# Archimedes: Developing a Model of Cognition and Intelligent Learning System to Support Metacognition in Novice Programmers

Phyllis J. Beck
*Computer Science and Software Engineering*
*Mississippi State University*
Mississippi State, MS, USA
pjb82@msstate.edu

M. Jean Mohammadi-Aragh
*Electrical and Computer Engineering*
*Mississippi State University*
Mississippi State, MS, USA
jean@ece.msstate.edu

***Abstract*—This is a Work-in-Progress Innovative Practice paper. This paper introduces a multidimensional model of cognition and an application programming interface (API) for an intelligent learning system, which we will refer to as Archimedes, that implements the first two components of the model of cognition into a web application. The model of cognition currently includes five components thinking processes, organizational strategy, design cohesion, skill mastery, and path of program development. Together these components seek to support six strategies of metacognition, metacognitive scaffolding, reflective prompts, self-assessment, self-questioning, self-directed learning, and graphic organizers. This paper focuses on the implementation and integration of the first two components of the model, thinking processes and organizational strategies into a web application. Together these components integrate Writing-To-Learn and literate programming paradigms to allow the Archimedes application to assess introductory programming students' current level of metacognition and provide automated feedback and allow them to self-monitor, self-assess and improve self-efficacy through the development of strategic knowledge for solving real-world programming problems.**

***Index Terms*—Metacognition, Intelligent Learning System, Machine Learning, API, educational data mining**

## I. Introduction

There is an increasing need for tools and methods for gathering and analyzing data on artifacts and understanding student behavior in introductory computing courses. Several novel approaches have been developed to support different aspects of metacognition, improve self-efficacy and provide automated feedback and assessment for students in introductory programming courses. However, these methods have not been able to successfully scale with studies that suffer from being too small or software that is abandoned due to the high overhead of developing a system to collect and analyze data on a large scale. Despite the success of some methods, most have not been integrated into a cohesive platform where instructors can automate the data acquisition process, develop a persistent trace of time series data such as code path development and see the effects of their interventions as they are affecting their classroom in real-time. There is a need to identify what programming concepts students struggle with the most and to identify at-risk students. Early identification ensures that instructors can intervene and provide the appropriate level of support and improve a student's ability to succeed during the most formative times in their computing education [8].

This research introduces a novel model of cognition and how we developed the Application Programming Interface (API) and integrated the first two components of the model into a web application. This research builds upon previously conducted research [1], [2], [10], [11] and is an exploration in applying machine learning methods to programming education to understand a student's level of metacognition. The motivation for this research stems from the well-documented difficulties students still face when learning to program in an introductory level course [17] and that students still lack the problem-solving skills to apply strategic knowledge to novel problems in the classroom and real-world contexts [9].

The Archimedes platform leverages state-of-the-art technology in natural language processing and machine learning to provide data analytics for students and instructors. For researchers in computing education, Archimedes seeks to automate data acquisition and analysis and provide a more reliable way to accurately measure student performance and mastery of introductory programming tasks. The web-based application allows us to acquire student artifacts at scale through a customizable platform that can be used to identify at-risk students and provide actionable teaching insights on classroom interventions. Additionally, the end goal for the Archimedes project is to develop a tool that allows researchers to conduct independent computing education research and understand what pedagogical practices are best suited for improving student success rates.

## II. Research Questions

**RQ1**: Can we provide a scalable platform that allows us to reliably assess students programming knowledge and process,

This material is based upon work supported by the National Science Foundation under Grant No. DUE-1612132.Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

understand what parts are the most cognitively demanding what are the most common errors that students make?

**RQ2**: What pedagogical insights can a persistent trace of student's programming artifacts tell us about their currently level of mastery?

## III. The Model of Cognition

The Archimedes model of cognition includes five dimensions skill mastery, design cohesion, the path of program development, thinking processes and organizational strategy. Together these components seek to support six strategies of metacognition outlined by Rum [14] metacognitive scaffolding, reflective prompts, self-assessment, self-questioning, self-directed learning, and graphic organizers.

### A. Overview of Metacognitive Strategies

Metacognitive scaffolding seeks to provide assistance to improve competence as it is needed by helping students reflect on their learning process through writing, reviewing key concepts and providing resources, feedback, and strategies for problem-solving at the appropriate time. Reflective prompts are implemented through low stakes writing and question prompts both generic and directed as part of Writing-To-Learn (WTL) strategies. Self-assessment allows students to leverage feedback by evaluating and revising their work improving understanding and learning outcomes. Self-Questioning helps students become more independent, organize their thinking and develop goal-directed learning by asking and answering internally developed questions. Self-directed learning is goal-driven and helps students manage their learning and gauge their learning outcomes. Finally, Graphic organizers are tools such as flowcharts and concept maps that let students visualize information throughout the life cycle of a project. The goal of Archimedes is to interweave these methodologies throughout the components of the model of cognition so that they become self re-enforcing [14].

### B. Model of Cognition Overview

The core of the Archimedes platform is based on a novel model of cognition that integrates five independent components, Skill Mastery, Design Cohesion, The Path of Development, Thinking Processes and Organizational Strategies. Skill Mastery focuses on code literacy and traditional metrics such as program correctness, syntax, mastery of control structures, programming logic, and debugging [8] by using test cases and rubrics which are specified in the system by the instructors.

Design cohesion integrates graphic organizers through the use of flow-charts and pseudo-code to measure a student's capacity for design and to measure prior knowledge activation [15]. We use a dissimilarity metric [13] of low, medium or high to analyze the cohesion level between initial program design and final code implementation to prompt the student to revise the initial design if needed.

The Path of Development looks at students' process over-time and a timeline of actions to illustrate time spent on specific tasks such as design and logs events when students create new program statements, run or compile code, edit an existing statement or delete code. Related work includes Blikstein [5], [13] who used machine learning to generate a Finite State Machine to discover sink states through seven different modeling events and was able to predict future performance by classifying students as either copy-pasters, mixed-mode or self-sufficient. Berland and Marten [3] logged data in open-ended environments and found that students followed one of two development paths planners who carefully structure programs and tinkers who slowly build up their programs as they progress.

The focus of this paper is on Thinking Processes and Organizational strategy. Archimedes builds on previous research [1], [2], [10], [11] in implementing WTL and Literate Programming methodologies to analyze students level of metacognition. The goal of writing-to-learn is to improve metacognition by providing a method for the visual representation of thinking. These activities are meant to allow students to reflect and analyze learning through short, informal tasks which can be used to develop ideas and critical thinking skills by externalizing their thought process. Emig [6] describes writing as "a unique form of feedback" that makes thinking visible immediately allowing it to be used to reinforce an iterative learning process [6]. Additionally, we build on the principles laid out in Knuth's Literate Programming paradigm [7] where the intention is to treat source code as a work of literature where plain English writing is used to explain the meaning of the source code. The programmer utilizes natural language to document the flow of their thoughts and to structure the code in a logical manner where the goal is to write code that is easier for humans to read and understand as opposed to writing code only meant as instructions given to a computer [7].

"Thinking Processes reflect a student's level of strategic knowledge and metacognition. We developed Thinking Processes to give us a method for analyzing the level of reasoning at which students are currently performing when designing and writing a programming solution" [10], [11]. Initial coding efforts have resulted in five classes: literal, conceptual, reflective, organizational, insufficient and none. Organizational Strategies help us understand how a student uses their writing to structure their code. We have developed several features, which are outlined in our previous work, to analyze a students use of white-space, commenting and code segments within a program. Utilizing these patterns we can gain an understanding of how students logically group sections of code and writing, which we refer to as sub-units. These sub-units visually communicate a students' underlying thought process and methods for implementing their design. Initial coding efforts have resulted in five independent organizational strategies: Every-line, Unitization, Block-level, Insufficient and None.

## IV. Methods

The primary goal of this project is to engineer a web-based intelligent learning system that utilizes machine learning and

natural language processing to provide feedback to students and instructors on programming tasks. To effectively track programming progress we first developed an environment where we can track the state of code as it develops using three nested classes, Line, Subunit, and Document. In the first phase of this project we focused on developing an API that lets us track various metrics about the first two model components, thinking processes and organizational strategies, and generate a text based report from the command line. The second phase integrates the machine learning models that were developed in previous research efforts to make predictions on students current level of metacognition. The third phase is the development of the web application which integrates the previous two phases and provides a web based user interface for easy student interaction. The web application allows students to log in, upload code, and generate predictions and metrics on their source code. With this application, we can present various writing-to-learn and literate programming concepts to the student in a user-friendly and interactive context. Feedback is given to help them understand their writing and programming styles and demonstrate what methods they can use to improve commenting and the logical flow of their program. An overview of the application architecture can be seen in Fig. 1.

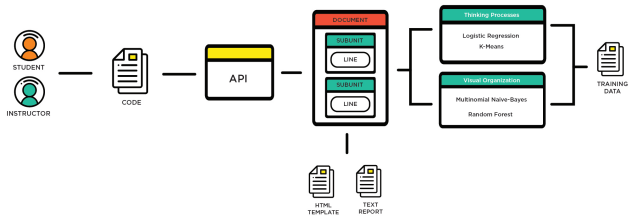


Fig. 1. Archimedes Application Architecture Overview

### A. Phase 1: API Development

The first phase necessary for the development of this application is to engineer an API that can be easily integrated into multiple applications. The primary applications will be to generate a text based report used primarily for testing and research, second to integrate into a web based application and finally to be used to build a plugin for an Integrated Development Environment (IDE). This research does not cover API integration into an IDE but will be part of future work. The API development process resulted in three main classes: Line, Subunit and Document. Each class was built to store core features that are important for statistical analysis and automated classification. We will look at each class independently and then detail the process from code input to report generation.

### B. The Line Class

The Line class is the smallest component and contains properties and methods that are essential to the evaluation of a single statement in code such as the original line number, the raw text value, the line class which is either code or comment, the original indentation level, the programming construct type, and the predicted thinking processes classification for comments. The programming construct type is only applied to code lines and represents various programming constructs such as methods, functions, branching, looping, and exceptions. The predicted comment classification classifies comments according to our thinking processes classification system which we will be discussed as part of the Model Integration phase. By using a class object the state of each line and its properties can be stored as it changes over time and used later to reconstruct the coding process.

### C. The Subunit Class

The Subunit class is the next component in the hierarchy. Each Subunit contains a list of Line objects and is given an identification number. The Subunit class currently has seven properties that keep track of features such as the subunit ratio, the number of comments and code lines, length, and the predicted subunit type based on organizational strategy. Subunits are determined by using a finite state machine to determine when a state change occurs. A new Subunit begins and the previous one ends when the state changes from code to comment. Full details of the development of these features are discussed in previous work [2]. The subunit ratio is determined by calculating the number of comment lines divided by the number of code lines. The subunit type is currently determined by the subunit ratio. If the subunit ratio is greater than or equal to 0.5 it is classified as Every-line. If the subunit ratio is below 0.5 but greater than or equal to 0.1 it is classified as Unitization and if the subunit ratio is below 0.1 it is considered Block-level. We are aware that there are limitations to using this naive classification approach. Currently, it does not account for the insufficient category and there are cases where misclassification occurs such as when there are multiple comments in a unit that can raise the subunit ratio. For example, when there are two comments followed by four code lines, this should be classified as Unitization however because the subunit ratio is exactly 0.5 it is classified as Every-line. The boundary cannot simply be moved as classifying a subunit with a single comment and two code lines with a subunit ratio of 0.5 is correct. We are conducting additional research in developing machine learning models and features that will improve upon subunit classification.

### D. The Document Class

The Document class is the largest component in the hierarchy and encapsulates both the Subunit and Line classes. It contains a majority of the core API functionality and works with the other two classes to generate the necessary features, reports, HTML needed for the web application integration. When the API is given a code file a Document object is created, the code is read and each line is assigned a line type and converted to a line object and stored in a line classification vector. This vector is used to find the subunits that are stored in the units vector on the Document. Once we have constructed

all Line and Subunit objects we can calculate any needed document statistics and finally the visual organization strategy is applied to the full document. Once all needed data is generated we can output a detailed report file containing the document signature, statistics and a text-based representation of each unit where each line has a line number, the line class, construct type, line value, and classification.

### *E. Phase 2: Model Integration*

The second project phase works to integrate two classification models into the core API: the Comment Classification Module and the Visual Organization Module. Both models were developed using NLTK[16] and Sci-Kit Learn[17]. The full process for acquiring the data, developing the features and training these models is outlined in previous research [1], [2]. The Comment classification module currently has two models that can be used for classification: Multinomial Naive-Bayes and a Random Forest Classifier. The classifiers are trained using hand-labeled data and then the model is saved to a pickle file along with the bag-of-words and the transformer model which are necessary components for both models. When a classification needs to be made the pickle file is opened and used to predict the thinking process class and sent back to the API and stored in the Line class. The process for the Visual Organization Module is similar to the first but does not need to store a bag-of-words or transformer. For classifying the organizational strategies we use a K-means Classifier trained on a labeled feature set and saved to a pickle file to be used later for predictions.

### *F. Phase 3: Web Application Development*

The final phase of this project is the development of the Web Application. For the back-end, the application uses MySQL for its database and a python based web framework called Flask. Flask utilizes a template engine called Jinja2 which allows for features such as template inheritance and the ability to easily pass data from the back-end to the front end. The front-end of the application consists of HTML, CSS, and JavaScript which is primarily used with ChartJS to create dynamic interactive charts for displaying some of the document statistics.

Students can register for an account and then login. Once logged in the student can upload a python file and generate a report. When the code is submitted Flask sends the code to the API where the document features are generated, comments are classified and the organizational structure is classified. Using these features an HTML template with the appropriate classes for each line and unit is automatically generated and then passed back to the Flask application. The full document object is given to the application so all data is available for use. Any data that needs to be kept long term is submitted to the database and the report can now be viewed by the user.

For the report view, the user will see three charts. The first chart is the Number of Code vs Comment Lines which shows the break down of line types in the file. The second is the visualization of the Code Signature, each point represents a section where a section consists only of a single line type. The third chart displays a few document statistics such as Average Token Length, Average Unit Length, Average Unit Ratio and the Number of Units. Below the data visualization section, we see the submitted code separated into individual units. Each unit has a colored bar on the left-hand side that indicates its unit type. At the very top of the document is the Visual Organization Class that the Document was classified as. For each line the user will see the original line number, some numbers are skipped because empty lines are not stored. After the line number, a three-letter code is used to label the construct type followed by the indention and the line value. Each comment line is colored according to its classification and assigned a label of sufficient or insufficient.

## V. Conclusions

This paper documents the development of an intelligent learning system and implements machine learning and natural language processing to automatically classify students thinking processes and visual organization which is built from the foundation of writing-to-learn and literate programming paradigms. The API can be used to generate text-based reports or users can engage with the Web Application. Students can upload their reports and receive feedback on their writing and code structure and view various document statistics that can help them self-monitor and self-assess their learning progress. We are currently researching how to determine what patterns indicate an at-risk student from a high performing student and once these patterns have been determined we can then provide feedback to the student on what their coding pattern indicates and provide appropriate intervention.

Further research is necessary to improve upon the classification process through the development of additional data sets and training models to classify comments according to the thinking processes codebook in addition to finding a more flexible approach to classifying document subunits. Additional features for visualizing the breakdown of subunit and comment classification can be added to the web-based application for an increased understanding of how the API decomposes the submitted code. Currently, the application is fairly static and once the report is generated there is not much interaction that the user can engage in. This can be remedied with continued development by incorporating more information on writing-to-learn, question prompts, interactive feedback components that guide students on how to improve their code and a place to write about current learning progress using low stakes writing prompts to further enhance the writing-to-learn integration.

Future work includes integrating an interactive IDE, developing an Instructor Dashboard that can be utilized by educators to monitor students' progress over time and provide personalized feedback to students where needed as well as developing the additional three components of the Model of Cognition.

## References

[1] Beck, P. J., Mohammadi-Aragh, M. J., Archibald, C., et al. (2019). An Initial Exploration of Machine Learning Techniques to Classify Source

Code Comments in Real-time. In American Society for Engineering Education Annual Conference and Exposition.

[2] Beck, P. J., Mohammadi-Aragh, M. J., Archibald, C., Jones, B. A., Barton, A., et al. (2018). Real-time Metacognitive Feedback for Introductory Programming Using Machine Learning. In IEEE Frontiers in Education (FIE).

[3] Berland, M. and Martin, T. (2011). Clusters and Patterns of Novice Programmers. The meeting of the American Educational Research Association. New Orleans, LA.

[4] Bird, S., Loper, E. and Klein, E. (2009). Natural Language Processing with Python. O'Reilly Media Inc.

[5] Blikstein, P. (2011). Using learning analytics to assess students' behavior in open-ended programming tasks. In Proceedings of the 1st International Conference on Learning Analytics and Knowledge (LAK '11). Association for Computing Machinery, New York, NY, USA, 110–116. DOI:https://doi.org/10.1145/2090116.2090132

[6] Emig, J. (1977). Writing as a mode of learning. College Composition and Communication, 28:122–128.

[7] Knuth, D. E. (1984). Literate programming. The Computer Journal, 27(2):97–111.

[8] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory programming: a systematic literature review. In Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018 Companion). Association for Computing Machinery, New York, NY, USA, 55–106. DOI:https://doi.org/10.1145/3293881.3295779

[9] McGill, T. J. and Volet, S. E. (1997). A conceptual framework for analyzing students' knowledge of programming. Journal of research on Computing in Education, 29(3):276–297.

[10] Mohammadi-Aragh, M. J., Beck, P. J., Barton, A., et al. (2019).A Case Study of Writing to Learn to Program: Codebook Implementation and Analysis. In American Society for Engineering Education Annual Conference and Exposition.

[11] Mohammadi-Aragh, M. J., Beck, P. J., Barton, A. K., Reese, D., Jones, B. A., Jankun-Kelly, M., et al. (2018). Coding the coders: A qualitative investigation of students' commenting patterns. American Society for Engineering Education Annual Conference and Exposition.

[12] Pedregosa et al., Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011

[13] Piech, C., Sahami, M., Koller, D., Cooper, S. and Blikstein, P. (2012). Modeling how students learn to program. In Proceedings of the 43rd ACM technical symposium on Computer Science Education (SIGCSE '12). Association for Computing Machinery, New York, NY, USA, 153–160. DOI:https://doi.org/10.1145/2157136.2157182

[14] Rum, S., and Ismail, M. (2017). Metocognitive Support Accelerates Computer Assisted Learning for Novice Programmers. Journal of Educational Technology and Society, 20(3), 170-181. Retrieved April 10, 2020, from www.jstor.org/stable/26196128

[15] Rus, V., Lintean, M., and Azevedo, R. (2009). Automatic Detection of Student Mental Models during Prior Knowledge Activation in MetaTutor. International working group on educational data mining.

[16] Ward, M. (2002). A Template for CALL programs for Endangered Languages.

[17] Watson, C. and Li, F. W. (2014). Failure rates in introductory programming revisited. In Innovation and technology in computer science education, pages 39–44.