Empirical Study on the Relationship Between Developer's Working Habits and Efficiency

Ariel Rodriguez
Kyushu University
Fukuoka, Japan
roda@kyudai.jp

Fumiya Tanaka
Kyushu University
Fukuoka, Japan
tanaka@f.ait.kyushu-u.ac.jp

Yasutaka Kamei
Kyushu University
Fukuoka, Japan
kamei@ait.kyushu-u.ac.jp

ABSTRACT

Software developers can have a reputation for frequently working long and irregular hours which are widely considered to inhibit mental capacity and negatively affect work quality. This paper analyzes the working habits of software developers and the effects these habits have on efficiency based on a large amount of data extracted from the actions of developers in the IDE (Integrated Development Environment), Visual Studio. We use events that recorded the times at which all developer actions were performed along with the numbers of successful and failed build and test events. Due to the high level of detail of the events provided by KaVE project's tool, we were able to analyze the data in a way that previous studies have not been able to. We structure our study along three dimensions: (1) days of the week, (2) time of the day, and (3) continuous work. Our findings will help software developers and team leaders to appropriatly allocate working times and to maximize work quality.

CCS CONCEPTS

• **Social and professional topics** → **Project and people management**; • **Software and its engineering** → *Development frameworks and environments*; *Software development methods*;

KEYWORDS

Human Factors, Developer Working Hours, Developer Activity, Efficiency, Productivity, User Studies

ACM Reference Format:
Ariel Rodriguez, Fumiya Tanaka, and Yasutaka Kamei. 2018. Empirical Study on the Relationship Between Developer's Working Habits and Efficiency. 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/10.1145/3196398.3196458

1 INTRODUCTION

Long hours, demanding workloads, and overtime experienced by software developers have increased in recent years [1, 3]. Certain types of developers have been shown to be more active in the early morning hours, whereas other types keep more normal working hours [9]. This type of work behavior has been shown to adversely affect human efficiency and productivity, particularly in the early hours of the morning and after prolonged periods of working [2].

This study investigates the working habits of software developers based on their total time worked throughout the week, continuous working time, and work during particular periods of the day. By using the detailed events provided by the FeedBaG++ tool, we can obtain greater granularity compared to previous studies [2, 9]. The FeedBaG++ tool includes the ActivityEvent, which records all developer actions, and includes the Build and TestRun events which provide information when developers build or test a program. Based on the collected data for developer working habits, the effects on development efficiency are considered by evaluating the frequencies of success and failure events.

The dataset collected by the KaVE project's FeedBaG++ tool and used for the MSR 2018 Mining Challenge gives an ideal basis to investigate relationships between developer working times and developer efficiency. This study uses the interaction dataset, which contains events that FeedBaG++ users have shared. These events contain detailed information about developer interactions within the IDE. We use 328,667 separate events from a 11,122,103-event dataset to find when developers were actively working within the IDE and when they were building or testing their programs.

Looking into developers actions, we reveal their working habits over the the week and during particular times of the day. We then evaluate the effects of these patterns on developer efficiency. The findings of this study help developers understand their working habits and how they affect efficiency, thereby facilitating better allocation of work time.

2 CASE STUDY DESIGN

2.1 Research Questions (RQs)

RQ1 - Days of the week: A developer's efficiency is expected to vary depending on the day of the week [6, 8, 9]. For example, working times and efficiency are assumed to vary between weekends and weekdays, possibly related to emotional factors surrounding the weekend [4]. Under this RQ, we try to answer whether the day of the week has an affect on development time and efficiency. This is motivated by the idea that if a day of the week can be correlated with development efficiency or quality, our finding can be used to increase the overall efficiency.

RQ2 - Time of the day: Under this RQ, we try to answer whether or not there is a relationship between development efficiency and working times within a 24-hour period. Identifying the times associated with higher and lower development efficiency will help developers plan their routines to produce the best results [9].

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

ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196458

Table 1: Overview of the dataset used in this study

Dataset	Event	Field
All Data (11,122,103)	ActivityEvent (311,245)	TriggeredAt (311,245)
	BuildEvent (13,813)	Successful:True (13,195)
		Successful:False (1,786)
	TestRunEvent (3,609)	Result:Success (2,416)
		Result:Failed (1,563)

RQ3 - Continuous working time: Under this RQ, we try to answer whether continous working time has an affect on development efficiency. Long continuous working times can lead to concentration loss [4], negatively affecting efficiency. This analysis aims to investigate if efficiency declines with continuous work.

2.2 Dataset

The dataset used in this study is the MSR 2018 Mining Challenge dataset, which was obtained through the KaVE Project's CARET platform [5]. The CARET platform records events along with contextual information when a developer performs an action within Visual Studio; these recorded actions can be anything from clicking a menu or selecting a new window.

The dataset contains data shared by 81 different developers of different skill levels and backgrounds; the data represents a total of 15,000 hours of work by developers.

Table 1 shows the events and fields used in this study along with the amounts of each event presented in the dataset. The events chosen allow us to determine the amount of a developer's time spent actively in the IDE as well as the success and failure rates when building or debugging a program.

Work Time Related Measure: Figure 1 shows an example of the data schema related to the ActivityEvent of user 001. The ActivityEvent is triggered when any developer activity (a mouse click or movement, typing, etc.) occurs inside the IDE. The TriggeredAt field in the ActivityEvent contains information about the time, date, and time zone of the triggered ActivityEvent. The activity time along with the time ranges during which the developer was working were determined using the TriggeredAt field.

Activity time refers to the time when a user is performing a task or action within the IDE and allows us to determine the amount of time a user is actually working within the IDE. When an ActivityEvent is detected, the time that the event was triggered is recorded. If another ActivityEvent is detected within five minutes of the preceding event, the time between those events is considered active time. The individual recorded time intervals are summed to calculate the total activity time.

Efficiency Measures: Since it can be difficult to judge the efficiency of a developer, we use a quantitative approach based on the Build and TestRun events. The BuildEvent is triggered when a developer builds their program, and the TestRunEvent is triggered when a user runs a test. These events contain a "Successful" field in the BuildEvent and the "Result" field in the TestRunEvent; these subfields indicate the success or failure of the Build or TestRun event. In this study, these values are used to calculate developer efficiency and establish correlations between efficiency and working times. The cumulative amounts of successful and failed Build and TestRun events during different days and time periods are analyzed to identify any patterns or correlations.

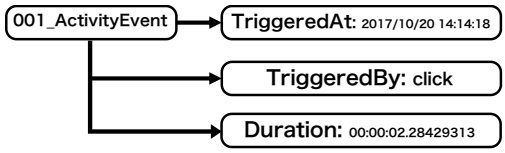

Figure 1: A part of data schema in the Challenge Data

3 CASE STUDY RESULTS

3.1 RQ1: Days Of The Week

Approach: Using the Activity, Build, and TestRun events, we compare working time and efficiency by day of the week. As shown in Figure 1, the TriggeredAt field contains the dates and times of triggered events in 24-hour format. The date part of this field can be used to determine the day of the week on which each event was triggered. To calculate work time, all generated ActivityEvents in a day are sequentially put in a list; the work time can then be calculated using the activity time method.

We use a similar approach to collect success and failure data, since the Build and TestRun events also contain the TriggeredAt field in the same format. We calculate success and failure rates using the fields shown in Table 1. The success and failure rates show the percentages of events that include success or failure fields within a particular event. A Build and TestRun event can contain multiple sub-events; therefore, the number of events in the Event column does not match the sum of numbers in the Field column.

We use chi-square tests and residual analysis for crosstabs to test the difference in ratios for days of the week, time ranges and continuous time ranges for each RQ respectively. The null hypothesis is that there is no difference in the population rate between days of the week in the chi-square test. By rejecting this, we show that there is a significant difference in the ratio.

Result: Figure 2 shows the total working time and continuous working time by day of the week. The total working time shows a decreasing trend after Thursday, and the total working times on the weekend are roughly half those of Monday, Tuesday, and Wednesday. The continuous working rate (i.e., the ratio of continuous working time to total working time) is the highest for Saturday (63%); for all other days, the rate was within 50±3%. The results of this study show that Friday is the weekday with both the lowest total working time and continuous working time. A previous study [8] also shows Friday to be the day where the most bug causing changes are made to code. Similarly to the results of the previous study, in our study we can see that Friday also negatively affects work possibly due to its proximity to the weekend. The high rate of continuous work on Saturday might result from developers being able to focus better on a particular task and have more time. Despite this, continuous work is not high on Sunday. This might reflect societal factors since people often go out with their families or attend social events on Sunday.

Figure 3 shows the success and failure rates by day of the week. Based on the residual analysis, the success rates are significantly higher on Wednesday and Sunday compared to on other days, whereas the failure rates are lower. The opposite trend is observed for Monday and Tuesday. These results indicate poor development efficiency on Monday and Tuesday, possibly resulting from a lack of

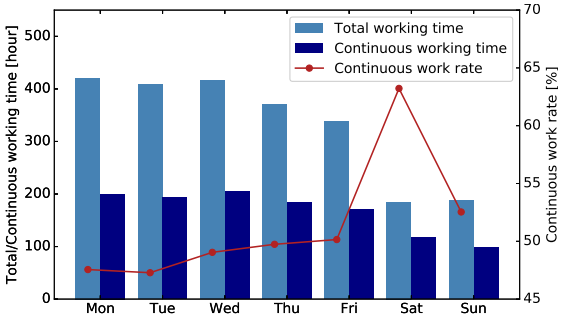

Figure 2: Total and continuous working times by day of week

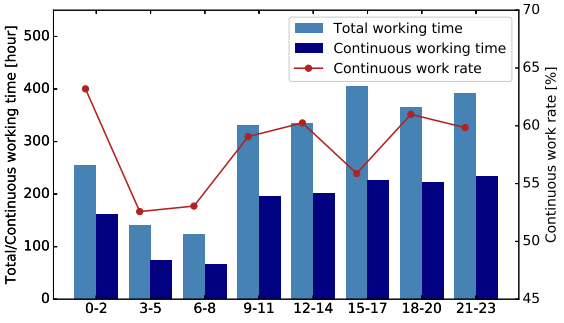

Figure 4: Total and continuous working time by time range

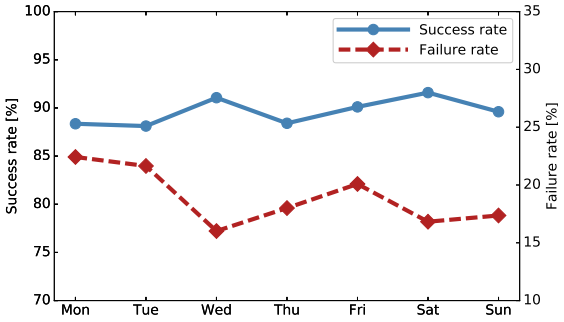

Figure 3: Success and failure rates by day of the week

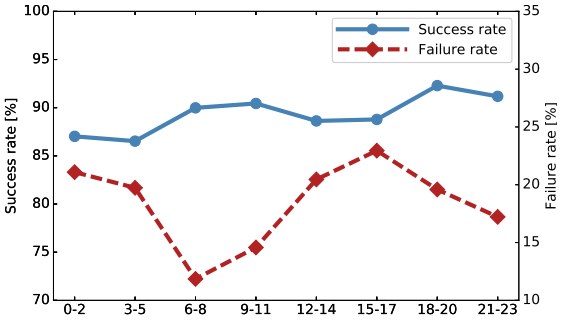

Figure 5: Success and failure rates by time range

motivation after the weekend or delays in getting back up to speed with the work that was performed in the previous week.

> Developers are more likely to work continuously on Saturdays. Efficiency seems to be lower at the beginning of the work week compared to later days.

3.2 RQ2: Time of the day

Approach: We use a similar approach used in the prevous section (i.e., the same events, particularly the TriggeredAt field, are employed). However, for RQ2, the date in the TriggeredAt field is used to identify events generated at a particular time of the day rather than dividing the data into groups by days. This event uses the local time for each developer, allowing us to collect and compare data across all developers.

Result: Figure 4 shows the total and continuous working times along with the continuous work rate divided into three hour blocks in a 24 hour period. The greatest total working time occurred in the 15-17 time range. Since the working day ends at 17:00 hrs for most people, the increased working time in this period might reflect workers trying to finish their work before the end of the day. In no time period was no work performed; however, an expected drop off in total work was observed in the 3-8 range. These results confirm our initial assumption that developers keep irregular working hours and often work into the early morning.

Figure 5 shows the rates of success and failure for all developers. The maximum failure rate (23.0%) was observed in the 15-17 range, whereas a high success rate (> 90.0%) is found between 18 and 23. A comparison of Figure 4 and Figure 5 indicates that the trend in failure rate generally follows the trend in total working time. This suggests that time periods with high total work time also have a low work efficiency. This could be due to a lack of concentration or not putting the same amount of thought into work because of lack of time or stress factors. The success rate in the 18-23 time range is significantly higher than those of the other time ranges, whereas the success rate is lowest from 0-5.

> Time periods with high total work times show high failure rates. Development efficiency improves in the evening (before midnight) and then declines after the day changes.

3.3 RQ3: Continuous Working Time

Approach: Under RQ3, we partition the collected events based on duration of continuous work and the total number of successful or failed events that fell within that period of continuous work.

To calculate continuous working time, the activity time is used to determine the length of the session in the IDE, the activity time records activity until the developer becomes inactive for a period five minutes. At that point the session is considered to be over and all successful and failed events up to that point are totaled and added to the data for that group.

Result: Figure 6 shows the success and failure rates after different periods of continuous work. After working continuously for two hours, the success rate decreases by 6.7%, whereas failure increases

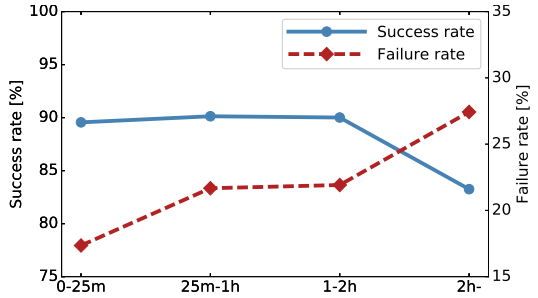

Figure 6: Success and failure rates by continuous work time

by 5.5%. The increase in failure rate after working for 25 continuous minutes is 4.3%. These results indicate that the success rate does not change significantly during the first two hours of continuous work; however, the failure rate is lowest within the first 25 minutes of work. Based on the residual analysis, working for more than two continuous hours significantly increases the failure rate and decreases the success rate. This reinforces previous findings that prolonged work time can affect efficiency [2, 3, 7]. For example, Meyer et al. [3] found that developers feel particularly productive when they are in a "flow" state, which is a period of time without having switches or breaks. We show that developers working continuously up to 2 hours also have good efficiency; however, we found that after 2 hours efficiency decreased.

> Working for long periods of time results in increased failure rate. Working continuously for more than two hours increases failure rate by 10% compared to shorter periods.

4 THREATS TO VALIDITY

External Validity: The main threat to the validity of this study is generalizability; all data within the dataset is taken from workers within the UTC+1 and UTC+2 time zones. While the effect of all participants residing in the same time zone is difficult to determine, results based on data collected in different regions might differ because of weather or cultural factors. Furthermore, the data used in this study was collected from people with varying experience levels from professional to hobby programmers; it is not clear whether different results would be obtained if the data was collected from only one type of programmer.

Internal Validity: In the calculation of activity time, a five minute window of inactivity is allowed before declaring the end of an activity session, this is done because developers may spend time reading code and/or considering methods/options; thus, they may be inactive within the IDE itself but still active on the project. This could cause the activity times of developers that regularly spend more than five minutes thinking about or reading code to be inaccurately calculated. However, catering the calculation of activity time to developers that routinely spend prolonged periods of time thinking about their work would alter the results for average developers. In addition, the measure of activity time is primarily focused on active time inside the IDE; thus, we consider five minutes to be an appropriate middle ground.

Construct Validity: The success and failure rates in this study are taken from the Build and TestRun events since they contain the appropriate fields that focus on success and failure; other events are not considered. Skill level of developers likely determine familiarity with the IDE and how often they use these events. Thus, some developers may use these events more than others.

5 DISCUSSION AND CONCLUSIONS

This paper empirically evaluated the working times of software developers and how they affect efficiency. The results reveal interesting correlations between developer working habits and work efficiency based on the success and failure rates of build and test events. The findings suggest that software developers work irregular hours, sometimes into the early morning; however, working during these times does not necessarily have a significant effect on efficiency. Poor developer efficiency may be attributed to poor concentration on a task during specific days and time ranges. The results indicate that working continuously for more than two hours without a break negatively affects developer efficiency, as evidenced by increased failure rate and decreased success rate.

Having this information about developer efficiency allows team leaders, managers or the developers themselves to plan their work times to be as efficient and productive as possible. Practical implementations could be scheduling non development tasks such as meetings during times where developers are less efficient and using more efficient times for actual development work. Managers could also use this information to efficiently plan and anticipate issues in projects by knowing which days developers could fall behind on work.

ACKNOWLEDGMENTS

This work was supported by JSPS KAKENHI Grant Numbers JP15H05306.

REFERENCES

[1] Jeff Hyman, Chris Baldry, Dora Scholarios, and Dirk Bunzel. 2003. Work-Life Imbalance in Call Centres and Software Development. *British Journal of Industrial Relations* 41, 2 (2003), 215–239.
[2] Harrington J. Malcolm. 2001. Health effects of shift work and extended hours of work. *Occupational and Environmental medicine* 58, 1 (2001), 68–72.
[3] André N. Meyer, Laura E. Barton, Gail C. Murphy, Thomas Zimmermann, and Thomas Fritz. 2017. The Work Life of Developers: Activities, Switches and Perceived Productivity. *IEEE Transactions on Software Engineering* 43, 12 (2017), 1178–1193.
[4] Sebastian C. Müller and Thomas Fritz. 2015. Stuck and Frustrated or In Flow and Happy: Sensing Developers' Emotions and Progress. In *Proc. of the International Conference on Software Engineering (ICSE)*. 688–699.
[5] 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 Working Conference on Mining Software Repositories (MSR)*.
[6] Emad Shihab, Akinori Ihara, Yasutaka Kamei, Walid M. Ibrahim, Masao Ohira, Bram Adams, Ahmed E. Hassan, and Ken-ichi Matsumoto. 2013. Studying Reopened Bugs in Open Source Software. (2013), 1005–1042.
[7] Pankaj Singh, Damodar Suar, and Michael P.Leiter. 2012. Antecedents, Work-Related Consequences, and Buffers of Job Burnout Among Indian Software Developers. *Journal of Leadership & Organizational Studies* 19 (2012), 83–104.
[8] Jacek Śliwerski, Thomas Zimmerman, and Andreas Zeller. 2005. When do changes induce fixes?. In *Proc. of the International Workshop on Mining Software Repositories (MSR)*. 1–5.
[9] Masateru Tsunoda, Akito Mondenand Takeshi Kakimoto, Yasutaka Kamei, and Ken-ichi Matsumoto. 2006. Analyzing OSS developers' working time using mailing lists archives. In *Proc. of the International Workshop on Mining Software Repositories (MSR)*. 181–182.