A Neural Network Model for a Tutoring Companion

Supporting Students in a Programming with Java Course

By

Melissa Day

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2019 by the
Graduate Supervisory Committee:

Javier Gonzalez-Sanchez, Chair
Ajay Bansal
Alexandra Mehlhase

ARIZONA STATE UNIVERSITY

May 2019

ABSTRACT

Feedback represents a vital component of the learning process and is especially important for Computer Science students. With class sizes that are often large, it can be challenging to provide individualized feedback to students. Consistent, constructive, supportive feedback through a tutoring companion can scaffold the learning process for students.

This work contributes to the construction of a tutoring companion designed to provide this feedback to students. It aims to bridge the gap between the messages the compiler delivers, and the support required for a novice student to understand the problem and fix their code. Particularly, it provides support for students learning about recursion in a beginning university Java programming course. Besides also providing affective support, a tutoring companion could be more effective when it is embedded into the environment that the student is already using, instead of an additional tool for the student to learn. The proposed Tutoring Companion is embedded into the Eclipse Integrated Development Environment (IDE).

This thesis focuses on the reasoning model for the Tutoring Companion and is developed using the techniques of a neural network. While a student uses the IDE, the Tutoring Companion collects 16 data points, including the presence of certain key words, cyclomatic complexity, and error messages from the compiler, every time it detects an event, such as a run attempt, debug attempt, or a request for help, in the IDE. This data is used as inputs to the neural network. The neural network produces a correlating single output code for the feedback to be provided to the student, which is displayed in the IDE.

The effectiveness of the approach is examined among 38 Computer Science students who solve a programming assignment while the Tutoring Companion assists them. Data is collected from these interactions, including all inputs and outputs for the neural network, and students are surveyed regarding their experience. Results suggest that students feel supported while working with the Companion and promising potential for using a neural network with an embedded companion in the future. Challenges in developing an embedded companion are discussed, as well as opportunities for future work.

DEDICATION

This is dedicated to family – my parents, Doug and Leanna Day, and my sister, Erica Day. They encouraged and helped me throughout my entire degree program. Their encouragement to tackle a thesis and to persevere each step of the way spoke volumes to me, and I am so grateful for them.

ACKNOWLEDGMENTS

First, I would like to thank my thesis advisor and chair, Dr. Javier Gonzalez-Sanchez. His guidance throughout the research, project, and writing has been invaluable. He asked me the right questions and shared constructive feedback to strengthen my work. I learned a great deal by working with him.

I am also grateful for my committee members, Dr. Ajay Bansal and Dr. Alexandra Mehlhase, in reviewing and providing feedback. I appreciate the help that Dr. Gonzalez-Sanchez and Professor Acuña provided by allowing their students to participate in the research study.

I am also extremely grateful for the assistance of Manohara Rao Penumala, an alumnus of this program, whose work I continued with this project. He shared technical details of his project with me and continued to make updates to the plug-in. This work was greatly enhanced by his input.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

The education of Computer Science (CS) students is a significant problem that has warranted much attention. In fact, issues surrounding CS education have been identified as one of the seven grand challenges in computing (McGettrick, Boyle, et al., 2005). A common need for CS students is personalized quality feedback (Seymour and Hewitt, 1997), but it is challenging to provide this kind of feedback. One might suggest that the faculty and teaching assistants should devote additional time to giving students feedback on their code. However, given the data on large class sizes (King, 2018; Pisan, Sloane, et al., 2002; Smaill, 2005; Hounsell, 2007), it is extremely challenging – perhaps even unrealistic – for instructors to devote significant time to providing individual feedback to students. Thus, providing feedback to beginning CS students is a problem that needs to be addressed.

The nature of giving feedback to students is complex, and there are many factors involved regarding why and how students engage with certain types of feedback (Price, Handley, et al., 2011). Specifically, in the context of CS programs, feedback may be perceived as even more important due to the difficulty of these courses and the high attrition rate. This can be seen based on the research regarding feedback in general, as it has been rated as being among the top 10 influences of 138 influences studied, using 800 meta-reviews (Hattie, 2009). Furthermore, some have argued that feedback is perhaps among the primary determinants for higher gains in student learning (Hattie and Timperley,

2007; Carless, 2006; Hounsell, 2003). The amount and type of feedback can even influence students' satisfaction with their degree programs (Jessop, Hakim, et al., 2013).

Beginning CS students often struggle to represent concepts in code. Winslow presents evidence that students struggle to produce "syntactically valid statements once they understand what is needed. The difficulty is knowing where and how to combine statements to generate the desired result" (Winslow, Sept. 1996). Additionally, students can experience feelings of anxiety about programming, and since these feelings are typically driven by their thinking patterns, perhaps instructors can help decrease these negative thought patterns by "changing the way the student constructs his/her thinking about writing computer programs" (Connolly, Murphy, et al., 2009). Thus, teaching students problem-solving strategies and, essentially, teaching them how to *think* about programming can bring significant results for the students not just academically, but also emotionally. Consequently, it is the intent of this thesis to provide students with knowledge about how to proceed by giving them hints about problem-solving strategies that will assist them not just with completing the assignment, but also providing them with improved knowledge regarding *why* something needs to change.

Thus, helpful, constructive feedback delivered in a timely manner is important to learning, and CS programs are certainly not an exception to this. Yet, the difficulty of providing this feedback persists. This begs the question: How can this kind of quality, formative feedback be delivered to students? Tutoring Companions as Intelligent Tutoring Systems (ITS) can help to provide such academic support to students. In this chapter we describe key concepts for this topic and the current research. After that, we discuss the problem to be addressed in this thesis and our hypotheses. Next, we provide our proposed

solution and outline our contributions, as well as how we will evaluate these. Lastly, we discuss the scope of this thesis and provide an outline of the structure of this document.

1.1 Key Concepts

It is important to begin by defining some terms and providing an introduction to their significance for this thesis. This section defines such concepts.

Feedback. Feedback is defined as an explanation or information for the student regarding the student's current skill set or knowledge and where it should be, allowing the student to understand what actions need to be taken to make the needed progress (Ramaprasad, 1983). The Companion implemented in this thesis provides feedback about student code.

Hints. Hints are messages to assist with solving the problem (Nesbit, Liu, et al., 2015). Since the goal is that this hint replicates the assistance of a human tutor, the hints must adapt to the student's progress and work (Crow, Luxton-Reilly, et al., 2018). Furthermore, the hints must extend beyond what could be received by utilizing only the compiler errors; otherwise, the hints cannot be considered truly "intelligent" (Crow, Luxton-Reilly, et al., 2018). *Hints* are a specific type of feedback. The Companion implemented in this thesis also provides hints.

Integrated Development Environment. Distinguishing characteristics of an Integrated Development Environment (IDE) are that it supports programming and is also integrated with a software application with key tools available, such as a code editor, compiler, and a debugger, which is accessible through a GUI interface (Chen and Marx, 2005). Specifically, in CS education IDEs are often used to aid students with the

complexities of writing code (Chen and Marx, 2005; Chatley and Timbul, 2005; Spacco, Hovemeyer, et al., 2004; Reis and Cartwright, 2003).

Intelligent Tutoring Systems. An Intelligent Tutoring System (ITS) is a computer system that "performs teaching or tutoring functions…and adapts or personalizes those functions by modeling students' cognitive, motivational or emotional states" (Nesbit, Liu, et al., 2015). Student modeling is a primary distinguishing factor of ITS compared to other instructive systems (Nesbit, Liu, et al., 2015; Sottilare, Graesser, et al., 2013), which involves identifying a student's problem in real-time (Shute and Psotka, 1996). Its ability to customize the teaching function by, essentially, "remembering" the states from student responses, in order to provide better feedback in the future to a student in a similar situation is what sets an ITS apart (Nesbit, Liu, et al., 2015; Ma, Adesope, et al., 2014). In other words, the knowledge of the tutor should be persistent, continually growing over time (VanLehn, 2006; Pillay, 2003).

Tutoring Companion. A companion is an instant, on-demand support system that emulates the traits of a human tutor and is a subcategory of ITS. We implement a companion in this thesis. It differs from an ITS in that it is part of an already existing system, which in this case is the Eclipse IDE. An ITS represents a standalone system, which has many types, such as an alternative IDE designed specifically for teaching or perhaps a web-based ITS that gives lessons to students. Regardless, it is important to distinguish between *Companion* and *ITS* in this thesis. The relationship between Companions, ITS, and other key concepts is shown in Figure 1.1.

Eclipse Platform. Eclipse is an IDE widely used by professionals (Judd and Shittu, 2005) and is commonly used in programming courses (Chen and Marx, 2005; Chatley and

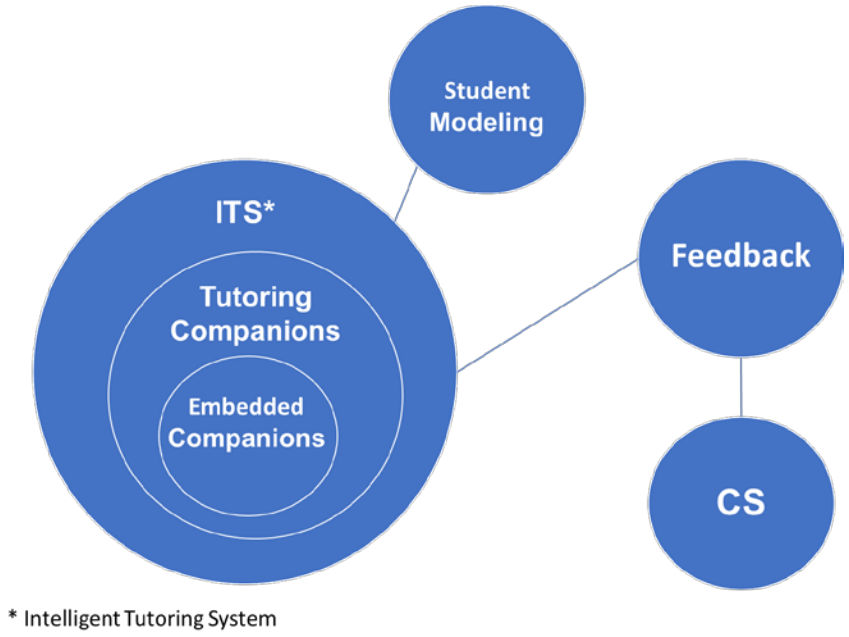


Figure 1.1. Relationship between ITS, Companions, and Embedded Companions as it pertains to student modeling, feedback, and CS

Timbul, 2005; Reis and Cartwright, 2003). One reason for its popularity is its extensibility (Chen and Marx, 2005), which allows other tools or applications to be smoothly integrated with the existing functionality of Eclipse (International Business Machines Corporation, 2006). As an open-source software platform, this allows many to contribute and facilitates adding onto it for various purposes by utilizing the existing components of the IDE (Judd and Shittu, 2005).

Eclipse Plug-ins. An Eclipse plug-in "is the smallest unit of Eclipse Platform function that can be developed and delivered separately" (International Business Machines Corporation, 2006). The Companion implemented in this thesis is an Eclipse plug-in that "lives" inside of the Eclipse IDE. This plug-in connects to the existing Eclipse Platform, allowing students to still use the IDE as they typically would with the Companion's added functionality on top of the existing system.

1.2 State-of-the-Art

ITS have been used for instruction in a wide range of disciplines, including math, medicine, reading, and language-learning, and have been used for students of all levels – from elementary school through university level students (Nesbit, Liu, et al., 2015). Numerous ITS already exist for teaching programming (Crow, Luxton-Reilly, et al., 2018; Nye, Graesser, et al., 2014; Keuning, Jeuring, et al., 2018), and engineers and researchers have made some progress addressing the need for individualized feedback in CS education, albeit in a somewhat limited manner. Many options exist, but more work is needed in this area in order to better meet the needs of a diverse student population.

Training students how to use tools like compilers is an important component of the educational process, so it is important that any type of support that is provided does not deprive students of this. Nevertheless, "the tools involved are complex and require a certain degree of user experience. For example, error messages are often cryptic or misleading and require tutor support to be resolved. Small exercises could help students learn how to read a runtime error or typical compiler errors and how to find a solution more independently" (Ott, Robins, et al., 2016). As experienced programmers, instructors may simplify the process of understanding the error messages, for "what is a problem to a beginner may be a task to someone more advanced" (Winslow, Sept. 1996). Hence, it seems that there is a gap between what beginning CS students can accomplish and the skill level required for troubleshooting the messages received from the compiler.

In fact, it may be detrimental to the learning process of new CS students if they are compelled to rely primarily on error messages to debug their code. Some have found that this process can lead beginning CS students to perceive the ongoing error messages as

failure on their part or that they do not have the ability to perform as needed (Perkins, Hancock, et al., 1989). Others have asserted that students can develop "programming anxiety" – even leading them to "fear" programming (Connolly, Murphy, et al., 2009). This anxiety, which negatively impacts these students' academic performance, is prompted by a response to the complex set of steps given to beginning CS students for which they are not mentally prepared, and these researchers even argue that it has negative effects on retention rates (Connolly, Murphy, et al., 2009).

Consequently, some have proposed the use of an educational tool integrated into the IDE itself (Chen and Marx, 2005; Chatley and Timbul, 2005; Reis and Cartwright, 2003). These modifications to the IDE include modifying the IDE to simplify it for beginners (Reis and Cartwright, 2004), implementing a plug-in so that students can use a simpler programming language in the same IDE (Chatley and Timbul, 2005), or even using a completely different, simplified IDE altogether (Storey, Damian, et al., 2003).

A companion can also provide this function to the student by offering assistance to the student with feedback regarding how to fix the issues in the code. These can be delivered in such a way that this does not interfere with the learning process of how to interpret the error messages received. Instead, a companion can help to bridge the gap between what a compiler is expecting the programmer can do and the typical level of beginning CS students.

Previously, a companion for teaching Java was developed with a focus on monitoring students' actions while completing a programming assignment (Penumala and Gonzalez-Sanchez, 2018). This previous work focused on collecting data from students while completing the assignments, providing the data to the instructor of the course through

a convenient Graphical User Interface to allow instructors to quickly evaluate the progress of their students (Penumala and Gonzalez-Sanchez, 2018). Certainly, instructors need to understand the needs of their students, which may allow them to modify lecture instruction accordingly. However, if class sizes are large, this still will not effectively remedy the problem that CS students need someone to help them individually with their code.

1.3 Problem to be Addressed

However, even with all these existing systems, it is rare for an ITS to offer a combination of features in which it provides feedback to students, as well as something that does not require the use of a new tool. It would be ideal if students could use a tool that would naturally be used in writing code so that they simultaneously learn how to understand the messages from the compiler.

It seems that a tool – a tutor, of sorts – could assist students in beginning CS courses by providing them with feedback. Building upon prior work (Penumala and Gonzalez-Sanchez, 2018), this thesis proposes the use of a companion designed to assist students with programming exercises on recursion in lower-division CS courses. As an embedded companion, this differentiates it in providing feedback to students about how to proceed. Although many ITS exist, it is rare for one to offer this combination of features.

Compiler messages are important but are complicated for beginning students to understand (Ott, Robins, et al., 2016). Furthermore, the IDEs themselves can be at times complex, which can, in fact, exacerbate the issues for beginning CS students learning to program (Chen and Marx, 2005; Reis and Cartwright, 2004), so feedback while using the IDE may be beneficial. Some may argue that this can hinder the learning process because,

after all, students need to acquire the skill of how to decipher the messages produced by the compiler. Few would dispute that students need to understand errors messages and learn how to approach solving these issues; however, for beginning CS students relying primarily on error messages by the compiler can be quite challenging (Ott, Robins, et al., 2016).

Regardless of the approach to resolving the complexity with the IDE, existing options do not effectively address both issues at hand: offering a tool to students that does not increase the complexity of what already exists *and* providing first-rate feedback to students simultaneously. In some cases, students are, in fact, given feedback directly in the IDE; however, it lacks a robust intelligence to allow feedback to improve over time (Spacco, Hovemeyer, et al., 2004). The limitations of the current work can be summarized as follows:

- Existing programming companions within the IDE lack a strong intelligence, especially one that applies current AI techniques.
- Few embedded companions exist that provide feedback to students on problem-solving skills or how to proceed with an assignment. These exist as ITS but are not common within the IDE.
- Many existing programming companions in the IDE for beginning CS students significantly modify the appearance of the IDE or the programming language. This leads to a disconnect between the IDE and language that students use and a standard IDE and programming language.

As a result, more investigation into this area would be beneficial.

1.4 Hypotheses

With the goal of improving the current practice and overcoming these limitations, this thesis investigates a solution to provide feedback on programming assignments to beginning CS students. The primary research goal for this thesis is to extend an existing Companion to enable it to provide valuable feedback to beginning CS students. This thesis addresses the aforementioned challenges and limitations and is stated as follows:

We propose a solution using a neural network to provide students with a sense of tutoring and affective support through feedback and implementing a modular, extensible, modifiable software solution.

Our hypotheses are that students will be receptive to trying this tool and perceive its importance, as they are personally aware of the challenges associated with programming and often seem willing to try new things to help address this situation. Current artificial intelligence (AI) trends will inform our decision regarding how to generate feedback. We expect that the feedback will be slightly helpful to students, although improvements to the AI may be expected in the future. Finally, we expect that the Companion can be fairly effective at generating feedback for programming assignments with recursion, as there are fairly specific problems that arise with recursion.

1.5 Proposed Solution or Approach

We propose a tutoring companion integrated into the Eclipse IDE that provides customized feedback to students while completing recursion programming homework assignments.

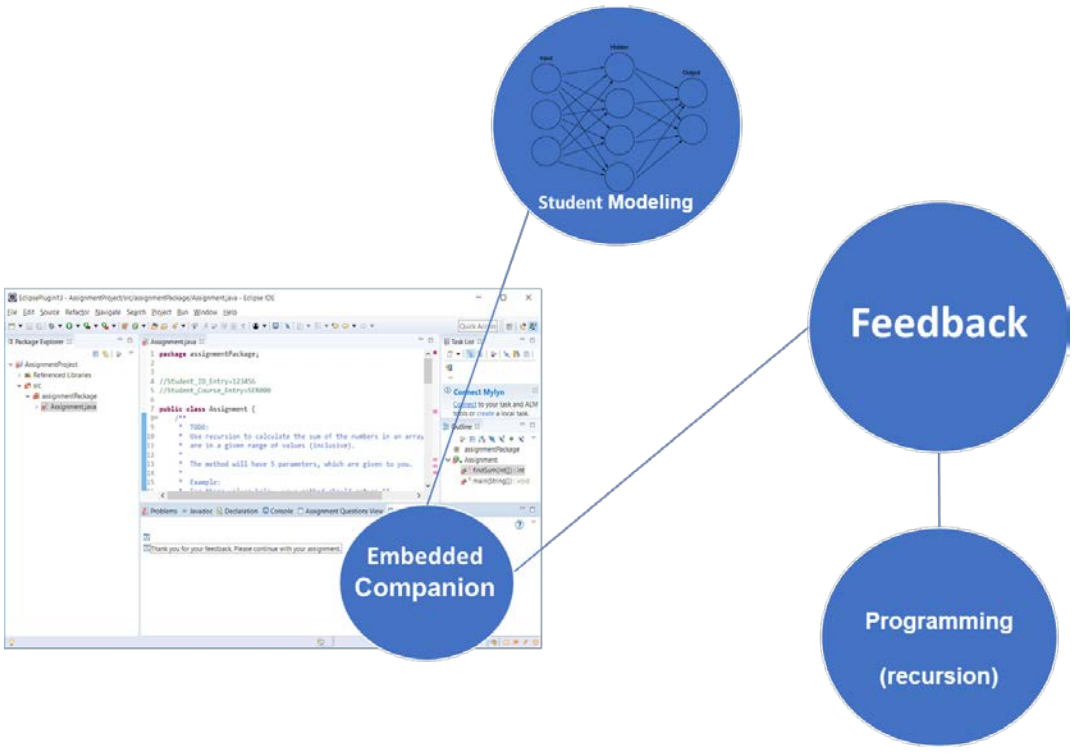


Figure 1.2. The Proposed Solution: The Companion is embedded into the Eclipse IDE. It models the student's knowledge through the use of a neural network and provides intelligent feedback on a recursion programming assignment.

The tutor's goal is to simulate a companion that will provide both content support to the student, as well as affective support. The results of this Companion are examined utilizing beginning CS students in a university programming course. The depiction of the proposed solution is given in Figure 1.2.

Every time a student initiates an action in the IDE, data will be sent from the student's code to the server. Simultaneously, the data from the code will be sent locally within the plug-in to the Brain of the Companion that calculates a response, using the intelligence facilitated through the use of a neural network. The Brain produces intelligent feedback tailored to this situation, which is then displayed to the student. This process, illustrated in Figure 1.3, continues until the student's code is complete.

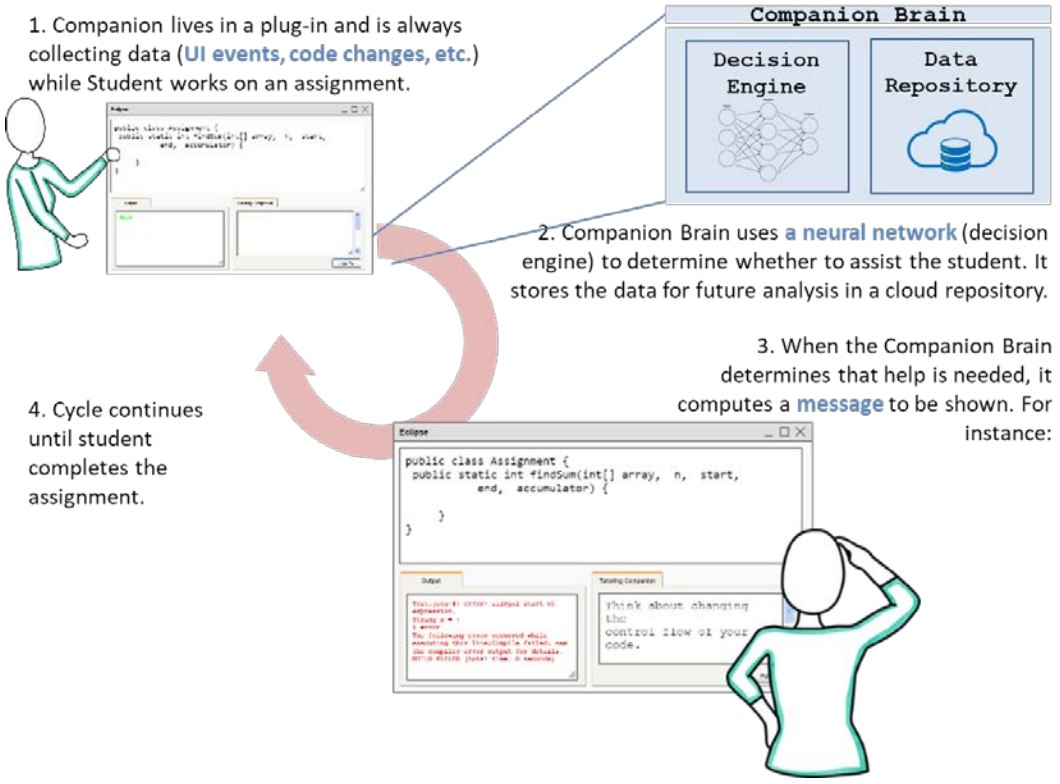


Figure 1.3. Overview of Proposed Solution: The Companion collects data while a student completes a programming assignment. The Brain of the Companion determines individualized intelligent feedback for the student, which the Companion displays inside the Eclipse IDE so that the student will not need to learn a new tool to use the Companion.

Additionally, this Companion provides support to the student so that this "programming anxiety" in beginning CS students is minimized. Connolly et al. conducted a longitudinal study of CS students, investigating the presence of programming anxiety, especially with how this occurred in their first year of the CS program. They write:

> *The extent of negative cognitions regarding control in computing situations, and the lack of sense of computing self-efficacy shown in this study strongly points to the need for strategies to be devised by computer programming course designers to foster student confidence and motivation…The findings…would suggest that providing learners with opportunities to gain experience in skills involved in programming computerized equipment routinely themselves, for example…dealing with error messages, would be important elements of developing confidence and perceptions about computer programming* (Connolly, Murphy, et al., 2009).

The Companion implemented in this study has this goal in mind. Since it is an additional feature in the Eclipse IDE, which already needs to be used as part of the software development cycle, it will scaffold the learning process for the student. This still allows the student to learn from the error messages produced by the compiler, but also provides support and encouragement through the feedback delivered.

1.6 Contributions

In this thesis, we assert the use of a tutoring companion integrated as a plug-in into the Eclipse IDE for providing feedback to students. We provide an overview of existing research, introduce the intelligence of the Companion, examine its effectiveness and whether it fulfills the goals set for this study, and evaluate the Companion via CS students using it for an assignment. This thesis advances the state-of-the-art in CS education by integrating a tutoring Companion into the Eclipse IDE, delivering intelligent feedback by utilizing current work in AI, and investigating its effects among beginning CS students. The key contributions of this thesis include:

- **The Software of the Companion.** Using best practices in software engineering, the product is a sophisticated working software. On the front-end, the students install it as an Eclipse plug-in. On the back-end, it sends the students' data to the neural network and the database – both of which are located on the server. Since all of these are implemented with existing enterprise-level tools, rather than creating it from scratch, this increases the complexity of the software in achieving the desired functionality.

- **The Brain of the Companion.** Leveraging AI technology through the implementation of a neural network, it represents an improvement in existing ITS for programming, allowing students to receive intelligent feedback without requiring a separate tool just for the ITS. Additionally, we determined necessary input data for the neural network to respond to possible problems in the code and created messages for the Companion designed for its educational purpose.
- **The Research Study of the Companion.** Demonstrating the effectiveness of the Companion developed for particular users, namely, beginning CS students, its effectiveness was examined among this target audience. Furthermore, the study provides evidence that the software developed works and functions according to expectations. These students are enrolled in at least one of two lower-division CS courses. One of these courses includes an introduction to different programming languages, and the other course is on elementary data structures.

1.7 Evaluation Plan

Evaluating the Companion includes:

- **Testing.** The software will be tested according to the principles of Software Engineering. This includes the use of component, integration, and Alpha testing to validate that the software works as expected.
- **Software Release.** The Companion's validity is further validated by releasing it to a set of users who are, in a sense, the "customers" for the software. Having students, the intended audience for the Companion, use the Companion validates not only

that it works, but also that it can be used for its intended purpose: delivering feedback to students about a programming assignment.

- **Evaluation of the Companion's Feedback.** The students using the Companion will provide opinions regarding how helpful the Companion's feedback is for various scenarios with their code. For each message delivered, the plug-in prompts students to evaluate how helpful it was, and the plug-in logs this data for the data in the student's code. Thus, this will be helpful in improving the Companion in future iterations and evaluating whether the Companion's feedback was effective. Additionally, students will complete a survey at the conclusion of the experience with the Companion, providing additional data regarding the helpfulness of the feedback and their reactions to the Companion.

1.8 Scope of Work

Several topics are beyond the scope of this thesis, including:

- **Continual Execution of a Neural Network.** Currently, the neural network developed in this project requires human intervention to train the model with collected student data. A fully automated Companion would learn at all times and provide feedback without requiring a human in the data training process. This, however, is beyond the scope of this work.
- **Cross-compatibility.** Plug-ins are typically developed for the current version of Eclipse, but they often work with other versions of Eclipse, as well as for other Java versions. Since this was not a focus of this study, development efforts were not focused on this facet of the software.

- **Companions in Other IDEs.** Companions can also exist in other IDEs used for Java development, such as Visual Studio, NetBeans, or IntelliJ. Although these may also be worthy of consideration, they are not evaluated or included in this study.

1.9 Document Roadmap

The rest of this thesis will describe the research and work on the Companion. Chapter 2 provides a survey of the existing work on this topic and establishes the need for an integrated tutoring Companion with this type of intelligence. Chapter 3 contains a detailed explanation of the Companion's intelligence, including the technical implementation details of the neural network and the model generation. Chapter 4 discusses the technical details behind the Companion's Brain, such as architecture, design, and testing. Chapter 5 comprises the details on the study completed with the students. Chapter 6 presents the results of the study, and, lastly, chapter 7 discusses the results and plausible conclusions, along with possible future research areas.

CHAPTER 2

BACKGROUND

The tutoring function performed by our Companion involves giving feedback to the students, as well as "offering prompts to provoke cognitive, motivational, or metacognitive change" (Nesbit, Liu, et al., 2015). Such feedback could improve a student's interest in learning through an interesting message or the use of humor (Ma, Adesope, et al., 2014).

2.1 Tutoring Companions Background

ITS for teaching programming typically exist for a distinct purpose, such as helping students with a particular concept in programming, and are often designed for a specific programming course at a certain university (Nesbit, Liu, et al., 2015; Crow, Luxton-Reilly, et al., 2018; Nye, Graesser, et al., 2014; Keuning, Jeuring, et al., 2018). They often fall into two categories for their intended function: assisting a student with a programming exercise or supporting the debugging of code (Pillay, 2003). Various facets of teaching or tutoring functions exist in these systems, such as providing lessons to students with an explanation of key concepts, giving short programming exercises for students to complete, or offering suggestions on improving the efficiency of the code (Pillay, 2003). At any rate, these are a *system*, rather than being integrated into an already existing tool.

Many different categories have been used to analyze ITS for teaching programming, ranging from learning outcomes (Nesbit, Adesope, et al., 2014) to the age of the students in the target audience (Pillay, 2003) to the classification of the feedback provided (Le, 2016). One type of feedback in ITS for programming is adaptive feedback.

Adaptive feedback consists of not just telling the user whether an answer is correct in a binary sense; rather, it provides different information for different students in different situations (Le, 2016). Some have suggested that the feedback in this approach could be the line number in the code with the error, an explanation of a concept, or a customized hint (Le, 2016).

Feedback in general, not just for programming, has been divided into various categories, including knowledge about mistakes, knowledge about how to proceed, and knowledge about concepts (Narciss, 2008). Automated feedback generation tools for programming assignments have been classified according to these types of general feedback. Knowledge about mistakes accounts for 96% of feedback tools, while knowledge about how to proceed accounts for about 45% of available tools, although tools may fall into multiple categories (Keuning, Jeuring, et al., 2018). Even though 45% of tools already developed are in this category, many in this category do not have strong features, such as being embedded into an IDE. A discussion of specific examples of programming ITS in each general feedback category will now be provided.

Feedback about Knowledge about Task Constraints. One example of a tutoring system that provides feedback on task constraints is INCOM, which is for teaching logic programming (Nguyen-Thinh, Menzel, et al., 2009). This operates by highlighting keywords about what the student missed. Other types, such as BASIC Instructional Program (BIP), which is a historical example of an ITS, require that a component of the language is used, so the system will deliver a message stating that the student is missing a keyword (Barr and Beard, 1976). These types of systems may have a benefit in assisting the student to complete the assignment successfully, but they will most likely not help the

student to think through similar types of problems in the future, as the student will not improve his or her understanding of how to solve that type of problem. Systems in this category represent about 15% of available tools, and the focus is on helping the student with breaking down the tasks (Keuning, Jeuring, et al., 2018).

Feedback about Knowledge about Concepts. Next, an example of knowledge about concepts is the FIT Java Tutor, that allows students to compare their solution side-by-side with a sample solution. In some cases, this ITS provides the student with an example containing mistakes, asking the student to correctly identify the mistakes.

Another example, the Lisp Tutor, which is an early ITS, continuously monitors the student's work for errors, and, once an error is detected, it provides guidance to the student (Anderson and Skwarecki, 1986). It also offers students reference materials on various topics, guiding him or her through a series of examples for similar situations and concepts (Crow, Luxton-Reilly, et al., 2018; Keuning, Jeuring, et al., 2018). Although the feedback offered by Lisp Tutor may have benefits, it lacks the ability to deliver feedback in non-erroneous scenarios.

Feedback about Knowledge about Mistakes. Tutors in this category often report test failure cases. Online Judge reports whether test cases pass or fail, and another system, ProgTest, which teaches testing, has students upload their code and test cases. Then, it reports the results of running the instructor's test, along with the code coverage analysis given the student's test cases (Keuning, Jeuring, et al., 2018). A well-known early programming tutor in this category is PROUST, which helps beginning Pascal students locate errors in their programs (Johnson and Soloway, 1984). This tutor functions by generating a list of all errors in the student program (Pillay, 2003). COALA is a tool that

provides the results of running JUnit test cases in the Eclipse IDE (Jurado, Redondo, et al., 2012). Knowledge About Mistakes may also involve providing more details on compiler errors, such that the students do not need to use an actual compiler, even going so far as to replace an actual compiler (Keuning, Jeuring, et al., 2018). However, it is a vital skill for students to learn to use the compiler directly, so replacing the standard compiler in an IDE is not desirable. Other ITS may give feedback on solution errors by attempting to match the student's code with a sample solution program. Singh13, which is one such type of tool, gives students the line number and function that needs to be changed with the exact change that should be made to the code (Singh, Gulwani, et al., 2012). Moreover, systems like Singh13 do not particularly benefit a student's problem-solving ability. In fact, only providing the solution to the problem will resolve the issue at that moment, but in a similar scenario in the future, the student will likely have the same issue. Although these types of debugging tutors might have been helpful in bygone eras, IDEs can, in general, provide this for students, so it seems that this type of system is not really relevant today. Furthermore, most IDEs have debugging support built into them, so to create this type of system is not a good use of resources. Assisting students with writing a program is still something that is relevant and needed, as much of what goes into this involves how to think about the problem, which is more than just understanding the syntax of the language. Thus, the programming Companion we developed falls into this first category.

Feedback about Knowledge about How to Proceed. In the literature review conducted by Keuning et al, they found that 45 of 101 tools reviewed provide this type of feedback (Keuning, Jeuring, et al., 2018). They further subdivide this category into three other categories: bug-related hints, program improvements, and task-processing steps.

Bug-related hints focus on assisting the student with fixing errors, such as fixing errors with spelling or checking input for validity. Some, such as CSTutor, offer these in the form of questions to greater assist the student's learning (Keuning, Jeuring, et al., 2018). Another category, program improvements, provides ways the student can improve the performance or style of the program (Keuning, Jeuring, et al., 2018). Even though these are important considerations, if novices cannot represent the basic logic of their task in a programming language (Winslow, Sept. 1996), these are secondary issues. Lastly, task-processing steps offer feedback for how to think about the problem without examining the student's current code. A Prolog tutor providing task-processing steps, Hong04, gives hints on how to logically step through the problem with templates for the student to fill in the blanks. Ask-Elle, which is used to teach the functional programming language Haskell, gives suggestions for how to attack the problem, which can help the student to learn how to think about the problem as if the instructor were present, walking the student through the thought process.

Feedback about Knowledge about Metacognition. Keuning et al. found only one example of a tutor focused on Knowledge About Metacognition, HabiPro, which asks students to justify their answers (Keuning, Jeuring, et al., 2018). This type of system is rare and is often implemented using natural language processing.

2.2 ITS in the IDE

While programming ITS with adaptive feedback offer many features, most of the features center around providing lessons and reference materials to students, rather than being directly embedded into an IDE (Crow, Luxton-Reilly, et al., 2018). Nevertheless, there are

a handful of ITS embedded into the IDE that are worth considering. One such ITS, Cimel ITS, which was used to teach program planning and modeling, integrates the tutor with the Eclipse IDE, sending UML designs from Eclipse to be evaluated by the system (Moritz, Wei, et al., 2005). Although Cimel is an Eclipse plug-in, its essential purpose is different because it teaches UML, instead of assisting with a programming exercise. Other similar plug-ins for UML have also been developed (Moritz, Blank, et al., 2007). Nonetheless, these are not really programming ITS because they teach modeling, not programming.

ITS as Eclipse plug-ins exist for a variety of purposes, such as helping with collaboration, especially with pair programming (Devide, Meneely, et al., 2008; Jurado, Molina, et al., 2013; Yusri, Mashita Syed-Mohamad, et al., 2015). An Eclipse plug-in was also developed to track keystrokes, classifying students' level of activity in a programming lab setting to aid tutors with knowing whom to assist (Karkalas and Gutierrez-Santos, 2014). Another example is a plug-in to assess students' code, evaluating whether it is suitable for submission (Silva, Leal, et al., 2018). An interactive tutorial that records students' actions in the IDE to be preserved as a tutorial for future students also exists, but this differs significantly from an ITS programming tutor that provides feedback (Zhang, Huang, et al., 2009). Another tutor, Coala, assists students and teachers with assessing a programming algorithm by collecting data from students' code through an Eclipse plug-in (Jurado, Redondo, et al., 2014). Thus, there are ITS worthy of consideration, but their focus is not on giving students feedback on how to proceed.

2.3 Intelligence for ITS

Various approaches have been used for intelligence in the ITS. Before considering pure AI methods in this section, we will consider a few other existing methods for providing feedback on programming in an ITS. Sometimes these possess some similarities to AI methods, but all aim to help students with their programming assignments regardless of the method for generating the intelligence.

Generating Feedback through Data Analysis. Data analysis, which can also be referred to as data-driven ITS (Rivers and Koedinger, 2017), involves "using large sets of historical student data to generate hints" (Keuning, Jeuring, et al., 2018), and only about 8% of existing systems use this method. Decisions within the ITS are based on prior students' work, instead of relying on an expert knowledge base that requires an instructor to enter solutions (Rivers and Koedinger, 2017). One reason that data-driven techniques for generating hints can be helpful is because it helps to resolve the "cold-start problem" (Chow, Yacef, et al., 2017). When the first students initially use an ITS, some of the early hints may be poor or even non-existent. The developers of the Grok Learning platform, an online programming tutoring system, found that when using a data-driven approach, they only needed data from 10 students to generate a quality hint (Chow, Yacef, et al., 2017). Thus, it seems using historical student data is worthwhile. The Intelligent Teaching Assistant for Programming (ITAP) is another ITS that uses data analysis, with the goal of being a self-improving tutor so that feedback is increasingly better matched to each unique student (Rivers and Koedinger, 2017). ITAP "creates a solution space graph with (intermediate) program states as nodes, in which directed edges represent next steps" (Rivers and Koedinger, 2017). After matching a student solution with a node in the graph,

it searches for a path from the student's current node to a correct solution, using the path as the basis for the hint (Rivers and Koedinger, 2017). Although this method has found some success with ITAP, ITAP cannot handle incorrect syntax (Rivers and Koedinger, 2017). Furthermore, if data analysis is used on its own without the advantage of machine learning techniques, it is time-intensive and involves a complex process to determine how to traverse the graph to an ideal solution, as well as how to match a student's code with a current node in the solution graph. Whereas these ITS rely, typically, on some graph traversal algorithm that the developers must create, a machine learning approach can build on existing software libraries for machine learning without being limited to the arena of ITS.

Generating Feedback through Static Analysis. Static analysis tools for code are ubiquitous, and these are frequently incorporated into university classes to enforce coding standards through a variety of tools (Keuning, Jeuring, et al., 2018). However, as Keuning et al. argue, the disadvantage of such tools is that their messages are often complex and not tailored for beginning programming students.

Generating Feedback through AI. AI techniques are relatively common for generating feedback, with approximately 28% of surveyed tutors in a major literature review relying on AI for feedback generation (Keuning, Jeuring, et al., 2018). Many different AI techniques exist, which range from giving intelligent feedback through the execution of test cases to more elaborate methods involving tracing the student's solution model compared with a model of a sample solution (Le, Strickroth, et al., 2013). Some so-called AI methods are limited to "detecting errors in student solutions in order to provide feedback" (Le, Strickroth, et al., 2013). However, a compiler can provide these types of

feedback, so this does not really represent a needed area of research. Another example of an AI method for feedback is through natural language processing. AutoTutor is perhaps the quintessential example of such a system (Graesser, Lu, et al., 2004). Seeking to simulate a conversation with a human tutor, it uses a predictive script to determine which direction to take the student. In these types of approaches, an expected answer exists for each question the tutor asks. If the answer given is not what the expected or "right" answer is, then a series of remediation steps is generated (Le, Strickroth, et al., 2013). Nonetheless, the focus of AutoTutor has not typically been assisting students with programming. PROPL, a natural-language processing ITS designed for programming, guides a student to a pseudocode solution using a series of questions and natural-language processing (Le, Strickroth, et al., 2013). Yet, PROPL only assists with creating a pseudocode solution, not with the actual coding process.

Generating Feedback through Machine Learning Techniques. Machine learning, a specific type of AI, is another possibility for generating feedback. JavaBugs sought to identify a student's intention by using a machine learning approach to generate a library of common bugs for beginning Java students (Suarez and Sison, 2008). This ITS utilized a set of sample solutions to the problems and compared the student's solution to these samples, using it to determine how the student intends to solve the problem. Using machine learning, they determined how the student diverged from this program and updated the error library accordingly. Although they did return feedback to the student, the true aim of this study was using machine learning techniques to generate a bug library, rather than using it to generate the feedback itself. Others have used neural networks to generate hints (MacNish, 2002; Beck, Woolf, et al., 2000).

2.4 Summary

Certainly, much work has been done in the area of programming ITS, including companions embedded in the IDE and even ITS with intelligence. One might wonder why yet another type of ITS could be worthwhile. Among these options, few, if any, exist as an embedded companion in the IDE, utilizing a neural network to generate feedback about how to proceed with a programming assignment, as well as offering affective support. Of course, companions embedded into the Eclipse IDE for programming assignments already exist. Coala, for instance, is an Eclipse plug-in that adds information to the IDE to help students with their code and extracts data from students' code (Jurado, Redondo, et al., 2014). Nevertheless, it also differs in significant ways, as this tutor requires an instructor to enter a sample solution to generate the data used for evaluating students' code. Furthermore, its feedback most closely resembles that from running test cases. CIMEL also exists as an Eclipse plug-in, but this is for UML, not for programming (Moritz, Wei, et al., 2005). Intelligent ITS also already exist, such as ITAP (Rivers and Koedinger, 2017), but it cannot handle incorrect syntax and also does not minimize instructor involvement. Thus, this type of companion – one in the Eclipse IDE delivering intelligent feedback with hints about how to proceed with an assignment – is situated in a unique niche among prior ITS.

CHAPTER 3

THE TUTORING COMPANION BRAIN

The intelligence for the Brain is implemented using a neural network with a supervised learning algorithm. We considered other options for the intelligence and found that a neural network is a good fit for this data and the intended purpose of the Companion. Before a more detailed discussion of the brain, we will briefly explain the rationale for choosing a neural network over other possibilities.

Besides neural networks, possibilities for the brain include linear regression and Bayesian networks. Linear regression has been used with ITS for simple classification problems (Beck, Woolf, et al., 2000), but it is too simple because the inputs in an ITS can have a high degree of variance, which requires a more sophisticated algorithm than that supported by linear regression. Bayesian networks have been effective in cases involving complex decision-making to determine the outcome, such as games within an ITS (Conati, Gertner, et al., 2002; Hooshyar, Binti Ahmad, et al., 2018), but these are too sophisticated. Moreover, these are usually when the ITS has many facets, such as lessons, quizzes, and games, and the intelligence must control how the student moves through these phases. Embedded companions do not require this level of sophistication. A neural network provides a perfect compromise between the two extremes of complexity offered by linear regression and Bayesian networks; it supports variance among the possible inputs to provide an accurate outcome and allows for additional complexity to be added in future versions of the Companion.

The following sections provide the details of the Companion's Brain. First, the structure of the neural network will be described, including the data that enters and leaves it. Next, the possible messages available in the Brain will be presented. Then, the flow of data within the Brain will be depicted and the necessary transformations to the data. Lastly, the approach to the training of the neural network will be described.

3.1 Inputs and Outputs to the Neural Network

A multi-layer perceptron neural network, as shown in Figure 3.1, with a back propagation learning algorithm was used for the intelligence of the Companion. The backpropagation utilizes a sigmoid transfer function. The input layer contains 16 nodes containing students' data. There is one hidden layer with seven nodes, which reflects others' neural network implementation for hint generation using a single hidden layer (MacNish, 2002). The output layer consists of a single node corresponding to the feedback to give to the student.

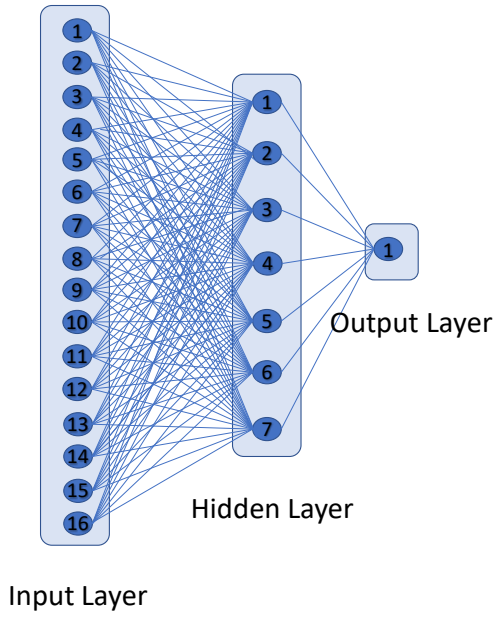


Figure 3.1. Structure of the Neural Network

The input to the neural network consists of 16 nodes with data extracted from the student's code and events in the IDE. Some of the nodes contain data about feedback given in this situation, which will be helpful for future training of the neural network. The specific input fields can be seen in Figure 3.2. After considering areas that often cause problems with recursion for software engineers and particularly students, such as a missing base case or StackOverFlowError, we selected inputs that can help to provide this information to the Companion.

1.	Action	Debug, Error, Help, Run, or Submit
2.	Assignment successfully completed	0, 1
3.	Comparator operator found	0, 1
4.	Keyword Double Found	0, 1
5.	Keyword Float Found	0, 1
6.	Keyword If Found	0, 1
7.	Keyword New Found	0, 1
8.	Keyword Return Found	0, 1
9.	Loop Found	0, 1
10.	Number of Comment Lines	0 to Infinity
11.	Total lines of code	0 to Infinity
12.	Cyclomatic Complexity	0 to Infinity
13.	Error Type	Most recent error message collected from compiler
14.	Message Code	Multiples of 10 from 10 to 280
15.	Message Given	Text for message matching message code
16.	Feedback on Message	Whole numbers, 0 to 4
17.	ID	Sequential number, automatically assigned for ID in database table
18.	Student ID	Participant identification number randomly assigned per IRB protocol
19.	Submission date and time	Current date and time stamp
20.	Course Name	"SER222", "CSE240", or "Both"
21.	Assignment Name	"Assignment.java"

Figure 3.2. Data Collected by the Plug-in and the Possible Values for Each Item: There were 16 values sent to the neural network, which are numbers 1-16. Numbers 17-21 are 5 other values that were collected but not sent to the neural network.

Now, we will provide a detailed explanation of each input and output field and the rationale for including this field in the neural network.

- **Action.** An *Action* is defined as an event that occurs while the student uses the plug-in. The plug-in detects which action has been executed and saves this data. Possible actions include *Debug*, *Error*, *Help*, *Run*, and *Submit*. *Debug* and *Run* actions occur when a student selects debug or run, respectively, within the plug-in. The plug-in detects the different ways that run or debug can be executed, such as right clicking, clicking the shortcut buttons, or selecting from the menu. Clicking the help button, which is visible within the view in the plug-in, triggers the *Help* action. The *Error* action is set when the student receives run-time error messages from the compiler. The Error state supersedes Run or Debug. For example, if the student runs or debugs the program and receives error messages, the state automatically is saved as *Error*, instead of as Run or Debug. The Error action also causes the exact error message from the compiler to be saved. *Submit* means that the student's code has passed all the test cases, so the assignment is completed successfully. The Submit action changes the assignment completed successfully field. The *Action* field can be used to interpret how the student is interacting with his or her code. For instance, students who utilize the debug function in the IDE may show a greater understanding of how to use the available tools while writing code. It also assists with determining what the student is doing, and if the student continues with many Error states, this could indicate a problem that the Companion should address with a message.
- **Assignment Completed Successfully.** This field indicates whether the assignment is finished correctly, and the test cases have been passed. It is important for the

Companion to display encouraging feedback when this happens, so this field helps detect this situation.

- **Comparator Operator Found.** A field representing whether the comparator operator is found represents whether the "==" is used in the code. Although it is not always required for all situations involving recursion, a common mistake for CS students is to use the assignment operator ("="), instead. This allows the Companion a chance to reiterate the importance of this operator.
- **Keyword Double and Keyword Float Found.** The presence of the keywords *double* and *float* assist the Companion in determining why a student's code might not be producing the correct result. While it is not necessary data for all recursion assignments, it is important for many situations. Thus, we opted to include this data in the neural network.
- **Keyword If Found.** Evaluating whether the student has used the keyword *if* is an essential component of recursion. This is important for the Companion to know whether the student has included this keyword because if the student omitted it, some review of the fundamentals of recursion is important.
- **Keyword New Found.** Whether the keyword *new* is found is not relevant for the assignment used in this study; however, in order to allow for greater versatility in the future, we opted to collect this field. It is common for students to try to call methods on objects that have not been created, so this allows a future version of the Companion to respond to that situation.

- **Keyword Return Found.** The keyword *return* must also be used in the recursive method. When *return* is not used, as is the case with *if*, the student is almost certainly missing the base case and likely misunderstands recursion.
- **Loop Found.** If the student used a loop, which is determined by whether the keywords *for*, *while*, or *do* are found in the code, this also represents a fundamental misunderstanding of recursion that the Companion will need to address.
- **Number of Comment Lines.** The number of comment lines are used as an input to evaluate whether the student is leaving a large amount of commented out, "dead" code. Although this is acceptable while working on an assignment, it should not be left there, so the Companion may give feedback on this situation.
- **Total Lines of Code.** This represents the lines of code for the recursive method of the student's program. Having this data can help in determining how much code the student has written before performing an action on it. For example, if the student has a large amount of code for a recursive method, this could indicate a problem, and, similarly, for an extremely small amount of code.
- **Cyclomatic Complexity.** The cyclomatic complexity can help in evaluating whether the student is using recursion correctly, for in a recursive method there exists a logical minimum cyclomatic complexity. It also follows that if the student's recursive method has a high cyclomatic complexity, this is also a warning that the student does not fully understand recursion.
- **Error Type.** This represents the type of error that has occurred in the student's code after running or debugging it. When an Error action is triggered in the plug-in, it collects the error message generated by the compiler. Not all errors are sent to

the neural network in their entirety; rather, in the data pre-processing phase they are mapped to a set of common errors for recursion.

- **Message Given, Message Code, and Message Feedback.** This contains the message that a student received, given the current inputs of the neural network, along with the corresponding numeric message code, indicating how helpful this message was for the student for this set of inputs. When generating feedback for a student while using the plug-in in real-time, these fields will not contain a value. However, they are important when training the model after collecting data from students, for they can help predict whether the same feedback should be delivered. After all, if unhelpful feedback was given previously for a similar set of inputs, the Companion should select different feedback, given this data.
- **Outputs from the Neural Network.** The neural network produces a single node as an output. This number will, in turn, be post-processed in order to correlate it with the exact feedback to display.
- **Extra Data Collected in Plug-in.** Some data was collected by the plug-in, but we chose not to send it to the neural network because it is needed more for administrative purposes for the study or the instructor, but is not pertinent for the Companion to have the data, as it will not influence the feedback the Companion gives to the student.

3.2 Messages in the Brain

Messages were developed for the Brain with the intention of providing students with feedback about how to proceed, especially with feedback about how to approach the

Table 3.1. Possible Feedback in the Brain: Feedback content focuses on two key areas: (1) Feedback about how to proceed with recursion, such as problem-solving methods and (2) Affective support. Feedback in the table is categorized by the type of feedback it provides. *PS* represents feedback about how to proceed or problem-solving feedback. *A* represents affective support. *O* represents feedback on another small coding issue.

	Situation	**Message**	**Type**	**Code**
1	No message available	Sorry, I don't have any messages for you right now. Try again later.	O	10
2	Problem with recursion identified	Think about breaking the problem into smaller parts. Ask yourself. What is the smallest problem I'm trying to solve?	PS	20
3	Debug initiated with no other problems identified	Good thinking - Using the debug tool is very helpful for problems with recursion.	A	30
4	Student passed at least one test case	Nice job! You got the right answer!	A	40
5	Cyclomatic complexity is too low	Think about changing the control flow of your code.	PS	50
6	Missing keyword: If	You're missing an important part for recursion. Think about using a keyword.	PS	60
7	Missing comparator operator	Try checking the value of something in your code to determine how the recursion unfolds.	PS	70
8	Too many comment lines detected	Remember to clean up your code! That's a lot of comment lines.	O	80
9	Student used a loop in the code	Uh-oh! I think your code using a feature that shouldn't be there in recursion. Review the concept of recursion.	PS	90
10	No return statement is present in code	I'm looking for an important keyword in your code, but I think it's missing.	PS	100
11	Used keyword "new" in the recursive method	I think I see a keyword in your code that's not really needed for this type of situation.	PS	110
12	Too many lines of code for the situation	That's a lot of code in this method! You might want to think about shortening it or double-checking what you're doing.	PS	120
13	StackOverFlowError received	Don't worry, StackOverflowError is a really common error with recursion. That's normal. Check that you reach your base case.	A	130
14	ArrayIndexOutOfBoundsException received	Software engineers get the ArrayIndexOutOfBoundsException quite a bit. It's a frustrating one. Using the debugger might help you.	A	140

	Situation	Message	Type	Code
15	Compilation error caused by missing syntax	I think you're missing some important syntax in your code.	O	150
16	No errors reported by the compiler and no other issues identified	Nice! No errors were found when you ran your code!	A	160
17	Student is doing well, but the method has too much code.	It looks like you're on the right track. That seems like a lot of code, though. It probably does not need to be that long.	A	170
18	Student has not successfully completed the assignment, but no issues are detected.	Nice work! Keep it up.	A	180
19	Student clicks the help button, but no issues are detected with the code.	I'm here to help you. Once you take an action with your code, I can help you with a hint.	A	190
20	Cyclomatic complexity is too high	That code looks pretty complex. Think about simplifying the logic a bit.	PS	200
21	Error is reported by the compiler, but the Companion is unable to identify a specific issue.	Hmmm, it looks like you should re-think something in your code.	O	210
22	ArithmeticException is reported	I think there's a problem with something with your math.	PS	220
23	Student is doing well, but not much code has been written yet.	Keep going. It looks like you'll need some more code.	O	230
24	A loop is present or some other syntax that is not needed.	I see some extra keywords that don't need to be used for this. Think about which ones you may not need.	PS	240
25	ArithmeticException is reported, but other features of the student's code are reasonable.	You're really close. Just check your mathematical operations.	A	250
26	Compilation error caused by missing syntax, but other features look good	Almost. I think you're missing some syntax, though.	A	260
27	The base case is missing.	This is common with recursion. Think about creating a base case.	A	270
28	The student has not written any code yet.	Try writing some code first, then I can help you.	O	280

problem. We envisioned situations that would frequently arise for students learning recursion. Such scenarios are comparable to an instructor who anticipates that students frequently have questions or confusion regarding certain problems with their code. Instructors likely often find themselves repeating similar explanations each time they teach the course, as these misunderstandings are extremely common for students with particular topics.

Thus, we applied this same type of thinking was applied to creating the messages for the Brain. We considered the most common challenges students face when learning recursion. After forming a list of several different situations, we selected a specific message for this situation that would assist the student with how to proceed. As a result, the focus was on giving the student feedback on how to solve the problem, rather than fixing syntax or a detailed explanation of the error message from the compiler. The messages in the Companion's repertoire are provided in Table 3.1. Since the output number from the neural network was scaled to correlate with the appropriate message, the codes for each message are also shown. The messages in the Brain comprise three categories: problem-solving or how to proceed, affective support, or other programming issues. The focus of the Companion's intelligence is on assisting the student with how to solve the problem or how to proceed with the assignment and also on providing affective support for the student. Consequently, the majority of the messages fall into these two main categories, as shown with the numerical breakdown in Figure 3.3. Some of the messages providing affective support could also be categorized as problem-solving feedback, but to emphasize the affective support, they are separated in the categorization here. A limited number of other general programming messages are also included to assist the student if he or she is stuck

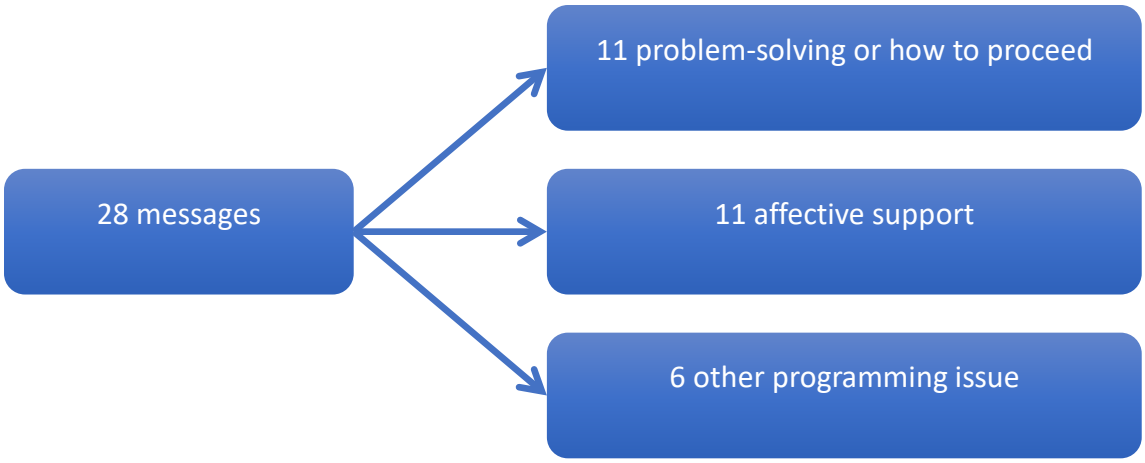


Figure 3.3. Categories of Messages in the Brain

on an issue involving syntax; nevertheless, the focus of this Companion is not on providing feedback about syntax, so a limited number of these messages were included.

3.3 Data Flow in the Brain

When the data initially enters the Brain of the Companion, it is not yet ready to enter the neural network. Some of the fields are initially represented as Strings from the plug-in and other similar situations. Moreover, the data must be normalized so that all numbers are between 0 and 1 before it can be processed by the neural network. As a result, several steps must occur before intelligent feedback can be displayed in the plug-in. The overview of how data enters and travels within the Companion's Brain is shown in Figure 3.4, which will now be explained in detail.

Data Pre-Processed. Since the data enters the Brain as different data types, the Brain must first produce corresponding numbers for each field that are all the same data type. The neural network expects all data as doubles, so conversions must occur as part of the data

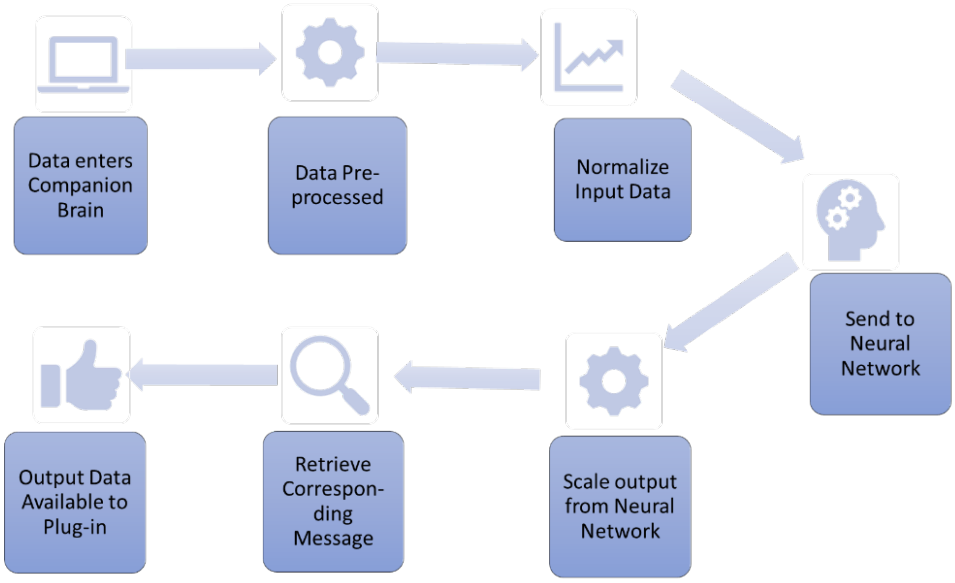


Figure 3.4. Data Flow in the Companion Brain

Table 3.2. Data Pre-Processing Needed after the Brain Receives Data from the Plug-in

	Input Neural Network	**Data Type**	**Pre-Processing Description**
1	Action	String	Mapped to corresponding number, saved as double
2	Assignment Successfully Completed	Integer	Converted to double
3	Comparator operator found	Integer	Converted to double
4	Keyword Double Found	Integer	Converted to double
5	Keyword Float Found	Integer	Converted to double
6	Keyword If Found	Integer	Converted to double
7	Keyword New Found	Integer	Converted to double
8	Keyword Return Found	Integer	Converted to double
9	Loop Found	Integer	Converted to double
10	Number of Comment Lines	Integer	Converted to double
11	Total lines of code	Integer	Converted to double
12	Cyclomatic Complexity	Integer	Converted to double
13	Error Type	String	Mapped to corresponding number, according to rules for errors messages included; saved as double
14	Message Code	Integer	Converted to double
15	Message Given	String	Mapped to corresponding number; saved as double
16	Feedback on Message	Integer	Converted to double

pre-processing. In some cases, as shown in Table 3.2, the conversion is simple; the integer is just converted to a double and saved as the appropriate variable. However, in other cases the conversion is not quite as straightforward. The *Action* field initially enters the brain as a String. Each possible value is mapped to a corresponding number and is subsequently saved as a double. The *Message Given* field functions in a similar way by looking up the matching value for the text of the message. Although the message code could serve this same purpose, we still chose to implement this look-up procedure for the message text to achieve redundancy in case there might be a mismatch between the message code and the message text.

Lastly, the *Error Type* comes into the Brain as a String representing the error message from the compiler. Since the compiler can produce a multitude of error messages for different situations, it could be possible to receive a detailed error message for a variety of situations. However, with the Companion's focus on assisting students with recursion assignments in Java, relevant errors were considered for this topic, which include:

- StackOverflowError
- ArrayIndexOutOfBoundsException
- NullPointerException
- ArithmeticException
- UnresolvedCompilationIssue

We include StackOverflowError because this is perhaps the most common error with students learning recursion. ArrayIndexOutOfBoundsException will, of course, only occur in some instances, but if the recursion utilizes an array as a data structure, this is important to detect. Although NullPointerException, ArithmeticException, and

UnresolvedCompilationIssue are not specific to recursion, since students encounter these relatively frequently, we also chose to include them. *Other* represents any other error message besides the aforementioned possibilities. *None* exists when no error messages are reported, and we want the Companion to be certain of this data, so this is indicated here.

Normalize Input Data. After the data is pre-processed, all values are represented as doubles; however, the neural network expects values as doubles between 0 and 1. Thus, the data must be normalized, that is, scaled to be between 0 and 1. Within the Neuroph library, several built-in normalization options exist (Neuroph Library, 2014). The *DecimalScaleNormalizer* was utilized for this data, as it provided the most consistent logical way of normalizing the data, and the other library options for normalization did not apply to this data set. The *DecimalScaleNormalizer* divided each input by a multiple of 10 to produce a result between 0 and 1.

Send to Neural Network. Once the data is normalized and pre-processed, it is ready for the neural network. The saved model for the neural network is stored on the server, so the Brain first retrieves the neural network. After this, the input data is loaded into the neural network, which computes a single number as an output.

Scale Output from Neural Network. Since the output from the neural network is also between 0 and 1, it must be scaled to the expected format for the corresponding message codes. All the output message codes are multiples of 10, so the output code from the neural network must be multiplied accordingly. For example, if the output from the neural network were 0.7, the message code would be 70. However, usually the outputs from the neural network are not such "tidy" numbers. For instance, if the output from the neural network were 0.7382, it should correlate to the message code of 70. Thus, the neural

network output must be multiplied by 100, then rounded to the nearest 10. One might wonder why this scaling was chosen for the output from the neural network. After all, the neural network should already produce the appropriate output. If we chose *not* to have this scaling, all codes for the messages would need to be between 0 and 1. Since we have 28 messages, this would only allow for a small numerical difference between the number for each message. As a result, numerically speaking, the messages would be very close together, which would require a high degree of precision from the neural network. Although we anticipated it being reasonably accurate, the difference between a message labeled 0.17 and a different message labeled 0.18 could be quite small. Consequently, we opted for message labels as multiples of 10 to reduce the required numerical precision. Thus, the output must be scaled from the neural network to map it to its corresponding message in a multiple of 10.

Retrieve Corresponding Message. The scaled output from the neural network serves as a key to look up the corresponding message. The message from the Brain for this situation is retrieved from the messages dictionary.

Output Data Available to Plug-in. Lastly, the data from the Brain is ready for the plug-in. The message and its code are saved, and the plug-in can retrieve them to display for the student in the Eclipse IDE.

3.4 Training of the Neural Network

Since no authentic student data existed before this study, the training of the neural network was completed using automatically generated data. We generated 100 data entries for the neural network using a JSON generator tool (Data Design Group, 2018). Constraints were

placed on the data generated to ensure that all data is within the possible ranges. The generated JSON data used for training is located in the online code repository for this project (Day, 2019).

Although all data was within the expected ranges, some slight imperfections were present in the generated training data that would not occur in an authentic setting with students due to the random nature of the generation. For example, in the generated data there were some instances where the *Action* field contained "Run," but an error message was still present in the data. This is not realistic because if the compiler reports an error message, the plug-in automatically classifies it as "Error" for the *Action*. Thus, it is impossible for the *Action* to be "Run" with an error message also reported.

Another case involved a high cyclomatic complexity reported, but the *if* keyword was not used, which is highly improbable. Even though other similar situations existed, they were the exception rather than the rule within the generated training data; in most cases, the data seemed realistic for an assignment from beginning CS students. Thus, the data was still used for training the neural network.

After the data was generated, we manually labeled the 100 input entries with a corresponding message code applicable for this situation. Human tutors and instructors can quickly scan a student's code and have an idea of what might be wrong with a recursive method. Similar logic applies to labeling the data for the training. In many cases, the type of problem with the code closely matches the situation description.

In some instances, multiple situations could apply, which would be comparable to what would happen with a human tutor. For example, if the data contains StackOverFlowError and the student also does not have the keyword *if*, the feedback could

either be regarding the error or it could be about the missing keyword, for either feedback would be valid. When a human tutor sees this type of case, typically one just quickly mentally chooses between the different possible options. We applied this same type of logic to data to which multiple messages applied. While a rule generator could be used to classify the data, when humans choose between options, there is still a small sense of randomness, so it seemed manually labeling the data would better simulate the human interaction. Once all generated data was labeled with a corresponding number for the appropriate feedback, the expected outputs were saved in the neural network to train it before use with the students.

3.5 Summary

This chapter provided an overview of the Brain of the Companion, including the details of the neural network. We discussed the decisions regarding the inputs and outputs of the neural network with details regarding each input to the neural network and possible messages in the Brain. We also provided an overview of the flow of data within the Brain to arrive at the appropriate feedback to display to the student. In this chapter, we concentrated on the intelligence of the Companion. The next chapter focuses on the technical details surrounding the Companion's Brain.

CHAPTER 4

IMPLEMENTATION OF THE TUTORING COMPANION BRAIN

This research builds upon prior work (Penumala and Gonzalez-Sanchez, 2018) available as open-source software (Penumala, 2017). The existing software facilitated the sending of information between the student and the instructor using the Eclipse plug-in. In order to achieve the goals of this project, the existing project required modifications. The core functionality of the improved software for this project involves collecting data from the student's code, sending the data, and displaying the Companion's feedback. On the surface, this functionality seems simple; however, in reality there are numerous technical decisions that must be made in order to achieve this functionality.

This chapter provides an overview of the technical implementation of the Companion's Brain. First, the user interface for the Companion plug-in will be discussed. Next, the software architecture will be described, followed by the details of the technical design and implementation of the plug-in and the Brain. Then, the libraries used for the project are described and the motivation for these selections. Lastly, the approach to software testing will be described.

4.1 User Interface

Views are an existing component in Eclipse and are established when Eclipse is launched. Eclipse comes with pre-installed views, such as the console at the bottom of the screen. Plug-ins can add views to the IDE, which is how the Companion displays feedback. This section describes the plug-in's modifications to the User Interface in Eclipse.

After the plug-in is installed, when Eclipse launches, two additional views are added to the IDE: the Help View and the Assignment View, as shown in Figure 4.1. The arrow and help icons are also added to this view. When the student clicks the yellow arrow in the Assignment View, this pulls in the assignment from the server, which displays the name of the assignment and is depicted in Figure 4.2. When the student double-clicks the name of the assignment, it opens inside the package explorer in Eclipse, as shown in Figure 4.3, and the student can open the project in the typical way. Once the student performs an action in the IDE, such as run, debug, or help, the plug-in displays the feedback from the Brain, as shown in Figure 4.4. Choices are also shown to the student to collect data from the student about how helpful the Companion's feedback is for the current situation. After

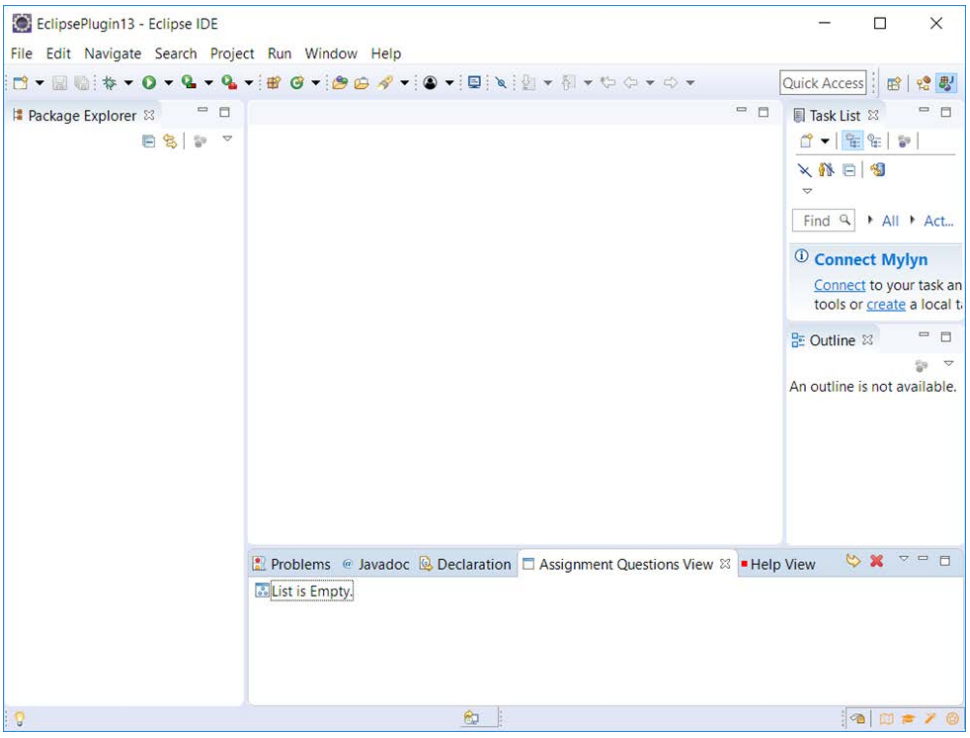


Figure 4.1. Assignment View and Help View are added to the plug-in when Eclipse launches.

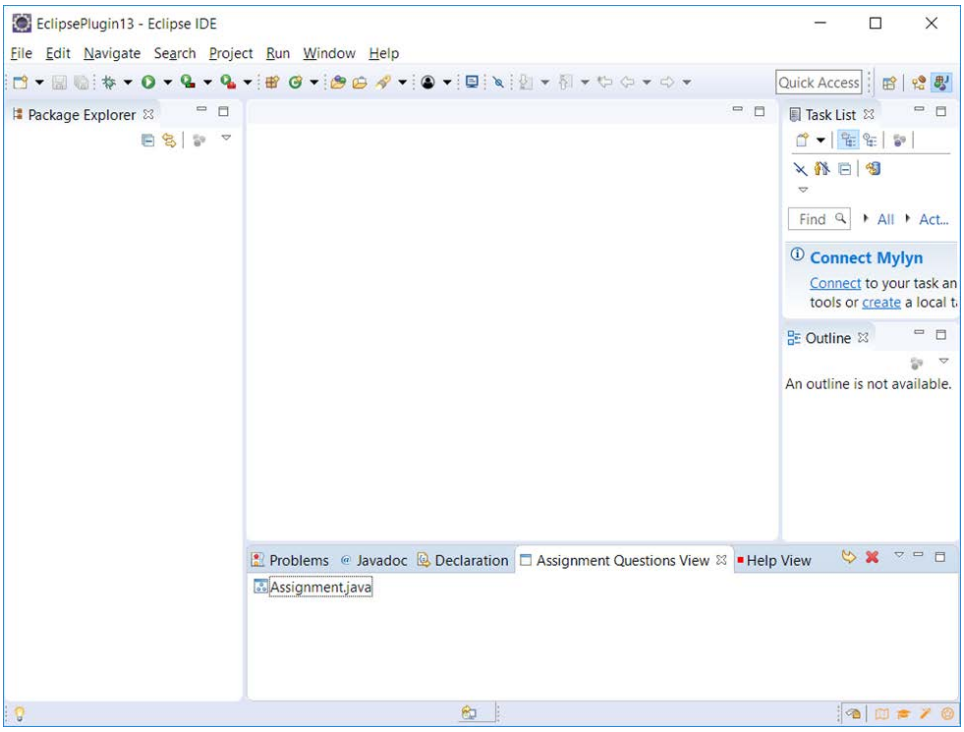


Figure 4.2. Assignment is pulled from the server, and the name is displayed in the Assignment View.

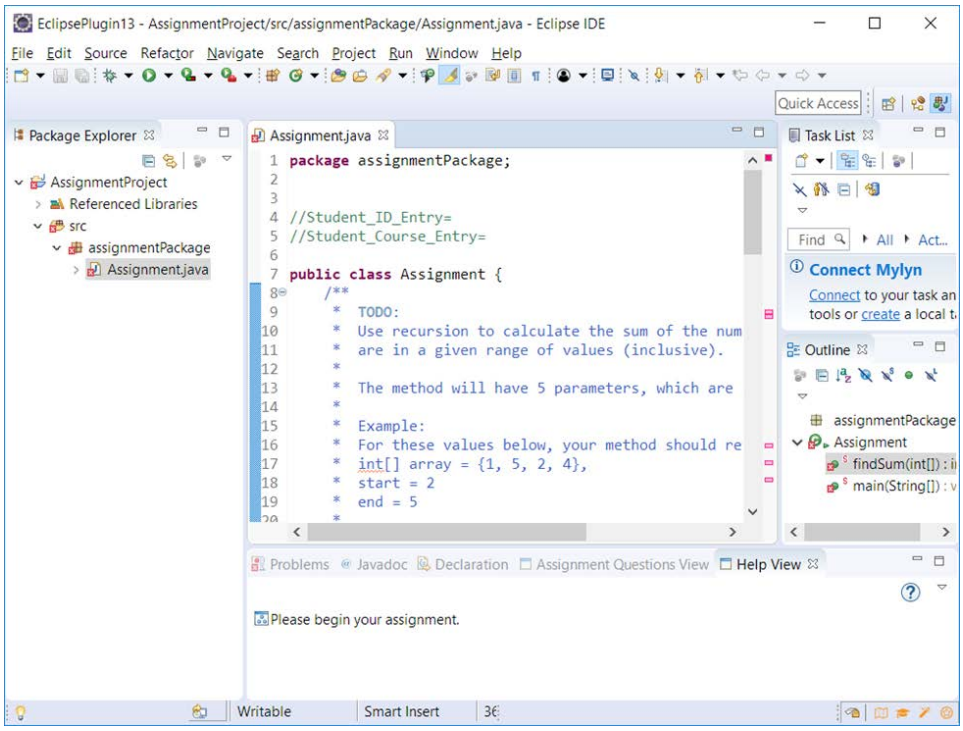


Figure 4.3. Assignment is opened inside the package structure in Eclipse.

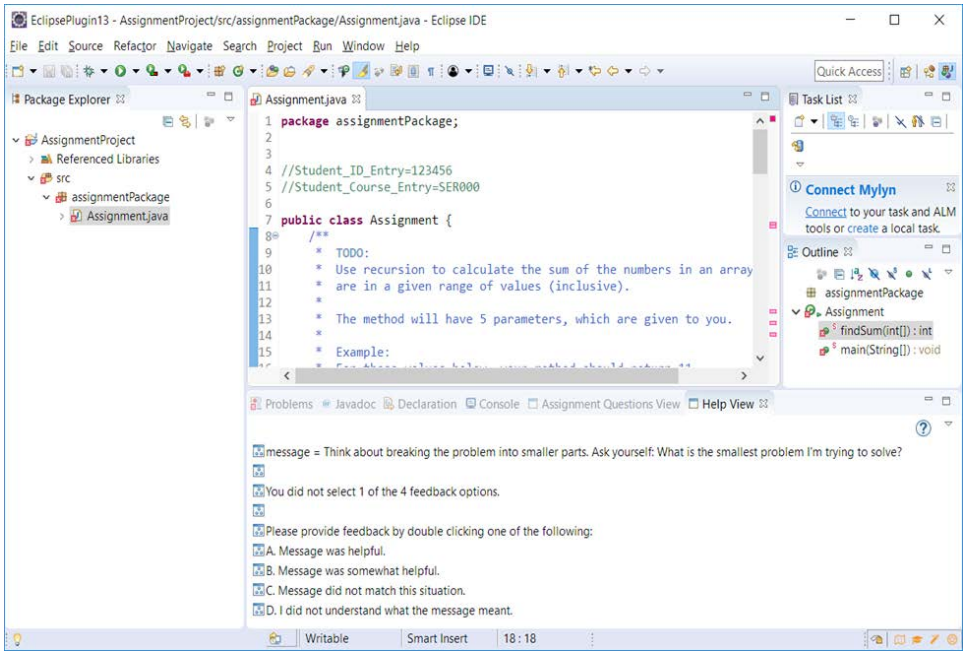


Figure 4.4. The plug-in displays feedback to the student after the student performs an action in the IDE.

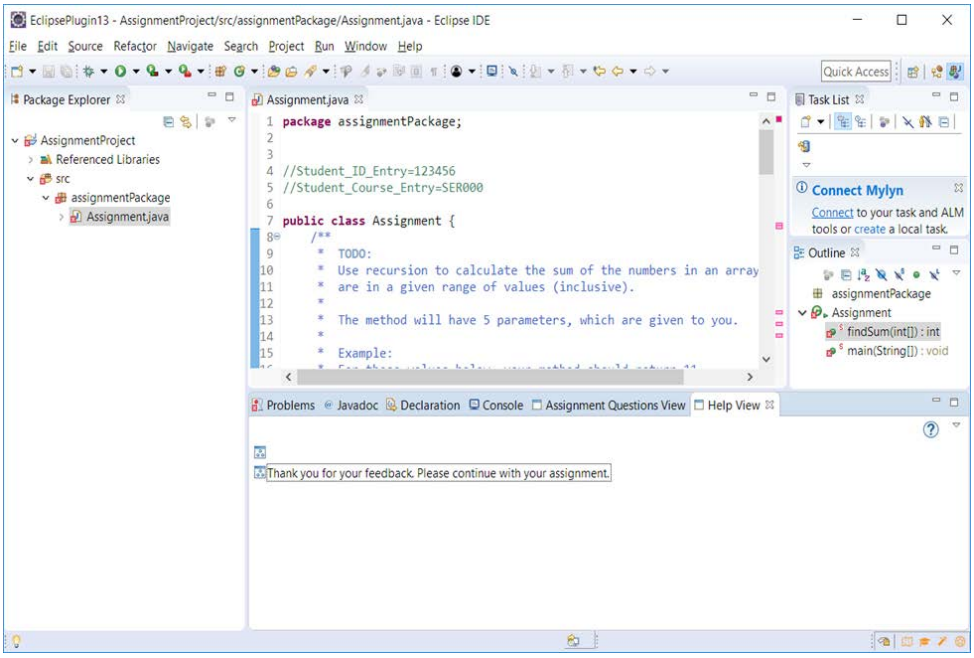


Figure 4.5. The plug-in acknowledges the student's response rating the Companion's feedback.

the student double-clicks the response about the feedback, the plug-in displays an acknowledgement of the response, which is illustrated in Figure 4.5.

4.2 Architecture

Now, the overall structure will be described for the Companion. The architecture is depicted in Figure 4.6. The Companion plug-in is added on top of the existing functionality in the Eclipse platform. The Brain contains all the logic and decision-making capabilities of the Companion, whereas the plug-in is responsible for the data collection from the student's code.

Each time the plug-in collects data when an action is detected in the IDE, it sends it to two places: a MySQL database and the Brain. The 21 pieces of data described in chapter three are sent to both these locations. Although all of these are saved in the database, the Brain does not use all 21 items, so only the 16 inputs to the neural network are retained in the Brain. A new MySQL database on a server hosted by Amazon Web Services was created for this project to separate the data from the existing data in the previous research.

The Companion's Brain is contained inside the plug-in, so when the plug-in is installed, the Brain will exist locally. The trained model for the neural network is stored in the cloud to achieve separation of concerns. This also makes it easy to update the neural network in the future. Someone can train a different model and replace the current neural network without the plug-in or the Brain requiring any changes.

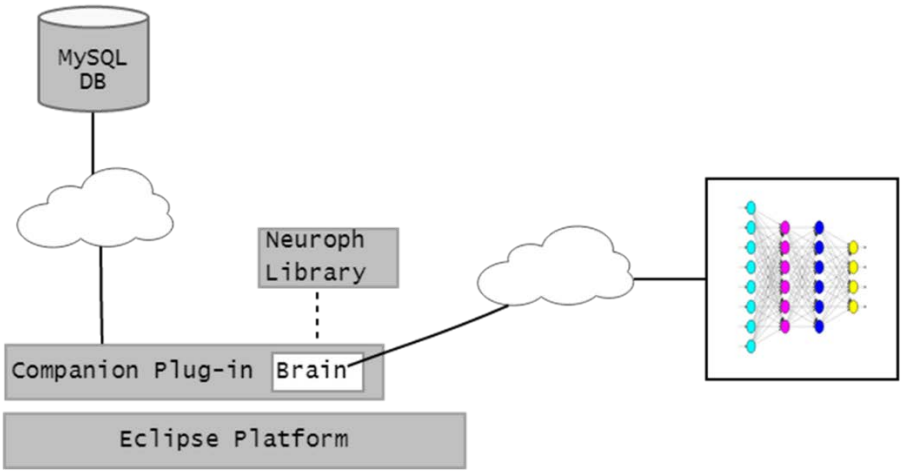


Figure 4.6. Companion Architecture

4.3 Eclipse Plug-in

The Eclipse platform is composed of many plug-ins. When developing plug-ins, it is possible to connect to existing components within Eclipse. The workspace and the view already exist within the Eclipse Platform. When the IDE is launched, the views and the workspace are created, and any existing plug-ins are initialized.

Since the Companion plug-in collects data from the student's code, it connects with the workspace and retrieves required information, including the code itself, as well as tracking when the student initiates a run or debug action. In order to display the help button, the assignment, and the Companion's feedback, it also connects to the views in the workbench. The required connections between the Companion plug-in and existing Eclipse components are shown in Figure 4.7. The logic for creating and displaying the assignment and feedback are inside the plug-in, so these views are directly tied to the life cycle of the plug-in.

4.4 Design

This section describes the design of the Companion plug-in. First, an overview will be provided of the design of the plug-in. Then, details of the Brain's design will be given with a description of the details of each class in the Brain.

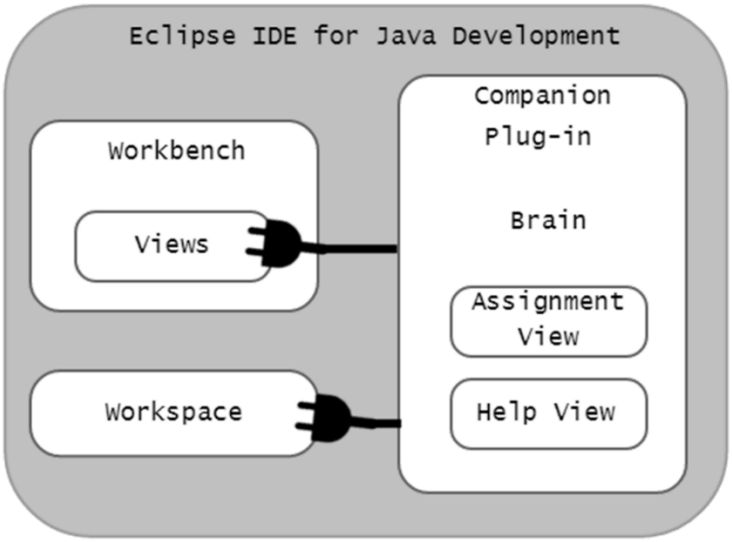


Figure 4.7. Eclipse Plug-in Architecture: The Companion plug-in connects to already existing components within the Eclipse IDE.

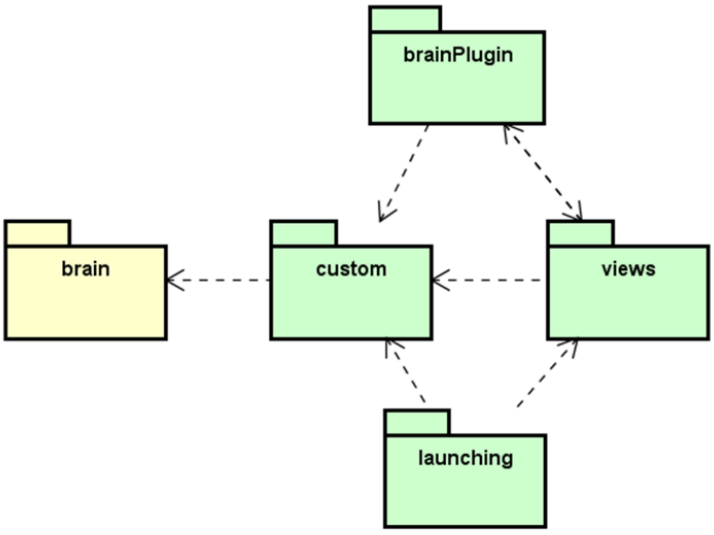


Figure 4.8. Package Diagram for the Companion Plug-in

The plug-in is composed of five packages, as shown in the package diagram in Figure 4.8. The BrainPlugin package contains classes that control the lifecycle of the plug-in and that directly interact with the workbench. The package entitled “custom” contains most of the logic for the Companion, such as handling listeners and sending data to the server. The launching package connects to the console to collect error messages. The views package, as its name indicates, contains the two views added to Eclipse. Data is sent between the plug-in and the views, and the plug-in also tracks various actions that occur in these views, such as clicking the help button or evaluating the feedback. After the student gives feedback on a message in the Help View, the plug-in updates the current data row stored in the database with the feedback received from the Brain and the student’s rating of the feedback. Further details of the plug-in are described in prior work (Penumala and Gonzalez-Sanchez, 2018).

The design of the entire Companion plug-in is depicted in Figure 4.9. The Brain, as a self-contained unit, can achieve its full functionality without any dependencies on the plug-in. The connection between the plug-in and the Brain is established by one simple connection: the plug-in creates a Brain object and calls its method to retrieve the feedback from the Brain. Thus, the intelligence of the Companion can be easily updated without impacting the plug-in.

Now, we will provide a discussion of each class within the Brain. The `TutorBrain` class controls the overall structure of the Brain. This class has an overloaded constructor. One of the constructors handles training the neural network. Its parameter is the training data as a JSONArray. The other constructor allows the plug-in to

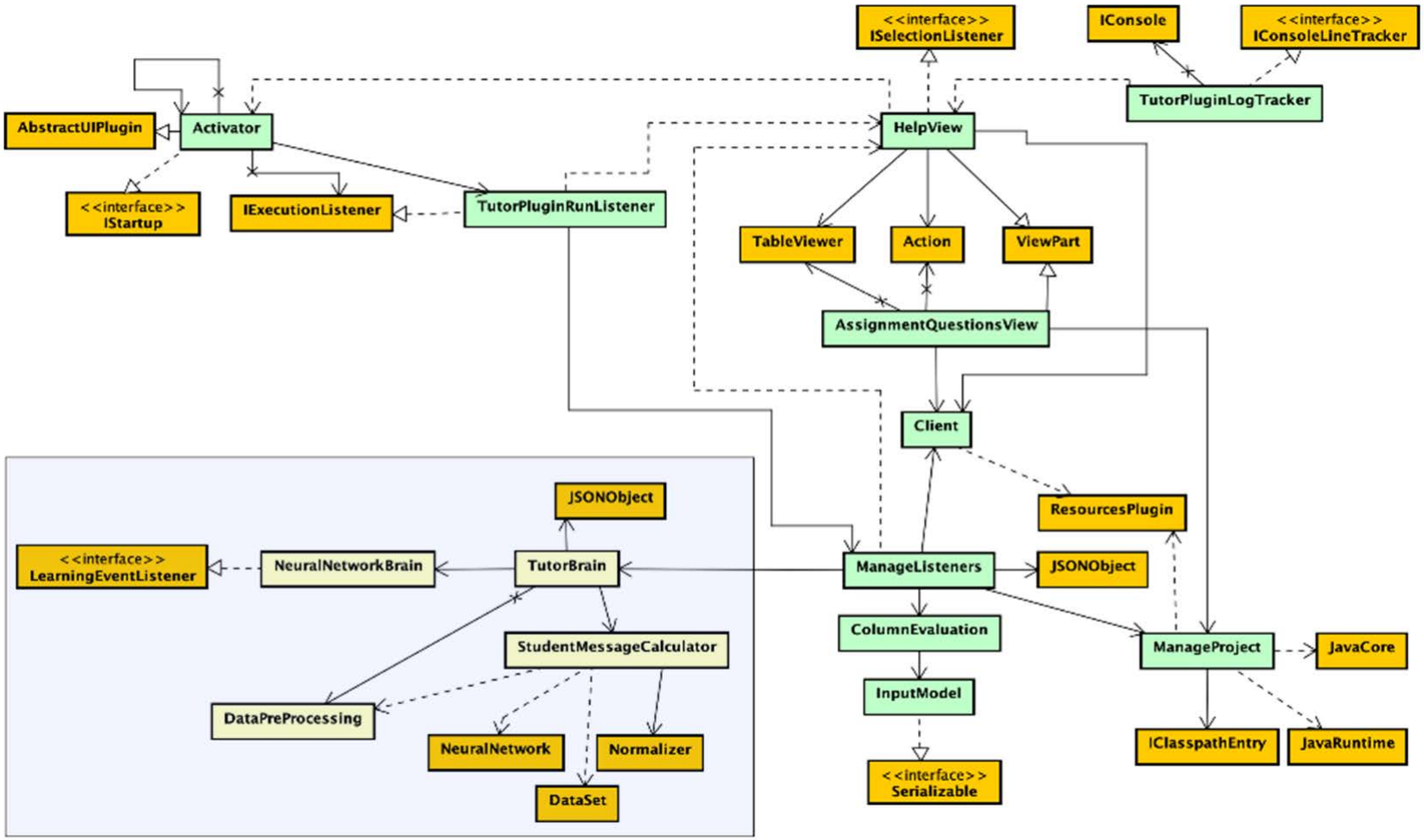


Figure 4.9. Class Diagram of the Companion Plug-in: Classes for the Brain are inside the gray box. Orange indicates classes from libraries. Green indicates existing classes in the plug-in. Tan indicates classes implemented for the Brain

get needed information from the Brain. The plug-in calls `TutorBrain`'s constructor with no parameters. Another method exists for the plug-in to retrieve the Brain's feedback, which expects the data for that situation as a JSONObject. This class also uses the Factory pattern and creates the necessary objects for use within the Brain.

The `NeuralNetworkBrain` class controls the training of the neural network. When a `TutorBrain` object is created with one parameter for the purpose of passing the training data, the data is, in turn, passed to this class. The structure of the neural network is set within this class, which is done primarily through the use of the Neuroph library. `NeuralNetworkBrain` creates the multilayer perceptron and sets the options, such as the transfer function, number of nodes in the hidden layer, learning rate, maximum error, and the number of iterations. It has a method to save the trained neural network locally, as well as one to normalize the data.

The `DataPreProcessing` class preprocesses all data for the Brain. Data initially enters the Brain as either a JSONArray or JSONObject, which is passed as a parameter to this class to pre-process the data. `DataPreProcessing` takes all input fields and produces outputs of corresponding doubles for the neural network represented as a two-dimensional array of doubles, which is the expected format for the neural network.

Lastly, the `StudentMessageCalculator` class in the Brain controls the feedback to display to the student and steps that must occur with the data received. First, it calls the method from `DataPreProcessing` to preprocess the data and normalize the data. Then, it retrieves the trained neural network stored in the cloud and sends the normalized data to it. Once it has the output from the neural network, which is a numerical code between 0 and 1, it uses a formula to calculate the corresponding code for the message.

Since messages are labeled as multiples of 10 between 10 and 280, the output from the neural network must be scaled to match this. Thus, the output from the neural network was scaled with the method in Figure 4.10. This method multiplies the output from the neural network by 100, then rounds it to the nearest 10. If the output is very small, then it is multiplied by 1000.

```
public long calculateMessageCode() {
      double nnOutputRaw = this.getNnOutput()[0];
      double nnOutput = 0;
      // Output is so small that special calculations need to be made
      if (nnOutputRaw < 0.1) {
            nnOutput = nnOutputRaw * 1000;
      }
      else {
        nnOutput = this.getNnOutput()[0] * 100;
        // Used for conversion to expected error code
      }
      long messageCode = Math.round(nnOutput / 10.0) * 10;
      return messageCode;
}
```

Figure 4.10. Code Used in the Formula to Scale the Output from the Neural Network

Several challenges were encountered relating to where to store the trained neural network created in `NeuralNetworkBrain.` The Brain exists as a .jar file within the plug-in, which is itself a .jar file. Thus, there is a .jar file within a .jar file, which is installed in Eclipse as a plug-in. Since a method from the Neuroph library is called within the plug-in to load the neural network file, this requires having the trained model stored within the local package structure. However, the challenge arises because the package structure for the Brain is not preserved when the Brain.jar is contained within the plug-in .jar file, causing the Brain not to find the neural network when it attempts to load the neural network.

As a result, we chose to store the neural network on a server to avoid the file structure issue with exporting the Brain as a .jar file. Moreover, the trained neural network model is external to the Brain, so the neural network can also be changed without impacting the functionality of the Brain. Each time the plug-in requests feedback, the trained neural network is loaded from the server. The ideal situation would allow the neural network to be stored locally after loading it, but the Neuroph library only supports the saving of a neural network file after training, which means it can only be saved locally immediately after training.

4.5 Libraries

Several external libraries were used in this project in order to minimize the amount of brand-new code to write. In some ways, writing code from scratch might have been easier because it would reduce the amount of time spent understanding existing code and how to use it. However, since we wanted to have a professional-level product, it seems that libraries are appropriate. Furthermore, this will make it easier to extend this project at a later time.

The libraries used in this project are shown in Figure 4.11. These libraries were selected because they are professional level tools that enabled us to incorporate existing functionality, such as JSON. Various libraries exist for implementing neural networks in Java with varying levels of features and sophistication (Heaton, 2015; Dogaru and Dogaru, 2013). Given the wide range of quality and extensibility of the libraries, some have even opted to create their own neural network libraries for Java (Dogaru and Dogaru, 2013).

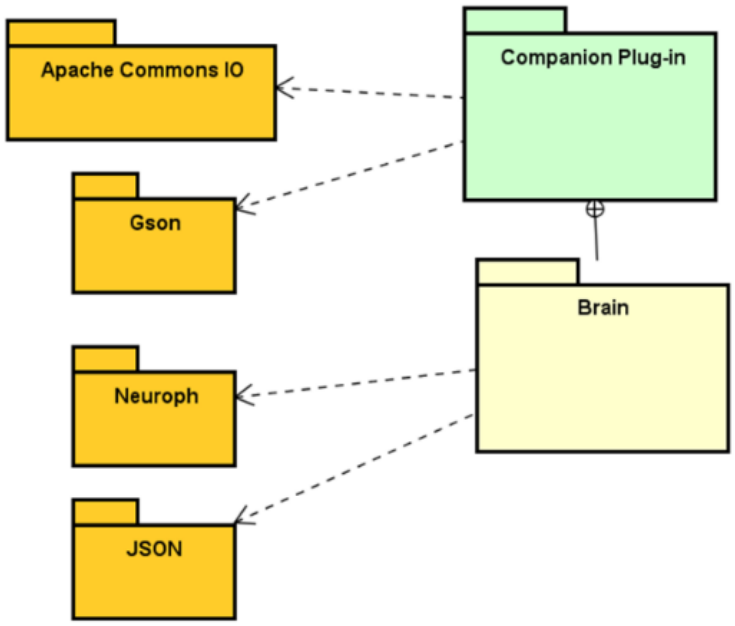


Figure 4.11. Libraries Used for the Companion Plug-in

We chose the Neuroph library for this project, which is an open-source Java library (SourceForge, n.d.). One of the goals of Neuroph is to simplify the implementation of neural networks so that users can start using them quickly. Even with its simplicity, it still offers a range of features to facilitate a fully customizable neural network. Additionally, since it is open source, many other open-source projects utilize it for more sophisticated use cases, ranging from image classification to cancer identification and many other projects (Neuroph Projects, 2019). Thus, it seemed an appropriate selection for the Companion because the library could be learned quickly, yet it would also allow for more sophistication at another time. Even though the tools from the libraries brought a great deal of functionality to the project, they also presented challenges, such as all of them functioning together without conflicts.

4.6 Testing Process

The agile development model best characterizes the software development process for the Companion. In a sense, we were both the customer and the developers for the project, so the process was not typical; however, we used the agile method because this would make the project easier to change. We began with a set of requirements and started to implement them. Along the way, the requirements began to change due to research being conducted in parallel, as well as unexpected technical hurdles. When this happened, the development effort was adjusted to match the revised requirements. Although formal sprints were not conducted, the process informally reflected sprints with development efforts focused on achieving functionality for various aspects of the system at different points along the way.

Prior to the release of the software, the system underwent several phases of testing. Since there were no dedicated testers in this project, the developers performed most of the testing. Although more testing could have been beneficial, since most of the testing was completing by a single developer, this posed a time constraint on the project. As a result, the tests were not as extensive as would have been ideal; nevertheless, the testing process exposed the major defects before the release of the software. Since we followed agile development methods, testing was fairly informal and was conducted throughout the entire development process, rather than just at the end.

Component Testing. We used component testing throughout the development process as we deemed various parts of the software complete. Once a portion of the software appeared to be complete, we connected it with a relevant existing part to determine whether the functionality worked. These tests typically involved calling methods in the main method and checking if the values were changed accordingly. This type of test

was a significant portion of the testing for the Brain in particular, as it involved much data manipulation with data being passed between multiple classes and methods. When the values were not updated as expected, the issues were documented to revisit at a later time.

Integration Testing. The development process was punctuated by integration testing at several key points. First, we integrated the server code and the database functionality into the Brain and the plug-in separately, allowing us to test whether the calls updated the data as expected in the database.

Next, we integrated the Brain and plug-in while still in development. This phase of testing led to errors that involved significant changes to both the plug-in and the Brain. The design had been determined, but until the two were actually integrated many of the challenges were unknown. In fact, this phase of testing exposed the most significant challenges of the software development. A couple of the major issues uncovered are listed in Table 4.1, as well as the chosen solution. As each part was changed and, at times, reworked, this resulted in further integration testing needed once all parts were integrated yet again. Thus, this phase of testing was a cycle between testing and more development time until everything worked together.

Table 4.1. Challenges and Resolutions during Integration Testing

Challenge	**Description of Issue**	**Solution**
Including the Brain as a jar within the plug-in	Libraries used by the Brain not included when generating jar for the Brain	Use a Maven packaging structure for the Brain
No messages displayed in plug-in	Plug-in waited to display message until after feedback was provided.	Updated when the View was refreshed

Alpha Testing. When the software seemed complete, Alpha Testing was conducted with a small group of users. These users were two students from the Master of Science in Software Engineering program, so they were high-ability users who would be more familiar with how to use the IDE and how to complete the assignment than the typical user. The goal was to determine whether multiple users could use the plug-in and, consequently, access the database simultaneously, along with whether the plug-in would work on a non-development computer. Neither user received feedback from the Companion. The assignment was loaded successfully, but feedback was not displayed because the neural network could not be located due to it not being found in the package structure. After this issue was resolved, Alpha Testing was repeated with another high-ability user. This time the Companion also failed to deliver feedback, but the problem was an error reported within the plug-in. Once this issue was also fixed, the test was repeated with the same user, which led to the expected behavior of the Companion. Then, the software was ready for release.

4.7 Summary

In this chapter, we described the student's interaction with the Companion in the Eclipse IDE. We also provided the details of the architecture, design, and technical implementation of the Companion plug-in and Brain. Lastly, we discussed the process of how the software was developed, especially the details around its testing. All of this information is important to understand the Companion fully, and the testing, to some extent, underscores that the software works. However, there is no substitute for authentic users interacting with the software. Thus, it is important to test it with real users, which we will describe in the following chapter about the case studies.

CHAPTER 5

CASE STUDIES

In order to test the effectiveness of the Companion, we conducted a research study with 28 students. Each student completed an assignment and solved a recursion programming problem while receiving feedback from the Companion. This allowed us to test our software with the intended user, evaluate the effectiveness of the feedback generated using the neural network, and measure students' perceptions of the usefulness of the Companion. All materials used in the study were approved by the IRB for research involving human subjects. IRB details can be found in Appendices A and B. This chapter describes characteristics of participants in the study and the protocol followed for the research study.

5.1 Participants

To recruit participants, courses were identified at Arizona State University in the Spring 2019 semester in which students would possess similar characteristics to the intended user for the software. Since the Companion is designed for beginning CS students in a Java programming course and would involve recursion, the intention was to identify lower-division courses in which students learn to program in Java and have already had some experience with recursion. Thus, the effectiveness of the software can be evaluated by testing with a subset of students representative of the target audience for the Companion. Two courses were identified that possess the aforementioned criteria.

Data Structures and Algorithms. The first course, Data Structures and Algorithms, is a required 200-level course offered to Software Engineering majors. Thus,

the course contains primarily beginning CS students, although there may also be a small number of graduate students required to take it as a prerequisite. In this course, students learn about elementary data structures, including when to use them with the data at hand, and Java is used as the programming language in the course (Acuña, 2018). Recursion is also covered through a required assignment involving a recursive implementation of a function. Thus, by mid-semester in the course, which is when the study was conducted, students have a strong understanding of the array data structure and a working understanding of recursion.

Introduction to Programming Languages. The other course, Introduction to Programming Languages, is also a required 200-level course offered for both Software Engineering and CS majors. As in the case of the other course, it primarily contains beginning CS students with a small number of graduate students. As prerequisite knowledge, this course assumes that students are proficient in a high-level language such as Java and that they have a working understanding of basic data structures like arrays (Gonzalez-Sanchez, 2018). This course provides an overview of different programming language paradigms, such as functional and procedural languages, as well as programming assignments in various languages, including a high-level review of Java and one assignment in Java (Gonzalez-Sanchez, 2018). Thus, most of the student population in both classes possesses the desired traits for students using the plug-in: first, they are students in the first two years of their CS program in a course that involves programming in Java, second, they have some exposure to recursion, and third, they understand arrays. There were 28 students who participated in the study over the course of 3 days. The composition of the study group can be seen in Figures 5.1 and 5.2.

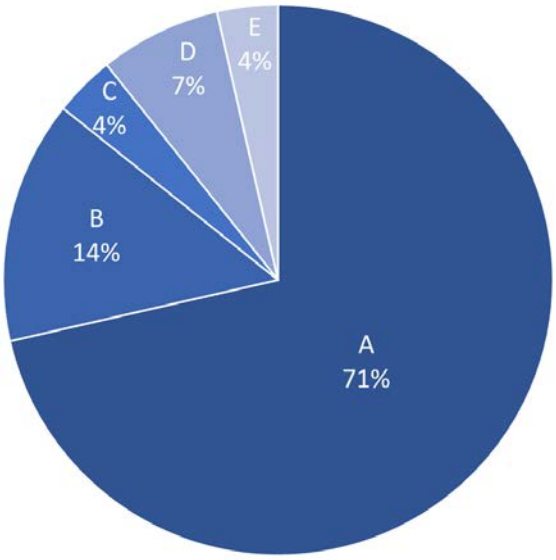


Figure 5.1. Student Population:
A. Second-year undergraduate B. Third-year undergraduate
C. Fourth-year undergraduate D. Graduate Student E. Other

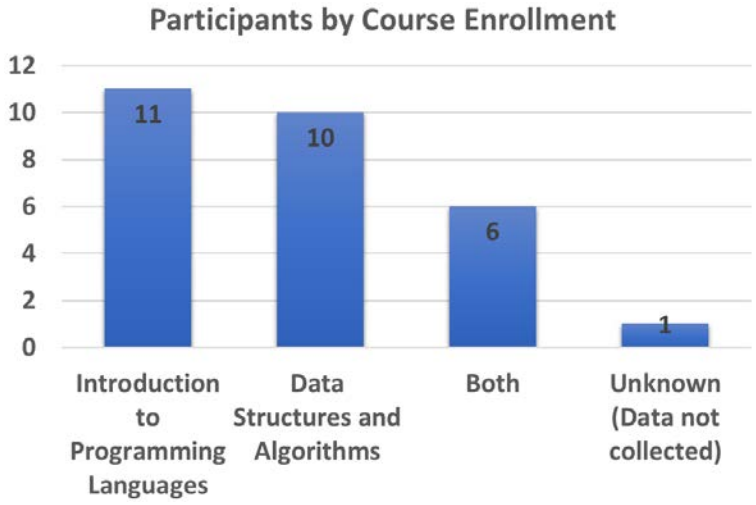


Figure 5.2. Participants by Course Enrollment

Students were recruited through an announcement on the course site describing the study and what participants would be asked to do in the study. In the Spring 2019 semester, there were 51 students enrolled in the Data Structures and Algorithms course and 36 students enrolled in Introduction to Programming Languages. Thus, 87 students total were invited to participate in the study. Students received 3% extra credit on their final course grade in exchange for study participation. Participants were instructed to send an e-mail message to the author to sign up for the study. After sending the e-mail message, students scheduled an appointment to come to a classroom where they would complete the assignment for the study.

5.2 Study Protocol

Students were given instructions for installing the needed version of the Eclipse IDE and an up-to-date Java version. While many students already use Eclipse, most needed to install the Enterprise version, which supports sending data to a server from the IDE. We gave them download and installation instructions for updating their Java version to Java 1.8 and the Eclipse version to Eclipse 2018-12 Enterprise edition. These instructions are located in Appendix C.

Although the goal is to eventually have students use the plug-in to complete a programming assignment at home, future work involves improving the usability of the plug-in, so it was expected that students would require some assistance with plug-in installation. Thus, we opted to have them come to a classroom to complete the assignment, instead of completing it at home. This allowed us to help them troubleshoot possible installation issues, as well as supervise their use of the plug-in to ensure that they used it

correctly. By confirming that their system was configured correctly and that they understood how to use the plug-in, this helped to achieve greater consistency in the data, as it would reduce the possibility of data discrepancies due to factors external to the Companion's intelligence, such as installation and system configuration issues.

When participants arrived for the study, we gave them a series of steps to complete as part of the study preparation. We also gave participants a flash drive containing two files: the directions with hyperlinks for the documents and surveys, as well as the plug-in as a .jar file. Unique participant identification numbers were created using a random number generator. This allowed us to identify all the data that came from the same person without compromising his or her privacy. Participants entered their participant identification number on all documents collected in the study.

First, participants reviewed the consent form, approved by the IRB, which is provided in Appendix D. After consenting to participate, they took a pre-assignment survey. The purpose of the survey was to collect basic demographical data from students and to gauge their background knowledge with the assumed prior knowledge for the study. For example, students were asked to rate their familiarity with recursion and using Eclipse as their IDE. The full survey is located in Appendix E. Additionally, another goal for this survey was to collect data about how easily they are able to get help with programming assignments, as well as what kind of struggles they have with programming assignments. The pre-assignment survey data was collected through Google Forms.

After this, we asked the students to install the Eclipse plug-in. They copied the .jar file for the plug-in on the flash drive to their local machine. Each participant received plug-in installation directions, which are located in Appendix F. We assisted participants as

needed with plug-in installation issues. Most issues with installing the plug-in were related to system configuration or clarifying the directions, given that there were many steps involved in the installation.

Once the plug-in was installed, participants opened Eclipse and could begin completing the programming assignment. In most cases, we double-checked that the plug-in was working correctly as participants were in the early stages of the assignment; however, in a few instances there were as many as 10 students in the room at once, so we were unable to check a few participants' set-up.

When students first use the plug-in, the two tabs for the views appear at the bottom of the IDE. This is evidence that the plug-in installation was successful. Prior to the start of the study, we posted the programming assignment on the server, so when students first use the plug-in, they must pull it from the server by clicking the refresh button in the Assignment View. This retrieves the assignment, which they can then open and use as a typical Java class in Eclipse.

The assignment, which is located in Appendix G, asked students to calculate the sum of the numbers in an array that fall within a given range of numbers. For example, in an array containing the numbers 1, 5, 2, and 4 with a starting range of 2 and an ending range of 5, the method should return the sum as 11. The parameters of the method were provided to students, as well as three test cases in the main method to demonstrate how to call the method. The method was set up to use an accumulator in order to calculate the result. Students were instructed to provide a recursive implementation of the method and were not permitted to use "helper methods." Once the Companion detected that the success message

was printed to the console, it received a different action notification so that the data is saved as “Success” in the server.

Students included their participant identification number and course number at the top of the page. This information was collected with the data sent to the server in order to facilitate better data analysis, as shown in Figure 5.3. Each time the student executed an action in the IDE, the Companion delivered feedback, and data was sent to the server.

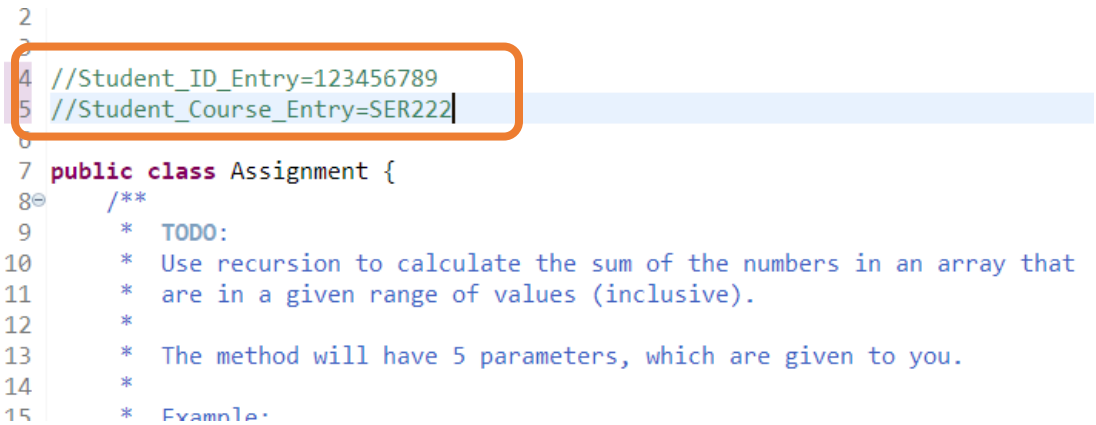


Figure 5.3. Students provided participant identification number and course number at the top of the assignment.

After each time the Companion displayed feedback, participants were asked to rate how helpful it was for the situation in the assignment. Although the tool did not require that they rate each feedback message, we frequently reminded them of this throughout the study. If the student did not provide feedback before proceeding, the data was still saved for this situation, but no feedback evaluation was included.

Most participants worked on the assignment for approximately 30 minutes. We told students to spend about 30 minutes working on the assignment, and we instructed them that after this time period had passed, they could decide whether to keep working or

whether they wanted to stop. This was to be respectful to the time frame that participants were told to allot in their schedule for participating in the study. The amount of time students spent on the assignment varied; some finished before 30 minutes, others did not finish the assignment within the allotted time, and others spent longer than 30 minutes.

Lastly, participants completed a post-assignment survey, which can be found in Appendix H, in which they were asked questions about their experience with the Companion. They were asked to rate how much they liked working with the Companion and how helpful the feedback was, as well as their opinion on different issues they noticed with the messages. This data was also collected through a Google Form.

The design of the post-assignment survey followed the principles of educational research for evaluating the effectiveness of teaching methods. Prior to my current degree program, I earned a Master of Arts in Education and took graduate courses on study design for measuring educational effectiveness. Thus, I have formal graduate-level training in evaluating this type of research. This helps to provide validity for the data collected through the surveys. After this survey, participants had completed the study, and they uninstalled the plug-in before leaving.

5.3 Summary

In this chapter, we provided a summary of the case study conducted with students test how well the Companion plug-in and Brain worked. We described the participants in the study from the two courses at Arizona State University. We also explained the procedures that were followed while conducting the study. All of these procedures were approved by the IRB. In the following chapter, we will analyze the data gleaned from this study.

CHAPTER 6

RESULTS

After the conclusion of the study, the data was analyzed through a combination of methods. Most of the data from the surveys could be analyzed by tallying the number of responses in each category. When responses were open-ended, the data was examined for trends. As categories emerged, these open-ended responses were classified by the category into which they fell. Some themes that emerged from the data will be discussed in this chapter.

This chapter discusses the results of the study. First, data that was gathered from the pre-assignment survey will be provided regarding the students' background. Next, the overall perceptions of the Companion will be explored, and the Companion's feedback will be examined. After that, the data collected from the students' code will be discussed. Finally, the software tool itself will be evaluated.

6.1 Students' Background

As part of the pre-assignment survey, students were asked about their background with recursion and Eclipse. They also answered questions about how often they need help with programming assignments and how they approach this. This section describes trends observed in this data.

Students had a moderate amount of exposure to recursion before using the Companion. Before using the Companion, all students had some exposure to recursion in other courses. In fact, more than half of the students already had two prior courses in which they studied recursion. Furthermore, the majority of students reported feeling that they

understand recursion, and over half reported “a lot” of prior experience with recursion in Java.

Many students understand how to use Eclipse, but the response is mixed whether they prefer to use it. We developed a Companion for Eclipse due to the belief that large numbers of students use Eclipse. Many students report that they understand how to use Eclipse, although nearly 30% of students surveyed feel they do not understand how to use Eclipse. Over half of students in the study do not prefer to use Eclipse as their IDE. In fact, only 46% of students in the study prefer to use Eclipse as their IDE, with only 14% strongly agreeing that it is their IDE of choice.

Students often feel that they need help with programming assignments. In terms of students’ perceptions of content with which they need help on programming assignments, 86% of students report that it is hard for them to understand how to fix the errors reported by the IDE. The overwhelming majority of students feel that they get stuck on programming assignments and are unsure how to proceed. Not only do 82% of students sense this, but 50% of all students feel this strongly.

Students like to get help on programming assignments, although they already have resources for getting help. Many students find it helpful to get help from someone when they are stuck on a programming assignment with 64% of students reporting this. However, 36% report that they do not find this helpful. All students report that they can easily find someone who can answer their questions when they are stuck, with the overwhelming majority strongly agreeing.

6.2 Overall Perceptions of the Companion

This section explores students' general reactions to the Companion and opinions about it. Overall, students were receptive to the idea of the Companion, and 68% reported that they liked working with it. The Companion helped 64% of the students to feel supported while completing the assignment, as depicted in Figure 6.1. Students could select more than one response for statements that were true about their interactions with the Companion, and 23 out of 28 students found the idea of using a Companion during programming assignments attractive.

Although the response differed, depending on how the survey question was worded, as many as 43% of students said that the feedback from the Companion was helpful to them. Yet, when the same question was asked in a different way, only 15% of students report that the Companion's feedback helped them. Even though the idea of the Companion is attractive to students, most feel that it needs some improvement in order to be helpful to them. According to 71% of students, the Companion does not seem "intelligent" to them.

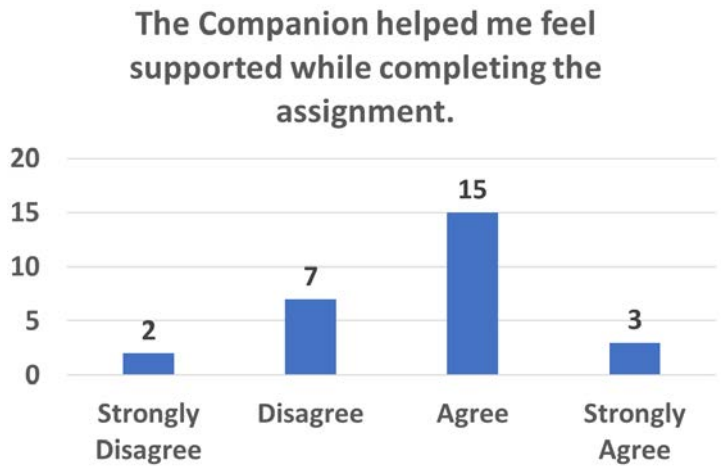


Figure 6.1. Students' response to whether the Companion helped them feel supported while completing the programming assignment

Thus, with improvement, students report that the Companion could help them, and they like the idea. As one student put it on the survey, “I love the idea of a companion while programming, [so] I can’t wait to see how this performs when it’s polished!” Given the data, this is a good summary of the students’ overall perception of the Companion.

6.3 Evaluation of the Companion’s Feedback

After collecting the data regarding the Companion’s feedback, we analyzed it for trends, looking for common patterns that emerged from it. The data used to evaluate the Companion’s feedback stems from the survey questions posed to the students and the ratings of the Companion’s feedback provided within the IDE while completing the assignment. This section categorizes the data based on the observed trends to evaluate how effective the Companion’s Brain was at generating intelligent feedback.

The Companion’s messages were successful in providing some of the intended types of hints. The intention for this thesis was to provide hints on how to proceed with an assignment, as well as problem-solving hints, specifically for a recursion programming assignment in Java. In some respects, some of these goals were accomplished. Over half of the students report that the Companion’s feedback gave hints about concepts needed for the assignment, which is shown in Figure 6.2. Similarly, 57% of students report that the Companion did not provide hints about syntax. Both of these trended towards the goals for the types of feedback the Companion should and should not provide.

However, only 36% of students report that the feedback provided hints about how to proceed with the assignment, which means that the majority of students did not perceive this to be true. Furthermore, only about a third of the students describe the Companion’s

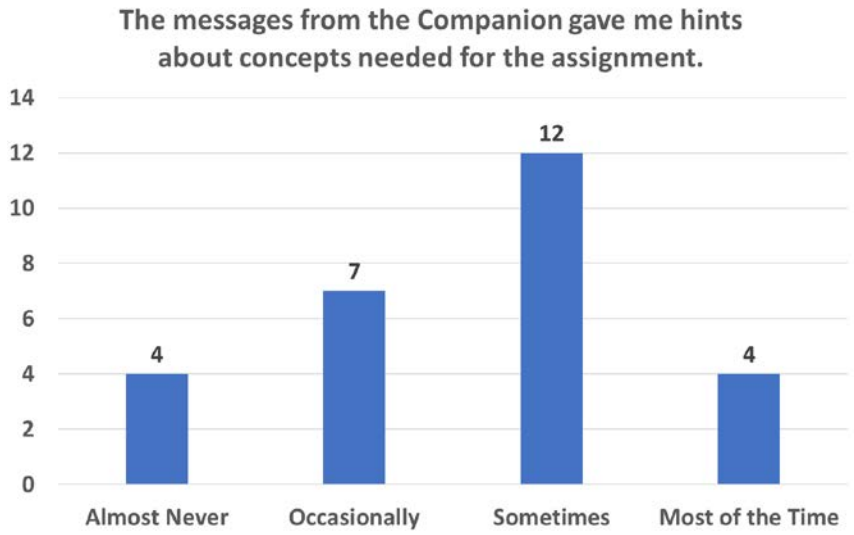


Figure 6.2. Students' perceptions of the type of feedback the Companion provided

feedback as logical for what was happening with their code. Also, only 36% say that the Companion improved their understanding of recursion.

Students suggest changing the messages themselves, the frequency, and the timing. Most commonly, students suggest changing the messages to something different. Students also believe that the frequency and the timing of the messages should be changed. Responses differ, but students suggest that it would be helpful if the Companion provided messages only in certain situations, such as when they have errors, after a time interval has passed, or when they request help. When problems were reported with feedback, students noticed that they frequently received the same feedback, making it seem repetitive. The other problems reported with the feedback indicate they did not understand what it meant. Open-ended responses regarding what improvements were needed on the feedback encompassed a range of topics, with wanting more specific feedback occurring the most frequently. Others ranged from giving an example of the problem to when to deliver the hint. One student suggests, "When the program compiles, it would be nice if it gave you

an explanation of what the error was before." Thus, this student would like an explanation concerning what he or she fixed and why it did not previously work.

Many messages were not saved in the database when they were given. Ideally, every time feedback was given to a student, the exact feedback would be saved in the database. However, this occurred less than half of the time. Thus, often students received feedback, but no data was collected regarding what the exact feedback was.

Students received a limited number of messages, given the values in the training data. Most of the 28 messages appeared at least once in the data used to train the neural network; however, students only received 10 different messages during the study. Thus, even though many different messages were labeled in the training data, only 10 of these were actually delivered to students.

Students reported mixed results about the helpfulness of the Companion's feedback. Among the 10 messages delivered and the times when students' ratings were collected in Eclipse, some messages were deemed more helpful than others. The three messages that least matched the situation were the following: "Sorry, I don't have any messages for you right now. Try again later," "Nice job! You got the right answer!" and "Remember to clean up your code! That's a lot of comment lines." Few messages were reported to be unequivocally helpful, but the ones reported to be the most helpful were the following: "Think about breaking the problem into smaller parts. Ask yourself: What is the smallest problem I'm trying to solve?" and "Think about changing the control flow of your code." However, the latter message was also commonly marked as one that students did not understand.

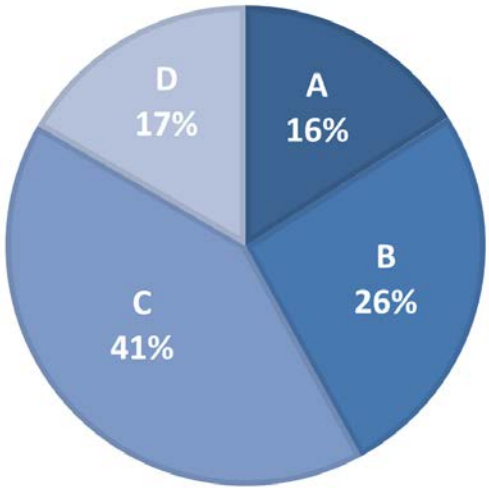


Figure 6.3. Students' ratings collected in the IDE of the Companion's feedback for the individual situation:
A. Message was helpful. B. Message was somewhat helpful.
C. Message did not match the situation. D. I did not understand what the message meant.

Students also provided mixed responses of their overall evaluation of the Companion's feedback. As shown in Figure 6.3, 42% of students reported that the feedback from the Companion was helpful or slightly helpful to them. Yet, 41% also stated that the Companion's feedback did not match the situation in their code. Some suggested that they did not understand the meaning of the feedback, which concerns what the message meant, but not necessarily the situation in which the Companion gave the feedback.

6.4 Evaluation of Data Collected from Students' Code

It may also be helpful to consider what data the plug-in collected from students' code. While the purpose of this thesis is mainly concerned with how to give helpful feedback to students in Eclipse, considering the input data may provide useful data for analyzing what types of feedback CS students would most require.

Students require support in understanding the features of recursion. Based on the data regarding keywords collected from students' code, approximately 20% of the time the keywords *if* or *return* were missing from the code or a loop was found. This suggests that a percentage of students require support in understanding which syntactical features are important for recursion.

The usefulness of the comparator operator depends upon the recursion assignment. The comparator operator appeared roughly half of the time in the programming assignment. The recursion assignment given to students did not require this operator; however, many other assignments do require it. Thus, it seems this may not be a useful input to the neural network, as the usefulness of this data varies based on the programming assignment.

Students at this level have a strong grasp of data types and when to use them. This assignment required students to use integers, so the double keyword only appeared 1% of the time, which was appropriate for the situation. Nonetheless, the usefulness of this data will vary based on the exact programming assignment, so this data may not be as relevant for the Brain.

6.5 Evaluation of the Software Tool

Since students were not specifically asked for feedback regarding the software tool itself, this data comes from any comments reported on the survey concerning the tool, as well as our observations throughout the study. Student comments on the survey made about the tool typically were provided as a response classified as "other," and often they did not necessarily pertain to the question asked. Nonetheless, they provided helpful data to

evaluate the software tool and the technical decisions that need to be considered for maximizing the effectiveness of the Companion.

There were six responses that indicated that the usability of the tool detracted from their experience with the Companion. When students were asked what could make the Companion's feedback more helpful, one student reported issues with the tool itself that impacted the effectiveness of the feedback. This student suggested that being taught how the tool works in advance would have been helpful for interacting with the feedback. Furthermore, this same student wrote, "I didn't know I had to manually check it myself after writing code." The tool required students to toggle between two views in Eclipse: the console view with compiler error messages and the Help View with the Companion's feedback. This student found that this factor affected his or her experience with the feedback.

Based on observation, the time spent installing the plug-in took almost as long as the completion of the assignment itself. Most students spent about 30 minutes installing the plug-in and 30 minutes completing the assignment. Although this installation time is somewhat misleading because a portion of the time was installing the correct version of Eclipse, it is still significant.

Furthermore, even though students were provided with detailed directions for the plug-in installation, they struggled with the installation process and frequently needed assistance with it. Most of the issues encountered were configuration problems between Eclipse and the plug-in, which could have been resolved with following the instructions. Nevertheless, the instructions were complex and required attention to detail.

An issue frequently occurred for students using Macs. The directions for the plug-in installation on a Mac were from the previous version of the plug-in implemented (Penumala and Gonzalez-Sanchez, 2018). Even though the directions for installation in Windows remained unchanged and still worked correctly, it appears that with an update to the Mac operating system, this resulted in the plug-in installation failing in these Macs, which accounted for approximately three participants. One student verbally hypothesized that the students with a Mac who got the plug-in to work had an older version of the OS than she did. She noticed that all students with Macs for whom the installation failed had the most recent Mac OS. In such cases when the installation failed, these students used the plug-in on our computer to complete the assignment, thus still allowing for data collection; however, due to our laptop already being used by another student, one student came for the study but could not use the plug-in.

Once students installed the plug-in, we observed that using the plug-in required some training. Most students did not intuitively know the process for interacting with the plug-in and checking the feedback. Since the IDE switches to the console view by default after compiling with an error message, some students did not even realize for a while that they were receiving feedback since they needed to click on the Help View in order to see it. Since the plug-in required students to enter their participant identification number and course number at the top of the assignment, the Companion would not deliver feedback if this was omitted. Consequently, some students did not receive any feedback for a period of time before realizing what the issue was. One student did not create a new workspace and even proceeded through the entire assignment without receiving any feedback because he did not realize he should receive it.

6.6 Summary

In this chapter, we discussed the results of the study. Initially, we reviewed students' background with recursion, how often they need help, and their success with resolving errors reported by the IDE. After that, we discussed how students perceived the Companion. They had an overwhelmingly positive response to the idea of the Companion and enjoyed working with it. Then we examined how the students reported the Companion did with the intelligent feedback. We found that the Companion successfully provided some of the types of feedback, but only 10 different messages were actually given to the students. Many students said the feedback was helpful or slightly helpful, although many also said the feedback did not make sense for the situation at hand. Following that, we discussed the data collected from students' code, and, lastly, we evaluated the Companion plug-in as a software tool. In the next chapter, we will provide a rationale for the results and discuss opportunities for improvement.

CHAPTER 7

DISCUSSION

Some clear trends were evident in the data collected from this study. The types of questions posed in the surveys were focused on getting suggestions for how to improve the Companion rather than ways the Companion was successful. It may be possible, then, that the students could have offered specific positive comments if we had asked different questions. Even so, both positive and negative traits can be seen in the Companion. This chapter explores possible reasons for this data. First, we will discuss strengths of the Companion as shown in this study. Next, we will provide an analysis of shortcomings of the Companion. Lastly, we will discuss opportunities for future work in this research area.

7.1 Strengths of the Companion

The Companion implemented in this study offers a worthwhile contribution to the previous work on ITS, specifically about Companions. There are several beneficial traits of our Companion that emerged from this research. This section provides an explanation of several of these characteristics.

Proof of Concept. Our Companion exists as a proof of concept for a Companion embedded into the IDE that provides feedback on how to proceed and affective support for students in a beginning Java programming course. The literature demonstrates a need for this type of work, as few companions in the IDE exist for teaching programming, and those existing do not provide feedback on how to proceed. Moreover, those implemented have not investigated using a neural network to generate the feedback. A proof of concept is an

important first step in advancing this area of research. Thus, our Companion provides a valuable contribution to the existing research. The software for the Companion worked to accomplish the goal of delivering intelligent feedback to students within the Eclipse IDE. Of course, it is the goal for all software to work. Nonetheless, due to the complexity of integrating existing software components to create a working product, it is important to fully appreciate the contribution of the Companion plug-in toward this end. Data was collected from each student's code, sent to the brain, saved to the server, and relatively intelligent feedback was displayed. This represents a new advancement in tutoring Companions. Thus, our Companion's role as a proof of concept is significant.

Neural Network for Feedback in the Eclipse IDE. Just as the Companion's contribution as a proof of concept is valuable, providing feedback within the IDE via a neural network is also noteworthy. Using neural networks in an ITS is not a novel contribution but delivering it within the IDE is rare. Neural networks are an area of AI attracting much attention and demonstrating the potential for using them with an embedded Companion for CS education is beneficial.

Support for Beginning CS Students. Even though some of the data was mixed regarding the effectiveness of the Companion's feedback, the students clearly conveyed that they liked the Companion and felt supported by it. They even saw the potential for a Companion helping them in the future. It is well-known that CS courses are difficult, especially for beginning students, which prompts many to leave CS (Giannakos, Pappas, et al., 2017). Retention is certainly not the topic for this thesis but having a tutoring Companion that can help CS students – even if only to provide affective support – is valuable for students. Perhaps in the future this data could contribute to these other

discussions. At any rate, this Companion succeeded in helping students feel supported. Nonetheless, our Companion succeeds at more than just affective support. Although the Brain will need some improvement, there was some data that showed that students received feedback that made sense at the appropriate time. Furthermore, students reported that it helped them with concepts for problem-solving, which was one of the stated outcomes for this research. Thus, some of the Companion's feedback provided affective support for students and guidance about problem-solving concepts for recursion.

7.2 Opportunities for Improvement in the Companion

Although we made meaningful contributions to the research in this area and the Companion met some of its objectives, there are opportunities to refine the Companion, particularly with its intelligence. This section provides a discussion of some opportunities for improvement in the Companion as shown through this study.

The messages did not always fit the situation. Students reported that sometimes the feedback did not make sense based on what was happening in their code. In fact, this was rather evident in the data and their responses. The data sent to the server with messages in different situations also shows that a human tutor would not select this same message for the situation. For instance, at times the students had an error in their code, and the Companion told them, "Nice job! You got the right answer!" Clearly, this was not the intent. This is, of course, the challenge with neural networks in any domain; it is difficult to simulate the thinking that occurs in the human brain and how humans reach those conclusions. Nonetheless, neural networks *can*, in fact, produce intelligent responses, and are used in many fields for this very purpose. Therefore, it is important to analyze why our

Companion missed the mark in this respect. One alarming piece of data was that the Companion only delivered 10 different messages regardless of the situation. The Brain had 28 unique messages, and these were labeled in the training data, so it leads one to wonder why this happened. After examining the data collected from the code, the same feedback was at times given even when the input data to the neural network was quite different. After closer investigation, a bug was identified in the Brain, which led to only 10 possible messages being generated. This bug was introduced when the output from the neural network was scaled to match the type of label for the number for the feedback. Since the output was multiplied by 100 to make it a multiple of 10, sometimes the code for the feedback should have been 200, for instance, instead of 20. In hindsight, this seems clear, but this bug was not detected in the testing plan since it was hard to notice what should be expected without authentic data entering the neural network. Furthermore, this decision to give a code to the messages as a multiple of 10 was to create a larger "spread" between the numbers for cases where the output from the neural network might be close together. This decision, in part, led to this bug. In the future, this is a technical decision that will require closer examination, but it seems that the neural network could be 28 outputs with the Brain choosing the node with the greatest weight. However, this will require greater investigation in the future.

The meaning of the feedback was not always clear to students. Students frequently reported that they did not understand what the feedback meant. This was a surprising trend in the data, as this does not pertain to the brain or even the software; rather, it concerns the wording of the feedback . Since it is not really a technical issue and is more

of an educational concern, it is beyond the scope of this thesis to analyze it, but it is, nevertheless, worth mentioning.

The usability of the plug-in appeared to detract from the Companion's feedback. The students seemed frustrated with navigating the views in Eclipse. This was not concrete data reported from the study, and it appears they may not have even consciously realized this affected the data. However, after finishing the assignment, numerous students verbally told us that they rated the Companion's feedback for a while, but then they stopped because they got tired of switching between views. A limitation of the plug-in implementation was that the Companion's feedback was only saved in the database *after* a student rated the feedback. Thus, if the student did not rate the feedback, the exact feedback was not recorded. Furthermore, since the students wanted to see the error messages from the compiler, this required navigating between the console view and the help view that was added for the plug-in. This limited the usability, and it seems this may have been a contributing factor to students not rating as much of the feedback or perhaps even checking the feedback at all.

7.3 Future Work

Now that strengths and weaknesses of the Companion have been discussed, we will consider the vision for future research. Some of these opportunities for future work involve improvements to the Companion by improving the software used in this study, which will be discussed first. Then, the potential of a self-training network will be discussed. Lastly, opportunities regarding when to give feedback will be explored.

Software Improvements to the Companion. It would be informative to see how the Companion's feedback can change once the bug is fixed in the Brain for the scaling of the output. One could examine whether the neural network delivers quality messages with even this small change. Since the usability of the plug-in seemed to affect the data from the Companion, improving the UI experience of the plug-in would be beneficial. Instead of adding the views as tabs next to the console view, the view could be placed somewhere else in the IDE where it would not interfere with students' typical interactions with the IDE. It is difficult to determine where this would be, but this could be another topic of research. Examining this could help to determine what role the feedback plays as compared to the UI experience. ITS often use an avatar to make the Companion more visually appealing and to seem more personal. It also could make it seem more "fun" to the student and perhaps increase the level of personal engagement. This represents another opportunity for future improvements on the Companion. Improvements to the Brain would also be beneficial. Now that authentic data has been collected from beginning CS students, this data can be used to retrain the neural network. This would greatly enhance which feedback to provide in which situation, instead of relying on automatically generated data as in this study.

Self-training Neural Network. Since one of the purposes of the Companion is to emulate the traits of a human tutor, it is important for the Companion to possess the capability to provide feedback without the intervention of a human. Currently, the neural network is a supervised learning algorithm that requires human intervention. Since the data is saved to the server and the neural network is also on the server, there is potential to have the neural network automatically train so that the brain continually learns from the data

from the students. Perhaps the neural network could train in a short daily scheduled downtime with an unsupervised learning algorithm. This would mark a significant advancement in this research, for it would allow the Companion to rely less on human intervention and, thus, each student using the Companion would benefit from what the Companion learned from every preceding student.

When to Give Feedback. This is a broader research question that has already warranted significant attention (VanLehn, 2006). Students reported many issues about the timing of the Companion's feedback, ranging from only wanting feedback when they have errors to wanting feedback after a certain amount of time has passed to almost every other conceivable option. Even more problematic is that students can abuse hints because they do not want to give a quality attempt at solving the problem (VanLehn, 2006). When to give feedback is a difficult question to answer, and it could inform future research with this Companion. In the context of a programming Companion embedded into the IDE, this could mean changing what triggers displaying the feedback. For us, this was the defined actions detected in the IDE. In the future, this could be giving feedback on a timer. Or, perhaps the Companion could project when the student will encounter problems and issue feedback preemptively. Indeed, a plethora of options exist that are worth exploring, especially since the neural network can be used to give a hint in many different types of scenarios.

7.4 Summary

This thesis provided important research in the area of embedded tutoring Companions. It contained some limitations, such as a small bug and the usability of the plug-in. However,

it suggests the opportunity to use neural networks in embedded Companions in the future. Most importantly, this thesis exists as a proof of concept that neural networks can be effectively used to provide intelligent feedback directly in the IDE for students' programming assignments. Since students like the feedback and even report that it is somewhat intelligent, this provides opportunities to leverage this technology in the future with beginning CS students. Many of these students struggle to obtain valuable feedback on their programming assignments and experience a lack of quality personalized feedback on their code. It seems that a Companion embedded into the IDE can help to provide this feedback, using a neural network for its intelligence. This, it seems, should warrant future research attention.

REFERENCES

Acuña, R. 2018. SER 222: Design & Analysis: Data Structures and Algorithms Syllabus. Retrieved February 13, 2019 from https://webapp4.asu.edu/bookstore/viewsyllabus/2191/16004.

Anderson, J.R. and Skwarecki, E. 1986. The automated tutoring of introductory computer programming. *Communications of the ACM, 29*(9), 842-849. DOI: https://doi.org/10.1145/6592.6593.

Barr, A. and Beard, M. 1976. An instructional interpreter for basic. In *SIGCSE '76 Proceedings of the ACM SIGCSE-SIGCUE Technical Symposium on Computer Science and Education*, New York, NY, ACM, 325-334. DOI: https://doi.org/10.1145/952989.803494.

Beck, J.E., Woolf, B.P. and Beal, C.R. 2000. ADVISOR: A machine learning architecture for intelligent tutor construction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence,* AAAI Press, 552-557.

Carless, D. 2006. Differing perceptions in the feedback process. *Studies in Higher Education, 31*(2), 219-233. DOI: https://doi.org/10.1080/03075070600572132.

Chatley, R. and Timbul, T. 2005. KenyaEclipse: Learning to program in Eclipse. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Lisbon, Portugal, ACM, 245-248. DOI: https://doi.org/10.1145/1095430.1081746.

Chen, Z. and Marx, D. 2005. Experiences with Eclipse IDE in programming courses. *Journal of Computing Sciences in Colleges, 21*(2), 104-112.

Chow, S., Yacef, K., Koprinska, I. and Curran, J. 2017. Automated data-driven hints for computer programming students. In *25th ACM International Conference on User Modeling, Adaptation, and Personalization, UMAP 2017*, Bratislava, Slovakia, ACM, 5-10. DOI: https://doi.org/10.1145/3099023.3099065.

Conati, C., Gertner, A. and VanLehn, K. 2002. Using Bayesian networks to manage uncertainty in student modeling. *User Modeling and User - Adapted Interaction, 12*(4), 371-417. DOI: https://doi.org/10.1023/A:1021258506583.

Connolly, C., Murphy, E. and Moore, S. 2009. Programming anxiety amongst computing students—a key in the retention debate? *IEEE Transactions on Education, 52*(1), 52-56. DOI: https://doi.org/10.1109/TE.2008.917193.

Crow, T., Luxton-Reilly, A. and Wuensche, B. 2018. Intelligent tutoring systems for programming education: A systematic review. In *Proceedings of the 20th*

Australasian Computing Education Conference, Brisbane, Queensland, Australia, ACM, 53-62. DOI: https://doi.org/10.1145/3160489.3160492.

Data Design Group. 2018. Generate Test Data: JSON Format. Retrieved February 1, 2019 from http://www.convertcsv.com/generate-test-data.htm#keywords.

Day, M. 2019. TutorBrain GitHub Repository. Retrieved April 3, 2019 from https://github.com/melissaDay1/TutorBrain.

Devide, J.V.S., Meneely, A., W Ho, C., Williams, L. and Devetsikiotis, M. 2008. Jazz Sangam: A real-time tool for distributed pair programming on a team development platform. In *Workshop on Infrastructure for Research in Collaborative Software Engineering,* Atlanta, GA.

Dogaru, I. and Dogaru, R. 2013. JLCNN: An object-oriented Java package for low complexity neural networks. In *2013 4th International Symposium on Electrical and Electronics Engineering (ISEEE)*, 1-6. DOI: https://doi.org/10.1109/ISEEE.2013.6674317.

Giannakos, M.N., Pappas, I.O., Jaccheri, L. and Sampson, D.G. 2017. Understanding student retention in computer science education: The role of environment, gains, barriers and usefulness. *Education and Information Technologies, 22*(5), 2365-2382. DOI: https://doi.org/10.1007/s10639-016-9538-1.

Gonzalez-Sanchez, J. 2018. Syllabus: Introduction to Programming Languages. Retrieved March 19, 2019 from http://javiergs.com/teaching/cse240.

Graesser, A.C., Lu, S., Jackson, G.T., Mitchell, H.H., Ventura, M., Olney, A. and Louwerse, M.M. 2004. AutoTutor: A tutor with dialogue in natural language. *Behavior Research Methods, Instruments, & Computers, 36*(2), 180-192. DOI: https://doi.org/10.3758/BF03195563.

Hattie, J. 2009. *Visible Learning: A Synthesis of Over 800 Meta-Analyses Relating to Achievement.* Routledge, New York, NY.

Hattie, J. and Timperley, H. 2007. The power of feedback. *Review of Educational Research , 77*(1), 81-112. DOI: https://doi.org/10.3102/00346543029848.

Heaton, J. 2015. Encog: Library of interchangeable machine learning models for Java and C#. *ArXiv,* 5.

Hooshyar, D., Binti Ahmad, R., Wang, M., Yousefi, M., Fathi, M. and Lim, H. 2018. Development and evaluation of a game-based Bayesian intelligent tutoring system for teaching programming. *Journal of Educational Computing Research, 56*(6), 775-801. DOI: https://doi.org/10.1177/0735633117731872.

Hounsell, D. 2007. Towards more sustainable feedback to students. In *Rethinking assessment in higher education: Learning for the longer term,* Boud, D. and Falchikov, N., Eds. Routledge, London, UK.

Hounsell, D. 2003. Student feedback, learning and development. In *Higher education and the lifecourse,* Slowey, M. and Watson, D., Eds. Open University Press, Buckingham, UK, 67-78.

International Business Machines Corporation. 2006. Eclipse Platform Technical Overview. Retrieved January 18, 2019 from https://www.Eclipse.org/articles/Whitepaper-Platform-3.1/Eclipse-platform-whitepaper.pdf.

Jessop, T., Hakim, Y.E. and Gibbs, G. 2013. The whole is greater than the sum of its parts: A large-scale study of students' learning in response to different programme assessment patterns. *Assessment and Evaluation in Higher Education, 39*(1), 73-88. DOI: https://doi.org/10.1080/02602938.2013.792108.

Johnson, W.L. and Soloway, E. 1984. PROUST: Knowledge-based program understanding. In *Proceedings of the 7th International Conference on Software Engineering,* Orlando, Florida, USA, IEEE Press, 369-380.

Judd, C.M. and Shittu, H. 2005. *Pro Eclipse JST: Plug-ins for J2EE Development.* Apress, New York, NY.

Jurado, F., Redondo, M. and Ortega, M. 2014. eLearning standards and automatic assessment in a distributed Eclipse based environment for learning computer programming. *Computer Applications in Engineering Education, 22*(4), 774-87. DOI: https://doi.org/10.1002/cae.21569.

Jurado, F., Molina, A.I., Redondo, M.A. and Ortega, M. 2013. Cole-programming: Shaping collaborative learning support in Eclipse. *IEEE-RITA Latin American Learning Technologies Journal, 8*(4), 153-162. DOI: https://doi.org/10.1109/RITA.2013.2284953.

Jurado, F., Redondo, M.A. and Ortega, M. 2012. Using fuzzy logic applied to software metrics and test cases to assess programming assignments and give advice. *Journal of Network and Computer Applications, 35*(2), 695-712. DOI: https://doi.org/10.1016/j.jnca.2011.11.002.

Karkalas, S. and Gutierrez-Santos, S. 2014. Eclipse student (in) activity detection tool. In *9th European Conference on Technology Enhanced Learning (EC-TEL 2014),* Springer International Publishing, 572-3. DOI: https://doi.org/10.1007/978-3-319-11200-8_75.

Keuning, H., Jeuring, J. and Heeren, B. 2018. A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education, 19*(1), 3:1-3:43. DOI: https://doi.org/10.1145/3231711.

King, C.E. 2018. Feasibility and acceptability of peer assessment for coding assignments in large lecture based programming engineering courses. In *2018 IEEE Frontiers in Education Conference*, San Jose, CA, IEEE, DOI: https://doi.org/10.1109/FIE.2018.8659246.

Le, N. 2016. A classification of adaptive feedback in educational systems for programming. *Systems, 4*(2), 22. DOI: https://doi.org/10.3390/systems4020022.

Le, N., Strickroth, S., Gross, S. and Pinkwart, N. A review of AI-supported tutoring approaches for learning programming. In *International Conference on Computer Science, Applied Mathematics, & Applications 2013*, Switzerland, Springer International Publishing, 267-279. DOI: https://doi.org/10.1007/978-3-319-00293-4_20.

Ma, W., Adesope, O.O., Nesbit, J.C. and Liu, Q. 2014. Intelligent tutoring systems and learning outcomes: A meta-analysis. *Journal of Educational Psychology, 106*(4), 901-918. DOI: https://doi.org/10.1037/a0037123.

MacNish, C. 2002. Machine learning and visualisation techniques for inferring logical errors in student code submissions. In *ICALT-2002: Proceedings of the IEEE International Conference on Advanced Learning Technologies*, IEEE, 317-321. DOI: https://doi.org/10.1017/S0890060419000027.

McGettrick, A., Boyle, R., Ibbett, R., Lloyd, J., Lovegrove, G. and Mander, K. 2005. Grand challenges in computing: Education—A summary. *The Computer Journal, 48*(1), 42-48. DOI: https://doi.org/https://doi.org/10.1093/comjnl/bxh064.

Moritz, S.H., Blank, G.D., Parvez, S. and Wei, F. 2007. A design-first curriculum and Eclipse tools. *Journal of Computing Sciences in Colleges, 22*(3), 51-52.

Moritz, S.H., Wei, F., Parvez, S.M. and Blank, G.D. 2005. From objects-first to design-first with multimedia and intelligent tutoring. In *ITiCSE '05 Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, Caparica, Portugal, ACM, 99-103. DOI: https://doi.org/10.1145/1151954.1067475.

Narciss, S. 2008. Feedback strategies for interactive learning tasks. In *Handbook of research on educational communications and technology.* Spector, M. J., Ed. Springer, New York, NY, 125-144.

Nesbit, J.C., Liu, L., Liu, Q. and Adesope, O.O. 2015. Work in progress: Intelligent tutoring systems in computer science and software engineering education.

Proceedings 122nd American Society Engineering Education Annual Conference, DOI: https://doi.org/10.18260/p.25090.

Nesbit, J.C., Adesope, O.O., Liu, Q. and Ma, W. 2014. How effective are intelligent tutoring systems in computer science education? In *2014 IEEE 14th International Conference on Advanced Learning Technologies (ICALT)*, IEEE Computer Society, 99-103. DOI: https://doi.org/10.1109/ICALT.2014.38.

Neuroph Library. 2014. API: Interface Normalizer. Retrieved March 29, 2019 from http://neuroph.sourceforge.net/javadoc/index.html.

Neuroph Projects. 2019. Neuroph Open-Source Projects GitHub Repository. Retrieved January 25, 2019 from https://github.com/neuroph/neuroph/tree/master/neuroph-2.9/Samples/src/main/java/org/neuroph/samples.

Nguyen-Thinh, L.E., Menzel, W. and Pinkwart, N. 2009. Evaluation of a constraint-based homework assistance system for logic programming. In *Proceedings of the 17th International Conference on Computers in Education (CDROM)*, Hong Kong, Asia-Pacific Society for Computers in Education, 51-58.

Nye, B.D., Graesser, A.C. and Hu, X. 2014. AutoTutor and family: A review of 17 years of natural language tutoring. *International Journal of Artificial Intelligence in Education, 24*(4), 427-469. DOI: https://doi.org/10.1007/s40593-014-0029-5.

Ott, C., Robins, A. and Shephard, K. 2016. Translating principles of effective feedback for students into the CS1 context. *ACM Transactions on Computing Education, 16*(1), 1:1-1:27. DOI: https://doi.org/10.1145/2737596.

Penumala, Manohara Rao. 2017. TutorHelpPlugin GitHub Repository. Retrieved September 8, 2018 from https://github.com/mpenumal/TutorHelpPlugin.

Penumala, M.R. and Gonzalez-Sanchez, J. 2018. Towards embedding a tutoring companion in the Eclipse integrated development environment. In *14th International Conference on Intelligent Tutoring Systems, ITS 2018*, Montreal, Canada, Springer, 352-358. DOI: https://doi.org/10.1007/978-3-319-91464-0_39.

Perkins, D., Hancock, C., Hobbs, R., Martin, F. and Simmons, R. 1989. Conditions of learning in novice programmers. In *Studying the novice programmer*, Soloway, E. and SPOHRER, J. C., Eds. Lawrence Erlbaum, Hillsdale, NJ, 261-279.

Pillay, N. 2003. Developing intelligent programming tutors for novice programmers. *SIGCSE Bulletin, 35*(2), 78-82. DOI: https://doi.org/10.1145/782941.782986.

Pisan, Y., Sloane, A., Richards, D. and Dale, R. 2002. Providing timely feedback to large classes. In *International Conference on Computers in Education,* Auckland, New Zealand, IEEE, 413. DOI: https://doi.org/1959.14/1232327.

Price, M., Handley, K. and Millar, J. 2011. Feedback: Focusing attention on engagement. *Studies in Higher Education, 36*(8), 879-896. DOI: https://doi.org/10.1080/03075079.2010.483513.

Ramaprasad, A. 1983. On the definition of feedback. *Behavioral Science, 28*(1), 4. DOI: https://doi.org/10.1002/bs.3830280103.

Reis, C. and Cartwright, R. 2004. Taming a professional IDE for the classroom. *SIGCSE Bulletin , 36*(1), 156-160. DOI: https://doi.org/10.1145/1028174.971357.

Reis, C. and Cartwright, R. 2003. A friendly face for Eclipse. In *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange,* Anaheim, California, ACM, 25-29. DOI: https://doi.org/10.1145/965660.965666.

Rivers, K. and Koedinger, K.R. 2017. Data-driven hint generation in vast solution spaces: A self-improving python programming tutor. *International Journal of Artificial Intelligence in Education, 27*(1), 37-64. DOI: https://doi.org/10.1007/s40593-015-0070-z.

Seymour, E. and Hewitt, N. 1997. *Talking about leaving: Why undergraduates leave the sciences.* Westview Press, Boulder, CO.

Shute, V.J. and Psotka, J. 1996. Intelligent tutoring systems: Past, present, and future. In *Handbook of research for educational communications and technology,* Jonassen, D., Ed. Macmillan, New York, NY, 570-600.

Silva, A., Leal, J.P. and Paiva, J.C. 2018. Raccode: An Eclipse plugin for assessment of programming exercises. In *7th Symposium on Languages, Applications and Technologies, SLATE 2018,* Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, DOI: https://doi.org/10.4230/OASIcs.SLATE.2018.4.

Singh, R., Gulwani, S. and Solar-Lezama, A. 2012. Automated feedback generation for introductory programming assignments. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '13),* Seattle, Washington, ACM, 15-26. DOI: https://doi.org/10.1145/2491956.2462195.

Smaill, C.R. 2005. The implementation and evaluation of OASIS: A web-based learning and assessment tool for large classes. *IEEE Transactions on Education, 48*(4), 658-663. DOI: https://doi.org/10.1109/TE.2005.852590.

Sottilare, R., Graesser, A.C., Hu, X. and Holden, H., Eds. 2013. Design Recommendations for Intelligent Tutoring Systems. U.S. Army Research Laboratory, Orlando, FL.

SourceForge. n.d. Neuroph: Java Neural Network Framework. Retrieved January 19, 2019 from http://neuroph.sourceforge.net/index.html.

Spacco, J., Hovemeyer, D. and Pugh, W. 2004. An Eclipse-based course project snapshot and submission system. In *Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange*, Vancouver, British Columbia, Canada, ACM, 52-56. DOI: https://doi.org/10.1145/1066129.1066140.

Storey, M., Damian, D., Michaud, J., Myers, D., Mindel, M., German, D., Sanseverino, M. and Hargreaves, E. 2003. Improving the usability of Eclipse for novice programmers. In *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange*, Anaheim, California, ACM, 35-39. DOI: https://doi.org/10.1145/965660.965668.

Suarez, M. and Sison, R. 2008. Automatic construction of a bug library for object-oriented novice Java programmer errors. In *International Conference on Intelligent Tutoring Systems (ITS 2008)*, Montreal, Canada, Springer, 184-193. DOI: https://doi.org/10.1007/978-3-540-69132-7_23.

VanLehn, K. 2006. The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education, 16*(3), 227-265.

Winslow, L.E. Sept. 1996. Programming pedagogy - a psychological overview. *ACM SIGCSE Bulletin, 28*(3), 17-22. DOI: https://doi.org/10.1145/234867.234872.

Yusri, N., Mashita Syed-Mohamad, S. and Rashid, N.A. 2015. ReCOOP: A collaborative tool to support teaching and learning programming. *ICIC Express Letters, 9*(10), 2703-10.

Zhang, Y., Huang, G., Zhang, N. and Mei, H. 2009. SmartTutor: Creating IDE-based interactive tutorials via editable replay. In *2009 31st International Conference on Software Engineering (ICSE 2009)*, IEEE, 559-62. DOI: https://doi.org/10.1109/ICSE.2009.5070555.

APPENDIX A

IRB PROTOCOL

Instructions and Notes:

- Depending on the nature of what you are doing, some sections may not be applicable to your research. If so, mark as "NA".
- When you write a protocol, keep an electronic copy. You will need a copy if it is necessary to make changes.

1 Protocol Title

Include the full protocol title: A Neural Network Model for a Tutoring Companion Supporting Students in a Programming with Java Course

2 Background and Objectives

Provide the scientific or scholarly background for, rationale for, and significance of the research based on the existing literature and how will it add to existing knowledge.

- Describe the purpose of the study.
- Describe any relevant preliminary data or case studies.
- Describe any past studies that are in conjunction to this study.

With large class sizes and instructors who may not be equipped to assist struggling students, many students abandon the field, deeming it to be too difficult and not for them. Consistent, constructive, supportive feedback through a Tutoring Companion can scaffold the learning process for students. This poster describes a reasoning model, using neural networks techniques, for a tutoring companion embedded into the Eclipse IDE. The companion provides support for students in a first-year university Java programming course. The companion collects data from students' events and programming assignments, analyzes it for relevant trends, and estimates each student's situation. The input data for the neural network comes from areas with which beginning computer science students often struggle, such as the presence of important keywords and the amount of time spent in a state with errors. Then, it determines the feedback to be provided for students to overcome a detected challenging situation, providing both hints on how to fix the problem with the code, as well as encouragement to help keep students motivated and learning. The effectiveness of the approach is examined among first-year computer science students through the completion of recursion and control flow programming assignments. The students complete surveys regarding their learning experience to assist in evaluating the companion's pedagogical effectiveness, which is discussed with an emphasis on the value of feedback provided.

3 Data Use

Describe how the data will be used. Examples include:

- Dissertation, Thesis, Undergraduate honors project
- Publication/journal article, conferences/presentations
- Results released to agency or organization
- Results released to participants/parents
- Results released to employer or school
- Other (describe)

The data will be used for an MS Thesis project. It will be analyzed and included the thesis document in aggregate form. It also may be used as part of a future journal article for publication.

4 Inclusion and Exclusion Criteria

Describe the criteria that define who will be included or excluded in your final study sample. If you are conducting data analysis only describe what is included in the dataset you propose to use.

Indicate specifically whether you will target or exclude each of the following special populations:

- Minors (individuals who are under the age of 18)
- Adults who are unable to consent
- Pregnant women
- Prisoners
- Native Americans
- Undocumented individuals

Students in two undergraduate software engineering courses (SER 222 CSE 240) will be recruited for participating in the study. Any student who chooses to participate may do so. The following fields will be collected from each student's code for the programming activity:

- Action on the code (run attempt, debug attempt, help requested)
- Number of lines of code
- Keywords found in the code
- Number of comment lines
- Errors found in the code
- Submission time of the assignment
- Whether the assignment was completed successfully
- Message given as a hint to the student for that situation
- Student's feedback on the message received for that situation
- Metrics for the code

5 Number of Participants

Indicate the total number of participants to be recruited and enrolled*:* 40

6 Recruitment Methods

- Describe who will be doing the recruitment of participants.
- Describe when, where, and how potential participants will be identified and recruited.
- Describe and attach materials that will be used to recruit participants (attach documents or recruitment script with the application).

- Recruitment will be conducted by a combination of the graduate student (Melissa Day) and the two professors (Ruben Acuña and Javier Gonzalez-Sanchez) of the courses from which the students will be recruited.
- Students will be invited to participate through two means: an announcement posted on Canvas and an in-person announcement in each of the two courses.
- See attached recruitment materials.

7 Procedures Involved

Describe all research procedures being performed, who will facilitate the procedures, and when they will be performed. Describe procedures including:

- The duration of time participants will spend in each research activity.
- The period or span of time for the collection of data, and any long term follow up.
- Surveys or questionnaires that will be administered (Attach all surveys, interview questions, scripts, data collection forms, and instructions for participants to the online application).
- Interventions and sessions (Attach supplemental materials to the online application).
- Lab procedures and tests and related instructions to participants.
- Video or audio recordings of participants.
- Previously collected data sets that that will be analyzed and identify the data source (Attach data use agreement(s) to the online application).

Participants will participate in the following research activities, which will take about 1 hour. Data collection will occur over a one-week time period, and there will be no long-term follow up.

These activities will be conducted in the presence of the graduate student conducting the research (Melissa Day). Melissa Day will assist with any technical issues with the software during the data collection process.

- Survey before using the software – approximately 10 minutes
  Students will be given a survey as a Google Form with questions regarding their prior experience and reactions to the topic presented in the assignment.
- Using the software – approximately 30 minutes
- Survey after using the software – approximately 10 minutes
  Students will be given a survey as a Google Form with questions regarding their experience with the software and opinions about it.

See attached documents for an overview of these items.

8 Compensation or Credit

- Describe the amount and timing of any compensation or credit to participants.
- Identify the source of the funds to compensate participants
- Justify that the amount given to participants is reasonable.
- If participants are receiving course credit for participating in research, alternative assignments need to be put in place to avoid coercion.

Students who participate will receive 3% extra credit on their final course grade.

If students do not wish to participate, another extra credit opportunity will be offered in the course.

9 Risk to Participants

List the reasonably foreseeable risks, discomforts, or inconveniences related to participation in the research. Consider physical, psychological, social, legal, and economic risks.

There are no risks, discomforts, or inconveniences related to participation in the research.

10 Potential Benefits to Participants

Realistically describe the potential benefits that individual participants may experience from taking part in the research. Indicate if there is no direct benefit. Do not include benefits to society or others.

Participants may receive extra knowledge from the tutoring software regarding a topic presented in computer science courses. If the tutoring software is not particularly effective, then there may be no direct benefit to the participants.

11 Privacy and Confidentiality

Describe the steps that will be taken to protect subjects' privacy interests. "Privacy interest" refers to a person's desire to place limits on with whom they interact or to whom they provide personal information. Click here for additional guidance on ASU Data Storage Guidelines.

Describe the following measures to ensure the confidentiality of data:

- Who will have access to the data?
- Where and how data will be stored (e.g. ASU secure server, ASU cloud storage, filing cabinets, etc.)?
- How long the data will be stored?
- Describe the steps that will be taken to secure the data during storage, use, and transmission. (e.g., training, authorization of access, password protection, encryption, physical controls, certificates of confidentiality, and separation of identifiers and data, etc.).
- If applicable, how will audio or video recordings will be managed and secured. Add the duration of time these recordings will be kept.
- If applicable, how will the consent, assent, and/or parental permission forms be secured. These forms should separate from the rest of the study data. Add the duration of time these forms will be kept.
- If applicable, describe how data will be linked or tracked (e.g. masterlist, contact list, reproducible participant ID, randomized ID, etc.).

If your study has previously collected data sets, describe who will be responsible for data security and monitoring.

Data collected will be stored on a secure server with password protection. The graduate student and the PI will have access to the data. It will be stored throughout the duration of the study and for 6 months following the conclusion of the study.

Participants will be given a participant ID that is collected with the data. Participants' names will be separated from the data collected so that their responses cannot be traced back to them. This will be implemented with the following procedure:

- Random number generator will be used to create a unique ID for each participant. For data analysis, they will enter this number on the surveys and the programming assignment. This allows the researchers to group responses from each participant together for data analysis.
- This participant ID will not be recorded on any documents or other materials with the participant's name.
- Thus, the data anonymity will be preserved so that the response cannot be traced back to the participant.
- Participants' names will be collected for extra credit purposes only. This will be in a secure online survey in which students enter their name. It will contain no language regarding the study or their participant ID.

12 Consent Process

Describe the process and procedures process you will use to obtain consent. Include a description of:

- Who will be responsible for consenting participants?
- Where will the consent process take place?
- How will consent be obtained?
- If participants who do not speak English will be enrolled, describe the process to ensure that the oral and/or written information provided to those participants will be in that language. Indicate the language that will be used by those obtaining consent. Translated consent forms should be submitted after the English is approved.

The graduate student conducting the research (Melissa Day) will be responsible for obtaining the consent of participants.

When participants come for the 1-hour session for the study, they will be provided the HRP-502a-Consent Document Social Behavioral. It will be posted in a Google Form, and they will acknowledge their agreement by proceeding with the pre-assignment quiz.

13 Training Provide the date(s) the members of the research team have completed the CITI training for human participants. This training must be taken within the last 4 years. Additional information can be found at: Training.
March 2019

APPENDIX B

IRB APPROVAL

EXEMPTION GRANTED

Javier Gonzalez Sanchez
Software Engineering
javiergs@asu.edu
Dear Javier Gonzalez Sanchez:
On 3/13/2019 the ASU IRB reviewed the following protocol:

Type of Review:	Initial Study
Title:	A Neural Network Model for a Tutoring Companion Supporting Students in a Programming with Java Course
Investigator:	Javier Gonzalez Sanchez
IRB ID:	STUDY00009864
Funding:	None
Grant Title:	None
Grant ID:	None
Documents Reviewed:	• StudyProtocol_RecruitmentMaterials_IRB.pdf, Category: Recruitment Materials; • StudyProtocolForm_IRB.docx, Category: IRB Protocol; • HRP-502a - TEMPLATE CONSENT SOCIAL BEHAVIORAL_Revised.pdf, Category: Consent Form; • Pre-Assignment Survey.pdf, Category: Participant materials (specific directions for them); • Programming Assignment.pdf, Category: Participant materials (specific directions for them); • Software Procedure & Screenshots.pdf, Category: Participant materials (specific directions for them); • Post-Assignment Survey.pdf, Category: Participant materials (specific directions for them);

The IRB determined that the protocol is considered exempt pursuant to Federal Regulations 45CFR46 (1) Educational settings, (2) Tests, surveys, interviews, or observation on 3/13/2019.

In conducting this protocol you are required to follow the requirements listed in the INVESTIGATOR MANUAL (HRP-103).

Sincerely,
IRB Administrator
cc: Melissa Day
Javier Gonzalez Sanchez
Melissa Day

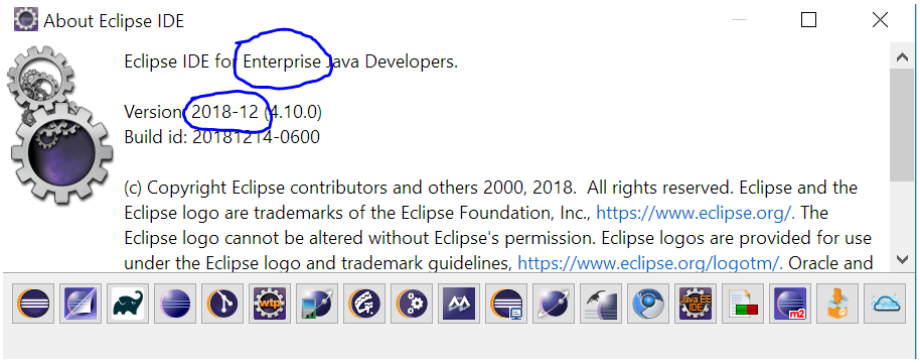
APPENDIX C

SYSTEM SET-UP DIRECTIONS

System Set-up

Before installing the plug-in, you need to confirm the following:

1. **The correct version of Eclipse is installed.**
   Eclipse IDE 2018-12, Java EE
   Make sure that it is the **Enterprise Edition**.
   If you already have Eclipse, check the version by doing the following:
   - Open Eclipse
   - Help
   - About -> It should say “Enterprise” and look like this.



   It can be downloaded here: https://www.eclipse.org/downloads/packages/.
   Download this option.

2. **Java 1.8 is installed.**
   If you need help determining this, see this article:
   https://www.java.com/en/download/help/version_manual.xml.
   It can be downloaded here:

   https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-

   2133151.html.

APPENDIX D

IRB SOCIAL BEHAVIORAL CONSENT FORM

Title of research study: A Neural Network Model for a Tutoring Companion Supporting Students in a Programming with Java Course

Investigator: Dr. Javier Gonzalez-Sanchez

Why am I being invited to take part in a research study?

We invite you to take part in a research study because we have developed a tutoring companion as a plug-in for the Eclipse Integrated Development Environment (IDE). This provides feedback to students in a course that involves programming with Java.

You must be at least 18 or older to participate in this study.

Why is this research being done?

The effectiveness of the approach is examined among computer science students through the completion of recursion and control flow programming assignments.

How long will the research last?

We expect that individuals will spend 1 hour participating in the proposed activities.

How many people will be studied?

We expect about 40 people will participate in this research study.

What happens if I say yes, I want to be in this research?

You will be given instructions and asked to install a plug-in to use in the Eclipse Integrated Development Environment and to complete a short programming assignment. You will receive hints/messages about your code and asked to complete a short survey before and after the assignment.

You are free to decide whether you wish to participate in this study. Instead of being in this research study, your choices may include: ***This study is for extra credit in your course, but it will have no impact on your "standard" grade in the course. You will not have any points deducted if you decide not to participate in the study. If you choose not to participate, another extra credit opportunity will be offered in the course.***

There are no foreseeable risks or discomforts to your participation.

What happens if I say yes, but I change my mind later?

You can leave the research at any time it will not be held against you.

Will being in this study help me in any way?

We cannot promise any benefits to you, but you may receive extra knowledge from the tutoring software regarding a topic presented in computer science courses.

What happens to the information collected for the research?

Efforts will be made to limit the use and disclosure of your personal information, including research study records, to people who have a need to review this information. We cannot promise complete secrecy. Organizations that may inspect and copy your information include the University board that reviews research who want to make sure the researchers are doing their jobs correctly and protecting your information and rights.

The results of this study may be used in reports, presentations, or publications but your name will not be used. Your identity will be separated from your responses by the following:

Data collected will be stored on a secure server with password protection. The graduate student and the PI will have access to the data. It will be stored throughout the duration of the study and for 6 months following the conclusion of the study.

You will be given a participant ID that is collected with the data. Your name will be separated from the data collected so that your responses cannot be traced back to you. This will be implemented with the following procedure:

Random number generator will be used to create a unique ID for each participant. For data analysis, you will enter this number on the surveys and the programming assignment. This allows the researchers to group responses from each participant together for data analysis.

This participant ID will not be recorded on any documents or other materials with the participant's name.

Thus, the data anonymity will be preserved so that the response cannot be traced back to the participant.

Participants' names will be collected for extra credit purposes only. This will be in a secure online survey in which students enter their name. It will contain no language regarding the study or their participant ID.

Who can I talk to?

If you have questions, concerns, or complaints, talk to the research team at ***javiergs@asu.edu.***

This research has been reviewed and approved by the Social Behavioral IRB. You may talk to them at (480) 965-6788 or by email at research.integrity@asu.edu if:

Your questions, concerns, or complaints are not being answered by the research team.
You cannot reach the research team.
You want to talk to someone besides the research team.
You have questions about your rights as a research participant.
You want to get information or provide input about this research.

If you agree to participate in this study, please begin by completing the Pre-Assignment Survey.

APPENDIX E

PRE-ASSIGNMENT SURVEY

Java Recursion Tutor Pre-Assignment Survey

For each question, rate how much you agree with the statement.

* Required

1. **Enter your study participation number (provided to you today). Do NOT enter your ASU ID number.** *

_____

2. **1 = Strongly Disagree, 2 = Disagree, 3 = Agree, 4 = Strongly Agree** *

Mark only one oval per row.

	1	2	3	4
I understand recursion.	◯	◯	◯	◯
I have a lot of prior experience with recursion in Java.	◯	◯	◯	◯
Sometimes, when I am completing programming assignments, I get stuck and don't know what to do next.	◯	◯	◯	◯
When completing programming assignments, I find it difficult to resolve syntax issues.	◯	◯	◯	◯
When completing programming assignments, I find it difficult to understand the concept I am trying to represent in code.	◯	◯	◯	◯
Eclipse is my preferred IDE for completing programming assignments.	◯	◯	◯	◯
I understand how to use Eclipse as my IDE.	◯	◯	◯	◯
Sometimes it is hard for me to understand how to fix the errors reported by the IDE.	◯	◯	◯	◯
I find it helpful to get help (e.g., ask questions) from someone when I get stuck on a programming assignment.	◯	◯	◯	◯
I can easily find someone to answer my questions when I am stuck on a programming assignment.	◯	◯	◯	◯

3. **Which best describes your status as a student?** *

Mark only one oval.

- ◯ First-year undergraduate student in the SER or CS program (Freshman)
- ◯ Second-year undergraduate student in the SER or CS program (Sophomore)
- ◯ Third-year undergraduate student in the SER or CS program (Junior)
- ◯ Fourth-year undergraduate student in the SER or CS program (Senior)
- ◯ Graduate Student
- ◯ Other

4. **Before this course, in how many other courses did you learn about recursion?** *

Mark only one oval.

- ◯ 0
- ◯ 1
- ◯ 2
- ◯ 3 or more

APPENDIX F

PLUG-IN INSTALLATION DIRECTIONS

System Set-up

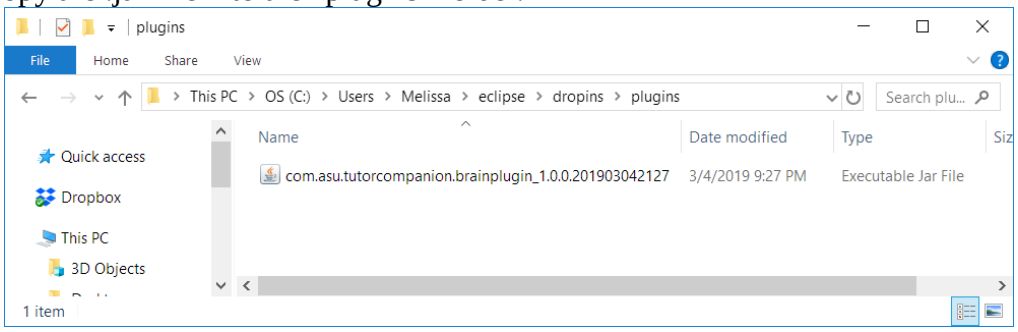
Before installing the plug-in, you need to confirm the following:

1. **The correct version of Eclipse is installed.**
   Eclipse IDE 2018-12, Java EE
   It can be downloaded here: https://www.eclipse.org/downloads/packages/.
2. **Java 1.8 is installed.**
   If you need help determining this, see this article: https://www.java.com/en/download/help/version_manual.xml.
   It can be downloaded here: https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html.

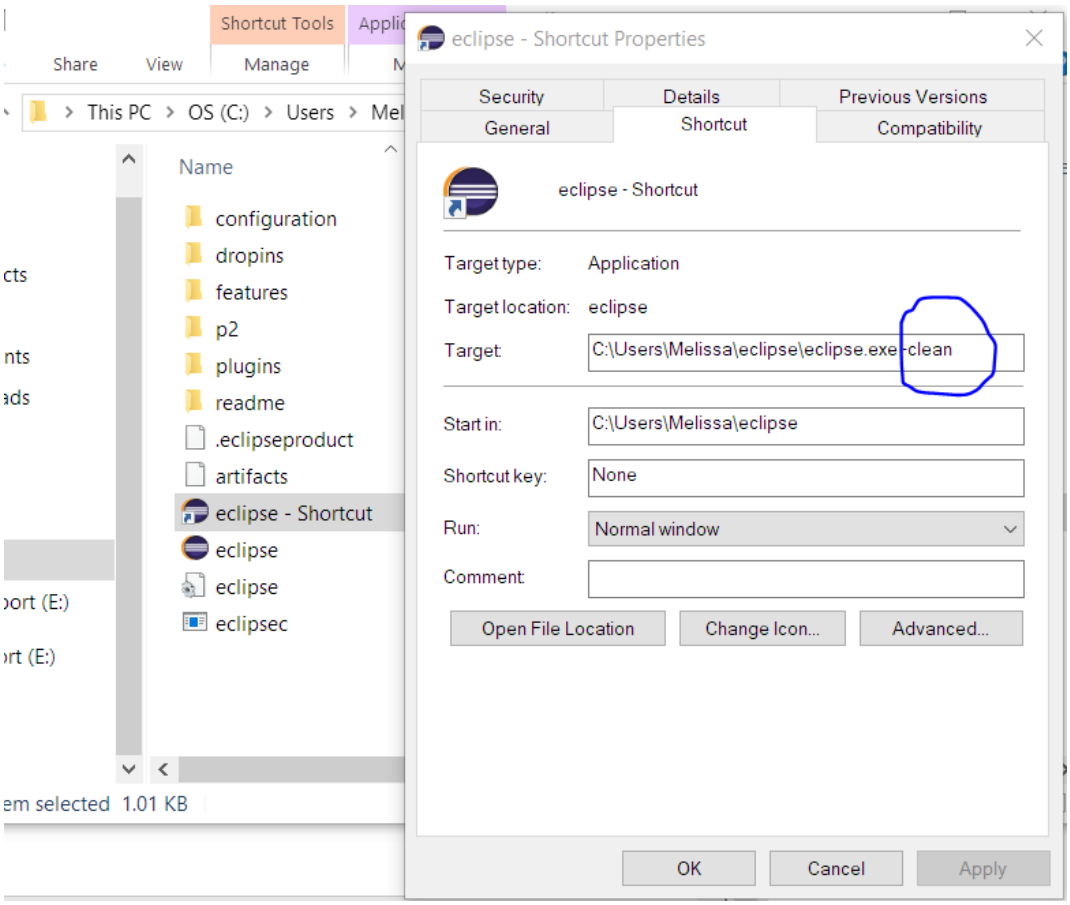
Directions to install plug-in

Windows

1. Go to the location where the **Eclipse 2018-12 Java EE IDE** is extracted in your local machine. In my case, it is:
   C:\Users\<Username>\eclipse
2. Go to
   …\eclipse\2018-12\eclipse\dropins
3. Create a new folder "plugins".
4. Copy the .jar file into the "plugins" folder.



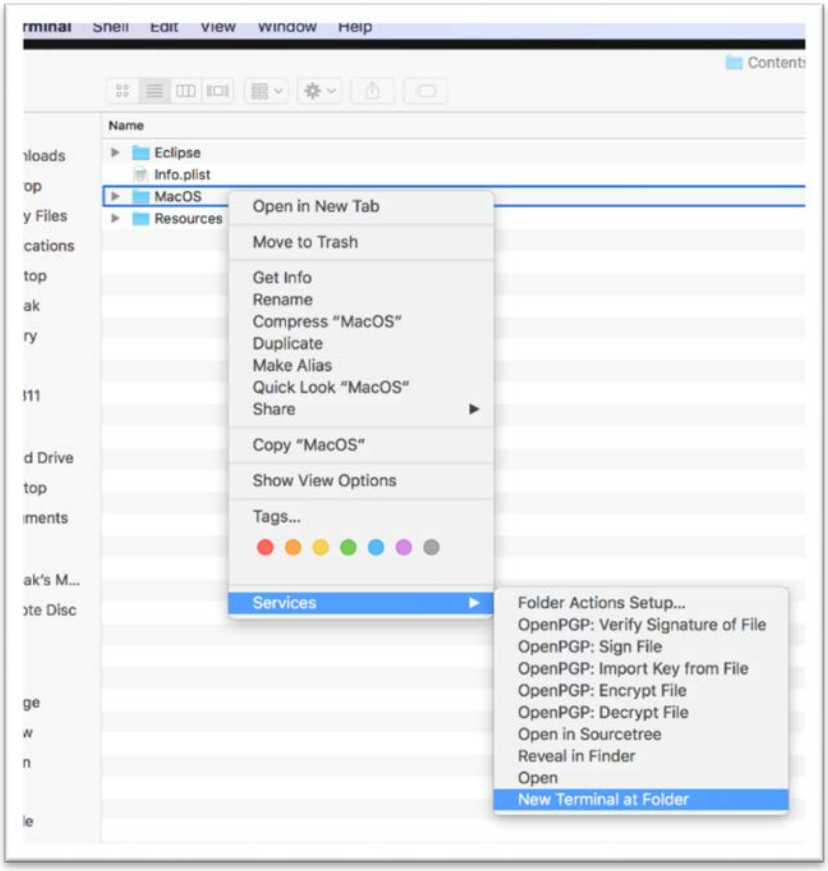
5. Create a shortcut of eclipse.
6. Right click on the shortcut, go to *Properties* → *Shortcut* → *Target*.
7. Add " -clean" (there is a space before -) at the end of the contents of Target. In my case, it looks something like this:
   C:\Users\Melissa\eclipse\eclipse.exe -clean



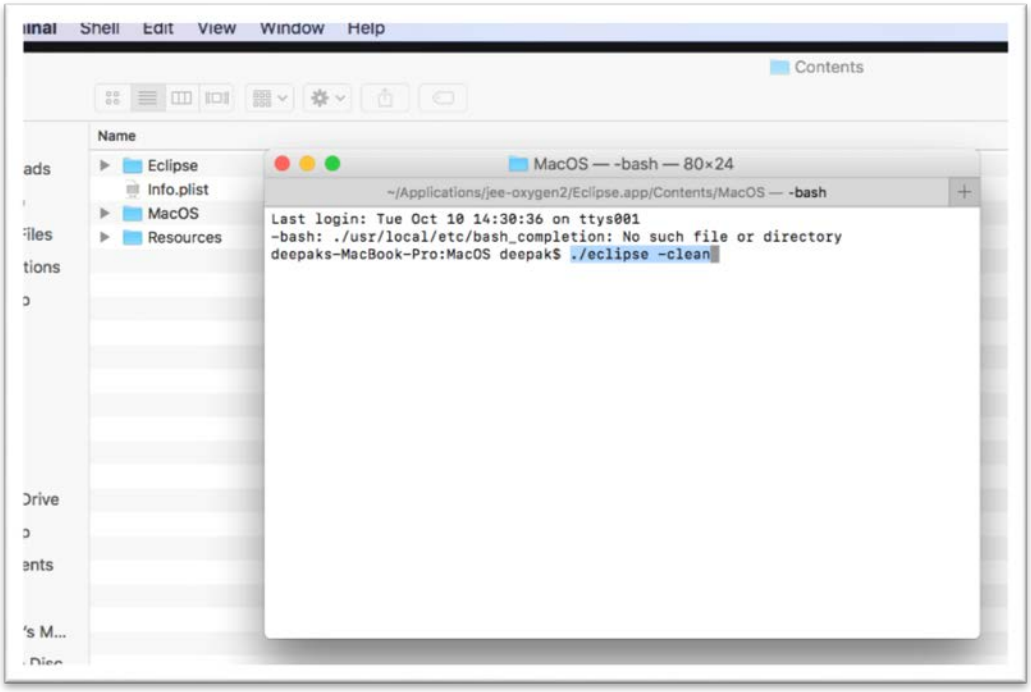
8. Start Eclipse. Eclipse will take a considerable amount of time to launch for the first time (5 – 15 minutes).
9. Once Eclipse starts, it will prompt the user to enter the workspace location. Select a new folder as workspace and continue.
10. Delete the Eclipse shortcut.

Mac

1. Go to the location where the **Eclipse 2018-12 Java EE IDE** is extracted in your local machine. In my case. It is at
   Users → <UserFolder> → Applications → Eclipse (it can be inside a folder like eclipse or 2018-12)
2. Right click on the executable Eclipse file and select “show Package contents”
3. Go to → *Contents* → *Eclipse* → *dropins*
4. The above 3 steps can be represented as below:
   /Applications/Eclipse.app/Contents/Eclipse/dropins
5. Create a new folder “plugins”.
6. Copy the .jar file into the plugins folder.
7. Go to */Applications/Eclipse.app/Contents/MacOS*. Here we should have an executable called eclipse.



8. Clean launch eclipse by running the command below on terminal:
   ./eclipse -clean

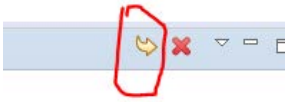


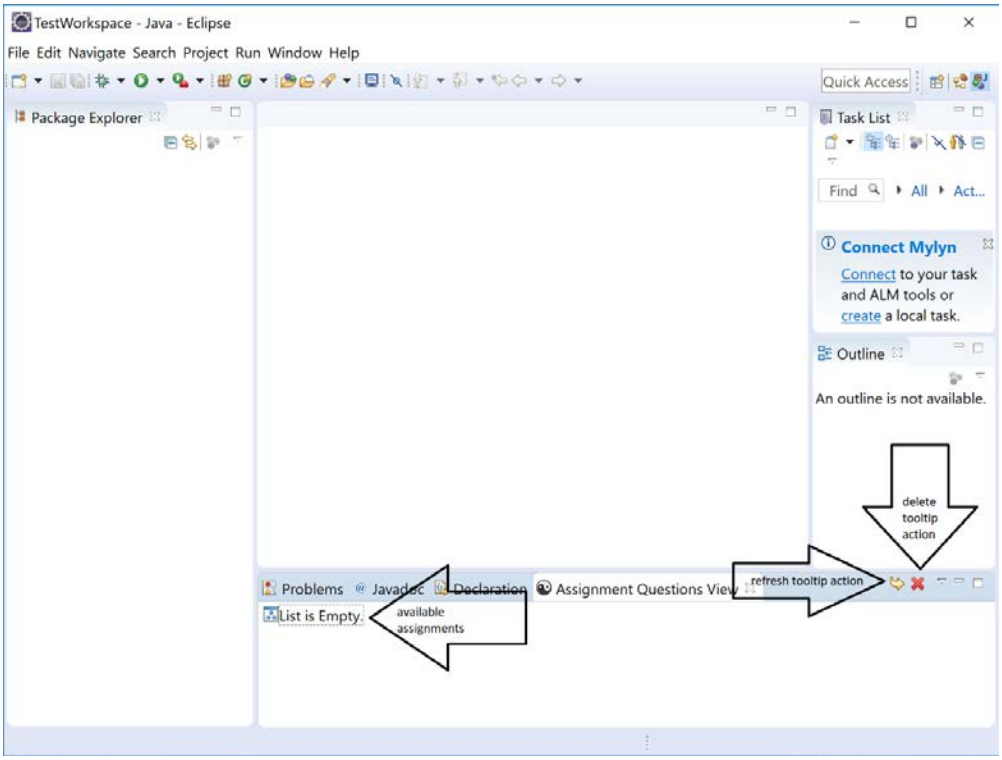
9. Start Eclipse. Eclipse may take some time to launch.

10. Once Eclipse starts, it will prompt the user to enter the workspace location. Select a new workspace for running eclipse.

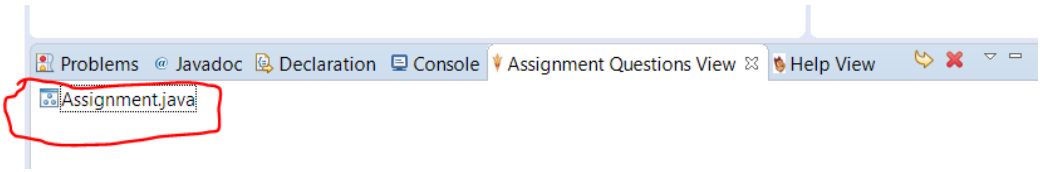
Using the Plug-In

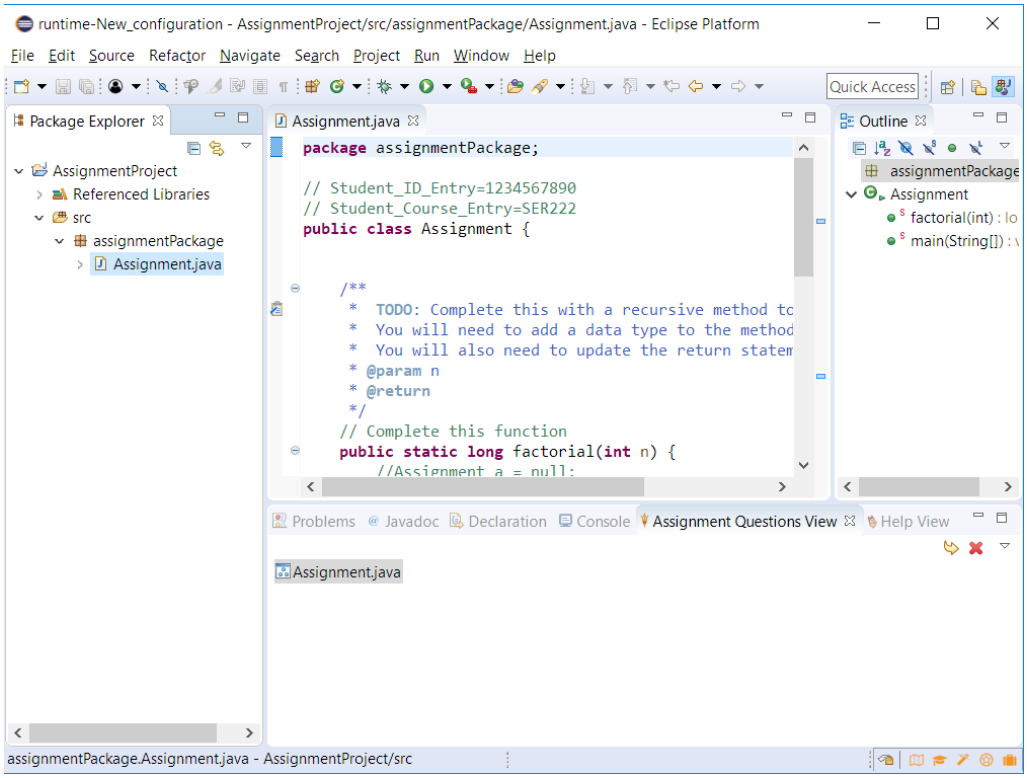
1. To use the plugin, make sure you have continuous internet connection.
2. Start eclipse. Go to *Window → Perspective → OpenPerspective → Java*.
3. If you don't see "Assignment Questions View" in the bottom section of eclipse, then go to
   Window → ShowView → Other → Assignments Category → Assignment Questions View.
4. Select "Assignment Questions View". This should have an "empty list".
5. Click the "Refresh Action Tooltip" (top right corner of the view). This should fetch the list of available assignment(s) from server. The list can be java file(s) and/or text file(s).





6. Open the assignment by double-clicking on it. This will open the Java project in Eclipse in project explorer.





7. Next, you will begin completing the assignment.

Completing the Assignment

Please follow these directions *exactly*.

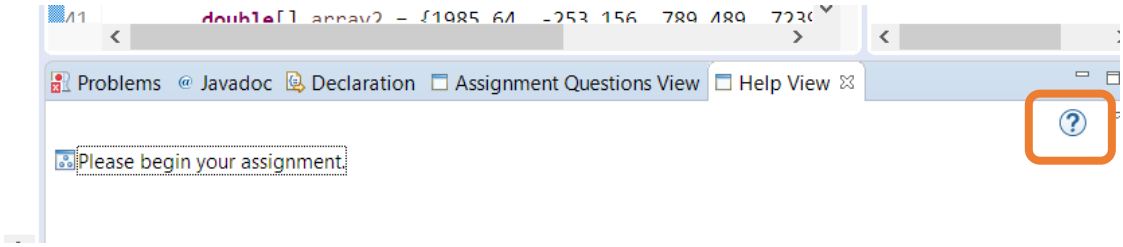
1. Open the Assignment.java file from the project.
2. Add study ID and course number to the comments at the top of the file.
    a. Student_ID_Entry
    Use the random ID assigned to you by the graduate student.
    Do NOT use your student ID number!
    b. Student_Course_Entry
    Complete your course number.
    This will be: “SER222” or “CSE240”. Do not use spaces.

```
2
3
4  //Student_ID_Entry=123456789
5  //Student_Course_Entry=SER222
6
7  public class Assignment {
8      /**
9       *  TODO:
10      *  Use recursion to calculate the sum of the numbers in an array that
11      *  are in a given range of values (inclusive).
12      *
13      *  The method will have 5 parameters, which are given to you.
14      *
15      *  Example:
```

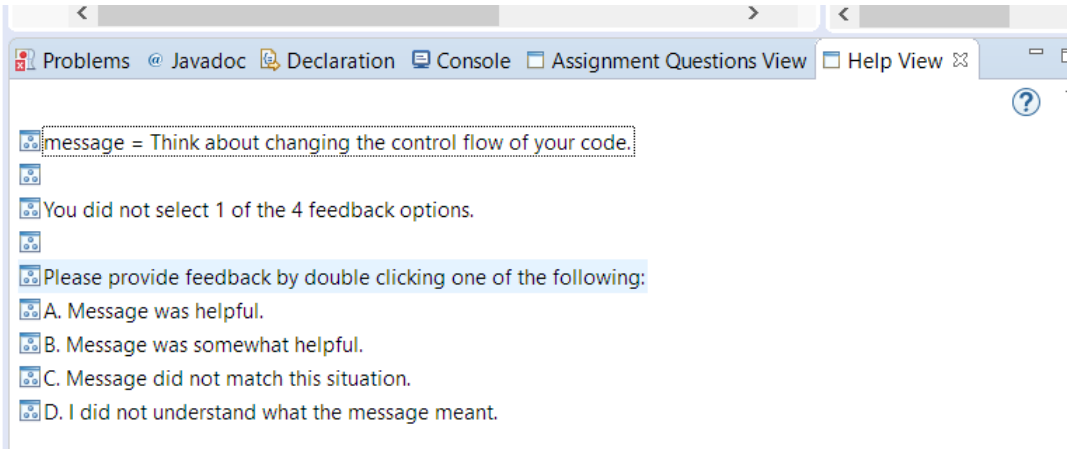
3. Complete the assignment according to the directions.
   Do not remove the comments surrounding the function.

```
28       *
29       */
30      // Complete this function -- Comment not to be removed
31      public static double findSum(double[] array,  n,  start,
32              end,  accumulator) {
33
34
35          }
36      }
37      // End of function -- Comment not to be removed
38
```

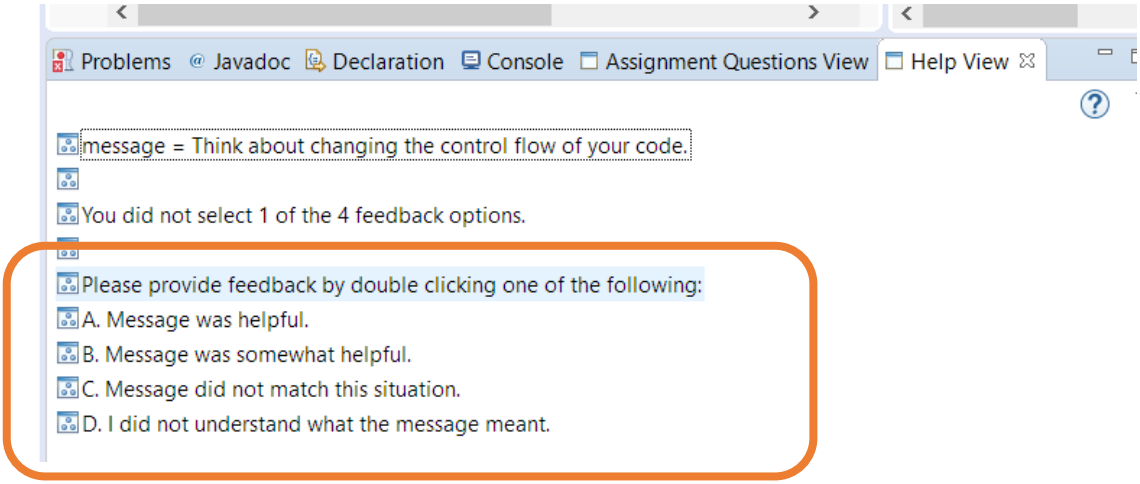
4. Interact with the Tutor Companion through the following actions.
   a. **Run** – Completed as usual
   b. **Debug** – Completed as usual
   c. **Help** – You can receive help by clicking the help icon in the view.



5. After each action, navigate (if needed) to the “Help View” tab to see the message displayed. It will look something like this:



6. After each message received, double-click a feedback option about the message. **Do not proceed until you have given feedback!**



7. Continue until you have completed the assignment. When all tests pass, you should see a "Success" message displayed in the Console.

Uninstalling the plug-in

Windows:

1. Go to the location where the eclipse IDE is extracted in your local machine. In my case. It is at
   C:\Users\<Username>\eclipse\dropins
2. Delete the "plugins" folder.
3. Create a shortcut of eclipse.
4. Right click on the shortcut, go to *Properties* → *Shortcut* → *Target*.

5. Add " -clean" (there is a space before -) at the end of the contents of Target. In my case, it looks something like this:
   C:\Users\manoh\eclipse\jee-neon\eclipse\eclipse.exe -clean
6. Start Eclipse. Eclipse will take a considerable amount of time to launch for the first time (5 – 15 minutes).
7. Once Eclipse starts, you can use it as usual. Delete the Eclipse shortcut.

Mac:

1. Go to the location where the eclipse IDE is extracted in your local machine. In my case. It is at
   Users → <UserFolder> → Applications → Eclipse (it can be inside a folder like eclipse or 2018-12)
2. Right click on the executable Eclipse file and select "show Package contents"
3. Go to → *Contents* → *Eclipse* → *dropins*
4. The above 3 steps can be represented as below:
   /Applications/Eclipse.app/Contents/Eclipse/dropins
5. Delete the "plugins" folder.
6. Go to */Applications/Eclipse.app/Contents/MacOS*. Here we should have an executable called eclipse.
7. Clean launch eclipse by running the below command on terminal:
   ./eclipse -clean
9. Start Eclipse. Eclipse may take some time to launch.
10. Once Eclipse starts, you can use it as usual.

APPENDIX G

RECURSION PROGRAMMING ASSIGNMENT

This programming assignment was used with students in the case study. It was saved on the server. When students used the plug-in, this assignment was retrieved from the server. Students completed the `findSum` method following the directions provided.

```
//Student_ID_Entry=
//Student_Course_Entry=

public class Assignment {
     /**
      *  TODO:
      *   Use recursion to calculate the sum of the numbers in an
array that
      *   are in a given range of values (inclusive).
      *
      *   The method will have 5 parameters, which are given to
you.
      *
      *   Example:
      *   For these values below, your method should return 11.
      *   int[] array = {1, 5, 2, 4},
      *  start = 2
      *  end = 5
      *
      *  1. You must use recursion.
      *  2. You may only use one method (no helper methods).
      *  3. Do NOT remove or change any comments.
      *  4. Do NOT remove any code inside the main method. You may
add more code
      *   inside main, but do not remove any.
      *   5. There are syntax errors with the skeleton code
provided,
      *   which you will need to fix as part of the assignment.
      *  6. Add your student ID number and course name at the top
of the
      *   file in the comment.
      *
      */
     // Complete this function -- This comment not to be removed
     public static int findSum(int[] array,  n,  start,
                 end,  accumulator) {



     }
     // End of function -- This comment not to be removed
```

```
public static void main(String[] args) {
      int[] array1 = {100, 5, 6, 0, -5, 10, 30, -6, 101};

      int[] array2 = {Integer.MAX_VALUE,
            -461, 33, -375, 408, -193, 496, -95, 52, -146,
            284, -153, 80, -203, 245, -472, 98, -228, 379, -
            179, 159, -172, 239, -139, 336, -298, 460, -162,
            390, -174, 304, -409, 330, -240, 135, -137, 111,
            -32, 490, -243, 141, -348, 254, -101, 42, -31,
            176, -58, 37, -123, 298, -97, 452, -87, 399, -293,
            468, -439, 367, -154, 239, -116, 498, -431, 136,
            -155, 146, -438, 106, -474, 369, -408, 72, -368,
            298, -458, 227, -111, 281, -29, 151, -40, 62, -
            153, 248, -42, 42, -339, 127, -187, 496, -380,
            376, -220, 379, -306, 239, -231, 489, -129, 436};

      int[] array3 = {-19, 2, -19, 12, -17, 16, -23, 17, -19,
            8, -5, 5, -5, 3, 0, 8, -10, 12, -3, 12, 0, 10,
            -10, 10, -15, 9, -13, 6, -15, 16, -10, 21, -10, 2,
            -13, 9, -6, 4, -10, 10, -3, 8, -2, 19, -5, 1, -22,
            12, -8, 8, -9, 20, -15, 21, -3, 15, -18, 15, -15,
            16, -6, 10, -4, 16, -18, 8, -11, 14, -18, 11, -10,
            6, -18, 4, -16, 13, -23, 11, -12, 16, -16, 18,
            -24, 18, -8, 21, -7, 8, -24, 20, -18, 17, -15, 22,
            -20, 13, -6, 20, -10, 8};

      if (findSum(array1, array1.length - 1, -5,
            100, 0) == 146) {
            System.out.println("Success!");
      }

      if (findSum(array2, array2.length - 1, -228,
            436, 0) == 5385) {
            System.out.println("Success!");
      }

      if (findSum(array3, array3.length - 1, 5,
            15, 0) == 271) {
            System.out.println("Success!");
      }
   }
}
```

APPENDIX H

POST-ASSIGNMENT SURVEY

Java Recursion Tutor Post-Assignment Survey

For each question, rate how much you agree with the statement. The Tutor Companion is the plug-in that gave messages to you in Eclipse.

* Required

1. **Enter your study participation number (provided to you today). Do NOT enter your ASU ID number.** *

2. **1 = Strongly Disagree, 2 = Disagree, 3 = Agree, 4 = Strongly Agree** *
   Mark only one oval per row.

	1	2	3	4
The Companion improved my understanding of recursion.	◯	◯	◯	◯
The messages from the Companion were helpful to me while completing the assignment.	◯	◯	◯	◯
The Companion helped me feel supported while completing the assignment.	◯	◯	◯	◯
I liked working with the Companion there to help me during the assignment.	◯	◯	◯	◯

3. **1 = Almost Never, 2 = Occasionally, 3 = Sometimes, 4 = Most of the Time** *
   Mark only one oval per row.

	1	2	3	4
The messages from the Companion were logical, given what was happening with my code.	◯	◯	◯	◯
The messages from the Companion gave me hints about how to proceed with the assignment.	◯	◯	◯	◯
The messages from the Companion gave me hints about concepts needed for the assignment.	◯	◯	◯	◯
The messages from the Companion gave me hints about syntax.	◯	◯	◯	◯
The Companion seemed "intelligent" to me during the assignment.	◯	◯	◯	◯

4. **What would make the messages from the Companion more helpful to you? (Choose all that apply.)** *
Check all that apply.

- [ ] Different messages (i.e. Change the comments to something different)
- [ ] The frequency of the messages (i.e. How often they're given to you)
- [ ] The timing of the messages (i.e. Giving them to you at a different point in your programming)
- [ ] Other: _____

5. **What would improve the frequency or timing of the messages? (Choose all that apply.)** *
Check all that apply.

- [ ] Give messages to me only when I have errors.
- [ ] Give messages to me only when I request it.
- [ ] Give messages to me only after a certain time interval has passed.
- [ ] Other: _____

6. **What were some of the problems you noticed with the messages? (Choose all that apply.)** *
Check all that apply.

- [ ] The message did not fit the situation.
- [ ] I did not understand the meaning of the message.
- [ ] I got the same message too often.
- [ ] The message did not help me fix my problem.
- [ ] Other: _____

7. **Which of these statement were TRUE while interacting with the Companion? (Choose all that apply.)** *
Check all that apply.

- [ ] The Companion gave me help when I did not need it.
- [ ] The "usability" of the software for the Companion detracted from my experience with the Companion.
- [ ] I like the idea of having a Companion during programming assignments.

8. **Which BEST summarizes your experience with the Companion?** *
Mark only one oval.

- ◯ I already understood recursion and did not need help with the assignment. Therefore, the Companion's messages were unwanted and not needed.
- ◯ I needed help with the assignment, but the Companion's messages did not help me.
- ◯ The Companion's message could help me if they were improved.
- ◯ The Companion's messages helped me.

9. **What changes to the messages would help improve them?** *

_____