Write2Code: Pen-Based Educational Tool for Java

https://doi.org/10.3991/ijet.v16i03.17979

Pedro Guillermo Feijóo-García (✉)
University of Florida, Gainesville, FL, U.S.A
Universidad El Bosque, Bogotá D.C., Colombia
pfeijoogarcia@ufl.edu

Yu-Peng Chen, Shaghayegh Esmaeili, Yingbo Ma,
Christina Gardner-McCune
University of Florida, Gainesville, U.S.A

Abstract—Java instruction generally involves the use of Integrated Development Environments (IDEs) to assist learners in understanding program structure and code syntax. However, the usage of IDEs can make learners' understanding dependent on the computer, which makes it challenging for them to transition to write their solutions by hand for programming assessments and coding interviews. Research suggests that handwriting slows down learners' thinking processes, helping them to better reflect on and engage in problem-solving contexts. We developed *Write2Code*, a handwriting-recognition application, to scaffold students' learning of code syntax and logic by recognizing Java input from their handwritten text. In this paper, we present the tool's architecture, features, and third-party technologies. We also describe how our design helps learners understand Java through an interface that promotes an unplugged experience with feedback similar to an IDE.

Keywords—Computing education, handwritten recognition, visual languages, instructional technologies, scaffolding

1 Introduction

The Computer Science (CS) community has been using the Java programming language for decades as a medium to instruct programming to novice learners [1,2]. Java instruction generally involves using Integrated Development Environments (IDEs) [3]. IDEs provide learners and developers with tools to visualize program structure and code syntax, assisting in the development of scripts and programs. Although the common IDEs for Java involve scaffolding features used in learning and industrial scenarios (e.g., views and controllers created from model classes; code created from UML diagrams), the usage of these tools can make learners rely on the computer as a medium of understanding [4].

Handwritten coding is an existing practice for CS interviews and CS assessments. Prior research suggests that handwriting helps learners to understand and reflect on concepts better than when they use typing as a modality [5,6,7]. However, there is

little research on the benefits of handwriting code as a strategy that helps learners better reflect on their coding skills.

Research from the CSEd community on text-based coding as a strategy [4,8] and findings on intelligent tutoring systems based on handwritten input for education [9,10,11,12] motivate us to explore the following research questions:

- How can a handwriting-based intelligent tutoring system be designed to foster learning of a programming language such as Java?
- What kind of scaffolds do learners need when handwriting code?
- How do learners perform when using a handwritten-based interface that provides feedback as an IDE?

Grammar and syntax are important elements when learning a new programming language, and they lead learners and educators to look for IDEs scaffolding. This prevents learners from experiencing the cognitive advantages of handwriting their own code. This paper describes the architecture, features, and the technologies used to develop Write2Code, as an innovative CSEd tool that allows learners to handwrite Java code as if they were writing on a piece of paper. The tool provides feedback to learners on pre-compilation errors (e.g., syntax and logical errors). We designed this tool to help learners understand the programming language independent from the computer as a medium, with an interface that promotes a CS unplugged experience with the benefits of feedback similar to an IDE.

2 Background

Introductory programming instruction generally involves the use of Integrated Development Environments (IDEs) to assist learners in understanding program structure and code syntax. However, the usage of IDEs can make learners' understanding dependent on the computer, making it challenging for them to transfer their knowledge to paper-based assessments. In this section, we review literature on

1. The advantages of handwriting against typewriting in learning contexts
2. CSEd studies addressing learners' performance in programming paper-based, and computer-based assessments
3. Handwriting-based tutoring systems.

Several studies from different disciplines have compared typing and handwriting to evaluate their pros and cons on learning and reflection [5,6,7]. Mueller & Oppenheimer [5] conducted research on note-taking in college classroom environments and found that learners who used handwriting to take notes performed better when assessed on conceptual questions than those who did not. Learners who handwrote their notes engaged in more processing than learners who typed, by being more selective about the information they wanted to keep and retrieve. Learners who typed their notes tended to transcribe lectures verbatim because they could type faster than when handwriting. This limited their reflection and cognitive processing of concepts. Similarly, Dahlström & Boström [6] compared different writing modalities with elemen-

tary school learners. They argue that the slower process of writing by hand allows for more thought in writing, meaning there are potential links between hand, brain, motor embodiment, and memory. We can relate that argument with Kongsgården & Krumsvik's work [7], who found that learners perceived to learn better when writing longhand versus typing.

When it comes to CSEd and programming as a context, Grissom et al. [8] found that learners assessed on the computer with an IDE performed better when compared to those who were assessed on paper. Nevertheless, regardless of the existing IDE scaffolds and documentation, learners assessed on the computer still exhibited persistent errors such as incorrect calculations and base cases. On the other hand, Corley et al. [4] found no difference in the performance between learners using an IDE format exam or a paper one. However, the authors reported significant differences when referring to syntax errors: learners assessed with the IDE format exam demonstrated fewer syntax errors because they had access to scaffolds, whereas those assessed on paper did not. The access to scaffolds appears to be the main difference between using an IDE or a piece of paper when assessing a CS learner.

As we have presented, handwriting can be used to foster learning and the perception of learning. However, how can a piece of paper compete with scaffolded IDEs that commonly feature typing-based input modalities?

Referring to handwriting as a modality on interactive surfaces such as tablets and hybrid laptops, Anthony, Yang & Koedinger [9] found that handwriting input on interactive surfaces can be useful to foster learning. Their work suggests that handwrite-input intelligent tutoring systems can especially help in problem-solving activities, e.g., algebra in K-12. Similar results can be seen in the work by Kang, Kulshreshth & LaViola Jr [10], which features handwritten recognition for math problems involving algebraic and geometric representations. Work similar to Le & Nakagawa's research [11], which introduces a system that recognizes online handwritten mathematical expressions (MEs) to assist on K-12. De Silva et al. [12] also found tutoring systems useful in contexts such as circuits calculi on Kirchhoff laws.

Tutoring systems built upon handwriting input can be used for problem-based contexts to foster learning. Programming is problem-based, and the language of Java features a grammatically structured context. Thus, a tool such as Write2Code can be useful to foster CSEd.

3 Write2Code: Features and Design

Write2Code has three main features that allow students to solve programming questions using hand-written input:

1. A main canvas for hand-written code
2. Two visual boxes to illustrate the digitalization of code and its corresponding compiled output
3. A button-set featuring actions on code edition and display (see Figure 1).

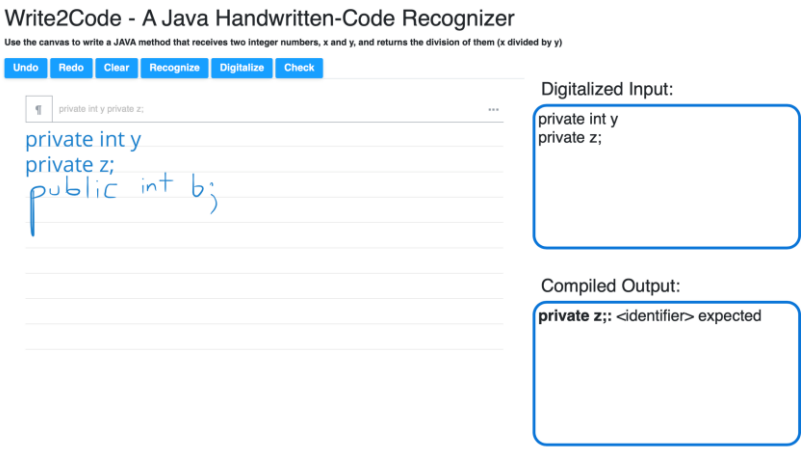

Fig. 1. Write2Code: Graphical User Interface

Fig. 2. Write2Code: GUI Button Menu

The button menu offered to the user features the following actions for editing the written code (see Figure 2):

1. Undo: This button allows the user to step back to previous versions of the written code. This feature serves to track back inputs in order to fix errors recognized by the user or by the system.
2. Redo: Opposite to Undo, this button allows the user to step forward to the most recent versions of the written code. In other words, this function serves as a way to ignore a previous undoing.
3. Clear: This button lets the user clear the input canvas, and the two boxes featured (i.e., digitalized input and compiled output).
4. Recognize: This feature updates the state of the system by asking it to recognize the input currently existing on the canvas.
5. Digitalize: This button transforms the handwritten text into digital text. The button allows users to 1) visualize their input as it will be interpreted by the inner checker, and 2) to edit in an easier way the handwritten input by having a better control of spaces and recognized characters.
6. Check: This feature takes the recognized code from the user's handwritten input and sends it to the inner checker of the system. The outcome of this action is displayed in the box entitled "Compiled Output" box.

In addition to these features, the main canvas recognizes MyScript predefined strokes [13] (see Section 5), allowing the user actions such as deleting a character or

word by scratching it, or adding a space between letters or words by painting a vertical line between them. Write2Code promotes the best of an IDE with an interface that emulates a piece of paper and a pen (https://write2codejavarecognizer.firebaseapp.com). See Figure 3.

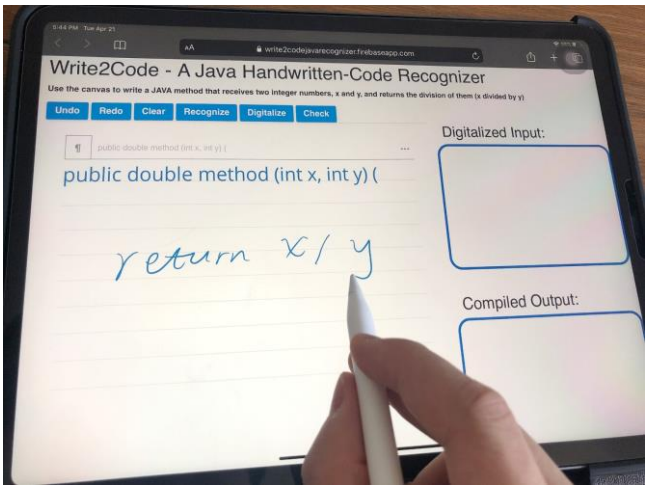

Fig. 3. Write2Code: Proposed Interface

4 System's Architecture

Write2Code is a system created to work as a web application on touch-based client devices (e.g., tablets, touchscreen laptops). As the user interacts with the system through the client device, a sequence of actions is triggered. The flow goes from handwritten text recognition to Java code compilation and syntax-semantic checking. It ends with presenting visual feedback to the user (see Figure 4).

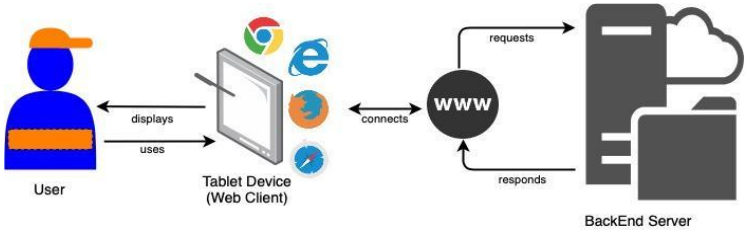

Fig. 4. Architecture: Context Diagram

As shown in Figure 5, our system is built using a 3-tier software architecture, constituted by two main layers. The first one is the Presentation Layer, which gathers those components built for the GUI and front-end features corresponding to the Java Handwritten code recognition. This layer was built using web technologies, was dis-

tributed in two tiers, and reunites the external development dependencies used for this project: Polymer-CLI and MyScript (see Section 5).

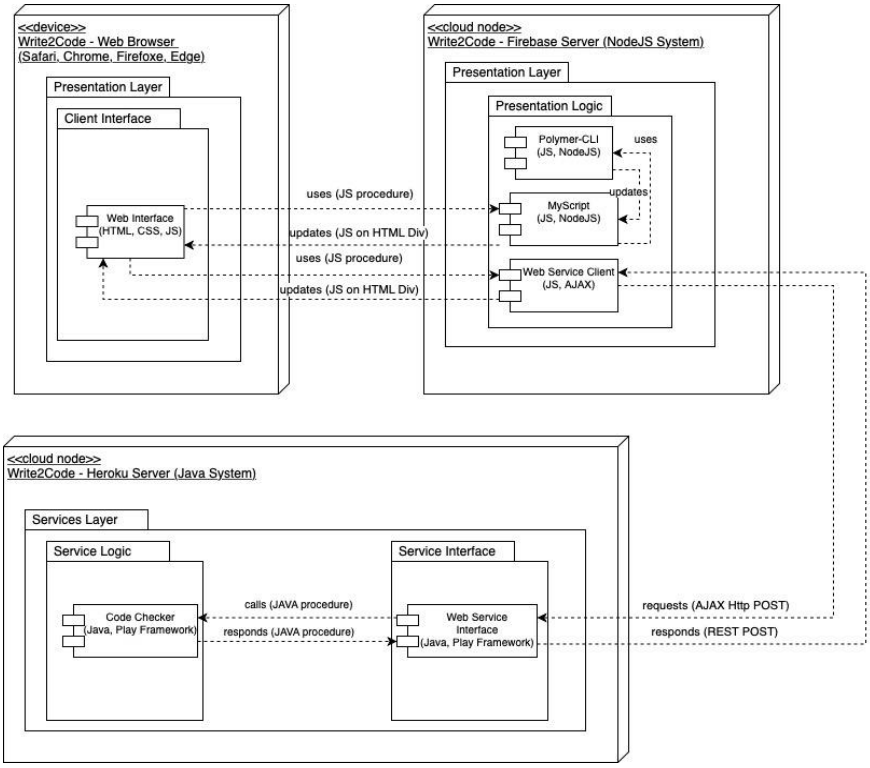

Fig. 5. Architecture: Deployment Diagram

The second layer is the Services Layer, in charge of the Java code compilation and checking tasks. The layer is constituted by two components distributed in two packages. It features a REST API that receives the requests from the Presentation Layer's components and responds with the compilation errors of Java code input by the user. The communication between layers happens using HTTPS-POST requests/responses with the REST API on the Services Layer, and AJAX on the Presentation Layer (see Section 5).

5 Third-Party Components and Technologies

To recognize the handwritten code on a tablet in real-time, we used MyScript API [13], which allows us to perform on-line handwriting recognition with 2D-point sequence as input. Specifically, we used the web component library myscript-text-web from MyScript Interactive Ink SDK [14], which is a development toolkit provided by MyScript. The library provided us with methods that enable desirable handwritten-

recognition functions. The handwriting recognition service is provided by MyScript Cloud [15] as a platform-based service. We accessed the MyScript recognition engine using the server API available in the platform. We also used Polymer-CLI [16] to create custom elements used as standard DOM elements for the GUI web page. Polymer-CLI was used based on implementation recommendations given by the MyScript documentation. SaaS Cloud Platforms were used to deploy and host both the Recognizer and the Checker, connected through the HTTPS protocol and the JavaScript library AJAX [17]. The Recognizer was deployed using Firebase Host Service [18]. The Checker was deployed using Heroku [19].

6 Synthesis, Discussion, and Future Work

In this paper, we have introduced Write2Code as an educational tool that allows novice programmers to learn Java on an interface for handwritten input, providing feedback on pre-compilation errors similar to an IDE. We have also discussed about the potential pedagogical benefit of this design concept based on existing literature on writing modalities, interactive tutoring tools featuring handwriting input, and IDE-based assessments in CSEd. As we presented in Section 2, tutoring systems built upon handwriting input can be used for problem-based contexts to foster learning. Programming is problem-based, and Java as a language features a grammatically structured context. Thus, a tool such as Write2Code can be useful to foster CSEd.

Future work will involve the evaluation of the tool from HCI and CSEd perspectives, addressing the research questions posed in Section 1. We expect to be able to evaluate the tool once the COVID-19 crisis is over, so that we can recruit human participants to interact with our solution.

7 Acknowledgement

The authors would like to express their gratitude to Dr. Jaime Ruiz, Ph.D. for the Natural User Interaction (NUI) course he instructed in Spring 2020 at the University of Florida, U.S.A. Thanks to it, we were able to brainstorm, design, and develop the project that led to this paper.

8 References

[1] Simon, Mason, R., Crick, T., Davenport, J. H., & Murphy, E. (2018). Language Choice in Introductory Programming Courses at Australasian and UK Universities. Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18). https://doi.org/10.1145/3159450.3159547

[2] Becker, B.A., & Quille, K. (2019). 50 Years of CS1 at SIGCSE: A Review of the Evolution of Introductory Programming Education Research. Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19). https://doi.org/10.1145/3287324.3287432

[3] Luxton-Reilly, A., Simon, Albluwi, I., Becker, B.A., Giannakos, M, Kumar, A.N., Ott, L., Paterson, J., Scott, M.J., Sheard, J., & Szabo, C. (2018). Introductory programming: a systematic literature review. Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018 Companion). https://doi.org/10.1145/3293881.3295779
[4] Corley, J., Stanescu, A., Baumstark, L., & Orsega, M. C. (2020). Paper Or IDE?: The Impact of Exam Format on Student Performance in a CS1 Course. Proceedings of the 51st ACM Technical Symposium on Computer Science Education. https://doi.org/10.1145/3328778.3366857
[5] Mueller, P. A., & Oppenheimer, D. M. (2014). The Pen Is Mightier Than the Keyboard: Advantages of Longhand Over Laptop Note Taking. Psychological Science, 25(6), 1159–1168. https://doi.org/10.1177/0956797614524581
[6] Dahlström, D., & Boström, B. (2017). Pros and Cons: Handwriting Versus Digital Writing. Nordic Journal of Digital Literacy, 12(04), 143-161. https://doi.org/10.18261/issn.1891-943x-2017-04-04
[7] Kongsgården, P. & Krumsvik, R. J. (2016). Use of tablets in primary and secondary school-a case study. Nordic Journal of Digital Literacy. 2016(04), 248–270. https://doi.org/10.18261/issn.1891-943x-2016-04-03
[8] Grissom, S., Murphy, L., Mccauley, R., & Fitzgerald, S. (2016). Paper vs. Computer-based: A study of Errors in Recursive Binary Tree Algorithms. Exams. Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE '16. https://doi.org/10.1145/2839509.2844587
[9] Anthony, L., Yang, J., & Koedinger, K. R. (2012). A paradigm for handwriting-based intelligent tutors. International Journal of Human-Computer Studies, 70(11), 866-887. https://doi.org/10.1016/j.ijhcs.2012.04.003
[10] Kang, B., Kulshreshth, A., & Laviola, J. J. (2016). AnalyticalInk: An Interactive Learning Environment for Math Word Problem Solving. Proceedings of the 21st International Conference on Intelligent User Interfaces - IUI '16. https://doi.org/10.1145/2856767.2856789
[11] Le, A.D., Nakagawa, M. A system for recognizing online handwritten mathematical expressions by using improved structural analysis. IJDAR 19, 305–319 (2016). https://doi.org/10.1007/s10032-016-0272-4
[12] De Silva, R., Bischel, D. T., Lee, W., Peterson, E. J., Calfee, R. C., & Stahovich, T. F. (2007). Kirchhoff's Pen: a pen-based circuit analysis tutor. Proceedings of the 4th Eurographics Workshop on Sketch-based Interfaces and Modeling - SBIM '07. https://doi.org/10.1145/1384429.1384445
[13] MyScript. (n.d.). Retrieved from https://www.myscript.com/
[14] Interactive Ink SDK (iink) for web-based platform. (n.d.). Retrieved from https://developer.myscript.com/docs/interactive-ink/1.3/web/overview/introduction/
[15] MyScript Cloud. (n.d.). Retrieved from http://doc.myscript.com/MyScriptCloud/2.3.0/index.html
[16] Polymer-CLI. Retrieved from https://polymer-library.polymer-prject.org/3.0/docs/tools/polymer-cli
[17] Ajax. (n.d.). Retrieved from https://api.jquery.com/jquery.ajax/
[18] Firebase. (n.d.). Retrieved from https://firebase.google.com/
[19] Heroku. (n.d.). Retrieved from https://www.heroku.com/

9 Authors

Pedro Guillermo Feijóo-García is a Fulbright Scholar and Ph.D. student of the Human-Centered Computing doctoral program. He works in the Virtual Experiences Research Group under the supervision of Dr. Benjamin Lok, Ph.D. He is affiliated to the Computer & Information Science & Engineering Department at University of Florida, Gainesville, FL, U.S.A. He is a *Core-Faculty* Assistant Professor of the Systems Engineering Program at Universidad El Bosque, Bogotá D.C., Colombia. (e-mail: pfeijoogarcia@ufl.edu).

Yu-Peng Chen is a Ph.D. student of the Computer Science doctoral program. He works in the INIT Lab under the supervision of Dr. Lisa Anthony, Ph.D. He is affiliated to the Computer & Information Science & Engineering Department at University of Florida, Gainesville, FL, U.S.A. (e-mail: yupengchen@ufl.edu).

Shaghayegh Esmaeili is a Ph.D. student of the Human-Centered Computing doctoral program. She works in the INDIE Lab under the supervision of Dr. Eric Ragan, Ph.D. She is affiliated to the Computer & Information Science & Engineering Department at University of Florida, Gainesville, FL, U.S.A. (e-mail: esmaeili@ufl.edu).

Yingbo Ma is a Ph.D. student of the Computer Science doctoral program. He works in the LearnDialogue Lab under the supervision of Dr. Kristy E. Boyer, Ph.D. He is affiliated to the Computer & Information Science & Engineering Department at University of Florida, Gainesville, FL, U.S.A. (e-mail: yingbo.ma@ufl.edu).

Christina Gardner-McCune is an Associate Professor and the Director of the Engaging Learning Lab. She is affiliated to the Computer & Information Science & Engineering Department at University of Florida, Gainesville, FL, U.S.A. (e-mail: gmccune@ufl.edu).

Article submitted 2020-08-22. Resubmitted 2020-10-05. Final acceptance 2020-10-06. Final version published as submitted by the authors.