Exploring the Relationship Between Programming Difficulty and Web Accesses

Duri Long
Georgia Institute of Technology
Atlanta, GA, USA
duri@gatech.edu

Kun Wang
UNC-Chapel Hill
Chapel Hill, NC, USA
wangk1@cs.unc.edu

Jason Carter
Cisco Systems-RTP
Raleigh, NC, USA
jasoncartercs@gmail.com

Prasun Dewan
UNC-Chapel Hill
Chapel Hill, NC, USA
dewan@cs.unc.edu

***Abstract*—This work addresses difficulty in web-supported programming. We conducted a lab study in which participants completed a programming task involving the use of the Java Swing/AWT API. We found that information about participant web accesses offered additional insight into the types of difficulties faced and how they could be detected. Difficulties that were not completely solved through web searches involved finding information on AWT/Swing tutorials, 2-D Graphics, Components, and Events, with 2-D Graphics causing the most problems. An existing algorithm to predict difficulty that mined various aspects of programming-environment actions detected more difficulties when it used an additional feature derived from the times when web pages were visited. This result is consistent with our observation that during certain difficulties, subjects had little interaction with the programming environment, they made more web visits during difficulty periods, and the new feature added information not available from features of the modified existing algorithm. The vast majority of difficulties, however, involved no web interaction and the new feature resulted in higher number of false positives, which is consistent with the high variance in web accesses during both non-difficulty and difficulty periods.**

Keywords— interactive programming environments, affective computing, distributed help, web foraging, intelligent tutoring

I. .INTRODUCTION

Automatic detection of programming difficulty could be used for several purposes including offering help to students and co-workers in academic and industrial contexts [1], identifying bugs [2, 3], determining the difficulty of tasks, programming constructs, and APIs [2, 4], and determining when automatic hints should be displayed by intelligent tutoring systems [5]. This field is part of the larger area addressing detection of task difficulty. A related subfield is detection of difficulty in web searches [6]. These two subfields have been investigated separately even though they are related in that web searches can be subtasks of programming tasks.

This work brings together these two fields by providing a preliminary answer to the following general question: What is the relationship between difficulty and web accesses in programming tasks that can benefit from web searches? We answer this general question by addressing the following related sub-questions regarding such tasks: To what extent do developers use web searches to solve problems and how successful are these searches in solving their problems? What kinds of problems are not resolved completely by such searches? How, to what extent, and why can information about web searches be used to improve an existing state of the art online algorithm for predicting and resolving programming difficulties? What kind of data can help answer these questions?

The remaining sections address these questions based on related work and data gathered from a study.

II. RELATED WORK AND BASELINE

A variety of features have been shown to correlate with programming difficulty and the related emotions of frustration and happiness in programming-based tasks. These features have been derived from a variety of information sources including the programming language constructs used [4], interaction with the programming environment [7], output of a Kinect camera [1], and output of eye-tracking electrodermal activity and electroencephalography sensors [3]. Some correlation algorithms have targeted code-comprehension [3], while others have considered code-creation [7]; and some have focused only on correlation [4], while others have also built models to predict difficulties of programmers [3, 7]. Some of predictive systems make inferences incrementally [7], during the task, while others do so after the completion of the task [3].

Features that correlate with and can be used to predict difficulty in web searches have, similarly, been drawn from a variety of information sources including users' search queries, clicks on search results, bookmarked web pages, mouse movements and scroll events, time spent on a web page, and task completion time [6]. Web searches can be performed as stand-alone tasks (such as reading information regarding a medical condition) or they can be sub-tasks of a larger task such as programming.

Web-supported programming has been studied in previous work. Fishtail is a system that recommends arbitrary web pages on the internet based on programming constructs on which the developer is working, but it does not have high accuracy [8]. Reverb is a similar system that achieves higher accuracy by restricting itself to web pages visited earlier by the developer [9]. To illustrate, in Reverb each method call in the current context is mapped to a set of keywords uniquely describing the call, and this set is used to match web pages.

Funded in part by NSF grant IIS 1250702

978-1-5386-4235-1/18/$31.00 ©2018 IEEE

The recommendations provided by Fishtail and Reverb, as well as search engines, consider only how relevant certain components of web pages are to the task at hand, which can be a function of several parameters including how well these components match a manually or automatically generated query, when the page was last updated, and, in case of Reverb, also when the developer last visited the page. Jin, Niu and Wagner [10] consider also the time cost of finding the relevant information in the returned page, which they argue is a function of two important features – the time taken on average to read the entire page and the shape of the expected foraging curve of the page, which plots information gain as a function of the time spent on the page. Each recommendation displayed to the user contains these two features to allow more efficient recommendations to be chosen. For an unknown page, a default foraging curve is displayed. For a known page, the foraging curve depends on whether it contains ranked answers (e.g. StackOverflow), a list of items (e.g. API documentation), a wiki/blog explaining a concept, or a forum with unranked answers. Users are expected to choose, for instance, a ranked answer site over a wiki.

The works above aim to reduce the time required to consult the web to solve software engineering problems. At least one study has found that some such problems – in particular, in web design – are not solved by web searches [11]. To the best of our knowledge, no previous work has used web accesses to predict difficulties with web-supported programming. Thus, our work addresses a new dimension in such programming.

As mentioned earlier, features other than web accesses have been considered by previous works for predicting programming difficulties. The only research that has addressed incremental prediction of difficulty in programming tasks involving code creation consists of several related algorithms, developed by our team, which mine logs of interaction with the programming environment and/or videos captured by a Kinect camera [1]. Only one of these algorithms [12] has an online implementation – that is, an implementation that processes live data, and thus can (and has) actually be used to detect, communicate, and ameliorate programming difficulty incrementally. Other algorithms have offline implementations validated using logged data. Because of space limitations, we use the online algorithm as the only baseline for determining the effectiveness of mining web accesses. We will refer to our implementation of this algorithm as the baseline system.

The baseline system is targeted at the Eclipse IDE and extends the Fluorite tool [13] to capture Eclipse commands. It divides the raw interaction into segments and calculates, for different segments, ratios of the following classes of commands: edit, debug, focus (in and out of the programming environment), and navigation within the programing environment. The training data for the algorithm is passed through the Weka SMOTE filter [14] to artificially increase the members of the minority class (difficulty). It uses the Weka J48 decision tree implementation to make raw inferences. Our algorithm operates on the assumption that the perception of difficulty does not change instantaneously. Therefore, it aggregates the raw predictions for adjacent segments to create the final prediction, reporting the dominant status in the aggregated segments. In addition, it makes no predictions from the first few events to ignore the extra compilation and focus events in the startup phase.

III. STUDY

An important case of web-supported programming is a task that involves the use of one or more APIs that are well documented on the web. Our study involved such a task – use of the Java AWT/Swing API to create an interactive graphical user interface. The main subtask was to create a program that composes a single or double-decker bus from rectangles and circles. To support input, the subjects were asked to allow users to provide keyboard input for moving the bus, and mouse input for converting a single-decker bus to a double-decker. An additional subtask, designed to pose algorithmic challenges, required subjects to draw a transparent square with yellow borders around the bus and ensure that the bus could not be moved outside the square.

Fifteen graduate and advanced undergraduate students at our university, many of whom had previously held industry internships, participated in the study. Each participant was given at least an hour and a half to complete as many subtasks as possible using Eclipse. On average, they spent about two hours working on the task. We have not yet determined to what extent and in what order they completed the subtasks. They were free to use the Internet to solve their problems. Our baseline system was used to capture interaction with Eclipse. A Firefox plug-in was used to determine web access information. As shown below, each web access contained three pieces of information – its time, the URL visited, and the input search string or exact URL that triggered the visit.

9/13/2013 16:20:47 PM
swing - Java keyListener - Stack Overflow| http://stackoverflow.com/questions/11944202/java-keylistener

During the task, the base system showed the programmers the predictions that it made. The difficulty notification was called "slow progress" and indicated that the programmers were making slower than normal or expected progress. Subjects were provided with a user interface through which they could correct a prediction and/or ask for help. When participants asked for help, they were instructed to indicate what they had done to solve the problem so far and discuss their issue with the third author or another helper. Help was given in the form of URLs to documentation or code examples. Given enough time, many difficulties can be eventually solved. We considered a difficulty insurmountable if the programmers did not think they could solve the problem within the given time constraints. In this study, help requests were considered insurmountable difficulties – other difficulties were considered surmountable. We used information about the predictions, corrections, and help requests to determine ground truth.

IV. WEB ACCESSES AND DIFFICULTY

To what extent did developers use web searches to solve their problems and how successful were these searches? To answer this question, we divided the web accesses of the subjects into web episodes, which are a series of web accesses with no intervening interaction with the programming environment. Let us assume that each web episode was used to address a set of related problems faced by the subject when the

episode started, and at the end of the episode the developer either felt the problem was solved, or they continued to face a surmountable or insurmountable difficulty. Let us also assume, conservatively, that if a difficulty segment had one or more web episodes, then the last web episode was not effective. This is a conservative estimate as the difficulty may not have resulted in a web episode. We found that the percentage of episodes that were ineffective was 8 percent, and the fraction of web episodes that were ineffective and led to insurmountable difficulties was 5 percent. Thus, while the vast majority of web episodes were successful, some did not solve the problem, and some even resulted in help requests, which motivates the design of both better web foraging tools and better tools for detecting difficulty.

What were the topics of unsuccessful searches? We found that the majority of sites visited during difficulties fell into one of three distinct categories: 1) API-related sites (primarily consisting of Java AWT/Swing introductory tutorials, sites related to how to represent 2D-graphics using Java AWT/Swing, sites discussing how to listen to keyboard, mouse, and button events, and sites related to non-graphical components of AWT/Swing such as JPanel), 2) design related sites (primarily Model-View-Controller tutorials), and 3) non-Swing/AWT Java related searches (mostly tutorials).

Table I shows the percentage of different types of topics searched during periods of difficulty, thereby characterizing the difficulty of these topics and/or the adequacy of the documentation about them. More importantly, it helps us understand the nature of some of the search-based difficulties faced by the programmers. It is consistent with user comments in [11] that some (uncharacterized) searches do not solve web-design problems. The fact that 2-D Graphics causes the most difficulty is likely because it involves overriding the paint() method and explicitly calling the repaint() method from a thread that is different from the one that executes the paint() method. We have not yet analyzed the videos to determine the exact causes of the difficulties.

TABLE I. CHARACTERIZING UNSUCCESSFUL WEB SEARCHES

Topic	Searched During Difficulties
API (overall)	**75%**
Tutorials	17%
2D-Graphics	38%
Events	10%
Components	10%
Design	**4%**
Java	**2%**
Other	**19%**

V. PREDICTIONS USING WEB ACCESSES

How, to what extent, and why could information about web searches improve our baseline online algorithm for predicting these difficulties? To answer this question, we extended the baseline algorithm with an additional feature representing the number of web links traversed during a segment - *weblinktimes*. We performed the following aggregate analysis to compare the two algorithms. The sizes of the warm-up phase and segments were fixed at 50 and 100 events, respectively, for both algorithms. Moreover, 5 segments were aggregated in both cases. We used 10-fold cross-validation on combined logs of all participants. Only 6% of the statuses were difficulties – that is, facing difficulty was a rare event, which is to be is to be expected if programmers are given tasks they are qualified to tackle with the available resources. The SMOTE filter was used to equalize the number of difficulty and non-difficulty segments in the training data.

10-fold cross-validation assumes all partitions are independent, which may not be the case in our study because the programmers can be expected to learn as they progress through the solution as they better understand the information gathered from the web visits. On the other hand, it is not clear this learning effect changes the pattern of interaction around difficulties – our online algorithm has been used in multiple lab studies and a field study [7] to successfully find difficulties. More importantly, different search strings and web episodes likely had some independence, and as Table I shows, the topics searched had a large degree of independence. Finally, 10-fold cross validation is more comprehensive than 1-fold validation and any dependence between the partitions should make our results conservative rather than inflated.

The true positive and negative rates of the baseline algorithm were 20% and 100%, respectively, that is, the baseline algorithm correctly predicted 20% of the difficulty statuses, and all of the non-difficulty statuses. The web-based algorithm increased the true positive rate substantially to 93%, but reduced the true negative rate slightly to 95%. Based on these two measures, the results are impressive. A more critical analysis is provided by the precision and recall measures. Recall is the same as the true positive rate. Precision is the fraction of flagged positives that were actual positives. The precision of the baseline and modified algorithms are 100% and 54%, respectively. This result is consistent with the inverse relationship seen between precision and recall in other work. The F-score has been devised to give equal weight to both and is defined as: $\frac{2 \times precision \times recall}{precision + recall}$. The F-scores of the baseline and modified algorithms are 33% and 71%, respectively. Intuitively, the better F-Score of the modified algorithm can be explained by considering detection of difficulties as being analogous to finding needles in a haystack. In our modified algorithm, one has to search not the whole haystack, but a subset of items whose size is about twice the number of actual needles, which arguably, is a very good result. Of course, how the two measures should be weighed depends on the cost of manually separating false and true positives, which in turn, depends on several factors including whether the helpers are face-to-face with the worker and how much context they have if they are distributed [15].

Why did the new feature make such a difference? We extended an interactive visualization tool we developed earlier [16] to help us understand the impact of weblinktimes in specific difficulties. The data of a participant is represented using our tool in Figure 1. The segment numbers, identified by their start times, are shown on the X-axis. The Y-axis shows different kinds of information about these segments. The four ratios used

in our algorithms as features (as well as some additional ratios) are plotted on the Y-axis using different colors. The code W(number) shows the weblinktimes of the corresponding segment. The row of Predicted bars indicates the status inferred by the online mechanism and the row of Actual bars represents the ground truth. The green bars represent normal progress and the pink bars represent difficulty points. Currently we do not show the status inferred by offline algorithms.

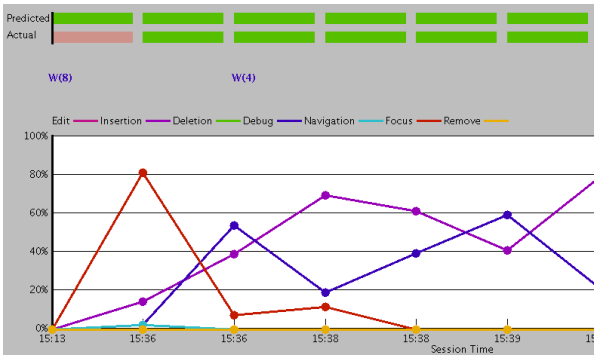

Figure 1. All ratios 0% at beginning of session

This figure shows that during a difficulty missed by the baseline system at the beginning of the interaction of a user, all ratios were at 0%, and the user visited eight web links, perhaps because the user did not know how to start the project. The number of visits is higher than normal - the average and standard deviation of weblinktimes in non-difficulty (difficulty) segments was 0.6 (2.4) and 2.8 (5.3), respectively. These data and this example show that high weblinktimes can be an indication of difficulty, and that this feature is particularly important for detecting difficulties when there is little or no interaction with the programming environment

Can difficulties be associated with low or even zero weblinktimes? 17% of the difficulty segments had no web access. The associated difficulties were presumably caused by algorithm rather than API related issues. Conversely, 15% of the non-difficulty segments had web accesses, which probably prevented burgeoning problems from blossoming into expressed surmountable or insurmountable difficulties. Thus, difficulties may be associated with higher or lower than normal weblinktimes. Intuitively, going to be web may imply either that the developers are lost, or are purposeful, knowing what they are looking for to solve potential problems.

The fact that the vast majority of difficulty and non-difficulty segments had no web access show that weblinktimes, alone, is not sufficient to predict difficulties reliably. As we see above, even when weblinktimes was combined with the features of the baseline system, the precision was low. This is consistent with the average and standard deviations of weblinktimes in non-difficulty and difficulty segments - there is a high overlap in the range of weblinktimes in these two kinds of segments.

We included the focus feature in our base algorithm to account for web searches. So why did adding weblinktimes change our results? There are several reasons. Different focus events can result in different number of web pages being visited. Moreover, web searches may not be preceded by a focus (out) event (Figure 1). Finally, such an event may not result in interaction with the browser, and even if it does, the resulting web episode may have a varying number of web page visits, as we have seen. In our study, 6% of the segments with non-zero weblinktimes had a focus ratio of zero, and conversely, 83% of segments with non-zero focus ratios had zero weblinktimes. In the remaining segments, we calculated the division of weblinktimes by focus ratio, which had an average of 0.78 and standard deviation of 1.05. Thus, both in theory and practice, focus ratio and weblinktimes do not have an obvious correlation.

VI. Discussion

This study is too specific and small to make general and final conclusions regarding the relationship between web accesses and programming difficulties. Its main contribution is providing first insights on this topic, which, arguably, have implications beyond the specific experiment.

Some of these are independent of difficulty prediction and are tied to the use of the general programming abstractions of widget composition, graphical output, and event-based input – supported in all (UI) toolkits known to us, and of particular relevance to this conference. These contributions include the percentage of difficulties involving web accesses, the relative frequency of web searches of toolkit topics during difficulty periods, and the number of web accesses during difficulty and normal periods. Unlike the numbers for these metrics, the concepts of these metrics are study-independent.

Our prediction-related contributions similarly have experiment-dependent and independent aspects. The general contributions include the idea of using weblinktimes as a new prediction feature, using well-known machine-learning metrics to determine the aggregate impact of adding this feature, exploring the relationship between it and the related existing prediction feature of focus ratio, and extending and using a special visualization tool to understand the impact of adding this feature. The actual prediction-related numbers we present, of course, can be expected to vary in different experiments.

It would be useful to do further analysis with our data such as using leave-one-out analysis rather than cross-validation, which will not have the issue of learning effect. Additional experiments can involve other forms of web-supported programming such as distributed programming. It would be useful to evaluate the prediction value of other features about web accesses such as time spent on a web page [6], the shape of the foraging curve associated with the page [10], and the topic of the page (e.g. 2D-graphic events). Future work can also use these features to make additional kinds of predictions such as whether a difficulty is surmountable or not. An in-depth manual analysis of the videos around difficulty points can reveal additional insights into the difficulties and how to predict them. Finally, it would be useful to implement an online algorithm based on web-accesses that is part of a workflow that also includes implementations of the related work on web-link recommendation [8, 9] and classification of web pages based on foraging cost [10].

This work provides a basis to explore these novel research directions.

Acknowledgments

Reviewer comments had a major impact on the final paper.

REFERENCES

[1] Carter, J., M.C. Pichiliani, and P. Dewan, Exploring the Impact of Video on Inferred Difficulty Awareness, in Proc. 16th European Conference on Computer-Supported Cooperative Work. 2018, Reports of the European Society for Socially Embedded Technologies.

[2] Dewan, P. Towards Emotion-Based Collaborative Software Engineering. in Proc. CHASE@ICSE. 2015. Florence: IEEE.

[3] Fritz, T., A. Begel, S. Mueller, S. Yigit-Elliott, and M. Zueger. Using Psycho-Physiological Measures to Assess Task Difficulty in Software Development. in Proceedings of the International Conference on Software Engineering. 2014.

[4] Drosos, I., P. Guo, and C. Parnin. HappyFace: Identifying and Predicting Frustrating Obstacles for Learning Programming at Scale. in IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). 2017. Raleigh, NC: IEEE.

[5] Price, T.W., Y. Dong, and D. Lipovac. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. . in ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17). 2017. ACM.

[6] Arguello, J. Predicting Search Task Difficulty. in Proceedings of the 36th European Conference on IR Research. 2014. Springer International Publishing.

[7] Carter, J. and P. Dewan. Mining Programming Activity to Promote Help. in Proc. ECSCW. 2015. Oslo: Springer.

[8] Sawadsky, N. and G. C. Murphy, " Fishtail: From task context to source code examples. in Proc. of the 1st Workshop on Developing Tools as Plug-ins. 2011. ACM.

[9] Sawadsky, N. and G.C. Murphy. Rahul Jiresal: Reverb: recommending code-related web pages. in Proc. ICSE. 2013.

[10] Jin, X., N. Niu, and M. Wagner:. Facilitating end-user developers by estimating time cost of foraging a webpage. : 31-35. in Proc. VL/HCC. 2017. Raleigh: IEEE.

[11] Dorn, B. and M. Guzdial. Learning on the Job: Characterizing the Programming Knowledge and Learning Strategies of Web Designers. in Proc. CHI. 2010. ACM.

[12] Carter, J. and P. Dewan. Design, Implementation, and Evaluation of an Approach for Determining When Programmers are Having Difficulty. in Proc. Group 2010. 2010. ACM.

[13] Yoon, Y. and B.A. Myers. Capturing and analyzing low-level events from the code editor. in Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools. 2011. New York.

[14] Witten, I.H. and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. 1999: Morgan Kaufmann.

[15] Carter, J. and P. Dewan, Contextualizing Inferred *Programming Difficulties*, in *Proceedings of SEmotion@ICSE, Gothenburg*. 2018, IEEE.

[16] Long, D., N. Dillon, K. Wang, J. Carter, and P. Dewan. *Interactive Control and Visualization of Difficulty Inferences from User-Interface Commands*. in *IUI Companion Proceedings*. 2015. Atlanta: ACM.