Comprehension Effort and Programming Activities: Related? Or Not Related?

Akond Rahman
North Carolina State University, North Carolina, USA
aarahman@ncsu.edu

ABSTRACT

Researchers have observed programmers to allocate considerable amount of effort in program comprehension. But, how does program comprehension effort relate with programming activities? We answer this question by conducting an empirical study using the MSR 2018 Mining Challenge Dataset. We quantify programmers' comprehension effort, and investigate the relationship between program comprehension effort and four programming activities: navigating, editing, building projects, and debugging. We observe when programmers are involved in high comprehension effort they navigate and make edits at a significantly slower rate. However, we do not observe any significant differences in programmers' build and debugging behavior, when programmers are involved in high comprehension effort. Our findings suggest that the relationship between program comprehension effort and programming activities is nuanced, as not all programming activities associate with program comprehension effort.

CCS CONCEPTS

• **Human-centered computing** → *Empirical studies in HCI*;

KEYWORDS

Comprehension, Halstead's complexity, Programmer behavior

ACM Reference Format:
Akond Rahman. 2018. Comprehension Effort and Programming Activities: Related? Or Not Related?. In *MSR '18: MSR '18: 15th International Conference on Mining Software Repositories , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3196398.3196470

1 INTRODUCTION

Researchers [5] [1] have extensively studied on how programmers spend their time and what activities they are involved with. In these studies researchers have observed that programmers spend a considerable amount of time for program comprehension i.e. reading and understanding source code, to accomplish software engineering tasks such as bug fixing, and refactoring. For example, Minelli et al. [5] observed that programmers on average spend 70% of their time on program comprehension.

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

ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196470

But how does the effort needed for program comprehension relate with other programming activities? Amann et al. [1] hinted that the activity of navigation may be related with program comprehension, stating *"the amount of navigation may correlate with the need for code understanding"*. How well-founded is this conjecture? Can we observe empirical evidence that demonstrates the relationship between program comprehension effort and programming activities including navigation? Such investigation can also reveal if other programming activities such as debugging and editing are related with program comprehension.

We conduct an empirical study using the MSR 2018 Mining Challenge Dataset [9] to investigate the relationship between program comprehension effort and programming activities. Similar to prior work [12] [8], we use a surrogate measure to quantify program comprehension effort of programmers by calculating Halstead's metrics [3]. We focus on four programming activities: navigation, building projects, debugging, and editing. We answer the following research question: ***How does program comprehension effort relate with navigation frequency, edit frequency, debugging frequency, and build event frequency?***

Our findings can be summarized as: *when programmers are involved in high comprehension effort they navigate and make edits at a significantly slower rate, but, we do not observe any significant differences in programmers' build and debugging behavior.* Our findings provides empirical evidence that support Amann at al. [1]'s conjecture i.e. program comprehension effort and navigation are related.

2 EMPIRICAL ANALYSIS

We use the MSR 2018 Mining Challenge Dataset provided by Proksch et al. [9], which includes interaction data from programmers and include event types such as 'Command Event', 'Debugger Event', and 'Navigation Event'. We collect the dataset on November, 2017.

2.1 Filtering

Programmer interaction traces achieved from integrated development environments (IDEs), are susceptible to noise, and before conducting any analysis we filter the available dataset. First we obtain the sessions, by using the session IDs and splitting the event stream by session IDs. Next, we use the following two measures to filter out sessions that we won't use for analysis:

- **Session Duration**: We filter out sessions that are less than 600 seconds in duration. From our initial exploration we observe the 25^{th} and 50^{th} percentile of all session duration to be respectively, 123.5 seconds, and 1439.0 seconds.
- **Availability of Method Information**: As we use methods to measure program comprehension effort, we discard any sessions, for which no information about the used methods are available.

2.2 Program Comprehension Effort

We measure programmer's comprehension effort for each method they are working on in each session. We select method, because from an analysis granularity perspective, methods provide the middle ground between details of statements and abstractions provided by classes [6]. For program comprehension effort we use three surrogate measures: 'difficulty', 'volume', and 'parameter count' of a single method. The metrics volume and difficulty, are proposed by Halstead [3], and are widely used in research studies related to program readability and understandability [12].

We compute the comprehension effort for each programming session in the following manner: for each programming session, we extract all the method bodies available using the 'Edit Event' available as part of the MSR 2018 Mining Challenge Dataset. Next, for each method body available in a session, we

- extract the number of parameters, total number of operators, total number operands, total number of unique operators, and total number of unique operands
- calculate metric 'volume', using Equation 1
- calculate metric 'difficulty', using Equation 2
- calculate the number of parameters passed to the method

$$\text{volume} = (\text{total operators} + \text{total operands}) \times \log_2 \times (\text{total distinct operators} + \text{total distinct operands}) \quad (1)$$

$$\text{difficulty} = \frac{\text{total distinct operators}}{2} \times \frac{\text{total operands}}{\text{total distinct operands}} \quad (2)$$

In a session, a programmer can work with multiple methods, and hence to determine the overall comprehension effort associated with the session, we need an aggregate measure. We use median as the aggregate measure, and this measure is less biased by outliers. Next to account for the number of methods that may vary from one method to another, we normalize the aggregated measure for volume, difficulty, and parameter count by the total number of unique methods. Our approach can be expressed using Equations 3, 4, and 5, respectively, for volume, difficulty, and parameter count.

$$\text{Normalized volume for session } x, \text{NV}(x) = \frac{\text{median of all volume measurements for all methods, in } x}{\text{total methods in } x} \quad (3)$$

$$\text{Normalized difficulty for session } x, \text{ND}(x) = \frac{\text{median of all difficulty measurements for all methods, in } x}{\text{total methods in } x} \quad (4)$$

$$\text{Normalized parameter for session } x, \text{NP}(x) = \frac{\text{median of all parameter counts for all methods, in } x}{\text{total methods in } x} \quad (5)$$

We use all three above-described normalized measures NV, ND, and NP, to determine comprehension effort for each session. We assume that by using all three measures collectively, we may obtain a suitable approximation of the associated comprehension effort. We report the summary statistics related to the three measures as a tuple, where the first and second item respectively presents the mean and the median values for each metric. The summary statistics of NV, ND, and NP are respectively, (0.85, 0.00), (0.06, 0.00), and (0.45, 0.25).

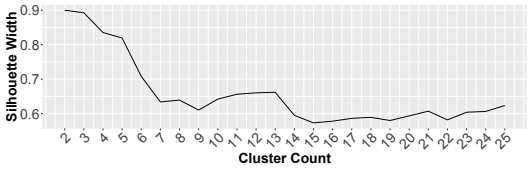

Figure 1: Silhouette Width is highest when cluster count is two, and decreases with the increase in cluster count.

2.3 Dividing Sessions Into Groups

To quantify program comprehension effort used in a session, we assign each session to a group based on the normalized values of volume (NV), difficulty (ND), and parameter count (NP). We perform this assignment using k-means clustering [13], an unsupervised learning technique. We specify two inputs for the k-means clustering technique: the count of clusters needed to create, and the data that will be used to create the groups based on k-means clustering. As input data we provide the three normalized measures of comprehension effort: NV, ND, and NP, as one single feature vector. The count of clusters is determined by Silhouette Width [11], a cluster validation measure, that computes how similar a data point is to it's own cluster, compared to other neighboring clusters [11]. Silhouette Width can have a value between -1 and +1, and a high value indicates a data point is well-matched to its own cluster [11]. To determine the cluster count, we individually computed Silhouette Width for 2, 3, ..., 25 clusters. The cluster count for which we obtained the highest Silhouette Width is used as input to the K-Means algorithm.

We implement k-means clustering and Silhouette Width using Scikit-Learn API [7]. The count of clusters will determine how many groups we have to divide the sessions into. The assignment of groups will be completed using k-means clustering. After completing these steps each session in our dataset will be divided into groups, and assigned appropriate labels. We summarize our methodology in Figure 2.

We report the Silhouette Width for each cluster count in Figure 1. As we observe from Figure 1, Silhouette Width is highest when cluster count is two, and according to our methodology, we use two clusters to classify the sessions into two groups: sessions that are associated with 'high comprehension effort (High)', and sessions that are associated with 'low comprehension effort (Low)'. We report summary statistics for the two groups 'High' and 'Low' in Table 1. We observe from Table 1, the two groups are unbalanced, with respect to count of sessions, navigation events, debugging events, build events, and edit events. Our findings suggest that programmers are not frequently involved with methods that require high comprehension effort.

2.4 Answer to the Research Question

We consider four activities that are common amongst programmers who use Visual Studio: building projects, debugging, editing, and navigation. In the following subsections we report how program comprehension effort is related with each of the four programming activities. To compare the programming activity-related metrics between the two groups, we use the Mann-Whitney U test [4], and

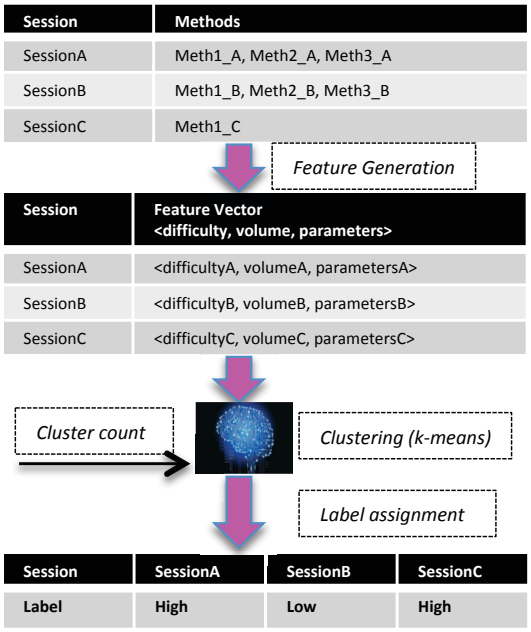

Figure 2: A hypothetical example to illustrate our process of assigning session labels. First, we extract three comprehension features (volume, difficulty, and parameter count) for each method. Next, we use k-means clustering to divide sessions and assign labels.

Table 1: Summary of 'High' and 'Low' Sessions

Measurement	High	Low
Session count	33	704
Build event count	239	42,880
Debugging event count	636	18,458
Edit event count	4,808	228,315
Navigation event count	2,888	184,720

also measure the effect size using Cliffs Delta [2]. We select these methods as they don't make any assumptions on (i) the distribution of the datasets, and (ii) the sample sizes of the groups. Along with reporting the results of the two statistical tests we also report the median values for both groups 'High', and 'Low'. We also apply normalization to account for sample size differences for the two groups 'High' and 'Low'.

2.4.1 Program comprehension effort and navigation frequency. We quantify the relationship between program comprehension effort and navigation frequency using the metric 'normalized navigation interval (*NNI*)', calculated using Equation 6.

$$NNI \text{ for session } x, \text{NNI}(x) = \frac{\text{median duration between two consecutive navigation events in session } x}{\text{total number of navigation events in session } x} \quad (6)$$

We hypothesize that comprehension effort will significantly relate with programmers' navigation behavior. We hypothesize programmers to spend more time on comprehension which can slow down their navigation behavior. According to our hypothesis, sessions in 'High' group i.e., sessions which are associated with high comprehension effort, will have significantly higher *NNI* values.

Table 2: Answer to RQ. Cells highlighted in light grey indicate significant differences.

Metric	Median (High, Low)	p-value	Cliffs Delta
NNI (Navigation Interval)	(1.11, 0.04)	< 0.001	0.45
NEI (Edit Interval)	(0.10, 0.03)	< 0.001	0.34
NDI (Debugging Interval)	(0.09, 0.13)	0.33	0.07
NBI (Build Interval)	(0.00, 0.00)	0.70	0.06

Results. : We report our findings in Table 2. From Table 2, we observe that our hypothesis holds i.e., programmers make navigations at a slower rate, for 'High' sessions. The difference in median values is significant, and the effect size is 'medium' [10]. The distribution of *NNI* values for both groups are available in Figure 3.

2.4.2 Program comprehension effort and edit frequency. We quantify the relationship between program comprehension effort and edit frequency using a metric called 'normalized edit interval (*NEI*)', which is computed in seconds. We compute *NEI* for each session in two groups 'High' and 'Low', using Equation 7.

$$NEI \text{ for session } x, \text{NEI}(x) = \frac{\text{median duration between two consecutive edit events in session } x}{\text{total number of edit events in session } x} \quad (7)$$

We hypothesize that comprehension effort will be significantly related to programmers' edit frequency. For 'High' sessions, i.e. sessions which involve high comprehension effort, we expect to observe edit events at a slower rate, which implies the time interval between edit events will be significantly higher for 'High' sessions.

Results. : We report our findings in Table 2. As we observe from Table 2, our hypothesis holds. We observe programmers to perform edit events at a significantly slower rate, i.e., time interval between edit events (*NEI*) is significantly larger for 'High' sessions. The effect size is 'medium', according to Romano et al. [10]. The distribution of *NEI* values for both groups are available in Figure 3.

2.4.3 Program comprehension effort and debug event frequency. We use the metric 'normalized debugging interval (*NDI*)', which is computed in seconds to quantify the relationship. We compute *NDI* for each session in the two groups 'High', and 'Low', using Equation 8.

$$NDI \text{ for session } x, \text{NDI}(x) = \frac{\text{median duration between two consecutive debug events in session } x}{\text{total number of debug events in session } x} \quad (8)$$

So far respectively, from Sections 2.4.1 and 2.4.2 we observe comprehension effort to be significantly related with navigation and edit frequency. Similar to navigation and edit events, we hypothesize that comprehension effort is associated with debugging behavior. In particular, we hypothesize that programmers make more frequent debugging events when involved with high comprehension, and as a result, debug events will happen at a faster rate, leading to significantly smaller *NDI* values for 'High' sessions.

Results. : We report our findings in Table 2. We do not observe significant empirical evidence for our hypothesis: $p - value = 0.33$, and Cliffs Delta is 0.07 ('negligible' according to Romano et al. [10].). The median *NDI* is smaller for the 'High', but as the Mann-Whitney U test results state, this difference is not significant. The distribution of *NDI* values for both groups are available in Figure 3.

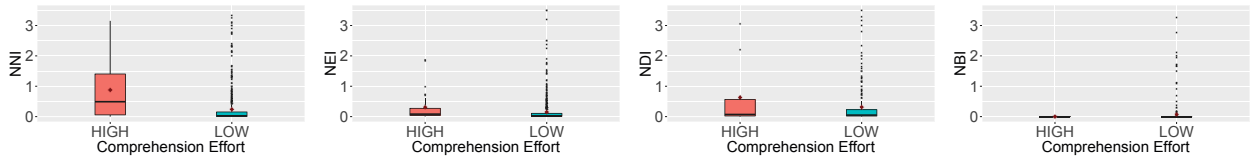

Figure 3: Navigation interval (*NNI*), edit interval (*NEI*), debugging interval (*NDI*), and build interval (*NBI*) values for sessions labeled as 'High' and 'Low'. In 'High' sessions, programmers navigate and make edits at a significantly slower rate.

2.4.4 Program comprehension effort and build event frequency. We use the metric 'normalized build interval (*NBI*)', to quantify this relationship. We compute *NBI* for each session in the two groups 'High', and 'Low', using Equation 9. We apply normalization to account for the differences in build event count. We hypothesize comprehension effort to be significantly related with build frequency. When involved with high comprehension, i.e. in 'High' sessions programmers will make build events at a slower rate. For 'High' sessions we hypothesize significantly larger *NBI* values.

$$NBI \text{ for session } x, \text{NBI}(x) = \frac{\text{median duration between two consecutive build events in session } x}{\text{total number of build events in session } x} \quad (9)$$

Results. : From Table 2, we do not observe significant differences between 'High' and 'Low' groups for *NBI* values ($p - value = 0.70$), and Cliffs Delta is 0.06 ('negligible' according to Romano et al. [10].). The distribution of *NBI* values for both groups are available in Figure 3. The distribution of *NBI* values for both groups are available in Figure 3.

2.5 Threats to Validity

Our technique of quantifying program comprehension effort relies on Halstead metrics which can be limiting. Scalabrino et al. [12] reported that Halstead's metrics might not be sufficient enough to estimate program comprehension effort. We also computed program comprehension effort at the method level, and the comprehension effort required for the other code elements have not been included. Furthermore, we have considered four programming activities, and have not considered other programming activities such as code search, unit testing, and version control activities.

2.6 Related Work

Our paper is closely related to prior research that have investigated programmer behaviors in IDEs. Amann et al. [1] mined programming interactions of 6,300 hours of work time, and observed programmers to spend a considerable amount of time working outside the IDE. Parnin and Gorg [6] proposed a novel technique to create usage contexts in order to facilitate better exploration of programmers' behaviors. Posnett et al. [8] and Scalabrino et al. [12] explored techniques to quantify programmer comprehension efforts. Minelli et al. [5] observed programmers to spend 70% of their time for program comprehension.

3 CONCLUSION

In our paper, by using Halstead's metrics, we have investigated if comprehension effort is related with other programming activities namely, editing, navigation, and debugging. We have observed that when programmers work with methods that demand high comprehension effort, they navigate and make edits at a significantly slower rate. We also do not observe any significant differences in debugging and build behavior, when programmers are associated with methods that demand high comprehension effort. We conclude that the relationship between program comprehension effort and the four studied programming activities is nuanced.

REFERENCES

[1] S. Amann, S. Proksch, S. Nadi, and M. Mezini. 2016. A Study of Visual Studio Usage in Practice. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 124–134. https://doi.org/10.1109/SANER.2016.39

[2] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin* 114, 3 (Nov. 1993), 494–509.

[3] Maurice H. Halstead. 1977. *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., New York, NY, USA.

[4] 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

[5] R. Minelli, A. Mocci, and M. Lanza. 2015. I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time. In *2015 IEEE 23rd International Conference on Program Comprehension*. 25–35. https://doi.org/10.1109/ICPC.2015.12

[6] Chris Parnin and Carsten Gorg. 2006. Building Usage Contexts During Program Comprehension. In *Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC '06)*. IEEE Computer Society, Washington, DC, USA, 13–22. https://doi.org/10.1109/ICPC.2006.14

[7] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (Nov. 2011), 2825–2830. http://dl.acm.org/citation.cfm?id=1953048.2078195

[8] Daryl Posnett, Abram Hindle, and Premkumar Devanbu. 2011. A Simpler Model of Software Readability. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11)*. ACM, New York, NY, USA, 73–82. https://doi.org/10.1145/1985441.1985454

[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] J. Romano, J.D. Kromrey, J. Coraggio, and J. Skowronek. 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys?. In *annual meeting of the Florida Association of Institutional Research*. 1–3.

[11] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20, Supplement C (1987), 53 – 65. https://doi.org/10.1016/0377-0427(87)90125-7

[12] Simone Scalabrino, Gabriele Bavota, Christopher Vendome, Mario Linares-Vásquez, Denys Poshyvanyk, and Rocco Oliveto. 2017. Automatically Assessing Code Understandability: How Far Are We?. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017)*. IEEE Press, Piscataway, NJ, USA, 417–427. http://dl.acm.org/citation.cfm?id=3155562.3155617

[13] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. 2005. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.