Interactive and Collaborative Source Code Annotation

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

***Abstract*—Software documentation plays an important role in sharing the knowledge behind source code between distributed programmers. Good documentation makes source code easier to understand; on the other hand, developers have to constantly update the documentation whenever the source code changes. Developers will benefit from an automated tool that simplifies keeping documentation up-to-date and facilitates collaborative editing. In this paper, we explore the concept of collaborative code annotation by combining the idea from crowdsourcing. We introduce Cumiki, a web-based collaborative annotation tool that makes it easier for crowds of developers to collaboratively create the up-to-date documentation. This paper describes the user interface, the mechanism, and its implementation, and discusses the possible usage scenarios.**

I. Introduction

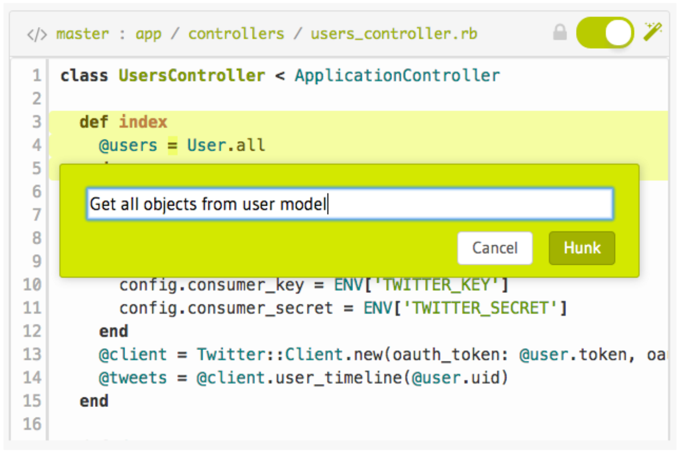
Software documentation is helpful for programmers to understand how features of software are implemented. To make a good and easy-to-read documentation, developers should not only keep it up-to-date but also add traceable link between code snippets and source code. Several ideas have been proposed to assist programmers to create documentation effectively. One solution is an embedded documentation generator. This kind of tools such as RDoc [1] enables to generate software documentation from a set of specially commented source code files. In addition, it has also been proposed that information retrieval can be used to link between source code and documentation [1]. Developers will benefit from an automated tool that (1) simplifies keeping documentation up-to-date and (2) facilitates collaborative editing.

In this paper, we explore the different solution through converting the documentation problem to a crowd-sourcing problem. We propose Cumiki [2], a web-based source code annotation tool, that 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.

II. Related Work

There is growing interest in the tools that support collaborative software development. Collabode [2] is a browser-based collaborative programming editor. Collabode provides an environment that enables multiple users to edit source code synchronousely. The other form of collaboration is crowdsourcing. In crowdsourcing, CrowdCode [5] 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. Prior study shows that over 92% of questions are answered in a median time of 11 minutes [6]. In the literature of IDE, several systems have been proposed to integrate rich contextual information into IDE. Codelets [7] is an online code editor that has an interactive helper widget to assist the user in understanding and integrating examples on the web. Codetrail [3] and HyperSource [4] embrace an idea that connects source code and online resources such as documentation, examples, error descriptions, and code snippets.

[1]http://rdoc.sourceforge.net/
[2]http://cumiki.com/



```
class UsersController < ApplicationController

  def index
    @users = User.all

      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

```

Get all objects from user model

Fig. 1. Interface for source code annotation

III. Implementation

Cumiki is implemented as a web application on Amazon EC2 and its user interface is built with HTML and JavaScript and runs entirely within the browser. 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, we use the rugged

[3]http://stackoverflow.com/

[4] 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 clicking and dragging. 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.

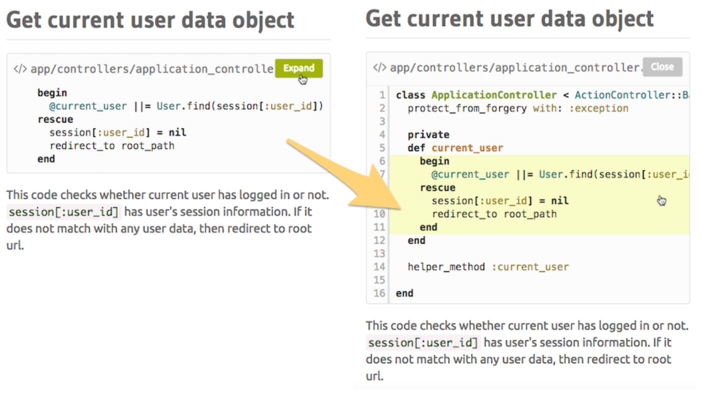
Traceability and automated updating: When a user finish annotating the code in browser, then the system saves the information of git commit id, git blob id, the file path, and the line number of the code snippet. When the linked file is updated, the system compares the diff of git commit history, and then calculates the change of the line number to adjust the positions of the code snippet. This mechanism provides two useful features. First, a user can see the association 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. Second, it also allows users to keep documentation up-to-date even when editing the code, changing the file path, and even merging with other git branches.

IV. 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.

V. 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.



Get current user data object

</> app/controllers/application_controlle

```
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.

Get current user data object

</> app/controllers/application_controller.

```
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.

Fig. 2. link between code and documentation.

Acknowledgement

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

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. Real-time collaborative coding in a web ide. In *Proc. of UIST'11*, pages 155–164. ACM, 2011.

[3] M. Goldman and R. C. Miller. Codetrail: Connecting source code and web resources. *Journal of Visual Languages & Computing*, 20(4):223–235, 2009.

[4] 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.

[5] 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.

[6] 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.

[7] 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.