Data Mining in-IDE Activities: Why Software Developers Fail

Naw Safrin Sattar, Md A. M. Faysal, Minhaz Zibran, Shaikh Arifuzzaman, Md Rakibul Islam
Department of Computer Science, The University of New Orleans
New Orleans, Louisiana, United States
(nsattar, mfaysal, mzibran, smarifuz, mislam3)@uno.edu

Abstract

This paper addresses the hindrances behind software developers' failures as perceived from software build failures. We capture and correlate the routines and patterns of developers' interactions in IDE, their backgrounds, expertise, and geographic locations with their failure instances. Our study is based on a large dataset of 85 developers' 11 million interactions/events in Microsoft Visual Studio IDE over 15,000 work-hours. The findings from this study will help developers and organizations in shaping their working style for higher success rate.

Keywords: build failure, data mining, IDE, exploratory study, quantitative analysis

1 Introduction

Unlike many other industries, software industry is highly dependent on human efforts from the developers. Modern software engineering incorporates various techniques and automation tools to speed up the core software engineering activities such as development, testing, and integration. Automated build and Continuous Integration (CI) have become common practice in industry for building and integrating contributions from multiple developers participating in a project. Software build process includes tasks such as assembling program components, data and libraries, then compiling the source code linking with libraries and external resources, running the tests, and packaging everything together to create an executable software system.

When any of the underlying tasks in the build process fails, the built attempt is also considered to have failed. In CI, a build process is initiated whenever contributions (e.g., code changes) from a human developer is pushed to the codebase for integration. A build may fail for various reasons such as compilation failure due to syntactic or linking errors, or test failure due to semantic flaws. Such failures typically occur due to human errors made by the developers.

This paper presents an exploratory study, where we analyze the behavioral and working patterns of developers to identify factors [6] that increase the chances of human errors causing build failures. The findings from this study are derived by mining and examining a large dataset consisting of 85 developers' 11 million in-IDE interactions/events over 15,000 work-hours.

From quantitative analyses of this large dataset, we found that frequent changes make source code vulnerable to failure. Developers who do not choose their work-hour wisely are more prone to failure despite their extra efforts. Beside expertise, developers' formal institutional education also influences the likelihood of their success at work. The results are validated in the light of statistical significance.

Outline: The remaining of this paper is organized as follows. Section 2 describes the setup of our study including our methodology for data collection and analysis. Section 3 presents the findings from this study. In Section 4, we discuss the possible threats to the validity of this work. Section 5 includes a discussion of related work, and finally Section 6 concludes the paper.

2 Methodology

2.1 Extraction and Parsing of Data

We use a large dataset made publicly available by the KaVE Project [5]. This dataset includes 85 developers' around 11 million events/interactions in Microsoft Visual Studio IDE over a period of more than 15,000 work-hours. The interactions are captured using `FeedBaG++` interaction tracker and recorded in **json** format. For different in-IDE activities, around 20 types of events have been captured [3]. We parse the data and collect the users' information, their involvement in the IDE, their interaction with the code, their build activity and their test activity. After parsing through the huge data set, we get the following events given in Table 1 required for our study. We store extracted data in **csv** format and directly export to MySQL database.

978-1-943436-13-2 / copyright ISCA, SEDE 2018
October 8-10, 2018, New Orleans, Louisiana, USA

Table 1: Analyzed Event Streams

Event Name	Description	Total
UserProfileEvent	User information	380
EditEvent	Track of edits	497514
ActivityEvent	Developer active in IDE	316927
BuildEvent	Build action of developers	15052
TotalBuildEvent	Target projects in each build	83431
TestRunEvent	Tests run by developers	3876
TotalTestRunEvent	Test results for each test	338381

2.2 Mining Data

83 unique users' multiple sessions create 380 **UserProfileEvent**s. Two users (id: 41, 84) having no profile information but their activities captured in other Events, comprises 85 developers. Figure 1 shows the tables of our database and the common attributes of each event is also shown separately. We have sorted out the individual events of each developer. In order to find out the behavior of each developer distinctly, we follow Algorithm 1 to generate database table **UserALLEvents**, attributes shown in Figure 1.

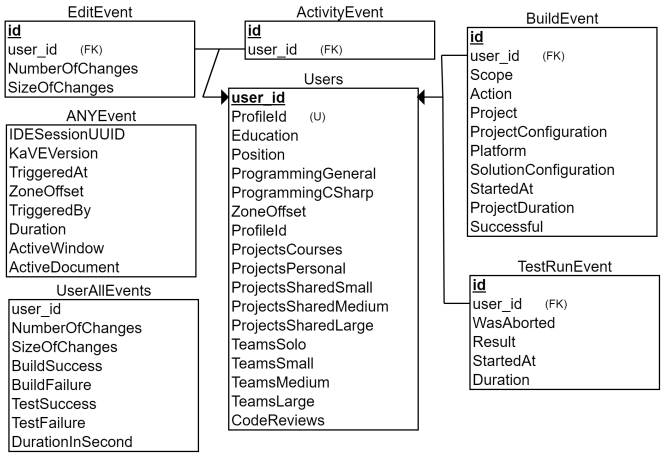


Figure 1: Database Schema with Attributes

As we aim at dealing with the Failure/Success of a developer, both **TestRunEvent** and **BuildEvent** need to be evaluated. After analyzing data of **UserAllEvents**, we figure out that we have information of only 24 developers corresponding to TestSuccess/TestFailure. The target audience being so small, we focus on only **BuildEvent** to analyze Failure of the developers. Some developers have neither Successful Build Event nor Failed Build Event. So, we exclude those data from our study and work with the data of 76 developers.

The **UserProfileEvent** of a developer has several attributes but we focus on his Programming Knowledge, Education Level and Geographical Location to analyze further. The following subsections describe how we deal with these criteria.

Algorithm 1: Update Individual UserEvents

```
1  for Each user in Users do
2      QUERY NumberOfChanges, SizeOfChanges
         from EditEvent;
3      SUM(NumberOfChanges),
         SUM(SizeOfChanges);
4      QUERY TriggeredAt from
5      ActivityEvent g1 INNER JOIN
         ActivityEvent g2
6      ON g2.id = g1.id + 1 AND g1.'TriggeredAt' =
         g2.'TriggeredAt' ;
7      Duration ← SECOND(g2.'TriggeredAt' −
         g1.'TriggeredAt');
8      SUM(Duration);
9      QUERY Successful from BuildEvent;
10     if TRUE then
11         SUM(BuildSuccess);
12     end
13     if FALSE then
14         SUM(BuildFailure);
15     end
16     Query Result from TestRunEvent;
17     if Successful then
18         SUM(TestSuccess);
19     end
20     if Failed then
21         SUM(TestFailure);
22     end
23     UPDATE UserAllEvents;
24 end
```

2.2.1 Expertise of Developers

Table 2: Categorization of Developers by Expertise

User Type	(Programming General, Programming C#)
Expert	(P,P)
Novice	(N,N),(N,n),(n,N),(n,n)
Medium	(P,N), (N,P), (P,n), (n,P), (P,U), (U,P), (N,U), (U,N), (n,U), (U,n)
Unknown	(U,U)
Here,	P=Positive, N=Negative, n=Neutral, U=Unknown

Of these 76 developers, 17 of them have not provided their expertise of Programming Knowledge. The values defining their Programming Skill are within the range - 3 to +3 where 0 denotes Neutral. So, according to their expertise, the categorization is shown in Table 2.

2.2.2 Geographical Locations

We aim at analyzing developers behavior according to their Geographical Location. All the events have been captured at local time of the developers. So, we can identify their location from the **TriggeredAt** attribute. This field is a **Java ZonedDateTime** Object. So, we capture the **ZoneOffset** from the timestamp and add it as an additional attribute in the database. All these developers belong to 15 different time-zones. According to the **UTC Offset** [2], we get that UTC -08:00 to -03:00 covers both North America and South America. Accordingly, we divide these zones into three inter-continental regions. Table 3 shows the classification.

Table 3: Categorization of Geographic Regions

Continent	ZoneOffset	Region
North America & South America	-08:00, -07:00, -06:00, -05:00, -04:00, -03:00	1
Africa & Europe	+00:00, +01:00, +02:00, +03:00	2
Asia Asia & Ocenia Ocenia	+07:00 +08:00, +10:00 +12:00, +13:00	3

2.2.3 Work-Hours

We also assess the developers behavior during Working Hour and Leisure Hour. We divide the 24 Hours of day into two time-intervals each with 12 hours. We have considered 8AM-8PM as Working Hour and 8PM-8AM as Leisure Hour. As, different countries around the globe may follow different schedule for work-time, adherence to this difference, we pick the 12 hour schedule. We distribute data of **UserALLEvents** into two similar database tables according to the Time Division.

3 Analyses and Findings

The findings from this study is organized in accordance with the criteria/attributes described in the previous section. To examine the statistical significance of the results, we use Pearson Product Moment Correlation Coefficient [7] and Chi-Square test [13] with significance probability level $p = 0.05$.

3.1 Edit Frequency

When a user frequently changes code, it has a negative impact on success. From Figure 2, we can see that with higher number of changes in code, success rate decreases. Again, users having higher success rate do fewer number of edits.

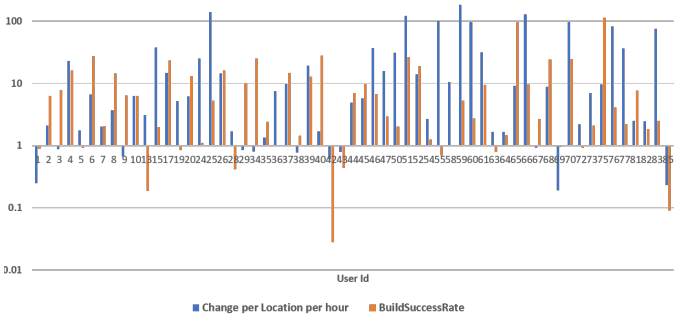


Figure 2: Relationship between Edit and Success

Table 4: Edit and Success Related to Expertise

Expertise	Edit	Success(%)	Education (%)
Expert	0.1644	96.3110	78.9474
Novice	1.1456	94.8454	66.6667
Medium	3.4679	92.2469	42.8571

We find that *Expert* users tend to do less edit compared to *Novice* users. But users who belong to *Medium* category do more edits compared to the *Novice* and their success rate is lower than that of novice, reflected in Figure 3.

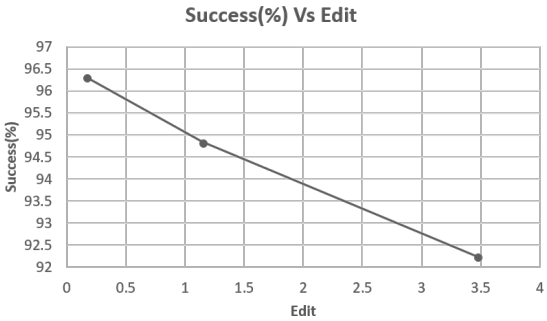
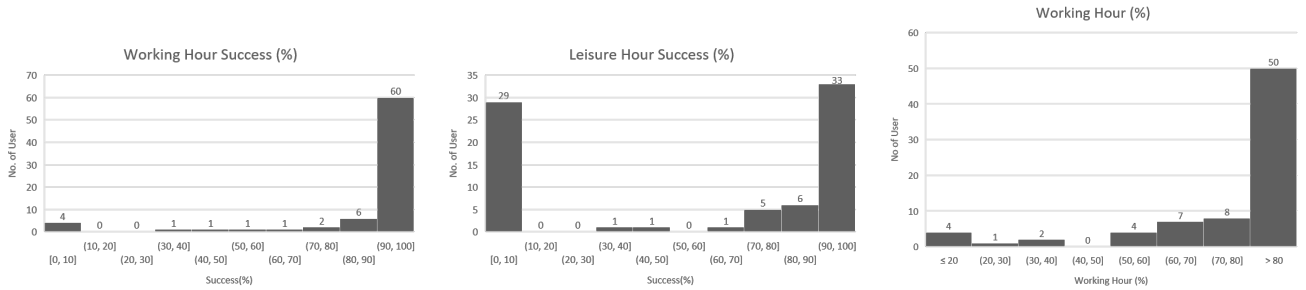


Figure 3: Success Rate at Different Edit Frequency

Table 4 explains the reason. Although *Novice* developers have less programming knowledge, they have proper education compared to the developers belonging to *Medium* Category. Edit sizes based on number of changed locations and size of changes in terms of characters induce the frequency of build success or failure. Tracking and fixing the causes of the build failure becomes difficult due to large amount of edits that may reside in different locations in the code.

We have found that success decreases when there is an increase in Edit. We compute the Pearson correlation coefficient, r, between all categorized users' edit frequency and success rate. $r = -0.0234$, indicated a weak negative correlation. Again, *Novice* developers have a lack of programming knowledge and so they are tend to make more changes in code. For them $r =$



(a) Regular Working-Hour (b) Leisure-Hour (c) % of Time Spent during Working-Hour

Figure 4: Relationship between Work-Hour and Success Rate

-0.3225, which indicates a slightly stronger negative correlation between success rate and edit frequency.

3.2 Work-Hours

Working during late-hours or very early-morning is one of the reasons for high failure rate. Reasons behind this could be lack of concentration, fatigue. About 38.16% of developers who work after/before their regular work hours have a failure rate over 90% shown in Figure 4a. Only 38.15% of them have a success rate over 90% which is below average.

On the other hand, Figure 4b reflects that only 5.26% of developers working in their regular work-hour have a failure rate over 90% and 78.94% of them have success rate over 90% . But it's a matter of relief that, 65.78% developers spend more than 80% of their total hours in regular work-hour perceived from Figure 4c. Although only 5.26% developers spend over 80% of their total time during leisure hour, this small number is insignificant and we can consider it as their personal trait. Still, 28.94% developers does not follow regular work-hour strictly. If this portion of the population can be directed to maintain a consistence work-time, it will contribute to increased success rate.

Table 5: Work-Hour and Failure (Observed and Expected) [F.=Failure]

	Low F.		**Med. F.**		**High F.**		**Total**
	Ob.	Ex.	Ob.	Ex.	Ob.	Ex.	
Work	60	46.5	12	13	4	16.5	76
Leisure	33	46.5	14	13	29	16.5	76
Total	93		26		33		152

To examine the statistical significance of the observations, we apply Chi-square test. The contingency table for the test is presented in Table 5. The Chi-square test, with $\chi^2(df = 2, N = 152) = 26.932, p = 0.05$, suggests statistical significance of the observed relationship between developers' work-hours and failure rates.

3.3 Geographical Regions

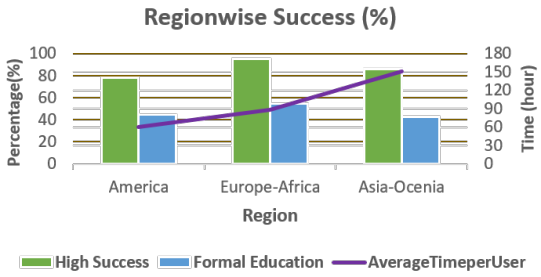


Figure 5: Region-wise Success Rate

In examining the success rate of the developers across geographical regions, we have considered above 75% of success rate as High Success. According to Figure 5, among the developers, 95.23% of them belonging to Europe-Africa region, have High Success. On the contrary, North-South-America region, has the lowest percentage of 77.78%. Again, Asia-Oceania region has 85.71% people, having High Success. This success depends on the expertise of the developers, their education and time spent in work summarized in Table 6.

Table 6: Region-wise Success Rate

Criteria	**Regions**		
	1	**2**	**3**
Success (%)	77.78	95.24	85.71
Expert (%)	55.56	54.76	85.71
Novice (%)	18.52	7.14	0
Formal Education (%)	44.44	54.76	42.86
Avg. Time/User (hour)	60.33	87.79	150.45

Although America and Europe-Africa regions have similar percentage of expert developers, Europe-Africa region has 54.76% formal education compared to 44.44% education among the Americans. Again in Asia-Ocenia region, although the percentage of expert developers is 85.7%, their education percentage is only

42.85%. So, in Asia-Ocenia region, they have increased their success rate by increasing their time spent in work compared to the developers in Europe-Africa region. Developers in America region spend less time compared to the developers in Europe-Africa region.The success rate in America region is also low than that of the Europe-Africa region.

Table 7: Geographic Region and Failure (Observed and Expected) [F.=Failure]

	Low F.		High F.		Total
Region	Ob.	Ex.	Ob.	Ex.	
1	78	86.333	22	13.667	100
2	95	86.333	5	13.667	100
3	86	86.333	14	13.667	100
Total	259		41		300

To verify the statistical significance of the observations, we again apply Chi-square test. The contingency table for the test is presented in Table 7. The Chi-square test, with $\chi^2(df = 2, N = 300) = 12.261, p = 0.05$, suggests statistical significance of the observed relationship between developers' geographic locations and failure rates.

3.4 Educational Backgrounds

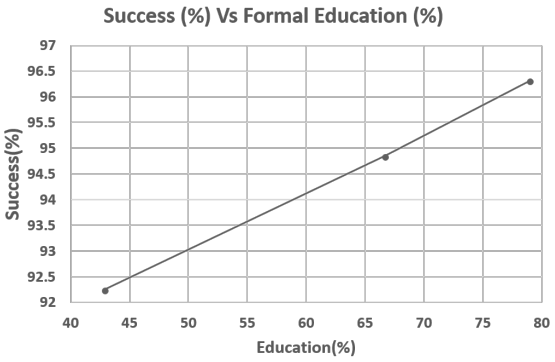


Figure 6: Relationship between Education and Success

Table 8: Success Rate and Educational Backgrounds

Expertise	**Success (%)**	**B**	**M**	**P**	**A**	**F.E. (%)**	**A (%)**
Expert	96.31	12	17	1	7	78.95	18.42
Novice	94.85	2	2	0	1	66.67	16.67
Medium	92.25	2	1	0	4	42.86	57.14
Here,	B=Bachelor, M=Masters, P=PhD, A=AutodidAct, F.E.=Formal Education						

According to the user profiles of the developers, their **Education** attribute has the following values: *Bachelor*, *Master*, *PhD*, *AutodidAct*, *Training*, *None*, *Unknown*. *AutodidAct* has been assigned to the developers who are self-learners. Developers having Formal Institutional Education, have more success rate irrespective of their programming knowledge. According to programming knowledge, developers having medium expertise tend to have more success rate than the novice developers reflected in Figure 6. But here the scenario is different. 66.67% of novice developers have Quality education compared to 42.81% of Medium-Expertise Developers. Again, 57.14% medium-expertise developers are self-learners. So, this decreases their success rate. Table 8 summarizes the results. So, Institutional Education plays a vital role to improvise the efficiency of a developer.

Table 9: Education and Failure (Observed and Expected)

	Low F.		Med. F.		High F.		Tot.
	Ob.	Ex.	Ob.	Ex.	Ob.	Ex.	
F.E.	79	63	67	63	43	63	189
S.L.	18	30.67	17	30.67	57	30.67	92
Oth.	3	6.33	16	6.33	0	6.33	19
Tot.	100		100		100		300
Here,	F.E. = Formal Education, S.L. = Self-learn Oth. = Others, Tot. = Total, F.=Failure						

To probe the statistical significance of the observations, we again apply Chi-square test. The contingency table for the test is presented in Table 9. The Chi-square test, with $\chi^2(df = 4, N = 300) = 67.444, p = 0.05$, suggests statistical significance of the observed relationship between developers' educational background and failure rates.

4 Threats to Validity

Although this study is based on a large dataset consisting of 11 million in-IDE event traces, these events/interactions are obtained from 85 developers only. Hence, one may argue against the applicability of the results to all developers in general. However, these 85 developers represent individuals with diverse expertise levels, education backgrounds and geographical regions. Thus, the aforementioned threat is minimized to a great extent.

The captured interactions in the dataset include developers' activities while their work with C# projects only inside the Microsoft Visual Studio IDE. Thus, the findings may not generalize to projects written in languages other than C#, or activities in IDE's other than the Microsoft Visual Studio. However, programming activities are not very different with respect to programming languages or IDE's in use.

The dataset used in this study is publicly available. The methodology of data mining, analyses, and results are well documented in this paper. Therefore, it should be possible to reproduce this work and thus we develop high confidence about the reliability of this study.

5 Related Work

Several earlier work studied software build failures. Typos are one of the the most common reasons for software build fails [12]. Mining IBM Jazz Dataset [8], Connor et. al [1] used multiple code metrics to detect build failure . They determined the software metrics being the best indicators to suggest successful or unsuccessful build, whereas we detected the factors leading to frequent build failure.

Another study [9] used the same dataset we have used and only replicated google's study [11] regarding the rate at which builds in developer workspace fail in the Visual Studio context. They calculated the time to fix a failing build. Again, on the same data set, Rodriguez et. al [10] described the efficiency of developers on weekdays, weekends and selective time of the day and showed that prolonged work time can affect efficiency [4]. Our study also reveals that if the developers do not choose their working-hour wisely, it affects their success rate.

6 Conclusion

In this paper, we have presented an exploratory study to throw spotlight on the determinants affecting the performance of software developers. We have analyzed 85 developers 11 million in-IDE interaction traces over 15,000 work-hours.

We have found that infrequent edits in source code and wise utilization of the regular work-hours stimulate high success rate. Programmers from disparate regions have diverse success rate while the developers from Europe and Africa regions are found to have better success rates. In addition to expertise levels, formal institutional education is found to have played a significant role behind higher success rate of certain developers. A self-taught programmer having good expertise with no formal education is unlikely to outperform a developer lacking experience but having formal educational background. The results are validated in the light of statistical significance.

The finding from this study will help individual developers and organizations in improving their working approaches to leverage success rates. In future, we plan to conduct a larger study including a larger number of developers from diverse background, expertise levels, and geographic regions.

Acknowledgement

This work is supported in part by the SCoRe grant at the University of New Orleans.

References

[1] A. Connor and J. Finlay. Predicting software build failure using source code metrics. *Intl. J. of Info. and Commun. Tech. Research*, 1(5):177–188, 2011.

[2] Coordinated universal time. `https://en.wikipedia.org/wiki/Coordinated_Universal_Time`.

[3] Event types - kave project. `http://www.kave.cc/feedbag/event-generation`.

[4] J. Harrington. Health effects of shift work and extended hours of work. *Occupational and Environmental Medicine*, 58(1):68 – 72, 2001.

[5] Kave project. `http://www.kave.cc`.

[6] A. Mockus and D. Weiss. Predicting risk of software changes. *Bell Labs Tech. Journal*, 5(2):169–180, 2002.

[7] D. Montgomery, C. Jennings, and M. Kulahci. *Introduction to Time Series Analysis and Forecasting*. John Wiley Sons, Inc., 1st edition, 2008.

[8] M. Prout. A guide to jazz source control management. `https://www.ibm.com/developerworks/rational/library/jazz-source-control-management`.

[9] N. Rabbani, M. Harvey, K. Gallaba S. Saquif, and S. McIntosh. Revisiting "programmers' build errors" in the visual studio context: A replication study using ide interaction traces. In *Intl. Conf. on Mining Softw. Repo. (MSR)*, 2018.

[10] A. Rodriguez, F. Tanaka, and Y. Kamei. Empirical study on the relationship between developers working habits and efficiency. In *Intl. Conf. on Mining Softw. Repo. (MSR)*, 2018.

[11] H. Seo, C. Sadowski, S. Elbaum, E. Aftandilian, and R. Bowdidge. Programmers' build errors: a case study (at google). In *36th Intl. Conf. on Softw. Eng. (ICSE)*, pages 724–734, 2014.

[12] M. Shohat. Google study: Typos are the top reason your code won't compile, venturebeat. `https://venturebeat.com/2014/07/04/google-study-typos-are-the-top-reason-your-/code-wont-compile/`.

[13] M. Zibran. Chi-squared test of independence. `https://pdfs.semanticscholar.org/0822/f125a21cfbd05e5e980c8017499fb966568f.pdf`.