Studying Developer Build Issues and Debugger Usage via Timeline Analysis in Visual Studio IDE

Christopher Bellman
Carleton University
Ottawa, Canada
Christopher.Bellman@Carleton.ca

Ahmad Seet
Carleton University
Ottawa, Canada
Ahmad.Seet@Carleton.ca

Olga Baysal
Carleton University
Ottawa, Canada
Olga.Baysal@Carleton.ca

ABSTRACT

Every day, most software developers use development tools to write, build, and maintain their code. The most crucial of such tools is the integrated development environment (IDE), in which developers create and build code. Therefore, it is important to understand how developers perform their work and what impact each action has on their workflow to further enhance their productivity. In this work, we study the KaVE dataset of developer interactions within the Microsoft Visual Studio IDE and analyze a number of topics extracted from the data. First, we propose a method for developing what we call "*timelines*" that chronologically map an individual development session, and from this, we study build failures, code debugger usage, and we propose a metric for measuring developer throughput. We find that the timeline analysis may prove to be an invaluable tool for developer self-assessment and key to uncovering problem areas regarding build failures. Moreover, we find that developers spend a significant amount of time debugging their code, utilizing features such as breakpoints to resolve issues. Finally, we see that the developer metric can be used for self assessment, giving value to the amount of effort, put forth by a developer, in a given session.

CCS CONCEPTS

• **Software and its engineering** → *Software notations and tools*; *Software creation and management*;

KEYWORDS

Visual Studio, FeedBag, KaVE, IDE, Build, Debug

ACM Reference Format:
Christopher Bellman, Ahmad Seet, and Olga Baysal. 2018. Studying Developer Build Issues and Debugger Usage via Timeline Analysis in Visual Studio IDE. 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/10.1145/3196398.3196463

1 INTRODUCTION

Visual Studio is a widely popular integrated development environment (IDE) used all over the development landscape for business and personal needs, and includes a vast number of features and plugins to assist developers in their day-to-day work and collaboration [4]. According to a Microsoft developer blog [9], there are over 1.5 million developers (as of October, 2016) writing C++ in Visual Studio alone. Current developers often experience build failures due to problems in their code from recent changes or issues with refactoring. We aim at improving developers' efficiency and output by analyzing these issues they are having and providing insight into the their session-by-session timelines. Over time, Visual Studio has seen many new features added, both for efficiency and utility. Knowing that so many developers write code and depend on this IDE for their daily work opens a gap for further understanding of how this software is used and in what ways it can be improved to assist the efficiency of the developer. Determining which tools and features developers use often may shed light on future development of IDEs. Studying how developers use these tools and whether their productivity increases or decreases can help future IDE developers improve their software for better use.

In this paper, we study how developers spend their time when working in the Visual Studio IDE including the construction of what we call "timelines", which chronologically define the events that take place throughout a development session. Building a timeline for each developer's session allows us to explore the more significant issues faced by developers such as build failures and how they recover from them using debugging tools, and the potential impact this may have on a developer's ability to maintain high efficiency. Based on these timelines and an exploration of their impact on developer efficiency, we address the following three *research questions*:

(1) How often do code builds fail, and what contextual events are causing them?
(2) How do developers utilize debugging tools, and how much time is invested in them?
(3) Can timelines allow us to quantify developers throughput for a work session?

2 RELATED WORK

Existing research offers several relevant studies on build failures, how developers use debugging tools, and developer performance measures which, in our work, are all based on and built from the initial production of the timelines.

Build failures are, unfortunately, a natural occurrence in a developer's work. A study by Seo et al. [7] gathered over 26.6 million builds from Google's cloud build system and were able to gather a number of pieces of information and found that build failure rates are significant at an average of almost 33%. Further, they examined

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

ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196463

the reason for the build failures. We too examine reasons for build failures, but as a comparison to edits made. In their work, Seo et al. provide deeper insights into the reasons for failures.

Debugging tools are a near-essential piece in a developer's toolkit for rapidly solving problems related to runtime errors in their code. Murphy et al. [5] built a plugin for the Eclipse IDE and analyzed the data gathered from developers. They note that most (97%) of surveyed developers made use of the debugging tools and find that 70% of the developers made use of breakpoints in code. Zamfir and Candea [10] proposed a system for debugging that helps to automatically locate bugs in production software. Their automated bug finding can help improve developer productivity.

Generally, we can correlate code output with productivity, however code output may be less well-defined and other activities within development may be just as beneficial. A study by Thadhani [8] measured the effects of high-quality computer services and compared them with developer productivity values (e.g., lines of code written, etc.). They discussed the measurement of the number of lines of code a developer is outputting. Similarly, Maxwell and Forselius [3] discussed how mathematical models can be used to determine output for any sort of project in any field. While Thadhani's work uses the single value of a developer's lines-of-code output as a raw metric for determining performance, we propose a ratio-based scheme.

3 METHODOLOGY

The data mined in this work was provided by the KaVE Project [2] and used for the MSR 2018 Mining Challenge [6]. The dataset [6] consists of Enriched Event Stream information captured by the FeedBaG++ software that tracks developer interactions with the Microsoft Visual Studio IDE. The recorded events are: Activity, Build, Command, Completion, Debugger, Document, Edit, Find, IDE State, Navigation, Solution, System, Test Run, User Profile, Version Control, and Window events. The distribution of the number of each event is not even (Figure 1), potentially shining some light on how developers may use the IDE in other ways than the few we examine here. Each event contains event information, including a session identifier and time signature that allows us to determine when an IDE session event took place.

3.1 Session/Timeline Construction

Each event contains a unique session identifier which identifies which session an event was triggered in. For each session ID, all session data was collected and sorted to produce 3,417 total timelines. Some of the unique sessions consist of very few actions or event entries, leading to a session time length that is very short. For our analysis, any sessions lasting less than 10 minutes were not considered in the analysis. 2,417 of the original 3,417 sessions remained (70.73%) with an average session length of approximately 12.4 hours. Session length is determined by the length of time between the first and last events of a session (all events with a matching session ID). This session length is inflated due to a handful of extremely long sessions. These longer sessions were not removed as a session may still be a valid session and have useful data to analyze even if the IDE was left open and returned to at a later time. Longer session times were considered in analysis where appropriate.

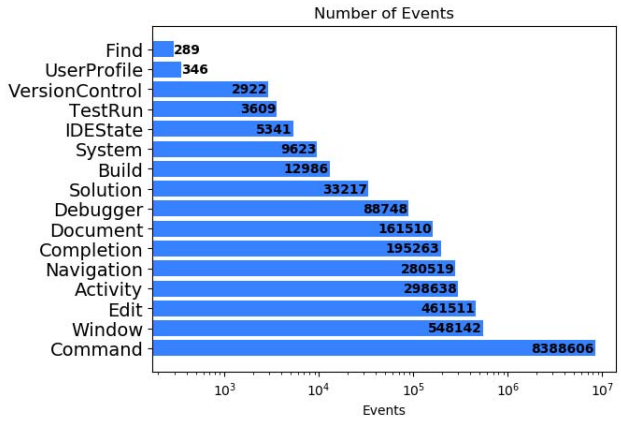

Figure 1: The number of times each event was seen in the dataset, log-scaled.

3.2 Build Failures

To determine the potential issues that may have caused build failures, a variety of contexts were investigated surrounding each build event. For each session, the following information was collected: time since last build event, build success status, number of pre-build Edit, Command, and Debugger events, number of post-build test-run and debug events, and if an exception was thrown after a build. Using this information, we were able to compute a variety of interesting statistics regarding build failures. Build failures were analyzed as singular events and statistics were computed across all valid sessions.

3.3 Debugger Usage

First, we looked at the context surrounding each Debugger event. This includes determining the time between debug entries, and the number of Edit events that precede the Debugger event. Second, further inspection of Debugger event entries allowed us to signify the start and end of the program execution and the intermediate incidence that occurred during code execution. In conjunction, by searching the intermediate incidents for specific keywords, we were able to draw several statistics from the data.

3.4 Developer Throughput Metric

We observed that certain events within the dataset tend to indicate more productiveness. We consider "productive" events to be events where more obvious steps towards a product goal were achieved. The productive events considered are: the Command, Completion, Build, Debugger, and Edit events. We attempted to measure what we call "developer throughput" by examining all useful actions done by a developer and comparing them to what we define for the purposes of comparison as the "optimal developer" (one productive event per second). To calculate this value, each session's productive events were summed and that sum was divided by the optimal working time to produce a value. The total optimal working time is the sum of every hour that has an event taking place within it.

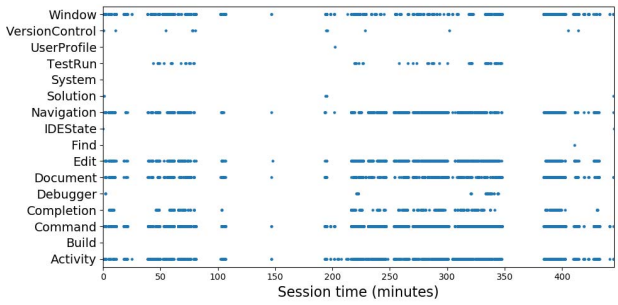

Figure 2: An example of a time-line plot for a single session of roughly eight hours in length.

4 RESULTS

In this section, we present our findings for the research questions.

4.1 Timelines

The core of our analysis revolves around the creation of session timelines. The goal is to provide developers with a look at how their time is being used within the IDE. The visual version of the timeline is a scatter-plot where each event is shown over time, as suggested by developer Benjamin Bengfort [1]. This method of viewing the data can be used to immediately determine the order of events. Figure 2 shows an example of these timeline plots for a session that lasted around eight hours in length. These can be easily traced and one can immediately tell where events may have happened.

Viewing representations like this can offer the developer insights into their session. The visual representation of a timeline is useful for a quick overview of a session's events, however, the real strength is in the organized sequence of events that can be further refined to gather more specific, targeted statistics. The timeline construction is used as the starting point for answering all three research questions proposed previously.

4.2 RQ1: How often do code builds fail, and what contextual events are causing them?

Over all examined sessions, 13% of builds failed. We examine a number of related statistics: 1.9% of all first-of-session builds fail, and 1.3% of all last-of-session builds fail. Seeing that these two numbers are similar, we may be able to make the connection and say that developers sometimes do not fix build problems before ending their session, so their next session may start with a build failure. On average, 598 edits are made before a failed build - 94% more than the average number of edits before a successful build of 308, indicating a relationship between code added and build failures. Similarly, there are, on average, 1,070 edits before the first failed build and 780 edits before the first successful build (37% less). If we consider that there are significantly more edits before both any failed build and the first failed build, we can see why the first-of-session build failures are generally higher than the last-of-session build failures.

Table 1: The top five developer session metrics of session lengths between 6 and 10 hours.

Session	Length (mins)	Value
02468b19-35f6 ...	460	0.2984
ac1ed485-1dbd ...	455	0.26507
30703dd7-a7c8 ...	417	0.23131
e0d29912-6643 ...	360	0.22286
94b661db-0e8e ...	386	0.21893

4.3 RQ2: How do developers utilize debugging tools, and how much time is invested in them?

Out of 966 unique sessions with Debugger events, we found that 81% of sessions build their code before execution. 55% of sessions perform actual code debugging with breakpoints, which indicates that a significant number of developers utilize debugging tools for code behavior inspection. Further, 32.5% of sessions contain runtime errors with 13 errors on average per session, of which 23.4% perform code debugging. Only 70% of sessions with runtime errors are investigated by code debugging, spending 122 min per session and performing 356 Edit events, on average. On the other hand, 30% of sessions with runtime errors never conduct code debugging. These sessions execute faulty code for 61 min with 150 Edit events, on average. 32% of sessions utilize the debugging tool to understand their code with no prior runtime errors. Developers appear to not use debugging tools when they believe they can fix errors quickly.

4.4 RQ3: Can timelines allow us to quantify developers throughput for a work session?

Table 1 shows some examples of session throughput metrics. The table consists of sessions that last between six and 10 hours. Based on the linear timeline of events – both productive and not – we can easily quantify a developer's throughput. We see that these sessions tend to average about 20 productive commands per minute versus the 100 productive commands that the "optimal" developer does (see Section 3.4). This is higher than expected for an average work-day length. A number of sessions were found to have a period of extremely high command repetition rates indicating a script or some automation was being done. Our comparison-based metric scores extremely high for these sessions, however they are unrealistic to compare to average developers.

5 DISCUSSION

In this section, we discuss our findings and implications of our work. We also address threats to validity.

5.1 Implications

The major purpose of the timeline is two-fold: for one, it provides an interface for other methods of analysis. Second, the timeline acts as a way for the developers to review their sessions. New tools need to be developed to analyze the timeline in a variety of ways, which can be personalized by each developer. A developer could easily extract very useful information such as times they were

distracted or productive. This quick visual analysis is the kind of information we primarily envisioned the timeline being used for by the developers themselves, however it should be considered as a stepping-stone for deeper analysis.

An interesting finding is that developers tend to write more code before a failed build than before a successful one. One hypothesis for this is that the more lines of code you write, the higher the chance of an error. Our findings suggest that an indicator for build failures is the number of edits that a developer is making before a build. Other potential events in the dataset such as the Window, System, or Find events do not contribute to the outcome of a build, and generally only the events that actually modify code or the way the IDE is running contribute to it.

The former motive showed us that developers invest at least two hours of their session troubleshooting exceptions, with the majority utilizing debugging tools. This indicates that they find it far easier to solve runtime errors by tracing their way through the code using the debugger. Although minor, our data shows that some developers never rely on the debugger to understand the cause of runtime error. A hypothesis for this is that developers might know where they made a mistake in their code and are able to fix it quicker than using the debugging tool, indicating that they may already know the source of the error. Given this, we can see that a significant portion of the developers choose to use the debugging tools and those that do can utilize them to great effect.

We believe that the timeline analysis allows us to quantify developer throughput per session. This metric is intended to provide developers with a convenient summation of their session's productivity and to be able to compare their current session's productivity with previous sessions. This allows a better understanding of their work and how they are performing on a session-to-session basis, potentially helping to highlight areas of high or low productivity. The metric was not intended as a foolproof system, but rather an experimental idea to augment the timeline-based analysis we have provided. Due to the events chosen to be included in the calculation of this metric, it may be the most beneficial when combined with other programmer performance metrics.

5.2 Threats to Validity

One of the main threats to the validity of this work is our choice for the low-end cut-off threshold of 10 minutes. This threshold was intended to help eliminate any noisy data that would not contribute well to the analysis. Conversely, no threshold was done on the upper-end of the session lengths and these longer sessions are potentially contributing to some inflation within the numbers, however, care was taken to reduce this bias. The User Profile event in the dataset is an optional event that the developers can fill out. As such, we are unsure of the skill levels of all developers included which may have biased some of the results such as debug time. All analysis in this work is performed within the scope of sessions rather than considering individual developers. Only about 10% of sessions had a User Profile event so the analysis could not be done on users exclusively.

6 FUTURE WORK

For potential future work, we envision a Visual Studio plugin being developed to make use of a timeline analysis. Adding a brief supplementary analysis of the developer's throughput metric would also be applied here as a more direct and easily referenced value to determine their performance in the session. We envision this being built off of the current FeedBaG++ plugin which gathers developer IDE interaction data. At the end of the session when the developers exit their session, their timeline is shown to them and they can explore a number of areas for analysis. This tool would be used as a way for a developer to reflect on his/her session. Offering the developers this overview can help them learn from their mistakes by providing insights into any problem areas or issues that caused a drop in productivity.

7 CONCLUSIONS

The aim of our work is primarily to explore each session where a developer does code development, in the context of a linear sequence of events. Based on this structure, we were able to further explore a few other areas where we feel insight can be given into events we see within the session timelines and how they impact developers. Producing both visual and pure data representations of session timelines, we were able to study build failures including some of the contexts and issues surrounding them, developer debugging habits including how often developers debug and the outcomes, and we proposed a metric for developer throughput which allows developers to get, at a quick glance, a single number that provides an indication of their throughput for the session. We hope that these four areas of analysis for the Visual Studio IDE interaction dataset provide an insight into how developers are using the IDE and its tools, and can help to promote future work regarding developer interaction.

REFERENCES

[1] Benjamin Bengfort. 2016. Timeline Visualization with Matplotlib. https://bbengfort.github.io/snippets/2016/01/28/timeline-visualization.html. (01 2016). (Accessed on 09/12/2017).
[2] KaVE Project. 2017. KaVE Project. http://www.kave.cc/home. (10 2017). (Accessed on 24/10/2017).
[3] Katrina D Maxwell and Pekka Forselius. 2000. Benchmarking software development productivity. *IEEE Software* 17, 1 (2000), 80–88.
[4] Microsoft. 2017. Visual Studio IDE. https://www.visualstudio.com/vs/. (2017). (Accessed on 01/10/2017).
[5] Gail C Murphy, Mik Kersten, and Leah Findlater. 2006. How are Java software developers using the Eclipse IDE? *IEEE software* 23, 4 (2006), 76–83.
[6] 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 *Proc. of the 15th Working Conf. on Mining Software Repositories*.
[7] Hyunmin Seo, Caitlin Sadowski, Sebastian Elbaum, Edward Aftandilian, and Robert Bowdidge. 2014. Programmers' Build Errors: A Case Study (at Google). In *Proc. of the 36th Int. Conf. on Software Engineering*. ACM, Hyderabad, India, 724–734.
[8] Arvind J. Thadhani. 1984. Factors Affecting Programmer Productivity During Application Development. *IBM Systems Journal* 23, 1 (1984), 19–35.
[9] Benjamin Xue. 2016. How Many Developers Use C++ vs. C# vs. Other Programming Languages. https://blogs.msdn.microsoft.com/zxue/2016/10/24/how-many-developers-use-c-vs-c-vs-other-programming-languages/. (10 2016). (Accessed on 01/10/2017).
[10] Cristian Zamfir and George Candea. 2010. Execution synthesis: a technique for automated software debugging. In *Proc. of the 5th European Conf. on Computer systems*. ACM, Paris, France, 321–334.