British Journal of Multidisciplinary and Advanced Studies 5(5),32-39, 2024
Education, Learning, Training & Development
Print ISSN: 2517-276X
Online ISSN: 2517-2778
https://bjmas.org/index.php/bjmas/index
Published by the European Centre for Research Training and Development UK

Exploring Learning Dynamics: A Comparative Study of IDEs and Command-Line Interfaces for Learning Java Programming language in the Department of Computer Science, Jigawa State College of Education and Legal Studies Ringim

Musa Aliyu
Department of Computer Science Education
Jigawa State College of Education and Legal Studies Ringim, Jigawa State Nigeria

doi: https://doi.org/10.37745/bjmas.2022.04187 Published October 12, 2024

Citation: Aliyu M. (2024) Exploring Learning Dynamics: A Comparative Study of IDEs and Command-Line Interfaces for Learning Java Programming language in the Department of Computer Science, Jigawa State College of Education and Legal Studies Ringim, *British Journal of Multidisciplinary and Advanced Studies* 5(5),32-39

Abstract: *This study explored the learning dynamics associated with Integrated Development Environments (IDEs) and Command-Line Interfaces (CLIs) for teaching and learning java programming language in the Department of Computer Science at the College of Education and Legal Studies, Ringim. The study evaluates the effectiveness of different programming tools (IDEs and CLIs) in enhancing the learning experience of Java programming students. It seeks to understand how these tools impact learners' confidence, skill development, and overall programming proficiency. The results revealed distinct advantages and challenges associated with each environment, underscoring the importance of context and learner preferences in selecting the appropriate tool for Java programming education. The study provides valuable recommendations for educators on optimizing programming instruction to better meet the needs of diverse learners.*

Keywords: IDEs, Java, environment, learning, tools selection, proficiency

INTRODUCTION

This research addresses a critical aspect of learning java programming language, providing educators, institutions, and industry stakeholders with valuable insights into the working tools that best contribute to student success. By examining both IDEs and Command Line interfaces, the study aims to enhance the quality of programming education and its alignment with contemporary industry practices.The findings of this research could potentially impact teaching methodologies, providing insights into the effectiveness of different practical working tools and influencing the selection of tools in educational settings.

Learning Java programming necessitates a deep understanding of various concepts, including OOP paradigms, data structures, concurrency, and low-level system interactions. The choice of learning

tools plays a crucial role in shaping students' conceptual comprehension, problem-solving skills, and professional preparedness. IDEs, equipped with graphical interfaces, interactive debuggers, and code completion features, offer user-friendliness and promote rapid development cycles. However, critics argue that these conveniences might mask underlying complexities and hinder the development of fundamental debugging and troubleshooting skills. CLIs, conversely, demand direct interaction with the system, fostering a deeper understanding of underlying mechanisms and fostering self-reliance. However, their steep learning curve and lack of visual aids can discourage novice learners.

LITERATURE LITERATURE/THEORETICAL UNDERPINNING

Existing research comparing the effectiveness of IDEs and CLIs in programming education has been analysed, focusing on studies related to Java specifically. Pedagogical considerations: The study examines how instructors can leverage the strengths of each tool and mitigate their limitations through effective curriculum design and teaching methodologies. The purpose of computer science teachers is to enable students to grasp the basic knowledge needed for further studies of computer science and related technologies and to understand its applications and also to help learners to acquire the skills of practical utilities, develop the skills to apply those skills in life situations.

Constructivists say learning is an active, ongoing process where students build their own understanding by linking new ideas to what they already know. In computer science, this means students solve problems and think critically, making them central to learning (Ben-Ari, 1998).An essential aspect of teaching computer science involves emphasizing the cultivation of computational thinking skills. Although the concept of computational thinking gained prominence only in 2006 through Wing (Wing, 2006), educators in computer science have been fostering these skills throughout the history of the subject. In 2014, guidelines have been established in England to guide teachers on explicitly incorporating computational thinking into the new curriculum (Curzon, Dorling, Ng, Selby, & Woollard, 2014).

Programming is the aspect of computer science in school which is perceived to be the most challenging. A range of activities can be used that allow students to collaborate and construct problem solutions. As an example, the following suggestions, drawing on a constructivist view of learning, are made by Van Gorp and Grissom: Code walkthroughs, writing algorithms in groups, Insert comments in pairs into existing code Develop code from algorithm in pairs, and finding the bugs in code (Van Gorp & Grissom, 2001).

Research by Lopez et al. (2008) and Lister et al. (2004) highlights the importance of reading and tracing code in mastering programming. These skills act as building blocks for the problem-solving required for independent coding. Notably, Lister (2011) suggests that novices need at least 50% accuracy in code tracing before venturing into writing their own programs. Constructivism, based on students' active participation in problem-solving and critical thinking, has profoundly influenced the teaching of programming (Ben-Ari, 1998). It implies a need for authentic and meaningful experiences to support learning based on prior experiences and models of the world.

Java Programming Education

Overview of Java Programming in Academic Curricula: Java is a fundamental language in many computer science programs due to its platform independence, robust libraries, and industry relevance. It is widely used in teaching object-oriented programming, data structures, and software engineering (Murray & Yang, 2022). Java's ability to run on various operating systems without modification makes it a valuable teaching tool in diverse academic settings.

Challenges and Opportunities in Teaching Java: While Java offers significant benefits for teaching programming, educators face challenges such as its complex syntax and the steep learning curve for beginners. However, these challenges are balanced by opportunities to introduce students to essential programming principles, such as strict typing and modular design, which are critical in software development (Jones & Smith, 2021). The key is to develop teaching strategies that mitigate the initial difficulties while leveraging Java's strengths.

Integrated Development Environments (IDEs)

Description and Examples of Popular IDEs for Java: Integrated Development Environments (IDEs) are vital in modern programming education. Tools like IntelliJ IDEA, Eclipse, and NetBeans provide features such as code autocompletion, debugging, and integrated version control, which help streamline the coding process (Brown & Robertson, 2023). These IDEs are widely adopted in both educational and professional settings for their ability to enhance productivity and reduce errors.Previous Studies on the Impact of IDEs on Learning Programming: Research indicates that IDEs can significantly improve learning outcomes by reducing cognitive load and allowing students to focus on higher-level problem-solving (Green & Johnson, 2023). However, there are concerns that reliance on IDE features like autocompletion may hinder the development of a deep understanding of programming concepts. For example, students using IDEs may struggle with basic syntax and problem-solving without these tools, suggesting a need for balanced instruction that includes both IDE and non-IDE experiences (Thompson & Walker, 2022).

Command-Line Interfaces

Command-Line Interfaces (CLIs) offer a more manual approach to programming, requiring students to use tools like javac for compilation and java for execution. These tools provide a deeper understanding of the compilation process and are crucial for learning the fundamental operations of the Java language (Perez & Lee, 2024). CLIs are often seen as a bridge to understanding the underlying mechanics of programming languages.

Comparative Studies

Comparative research on IDEs and CLIs often highlights the trade-offs between ease of use and depth of understanding. IDEs are generally favored for their user-friendly interfaces and features that simplify coding for beginners, while CLIs are praised for fostering a deeper engagement with the programming process (Williams & Harris, 2022).

British Journal of Multidisciplinary and Advanced Studies 5(5),32-39, 2024
Education, Learning, Training & Development
Print ISSN: 2517-276X
Online ISSN: 2517-2778
https://bjmas.org/index.php/bjmas/index
Published by the European Centre for Research Training and Development UK

Tools selection between IDE and Command-line interface-based platforms

IDEs, or Integrated Development Environments, play a crucial role in the development and execution of Java programs. A study by Lorenzo and Peirluigi (2015). Proved that IDEs provide a sophisticated code editor where developers can write, edit, and organize Java code. Features such as syntax highlighting, auto-completion, and code formatting enhance the coding experience when compared with using command-line tools. However, IDEs have built-in compilers that translate the human-readable Java source code into bytecode, which is a lower-level representation that can be executed by the Java Virtual Machine (JVM). In the other hand, IDEs often include tools for managing the build process. They can compile the code, manage dependencies, and create executable files or JAR (Java Archive) files. Debugging tools within IDEs assist developers in identifying and fixing errors in their code. IDEs can seamlessly execute Java programs within the development environment. They interact with the JVM to run the compiled bytecode, allowing developers to test their applications easily.

A study by Chen & Marx (2005) suggests that while Eclipse offers powerful tools, its initial learning curve might be detrimental for complete beginners, hindering their progress in both programming and IDE familiarity. The study also discusses other IDEs that are designed for educational purposes. These IDEs will have less features and will be less overwhelming to students but fail to expose the students to real life programming environments. Another study performed by Milne and Row (2002). found some java concept difficult to grasp such as polymorphism due to lack of understanding what is actually going on in the background, they are in the opinion that java should be taught without IDEs. The study is expected bridged the gaps on literature, theory and methodology.

METHODOLOGY

The study employed survey research to obtain the opinion of the students of java programming from the department of computer science College of Education and legal studies, Ringim. Close ended questionnaires were administered to 209 students to gain deeper insights into their experiences, preferences, and perceived learning outcomes. The survey conducted in the study took several measures to ensure that participants understood the differences between Integrated Development Environments (IDEs) and Command-Line Interfaces (CLIs) by providing clear definitions, contextual examples, structured comparative questions, assessing familiarity, and incorporating a feedback mechanism. These strategies collectively contributed to a more informed participant base, leading to more accurate and meaningful responses in the study. The survey included questions about their level of experience, formal instruction received, frequency of tool usage, and perceived impact on their programming skills.

The survey included questions that directly asked participants to compare the ease of use of IDEs and CLIs. Respondents were given options to express their opinions, such as "IDEs are much easier to use," "IDEs are slightly easier to use," "No difference," "Command-Line is slightly easier to use," and "Command-Line is much easier to use." This structured approach allowed for quantifiable data on user preferences and experiences with both tools

British Journal of Multidisciplinary and Advanced Studies 5(5),32-39, 2024
Education, Learning, Training & Development
Print ISSN: 2517-276X
Online ISSN: 2517-2778
https://bjmas.org/index.php/bjmas/index
Published by the European Centre for Research Training and Development UK

RESULT /FINDINGS

The study evaluated the effectiveness of IDEs versus CLIs based on user experience, impact on learning efficiency, confidence levels, challenges encountered, overall learning satisfaction, and the importance of tool familiarity. These criteria collectively provide a comprehensive understanding of how these tools influence the learning outcomes of Java programming students. The overwhelming majority of respondents identified as beginners (83.25%), indicating a need for educational resources tailored to novice programmers. This suggests that any instructional material should focus on foundational concepts and user-friendly tools. A significant portion of respondents received instruction in both IDEs and CLIs, highlighting the importance of a balanced approach in teaching programming tools. The absence of respondents with no formal instruction suggests that most learners are exposed to some form of guidance.

A majority of respondents (74.64 %) perceive IDEs as easier to use, which may reinforce their preference for these tools with 58.85% that frequently used IDEs as a tool of their choice. However, a notable minority (22.96%) recognizes the advantages of CLIs, indicating that there is room for promoting CLI usage among beginners. The high percentage of users (64.11%) encountering challenges with CLIs suggests that additional support and resources are needed to help users overcome these obstacles. In the other hand the participants understanding of java concept with IDEs gained much weight (74.64%) as compared to CLIs with (17.23%) and 8.3% chooses both tools. More over when it comes for ease of debugging (77.99%) believed that IDEs influenced their ability to debug codes and (19.62) are in the opinion that CLIs were the best.

The results indicate a strong preference for IDEs among learners, with a significant majority reporting that IDEs improve their confidence and ability to write efficient code. The data shows that IDEs are favoured for their user-friendly interfaces, real-time feedback, and integrated debugging tools, which collectively enhance the learning experiences. while also promoting a more balanced exposure to both IDEs and CLIs. By understanding user preferences and obstacles, this research seeks to enhance the overall programming learning experience, ensuring that novice programmers are well-equipped to navigate their chosen tools effectively. Furthermore, it is essential to explore the potential benefits of integrating both environments into the curriculum, allowing learners to appreciate the strengths and weaknesses of each tool.

DISCUSSIONS

Based on the findings of this research we can analysed the effectiveness of using Integrated Development Environments (IDEs) and Command-Line Interfaces (CLIs) in teaching Java programming in our Colleges of Education with significant number of leaners with diverse learning style and preferences, mostly novice programmers who undergoes java programming classes to receive formal educations. A significant portion of respondents (42.58%) reported feeling "very confident" when using an IDE, while only 8.61% felt "very confident" with the Command-Line interface. This suggests that IDEs may foster greater confidence among learners which will in turn lead to better coding capabilities. Previous Studies on the Impact of Command-Line Interfaces on programming

skills, have shown that students who learn through CLIs develop a stronger grasp of programming concepts and are better prepared for real-world programming tasks (Taylor & Martin, 2022). While CLIs present a steep learning curve, they are associated with higher programming competence and a more thorough understanding of how code is executed and managed (Nguyen & Hsu, 2023). When participants were asked which tool better supports the development of their coding skills, 57.89% preferred IDEs, while 32.06% favoured Command-Line interfaces. This indicates a strong preference for IDEs in skill development

Learning Support: A substantial majority (77.99%) of participants felt that IDEs better support their learning and retention of Java programming concepts compared to Command-Line interfaces (18.18%). This highlights the effectiveness of IDEs in facilitating understanding.The ability of learners to write error free code had been accessed with (69.86%) who believed that IDEs supports them to achieve such a mile stone, while (29.19%) are in the opinion that CLIs are the best, going by this the percentage of those who chooses IDEs are inline with the existing research as ascertain by Lorenzo and Peirluigi (2015).

Task Suitability: Respondents indicated that IDEs are preferred for most programming tasks, while Command-Line interfaces are seen as more suitable for specific tasks like compiling and running scripts. This suggests that while both tools have their place, IDEs are generally favoured for broader programming activities. The study assessed how the ease of use of IDEs and CLIs influenced the overall learning experience in Java programming. Participants were asked to describe their learning experiences, which included reflections on how user-friendliness of the tools contributed to their confidence and efficiency in coding. the combination of a user-friendly interface, real-time feedback, integrated debugging tools, intelligent code assistance, access to documentation, and community support are key features of IDEs that contribute to increased confidence among Java programming learners. These features create a conducive learning environment that encourages exploration and mastery of programming skills.

Implication to Research and Practice

The findings suggest that educators should consider prioritizing the use of IDEs in teaching Java programming to foster a more supportive and effective learning environment. The study highlights the importance of tool selection in programming education and its impact on student outcomes.

CONCLUSION

The study suggests that Integrated Development Environments (IDEs) are generally more effective than Command-Line Interfaces (CLIs) for teaching Java programming. The findings indicate that:
-IDEs boost confidence and skill development among learners.
-They are preferred for learning and retaining programming concepts.
-IDEs are more suitable for a wider range of programming tasks, while CLIs serve specific functions.
Based on the above the study suggested that educators may consider prioritizing IDEs in their teaching strategies to enhance the learning experience and outcomes for students in Java programming in our colleges of Education in Nigeria. However, the findings suggest that educational programs should

focus on enhancing the learning experience with both tools, addressing misconceptions, and providing targeted support for users struggling with CLIs. Overall, the data highlights the importance of balanced tool exposure and the need for resources that cater to the unique challenges faced by novice programmers

Future Research

Future research could focus on comparative effectiveness studies, user experience research, addressing misconceptions, cross-language comparisons, hybrid approaches, demographic studies, and the impact of emerging technologies. These avenues could significantly contribute to the understanding and optimization of IDEs and CLIs in programming education and practice. The futures researches may focus on the following:

Longitudinal Studies: Conduct longitudinal studies to assess how the use of IDEs versus CLIs impacts the learning outcomes of novice programmers over time..

User Experience Research

-*User Satisfaction Surveys*: Implement comprehensive user satisfaction surveys that delve into the specific features of IDEs and CLIs that users find most beneficial or frustrating.

-*Targeted Educational Interventions*: Research could focus on developing targeted educational interventions aimed at addressing the misconceptions identified in the current study.

-*Incorporating New Technologies*: Investigate how emerging technologies, such as cloud-based development environments and AI-assisted coding tools, influence the use and effectiveness of traditional IDEs and CLIs. This research could provide insights into the future landscape of programming tools.

REFERENCES

Ben-Ari, M., **(1998)**. Constructivism in computer science education. *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*. Atlanta, Georgia, United States: ACM.

Brown, A. & Robertson, T., **(2023)**. Enhancing programming education with modern IDEs: A comprehensive review. *Journal of Computer Science Education*, 35(1), pp.45-62. https://doi.org/10.1080/10494820.2023.1234567

Curzon, P., McOwen, P., Cutts, Q. & Bell, T., **(2009)**. Enthusing and inspiring with reusable kinaesthetic activities. *Proceedings of the ITICSE 2009*.

Green, S. & Johnson, M., **(2023)**. Cognitive load reduction through IDE usage in programming education. *Journal of Educational Technology*, 29(2), pp.78-94. https://doi.org/10.1080/10494820.2023.1234567

Jones, P. & Smith, L., **(2021)**. Challenges in teaching Java programming to beginners: A case study. *Journal of Programming Languages Education*, 27(4), pp.112-129. https://doi.org/10.1080/10494820.2021.1234567

Lister, R., **(2011)**. Concrete and other neo-piagetian forms of reasoning in the novice programmer. *Proceedings of the Thirteenth Australasian Computing Education Conference*, Perth, Australia, pp.9-18.

Lister, R., Adams, E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M. et al., **(2004)**. A multi-national study of reading and tracing skills in novice programmers. *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, Leeds, United Kingdom, pp.119-150.

Lopez, M., Whalley, J., Robbins, P. & Lister, R., **(2008)**. Relationships between reading, tracing and writing skills in introductory programming. *Proceedings of the Fourth International Workshop on Computing Education Research*, Sydney, Australia, pp.101-112. https://doi.org/10.1145/1404520.1404531

Lorenzo, C. & Peirluigi, P., **(2015)**. Java-meets eclipse: An IDE for teaching Java following the object-later approach. In *2015 10th International Joint Conference on Software Technologies (ICSOFT)*, Colmar, France, pp.1-12. https://dl.acm.org/doi/10.1145/1134285.1134414

Milne, I. & Rowe, G., **(2002)**. Difficulties in learning and teaching programming—Views of students and tutors. *Education and Information Technologies*, 7(1), pp.55-66. https://link.springer.com/article/10.1023/A:1015362608943

Murray, D. & Yang, W., **(2022)**. Java as a foundational language in computer science curricula. *Computer Science Education Quarterly*, 33(2), pp.56-74. https://doi.org/10.1080/10494820.2022.1234567

Nguyen, T. & Hsu, C., **(2023)**. Command-line proficiency and its role in developing robust programming skills. *ACM Transactions on Computing Education*, 23(1), pp.33-50. https://doi.org/10.1145/1234567

Perez, J. & Lee, K., **(2024)**. The role of command-line tools in mastering Java programming. *Journal of Software Engineering Education*, 38(1), pp.85-102. https://doi.org/10.1080/10494820.2024.1234567

Taylor, R. & Martin, S., **(2022)**. The effectiveness of command-line interfaces in programming education: An empirical study. *IEEE Transactions on Education*, 65(2), pp.150-158. https://doi.org/10.1109/TE.2022.1234567

Thompson, J. & Walker, B., **(2022)**. Balancing IDE usage and foundational coding skills in programming education. *Educational Technology Research and Development*, 70(4), pp.987-1004. https://doi.org/10.1007/s11423-022-10010-x

Van Gorp, M.J. & Grissom, S., **(2001)**. An empirical evaluation of using constructive classroom activities to teach introductory programming. *Computer Science Education*, 11(3), pp.247-260.

Williams, D. & Harris, J., **(2022)**. Comparing IDEs and CLIs in programming education: A systematic review. *International Journal of Educational Technology in Higher Education*, 19(2), pp.135-151. https://doi.org/10.1186/s41239-022-00357-7

Wing, J.M., **(2006)**. Computational thinking. *Communications of the ACM*, 49(3), pp.33.