Received: 1 Oct. 2023 | Revised: 14 Oct. 2023 | Accepted: 02 Nov. 2023 | Published online: 24 Nov. 2023

RESEARCH ARTICLE

Contemporary Education and Teaching Research
2023, Vol. 4(11)556-561
DOI: 10.61360/BoniCETR232015151101

Adapting Jupyter for C++ Programming Education : An Empirical Study on Lab Instruction Strategies and Student Perspectives

Mengye Lyu*[,1], Yuan Zhang[1], Shaojun Liu[1] & Lingling Chen[1]
[1]*College of Health Science and Environmental Engineering, Shenzhen Technology University, China*

Abstract: This paper explores the use of the Jupyter platform in teaching C++ programming at the higher education level. It includes a comparative analysis between traditional integrated development environments and online judge systems. The study discusses a teaching approach that leverages JupyterHub and Xeus-Cling to create an interactive learning setting. This approach was pilot-tested during a C++ lab session at Shenzhen Technology University, and student feedback was subsequently gathered and analyzed. A substantial number of students found value in Jupyter's interactive capabilities and the seamless integration of code execution and report writing. However, others experienced difficulties with the platform due to adaptation hurdles, code compatibility issues, and the lack of functionality compared to traditional integrated development environments. The paper further explores the potential of integrating Jupyter with the Visual Studio Code environment to mitigate these limitations and amplify its benefits. The paper concludes by recommending ongoing research and evaluation to refine and adapt this innovative teaching strategy.

Keywords: computer programming; higher education; lab; practice; Jupyter

1. Introduction

The rapidly evolving world of computer science education continuously embraces a diverse range of tools and platforms, aiming to improve the learning and teaching experiences. One of the critical aspects of this education is lab instruction, an area that has seen significant strategic shifts over the years. This paper focuses on the organization of lab instruction for computer programming, with an emphasis on C++ programming and the rising Jupyter platform. Through this research, we aim to contribute to the ongoing discourse on how the instructional methods are changing in computer programming education. We begin with a review of traditional strategies involving integrated development environments and online judge systems, examining their benefits and limitations. We then delve into the possibilities offered by the Jupyter platform, which has gained considerable traction for its interactive nature. To provide a comprehensive analysis, we also describe an empirical study conducted with a class of Intelligent Medical Engineering major at Shenzhen Technology University. This class, during their final C++ programming lab session, was exposed to the Jupyter-based lab instruction. We capture their experiences and feedback, further exploring the potential of incorporating the Jupyter platform in C++ programming lab sessions. In essence, this paper serves as a comparison study, a feedback repository, and a reflective discourse on the potential transition from traditional to more interactive lab teaching environments, while considering the challenges and prospects. The ultimate goal is to enrich the pedagogical approach in C++ programming education, making the learning process

Corresponding Author: Mengye Lyu
College of Health Science and Environmental Engineering, Shenzhen Technology University, China
Email: lvmengye@sztu.edu.cn

more engaging, effective, and rewarding.

2. Review of Existing Lab Instruction Strategies in Computer Programming Courses

2.1. Integrated development environments

Integrated Development Environments (IDEs) have long been the cornerstone of programming education. These comprehensive platforms integrate several tools necessary for software development into a single graphical user interface. This integration allows students to write, compile, debug, and run their code within a unified workspace, thereby streamlining the learning process. For C and C++ programming, tools such as Visual Studio and Dev C++ have been widely adopted. Visual Studio, a product of Microsoft, is a commercial IDE that yet has free community edition and supports multiple languages but is particularly known for its features tailored to C and C++. Dev C++, on the other hand, is a lightweight IDE that is popular for its simplicity and ease of use, particularly in educational settings (Yevick, 2012). Other choices include Code::Blocks (Soto & Figueroa, 2018), Eclipse (Allowatt & Edwards, 2005) and QT creator (Woon & Bau, 2017).

These IDEs offer a range of features, including a code editor for writing and formatting code, a compiler for translating written code into a language that can be executed by a computer, and a debugger for identifying and correcting errors in the code. They also provide build automation tools, which automate common tasks like compiling and running programs, thereby allowing students to focus more on coding and less on the administrative aspects of programming. However, despite their benefits, these IDEs can sometimes present challenges, particularly for beginners. The complexity and multitude of features offered by these tools can be overwhelming, and the need to install and configure the software on individual machines can pose additional barriers to entry. Furthermore, using an Integrated Development Environment (IDE) for programming practice presents challenges for the teachers in assessing the accuracy of a student's code. Given the versatility of solutions for a specific programming problem and the subtlety of some bugs, error identification based solely on code reading can be complex. A program may generate correct outputs from a few test inputs in the IDE, yet fail to function for other edge cases. Consequently, the teachers may struggle to provide timely feedback when relying solely on IDE in lab sessions. This situation can result in students falling into a passive learning mode due to a lack of prompt, corrective feedback.

2.2. Online judge systems

In contrast to the traditional IDEs, online judge (OJ) systems (Wasik et al., 2018), such as HustOJ (Jiang & Xu, 2019), DMOJ (https://github.com/DMOJ/online-judge), and LeetCode (https://leetcode.com) have emerged as popular tools in the modern landscape of programming education. OJ systems were originally used in programming competitions then quickly adopted for computer programming lab sessions in many colleges to provide more practice opportunities for students (Zhang et al., 2023). These platforms provide a collection of problems that students can attempt to solve using various programming languages, including C and C++. The solutions submitted by students are automatically evaluated against a set of predefined test cases, providing instant feedback on the correctness and efficiency of their code.

This approach to learning encourages self-paced, problem-based learning. Students can choose problems that align with their current learning objectives, work on their solutions independently, and receive immediate feedback on their work. This instant feedback loop allows students to quickly identify and learn from their mistakes, a process that is crucial for skill development in programming. Moreover, the competitive aspect of these platforms, where students can compare their performance with others, can serve as a powerful motivator. However, while these systems excel at teaching problem-solving skills and algorithmic thinking, they may not provide comprehensive coverage of all programming concepts. For instance, students may

focus on passing the OJ questions as the only objectives while paying less attention to writing high quality code that is human readable and easy to maintain. Additionally, OJ systems mainly focus on automated evaluation of solutions, but often lack comprehensive code editors and auto-completion features that new learners find helpful. As a result, OJ systems are frequently used alongside IDEs for lab sessions, which adds another level of complexity for beginners. Another common downside of both IDE-based and OJ-based lab teaching strategies is the separation of coding and report writing processes, which makes it cumbersome to document the coding process and insights and increases the homework burden of the students.

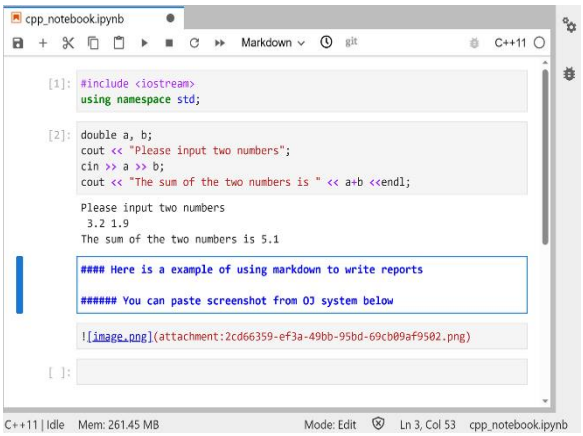
2.3. Jupyter platform

The Jupyter platform (https://jupyter.org), which mainly includes Jupyter Notebook (1st generation), JupyterLab (2nd generation), and JupyterHub (multi-user version), represents a newer approach to programming education. This platform allows for the creation and sharing of documents that contain live code, equations, visualizations, and narrative text. This combination of code and rich text elements allows for a more interactive and exploratory style of learning (Reades, 2020). For C and C++ programming, tools like Cling (https://github.com/root-project/cling) and Xeus-Cling(https://github.com/jupyter-xeus/xeus-cling) can be used to provide an interactive programming environment within the Jupyter platform. Cling is an interpreter for C and C++, while Xeus-Cling is a Jupyter kernel for C++ based on Cling and Jupyter protocol Xeus. These tools allow students to write and execute C and C++ code directly within a Jupyter notebook using any web browsers, providing a level of interactivity and immediacy that is not typically available in traditional C and C++ development environments (Diehl & Brandt, 2023).

However, while the Jupyter platform has been widely adopted for languages like Python and R, its support for C and C++ is not as mature. The use of Jupyter notebooks for C and C++ programming education is still relatively new and less studied, and there may be challenges and limitations that have yet to be fully explored. In this study, we adopt Jupyter platform for one lab session of the C++ programming course with one class in our college, and analyze the results to reveal the benefits and limitations.

Figure 1

The user interface of the experimental C++ programming environment based on JupyterHub and Xeus-Cling. It supports both C++ coding and markdown-based report writing.



3. Research Methodology

To investigate the potential of Jupyter-based C++ lab teaching, we created a Jupyter C++ environment by installing the latest version of JupyterHub and Xeus-Cling on a cloud server. The user interface is shown in Figure 1. This approach was tested in a class of 50 students in Intelligent Medical Engineering major at Shenzhen Technology University during their final C++ programming lab session. In previous sessions, the students had used Visual Studio for programming exercises, submitted solutions to a HustOJ system, and written separate, traditional format lab reports. In this particular lab session, the students were instructed to interactively program in C++ using Jupyter, and to write their lab reports directly in Jupyter using markdown after executing the code. The Pyppeteer package (https://pypi.org/project/pyppeteer) was installed on the cloud server, enabling students to export their lab reports as PDFs, which would then be used by teachers for grading purposes. The students were allowed to choose freely between the traditional lab

report format and the new Jupyter format for their submissions. Those who opted for the Jupyter format were asked to provide feedback on their experiences.

The feedback collected from the students was subsequently analyzed. Comments were categorized as either positive, neutral, or negative. The total number of students who selected each report format was determined, as was the number of students who provided feedback within each category.

4. Results

In terms of report format preference, while a substantial number of students (40%) still opted for the traditional method, the majority of the students (60%) chose the Jupyter Lab format. Specifically, out of a total of 50 students, 30 opted to write their lab reports using Jupyter notebooks, while 20 students chose the traditional format. Regarding the feedback from the 30 students, 6 students expressed positive feelings and 10 students expressed neutral feelings. The advantages they found are summarized as follows.

(1) Interactive and visual learning experience: The students appreciated the interactive nature of Jupyter and found it convenient to run code blocks step by step and view results immediately.

(2) Ease of report writing: Some students found the ability to write code and reports in the same notebook helpful. It saved them the hassle of copying and pasting code from other platforms to the report, thus making the process more efficient.

(3) Less traditional report writing: The format of Jupyter was praised for reducing the workload of traditional report writing, as some elements (like experiment contents) were provided in the template.

On the other hand, 14 students shared negative feedback. The disadvantages and challenges they found are summarized as follows.

(1) Adaptation challenges: A large number of students found it difficult to adapt to the new environment, especially in the beginning. This included issues with syntax, error handling, and unfamiliarity with the interface.

(2) Code compatibility issues: A significant drawback raised was that some code that could run on other platforms like VS and OJ failed to execute in Jupyter. This caused confusion and was a source of frustration.

(3) Limited functionality compared with IDEs: Some students criticized Jupyter for lacking some features they were used to in their previous environments, like keyword or member variable autocomplete.

5. Discussion

The student feedback provided valuable insights into the benefits and challenges of the Jupyter-based lab teaching approach. It was seen as beneficial in terms of providing an interactive programming environment and saving time on lab report writing. However, some students faced difficulties with running code as they are used to in IDEs, and some believed it was more time-consuming than the traditional way.

To partially address the negative feedback, there is an enticing potential in merging the capabilities of the Visual Studio (VS) Code environment with the Jupyter platform to further enrich the learning and teaching experiences in C++ programming. VS Code has strong capabilities for code editing, debugging, and a host of additional features that students are already familiar with, such as keyword or member variable autocomplete. On the other hand, Jupyter provides an interactive platform that allows for efficient code-experimentation and report writing in one place. The integration of these two platforms could take advantage of the best of both worlds and is already made possible with the VS code Jupyter extension(https://github.com/Microsoft/vscode-jupyter). A collaborative workspace that combines VS Code's comprehensive development functionalities with Jupyter's interactive and visual learning features could mitigate many of the issues students reported, while also maintaining and enhancing the advantages. This integration could provide students with a more comprehensive and efficient platform, thus making their learning process smoother and more productive.

The difference between standard C++ grammar

and cling/Xeus grammar can indeed pose challenges for new learners, who therefore reported code compatibility issues. Standard C++ grammar is the established syntax and semantic rules that guide the coding in C++. Most educational resources, online tutorials, and textbooks are based on this standard grammar, and most compilers are designed to interpret it. On the other hand, Cling, the underlying interpreter of C++ kernel in Jupyter, allows for a more interactive C++ experience. While it tries to be consistent with standard C++ as much as possible, there are still differences due to the interactive nature of the interpreter.

For instance, in Cling, code can be written in a more incremental and exploratory manner, running snippets of code in Jupyter cells. This is in contrast to the more structured and holistic approach often required by standard C++, where the entire code needs to be compiled before it can be run. We have noticed some practical differences, including the handling of function declarations and operator overloading, where the current version of Cling only allows one function declaration per Jupyter cell and operator overloading cannot be defined outside a class as a non-member function. These differences can create confusion for new learners, especially those who have already gained familiarity with standard C++ grammar. The cognitive dissonance of working with two different grammars could potentially slow down the learning curve and introduce more room for errors and misunderstandings. As such, the transition between the two grammars, or integrating them into a single learning experience, should be approached with consideration to mitigate these challenges.

6. Conclusion

In summary, while Jupyter provided a new, efficient, and more interactive way of learning C++ programming, it also presented several challenges, mainly related to the adaptation of the new platform, code compatibility, and functionality. These issues might have caused a certain level of frustration among the students, suggesting there is room for improvement and adaptation to optimize the use of Jupyter for C++ programming labs. Continuous evaluation and feedback collection are necessary to enhance and adapt this new teaching method.

Conflict of Interest

The authors declare that they have no conflicts of interest to this work.

Acknowledgement

This research was funded by:

The Teaching Reform Research Project of Shenzhen Technology University (No. 20231026 and No. 20231023).

References

Allowatt, A., & Edwards, S. H. (2005). IDE Support for test-driven development and automated grading in both Java and C++. *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology EXchange*, 100-104.

Diehl, P., & Brandt, S. R. (2023). Interactive C++ code development using C++Explorer and GitHub classroom for educational purposes. *Concurrency and Computation: Practice and Experience, 35*(18), e6893.

Jiang, Z., & Xu, X. (2019). Design and Implementation of Fill-in-the-blank Questions based on Open Source Online Judge System. In *3rd International Conference on Computer Engineering, Information Science & Application Technology (ICCIA 2019)* (pp. 66-70). Atlantis Press.

Reades, J. (2020). Teaching on jupyter. *Region*, *7*(1), 21-34.

Soto, M. S., & Figueroa, I. (2018). Heuristic Evaluation of Code::Blocks as a Tool for First Year Programming Courses. *2018 37th International Conference of the Chilean Computer Science Society (SCCC)*, 1-8.

Wasik, S., Antczak, M., Badura, J., Laskowski, A., & Sternal, T. (2018). A Survey on Online Judge Systems and Their Applications. *ACM Computing Surveys, 51*(1), 3:1-3:34.

Woon, H.-C., & Bau, Y.-T. (2017). Difficulties in Learning C++ and GUI Programming with Qt

Platform: View of Students. *Proceedings of the 1st International Conference on E-Commerce, E-Business and E-Government*, 15-19.

Yevick, D. (2012). A Short Course in Computational Science and Engineering. *In A Short Course in Computational Science and Engineering.*

Zhang, Y., Li, Z., Du, B., Wu, Y., & Jiang, H. (2023). Data Analysis of Online Judge System-Based Teaching Model. In W. Hong & Y. Weng (Eds.) *Computer Science and Education* (pp. 531–543). Springer Nature.

How to Cite: Lyu, M., Zhang, Y., Liu, S. & Chen, L. (2023). Adapting Jupyter for C++ Programming Education: An Empirical Study on Lab Instruction Strategies and Student Perspectives. *Contemporary Education and Teaching Research,* 04(11),556-561. https://doi.org/10.61360/BoniCETR232015151101