CodeSketch: Drawing Diagrams in Source Code

Reza Adhitya Saputra
University of Waterloo
radhitya@uwaterloo.ca

Raminder Sodhi
University of Waterloo
rjsodhi@uwaterloo.ca

ABSTRACT

A picture is worth a thousand words. Similarly, software developers sketch diagrams for various purposes. The diagrams they draw can be flowcharts, architecture diagrams, or algorithm descriptions. We develop a tool called CodeSketch based on diagramming practice in Open Source Software (OSS) development. Using CodeSketch, a developer can draw a diagram consists of rectangles, lines, arrows, and texts. The tool then converts the diagram to ASCII art which is directly embedded in source code. Compared to other diagram representation, created diagrams using CodeSketch can coexist with source code files, they do not require specific graphics viewer tool other than a text editor, and they are easy to be stored in a revision control system. Finally, we conduct a user study where participants can use our tool followed by collecting their feedback.

Keywords

Source code comments, documentation, diagrams, ASCII art

1. INTRODUCTION

Diagrams are important visual representation which are widely used in design arts, engineering, and science. The use of diagrams and images is also pervasive in software development since diagrams are viewed as alternatives to verbal communication. Furthermore, diagrams enables developers to visualize concepts and communicate ideas. For example, a developer would doodle diagrams on scrap papers or whiteboards during a discussion with their coworkers. In more formal purposes, diagrams can be created using a specific designer tool. However, the designer tool restricts the developer to use a particular framework so the created diagrams cannot be exported easily to other platforms with different development tools.

In co-located software development, diagrams are used during discussions, meetings, or presentations. However, these diagrams have transitory nature [5]. One reason is that because they only serve a purpose to explain concepts to coworkers. After the coworkers understood, sketches on whiteboard will be erased and diagrams on scrap papers would be thrown into garbage bins. Although these diagrams can be converted digitally for long term purposes, such as code documentation, redrawing these diagrams is wearisome.

In Open Source Software (OSS) development, developers avoid creating a lot of bitmap-based diagrams on documentation or wiki pages. This is because they do not bother to keep these diagrams be updated when they changed their source code [13]. Nevertheless, if they must draw diagrams to communicate their ideas, they tend to use ASCII art. Although ASCII art is a simple form of diagrams, it has a practical benefit compared to other graphical representations: ASCII art enables developers not to depend on a specific graphical tool. Furthermore, in an open source project every developer may not use the same tools. Therefore, a tool-independent diagram representation is required.

This project focuses on OSS settings and the use of ASCII art. We developed a diagramming tool to create ASCII art based on criteria we have collected during literature study. The created diagrams can be edited easily and viewed without a specific tools. We did a user study conducted on (X) participants. The user study consists of a number of exercises on which participants can try to use our tool, then followed with questionnaire. As a final step, we gather their interesting feedback.

2. RELATED WORK

Our proposed idea is related to why developers draw diagrams and creating diagrams as source code comments.

Why developers draw diagrams. In co-located teams, diagrams help developers to understand source code, starting discussions, improve documentations and they can be used as presentation aids during a meeting [5]. In OSS community, developers agree that diagramming is useful but they do not like to redraw diagrams after they updated their source code [13, 6]. If they must draw diagrams, they tend to use ASCII art since they have low cost in creation. Moreover, they prefer to put diagrams inside a revision control system, so everyone can keep track changes. In another study, the use of ASCII art in bug reports also has been observed [12].

Creating diagrams as source code comments. GUIIO is a tool to create a GUI mockup embedded in source code [11]. Similar to our tool, they also use ASCII art. The mockup is used as an explanation to source code snippets which actually implement the actual GUI. They argue that the approach can reduce window switching between source code and designer tool. However, GUIIO cannot be used to draw more general diagrams, e.g, flowcharts.

3. SYSTEM DESIGN

We summarize nine criteria of an ideal diagramming tool based on literature study of previous research of diagramming practice in OSS development [13, 6]. Subsequently, table 1 shows the design decision of CodeSketch based on these criteria. Note that CodeSketch does not satisfies all criteria.

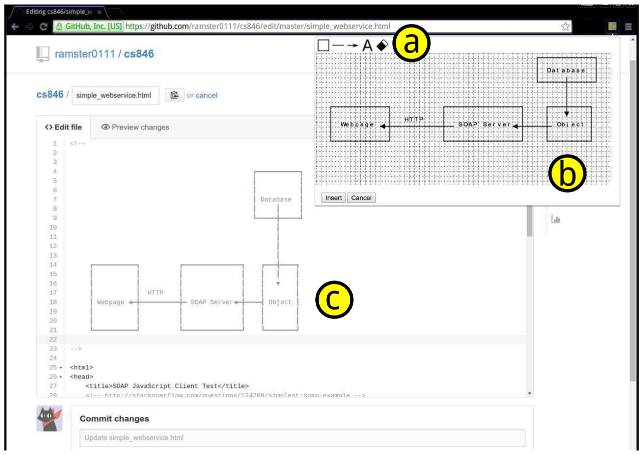
1. Diagrams coexist with existing communication tools.
2. The diagramming tool is widely available and free.
3. Diagrams can be easily published based on who the intended audience is.
4. The diagramming tool can be easily integrated into development infrastructure.
5. Diagrams are easily to be stored in a revision control system.
6. The ”source code” of diagrams can be shared, for example XML-like language for GUI design.
7. Diagrams are not tedious to create and edit (Possibly fully automated).
8. Diagrams do not require specific tools to create and display.
9. A change of a diagram can be observed, similar to diff tool

Criteria - Supportability	Design Decision
1 - Yes	Diagrams created using CodeSketch should be easily embedded into various text-based communication tools, such as mailing list, IRC, bug reports, and code review tools. Because the diagrams are text-based, this criteria can be fulfilled.
2 - Yes	We design CodeSketch as a plugin for Github to be a free web-based app.
3 - No	We represent diagrams with unicode characters, so we cannot satisfy this criteria. For example, yhe diagrams are sufficient enough for explaining code to coworkers, but unsuitable for a formal meeting e.g. with stakeholders or end users.
4 - Yes	We implement CodeSketch as a web based tool and will be embedded as source code comment, so it is platform independent.
5 - Yes	Our diagrams are text-based. Unlike binary images, they are compatible with any revision control system.
6 - Yes	The diagrams themselves are source code composed of Unicode characters.
7 - No	An ambitious goal is to automate the creation of diagrams. However, our tool is not designed to encourage developers draw more diagrams or to make diagramming easier.
8 - Partially	ASCII diagrams require a specific tool to be created, since manually placing every character would be cumbersome. However our diagrams can be viewed without using a specific viewer tool.
9 - No	Diff tool can be used to compare two ASCII diagrams. However, diff tool is line based so a single character editing will highlight an entire line.

Table 1: CodeSketch's design decision

4. IMPLEMENTATION

As can be seen in figure 1 Our prototype of CodeSketch is a chrome extensions built using HTML5 canvas and Javascript.


Editing cs846/simple_w
GitHub, Inc. [US] https://github.com/ramster0111/cs846/edit/master/simple_webservice.html
ramster0111 / cs846
cs846 / simple_webservice.html
or cancel
Edit file
Preview changes
A
a
Database
Webpage
HTTP
SOAP Server
Object
b
Insert
Cancel
c
Commit changes
Update simple_webservice.html

```
<!--

-->

<html>
<head>
    <title>SOAP JavaScript Client Test</title>
    <!-- http://stackoverflow.com/questions/124269/simplest-soap-example -->
```

Figure 1: CodeSketch Interface. (a) a drawing toolbar. (b) a drawing canvas. (c) generated ASCII art as a HTML comment.

We provide a drawing canvas where the user can draw diagrams with the help of drawing tools likes rectangles, lines, arrow lines, text and eraser. The drawing diagram can then be embedded in ASCII format at the current cursor position in GitHub code edit boxes. We also keep the UI to be simple to make the tool easier to use.

5. EVALUATION

To investigate diagrams in source code and answer our research questions, we designed a study in which participants were asked to draw diagrams using our tool for 4 different exercises and then answer a set of questions following it. In the beginning of the survey we gave participants time to get comfortable with the tool. Demonstrations of some basic patterns like loops was done so that the users are able to complete the tasks comfortably. The survey consisted of 3 drawing tasks and a comparison task. The drawing tasks consisted of drawing a diagram for linked list traversal, a class diagram and an architecture diagram for any system with which the users are most familiar with. The comparison task was a comparison between text based documentation and text + image based documentation.

5.1 Drawing Task 1

In this task the participants were presented with a simple link list traversal function and were asked to draw meaningful drawings. Majority of the participants gave importance to the data structure being used while some concentrated more on the input and output of the function.

5.2 Drawing Task 2

In this task the participants were asked to draw a class diagram for any scenario they want in the animal/plant kingdom. We noticed that the majority of the people concentrated more on relationships between the classes, little importance was given to member functions and members as such.

5.3 Drawing Task 3

In this task the participants were asked to draw system they know the most about. We observed that all the participants were successful in drawing basic interactions between the various components of these system. The representations were clear and concise.

5.4 Comparison Task

For the comparison task the participants were asked their feedback on code understanding based on text based documentation against ASCII image based documentation. They were first ask to see the code and think loud about it, after going through the code they were asked to see the both the documentation and their feedback was taken. In general the feedback was that the image based documentation was helpful in visualizing the scenario better. The participants expressed that on just seeing the image they could get a general idea on what is happening in the code in comparison to textual documentation.

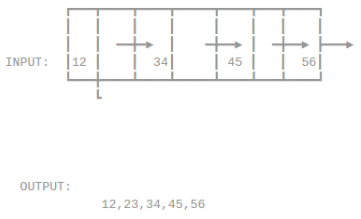


Figure 2: Drawing Task 1 *(drawn by P2)*

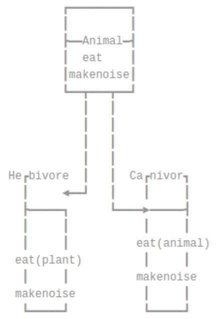


Figure 3: Drawing Task 2 *(drawn by P4)*

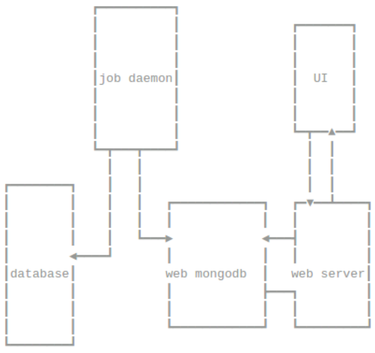
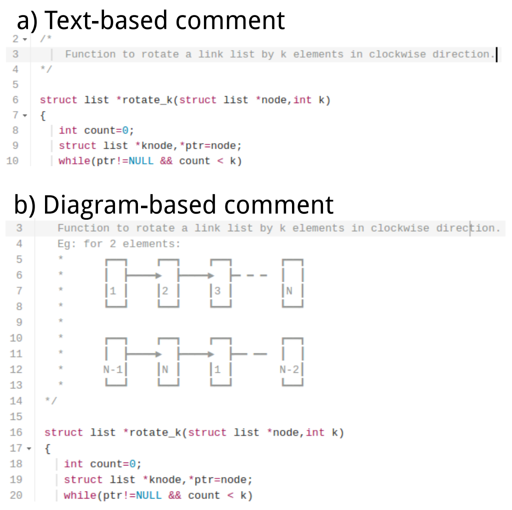


Figure 4: Drawing Task 3 *(drawn by P3)*



a) Text-based comment

```
/*
  | Function to rotate a link list by k elements in clockwise direction.
*/

struct list *rotate_k(struct list *node,int k)
{
  | int count=0;
  | struct list *knode,*ptr=node;
  | while(ptr!=NULL && count < k)
```

b) Diagram-based comment

```
   Function to rotate a link list by k elements in clockwise direction.
   Eg: for 2 elements:
   *
   *
   *
   *
   *
   *
   *
   *
   *
 */

struct list *rotate_k(struct list *node,int k)
{
  | int count=0;
  | struct list *knode,*ptr=node;
  | while(ptr!=NULL && count < k)
```

Figure 5: Comparison Task

6. DISCUSSION

7. LIMITATIONS AND FUTURE WORK

Although we receive positive feedback from participants about the use of CodeSketch, our evaluation does not answer how embedding diagrams improve code understanding. Therefore, we would like to know the effectiveness of embedding diagrams in various settings, for example, in code reviews and bug reports. When reviewing source code, developers have a problem in understanding code which they do not own. The lack of understanding makes them difficult to find defects on the source code [1]. A similar problem is also found in bug reports where reporters do not give much explanation [2]. We also would like to do comparison with a completely different approach called Literate Programming [9]. In contrast to diagramming, literate programming also aims for code understanding through incorporating compilable source code to literature writing.

Apart from code understanding, we also have several other future work in mind. First, we would like to investigate how can CodeSketch reduce task switching. Embedded diagrams in source code does not need window switching between code and diagrams, which is the design goal of CodePad [10]. Second, we feel that it is necessary to develop a way for easier diagramming. Ideally, editing and updating ASCII diagrams should not be cumbersome, or even fully automatic. Lastly, developing a customized diff tool to visualize how a diagram changes over time would be an interesting research direction. We envision the diff tool would be similar to Nonlinear Revision Control for Images [4].

8. CONCLUSIONS

9. REFERENCES

[1] A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, Piscataway, NJ, USA, 2013. IEEE Press.

[2] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '08/FSE-16, New York, NY, USA, 2008. ACM.

[3] A. Bragdon, R. Zeleznik, S. P. Reiss, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. LaViola, Jr. Code bubbles: A working set-based interface for code understanding and maintenance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, New York, NY, USA, 2010. ACM.

[4] H.-T. Chen, L.-Y. Wei, and C.-F. Chang. Nonlinear revision control for images. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, New York, NY, USA, 2011. ACM.

[5] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko. Let's go to the whiteboard: How and why software developers use drawings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, New York, NY, USA, 2007. ACM.

[6] E. Chung, C. Jensen, K. Yatani, V. Kuechler, and K. Truong. Sketching and drawing in the design of open source software. In *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on*, Sept 2010.

[7] GitHub. Github, 2015.

[8] M. Goldman, G. Little, and R. C. Miller. Real-time collaborative coding in a web ide. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, New York, NY, USA, 2011. ACM.

[9] D. E. Knuth. Literate programming. *The Computer Journal*, 27, 1984.

[10] C. Parnin, C. Görg, and S. Rugaber. Codepad: Interactive spaces for maintaining concentration in programming environments. In *Proceedings of the 5th International Symposium on Software Visualization*, SOFTVIS '10, New York, NY, USA, 2010. ACM.

[11] J. Simpson and M. Terry. Embedding interface sketches in code. In *Proceedings of the 24th Annual ACM Symposium Adjunct on User Interface Software and Technology*, UIST '11 Adjunct, New York, NY, USA, 2011. ACM.

[12] M. Twidale and D. Nichols. Exploring usability discussions in open source development. In *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on*, Jan 2005.

[13] K. Yatani, E. Chung, C. Jensen, and K. N. Truong. Understanding how and why open source contributors use diagrams in the development of ubuntu. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, New York, NY, USA, 2009. ACM.