Short-Term Revisit during Programming Tasks

Xiaoyu Jin, Nan Niu
Department of Electrical Engineering and Computing Systems
University of Cincinnati
Cincinnati, OH, 45221, USA
jinxu@mail.uc.edu, nan.niu@uc.edu

***Abstract*—Previous studies of Web page revisitation were only focused on long-term revisit ranging from hours to days. In this paper, we study the short-term revisit of less than one hour such as the revisit behavior during a small programming task. We first perform an exploratory study to observe the short-term revisit phenomenon. We then perform controlled experiments with our designed tool support as treatment by inviting 20 biomedical software developers to perform two software change tasks. Our results show that the participants with tool support used 19.7% less time than the ones without tool support.**

***Keywords*-revisitation; foraging theory; software change tasks; end-user programming**

I. Introduction

Programmers constantly need to go back and revisit various kinds of documents during programming, which could be inefficient thus hurting productivity [9]. Previous effort on revisit behavior mainly focuses on 4 aspects: (1) high percentage of revisit behavior occupied in the overall browsing history, (2) various revisit patterns, (3) reasons behind the revisit behavior, and (4) revisit prediction. Previous studies consistently found a high revisit rate ranging from 45% to 81% during Web browsing and usage [4]–[6], indicating the revisit behavior is an important research topic. However, these studies were all based on log analysis for a long time span ranging from days to months. We would like to zoom in to further study the short-term revisit behavior within several hours during end-user developers' everyday programming tasks. In this work, we first perform an exploratory experiment to observe the revisit behavior and further develop tool support. We then perform controlled experiments with our designed tool support as a treatment. The results show that our preliminary tool support reduced 19.7% of programmers' time in finishing their tasks.

II. Related Work

The study of revisit behavior started with the focus of the rates that the revisits are occupied during searching and browsing. Although previous studies reported varying revisit rates of 58% [10], 81%[5], and 51%[6], we can draw a consistent conclusion that revisit is a constant and repetitive behavior worth research attentions. Obendorf *et al.* [7] categorized revisit behavior according to the time span. As shown in Fig. 1, they reported the categories of revisit as short-term revisit, medium-term revisit, and long-term revisit along with the observed proportions for each category. They

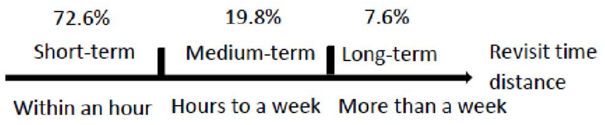
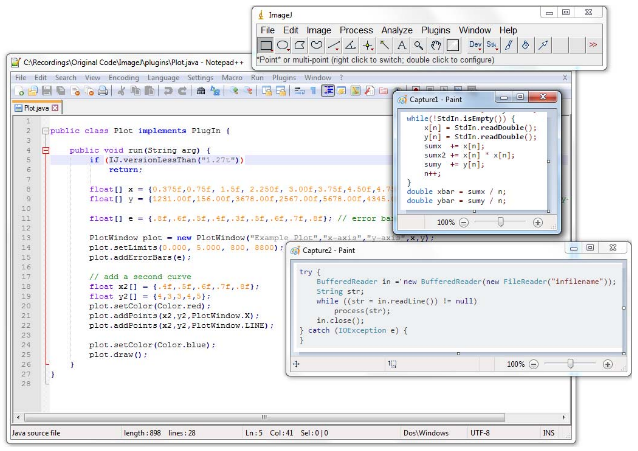


Fig. 1. Categories of revisit behaviors (adopted from [7])

also identified users' strategies for each category. For short-term revisits, people switch between windows or tabs instead of navigating back and forth to achieve fewer revisits and page requests. For medium-term revisits, direct access strategies (URL-entry, bookmark selection) were most frequently used. Long-term revisits generally aim to rediscover content accessed earlier, and hyperlinks initiated the most long-term revisitations. Although they concluded that users encountered severe problems in the category of long-term revisits, we study short-term revisit in this paper. Our rationale is that even though people have no problem re-finding previously visited information after a short while, the time used for revisit can still be reduced whereby improving overall efficiency, especially for programming, which involves large volume of information digestion and highly intelligent challenges.

III. Exploratory Study

Our first objective is to investigate how the revisit behavior looks like during end-user developers' programming. Specifically, we invited five bioinformatics researchers to perform the same software change task we designed. Each participant was given one hour to perform the task, and the process was recorded as a video. By analyzing these videos, we have three observations. First, if a participant revisited an entity, he would probably revisit it more times later. Second, even though participants have no difficulty finding the entity information previously visited, the process can be costly sometimes. Since there are always many windows, tabs, and pages opened when performing the change task, the participants may continuously go to the wrong places to find information and sometimes even get lost and forget what he wanted initially. Third, the revisit behavior itself is essential in finishing the task since it follows participants' inner goal, but the context switch can interrupt participant's flow. It takes time for the participant to refocus on the source code file that he was working on and to figure out which step he was in. The longer the participant was deviated by the revisit, the longer he would need to re-concentrate on the source code file to be edited.


ImageJ
File Edit Image Process Analyze Plugins Window Help
Point or multi-point (right click to switch; double click to configure)
C:\Recordings\Original Code\ImageJ\plugins\Plot.java - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Plot.java
Capture1 - Paint
Capture2 - Paint
100%
Java source file length : 898 lines : 28 Ln : 5 Col : 41 Sel : 0 | 0 Dos\Windows UTF-8 INS

```
public class Plot implements PlugIn {

    public void run(String arg) {
        if (IJ.versionLessThan("1.27t"))
            return;

        float[] x = {0.375f,0.75f, 1.5f, 2.250f, 3.00f,3.75f,4.50f,4.7
        float[] y = {1231.00f,156.00f,3678.00f,2567.00f,5678.00f,4345.

        float[] e = {.8f,.6f,.5f,.4f,.3f,.5f,.6f,.7f,.8f}; // error ba

        PlotWindow plot = new PlotWindow("Example Plot","x-axis","y-axis",x,y);
        plot.setLimits(0.000, 5.000, 800, 8800);
        plot.addErrorBars(e);

        // add a second curve
        float x2[] = {.4f,.5f,.6f,.7f,.8f};
        float y2[] = {4,3,3,4,5};
        plot.setColor(Color.red);
        plot.addPoints(x2,y2,PlotWindow.X);
        plot.addPoints(x2,y2,PlotWindow.LINE);

        plot.setColor(Color.blue);
        plot.draw();
    }
}
```

```
while(!StdIn.isEmpty()) {
    x[n] = StdIn.readDouble();
    y[n] = StdIn.readDouble();
    sumx  += x[n];
    sumx2 += x[n] * x[n];
    sumy  += y[n];
    n++;
}
double xbar = sumx / n;
double ybar = sumy / n;
```

```
try {
    BufferedReader in = new BufferedReader(new FileReader("infilename"));
    String str;
    while ((str = in.readLine()) != null)
        process(str);
    in.close();
} catch (IOException e) {
}
```

Fig. 2. A use case of our tool support

Based on the three observations, we designed a tool (Fig. 2) to help improve the revisit efficiency. There are two features for the tool. First, it utilized the screenshot tool and set the hot key to access the function conveniently. Second, the image generated by the screenshot tool has the feature of always-on-top of the screen, adapted from FileBox Extender [1]. Our tool can also support multiple windows simultaneously floating on top. Fig. 2 shows that the user can program on the source code file and always refer to the two small floating windows in the meantime. The design will reduce the redundant behavior in programming and keep programmer's flow smoothly. We expect the tool to be practical and effective in reducing the time cost of revisit because the tool is designed to address the three observations we made earlier.

IV. Controlled Experiment

After having a designed tool support that we named as EasyRevisit, we redesigned our experiment to integrate our tool as treatment and also added a second change task. The two change tasks were designed by adding features to two software systems named ImageJ [2] and StochKit [3] respectively. The aim is to test if EasyRevisit is effective during the actual software change tasks. We invited 20 biomedical researchers to perform the two change tasks. Each participant performed one task with EasyRevisit and the other without it. We counterbalanced both EasyRevisit-treatment order and the task order. We present our analysis and results next.

Tables I and II summarized the results for ImageJ task and StochKit task respectively. The results were generated by calculating the average and standard deviation of the 20 participants' data. The number of visits and revisits were calculated in two ways: nonredundant and redundant. Nonredundant means that we only count the distinct entities that a participant visited while redundant means we count all entities visited including the ones visited repeatedly. From the two tables, we can first draw some general conclusions that when there is no tool support, the time of revisit occupies about 20% of total time, and the number of revisits correspondingly occupies 25.9% to 32.5% out of total number of revisits.

TABLE I
Statistical results for ImageJ.

	Average # with EasyRevisit (without)	Average time (min) with EasyRevisit (without)
Total visits (nonredundant) Total visits (redundant)	32.4 (29.8) 73.2 (128.3)	39.2 (48.8)
Revisits (nonredundant) Revisits (redundant)	7.5 (6.8) 13.8 (33.2)	4.2 (10.3)
Revisits rate (nonredundant) Revisit rate (redundant)	23.1% (22.8%) 18.9% (25.9%)	10.7% (21.1%)

TABLE II
Statistical results for StochKit.

	Average # with EasyRevisit (without)	Average time (min) with EasyRevisit (without)
Total visits (nonredundant) Total visits (redundant)	17.3 (18.5) 66.4 (86.7)	54.8 (59.6)
Revisits (nonredundant) Revisits (redundant)	5.8 (5.5) 16.8 (28.2)	7.2 (10.3)
Revisits rate (nonredundant) Revisit rate (redundant)	33.5% (29.7%) 25.3% (32.5%)	13.1% (17.3%)

For ImageJ task, although the distinctly revisited entities increased only from 6.8 to 7.5 when using EasyRevisit, the total number of revisits decreased 58.4% from 33.2 to 13.8, which is significant. Correspondingly, the revisit time decreased 59.2% from 10.3 minutes to 4.2 minutes. The total task completion time also decreased from 48.8 minutes to 39.2 minutes. These decreased trends show the effectiveness of our tool support. We also noticed that the revisit time reduced 6.1 minutes while the total task completion time reduced 9.6 minutes. We speculate this is because the reduced revisit can further reduce distraction thus improving efficiency.

For StochKit task, however, the effect is not as significant as ImageJ task. The average revisit time decreased from 10.3 to 7.2 minutes and the average task completion time reduced only 8% from 59.6 to 54.8 minutes. We found the reason is probably because the complexity of StochKit task lies mainly in the original source code comprehension. The features to be added by participants are relatively simpler than ImageJ task. Participants need more external information to solve ImageJ task, while they mainly focus on source code for StochKit task. We label the two kinds of tasks as external information intensive task and internal information intensive task. We expect that our tool can facilitate former kind of task more effectively by reducing the time cost for revisit behavior.

In summary, our tool support is effective in improving the efficiency of revisit and overall task completion that it reduced 19.7% of programmers' time averagely in finishing their change tasks. The tool support is helpful especially when a programming task needs to refer largely to external information besides the original source code.

Acknowledgment

This research is partially supported by the U.S. National Science Foundation (Award CCF-1350487) and the National Natural Science Foundation of China (Fund No. 61375053).

REFERENCES

[1] C. Nicora, “FileBox eXtender” https://www.hyperionics.com/files/ Last accessed: February 2017.

[2] W.S. Rasband, C.A. Schneider, and K.W. Eliceiri, ”ImageJ: Image Processing and Analysis in Java.” https://imagej.nih.gov/ij/ Last accessed: February 2017.

[3] K.R. Sanft and S. Wu, M. Roh, J. Fu, R.K. Lim, and L.R. Petzold, ”StochKit: Stochastic Simulation Kit.” http://www.engineering.ucsb.edu/~cse/StochKit/ Last accessed: February 2017.

[4] P. Baldi, P. Frasconi, and P. Smyth, 2003. *Modeling the Internet and the Web: Probabilistic Methods and Algorithms.* John Wiley and Sons.

[5] A. Cockburn and B. McKenzie, 2001. What do Web users do? An empirical analysis of Web use. *International Journal of Human-Computer Studies,* 54(6), pp. 903-922.

[6] E. Herder, 2005. Characterizations of user Web revisit behavior. In *Proceedings of the Workshop on Adaptivity and User Modeling in Interactive Systems* (pp. 32-37). DFKI.

[7] H. Obendorf, H. Weinreich, E. Herder, and M. Mayer, 2007, April. Web page revisitation revisited: implications of a long-term click-stream study of browser usage. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 597-606). ACM.

[8] D.W. Stephens and J.R. Krebs, 1986. *Foraging theory.* Princeton University Press.

[9] N. Sawadsky, G.C. Murphy, and R. Jiresal, 2013, May. Reverb: Recommending code-related Web pages. In *Proceedings of the 2013 International Conference on Software Engineering* (pp. 812-821). IEEE Press.

[10] L. Tauscher and S. Greenberg, 1997. How people revisit Web pages: Empirical findings and implications for the design of history systems. *International Journal of Human-Computer Studies,* 47(1), pp.97-137.