jp/public. With this dataset, we will promote in-depth and fine-grained software evolution research with diversity of programming languages.**

I. Introduction

Mining source code repositories tell us not only how software had been developed but also how we are ablet to develop software more rapidly and efficiency. For example, by mining history of source code changes, we can predict the location of the bugs [12], [18], [31], analyze source code evolution [4], [27], automatically generate patches of the bugs [16], [21], and so on. Modern topics of MSR have combined various sources such as bug reports [6], [24], crash reports [5], [17], [29], and energy consumptions [13], [22].

Since empirical studies rely largely on data, furthering rich data should deepen the Mining Software Repositories (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 [2], [9], [28], [32]. However, Previous tools have limitations, that is, selective (not entire) versions and/or limited rename identification. We have addressed these problems by proposing `Historage`, fine-grained source code repositories [10]. 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 procedures used for file histories. In addition, entire histories are stored, and renames are identified similarly to file-level code repositories. For the MSR community, we have provided Historage repositories of Java projects [7].

As pointed out by Nagappan et al., one of the goals of software engineering research is to achieve generality [20]. Since the previous version of Kataribe provided only Java repositories, it was not viable to perform MSR research with wide diversity. To support diversity in MSR research, we will provide 583 of Historage repositories written in C#, Java, Python and Ruby, by extending the previous Kataribe [7]. The following two points are the criteria we focus on for selecting the languages to be newly supported in Kataribe. First, the language is object oriented. Second, there are an adequate number of repositories written in that languages registered on Github. We selected Python, and Ruby not only because they are widely used, but also they are script languages, which are different from Java. In addition, since C# is a compiled language like Java, it is also selected as one of the newly supported languages in Kataribe.

II. Background on where the data came from

We collected (file-level) Git repositories from the Github[1]. The selected projects are mainly written in C#, Java, Python or Ruby. The current project collection procedure for each language as follow.

1) Search projects that have been forked at least once and stared from more than 300 users, from https://github.com/search/advanced.
   There are 265, 1,343, 1461, and 1,314 projects for C#, Java, Python, and Ruby respectively. Currently, we chose top 100 projects of sort-result by Most Stars and top 100 projects of sort-result by MostForks, for each language.

[1]https://github.com

2) Add trending projects (25 projects for each) from https://github.com/trending.
3) Remove projects that have less than 30 commits or only one developer contributed to the project.

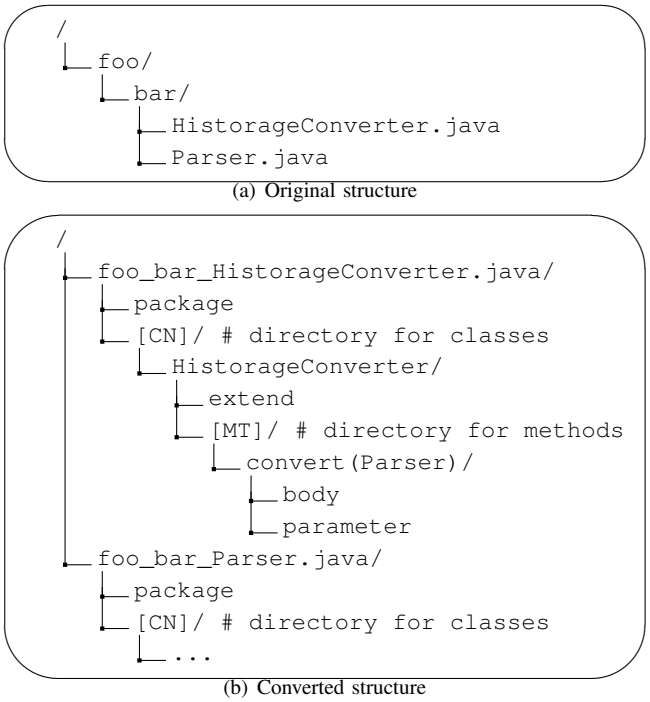
III. Historage: What types of data the dataset contains

We then convert file-level Git repositories to Historage repositories with a tool, `Kenja`[2]. Kenja constructs a directory structure for the Historage based on the result of syntactic analysis of all source code files in each commit. The original version of the Historage proposed by Hata et al. keeps all original files in the file-level repository [10]. However, the Historage repository made by Kenja does not store original files to save the building time. Thus, 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.

Figure 1(b) shows an example of directory structure converted by Kenja from the original structure shown in Figure 1(a). Kenja creates directories that correspond to original source code files under the root of the converted repository. Names of these directories are determined from the paths of original source code files. Each top directories in the converted repository contain 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 to 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.

C# Support: In order to support C# on Kenja, we adopted the Roslyn[3] a .NET compiler platform provided by Microsoft for parsing C# source code. Since the grammar and concept of C# are similar to Java, Kenja creates same directory structure from C# code. Compared with other languages, C# has a unique specification called as a **partial class**. In C# language, a developer can write a definition of a specific class into the separated files. Since Kenja constructs a structure of the directory that represents fine-grained entities per source code file, definitions of the partial class will be stored in the multiple directories. Thus, a researcher who mines C# repository should have in mind that limitation of Historage.

Python Support: The standard library of Python contains the **ast** package that allows us to access abstract syntax tree of the Python code. Kenja uses that packages to parse a Python script file and construct a directory structure for Historage. Unlike Java and C#, Python is a kind of dynamic programming language, and it allows writing statements in the top-level of the file. To represent this characteristic of Python into the Historage, Kenja creates `top-level-statements` file that contains statements on the top-level of the file. In other words, all statements without a declaration of the classes and functions will be stored in this file.



(a) Original structure

(b) Converted structure

Fig. 1. Directory structure converted by Kenja for Java source files

Ruby Support: In order to support Ruby on Kenja, we adopt a parser library its name is also Parser[4]. Same as Python, programmers can write statements on the top-level of the file, Kenja also creates a top-level-statements file for each Ruby script file as we mentioned before.

IV. Kataribe: Hosting Service of Historage Repositories

A. How to make the dataset easily accessible

We have developed Kataribe, a hosting service of Historage repositories [7]. However, in our previous paper, Kataribe hosted only Java repositories, we expanded it to host other repositories too, which are C#, Python and Ruby 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 repositories, users of Kataribe can continue their development regardless of their Historage repositories. When developer pushes her/his commits into their original

[2]http://github.com/niyaton/kenja

[3]https://github.com/dotnet/roslyn

[4]https://github.com/whitequark/parser

TABLE I
DETAIL OF THE DATA

Language	Number of Projects	Calculation Method	Number of Files	Number of Commits	Number of Authors
C#	165	Min	26	40	2
		Max	10,288	11,333	859
		Average	549.76	719.58	29.27
Java	186	Min	35	49	3
		Max	45,049	159,324	2,339
		Average	1,673.60	5,449.10	148.70
Python	151	Min	23	46	2
		Max	74,529	148,304	3421
		Average	1,461.91	5,216.95	190.88
Ruby	81	Min	24	82	7
		Max	9,828	49,993	5,818
		Average	975.62	5,210.77	309.47

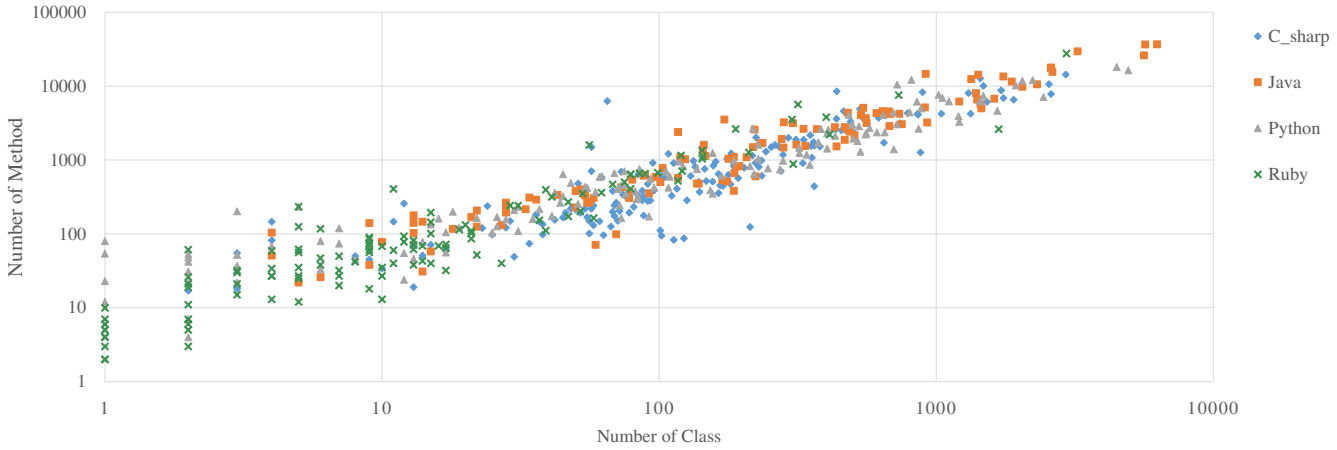


Fig. 2. Scatterplot: Number of class and method

repositories, Kataribe automatically converts pushed commits into the corresponding Historage repositories. These features allow researchers to get latest fine-grained histories when they want to start their new research.

B. Detail of hosting repositories by Kataribe

Table I shows details of projects which are selected by the procedure described in Section II and they are available in Kataribe. Kataribe provides 583 projects in total. Figure 2 shows the relationship between the number of classes and the number of methods for all investigated projects, which are written in different languages.

C. Where the data set currently is

The followings are Kataribe web cite and its support page:

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

$$http://sdlab.naist.jp/kataribe/$$

D. How the data can be obtained

Since Historage repositories are Git repositories, users can easily obtain Historage repositories by cloning from Kataribe. To analyze the histories of methods or functions, git commands like `git log` can be available. For example, the history of method including renaming can be obtained with the following Git command:

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

E. Any challenges in obtaining the data

Since Historage repositories retain the same commit information in original Git repositories, it is easy to link method histories with other sources like bug reports, crash reports, and code reviews. We recommend users to do some linking, and work with repositories written with different languages and sizes.

V. EXAMPLES OF ANALYSIS WITH KATARIBE AND HISTORAGES

In this section, we introduce several examples of analysis with Kataribe and Historages. First, we analyzed transition of the project scale based on the number of the classes and the number of the methods in section V-A. Secondly, we analyze code clone at method-level in section V-B.

A. Analysis of Fine-Grained Project Scales

In Historage, the structure of source code is represented as directories structure. Even though each programming language has its own characteristics, it introduces no difference when the

TABLE II
DETAIL OF TARGET PROJECT

Repository Name	Language	#Commit
Nrefactory	C#	4,380
druid	Java	4,469
CouchPotatoServer	Python	4,869
redcar	Ruby	4,849

TABLE III
RESULT OF DETECIT METHOD LEVEL CODE CLONES

Project Name	Language	#Class	#Method	#Similar Method
libopenmetaverse	C#	2,596	7,817	518
storm	Java	1,392	8,056	525
nupic	Python	1,021	7,660	235
ruby	Ruby	732	7,557	314

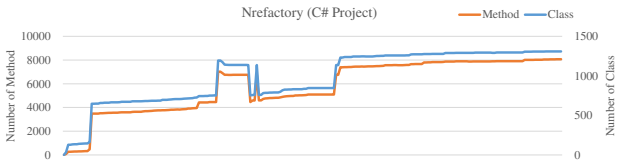
structure of methods and classes is interpreted as a directory structure. Therefore, with this directory representation, analysis on classes and methods can be performed independently from the written language. For example, the number of classes can be known by counting the directories within the directory named as [CN]. Similarly, the number of methods can be retrieved by counting the directories within the directory named as [MT]. In addition, since Historage applies git for version management, changes within each commit can be conveniently investigated as well.

From here, we present the result from an investigation that we actually conducted by using Historage provided in Kataribe. Table II shows the four selected projects for our investigation. They are intentionally chosen because they have similar number of commits and are respectively written in four languages. Figure 3 shows the relative transitions between the number of classes and the number of methods for the projects. In general, the number of classes and the number of methods within each project change simultaneously, and neither of the number reflects any unilaterally large fluctuation. In addition, beside druid, sudden changes in terms of the number of classes and the number of the methods can be observed in the other three projects. It can be due to the possibility that large number of functions are added/deleted, or the source code was reorganized by refactoring.

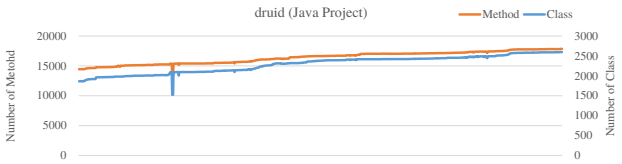
As indicated by this analysis, Kataribe can be used for performing fine-grained analysis or visualization on source code at class or method level. Besides, since 583 projects from four major programming languages are available in Kataribe, investigating the respective characteristics of each language is considered to be possible.

B. Analysis of Similar Methods

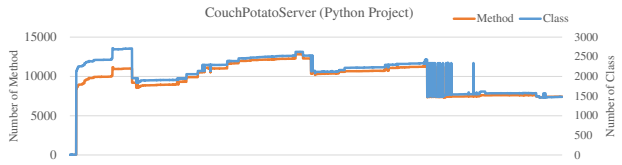
In Historage, every method directory contains a body file that describes the contents of method in plain text. Therefore, by detecting code clones from such body file, we can perform investigations on code clones at method level. In this subsection, we present how we detect similar methods by using CCFinderX, which the most commonly used code clone detection tool.



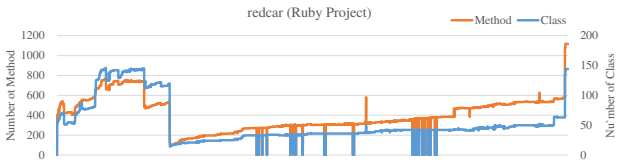
(a) Nrefactory



(b) druid



(c) CouchPotatoServer



(d) redcar

Fig. 3. Transition of Number of Class and Method

CCFinderX can detect code clone in source code which is written in C++, java, C#, or COBOL. In addition, it can detect clone from plane text file as well. For detecting the code clone from body file in Historage, we set the detecting mode to target the type of plain text file. From the detection result, we can identify similar methods by extracting the group of body files with high RSA values, which indicate the ratios of code clones occupying the body files.

We performed out test case of detecting similar methods on four purposely selected projects. The chosen projects have similar number of methods and they are respectively written in four different languages. The threshold of RSA for clone detection is set to be 0.8. The detection result and the scatter plot of code clone for each project are respectively shown in Table III and Figure 4. Within the scatter plot, the black dots represents the location of similar method. The vertical axis and the horizontal axis refer to the number of lines in the source code, while the gray line indicates the border lin of the file. According to the result, projects of JAVA or C# have relatively more similar methods, compared to those written in Python and Ruby. Furthermore, for the projects written Java

List 1. Sample of Similar Method

```
def _mapParamFromPythonToC(self, paramName):
  '\n    Map Python object values to
      equivalent enumerated C values.\n    '
  if (paramName == 'boundaryMode'):
    if (self._boundaryMode == 'constrained'):
      enumValue = 0
    elif (self._boundaryMode == 'sweepOff'):
      enumValue = 1
    return self._convertEnumValue(enumValue)
  elif (paramName == 'phaseMode'):
    if (self._phaseMode == 'single'):
      enumValue = 0
    elif (self._phaseMode == 'dual'):
      enumValue = 1
    return self._convertEnumValue(enumValue)
  elif (paramName == 'normalizationMethod'):
    if (self._normalizationMethod == 'fixed'):
      enumValue = 0
    elif (self._normalizationMethod == 'max'):
      enumValue = 1
    elif (self._normalizationMethod == 'mean'):
      enumValue = 2
    return self._convertEnumValue(enumValue)
  elif (paramName == 'perPlaneNormalization'):
    if (not self._perPlaneNormalization):
      enumValue = 0
    else:
      enumValue = 1
    return self._convertEnumValue(enumValue)
  elif (paramName == 'perPhaseNormalization'):
    if (not self._perPhaseNormalization):
      enumValue = 0
    else:
      enumValue = 1
    return self._convertEnumValue(enumValue)
  elif (paramName == 'postProcessingMethod'):
    if (self._postProcessingMethod == 'raw'):
      enumValue = 0
    elif (self._postProcessingMethod == 'sigmoid'):
      enumValue = 1
    elif (self._postProcessingMethod == 'threshold'):
      enumValue = 2
    return self._convertEnumValue(enumValue)
  else:
    assert False
```

or C#, the similar methods tend to exist closely together. List 1 shows a sample of similar method which was detected. The method which is named as mapParamFromPythonToC exists both in GaborNode2.py and Convolution.py from the nupic project which is written in Python.

We consider that the detected similar methods can be regarded as the candidates for refactoring.

VI. A SAMPLE LIST OF WAYS THAT THE DATA COULD BE USED

Here are some ideas of research using our method/function level datasets, and some related work on file-level studies.

Quality:

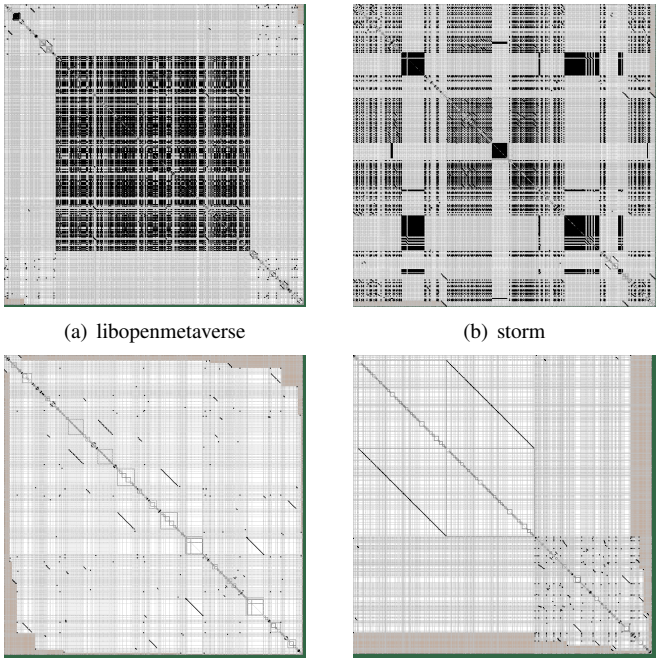
Investigate quality of methods and its histories. *Related studies:* Code ownership and software quality [8]. Dormant bugs [3].

Human Aspects:

Analyze developers' behaviors. *Related studies:* The intent of changes [19].

Process:

Identify the relation between method evolution and code review or test. *Related studies:* Modern code review practices in defective and clean files [26]. Co-evolution of infrastructure and source code files [14]. Patterns of changes [23].



(a) libopenmetaverse (b) storm

(c) nupic (d) ruby

Fig. 4. Scatterplots of Method Level - Code Clone

VII. ANY RESEARCH THAT HAS USED THE DATASET UP TO THIS POINT

The followings are studies that used Historage repositories and Kataribe service.

Hata et al. collected method-level historical metrics, and realized method-level defect prediction with them [12]. The collected method-level metrics include churn, the number of changes, ages, logical coupling, process complexity, ownership, and so on.

Tantithamthavorn et al. conducted information retrieval (IR)-based bug localization at method level [25]. They discussed the impact of granularity levels on IR-based bug localization between method level and class level.

Yuzuki et al. conducted an empirical study of method-level conflict resolution [30]. They reported the statistics of patterns in conflicts and conflict resolutions from 10 OSS projects written in Java.

Hata et al. proposed a technique to identify logical structural changes [11]. Using Historage repositories, it can provide the summary of changes including modify, rename, hide, unhide, and move to methods, classes, and packages for every commit.

Kashiwabara et al. proposed a technique to recommend candidate of verbs for method names so that developers can use various verbs consistently. They evaluated their proposal with actual method renaming extracted from Historage repositories [15].

The Fujiwara et al. proposed high-speed and highl-precise refactoring detection technique using Historage [1]. Based on this technique, an implemented tool named as Kenja is able to detect two "Extract Method" and of "Pull Up Method". Not only as a tool to construct Historage, Kenja is able to detect refactoring in any existing Historages as well.

VIII. Conclusion

In this study, based on our precedent works on Historage which is the fine-grained git-repository and its hosting service, Kataribe, we expand the usage by supporting more languages. Currently, Historage is compatible with C#, JAVA, Python and Ruby. As such, a total of 583 repositories from these four languages are presented in Kataribe. By using the Historage provided in Kataribe, it is possible to perform fine-grained analysis on source code. Such analysis can be carried out by applying the same techniques used in git repository analysis from traditional MSR research. As an example of demonstrating how analysis on Historage can be conducted, we investigated the relative transitions between the number of methods and the number of classes on four projects from four different languages. Furthermore, another example we showed is investigating similar methods by applying code clone detection technique. Among all the conducted investigations in this study, we selected only one project from each language. We consider that by performing further investigations on more subjects, we can analyze the differences among the characteristics of each language.

Kataribe is a scalable data source. As GitHub grows, Kataribe will also grow. This allows MSR researchers to conduct their studies with latest datasets. Kenja, a tool to create Historage, is publicly available. Therefore, it is possible for users to prepare other Historage repositories by themselves. In addition, supporting other languages is also possible. We are welcome to any contributions that can expand our datasets.

IX. Acknowledgments

This work has been supported by JSPS KAKENHI Grant Number 16H05857 and Program for Advancing Strategic International Networks to Accelerate the Circulation of Talented Researchers: Interdisciplinary Global Networks for Accelerating Theory and Practice in Software Ecosystem.

The first author would like to thank Chan Kar Long for his corrections.

References

[1] K. Fujiwara,N. Yoshida,H. Iida. An Approach for Fine-grained Detection of Refactoring Instances using Repository with Syntactic Information(in Japanese). IPSJ Journal,volume 56, number 12, pages 2346-2357 December 2015.

[2] J. Bevan, E. J. Whitehead, Jr., S. Kim, and M. Godfrey. Facilitating software evolution research with kenyon. ESEC/FSE-13, pages 177–186, 2005.

[3] T.-H. Chen, M. Nagappan, E. Shihab, and A. E. Hassan. An empirical study of dormant bugs. MSR '14, pages 82–91, 2014.

[4] J. Davies, D. M. German, M. W. Godfrey, and A. Hindle. Software bertillonage. *Empirical Softw. Eng.*, 18(6):1195–1237, Dec. 2013.

[5] T. Dhaliwal, F. Khomh, and Y. Zou. Classifying field crash reports for fixing bugs: A case study of mozilla firefox. ICSM '11, pages 333–342, 2011.

[6] M. Erfani Joorabchi, M. Mirzaaghaei, and A. Mesbah. Works for me! characterizing non-reproducible bug reports. MSR '14, pages 62–71, 2014.

[7] K. Fujiwara, H. Hata, E. Makihara, Y. Fujihara, N. Nakayama, H. Iida, and K. Matsumoto. Kataribe: A hosting service of historage repositories. MSR '14, pages 380–383, 2014.

[8] M. Greiler, K. Herzig, and J. Czerwonka. Code ownership and software quality: A replication study. MSR '15, pages 2–12. IEEE, May 2015.

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

[10] H. Hata, O. Mizuno, and T. Kikuno. Historage: Fine-grained version control system for Java. IWPSE-EVOL '11, pages 96–100, 2011.

[11] H. Hata, O. Mizuno, and T. Kikuno. Inferring restructuring operations on logical structure of Java source code. IWESEP '11, pages 17–22, 2011.

[12] H. Hata, O. Mizuno, and T. Kikuno. Bug prediction based on fine-grained module histories. ICSE '12, pages 200–210, 2012.

[13] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky. Greenminer: A hardware based mining software repositories software energy consumption framework. MSR '14 pages 12–21, 2014.

[14] Y. Jiang and B. Adams. Co-evolution of infrastructure and source code – an empirical study. MSR '15, pages 45–55, 2015.

[15] Y. Kashiwabara, T. Ishio, H. Hata, and K. Inoue. Method verb recommendation using association rule mining in a set of existing projects. *IEICE Transactions on Information and Systems*, E98-D(3):627–636, 3 2015.

[16] D. Kim, J. Nam, J. Song, and S. Kim. Automatic patch generation learned from human-written patches. ICSE '13, pages 802–811, 2013.

[17] D. Kim, X. Wang, S. Kim, A. Zeller, S. Cheung, and S. Park. Which crashes should i fix first?: Predicting top crashes at an early stage to prioritize debugging efforts. *IEEE Trans. Softw. Eng.*, 37(3):430–447, May 2011.

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

[19] A. Mauczka, F. Brosch, C. Schanes, and T. Grechenig. Dataset of developer-labeled commit messages. MSR '15, pages 490–493, 2015.

[20] M. Nagappan, T. Zimmermann, and C. Bird. Diversity in software engineering research. ESEC/FSE 2013, pages 466–476, 2013.

[21] F. S. Ocariza Jr., K. Pattabiraman, and A. Mesbah. Vejovis: Suggesting fixes for javascript faults. ICSE 2014, pages 837–847, 2014.

[22] G. Pinto, F. Castor, and Y. D. Liu. Mining questions about software energy consumption. MSR '14, pages 22–31, 2014.

[23] B. Ray, M. Nagappan, C. Bird, N. Nagappan, and T. Zimmermann. The uniqueness of changes: Characteristics and applications. MSR '15, pages 34–44. 2015.

[24] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani. Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation. MSR '13, pages 2–11, 2013.

[25] C. Tantithamthavorn, A. Ihara, H. Hata, and K. Matsumoto. Impact analysis of granularity levels on feature location technique. APRES '14, pages 135–149, 2014.

[26] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida. Investigating code review practices in defective files: An empirical study of the qt system. MSR '15, pages 168–179, 2015.

[27] 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.

[28] Q. Tu and M. W. Godfrey. An integrated approach for studying architectural evolution. IWPC '02, pages 127–136, 2002.

[29] S. Wang, F. Khomh, and Y. Zou. Improving bug management using correlations in crash reports. *Empirical Softw. Eng.*, pages 1–31, 2014.

[30] R. Yuzuki, H. Hata, and K. Matsumoto. How we resolve conflict: an empirical study of method-level conflict resolution. SWAN '15, pages 21–24, 2015.

[31] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou. Towards building a universal defect prediction model. MSR 2014, pages 182–191, 2014.

[32] T. Zimmermann. Fine-grained processing of CVS archives with APFEL. eclipse '06, pages 16–20, 2006.