ROCAT on KATARIBE: Code Visualization for Communities

Tomohiro Ichinose*, Kyohei Uemura*, Daiki Tanaka*, Hideaki Hata*, Hajimu Iida*, Kenichi Matsumoto*
*Graduate School of Information Science, Nara Institute of Science and Technology
Takayama 8916-5, Ikoma, Nara, 630-0192 Japan
Email: {ichinose.tomohiro.ik1@is, uemura.kyohei.ub9@is, tanaka.daiki.sx4@is, hata@is, iida@itc, matumoto@is}.naist.jp

***Abstract*—As globally distributed software development has become generalized, social coding platforms like Github are becoming increasing needed for team collaboration. To share and help understand the overview of projects among teams, source code visualization is useful. In this paper, we present a new visual software development tool, Rocat, a city-like software code visualization in virtual reality. In addition, we integrate Rocat with a GitLabbased code hosting service, Kataribe. By integrating Rocat into Kataribe, fine-grained code information can be visualized, and city visualization can be easily shared. We present our tool and provide the features of Rocat on Kataribe. A Screencast for demonstration is available at the following URL. https://www.youtube.com/watch?v=ZQTTO91v4No**

I. Introduction

Since the beginning of software development, the size and complexity of software have been continuously on the rise. If the software for a space shuttle 30 years ago was 400,000 lines of code, nowadays even a smartphone application can easily exceed that size. Not only has the size of software increased in terms of LOC, but integration and interaction with external services has also dramatically increased. However, with the increase in size and complexity, comprehending the structure and analytics of software, especially without a good way to visualize the information has become increasing difficult.

One good way to represent software structure, size, and modularity is CodeCity [1], which represents classes as buildings and packages as districts in 3D, similar to a city. Since the original CodeCity does not offer a great number of interactions for a developer with a city, there has been some work to extend it. One interesting approach is CodeMetropolis [2], where software cities are represented in Minecraft, allowing for better collaboration within the Minecraft game. Showing similar data as CodeCity, CodeMetropolis allows the user to walk around the city and interact closely with the buildings (this is the source code). The same representation has also been used to specifically represent the development history of a city using a bird's-eye view [3]. Each approach has its own purpose, but all approaches assist in the understanding of the software.

The reason we believe these visualization techniques are good is because of the ease in understanding the structure of the software, the class size, and the number of packages within seconds. Furthermore, humans are very good at processing very large amounts of information in real-world city-like views quite quickly since we learn and adapt to the environments we live in. For example, just by glancing at a building, it is usually possible to tell the age of the building, i.e., whether it has a modern or classic design, the material it is made of, the color, and more. That being said, using a visualization technique that is similar to the real-world allows us to map the meanings of characteristics from the real-world to some aspect of the software, processing it with a similar speed as we would process the meaning of the real-world object.

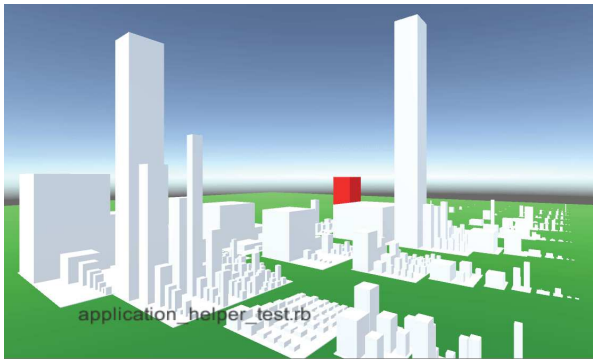

Fig. 1. City-like code visualization

Even though we believe the previously mentioned approaches are useful for visualizing software, we think that the full potential of such visualization techniques have not been fully utilized. These approaches use the concepts of a city to some extent, but they have not included many other aspects that real cities have and that can be used to deliver even more information to software engineers. If we manage to grasp this potential and use it in a way that is consistent with how we process information in a real environment, understanding the elements of a software system will get become faster and less superficial.

For this reason we want to extend the ways in which software cities can reflect real cities. We want to provide an experience where the software engineer is immersed in the software, exploring and learning (maybe even subconsciously) about it while walking around, in the same way a person would explore a real city. We would like to allow the engineer to naturally interact with the information and source code of the software so that such an enriched experience could help in better retention of information, as well as in easier comprehension. To realize such system, we present a visualization

tool, Rocat, which shows a city-like code visualization in a virtual reality environment.

In addition, we would like to introduce a new concept of code visualization, called *code visualization for communities*. Previous visualization tools focused mainly on supporting developers, a single developer [1] or a team of developers [2]. However, we think that visualization should be beneficial for communities too since the effectiveness of social coding [4] and pull-based development [5] has been made clear. To address this challenge, we integrated the Rocat visualization system on Kataribe, our code hosting service. With this kind of system, communities can share a large amount of information about the underlying software in a way that is quick and easy to understand.

II. Rocat: Software Visualization

To allow and support developers in comprehending source code and the conditions of projects, we propose Rocat, where developers can feel the code as if in an actual city, which will provide richer experiences for the developer by enabling exploration of the city. If the information represented about a building (or city) and its meaning is similar with a real-world city, it will be easy to quickly retrieve the meaning of what they view. In this section, we provide an overview and an implementation of Rocat for further usage.

A. Overview and Implementation

Rocat is a visualization tool that can be applied to multiple software development environments. Figure 2 illustrates a snapshot of a city-like view in Rocat. Similar to CodeCity [1], extracted metrics are mapped on the height and the base size of buildings. The height represents the LOC and the base size represents the number of comments of the source code in the current system. Users can explore the city among the buildings. When one building is selected (for example, the red building in the figure), its source code can be available on demand. It is possible to edit this source code with this view. The color of the building describes the top contributor of the file corresponds to the building (the color of each contributor is selected randomly). These colors can be used to find out which developer understands the software and is responsible for certain source code file.

Although CodeMetropolis is realized on top of an attractive popular role-playing game, Minecraft[1] [2], we have started implementation using a cross-platform game engine, Unity[2]. This is because we do not want to limit Rocat to specific purposes, but want to develop a general software development platform in the future. By using Unity, Rocat can be run on a web browser or any platform and can adopt Oculus Rift to using Rocat in virtual reality.

Rocat is not the only visualization possible, but it leads to a novel software development environment. We describe our concept with the following related approach.

[1]https://minecraft.net/
[2]https://unity3d.com/

B. Concept

1) Virtual Museum: A virtual museum is a digital entity that draws on the characteristics of a museum in order to complement, enhance, or augment the museum experience through personalization, interactivity, and a richness of content[3]. torytelling is one of the aims of virtual museums, which aim to make cultural content more attractive for the public and, consequently, the learning process easier [6]. This aim is also important for Rocat.

2) Web Maps: Web maps like Google Maps[4] are indispensable for exploring real world cities. Among various useful features, a zooming feature is one of the most basic and essential in comprehending the structure of cities. Rocat provides hierarchical views from a bird's-eye view to an explorative view. From bird's-eye views, developers can glance at the characteristics of parts of projects or the entire project. With explorative views, developers can investigate each building as well as the neighboring buildings.

3) Visual Software Analytics Platform: Software analytics is an analytics on software data for managers and software engineers with the aim of empowering individual and team software developers to gain and share insights from their data to make better decisions [7] SAMOA is a visual software analytics platform [8]. Because Rocat is also a software development environment, it needs help in decision-making, that is, Rocat can be a visual software analytics system too. SAMOA is a 2D system with different views: a snapshot view, an evolution view, and an ecosystem view. Introducing these kinds of views in 3D environments should be helpful.

4) Team Collaboration Platform: From an empirical study of modern code review, Bacchelli and Bird report the motivations for code review [9]. Although finding defects is still the main motivation, reviewers can also expect additional benefits such as knowledge transfer, increased team awareness, sharing code ownership, and so on. To support these outcomes, Rocat can also be helpful; that is, Rocat can be a team collaboration platform too.

III. Kataribe: Fine-grained Code Hosting

Kataribe[5] is a hosting service of Historage repositories [10]. Historage repositories are fine-grained source code repositories [11], and are converted from file-level Git repositories with a tool, Kenja[6]. Kenja constructs a directory structure for the Historage based on the result of syntactic analysis of all source code files in each commit. Currently, Kataribe hosts Historage written in C#, Go, Python and Ruby, since Kenja can only construct on those programming languages. 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

[3]https://en.wikipedia.org/wiki/Virtual_museum
[4]https://maps.google.com/
[5]http://sdlab.naist.jp/kataribe/
[6]http://github.com/niyaton/kenja

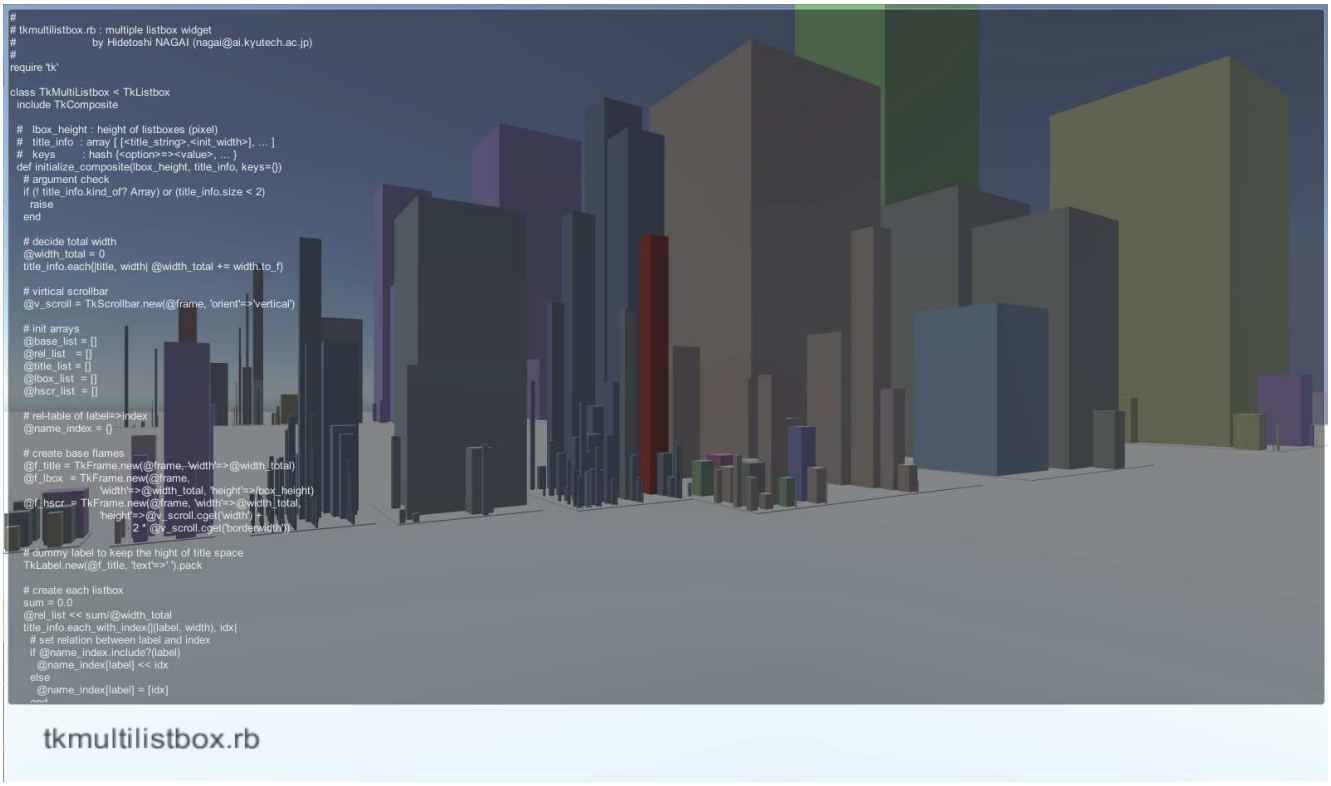

Fig. 2. Snapshot of Rocat

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

IV. Rocat on Kataribe

We integrate Rocat into Kataribe to provide software analytics for a community. Figure 3 shows a screen capture of Rocat on Kataribe. With Web browsers, users can see Rocat visualizations. In addition, the source code of a selected module is available on the same web page.

Rocat on Kataribe runs by a Unity Web Player[7], one of Unity's supported platforms for executing the project on a Web browser. The metrics data for constructing the building are managed on Kataribe, so updating the city and each building can be easily performed. In addition, data and source code shown by Rocat are always updated to the newest version.

[7]http://unity3d.com/webplayer

Since Kataribe uses Gitlab, in addition to developers, users without registration are also able to look through the overview of the project using Rocat. This characteristic of sharing visualization in public differs from prior studies and tools which targeted a single developer or project team. Also, by integrating Rocat to a source code hosting service, the user will no longer need to do extra work to share the visualization and can concentrate on their original work. To make better use of Kataribe, the Historage metrics can be used to construct the building. The metrics included in Historage helps the user obtain a more in-depth analysis and extend the possibilities for a more understandable visualization.

V. Discussions and Future Work

We strongly believe that visualization is a crucial part of successful software engineering. Currently topics such as software analytics, team collaboration, learning/comprehension, and knowledge sharing are being studied separately, but we believe that a visual environment can be the center of these topics.

This paper presents a code city for multiple software development environments, named Rocat. To develop this new visual software environment, we have also presented Rocat on Kataribe to provide software analytics for communities.

Now we are developing a system using Rocat that visualizes self-admitted technical debt (SATD). Technical debt is a metaphor that has been used to express non-optimal solutions

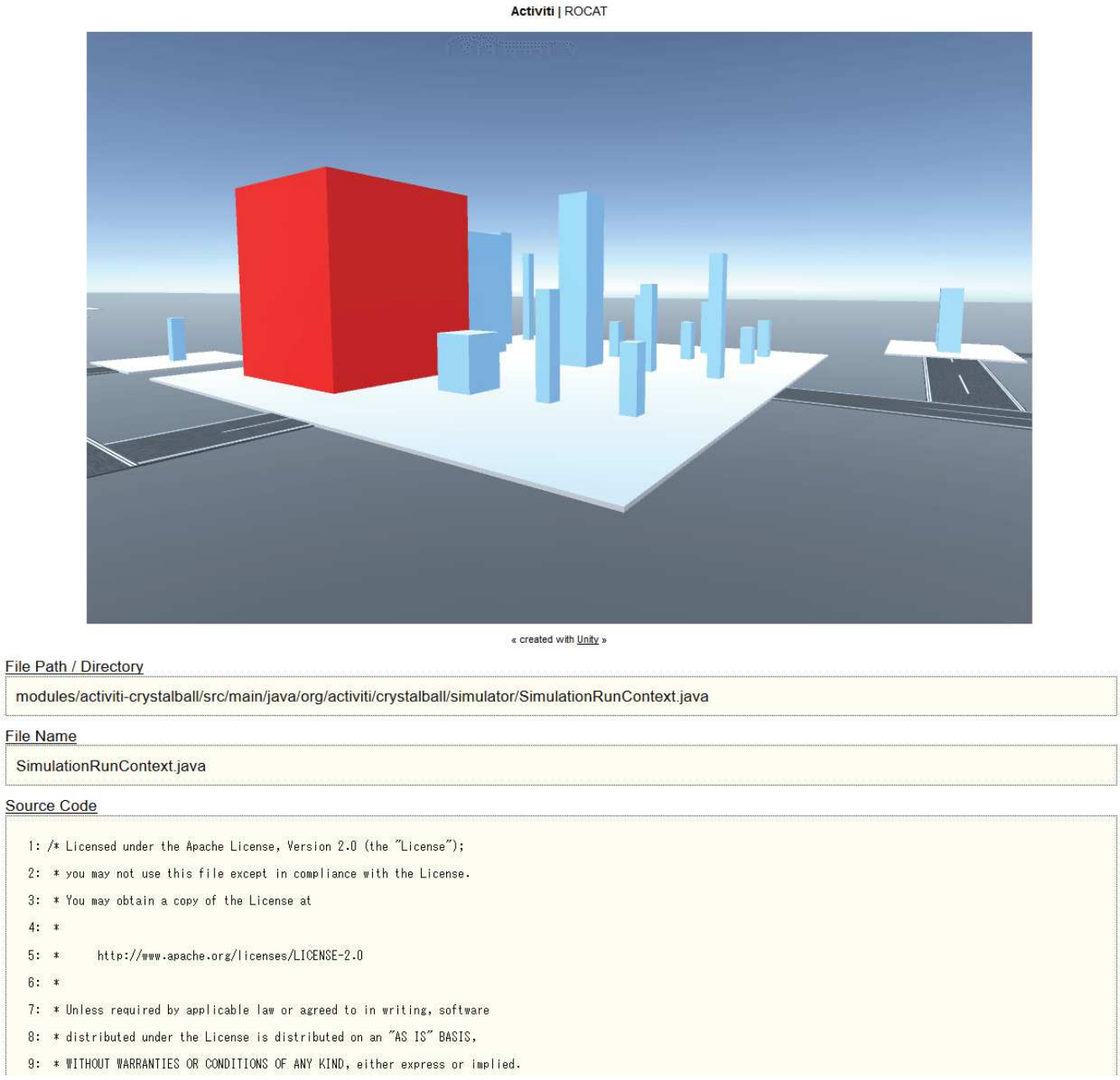

Fig. 3. Snapshot of Rocat on Kataribe

of software development, and SATD is a kind of technical debt that is intentionally introduced into source code with keywords such as “TODO” or “FIXME” [12]. Figure 4 shows a screen capture of the system. The green ball and orange bubbles are markers of SATD. The users can find files that contain SATD without reading source codes and share information about which files need refactoring. We would like to evaluate our system with interviews with OSS developers, and then analyze their repositories to see the changes after introducing the system into development.

Furthermore, Rocat can visualize the relationship among source codes with program structures at the same time. For example, in the case of call-graph visualization, 2D visualization can represent this relationship by using nodes and edges, but it is difficult to visualize other metrics clearly in the same view. Rocat, however, can visualize such a relationship with a bridge between buildings. Developers can understand a call graph and program structure at the same time. This approach will be able to accelerate debugging tasks, for example.

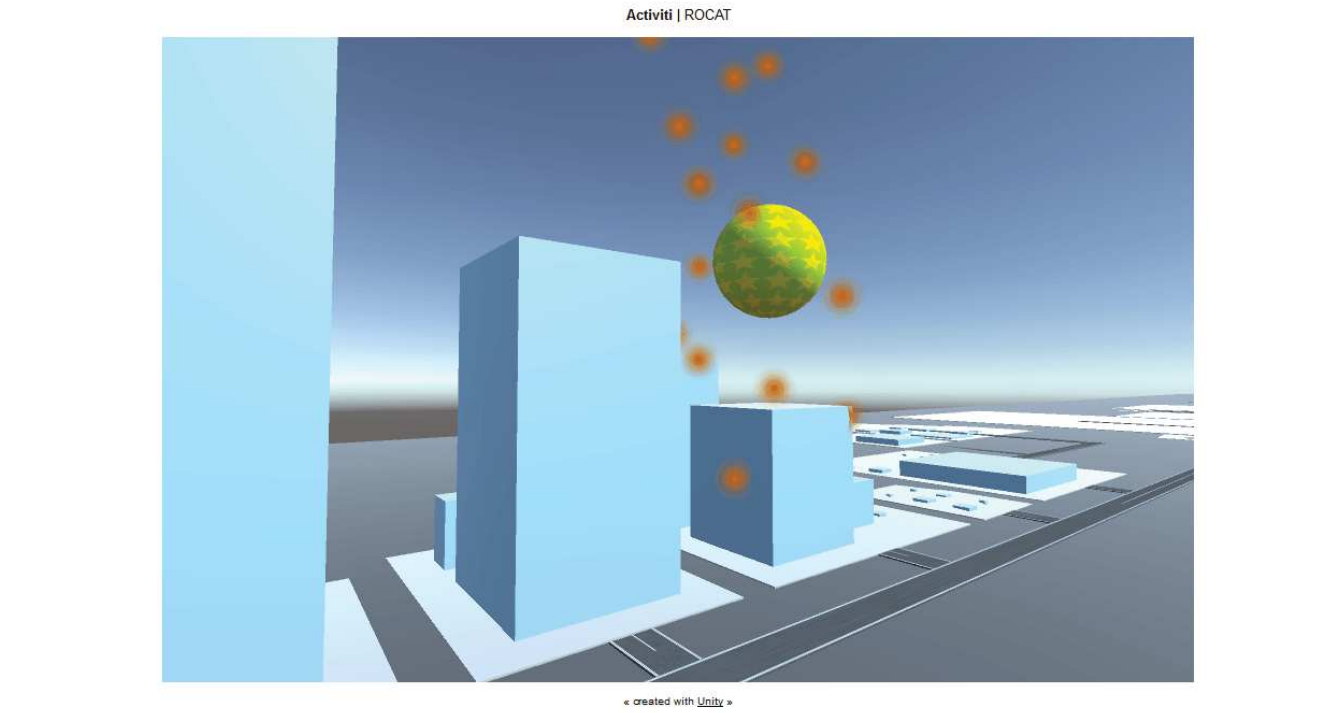

Fig. 4. Visualization for self-admitted technical debt

ACKNOWLEDGMENTS

This work has been supported by JSPS KAKENHI Grant Number 16H05857 and the JSPS Program for Advancing Strategic International Networks to Accelerate the Circulation of Talented Researchers: Interdisciplinary Global Networks for Accelerating Theory and Practice in Software Ecosystem (G2603). We would like to thank Yusuke Saito and Shin Fujiwara for their initial work on Rocat, and Takao Nakagawa and Stevche Radevski for their suggestions and advices.

REFERENCES

[1] R. Wettel, M. Lanza, and R. Robbes, "Software systems as cities: A controlled experiment," in *Proc. of 33rd Int. Conf. on Softw. Eng.*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 551–560. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985868

[2] G. Balogh and A. Beszedes, "Codemetropolis - code visualisation in minecraft," in *Proc. of 13th IEEE Int. Work. Conf. on Source Code Analysis and Manipulation*, ser. SCAM '13, Sept 2013, pp. 136–141.

[3] F. Steinbrückner and C. Lewerentz, "Representing development history in software cities," in *Proc. of 5th Int. Symp. on Softw. Visualization*, ser. SOFTVIS '10. New York, NY, USA: ACM, 2010, pp. 193–202. [Online]. Available: http://doi.acm.org/10.1145/1879211.1879239

[4] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: Transparency and collaboration in an open software repository," in *Proc. of 2012 ACM Conf. on Comput. Supported Cooperative Work*, ser. CSCW '12. New York, NY, USA: ACM, 2012, pp. 1277–1286. [Online]. Available: http://doi.acm.org/10.1145/2145204.2145396

[5] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proc. of 36th Int. Conf. on Softw. Eng.*, ser. ICSE '14. New York, NY, USA: ACM, 2014, pp. 345–355. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568260

[6] E. Pietroni and A. Adami, "Interacting with virtual reconstructions in museums: The etruscanning project," *J. Comput. Cult. Herit.*, vol. 7, no. 2, pp. 9:1–9:29, 2014.

[7] T. Menzies and T. Zimmermann, "Software analytics: So what?" *IEEE Softw.*, vol. 30, no. 4, pp. 31–37, Jul. 2013. [Online]. Available: http://dx.doi.org/10.1109/MS.2013.86

[8] R. Minelli and M. Lanza, "Samoa – a visual software analytics platform for mobile applications," in *Proc. of 29th IEEE Int. Conf. on Softw. Maintenance*, ser. ICSM '13, Sept 2013, pp. 476–479.

[9] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proc. of 35th Int. Conf. on Softw. Eng.*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 712–721. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486882

[10] K. Fujiwara, H. Hata, E. Makihara, Y. Fujihara, N. Nakayama, H. Iida, and K. Matsumoto, "Kataribe: A hosting service of historage repositories," in *Proc. of 11th Work. Conf. on Mining Softw. Repositories*, ser. MSR '14. New York, NY, USA: ACM, 2014, pp. 380–383. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597125

[11] 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*, ser. IWPSE-EVOL '11. New York, NY, USA: ACM, 2011, pp. 96–100. [Online]. Available: http://doi.acm.org/10.1145/2024445.2024463

[12] A. Potdar and E. Shihab, "An exploratory study on self-admitted technical debt," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, Sept 2014, pp. 91–100.