Detecting and Characterizing Developer Behavior Following Opportunistic Reuse of Code Snippets from the Web

Agnieszka Ciborowska
Virginia Commonwealth University
Richmond, VA, USA
ciborowskaa@vcu.edu

Nicholas A. Kraft
ABB Corporate Research
Raleigh, NC, USA
nicholas.a.kraft@us.abb.com

Kostadin Damevski
Virginia Commonwealth University
Richmond, VA, USA
kdamevski@vcu.edu

ABSTRACT

Modern software development is social and relies on many online resources and tools. In this paper, we study opportunistic code reuse from the Web, e.g., when developers copy code snippets from popular Q&A sites and paste them into their projects. Our focus is the behavior of developers following opportunistic code reuse, which reveals the success or failure of the action. We study developer behavior via a large, representative dataset of micro-interactions in the IDE. Our analysis of developer behavior exhibited in this dataset confirms laboratory study observations that code reuse from the Web is followed by heavy editing, in some cases by a rapid undo, and rarely by the execution of tests.

KEYWORDS

opportunistic reuse, code snippet, developer behavior, interaction data, field study

ACM Reference Format:
Agnieszka Ciborowska, Nicholas A. Kraft, and Kostadin Damevski. 2018. Detecting and Characterizing Developer Behavior Following Opportunistic Reuse of Code Snippets from the Web. In *MSR '18: MSR '18: 15th International Conference on Mining Software Repositories , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3196398.3196467

1 INTRODUCTION

Resources to aid software development — e.g., tutorials, blog posts, Q&A forums, and chat communities [4] — are abundant on the Web. Developers consult these resources as part of their daily workflow [6], sometimes copying code snippets from Q&A sites such as Stack Overflow into their code bases [17]. Indeed, multiple laboratory studies [2, 3, 12] indicate that a pervasive strategy for opportunistic reuse among developers is to copy code snippets from the Web. For example, Brandt et al. [2, 3] report that developers use the Web numerous times per day as a source to learn or be reminded, as well as to reuse code snippets for certain repetitive tasks.

The Enriched Event Streams dataset [14] provides field data for a group of 81 developers. Its logs contain developer micro-interactions with the Visual Studio IDE and list over 11M events that correspond to clicks or key presses recorded by the developers. Similar datasets have been used to detect IDE-usage smells from sequences of developer interactions [8, 9] and to estimate technical debt [15, 16]. In this paper we use the field data from the Enriched Event Streams dataset to detect and characterize developer behavior surrounding opportunistic reuse of code snippets from the Web. The goals of our study are (1) to understand the behaviors exhibited by developers subsequent to pasting code from outside of the IDE and (2) to confirm (or dispute) the developer behaviors observed during past laboratory studies [2, 3, 5, 11–13].

We use a two-step process to understand developer behavior after pasting code from outside of the IDE. First, we use periods of inactivity to detect pastes of code snippets from the Web. Next, we analyze occurrences of specific events and use the Hidden Markov Model [7] to model the behavior that developers exhibit after such pastes. Via analysis of the resulting patterns, we look for evidence of developer behaviors observed in previous laboratory studies. The observed behaviors of interest include:

B1: If code is copied from the Web and not examined, the paste may be followed by an undo or delete [2]
B2: Developers do not immediately test code copied and pasted from the Web (because they assume that it is correct) [2]
B3: Developers copy and paste code from the Web to resolve errors encountered while debugging [2]
B4: Quick, repeated insertion of small changes can lead to cascading undo operations [12]
B5: Pastes may be followed by field or parameter additions to achieve syntactic/semantic correctness [12]
B6: Pastes may be followed by variable renames to make snippets fit in the desired context [12]
B7: Pastes may be followed by commenting or removal of code sections to reject uneeded portions of copied code [12]
B8: Pastes may be followed by an edit/paste-compile loop to resolve dependencies [1]
B9: Developers need to adapt code copied and pasted from the Web (e.g., by adding glue code or refactoring variable names) [13]

ACM ISBN 978-1-4503-5716-6/18/05...$15.00
https://doi.org/10.1145/3196398.3196467

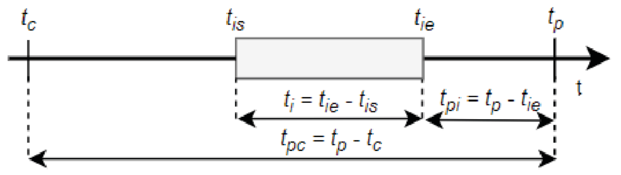

Figure 1: Concept of a pattern for recognizing paste from the Web based on time dependencies.

2 RESEARCH METHOD

Our research method consists of two distinct steps, performed in sequence. First is recognizing the opportunistic reuse events in the interaction data. Since there are no explicit indicators of these events in the data, we resort to developing and using a set of heuristics. Following this, we discover an interpretable Hidden Markov Model representation of the interaction sequences following the opportunistic reuse events.

2.1 Recognizing Pastes of Code from Outside of the IDE

Reuse of code snippets from outside of the IDE (e.g. from the Web) is usually preceded by some period of inactivity in the IDE. During this time a developer leaves the IDE to browse the Web and to find a relevant code snippet, which is then pasted into the code via the IDE. Several researchers describe this common behavioral pattern [2, 11, 12]. Thus, we base our recognition of pasted code on periods of inactivity and the time elapsed between two specific actions in the Enriched Event Streams dataset: copy (or cut), and paste. We recognize copy, cut, and paste actions by identifying events containing the corresponding command identifiers. We cannot recognize developer inactivity in quite the same way, as there is no particular IDE event associated with losing or restoring focus to the IDE. Yet, we note that if a developer is using the IDE, then every performed action is registered as an event. Hence, we can recognize inactivity based on extended time elapsed between consecutive events.

Figure 1 illustrates interaction characteristics of pastes from outside of the IDE. We denote time of paste command as t_p and time of a preceding copy or cut command as t_c, whereas t_{pc} stands for time interval between copy or cut and paste. Inactivity duration is denoted as t_i, which is the difference between inactivity ending (t_{ie}) and starting time (t_{is}). The timespan between inactivity end and occurring paste command is represented by t_{pi}.

Another important characteristic of pasting code from Web is that the preceding copy/cut does not occur in the IDE, so it is not recorded in our dataset. Therefore, to ensure that pasted content did not originate from within the IDE, we conservatively exclude paste commands which occurred less than 5 minutes after cut/copy action was recorded as it is highly possible that a developer was copy and pasting code inside the IDE. The requirements can be specified as $t_{pc} > 5$ min. As a first step, we preprocessed the dataset and collected every paste command meeting this requirement.

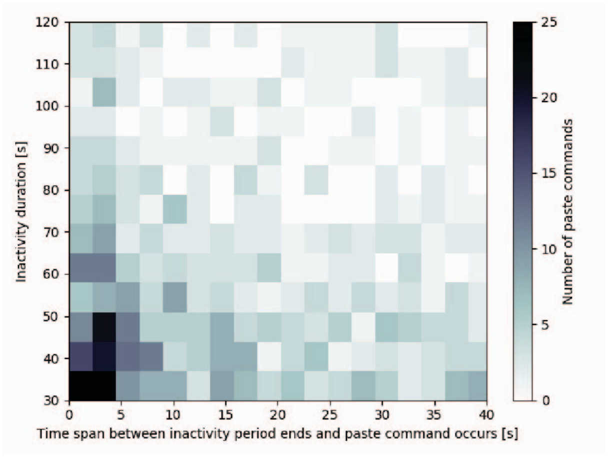

Figure 2: Heat map presenting frequency of paste command occurrence in relation to inactivity duration and time span between inactivity end and paste command.

We defined a lower *opportunistic reuse inactivity* threshold as a period of time lasting at least 30 seconds, which is the minimum period we deemed sufficient to launch a Web browser, search and analyze some code snippets, and copy one of them into the IDE. We also defined an upper threshold of 20 minutes. After 20 minutes of inactivity, every new event is considered to belong to a new session, and any information about the most recent copy/cut command (or inactivity period) is discarded as invalid for the new session.

Figure 2 presents a heat map that illustrates the number of paste commands in relation to preceding inactivity duration(t_i) and time span between inactivity ended and paste occurred(t_{pi}). To increase plot readability, we adjusted the range of the x and y axes and rejected statistically insignificant values that were more than three standard deviations from the mean of t_{pi}. As can be observed in this figure, most pastes took place up to 15 seconds after inactivity period ended and were performed after an inactivity duration lasting up to 80 seconds. In the end, we arrived at the following set of rules to recognize a paste from outside of the IDE:

(1) $t_{pi} < 15$ seconds,
(2) $t_{pc} > 5$ minutes,
(3) $t_i > 30$ seconds.

We did not set any upper bound on inactivity duration, excluding the session break threshold of 20 minutes, so as not to lose potentially valid paste commands.

Using this approach we identified 631 opportunistic reuse of code snippets from outside the IDE.

2.2 Interactive Hidden Markov Model Discovery

Developer behavior immediately following opportunistic code reuse, in a tight window of time, can be indicative of the steps developers take to ensure the quality of the reused code. To investigate this set of behaviors our focus are sequences of commands directly following pasting code from the Web.

Understanding sequences of low-level developer interactions with the IDE, occurring across numerous scenarios by many individual developers is challenging. In order to build a succinct representation, we leverage interactively building a Hidden Markov Model (HMM) that comprehensively captures these sequences [7]. HMMs consist of a set of hidden states, corresponding to high-level behaviors of interest, and a set of observed states, corresponding to messages in our input sequences. The approach we use constructs an interpretable HMM model by using human feedback to iteratively add one hidden state at a time to the model.

The process is as follows. At the outset, the HMM consists of a single hidden state, which is able to emit all the commands in the interaction data. To begin each iteration, the system informs the human expert of a set of highly probable sequences that exist in the input data, but are not captured by the current HMM. Using this prompt, and an understanding of the semantics of different commands (e.g. all commands related to debugging) the expert directs the system to create a new hidden state specifically targeted to emitting a set of semantically related messages (e.g. related to debugging). A new HMM is then computed using the Baum-Welch algorithm. The process then repeats to again presenting to the expert the sequences that this new HMM fails to capture, prompting for commands that can be used for the next hidden state. The final model is produced when the expert deems the model is successful at capturing the most probable sequences in the input data. In addition, at each iteration, the system outputs a probabilistic rating of the quality of the model, ensuring that the final produced HMM fits the input sequences.

Specifically to the Enriched Event Stream dataset, we selected all of the CommandEvents, and used their descriptive names and executions of the FeedBag Visual Studio collection tool to understand the semantics of ambiguous command names. We finally constructed the model with 6 hidden states and quality rating of over 99% in Figure 3. To render the model more clearly, we removed the initial state, which always captures only the remaining commands in the dataset, and edges that were loops, subsequently rescaling the edge probabilities.

3 RESULTS

In this section, we examine the results of our analysis of behavior following opportunistic reuse of code from the Web, including results based on the short-term HMM representation and a longer term exploration of specific behaviors of interest (e.g., testing [10]). We present evidence as it pertains to the behaviors of interest we intended to confirm or understand with this field study.

Opportunistic reuse is predominantly followed by editing code (B3, B5, B6, B7, B8, B9).
Our HMM representation of short-term behavior (Figure 3) indicates that the most likely action following pasting code

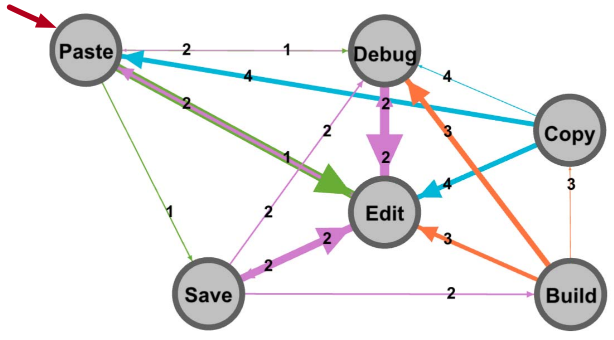

Figure 3: Hidden Markov Model of the behavior immediately following opportunistic code reuse. Nodes represent high-level behaviors, while edges represent transitions. The transition probability between two behaviors is encoded as edge thickness. The edges are numbered (and colored) by the length of the shortest path from the initial state – Paste.

from the Web is editing. The Edit state in the HMM corresponds to a large set of commands that add, edit, delete or refactor lines of code in the IDE. Second to editing behavior, following a paste, developers may save the code or initiate or continue a debugging session. These two behaviors tend to follow editing with a considerable probability as well.

Undo operations, when they occur, follow opportunistic reuse in a short timespan (B1, B4).
In examining all undo operations that occurred in the 5 minutes following an opportunistic reuse as detected by our interaction data pattern, the median time to their occurrence was 86 seconds. This is as compared to edit operations, which followed pasting code with a median time of 125 seconds, test executions with a median time of 185 seconds, and building code with a median time of 132 seconds. Further, though Figure 4 shows a small subset of scenarios following a paste, undo commands tend to quickly follow the paste.

Developers rarely test immediately after opportunistic reuse (B2). In Figure 4 we identify all of the opportunistic reuse sessions that contained a test in the 15 minutes following the paste, consisting of 21 out of the 631 opportunistic reuse sessions, meaning that only 3% of recognized paste commands were followed by running tests. To visualize developer behaviour occurring after opportunistic reuse, we delineate several behaviors, including building the project (successfully and unsuccessfully), testing (with failing tests or not), and undos. We observe both the small number of these sessions and the relatively infrequent occurrence of tests executed within the first 2 minutes after the paste (only 6 out of 21 sessions).

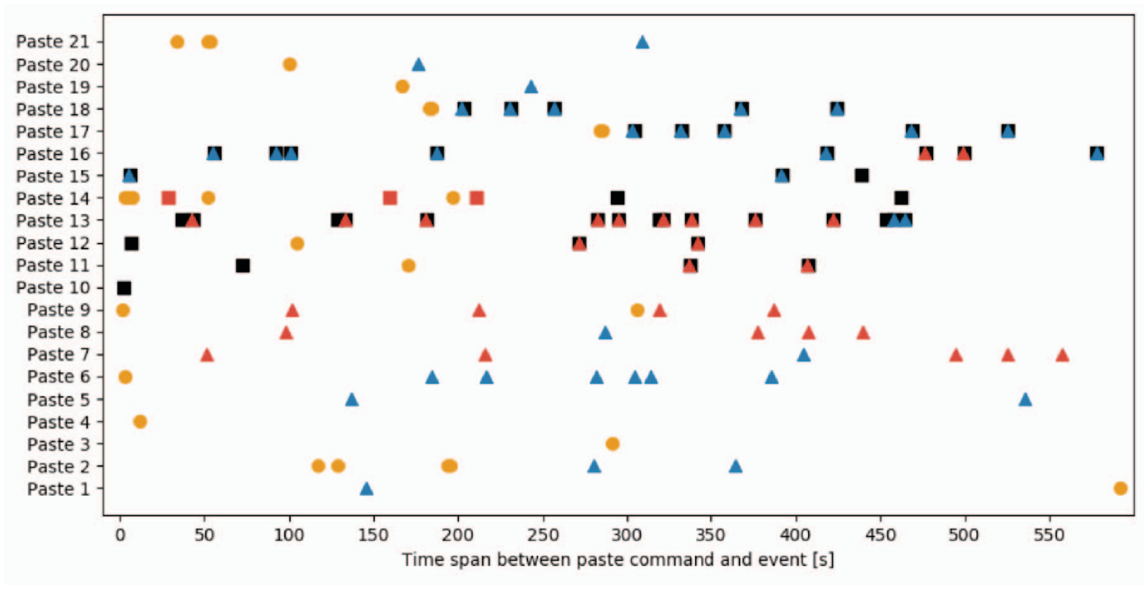

Figure 4: A timeline of actions by developers after opportunistic reuse that include at least one test execution in 15 minutes. Triangles denote running tests: blue color represents successful or unknown results, red refers to failure. Black squares stand for successful build, and red for failing build. Undo operations are denoted by yellow circles.

4 CONCLUSIONS

In this paper, we use interaction data, gathered in the field during software developers' daily work, to evaluate a set of observations made during laboratory studies of opportunistic code reuse. Our findings indicate a strong prevalence of edits following pasting code from the Web, the existence of rapid undo commands following paste, and that test execution is not a typical validation strategy for code that is reused. Each of these findings lends more support to the prior observations made in laboratory studies. The prevalence of opportunistic code reuse in modern development, and the difficulty for developers to validate the pasted code, exhibited in their IDE interactions, motivates the need for novel tools or techniques to further improve code reuse from the Web.

REFERENCES

[1] A. Armaly and C. McMillan. 2016. Pragmatic source code reuse via execution record and replay. *Journal of Software: Evolution and Process* 28, 8 (2016), 642–664.
[2] J. Brandt, P.J. Guo, J. Lewenstein, M. Dontcheva, and S.R. Klemmer. 2009. Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. In *Proc. ACM CHI Conference on Human Factors in Computing Systems (CHI'09)*. 1589–1598.
[3] J. Brandt, P.J. Guo, J. Lewenstein, M. Dontcheva, and S.R. Klemmer. 2009. Writing Code to Prototype, Ideate, and Discover. *IEEE Software* 26, 5 (Sept.-Oct. 2009).
[4] P. Chatterjee, M.A. Nishi, K. Damevski, V. Augustine, L. Pollock, and N.A. Kraft. 2017. What Information about Code Snippets Is Available in Different Software-Related Documents? An Exploratory Study. In *Proc. 24th IEEE Int'l Conf. on Software Analysis, Evolution, and Reengineering (SANER'17)*.
[5] D. Chatterji, J.C. Carver, N.A. Kraft, and J. Harder. 2013. Effects of cloned code on software maintainability: A replicated developer study. In *Proc. 20th Working Conf. on Reverse Engineering (WCRE'13)*. 112–121.
[6] C.S. Corley, F. Lois, and S.F. Quezada. [n. d.]. Web Usage Patterns of Developers. In *Proc. 31st Int'l Conf. on Software Maintenance and Evolution (ICSME'15)*.
[7] K. Damevski, H. Chen, D. Shepherd, and L. Pollock. 2016. Interactive Exploration of Developer Interaction Traces using a Hidden Markov Model. In *Proc. 13th Int'l Conf. on Mining Software Repositories (MSR'16)*.
[8] K. Damevski, D. Shepherd, and L. Pollock. 2015. A Field Study of How Developers Locate Features in Source Code. *Empirical Software Engineering* 21, 2 (2015), 724–747.
[9] K. Damevski, D. Shepherd, J. Schneider, and L. Pollock. 2016. Mining Sequences of Developer Interactions in Visual Studio for Usage Smells. *IEEE Trans. Softw. Eng.* (2016).
[10] R. Delamare and N.A. Kraft. 2012. A genetic algorithm for computing class integration test orders for aspect-oriented systems. In *Proc. 5th Int'l Conf. on Software Testing, Verification, and Validation*. 804–813.
[11] E. Duala-Ekoko and M. Robillard. 2012. Asking and answering questions about unfamiliar APIs: An exploratory study. In *Proc. 34th Int'l Conf. on Software Engineering (ICSE'12)*.
[12] R. Holmes and R. Walker. 2012. Systematizing pragmatic software reuse. *ACM Transactions on Software Engineering and Methodology* 21, 4 (Nov. 2012).
[13] H. Li, Z. Xing, X. Peng, and W. Zhao. 2013. What help do developers seek, when and how?. In *Proc. 20th Working Conf. on Reverse Engineering (WCRE'13)*.
[14] S. Proksch, S. Amann, and S. Nadi. 2018. Enriched Event Streams: A General Dataset for Empirical Studies on In-IDE Activities of Software Developers. In *Proc. 15th Working Conference on Mining Software Repositories (MSR'18)*.
[15] V. Singh, L. Pollock, W. Snipes, and N.A. Kraft. 2016. A Case Study of Program Comprehension Effort and Technical Debt Estimations. In *Proc. 24th IEEE Int'l Conf. on Program Comprehension (ICPC'16)*.
[16] V. Singh, W. Snipes, and N.A. Kraft. 2014. A Framework for Estimating Interest on Technical Debt by Monitoring Developer Activity Related to Code Comprehension. In *Proc. 6th IEEE Int'l Wksp. on Managing Technical Debt (MTD'14)*.
[17] D. Yang, P. Martins, V. Saini, and C. Lopes. 2017. Stack Overflow in GitHub: Any Snippets There?. In *Proc. 14th Int'l Conf. on Mining Software Repositories (MSR'17)*. 280–290.