Interactive and Collaborative Source Code Annotation

Ryo Suzuki
University of Tokyo
Tokyo, Japan
1253852881@mail.ecc.u-tokyo.ac.jp

ABSTRACT

Software documentation plays an important role in the collaborative software development. A good documentation makes the source code easier to understand; on the other hand, developers have to constantly update it whenever they change the source code. This paper proposes Cumiki, a web-based interactive and collaborative code annotation tool that generates documentation which is closely associated with the source code. Cumiki makes much easier to keep the documentation up-to-date by automated updating based on the commit history of Git version data. In this paper, we explain how our system works and discuss why it is useful to foster collaboration between programmers.

Categories and Subject Descriptors

D.2.3, D.2.6, D.2.9 [**Software Engineering**]: Coding Tools and Techniques, Programming environments; H.5.2 [**User Interfaces**]: Training, Help, and Documentation

General Terms

Documentation, Human Factors, Design

Keywords

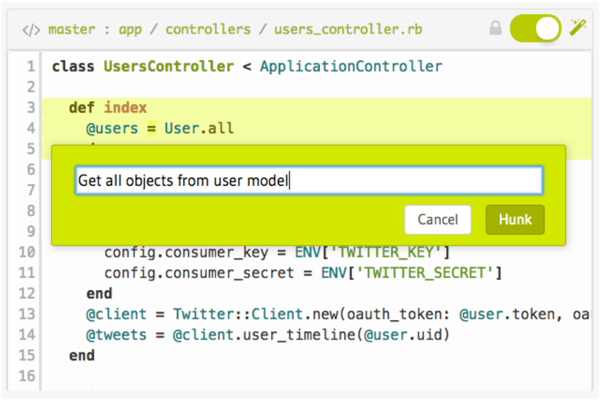
Source Code Annotation, Program Understanding, Collaborative Software Development

1. INTRODUCTION

Software documentation is helpful for programmers to understand how features of software are implemented. To make a good and easy-to-read documentation, developers should not only keep it up-to-date but also add traceable link between code snippets and source code. Several ideas has been proposed to assist programmers to create documentation effectively. One solution is an embedded documentation generator. This kind of tools such as RDoc [1] and JSDoc [2] enables to generate software documentation from a set of specially commented source code files. In addition, it has also been proposed that the information retrieval can be used to link between source code and documentation [1]. This approach has the great benefit of making it easier to keep documentation up-to-date, however, there is a limitation that none but the owner of the code can edit documentation.

[1]http://rdoc.sourceforge.net/
[2]http://usejsdoc.org/

ICSE '15, May 16 – 24, 2015, Firenze, Italy
ACM 978-1-4503-2768-8/14/05 ..$15.00.



```
class UsersController < ApplicationController

  def index
    @users = User.all
```

Get all objects from user model

```
      config.consumer_key = ENV['TWITTER_KEY']
      config.consumer_secret = ENV['TWITTER_SECRET']
    end
    @client = Twitter::Client.new(oauth_token: @user.token, oa
    @tweets = @client.user_timeline(@user.uid)
  end
```

Figure 1: Interface for source code annotation

In this paper, we explore another approach through the creation of Cumiki, a web-based source code annotation tool. Cumiki enables users to generate documentation that is closely integrated with the source code hosted on GitHub. Moreover, our interactive annotation system makes it possible that crowds of developers annotate the source code without editing the source code. Therefore, it is likely that they are able to share tips, ask and answer questions and accumulate their knowledge on the source code interactively and collaboratively.

2. RELATED WORK

There is growing interest in the tools that support collaborative software development. Collabode [2, 3] is a browser-based collaborative programming editor. Collabode provides an environment that enables multiple users to edit source code synchronously. The other form of collaboration is crowdsourcing. In crowdsourcing, CrowdCode [6] is an online Integrated Development Environment (IDE) for microtask programming that enables crowds of developers to write code. Moreover, Stack Overflow [3] is one of the examples of successful crowdsourcing in software engineering.

[3]http://stackoverflow.com/

Prior study shows that over 92% of questions are answered in a median time of 11 minutes [7]. In the literature of IDE, several systems have been proposed to integrate rich contextual information into IDE. Codelets [8] is an online code editor that has an interactive helper widget to assist the user in understanding and integrating examples on the web. Codetrail [4] and HyperSource [5] embrace an idea that connects source code and online resources such as documentation, examples, error descriptions, and code snippets.

3. IMPLEMENTATION

Cumiki is implemented as a web application on Amazon EC2 [4], providing an infrastructure for a scaling application. The user interface is implemented in HTML and JavaScript and runs entirely within the browser. The server side of Cumiki is implemented in Ruby. The system is closely integrated with a GitHub repository and users can annotate source code of any public repository hosted on GitHub. For extracting information from a git repository via Ruby, we use rugged [5] Ruby library. As a web-based annotation tool, Cumiki has the following features.

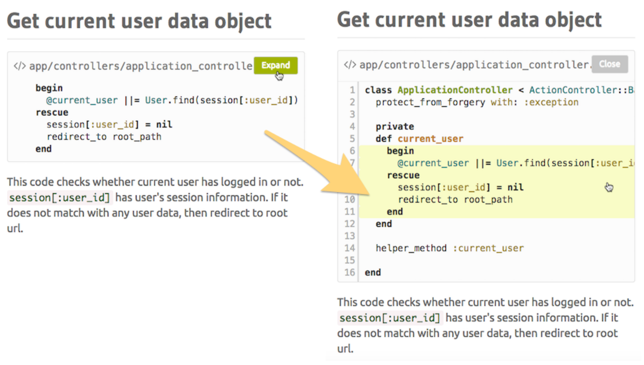
Interactive and collaborative annotation: JavaScript-based user interface enables users to annotate the code by simply dragging the mouse. After annotating the source code, Cumiki automatically generate documentation with code snippets. The user can edit the annotation with a rich content such as code blocks, images, videos, and even mathematical equations. These rich contextual information makes it easier to understand how the code works. Unlike the embedded documentation generator, we take an approach to separate code and documentation. Therefore, different developers can annotate the same source file, as a result, crowds of developers can share and accumulate their knowledge about the source code.

Traceability and automated updating: We implement a one-click link that associates between source code and documentation. As Figure.2 shows, when a button is clicked, Cumiki shows the entire code and highlights a certain piece of code. Moreover, our system is able to automatically update the code snippet by analyzing how the code has changed with the information of git versioning system. This feature frees from worry about updating documentation continuously. The mechanism behind this feature is that Cumiki extracts meta data such as line numbers, file name, and commit id from Git repository, and calcuarate based on the diff data of commit history.

4. USAGE SCENARIO

We consider the following usage scenarios in which this system can be the most effectively used. First use case is in a large open source software project. Separation between an annotation and source code makes it easier for developers to Second scenario is in an organization. Our system can be also useful to promote effective knowledge sharing regarding with software development within a group. Finally, Cumiki can also be used for educational purpose by assisting instructors to create a step-by-step tutorial. In general, it is difficult for students without coding experience to read the code, but our system can help them to understand the source code by highlighting an important point.

[4]http://aws.amazon.com/ec2/
[5]http://rubygems.org/gems/rugged/


Get current user data object
</> app/controllers/application_controlle
Expand
Get current user data object
</> app/controllers/application_controller.
Close

```
begin
  @current_user ||= User.find(session[:user_id])
rescue
  session[:user_id] = nil
  redirect_to root_path
end
```

This code checks whether current user has logged in or not. `session[:user_id]` has user's session information. If it does not match with any user data, then redirect to root url.

```
class ApplicationController < ActionController::B
  protect_from_forgery with: :exception

  private
  def current_user
    begin
      @current_user ||= User.find(session[:user_i
    rescue
      session[:user_id] = nil
      redirect_to root_path
    end
  end

  helper_method :current_user

end
```

This code checks whether current user has logged in or not. `session[:user_id]` has user's session information. If it does not match with any user data, then redirect to root url.

Figure 2: link between code and documentation.

5. CONCLUSION

This paper introduces an interactive code annotation tool for collaborative documentation creation. Our contribution is to propose a concept of social and collaborative source code annotation, and to implement the system that makes it possible that crowds of developers can annotate the source code without conflicting each other. Finally, we discuss our system can be used not only for sharing the knowledge within a group of developers but also for the purpose of education.

6. ACKNOWLEDGEMENT

This work has been supported by the JSPS KAKENHI grant numbers 25-10499.

7. REFERENCES

[1] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *Software Engineering, IEEE Transactions on*, 28(10):970–983, 2002.
[2] M. Goldman, G. Little, and R. C. Miller. Collabode: collaborative coding in the browser. In *Proc. of CHASE'11*, pages 65–68. ACM, 2011.
[3] M. Goldman, G. Little, and R. C. Miller. Real-time collaborative coding in a web ide. In *Proc. of UIST'11*, pages 155–164. ACM, 2011.
[4] M. Goldman and R. C. Miller. Codetrail: Connecting source code and web resources. *Journal of Visual Languages & Computing*, 20(4):223–235, 2009.
[5] B. Hartmann, M. Dhillon, and M. K. Chan. Hypersource: bridging the gap between source and code-related web sites. In *Proc. of CHI'11*, pages 2207–2210. ACM, 2011.
[6] T. D. LaToza, W. B. Towne, C. M. Adriano, and A. van der Hoek. Microtask programming: Building software with a crowd. In *Proc. of UIST'14*, pages 43–54. ACM, 2014.
[7] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann. Design lessons from the fastest q&a site in the west. In *Proc. of CHI'11*, pages 2857–2866. ACM, 2011.
[8] S. Oney and J. Brandt. Codelets: linking interactive documentation and example code in the editor. In *Proc. of CHI'12*, pages 2697–2706. ACM, 2012.