A Prospective Study on IDE Usage*

-

Hasan Khwaja
Khoury College of Computer Sciences, Northeastern University, Boston, MA, USA
khwaja.h@northeastern.edu

Brandon Onyejekwe
Khoury College of Computer Sciences, Northeastern University, Boston, MA, USA
onyejekwe.b@northeastern.edu

Ahnaf Inkaid
Khoury College of Computer Sciences, Northeastern University, Boston, MA, USA
inkaid.a@northeastern.edu

ABSTRACT

While programming, Integrated Development Environments (IDEs) are necessary to facilitate the process of developing code. There are many different IDEs for each coding language, each equipped with their own features and capabilities. This work examines how an IDE affects an individual's programming ability. We gathered a set of college-level Python programmers and experimented with a set of questions designed to test their coding, navigation, and debugging skills in Python while using a certain IDE. In addition, we gave each participant a post-experimental survey to assess their subjective opinions about the ease of use, design, and comfort while using each IDE. By combining the quantitative experimental results with the qualitative survey input, we are able to get a comprehensive overview of the effects of IDEs on coding performance.

INTRODUCTION

The role of IDEs in software development cannot be overstated. IDEs are the cornerstone of a developer's toolkit, providing a comprehensive environment for coding, debugging, and testing software projects. With the plethora of IDEs available, ranging from feature rich offerings like IntelliJ IDEA and Visual Studio to lightweight, streamlined options like VS Code and Sublime Text, developers are faced with a multitude of choices. Each IDE claims to offer unique features and optimizations that purportedly enhance productivity and efficiency.

Despite their central role, the extent to which the selection of an IDE influences software development outcomes remains under-explored. While anecdotal evidence and preliminary studies suggest variations in developer preference and performance across different IDEs, systematic, empirical research in this area is sparse. This gap in knowledge forms the foundation of our study, which aims to empirically investigate whether and how the choice of IDE effects developers' performance on specific coding tasks.

To address this question, our study proposes a structured empirical experiment involving a diverse group of 20 participants. These individuals, hailing from various backgrounds in software engineering, will undertake a series of standardized tasks that reflect common challenges in software development such as debugging, code navigation, and algorithmic problem-solving. The tasks are designed to highlight the functionalities and usability of different IDEs under controlled conditions. Each participant will engage with these tasks using multiple IDEs on a clean installation to ensure that performance metrics are not influenced by prior customization. Participants' performance will be quantitatively measured in terms of task completion time, providing a direct metric of efficiency. In addition, qualitative data will be gathered through a series of survey questions, including both Likert scale and open-ended responses, to capture subjective experiences and preferences. This dual approach will allow us to correlate quantitative data with qualitative insights, thus offering a comprehensive understanding of the impact of IDE choice on developer efficiency. By thoroughly examining these facets, this study aims to contribute valuable insights into the software development process, potentially guiding future tool development and aiding developers in making more informed decisions about their working environments.

RELATED WORK

Significant work has been done in the past to test how IDEs and other programming software has impacted coding performance. Notably, most modern IDEs boast features and tools that can significantly reduce the time and commitment required for coding tasks, such as integrated debuggers, real time syntax checking, tab completion, static analysis, and code inference. Some studies have performed an in-depth analysis of how these types of features can affect code performance [2, 6]. By utilizing these features, IDEs are meant to help the user in many ways, mostly revolving around speeding up the process to create, navigate, and debug code. They can also influence the learning curve for new programmers by providing

immediate feedback and suggestions. Furthermore, the customization and extensibility of IDEs allow developers to tailor the programming environment to their specific needs, potentially increasing productivity and efficiency. Past studies have been conducted to attempt to quantify the effects of IDE features on these various measures of coding speed and performance. For example, one study was conducted to look into the benefits (and drawbacks) of using IDEs when looking at general productivity [10]. Similarly, another study analyzed the extent to which IDEs impact one's ability to debug code using a debugger and any other available features [1]. The difference in time spent debugging has been established to correlate with the debuggers specific to an IDE rather than classic programming [11]. The effects of IDEs have also been explored in numerous different contexts. Studies have explored the use of IDEs on pair vs solo programming [5] and have also been conducted to look at how IDEs support programmers when factoring in the effects of the COVID-19 pandemic [7]. Even when comparing between languages, it has been shown that languages such as Python can be more intuitive to use than their counterparts [8, 9]. Python itself has proven to be preferred when learning to code, making advantageous IDEs sought after since it provides a complete ecosystem for student learning. [8, 4]. In addition to this, the tools available in each programming software was shown to have an effect on the efficiency of coding given how some have a variety of built in tools to perform tasks when others do not [3]. This may have an implication on the usage of IDEs as well due to how each IDE will offer its users its own set of tools given some overlap in the language of choice. Debugging tools are especially important in programming since programmers have to frequently debug their code when they get stuck in a certain function or a function does not work as expected. The effect IDEs have on coding performance should be further explored as the set of tools they offer may indeed play a significant role in enhancing or detracting from the overall coding performance of developers.

METHODS

To test the differences between IDE usage, an experimental procedure was created in which participants would complete a series of tasks in which after completing one set of tasks they would switch to a different IDE and complete a similar set of tasks. The tasks were as follows:

1 Coding

We wanted to assess each individual's ability to write and develop code in each IDE. The questions we asked were:

- **C1**: Create a list of integers from 1 to 100 (inclusive) backwards, skipping every 3rd integer
- **C2**: For a list of numbers from 2 - 10, if the number is divisible by 2 print "Fizz", if the number is divisible by 3, print "Buzz", if the number is divisible by both 2 and 3, print "FizzBuzz"
- **C3**: Create a program that takes an integer as an input and returns that integer +1 if it is odd and +2 if it is even

2 Debugging

We also wanted to examine how each IDE facilitates the process of debugging and fixing code. The questions we asked were:

- **D1**: Give the error message, and line number for the following function
- **D2**: Figure out why the expected output is not being returned and fix the issue
- **D3**: Provide a test case for the expected output, "Jello World"

3 Navigation

Finally, we wanted to see how the IDE affects a participant's ability to search, find, and navigate sections of code. We asked the following navigation questions:

- **N1**: Discover how to quickly jump to where a function is defined or declared in the code
- **N2**: Find out how to outline the entirety of a function's body from its beginning to its nd
- **N3**: Figure out a method to scour through the entire codebase for occurrences of the specific variable or identifier "new_word". How many occurrences are there?

To ensure that the testing remained consistent in the debugging and navigation sections of testing, the participants were given a pre-developed function to work with.

```python
"""
A Prospective Study on the Usage of Various IDEs
Test Function
Debugging
Navigation
"""

def word_play(list_of_words: list, letter_1: str, letter_2: str):

    words = " ".join(list_of_words)

    for i in range(len(words + 1)):
        if letter_1 == words[i]:
            new_word = words.replace(letter_1, letter_1)

    return new_word

def main():

    word = word_play(["hello", "world"], "h", "j")

    print(word)

    print("Expected Output: Jello World")

if __name__ == "__main__":
    main()
```

Figure 1: Screenshot of a sample code function we presented to participants during each experiment

There are two main corrections that should be made for this code. First, on line 12, the line should be: for i in range(len(words)):. Second, on line 14, the code should be: new_word = words.replace(letter_1, letter_2.

When creating these tasks, we adhered to a design philosophy in which the tasks were kept from being both too difficult and too easy. To ensure that a middle ground was found, the tasks were tested by an individual who did not create the tasks themselves. The goal was to have the entire testing process take ~15 minutes. This target was used to ensure that the IDEs were being tested and not the individual's programming skills. This also helped to ensure that the questions would not take any one individual enough time that they become so frustrated they quit the study.

The procedure for the order in which the participants would do these questions was also important to consider. This would help to ensure that any difference observed in efficiency would be due to the differences in IDEs and not the tasks given. First, every participant would start with the same task but use a different IDE to start. Upon starting and ending each task, a study team member will record the time it took for each task to be completed. After one set of tasks was completed, each participant would move onto a different IDE in a set order before beginning the second set of tasks. Each set of tasks consisted of one task programming, debugging and navigation. Please refer to Figure 2 for a visual representing a simulation of testing 3 participants.

Upon completion of all the tasks, the participants were then directed to fill out a survey detailing the results of their IDE usage experience. The survey was built to focus on the overall experience of using each IDE when compared to one another, as well as collect demographic data for a better understanding of where each participant is coming from.

RESULTS

For our first test we used repeated measures ANOVA to test if questions within the same question type have similar responses. For example, we wanted to make sure that coding question #2 was not significantly easier or harder to complete than coding questions #1 and #3. We found that the p-values were not significant at a significance level of 0.05 for any of the three question types (Coding: p=0.5395, Navigation: p=0.3020, Debugging: p=0.0961). Thus, we conclude that, for each question type, the questions are all of relatively similar difficulty levels. This validates our approach to randomly assign questions within each question type for each participant, as the specific question they are assigned should not make a significant difference.

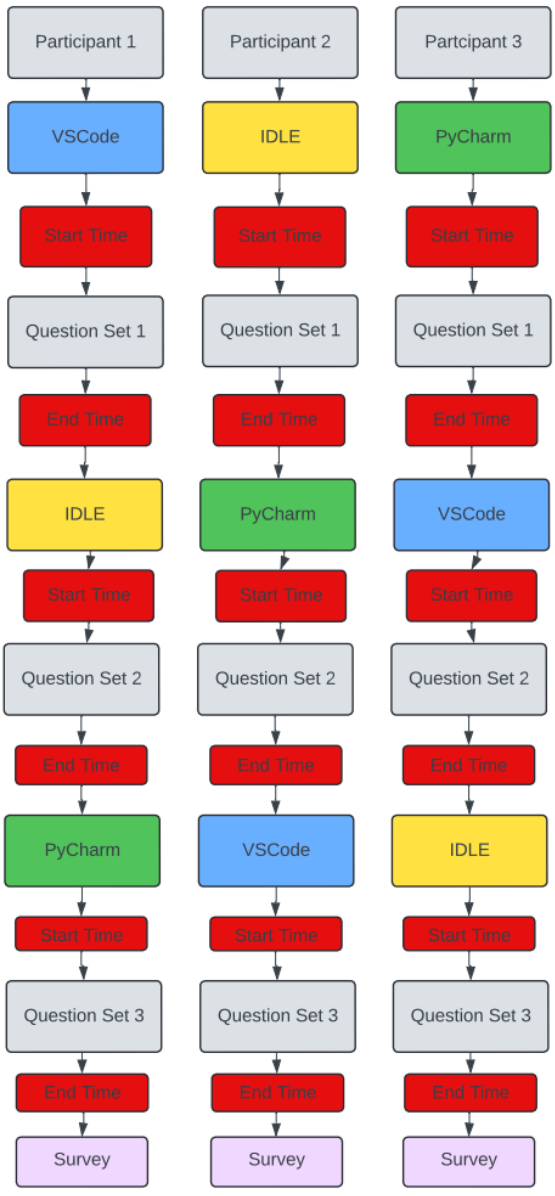

Figure 2: Workflow diagram of our experimentation process.

For our second test, we used repeated measures ANOVA to test if IDEs yield similar response times. This is the main question we want to answer: whether time to complete tasks for each question type (coding, debugging, and navigation) differs when using one IDE over another. Once again using 0.05 as the alpha value, ur resulting p-values show that there was not a significant difference between IDEs for any of the question types (Coding: p=0.7515, Navigation: p=0.9302, Debugging: p=0.2761). Our conclusion is that IDE response times are not significantly different relative to question type.

To analyze the results of the survey, Friedman's test was used due to how, here, we are both looking to compare the means between multiple groups that are not independent from one another. When looking at the results of the survey, a trend could be seen in terms of each user's rankings for different IDE usage attributes. We saw significant differences in user rankings for positive attributes such as **usefulness** and **intuitiveness**, with p-values equal to 0.0112 and 0.0011 respectively. The only ranking that did not have a significant difference in user rankings was **comfort** with a p-value of 0.2164. When looking at the negative attributes such as **frustration**, **inconvenience** and **time consumption**, all user rankings were shown to have a significant difference between IDE types with p-values of 0.0058, 0.0125 and 0.0079 respectively.

For a post-hoc analysis, we conducted the multiple Wilcoxon signed rank tests to examine the differences between IDE pairs for each positive and negative attribute that had a significant p-value. We found that there were significant differences between IDLE and VSCode for the usefulness and intuition rankings with p-values of 0.0064 and 0.0015 respectively. For frustration, we saw a significant p-value for VSCode vs IDLE where the p-value = 0.0046. In terms of the overall user rankings of IDEs, we found significant differences for programming in VSCode vs IDLE, VSCode vs PyCharm, and PyCharm vs IDLE where the p-value = 0.0006, 0.012, and 0.036 respectively. When looking at debugging, significance was found for VSCode vs IDLE and PyCharm vs IDLE where the p-value = 0.0015 and 0.0056. In terms of navigation, significance was found for PyCharm vs IDLE, PyCharm vs VSCode and VSCode vs IDLE where the p-value = 0.0027, 0.0002, and 0.009 respectively.

When asked to rank IDEs in terms of overall usage from best to worst, we saw a trend in which the majority of users ranked VSCode as the best, PyCharm as the second best and IDLE as the worst. This claim is further supported by the Friedman's tests that were conducted on each tested area of IDE usage, programming, debugging, navigation, and design. The p-values for each of these areas all showed significant differences with the p-value being less than 0.001 for all of them. Figure 3 below shows how each participant's preference differs between IDEs.

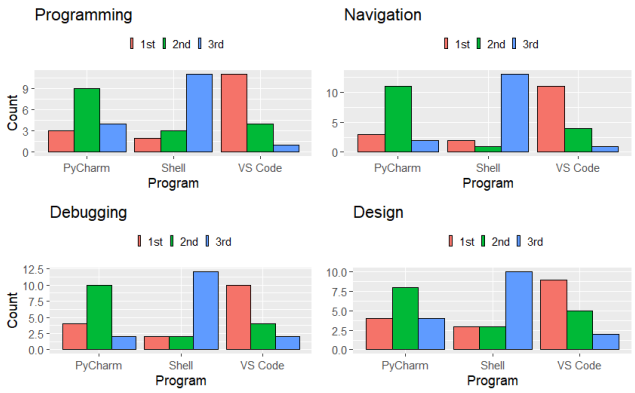

Figure 3: Histogram of survey results, showing IDE 1st place (red), second place (green), and third place (blue) preferences for each IDE.

DISCUSSION

The results from our experiment lead to different conclusions than the results of our survey. We found that, in practice, there is no significant difference in performance between IDEs on the three types of questions we examined. However, the survey results lead us to conclude that people think the IDE impacts their performances in different ways. This disconnect reflects the possible theory that a person's perceived comfort with a given tool does not necessarily directly relate to their performance with that tool.

Generally, people preferred PyCharm and VSCode to IDLE, which matched up with our expectations prior to conducting the experiment. PyCharm and VSCode are very common IDE choices for development in classes and in industry, while IDLE is almost exclusively used when one first learns how to program in Python. The fact that we were unable to concretely establish that participants perform better using PyCharm and VSCode implies that it is possible that the actual tangible differences between them is not as large as we thought beforehand.

LIMITATIONS

While our experimental design is to determine the effects of IDE choice on coding performance, there are external variables separate from IDE choice that can impact coding

performance. Our participants were a convenience sample of 17 students of different years and majors (although a majority of them were from Khoury College). This implies that the participants are at differing levels of general coding experience, which could confound their experience trying to navigate and understand different functionality in a coding environment. Similarly, they have different levels of experience using the Python programming language, which could affect the times it takes to complete Python coding questions independent of the IDE used.

We took steps to mitigate the differences in difficulty between specific questions by establishing beforehand that the response times for different questions in the same question type did not differ significantly. However, a given individual could find some of the questions significantly harder than others due to the difference in experience. A notable example of this was the question “Create a list of integers from 1 to 100 (inclusive) backwards, skipping every 3rd integer”. Participants tackled these questions in various ways, revealing their relative Python experience levels. There were a few participants who were veteran Python users that immediately found a simple, one-line solution involving intermediate-level indexing. Others, who were not as experienced in Python, took significantly longer to write out entire blocks of code in a loop. These differences are almost completely independent of IDE choice.

Due to the long process to perform this experiment for each participant, our sample size was 17. This meant that we were not able to immediately apply the Central Limit Theorem in our analysis, and needed to check that the data was normally distributed before proceeding with our tests. We could mitigate variation by collecting more samples, allowing us to be more confident in our results.

Since our long experimental process included asking individuals 9 questions, we often found that participants felt fatigued while completing the last few questions. While we mitigated the effects of this on our experiment by randomizing ordering the IDEs, we would ideally design the experiment to be completed in a shorter time period. Finally, our survey was given to each participant after the experimentation, while they were already fatigued from completing the 9 questions. We received feedback that the survey felt long to complete, so we should omit questions we did not use for analysis to shorten the survey while also keeping the information we need to analyze.

FUTURE WORK

While we decided to look at 3 specific commonly-used IDEs (VSCode, PyCharm, and IDLE), we could extend this research by looking at other commonly used Python IDEs for a greater assessment of IDEs in general. Many other IDEs (Spyder, Atom, etc.) are also very popular and used by many developers, so it is important to consider that they could have different effects on participants than the three IDEs we chose to examine in this study.

Other languages could also be examined. While we chose to focus on Python due to relative ease and diversity in set of IDE choices, other common programming languages (such as Java, C, etc.) could reveal if different languages yield different results. It is possible that programmers in other languages could be more dependent on the specific features of their IDE of choice, and would find more variance between performance while using other IDEs.

Future studies could incorporate a broader range of programming tasks, varying in complexity and closer mimicking real-world software development scenarios. This could include integrating tasks that require interaction with databases, web development, or the use of APIs, which might highlight different strengths and weaknesses of each IDE more effectively. Additionally, increasing the number of participants and including developers from different professional backgrounds and expertise levels could provide more generalizable results. A more diverse participant pool may also help identify whether certain IDEs are more suited to specific programming languages or dev environments. Exploring how developers customize their IDEs and the use of extensions could significantly affect performance and user satisfaction. Future research could involve participants setting up their IDEs as they would in their usual dev environment, then performing tasks to see how customized setups compare with default installations.

Investigating the cognitive and psychological impact of using different IDEs could also be worthwhile. For example, studying cognitive load, frustration levels, and overall user experience could provide deeper insights into why developers prefer certain IDEs over others despite similar performance outcomes. Future work should aim to assess new versions of IDEs and newly introduced IDEs as they become available. With the rise of mobile and cross platform development, studying the effectiveness of IDEs in these contexts would be valuable. This includes examining IDEs tailored for mobile development like Android Studio and Xcode, as well as those that support cross platform development like Flutter or React Native.

CONCLUSION

This study sought to empirically evaluate the impact of Integrated Development Environments (IDEs) on developer performance across a series of standardized programming tasks. Our findings indicate that there were no significant differences in task completion times across different IDEs

when measuring coding, debugging, and navigation tasks. This suggests that, at least for the tasks and environments tested, the choice of IDE does not dramatically influence the efficiency of task performance in a controlled setting. However, our survey results revealed a different story, showing significant differences in developers' subjective assessments of the IDEs, particularly regarding ease of use, intuitiveness, and overall satisfaction. Developers often favored certain IDEs, indicating a perceived enhancement in their coding experience, which did not necessarily correlate with the objective measures of task completion time. This divergence underscores the importance of considering both objective performance metrics and user satisfaction in the evaluation of development tools. The study also highlighted the critical role of personal preference and familiarity in choosing an IDE. Developers' preferences varied widely, influenced by factors such as prior exposure, specific toolsets, and the nature of the tasks they commonly perform. This aligns with the notion that while IDEs might not differ significantly in terms of raw performance, the features and user experience they offer can significantly influence individual productivity and preference. In conclusion, while our study did not find a significant difference in performance across IDEs for the specified tasks, it did emphasize the importance of subjective preferences and the perceived impact of IDEs on developers' efficiency and satisfaction. These insights are invaluable for both developers and organizations as they choose the tools that best fit their needs and for software tool developers aiming to improve and tailor their products to better meet the needs of the software development community. Future research should continue to explore these aspects, ensuring that the evolution of development environments aligns with the diverse needs and preferences of users across the software development spectrum.

STATEMENT OF CONTRIBUTION

Hasan: conducted experiments, surveyed participants, helped write the survey, analysis on the survey responses, created the tasks, created the test function for the debugging and navigation tasks, helped put together the report.

Brandon: conducted experiment, surveyed participants, performed statistical tests on experimental data, formatted survey responses, presented with sprained ankle, typed report.

Ahnaf: conducted experiment and surveyed participants, wrote demographic questions and some of the likert questions, typed report.

REFERENCES

1] Afzal, A., & Goues, C. L. (2018, May). A study on the use of IDE features for debugging. In Proceedings of the 15th International Conference on Mining Software Repositories (pp. 114-117)

2] Beigelbeck, A., Aniche, M., & Cito, J. (2021, May). Interactive Static Software Performance Analysis in the IDE. In 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC) (pp. 490-494). IEEE. - (link)

3] Brittain, J., Cendon, M., Nizzi, J., & Pleis, J. (2018). Data scientist's analysis toolbox: Comparison of Python, R, and SAS Performance. SMU Data Science Review, 1(2), 7.

4] Edwards, S. H., Tilden, D. S., & Allevato, A. (2014, March). Pythy: Improving the introductory python programming experience. In Proceedings of the 45th ACM technical symposium on Computer science education (pp. 641-646).

5] Gómez, O. S., Aguileta, A. A., Aguilar, R. A., Ucan, J. P., Rosero, R. H., & Cortes-Verdin, K. (2017). An empirical study on the impact of an IDE tool support in the pair and solo programming. IEEE Access, 5, 9175-9187.

[6] Grotov, K., Titov, S., Sotnikov, V., Golubev, Y., & Bryksin, T. (2022, May). A large-scale comparison of Python code in Jupyter notebooks and scripts. In Proceedings of the 19th International Conference on Mining Software Repositories (pp. 353-364).

[7] Kusumaningtyas, K., Nugroho, E. D., & Priadana, A. (2020). Online integrated development environment (ide) in supporting computer programming learning process during covid-19 pandemic: A comparative analysis. IJID (International Journal on Informatics for Development), 9(2), 66-71.

[8] Simon, Mason, R., Crick, T., Davenport, J. H., & Murphy, E. (2018, February). Language choice in introductory programming courses at Australasian and UK universities. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education (pp. 852-857).

[9] Wainer, J., & Xavier, E. C. (2018). A controlled experiment on Python vs C for an introductory programming course: Students' outcomes. ACM Transactions on Computing Education (TOCE), 18(3), 1-16.

[10] Zayour, I., & Hajjdiab, H. (2013). How much integrated development environments (IDEs) improve productivity?. J. Softw., 8(10), 2425-2431.

[11] Zayour, I., & Hamdar, A. (2016). A qualitative study on debugging under an enterprise IDE. Information and Software Technology, 70, 130-139.