CS25-312: Java Pedagogical Libraries for Code Analysis

Final Design Report

Prepared for

Luke Gusukuma

VCU College of Engineering

By

Derek Chiou, Luca Doutt, Ghulam Mujtaba Qasimi, Kennedy Westry

Under the supervision of

Luke Gusukuma

2025-05-06

Executive Summary

Growing classroom sizes and propagation of online classes in a post-pandemic world have presented challenges for both students and instructors in introductory computer science courses, in which timely, quality feedback plays a key role in adjusting to the thought processes and skills needed for students to grow into self-sufficient programmers. The existing Pedal framework, developed by Austin Bart and Luke Gusukuma, integrates with autograding platforms such as Gradescope to help instructors deliver this feedback. While Pedal has found success in the classroom, it is specialized for Python source files where many courses focus on Java. Our aim is to work towards JPedal, a port of Pedal into Java.

Pedal is split into several modules, each of which contributes some functionality for instructors. Since the complete Pedal suite is anticipated to be infeasible to port within the academic year, our focus lands on the CAIT module, which allows instructors to match patterns from the AST (abstract syntax tree) generated from student submissions. We plan to build the CAIT module in steps while leaving the project open for the rest of the modules to be added later.

Table of Contents

Section A. Problem Statement

As class sizes grow and more classes are taught remotely, instructors of introductory-level programming classes face challenges in delivering quality individualized feedback on students' code submissions (Gusukuma et al., 2018). While many courses implement auto-grading technology to display unit testing results or compiler errors, the feedback is usually insufficient for introductory students who are likely unfamiliar with the language of error messages. For example, a student may be tasked with writing a function that takes the sum of a list of numbers. A currently implemented unit test could give the error "expected 10 but was 3," which is usually not specific enough to help identify the underlying gap in understanding.

For this project, we focus on improving feedback in computing and programming courses, specifically the feedback provided to students on their submitted code for unit tests. Since the 1960s, automated feedback systems have primarily emphasized scalability and correctness (Douce et al., 2005). However, these systems often lacked the pedagogical depth to effectively support novice learners. Recent research shows that incorporating immediate, adaptive feedback—tailored to the individual's misconceptions—can improve student engagement and increase their intention to persist in computer science (Marwan et al., 2020). This kind of feedback provides real-time corrective responses, which are crucial for beginners who struggle with debugging their code.

Pedagogically driven tools like Pedal, created by Luke Gusukuma and Austin Bart, offer real-time, targeted feedback that is particularly valuable for novice programmers. This capstone project expands on Pedal by adapting it for Java and working towards a language-agnostic system that can support multiple programming languages. Ultimately, the project aims to enhance both the learning experience and the scalability of feedback, overcoming the limitations of traditional automated grading systems.

Section B. Engineering Design Requirements

B.1 Project Goals (i.e., Client Needs)

Pedal has already been successfully used in the classroom, but it is limited to Python source code files. Many introductory computer science courses instead use Java. The project goals are as follows:

- To port the CAIT (Capturer for AST-Included Trees) module from Pedal to Java
- To build a foundation for the other Pedal modules to also be ported to Java
- To begin building a foundation for a language-agnostic syntactic analysis backend

B.2 Design Objectives

Much of the functionality will replicate modules in Pedal. The JPedal design will:

- extract **syntactic information** from Java source files and expose an **API** to interact with this information.
- provide an interface **for instructors** to control how feedback is shown to students.
- support **immediate** delivery of feedback to students.
- run predefined **unit tests** to check code output.

B.3 Design Specifications and Constraints

Being a piece of software, mechanical constraints are not applicable to this project. With that in mind, the design must:

- be **compatible** with **Java versions 8 and newer**
- process each student submission in **under 5 seconds**
- expose its **source code** for **open access**
- incur **zero monetary cost of use** for instructors

Section C. Scope of Work

C.1 Deliverables

All project deliverables are digital files, minimizing the risk of physical disruptions.

- **UML diagram**: This diagram describes the functionality of each module in the JPedal framework and the channels they have available to interact.
- **Shell code**: A fully interfaced module with test messages in place of implementation.
- **Minimum viable product**: A module with core functionality implemented and tested.
- **Full implementation**: A module with feature-parity with the corresponding Pedal module and successful tests against simulated real-world scenarios.
- **Documentation**: Written information on the module and how to use it.

C.2 Milestones

Refer to the table below for the timeline of different completion milestones as seen throughout the project. Academic deliverables are highlighted in blue.

Task	Completion date
Java library tooling setup (Gradle)	2024-11-07
Fall poster	2024-11-15
CAIT module UML	2024-12-09
CAIT module shell code	2024-12-16
Preliminary design report	2024-12-16
Simple node matching (structural)	2025-02-20
Unit testing framework	2025-02-27
Abstract and EXPO poster	2025-03-28
CLI demo	2025-04-17
Final report	2025-05-06
Continuation package	2025-05-06

C.3 Resources

All the resources we have used for this project are freely available to us:

- IntelliJ Community Edition for writing code, or any alternative IDE
- Git and GitHub for version control

- Open-source Java libraries
- Pedal and its documentation

Notably, we have no particular hardware requirements - our project should be able to be built and run on any modestly powerful machine. In practice, much of the library's code execution will happen on Gradescope's servers, so hardware is not an anticipated obstacle.

Section D. Project Organization

As this is a software project, we use Git and GitHub for project collaboration. Although the original intention was to work off of the previous group's repository, we have instead been mandated to use a fresh repository forked off of the capstone template provided. The predetermined structure of the template is as follows:

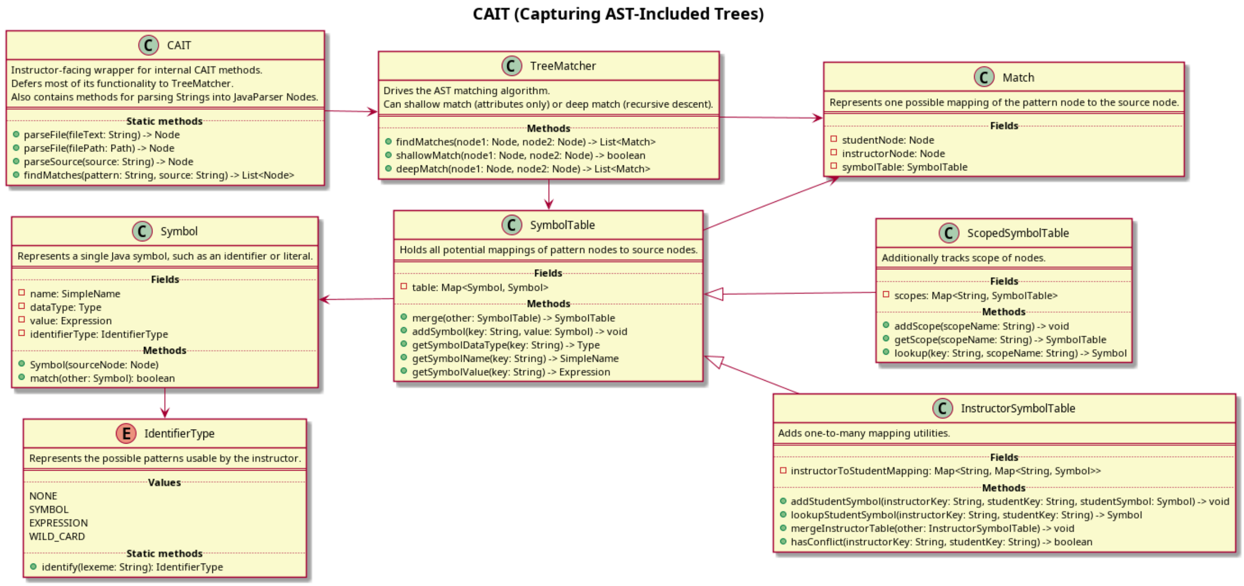
Filename/Directory	**Description**
Documentation/	"All documentation the project team has created to describe the architecture, design, installation, and configuration of the project." Currently empty, as this documentation is currently stored in the Google Drive folder (to include this design report)
Notes and Research/	"Relevant helpful information to understand the tools and techniques used in the project." We have opted to not upload the academic papers cited into this folder due to concerns of usage rights.
Project Deliverables/	"Folder that contains final pdf versions of all Fall and Spring Major Deliverables."
Status Reports/	"Project management documentation - weekly reports, milestones, etc." Also includes weekly meeting summaries.
src/	"Source code." All the .java files go here, organized into packages. IntelliJ was used to assist in the project structure.
.gitignore	As is standard for Git repositories, this file specifies which files to exclude from Git tracking. Ignores are sourced from several online resources (specific to Java and IntelliJ).
build.gradle	Gradle build file. Contains information about required libraries, and files to exclude from the build, such as sample code that would throw compiler errors.

The JPedal code is stored in the "src/edu/vcu/jpedal" package, per the [Oracle standard for naming Java packages](). All code outside of this directory is for testing purposes.

Whenever possible, documentation is written in Markdown (.md) format and stored in the GitHub repository. Markdown has many benefits; it's lightweight, accessible from any text editor, and readable both in plaintext and rendered form. In addition, GitHub exposes the document history for all contributors in case we end up needing to revisit old revisions.

The Capstone course provides many documents in .docx format and requires submissions in .pdf format. For these deliverables, we also incorporate Google Drive into our file organization structure. There is an unfortunate overhead associated with accessing project files in multiple different places with different modes of operation (namely, last year's Google Drive folder, this year’s Google Drive folder, last year’s GitHub repository, and this year’s GitHub repository) in that one might have to search multiple places to access a certain resource. In addition, the physical separation of files makes it behaviorally less likely for team members to reference last year’s resources. However, the concurrent editing capabilities combined with the tool familiarity offered by G Suite are too valuable for us to consider deprecating it.

Section E. CAIT UML



The UML diagram represents the structure and relationships of modules used to achieve the objectives of implementing Pedal to Java and supporting automated, real-time feedback systems for introductory programming courses.

CAIT

Purpose: Extracts Abstract Syntax Tree (AST) data and supports subtree comparison for Java source code.

Functions:

- parseFile: Converts a file (path) to an AST node
- parseSource: Converts a String to an AST node
- findMatches: Looks for matches between two AST nodes

TreeMatcher

Purpose: Compares subtrees to identify matches.

Functions:

- findMatches: Locates matching subtrees between two nodes.
- shallowMatch: Match on types, ignoring children.
- deepMatch: Match recursively on all subtrees.

SymbolTable

Purpose: Tracks identifiers between the two sources.

Functions:

- addSymbol: Adds a symbol to the table.
- getSymbolTable: Shows the symbol table.
- getSymbolName/getSymbolValue: A1 simple method to lookup key properties.

ScopedSymbolTable

Purpose: Additionally tracks scope of nodes.

Functions:

- addScope: Introduces a scope.
- getScope: Retrieves a scope.

InstructorSymbolTable

Purpose: Adds one-to-many mapping utilities.

Functions:

- addStudentSymbol/lookupStudentSymbol: Manages student symbols.
- mergeInstructorTable: Merges with another table.
- hasConflict: Reports on conflicts.

Symbol

Purpose: Supporting module for SymbolTable and Match.

Functions:

- match: Determines whether another Symbol can match.

Match

Purpose: Integration of subtree matching with symbol tracking for code analysis. This module will comprare trees and symbol analysis for reporting feedback.

IdentifierType

Purpose: Represents the different types of identifiers within a code submission. This module helps standardize identifiers.

Values:

- NONE
- SYMBOL
- EXPRESSION
- WILD_CARD

Section F. Design Evaluation Considerations

Python to Java Port

This project is somewhat unique among capstone projects, in that a full implementation of the target functionality, Pedal, is publicly available. However, just as translating a novel from one language to another involves more than a simple chain of dictionary lookups, there are aspects of the design that we have to reconsider when programming for a Java environment.

When one requires a data container, it is common in Python to use a dictionary (key-based) or tuple (index-based), whereas in Java it is more common to use a class (field-based). Therefore, for holding small amounts of metadata, the JPedal design may require more minor classes than are seen in the original Python design. Besides this fact, we must also consider that modularity is essential for any project of significant scale; one should be able to develop and test components in isolation. This holds doubly so for collaborative projects - team members need to be able to work in parallel without interfering with each others' work. For these reasons, the final library is highly likely to contain more actual classes than are outlined in the UML diagram.

Java is a statically typed language, and so its symbols hold type information. This information is accessible from the ASTs produced by JavaParser, so no additional algorithm is needed to check for syntactic correctness as was needed with Python.

Using Previous Work

Last year's group did not produce a functional prototype, so the resources they left for us are primarily informational. Last year's notes serve as a firsthand reference for the concepts and background information required to carry out our design.

Micro Principles

- Each method should have one responsibility.
- For methods with unhappy paths (unmet preconditions), return early rather than wrapping the main body in a conditional.
- Include Javadocs on methods that need them. Javadoc descriptions should be concerned with the results, not the implementation.

Section G. Results and Design Details

Our final implementation reflects a robust symbol comparison system that evolved directly from professor and teammate feedback. We redesigned the *SymbolTable*, *ScopedSymbolTable*, and *InstructorSymbolTable* to more clearly support structural and scoped symbol comparisons. These changes enable us to detect key conflicts during merges, respect nested scopes like methods and classes, and maintain clear separation between instructor expectations and student submissions.

The final **UML** reflects this design: Match objects now hold a reference to a *SymbolTable*, and *TreeMatcher* makes use of both scoped and instructor-level tables to ensure matching is both recursive and semantically accurate. Our approach doesn't just check syntax trees — it also compares variable names, types, and values under hierarchical scopes. This makes it much more useful for pedagogical purposes, such as comparing student submissions with expected templates.

We also added clear conflict flags and improved test coverage. The final code is built for clarity over optimization, using simple logic and student-friendly comments. This lays a strong foundation for future improvements in automated feedback tools for programming education.

Section H. Societal Impacts of Design

When designing a project of this scale, all consequences, whether they are intended or unintended, positive or negative, must be considered. In this section, we will be detailing the possible greater impacts of the completed JPedal project.

H.1 Public Health, Safety, and Welfare

We do not anticipate that our software design will access sensitive data that might pose a public safety risk, nor do we anticipate the potential for malicious actors to use this library in dangerous ways. However, we acknowledge that these risks can be difficult to predict.

Optimistically, the time and hassle saved by this library will go to improving the well-being of both students and instructors. If it performs poorly, however, the opposite may result.

Another safety consideration is that of internal security. Namely, the design should avoid exposing any restricted data to the students which is meant only for instructors. If implemented poorly, auto graded tests might be “gameable” by students, putting academic integrity at risk.

H.2 Political/Regulatory Impacts

Automation is a highly discussed topic in recent years, in no small part due to the boom in publicly available generative AI tools and the associated presence of “AI” in the general culture. While our project does not utilize generators, it does have the goal of automating a task which would typically be done manually. Like most academic research, the regulatory impacts of this one specific project are difficult to know for sure. However, it is conceivable that JPedal contributes to a world of increased automation, decreasing demand for human laborers and bringing with it all the broader political implications of automating work away.

H.3. Economic Impacts

Since we are not manufacturing a product for market but rather writing an open-source library, the economic impact is not immediately evident. Many of the market effects of this design would be indirect and significantly delayed. If we presume that we achieve our goal of improving education quality (primarily in postsecondary education contexts), then it is clear that some economic impacts would result, though they would be difficult to measure.

The other factor to consider is the displacement of human graders, mostly teacher’s assistants. If successful, this library could considerably cut down on the demand for part-time student workers to grade and write feedback on code. While a full discussion of the society-level implications of removing part-time minimum-wage jobs from the market is outside the scope of this report, it is nonetheless an important possibility to consider.

H.4 Environmental Impacts

The only direct environmental impact of JPedal comes from power consumption while running this code on electronic hardware. It is unclear whether this consumption would be larger or smaller than the alternative of manually submitting feedback digitally by humans, but we conjecture that the absolute difference is negligible.

Section I. Cost Analysis

Since our project is purely a software library using freely available tooling, the project has incurred no monetary costs.

Section J. Conclusions and Recommendations

As we concluded our project, we were able to successfully implement the various classes from our UML design, i.e., Match.java, TreeMatcher.java, InstructorSymbolTable.java, etc. These classes demonstrate the system's ability to analyze and match elements within ASTs, which are the backbone of our project's functionality.

Although we were not able to deliver a fully functional demo due to unresolved issues, we did develop and test many foundational features of the CAIT module. Our documentation provides clear guidance for future development. The tests included in the GitHub repository test and validate individual class functionality, and are a good starting point for the continuation of this project.

Looking ahead, we recommend that future teams focus on completing demo integration and improving code robustness. Strengthening internal deadlines and allocating more time to integration testing would also enhance project outcomes. With the solid foundation we have for the next senior design team, this project is well-positioned to be advanced in the future.

See the READMEs in the project repository (root and in /src) for additional notes for continuation of this project.

Appendix 1: Team Contract (i.e. Team Organization)

Step 1: Get to Know One Another. Gather Basic Information.

Task: This initial time together is important to form a strong team dynamic and get to know each other more as people outside of class time. Consider ways to develop positive working relationships with others, while remaining open and personal. Learn each other's strengths and discuss good/bad team experiences. This is also a good opportunity to start to better understand each other's communication and working styles.

Team Member Name	***Strengths Each Member Brings to the Group***	***Other Info***	***Contact Info***
Kennedy Westry	*Proficient in SQL, great with organization, enjoys designing*	*Scrum master certified, so I am proficient in planning and project management*	*westrykj@vcu.edu*
Luca Doutt	*UI, web design, optimization of memory/CPU*	*Experience with Java, HTML/CSS, and game design. Interested in user interactions with software.*	*douttl@vcu.edu*
Derek Chiou	*Algorithms/data management, software architecture, learning software tools*	*Experience as a computer science tutor/TA*	*chioudj@vcu.edu*
Ghulam Mujtaba Qasimi	*QA testing software, white test, and black testing (database and UI)*	*Test the end-to-end software workflow to ensure the expected result meets the actual result.*	*gqasimi@vcu.edu*

Other Stakeholders	***Notes***	***Contact Info***
Luke Gusukuma	*Acts as both our sponsor and faculty advisor*	*gusukumals@vcu.edu*

Step 2: Team Culture. Clarify the Group's Purpose and Culture Goals.

Task: Discuss how each team member wants to be treated to encourage them to make valuable contributions to the group and how each team member would like to feel recognized for their efforts. Discuss how the team will foster an environment where each team member feels they are accountable for their actions and the way they contribute to the project. These are your Culture Goals (left column). How do the students demonstrate these cultural goals? These are your Actions (middle column). Finally, how do students deviate from the team's culture goals? What are ways that other team members can notice when that culture goal is no longer being honored in team dynamics? These are your Warning Signs (right column).

Resources: More information and an example of Team Culture can be found on the Biodesign Student Guide "Intentional Teamwork" page (webpage | PDF)

Culture Goals	*Actions*	*Warning Signs*
Showing up to each meeting on time and prepared	- *Set up meetings in shared calendar* - *Make sure each person is clear on what they need prepared before meetings*	- *Student misses first meeting, warning is granted* - *Student misses meetings afterwards – issue is brought up with faculty advisor*
Proactive communication	- *Set and communicate reasonable deadlines and note when an extension is needed* - *Notify in advance whenever changes need to be made*	- *Student shows up for weekly meeting with no considerable work done* - *Student does not give prior notice when circumstances arise that hinder scheduled attendance*
Being respectful	*-Exercise basic mutual respect* *-Display empathy and understanding for others' viewpoints*	*-When a team member feels hurt or unheard, they should communicate that*

Step 3: Time Commitments, Meeting Structure, and Communication

Task: Discuss the anticipated time commitments for the group project. Consider the following questions (don't answer these questions in the box below):

- What are reasonable time commitments for everyone to invest in this project?
- What other activities and commitments do group members have in their lives?
- How will we communicate with each other?
- When will we meet as a team? Where will we meet? How Often?
- Who will run the meetings? Will there be an assigned team leader or scribe? Does that position rotate or will the same person take on that role for the duration of the project?

Required: How often you will meet with your faculty advisor, where you will meet, and how the meetings will be conducted. Who arranges these meetings?
See examples below.

Meeting Participants	***Frequency*** ***Dates and Times / Locations***	***Meeting Goals*** ***Responsible Party***
Students Only	***Tuesday****: In person 5:30 pm, West Hall Atrium* ***Thursday****: In person 6:00 pm, Location flexible (West 101, Cabell study room, etc)*	*Update group on day-to-day challenges and accomplishments (Luca will record these for the weekly progress reports and meetings with the advisor)*
Students + Faculty advisor	***Monday****: virtually at 1:00 pm (Zoom / Discord)*	*Update faculty advisor and get answers to our questions (Derek will scribe; Kennedy will create the meeting agenda and lead meeting)*

Step 4: Determine Individual Roles and Responsibilities

Task: As part of the Capstone Team experience, each member will take on a leadership role, ***in addition to*** contributing to the overall weekly action items for the project. Some common leadership roles for Capstone projects are listed below. Other roles may be assigned with the approval of your faculty advisor as deemed fit for the project. For the entirety of the project, you should communicate progress to your advisor specifically with regard to your role.

- **Before meeting with your team**, take some time to ask yourself: what is my "natural" role in this group (strengths)? How can I use this experience to help me grow and develop more?
- **As a group,** discuss the various tasks needed for the project and role preferences. Then assign roles in the table on the next page. Try to create a team dynamic that is fair and equitable, while promoting the strengths of each member.

Communication Leaders

Suggested: Assign a team member to be the primary contact <u>for the client/sponsor</u>. This person will schedule meetings, send updates, and ensure deliverables are met.

Suggested: Assign a team member to be the primary contact <u>for faculty advisor</u>. This person will schedule meetings, send updates, and ensure deliverables are met.

Common Leadership Roles for Capstone

1. **Project Manager:** Manages all tasks; develops overall schedule for project; writes agendas and runs meetings; reviews and monitors individual action items; creates an environment where team members are respected, take risks and feel safe expressing their ideas.
   Required: On Edusourced, under the Team tab, make sure that this student is assigned the Project Manager role. This is required so that Capstone program staff can easily identify a single contact person, especially for items like Purchasing and Receiving project supplies.
2. **Logistics Manager:** coordinates all internal and external interactions; lead in establishing contact within and outside of organization, following up on communication of commitments, obtaining information for the team; documents meeting minutes; manages facility and resource usage.
3. **Financial Manager:** researches/benchmarks technical purchases and acquisitions; conducts pricing analysis and budget justifications on proposed purchases; carries out team purchase requests; monitors team budget.
4. **Systems Engineer:** analyzes Client initial design specification and leads establishment of product specifications; monitors, coordinates and manages integration of sub-systems in the prototype; develops and recommends system architecture and manages product interfaces.
5. **Test Engineer:** oversees experimental design, test plan, procedures and data analysis; acquires data acquisition equipment and any necessary software; establishes test protocols and schedules; oversees statistical analysis of results; leads presentation of experimental finding and resulting recommendations.
6. **Manufacturing Engineer:** coordinates all fabrication required to meet final prototype requirements; oversees that all engineering drawings meet the requirements of machine shop or

vendor; reviews designs to ensure design for manufacturing; determines realistic timing for fabrication and quality; develops schedule for all manufacturing.

Team Member	***Role(s)***	***Responsibilities***
Kennedy Westry	*Project Manager*	*Manages all tasks; develops an overall schedule for the project; writes agendas and runs meetings; reviews and monitors individual action items; creates an environment where team members are respected, take risks, and feel safe expressing their ideas.*
Ghulam Mujtaba Qasimi	*Test Engineer*	*Oversees the experimental design, test plan, procedures, and data analysis; acquires data acquisition equipment and any necessary software; establishes test protocols and schedules; oversees statistical analysis of results; leads presentation of experimental findings and resulting recommendations.*
Luca Doutt	*Systems Engineer*	*Analyze Client initial design specification and lead establishment of product specifications; monitor, coordinate, and manage the integration of subsystems in the prototype; develop and recommend system architecture and manage product interfaces.*
Derek Chiou	*Logistics Manager*	*Coordinates all internal and external interactions; leads in establishing contact within and outside of the organization, following up on communication of commitments, obtaining information for the team; documents meeting minutes; manages facility and resource usage.*

Step 5: Agree to the above team contract

Team Member: Kennedy Westry *Signature:* **Kennedy Westry**

Team Member: Derek Chiou *Signature:* **Derek Chiou**

Team Member: Luca Doutt *Signature:* **Luca Doutt**

Team Member: Ghulam Mujtaba Qasimi *Signature:* **Qasimi**

References

Gusukuma, L., Bart, A. C., & Kafura, D. (2020). Pedal: An Infrastructure for Automated Feedback Systems. *SIGCSE*, *Paper Session: Python Debugging*, 1061-1067. https://dl.acm.org/doi/pdf/10.1145/3328778.3366913

Gusukuma, L., Bart, A. C., Kafura, D., & Ernst, J. (2018). Misconception-Driven Feedback: Results from an Experimental Study. *ICER*, *Session 7: Misconceptions*, 160-168. https://dl.acm.org/doi/pdf/10.1145/3230977.3231002

Marwan, S., Gao, G., Fisk, S., Price, T. W., & Barnes, T. (2020). Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science. *ICER*, *Day 3: CS-1*(Novices), 194-203. https://dl.acm.org/doi/pdf/10.1145/3372782.3406264