Do Software Engineers Use Autocompletion Features Differently Than Other Developers?

Rahul Amlekar
McGill University, Canada
rahul.amlekar@mail.mcgill.ca

Andrés Felipe Rincón Gamboa
McGill University, Canada
andres.rincon@mail.mcgill.ca

Keheliya Gallaba
McGill University, Canada
keheliya.gallaba@mail.mcgill.ca

Shane McIntosh
McGill University, Canada
shane.mcintosh@mcgill.ca

ABSTRACT

Autocomplete is a common workspace feature that is used to recommend code snippets as developers type in their IDEs. Users of autocomplete features no longer need to remember programming syntax and the names and details of the API methods that are needed to accomplish tasks. Moreover, autocompletion of code snippets may have an accelerating effect, lowering the number of keystrokes that are needed to type the code. However, like any tool, implicit tendencies of users may emerge. Knowledge of how developers in different roles use autocompletion features may help to guide future autocompletion development, research, and training material. In this paper, we set out to better understand how usage of autocompletion varies among software engineers and other developers (i.e., academic researchers, industry researchers, hobby programmers, and students). Analysis of autocompletion events in the Mining Software Repositories (MSR) challenge dataset reveals that: (1) rates of autocompletion usage among software engineers and other developers are not significantly different; and (2) although several non-negligible effect sizes of autocompletion targets (e.g., local variables, method names) are detected between the two groups, the rates at which these targets appear do not vary to a significant degree. These inconclusive results are likely due to the small sample size ($n = 35$); however, they do provide an interesting insight for future studies to build upon.

ACM Reference Format:
Rahul Amlekar, Andrés Felipe Rincón Gamboa, Keheliya Gallaba, and Shane McIntosh. 2018. Do Software Engineers Use Autocompletion Features Differently Than Other Developers?. 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/10.1145/3196398.3196471

1 INTRODUCTION

Most modern Integrated Development Environments (IDEs) provide autocompletion features, which allow developers to complete a code fragment by selecting from a list of recommended options that appear as they type. Developers who leverage autocomplete no longer need to remember or look up API methods and parameters that exist in large codebases. Autocomplete may also save developers' time by decreasing the amount of keyboard input that is required to reference identifiers in the source code.

Recently, there has been a push to improve IDE features like autocomplete [7]. To build an effective and useful suite of autocomplete features, knowledge of how autocomplete tools are being used is essential. Historical tool usage data has been effectively used to reduce clutter in the IDE workspace [5]. Knowledge of autocomplete usage could be used to guide the following: (1) research towards areas that autocomplete features do not yet support; (2) development effort towards popular (or unpopular) features; and (3) the generation of training material towards infrequently or often misused autocompletion features.

There are only a limited amount of studies in understanding the differences between the software development process across job roles [1][3]. In this paper, we set out to study whether software engineers use autocompletion features differently than other developers (i.e., academic researchers, industry researchers, hobby programmers, and students). We measure the overall use of autocompletion features using *autocompletion rates*, i.e., the frequency of autocomplete use in an IDE session. Moreover, we measure specific autocompletion use with *autocompletion targets*, i.e., the type of code element that is being autocompleted, e.g., variables, methods, fields, and types. More specifically, through the analysis of the MSR challenge dataset [9], we address the following research questions:

(RQ1) How do the autocompletion rates of software engineers and other developers compare?
Although we observe a slight difference in the median autocompletion rates between software engineers (0.703%) and other developers (0.846%), the differences are not statistically significant (Mann-Whitney U test, $p = 0.814$) and the effect size is negligible (Cliff's delta = 0.012).

(RQ2) How do the autocompletion targets of software engineers and other developers compare?
Local Variable and *Parameter* autocompletion targets tend to appear more frequently in the interaction traces of software engineers, where we detect small Cliff's delta effect sizes of 0.287 and 0.300, respectively. On the other hand, *Method* and *Type* autocompletion targets tend to appear more frequently in the interaction traces of other developers, where we detect small and medium Cliff's delta effect sizes of 0.296 and 0.333,

MSR '18, May 28–29, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196471

respectively. However, Mann-Whitney U tests indicate that none of these results are statistically significant ($p \geq 0.05$).

Unfortunately, none of our results are statistically significant. The negative statistical hypothesis test results indicate that either there is no difference between autocompletion behaviour of software engineers and other developers or that we have not collected enough data to distinguish between the two role types. Given the small number of users ($n = 35$) who volunteered their job information, it is likely that more data is required to arrive at more definitive conclusions.

2 STUDY DESIGN

The MSR challenge dataset provides over 11 million IDE interactions from 85 developers spanning 15,000 hours of work [9]. Figure 1 provides an overview of our approach to answer our research questions using the MSR challenge dataset, which, at a high level, is composed of the data filtering and data analysis steps that we describe below.

In order to aid in future replication of our results, we make our data and scripts available online.[1]

2.1 Data Filtering

The MSR challenge dataset contains plenty of entries that need to be preprocessed before it can be used to address our research questions. Below, we describe our data extraction and filtering approach.

DF1: Extract Raw Data. We begin by uncompressing all of the archives in the dataset. We extract the following information for each IDE interaction: (1) a unique identifier for the user who performed the interaction (given by the archive name), (2) the identifier for the session that the interaction appeared within, (3) the time at which the event was recorded, and (4) the event type, based on the format of the *IDEEvent* class of the KaVE project.[2]

DF2: Select Required Events. To analyze whether the use of autocomplete varies according to job role, we need to analyze two types of *IDEEvents*. First, we need *UserProfileEvents*[3] to extract the job role of each IDE user and *CompletionEvents* to extract data about the autocompletion events that were performed by each user.

The job role field is optional—35 of the 85 users in the dataset have provided a job role within their profile. We group the job roles into *software engineer* and *other developer* categories. The software engineer category only includes users who provided the job role of *software engineer*, whereas those who provided the job roles of *student*, *academic researcher*, *industry researcher*, and *hobby programmer* were merged into the other developers category. There were 16 other users who had volunteered other self-identifying data but not the job profile data.

We extract 194,500 autocompletion events from the dataset. For each autocompletion event, we extract its termination state (i.e., canceled, filtered, applied), and the autocompletion target type.

DF3: Filter Unsuitable Data. We start by filtering out the 16 users who did not provide job profile information at the time of recording developer-IDE interactions.

Due to the difficult nature of recording developer interactions, the dataset has events that were not correctly captured. For example, there are 14,055 autocompletion events of type *GeneralName*, which indicates that the KaVE tool could not classify the autocompletion event into a known type. We filter these 14,055 incomplete autocompletion events out of our dataset.

Furthermore, there are three termination types for autocomplete events. *Applied* means that the developer selected an autocompletion proposal. *Cancelled* means that the developer cancelled the prompted autocompletion proposal. *Filtered* means that the list of autocompletion proposals was updated to better match what the developer was typing. The only autocompletion event that is of interest for our analysis is the *Applied* type. Therefore, we filter out the 99,453 autocompletion events of type *Filtered* and the 42,365 autocompletion events of type *Cancelled*.

2.2 Data Analysis

After data extraction and filtering have been applied, 38,627 suitable autocompletion events remain in our dataset. To address our research questions, we analyze the surviving autocompletion events using the following four steps.

DA1: Compute Autocomplete Ratio. To better understand how often developers are using autocomplete features, we begin by computing the number of autocompletion events in each session. Since session length may be a confounding factor, we normalize the raw count of autocompletion events in a session by the total number of events in that session. We also normalize the raw count of autocompletion events in a session by the session duration in minutes but find that developers who executed long-running jobs (e.g., builds of a large system) have disproportionately larger development time windows than others. Hence, we select the ratio of the autocomplete events to the total number of events in a session as our measure of autocompletion usage.

DA2: Compare Across Roles. To address RQ1, we compare the autocomplete usage of software engineers to that of other developers (researchers, students, and hobby programmers). After calculating the autocomplete ratios for both of these job profile categories, we plot the distributions in *beanplots*. Beanplots are boxplots in which the vertical curves summarize and compare the distributions of different data sets [4], which in our case correspond to the session-specific autocomplete ratios of software engineers and other developers. The long horizontal black lines show the median of the autocomplete ratios of software engineers (gray) and other developers (black). The smaller horizontal white lines provide histogram-like frequency data.

We then perform Mann-Whitney U tests [6][10] ($\alpha = 0.05$) to check if there is a significant difference between the ratios of autocompletion usage of software engineers and other developers. Finally, we use Cliff's delta [2] to measure the effect size, which is *negligible* when $0 \leq \text{delta} < 0.147$, *small* when $0.147 \leq \text{delta} \leq 0.33$, *medium* when $0.33 \leq \text{delta} \leq 0.474$, and *large* otherwise. We perform Mann-Whitney U tests and Cliff's delta test since these are non-parametric tests and our data is not normally distributed.

DA3: Identify Autocomplete Type. There are 12 types of targets that can be autocompleted, e.g., type, variable, or method names. In RQ2, we set out to study which targets tend to be autocompleted

[1]https://doi.org/10.6084/m9.figshare.5984941
[2]https://github.com/stg-tud/kave-java
[3]http://www.kave.cc/feedbag/event-generation

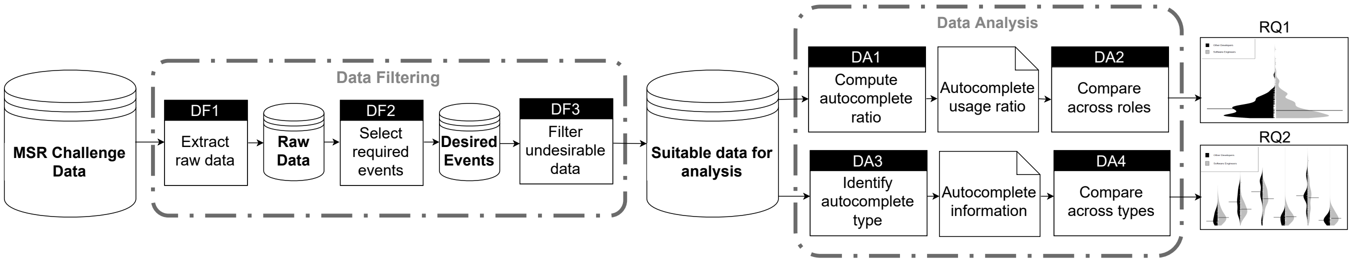

Figure 1: An overview of our approach to analyzing the MSR challenge dataset.

more often and whether the frequency of the target autocompletion varies based on job role.

We identify which target a particular autocomplete selection belongs to using a "name grammar for code elements" [8]. There are 12 autocomplete target types in the datset but six are filtered out because they have less than 20 data points or are corrupted. So we base our analysis on the remaining six autocomplete target types:

- **FieldName:** The name of a field of an object is autocompleted
- **LocalVariableName:** The name of a variable that is defined in the local scope is autocompleted
- **MethodName:** The name of a method is autocompleted
- **ParameterName:** An argument or a parameter to be included in a method call is autocompleted
- **PropertyName:** A getter or a setter is autocompleted
- **TypeName:** A type (e.g., Int, Char, Byte) is autocompleted

For the sake of completeness, the filtered out targets are *NamespaceName*, *EventName*, *AliasName*, *GeneralName*, *TypeParameterName*, and *PredefinedTypeName*

DA4: Compare Across Types. We merge the autocompletion event type data with the user profile data and plot the frequency of autocompletion usage stratified by event type. We again use beanplots (one plot per event type) to compare the distributions of autocomplete usage of software engineers and other developers. The code for this can be found in our replication package.[4] We also use the Mann-Whitney U tests and Cliff's delta to test for significant differences among the event types and measure the effect size between the event types, respectively. As in DA2, we choose these tests since these are non-parametric and our data is not normally distributed.

3 STUDY RESULTS

In this section, we present the results of our study with respect to our two research questions. For each research question, we first discuss the motivation and then present the results.

RQ1: How do the autocompletion rates of software engineers and other developers compare?

Motivation. To improve autocompletion tools and techniques, it would be useful to know who is using existing features. This data would help autocompletion tool developers to better tailor existing solutions to its userbase. Moreover, this data may reveal types of users who are not currently benefiting from autocompletion features. Such types of users may need better support in order to fully leverage autocompletion features.

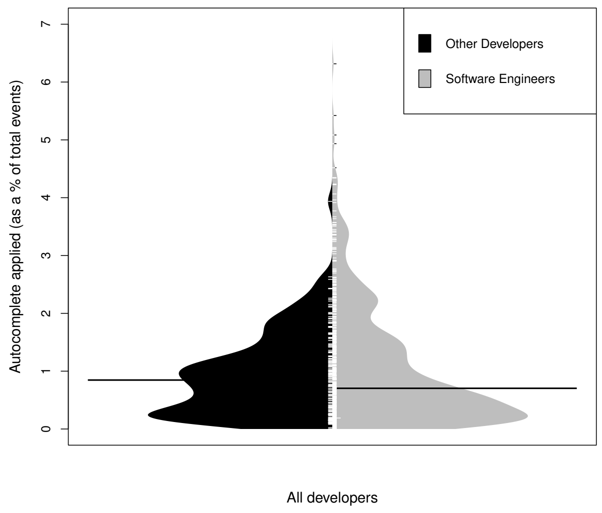

Figure 2: Beanplot of autocompletion applied as a percentage of total events

Results. Autocompletion rates are similar for both software engineers and other developers. Figure 2 provides an overview of the distributions of autocompletion rates using a beanplot (see Section 2.2). The plots show that the median rate for other developers (0.846%) is slightly higher than that of software engineers (0.703%); however, a Mann-Whitney U test shows that the difference is not statistically significant ($p = 0.814$). Moreover, the Cliff's delta is 0.012, which is considered *negligible*.

Figure 2 also shows that the difference in data dispersion is minimal. Indeed, the standard deviation in autocompletion rates for other developers is 0.682 and for software engineers is 1.017.

Observation 1*: Autocompletion rates are similar for both software engineers and other developers.*

[4]https://doi.org/10.6084/m9.figshare.5984941

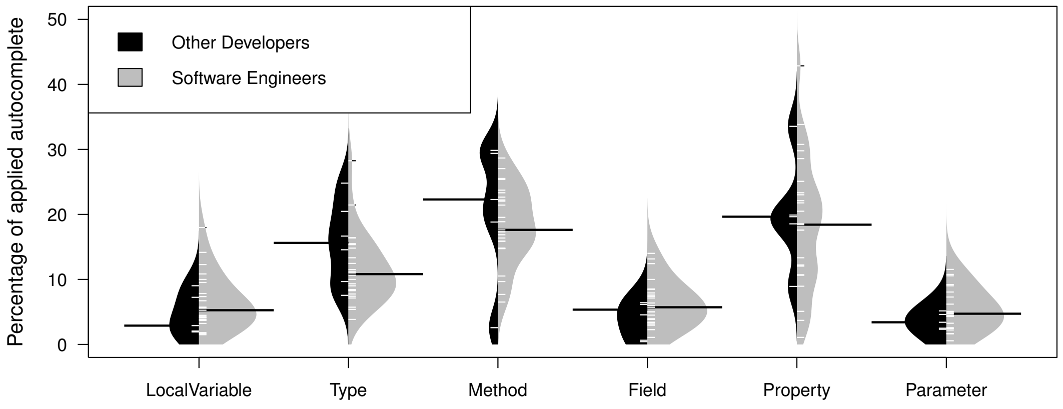

Figure 3: Beanplot of autocomplete target types by job role as a percentage of autocomplete applied

RQ2: How do the autocompletion targets of software engineers and other developers compare?

Motivation. To gain a more detailed perspective of autocomplete usage, we are interested in breaking autocompletion down by its target, and checking whether software engineers are more likely to autocomplete targets of a particular type.

Results. Both software engineers and other developers autocomplete similar target types. Figure 3 shows the distributions of autocomplete usage in a beanplot. We again notice slight differences in the medians of the distributions. Moreover, there are non-negligible effect sizes in four of these six distributions. For Local Variable (small Cliff's delta of 0.287) and Parameter (small Cliff's delta of 0.300), software engineers tend to autocomplete more than other developers. For Type (medium Cliff's delta of 0.333) and Method (small Cliff's delta of 0.296), other developers tend to autocomplete more than software engineers. However, Mann-Whitney U tests indicate that these sample differences are not statistically significant (p-value ≥ 0.05) for any of the autocompletion target types. Therefore, we cannot reject the null hypothesis that the samples of both job roles come from the same population (i.e., we cannot detect a significant difference).

***Observation 2**: While we do observe non-negligible effect sizes in four of the six autocompletion targets, Mann-Whitney U test results are inconclusive. Thus, we cannot conclude that the software engineers and other developers autocomplete target types differently.*

4 CONCLUSIONS

Understanding how developers use autocomplete is essential for autocomplete to be improved. Our potential variation point among autocomplete users is their job role. To understand how a software engineer's use of autocomplete features differs from that of other developers, we analyze autocompletion usage in the MSR challenge dataset, observing that: (1) job role does not share a significant relationship with rates of autocompletion; and (2) four of the six studied autocompletion target types have non-negligible effect sizes; however, Mann-Whitney U tests are inconclusive.

The results from our statistical analysis are inconclusive. No significant differences in the autocompletion usage of software engineers and other developers were discovered. However, we suspect that the issue is due to the relatively small size of users who provided job role information in the MSR challenge dataset ($n = 35$).

Our observations are an interesting starting point for future studies. Developers with different job roles usually have different goals, e.g., a student may develop software with the goal of completing an assignment, a software engineer may develop to meet a client's expectations, and a hobby programmer may be working on a self-chosen project. Therefore, we believe that by comparing autocomplete data of different job roles, we can gain more insight in the differences in their software development process. More data is needed to arrive at more concrete conclusions.

REFERENCES

[1] J. Carver, L. Jaccheri, S. Morasca, and F. Shull. 2003. Issues in using students in empirical studies in software engineering education. In *Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No.03EX717)*. 239–249. https://doi.org/10.1109/METRIC.2003.1232471
[2] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin* 114, 3 (1993), 494–509. http://dx.doi.org/10.1037/0033-2909.114.3.494
[3] Martin Höst, Björn Regnell, and Claes Wohlin. 2000. Using Students as Subjects—A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. *Empirical Software Engineering* 5, 3 (01 Nov 2000), 201–214. https://doi.org/10.1023/A:1026586415054
[4] Peter Kampstra. 2008. Beanplot: A Boxplot Alternative for Visual Comparison of Distributions. *Journal of Statistical Software, Code Snippets* 28, 1 (2008), 1–9.
[5] Mik Kersten and Gail C. Murphy. 2006. Using Task Context to Improve Programmer Productivity. In *Proceedings of the International Symposium on the Foundations of Software Engineering (FSE)*. 1–11.
[6] H. B. Mann and D. R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* 18, 1 (1947), 50–60. http://www.jstor.org/stable/2236101
[7] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. 2014. Mining StackOverflow to Turn the IDE into a Self-confident Programming Prompter. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*. ACM, New York, NY, USA, 102–111. https://doi.org/10.1145/2597073.2597077
[8] Sebastian Proksch. 2017. *Enriched Event Streams: A General Platform For Empirical Studies On In-IDE Activities Of Software Developers.* Ph.D. Dissertation. Technische Universität, Darmstadt. http://tuprints.ulb.tu-darmstadt.de/6971/
[9] Sebastian Proksch, Sven Amann, and Sarah Nadi. 2018. Enriched Event Streams: A General Dataset for Empirical Studies on In-IDE Activities of Software Developers. In *Proceedings of the 15th Working Conference on Mining Software Repositories.*
[10] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83. http://www.jstor.org/stable/3001968