A Study on the Use of IDE Features for Debugging

Afsoon Afzal
Carnegie Mellon University
Pittsburgh, Pennsylvania
afsoona@cs.cmu.edu

Claire Le Goues
Carnegie Mellon University
Pittsburgh, Pennsylvania
clegoues@cs.cmu.edu

ABSTRACT

Integrated development environments (IDEs) provide features to help developers both create and understand code. As maintenance and bug repair are time-consuming and costly activities, IDEs have long integrated debugging features to simplify these tasks. In this paper we investigate the impact of using IDE debugger features on different aspects of programming and debugging. Using the data set provided by MSR challenge track, we compared debugging tasks performed with or without the IDE debugger. We find, on average, that developers spend more time and effort on debugging when they use the debugger. Typically, developers start using the debugger early, at the beginning of a debugging session, and that their editing behavior does not appear to significantly change when they are debugging regardless of whether debugging features are in use.

KEYWORDS

mining, debugging, IDE, integrated development environment

ACM Reference Format:
Afsoon Afzal and Claire Le Goues. 2018. A Study on the Use of IDE Features for Debugging. 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/10.1145/3196398.3196468

1 INTRODUCTION

More than 70% of software budgets are spent on maintenance. Of all maintenance activities, 35% are dedicated to *corrective maintenance*, or what we simply call debugging [3]. Hours of developer time is spent on manually finding and fixing bugs [10]. Reducing developer time spent on debugging can thus significantly lower the cost of software production.

Integrated development environments (IDEs) provide features to help developers create and understand code [9]. IDEs have long integrated debugging features to make it as convenient as possible for developers to understand code and discover and address defects. Many of the most popular IDEs such as *Eclipse* and *Visual Studio* include debugging views that allow one to set up breakpoints and walk through execution step-by-step. This allows the observation of variables and expression values at each step, often providing a better, more granular understanding of execution state. Murphy et al. [7] investigate how 99 Java developers use the Eclipse IDE. They find that more than 90% of the participants use Eclipse debugging features. Visual attention patterns of developers performing debugging with the jGrasp IDE vary based on participant programming experience, familiarity with the IDE, and debugging performance [4]. Similarly, of participants performing five maintenance tasks using Eclipse, programmers spend 35% of their time navigating through dependencies, and 46% of their time inspecting irrelevant code [5]. Qualitative and quantitative studies of real-world debugging finds that both knowledge and use of advanced debugging features in IDEs are low [1]. To the best of our knowledge, prior work has not looked into the effectiveness of IDE debugging features with respect to reducing maintenance time and cost.

In this work, we conduct an empirical study on a rich data set of in-IDE activities of software developers provided by the Mining Software Repositories (MSR) 2018 challenge track [8]. The data set contains over 11 million events, corresponding to 15K hours of working time of 81 developers; they have been collected using FeedBaG, a general-purpose interaction tracker for Visual Studio. We investigate developer debugging behavior. First, we show that there is a significant difference between the time spent on debugging performed with and without IDE debugging features: On average, people spend more time on debugging when a debugger is used along the way. Second, we show that in most debugging tasks, people switch to debugging mode early on. In 80% of cases, developer starts using the debugger before 13 minutes have elapsed from the beginning of debugging. Finally, in the majority of cases, the editing behavior of developers does not change significantly when they start or end debugging.

2 RESEARCH QUESTIONS

Understanding the behavior of programmers using IDEs to debug can elaborate limitations and opportunities for enhancement in designing these tools. Our focus in this paper is specifically on debugging features. We have access to a large data of millions of events happening during the normal usage of *Visual Studio* by 81 developers. Using this data set, we answer three research questions that capture the impact IDE debuggers may have on programming behavior and outcomes:

- **RQ 1:** How effective are *Visual Studio*'s debugging features in reducing time and effort spent by programmers on debugging tasks?
  More specifically, we investigate how similar are the distribution of time spent on debugging with or without IDE features. We ask: Is there a significant difference between the distribution of the number of edits or navigation activity on source code during debugging periods that include (or avoid) the IDE's debugger?

MSR '18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196468

Responding to these questions contributes to understanding the influence of the debugger on debugging performance in terms of spent time, number of characters edited, and the number of navigation activities.

- **RQ 2:** When do the developers start using the IDE's debugging features? Do they start using the debugger right away, or do they postpone it to later time?
  This research question can provide insight into how people actually use the debugger, and whether they find it easy enough to use from the beginning of debugging.
- **RQ 3:** Do developers significantly change their editing behavior when they are debugging?
  Investigating the alteration of developers behavior in development mode versus debugging mode can demonstrate diverse needs of programmers, and encourage IDE designers to take them under consideration.

Unfortunately the corresponding data set does not supply information about the bugs under study. We would have liked to respond to questions related to the bugs themselves, such as on which kinds of bugs the IDE features seem to be most effective with respect to discovering the bug and fixing it. Since this data set is not suitable for such study, we leave this for future work.

3 DATA PROCESSING

For this study, we used the MSR Challenge *Events* data set, released on March 1, 2017. To distinguish between debugging mode and development mode, we need to have a clear definition of what we consider a bug, and how to detect the beginning and end of a debugging session. Several methods in the literature have been used to distinguish debugging from software development. In many studies, people look at version control history messages and comments to find bug fixing commits and edits [2]. In others, a bug is indicated by *failing test cases* in the project [6]. With the information available in the data set, we consider the time from a test failure until its success as a debugging period. All edits and events happening in those periods are considered debugging events. If multiple test failures happen before a successful build or test run, we treat the first failure as the beginning of debugging and subsequent failures as attempts to fix.

In total, we found 459 debugging tasks based on our definition, of which 281 are addressed without using the debugger. 178 include debugger events. To calculate the duration of debugging tasks, we summed the duration of all events during the debugging period. We do not simply calculate the duration using the start and end time of the debugging events, because this would include the IDE idle time, during which the user is not actively using the IDE. The activity events, representing times when the user is using IDE but not triggering an action, that distinguish active or thinking time from idle time are included in our calculations.

On average, debugging tasks last 16767 seconds in duration, with median of 623 seconds. The large difference between mean and median suggests the presence outliers in the data. By removing the tasks that spent more than 5 hours on debugging, we exclude 51 tasks (almost 10%) which results in mean of 1679 seconds and median of 466 seconds. From the remaining 408 debugging tasks, 269 do not use the debugger, and 139 do.

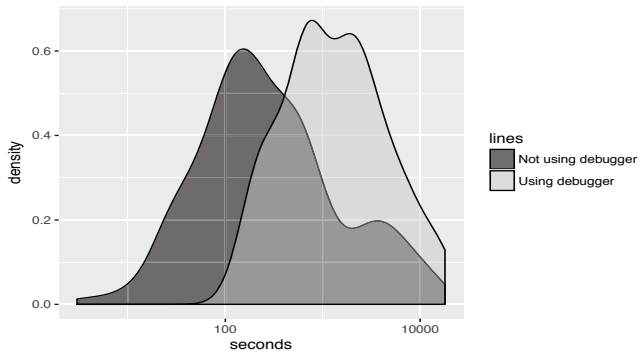


Figure 1: The normalized density plot of debugging duration, with and without the debugger.

4 METHODOLOGY AND RESULTS

In this section, we discuss the methodology used to explore each research question, and present the results.

4.1 RQ1: Debugger Impact

As described in Section 3, we divided the collected data in two: the first group used the debugger (indicated by debugger events in the data set) at some point during the debugging period, and the second group did not. These two datasets provide distributions of time spent on debugging, the number of edits needed to fix an issue, and the frequency of source code navigating events. We analyze these distributions with respect to the following null hypothesis:

> $\mathbf{H_0}$: The distribution of debugging duration is the same with and without the debugger.

Figure 1 presents the two distributions. On average, the mean time spent on debugging using the debugger is 1535 seconds higher than without the debugger. The 2-sample Kolmogorov-Smirnov test finds significant difference ($\alpha = 0.05$) between these two distribution ($p < 0.01$) with medium effect size (*Cohen's d* = 0.52). In other words, when developers spend a small but significant amount of additional time debugging when they use the IDE debugging features.

One possibility is that task difficulty might play a role in the decision to use the debugger. Developers may be more likely to use the debugger if the debugging task is difficult. Although the data does not provide information about the debugging task in question, we still can define proxies for difficulty. For example, we can use the number of files edited during a debugging period as a proxy for task difficulty. Editing more files does not necessarily mean that the bug is more difficult to understand. However, in general, defects that are distributed among several files may be harder to find and fix.

We constructed a linear regression model to include the duration of the task, the number of files edited, and whether the debugger is used. The model finds a significant relationship between duration and the number of edited files ($p < 0.01$), and finds a significant fit to using debugger ($p = 0.011$). Figure 2 shows the regression model, only considering duration and number of files. These results show a weak (R^2=0.15) but significant correlation between the difficulty

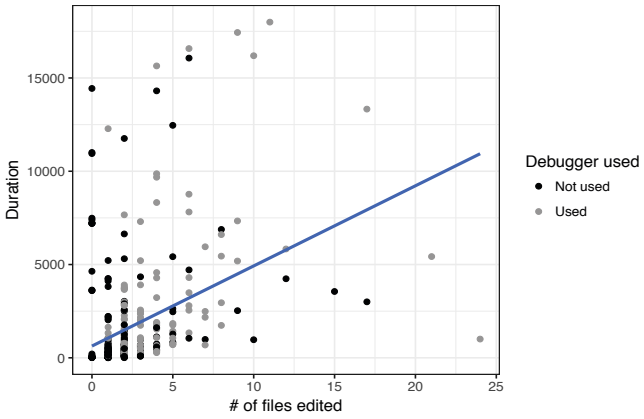


Figure 2: Linear regression model applied the number of edited files and the duration of debugging.

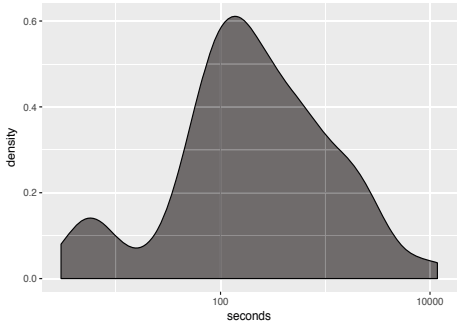


Figure 3: The density function of the seconds spent before starting to use the debugger.

of the task and the use of debugger, with time spent on debugging. We cannot simply reject the idea that the debugger is used on more difficult bugs and the difficulty could result in longer sessions.

Beyond time, it is possible that using a debugger may influence the number of edits a developer needs to apply to find and fix a defect. For example, if the developer is not using the debugger to localize the defect, they may add *print* statements to inspect variable values. We therefore examined the number of characters edited by the developer over the debugging period. As with duration, the mean number of edits is higher (by 1,317,324.4 characters) in cases where debugger was used. However, the t-test cannot reject the null hypothesis ($p > 0.1$); the number of edits made by developer does not statistically significantly differ with or without the debugger.

Developers spend almost 35% of their time navigating through the source code during maintenance activities [5]. As navigation is as an essential part of debugging, and representative of developer effort on fixing issues, we conducted the same experiment on the number of navigation events.[1] The mean number of navigation events is 69.17 using the debugger, and 34.74 not using it. The t-test show significant difference between these two distributions ($p < 0.01$). The number of times the developers navigated through the source code is significantly higher when the debugger is used.

In general, since the IDE and debugger are designed to simplify debugging, we expect that using them results in better performance in terms of fixing the issues faster, or with less effort. However, these results suggest that there are no such immediate effects from using the debugger either because the defects that the debugger was used for are genuinely more complicated, requiring developers to spend more time on debugging, or because using the debugger actually results in spending more time and effort on debugging.

4.2 RQ2: Starting Point of Debugger Use

For this research question, we study the effort that developers spend resolving an issue before switching to the debugger. One simple measure of effort is time: Developers may spend some time investigating a defect without the help of a debugger, switching to the debugger when it seems that it might be helpful. Another way of measuring effort is the number of times a developer navigates through the source code, an important debugging activity.

One possibility is that developers may first try to fix a bug issues without additional tools, seeking help from debuggers at the point that they realize the task is difficult. However, the data does not support this conjecture. Figure 3 shows the density function of the number of seconds spent on debugging before first use of the debugger. In 80% of cases, the developer started using the debugger in under 13 minutes; in 61% of cases, in under 5 minutes. Considering that the mean time spent on debugging tasks in this data is almost 45 minutes, this suggests that in most cases, the programmer decides to use the debugger early. Considering relative measures convey similar results. Overall, in 46% of sessions, the developer started using the debugger when less than 25% of the whole session is passed.

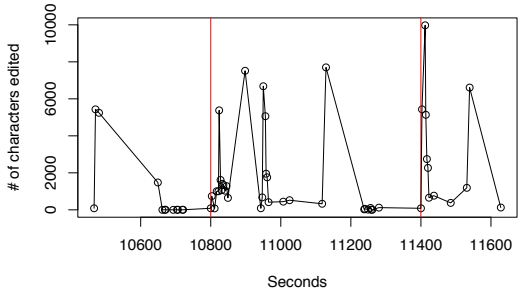
The number of navigation events show the same results. On average, developers navigate through code 18.34 times before using IDE debugger; on 72% of the tasks fewer than 15 navigation attempts were made before debugger use. The mean number of navigation events in the whole debugging period is 69.18.

In most cases, developers start using the IDE's debugging features relatively early in a debugging session. One possibility is that whether to use the debugger or not is mostly a question of user habit. We cannot further investigate this possibility with provided data set as it does not distinguish between individual developers.

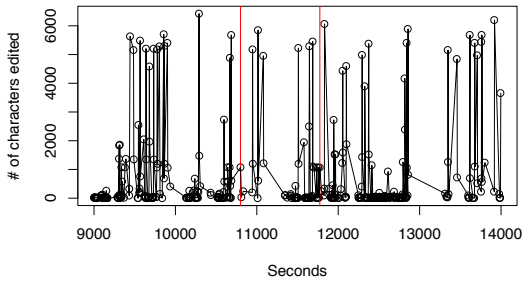
4.3 RQ3: Behavioral Change When Debugging

Finally, we investigate whether debugging affects the way developers edit their code. For example, they may frequently apply small edits to the source code when they develop code, but apply less frequent or larger edits while repairing defects. For this purpose, we conduct a time series analysis on every single debugging task. In this time series analysis, we collect data related to the number of edit events sorted by time, and treat the beginning and end of the debugging process as the interruptions. The beginning of analysis is at most three hours before first interruption, and the end is at most three hours after the second interruption. The time series analysis looks into significant changes in the distribution as interruptions occur.

[1] Each navigation event represents jumping from one soure code location to another.



(a) Significant change



(b) No significant change

Figure 4: Time series diagram of number of edits, with interruption points at the start and end of a debugging period.

Figure 4a shows an example of the time series analysis done on a debugging task by a sample developer. In this example, the number of edits increases when the developer starts debugging and drops right before it is done.

Although the example in Figure 4a is helpful in understanding the overall process and idea of this experiment, it does not show the majority of cases. Figure 4b shows one of the 63.3% of cases where we could not find a significant change ($\alpha = 0.05$) before and after the debugging period. In most cases, developers do not change their editing behavior as they debug.

4.4 Confounds and Threats

In all aforementioned methods, developer expertise and experience could be considered as confounding variables. However, we have no information about the developers and their level of expertise and have to rely on the assumption that the developers have been sampled from a balanced and diverse group from a variety of backgrounds and expertise levels [8].

Note that we identify debugging sessions as delimited between pass/fail and fail/pass transitions of the test suite. Although this is a common practice among studies [6], it could result in a sequence of activities that is falsely identified as debugging session.

As mentioned in Section 3, we excluded 10% of debugging sessions which are longer than 5 hours from our study. This decision could result in loss of valid data points that may affect the results.

5 CONCLUSIONS AND FUTURE WORK

We have explored debugging behavior using IDEs by mining the 2018 MSR challenge data set [8]. We showed that time spent on debugging is significantly higher when the debugger is used. One possibility is that this is related to the difficulty of the bugs under repair. Considering the number of files edited as a proxy for task difficulty, we show that there exists a statistically significant correlation between debugging duration and its difficulty. These results motivate future human studies to better understand this correlation. In particular, future user studies could carefully control confounds to solely investigate the impact of debuggers on task time.

We also found that, in a majority of cases, the developers switch to debugger mode relatively early while debugging. Future work can look into personal habits and preferences of developers to better explain these results. Finally, we conducted time series analysis to study the change in developers behavior at starting and ending points of debugging. Our study shows that in 63.3% of cases, no significant change is observable.

Overall, our results help us in better understanding developer debugging behavior, and more importantly motivating future research and improvements to IDE design.

6 ACKNOWLEDGEMENTS

This work is partially supported by NSF (#CCF-1563797); the authors are grateful for the support. All statements are those of the authors, and do not necessarily reflect the views of the funding agency.

REFERENCES

[1] Moritz Beller, Niels Spruit, Diomidis Spinellis, and Andy Zaidman. 2018. On the Dichotomy of Debugging Behavior Among Programmers. In *International Conference on Software Engineering (to appear) (ICSE 2018)*.

[2] Marcel Böhme, Ezekiel Olamide Soremekun, Sudipta Chattopadhyay, Emamurho Ugherughe, and Andreas Zeller. 2017. Where is the Bug and How is it Fixed? An Experiment with Practitioners. In *Joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2017)*. 1–11.

[3] Warren Harrison and Curtis Cook. 1990. Insights on improving the maintenance process through software measurement. In *Conference on Software Maintenance*. 37–45.

[4] Prateek Hejmady and N Hari Narayanan. 2012. Visual attention patterns during program debugging with an IDE. In *Symposium on Eye Tracking Research and Applications*. 197–200.

[5] Andrew J Ko, Htet Htet Aung, and Brad A Myers. 2005. Eliciting design requirements for maintenance-oriented IDEs: a detailed study of corrective and perfective maintenance tasks. In *International Conference on Software Engineering (ICSE 2005)*. 126–135.

[6] Claire Le Goues, Neal Holtschulte, Edward K Smith, Yuriy Brun, Premkumar Devanbu, Stephanie Forrest, and Westley Weimer. 2015. The ManyBugs and IntroClass benchmarks for automated repair of C programs. *IEEE Transactions on Software Engineering* 41, 12 (2015), 1236–1256.

[7] Gail C Murphy, Mik Kersten, and Leah Findlater. 2006. How are Java software developers using the Elipse IDE? *IEEE software* 23, 4 (2006), 76–83.

[8] 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 *Working Conference on Mining Software Repositories*.

[9] Martin P Robillard, Wesley Coelho, and Gail C Murphy. 2004. How effective developers investigate source code: An exploratory study. *IEEE Transactions on software engineering* 30, 12 (2004), 889–903.

[10] Cathrin Weiss, Rahul Premraj, Thomas Zimmermann, and Andreas Zeller. 2007. How long will it take to fix this bug?. In *International Workshop on Mining Software Repositories*. 1.