Virtual Teaching Assistant to Support Students' Efforts in Programming

Mukund Babu Manniam Rajagopal

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Computer Science and Applications

Stephen H. Edwards, Chair
Donald Scott McCrickard
Francisco Javier Servant Cortes

May 25, 2018
Blacksburg, Virginia

Keywords: Pedagogical Agents, Virtual Teaching Assistant, Computer Science Education

Virtual Teaching Assistant to Support Students' Efforts in Programming

Mukund Babu Manniam Rajagopal

ABSTRACT

Novice programmers often find learning programming difficult. They suffer from various misconceptions and difficulties in understanding the subject. The overall experience with programming can be negative for many students. They may feel isolated in the programming environment and think that programming is difficult for them.

Many schools use automated grading tools to process student work and provide them with early feedback. Web-CAT, an open-source software system that is widely used by many universities, is an example of such an automated grading tool. We have developed a Virtual Teaching Assistant for Web-CAT, called Maria, who can support the students to help alleviate some of the negative emotions towards programming. We have used an animated pedagogical agent as the virtual assistant as certain characteristics of the agent can help with the students' perception about the virtual teaching assistant.

Often, students have a fixed mindset about programming. But it is easy to master programming with practice. To promote a growth mindset, Maria also provides feedback recognizing the effort of the student in addition to the performance-oriented feedback of the students' programs. Maria can also provide motivating or encouraging comments to continue working on the assignment to get a good score. Maria can also provide information about the various errors displayed in student feedback.

Virtual Teaching Assistant to Support Students' Efforts in Programming

Mukund Babu Manniam Rajagopal

ABSTRACT (GENERAL AUDIENCE)

Beginners often find learning computer programming difficult. They may suffer from various misconceptions and difficulties in understanding the subject. Also, there can be a negative experience surrounding programming for many students. They may feel isolated in the programming environment and think that programming is difficult for them.

Many schools use automated software tools to grade student programs and provide them with early feedback. Web-CAT, a software system that is widely used by many universities, is an example of such an automated grading tool. We have developed a Virtual Teaching Assistant to reside within Web-CAT, called Maria, who can support the students to help alleviate some of the negative emotions towards programming. We have used an animated human-like character, known as pedagogical agent, for Maria as it is widely use in pedagogy to help students.

Often, students think programming is an innate skill and it is difficult to acquire. But it is easy to master programming with practice. To encourage students to continue working, Maria also provides feedback recognizing the effort that the student has put in towards completing the programming assignment or project. In certain cases, Maria can also provide motivating or encouraging comments to the students to help them continue working on the assignment. Maria can also provide explanation about the various programming errors that students encounter during their submission to Web-CAT.

Acknowledgments

I have to thank lot of people who have helped me and guided me in completing this master's degree and without whom 2 years of grad life would not have been possible.

First of all, I would like to thank Dr. Edwards who is the nicest professor I have ever seen. He has been a great mentor to me and without him 2 years of Master's would not have been possible for me. He is also the most technically skilled professor I have seen - I could go to him with any queries and he w ill solve them all. It has been a great learning experience for me. Thank you also for providing the financial support throughout my Master's journey.

I would also like to thank Dr. McCrickard and Dr. Servant for agreeing to be on my committee and carefully reviewing my work providing inputs, feedbacks and suggestions.

I am forever indebted to Nischel for always being there for me and continuously pushing me in all regards. Without you, I would not have got through grad school. I enjoyed all our conversations about life, jobs etc. Thanks for also contributing to my thesis work.

I would like to thank my family for supporting my decision to pursue Master's and continuously backing me. I would also like to thank all my nice roommates in Blacksburg, Shreyas, Radhakrishnan, and Kanagaraj for all the fun, support and guidance.

This work is supported in part by the National Science Foundation under grant DUE-1625425. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we provide a brief overview about our system—Web-CAT and the motivation for having a virtual teaching assistant in Web-CAT. We provide a background about pedagogical agents that we have modeled as a teaching assistant in Web-CAT. We describe the various problems that we try to address by the use of an animated pedagogical agent as a teaching assistant.

1.1 Web-CAT

Web-CAT, a Web-based Center for Automated Grading, is a fully open-source tool that is used to automatically grade programming assignments [20, 21] . It is a very widely used grading tool across the world, used by over 90 different universities. Web-CAT is a plug-in based software that processes assignment and provides feedback and grades[22]. Currently, these plug-ins allow for the grading of assignments written using programming languages like Java, Python and C++. In addition to the programming language support, few plug-ins allow to perform static analysis of student programs to check for coding style and adherence to coding standards. These plug-ins also allow us to dynamically check the runtime behavior

of the student programs and enables us to execute and analyze student written software tests.

Web-CAT is very flexible for the instructors to choose their own grading policy. It encourages the students to test their code well by grading the students unit tests as well, as this has shown to increase the proficiency of student programming and reduction of bugs in their code. However, the instructor can choose to omit grading the students unit tests in their grading approach. Apart from automatic grading, Web-CAT also allows for instructors to manually grade students work. The instructor can select the grading policy by balancing the manual grading and the automatic grading.

Students can submit their assignments to Web-CAT using plug-ins developed for the various IDEs they use for programming or they can manually upload it through the web. Once the assignment is submitted, the plug-ins compile the student program, run the program against instructor provided reference tests, performs static analyses and provides feedback about the assignment. If the students assignment does not produce correct output, or if the students unit tests are not as per expectations or if any of the static analyses checks fail, Web-CAT provides intelligent feedback to the student detailing the issues with the students assignment.

1.2 Motivation

In the United States, the projected need for computer and information science professional is much more than the number of graduates in these fields. Programming is an important part of the curriculum in Computer Science education. It enables students to be successful in their Information Technology careers. Nowadays, computer programming has also become an integral part of other fields of science like physics, biology, medicine etc. Programming is a process of converting a problem into a computer executable program. Naturally it involves analyzing the problem at hand, developing an algorithm to solve the problem, verification of the algorithm to check for its correctness and resource consumption, knowledge of program-

ming language concepts and constructs and finally implementation of the algorithm using a programming language. Because of this growing demand for computer science professionals, teachers are required to help students develop accurate understanding of programming language.

Learning computer science is difficult as evidenced by the fact that it has some of the highest attrition rates among undergraduate majors in the US [27]. A major contributing factor to this attrition is computer programming courses. Programming is a difficult subject for many students as it requires correct understanding of many abstract concepts. Winslow [50] suggests that it takes about ten years for a novice student to become an expert in programming. Also, there are not a lot of resources that can help the students and often they lack personal instruction. Students usually demonstrate misconception of fundamental concepts of programming and difficulties in syntactic and strategic knowledge.

Given these difficulties, students have a general negative affect while learning programming. In fact, Bosch et al [13] in their study found that 71% of students display negative emotions when learning programming for the first time. Students often feel isolated in the learning environment and think that they do not belong there. Many of the introductory programming courses are offered in Java. Our college offers a CS1 course in Java as well. Students taking this course are required to complete lab exercises, assignments and projects and submit their work to Web-CAT for grading. So Web-CAT is an important part of the learning process in programming. The feedback from Web-CAT contains a list of errors and failures in students' programs. Bandura [8] theorized that the students tie the feedback about the performance outcome directly to their perceived efficacy. Senko and Harackiewicz [43] found that the negative feedback can influence the student's perceived competence. Web-CAT's feedback of insufficiencies in student's program can lead to a student having a negative affect and this contributes to the overall negative emotions associated with learning programming. Since students associate the negative feedback to their competence level, negative feedback can discourage the students from learning programming and lead to either dropping the course or change majors. Currently, there is no way for Web-CAT to acknowledge these negative

emotions and influence student's affect. But it is important to address this problem to create a positive learning environment for the students. This work focuses on providing support to address the negative emotions and this support can range from helping students understand errors or provide them with motivation or even just a relatable teaching assistant to talk to. This quest gave birth to the idea of developing a virtual teaching assistant for Web-CAT.

1.3 Virtual Teaching Assistant

Instructors and teaching assistants play an important role in any CS1 and CS2 programming courses. They provide the instructional support necessary for students and also address the negative emotional state of the students sometimes. Given the volume of students in programming courses, it is almost impossible for the instructors and TAs to provide individual attention to students supporting them emotionally to create a positive experience. Also while still in the process of learning programming, students usually encounter a lot of errors - both syntactically and conceptually. This can add to the negative experience and at the same time students might perceive these errors in their programs as their incompetency. Now these errors can be attributed to a lot of factors like misconception, lack of proper understanding of programming constructs, lack of self-belief, etc. Based on the type of error they encounter, students seek help of various sources. In case of a compiler error, for clarifying a syntax or certain types of runtime errors, few of the students use various resources available on the web to learn more whereas most of the students solicit the help of a teaching assistant or an instructor to clarify the programming concepts. For errors relating to behavior of their program, students always elicit the help of a human teaching assistant or an instructor to examine the code and provide pointers to fixing the errors. To overcome the negative emotions and to resolve the errors, students interacting with the instructor or a teaching assistant is ideal, but it is not possible to have a teaching assistant or the instructor available at all times to the students. Hence, we developed the idea of developing a virtual teaching assistant for Web-CAT that can provide the emotional support and support with

error explanations when students are interacting with Web-CAT.

Another problem with programming exercises in computer science is with the mindset of students. Students working on their programming assignments and submitting their work to Web-CAT can see static performance-oriented feedback that can easily induce a fixed mindset among students. It can definitely have a negative influence on learning where students believe programming is a skill that they either posses or they do not. The alternative to this is the concept of a growth mindset where students possess the belief that they can improve their skills with constant effort and hard work. An empathetic human teaching assistant can reinforce the growth mindset among students as they know what it requires hard work and consistent effort to learn programming successfully. We have devised twelve performance indicators that helps us devise feedback strategies that can reward hard work and effort and can be delivered to students during submission of their work to Web-CAT. This rewarding feedback can be delivered by a virtual teaching assistant to be more effective among students.

Many a times students get a zero on Web-CAT's evaluation of their program submission due a variety of errors. Other times, students are unable to make sufficient progress in their assignment with Web-CAT's feedback showing a recurring error or the feedback containing a plethora of errors. This can be very daunting to students, especially to the students who are new to programming. In such scenarios, when students are working with a teaching assistant, they can quickly identify students losing motivation or getting frustrated and provide motivation or encouragement to the students to keep continuing. We wish to replicate the same behavior in Web-CAT by having the virtual teaching assistant identify scenarios where students get flustered and providing sympathetic comments to help them continue working on the assignment.

Finally, we are using an animated pedagogical agent as a virtual teaching assistant in Web-CAT. Animated pedagogical agents are widely used in online learning environments these days. They are characters that simulate human behavior—they can have gestures, they can have voice and they can look like humans. These animated pedagogical agents have been

proven to be effective in pedagogy if designed carefully. Also, we call our virtual teaching assistant **Maria**.

1.3.1 A Support Agent

In many learning environments, pedagogical agents are used as a learning agent where the agent facilitates the learning. Pedagogical agents are integral to the learning activity when used as a learning agent. In most cases, the learning agent spells out the task involved in learning, interacts with the students and provide feedback about their work. But in our work, we are using pedagogical agent as a support agent. Support agent helps with tasks that are peripheral to the learning process like the emotions involved in learning. Through our support agent, we would like to address the negative emotions associated with learning programming by providing reinforcing or encouraging feedback about students' efforts and sympathizing with students when they are frustrated.

1.4 Problem Statement

A primary goal of this project is to provide adequate emotional support to students to enable them assuage some of the negative emotions surrounding Java programming assignments and projects. We want to support the students in ways that a human teaching assistant would be able to. We believe current Web-CAT system provides useful feedback to students but lack resources to provide emotional support to influence the affect of the students. We have identified the following problems in Web-CAT with respect to lack of emotional support to students in their assignments:

1. **Minimal Web-CAT support:** Web-CAT is an important part of the learning process as it is used for grading the assignments and projects. Web-CAT also contributes to the negative emotions associated with the learning process but currently has no way

of addressing these negative emotions or influencing the student mindset while using Web-CAT.

2. **Limited access to TAs:** Instructors and teaching assistants provide the instructional support and sometimes provide the emotional support required for students. But they may not be available all the time to students to provide this emotional support.

3. **A fixed mindset of students:** The performance-oriented objective of Web-CAT grades the students' assignments and displays a feedback that contains a score based on the correctness, unit testing and adherence to coding standards. This can easily induce a fixed mindset among students, which is detrimental to learning programming.

4. **Student frustrations:** Students often become frustrated if they get a low score or if they are unable to make progress even after hard word and effort. They might need some encouraging or motivating comments to keep working and get a good score but the current system does not have provisions for that.

Key design aspects of our research that will address the above mentioned problems are:

- **A relatable virtual teaching assistant:** The animated pedagogical agent that is used as a virtual teaching assistant in Web-CAT can be designed in such a way that students perceive the agent to be very relatable so that students can receive the support provided by the virtual teaching assistant.

- **Explain all errors:** We have provided links next to errors displayed on the feedback page in Web-CAT that can provide explanation of specific errors. This can be very helpful to the students as the help is available to them very handy. We have equipped the teaching assistant to be able to provide students with a description of cause of various Java compiler errors, runtime errors and static analysis errors. In addition to the description, the virtual teaching assistant can also provide examples to fix the errors, in certain cases.

- **Promote a growth mindset:** As mentioned before, a fixed mindset can be harmful to the learning process and we should foster a growth mindset among students. We believe appreciating students work and effort, outside of the performance goals set by the instructors for an assignment, will help in developing a growth mindset among students.

- **Provide sympathetic support:** We can use the virtual teaching assistant to also provide sympathetic comments to students finding it difficult to make progress in their assignment. This will help them to continue putting in the effort on the assignment.

1.5 Outline

In the further chapters, we describe the learning theories and design techniques used to implement pedagogical agents (Chapter 2). The we describe the literature review (Chapter 3) surrounding the work we are doing. We move on to detail the design goals and decisions taken (Chapter 4) to complete this thesis and then outline the implementation of this design (Chapter 5). We also provide a walk through (Chapter 6) of the scenarios that the students will encounter while using Web-CAT. In Chapter 7, we describe the evaluation and end the thesis by detailing our conclusions and future work in Chapter 8.

Chapter 2

Maria: A Virtual Teaching Assistant

Web-CAT is an automated grading tool that grades the students' assignments on the correctness, styling, best practices in coding and also on how well they test their own code. When students submit their code to Web-CAT for grading, the tool assesses the code and provides students with a comprehensive summary of errors in their program, if any. There is an overall negative feeling among students while learning programming and Web-CAT's feedback only enhances this negative experience and currently there is no way of handling it. The primary goal of this project is to provide adequate emotional support to students to enable them to overcome the negative experience with programming. This emotional support includes providing useful feedback to students about their progress/effort in the assignment, providing motivating or encouraging comments through the submission process and helping students to resolve errors that arise from their program submission to Web-CAT. Typically, students seek the help of a teaching assistant or an instructor to resolve their persistent errors or expect them to provide emotional support if they are unable to make significant progress in the assignment. But the teaching assistant or the instructor may not always be available. So we want to provide continuous support to students to help with their programming assignments. This idea gave birth to Maria—the virtual teaching assistant of Web-CAT.

2.1 Pedagogical Agent Based Teaching Assistant

The process of learning and pedagogy is a highly interactive process between the learner and the teacher[30]. Distinguished psychologists have postulated that social interaction is key to learning and development [49, 8]. Pedagogical agents are animated characters designed to simulate human-like interface to facilitate interaction with students in a pedagogical environment. These pedagogical agents are widely used in pedagogy mainly in Intelligent Tutoring Systems. What makes these pedagogical agents unique in these learning environments is their ability to simulate social interaction. In learning environments such as the Intelligent Tutoring Systems, the learning content is delivered to the learner by interacting with pedagogical agents that are coded to provided such information and also to provide motivation to the learner. In traditional computer-based learning systems, the focus is mainly on the educational content to meet the objectives of the learning or the learner. The social interaction aspect of the learning process, demonstrated as significant in the learning process, is missing in these traditional learning environments. The use of pedagogical agents can help immensely with social interaction with the learner.

Various empirical studies show that peer interaction is valuable to the learning process and the motivation of the learner. Learners are able to learn a lot just by observing the behaviors of their peers. Various theories state that when learners are exposed to people of various capabilities performing the same learning task, it improves their intellectual ability and their ability to think differently [35]. Peer interaction always provides a mechanism for free flowing of ideas thus impacting the learning capability [16]. In Web-CAT, it is difficult to promote peer learning since students work on their assignments individually. Although, they may seek the help of a teaching assistant or an instructor sometimes who provide a different way of approaching a problem or an error, it is nearly impossible to provide this peer support all the time. The use of pedagogical agents can help with the peer interaction as we intend to use the pedagogical agents to provide pointers to fixing errors that can help students think differently while fixing errors.

Embodied cognition is another area of research that justifies the use of pedagogical agents in pedagogy. This theory proposes that bodily movements have an impact on learning. Embodied research on perspective by Tversky et. al [46] in their work found that when students are showed a picture with a person in it, they are more likely to think from the perspective of the person in the picture. The use of a life-like animated agent with facial expressions, will convey different sensorimotor information to the learner and will improve the learning process. This also justifies our use of animated agents for virtual teaching assistant as a smart choice of the agents will enable students perceive as a teaching assistant and will connect to it as they do with a physical teaching assistant. Another research on memory [42] suggest that people are likely to remember the gist of a story if it is acted out than when the information is provided in the form a plain text.

Based on the above mentioned points of view about learning in web based applications, it seems justified to use an animated pedagogical agents as a virtual teaching assistant in Web-CAT.

2.2 Learning Theories Modeling Pedagogical Agents

Traditionally, the pedagogical agents are often modeled after following different learning theories[4, 30]:

2.2.1 Distributed Cognition Theory

This theory proposes that knowledge is not just confined to one individual rather it is distributed among individuals and things in the society. Traditionally, it has been perceived that human mind works individually during cognitive activities but various studies have proved that it is not the case. Leveraging this concept, the pedagogical agent can be used as one of the tools that humans collaborate during the learning activity where it assumes the

role of an information warehouse. In this case, the pedagogical agents act as the artifacts possessing knowledge that the user do not possess, so that the users can interact with the pedagogical agents to learn things that they do not know. This way, the pedagogical agents can build a partnership with the learner that will enhance the learning process.

2.2.2 Socio-Cultural Learning Theory

As mentioned before, learning is a highly social process. Cognitive development happens when the learner interacts with agents during a learning activity [49]. In fact, the development of the thinking process happens by observing or interacting with the social cues. There is always an emotional experience associated with any learning event. Interaction with the social environment can invoke a variety of emotions from the learner: curiosity, anxiety, confusion, frustration, encouragement, motivation. All these emotional states can affect the learner and have an impact on learning [5]. Based on this, the pedagogical agent can be used to address the emotions and the states of the learner to build an affective relationship [8]. Following this line of thought, we can build the pedagogical agent to have a positive influence on the learner by motivating and encouraging the learner in necessary scenarios.

2.2.3 Social Learning Theory

Human learning happens in a social context where humans learn by observing and imitating others in the society. In Bandura's work [8], it is proposed that surrounding social agents have an impact on how humans shape their attitudes, goals and behavior. This gives immense power to design a pedagogical agent that will affect human behavior. This theory proposes the different roles that the pedagogical agent can assume:

1. **Personal agent** where the agent responds to learner's requests,

2. **Social Modeling** where the agents share attributes of the learner and transfer knowl-

edge and skills,

3. **Collaborative agency** where there are multiple agents assigned specific roles combined to execute a learning task.

Vygotsky [49] introduced a theory called zone of proximal development that mentions that when students collaborate with others more capable than themselves, it has a greater impact on their intellectual development. Using this theory, Greenfield [26] suggested that a well-designed agent with a higher intellectual state than students could simulate important characteristics of scaffolding.

Based on all the learning theories mentioned above, we designed Maria to act as a partner to the learners responding to requests from learner. For our work to be successful, it is important that appearance of Maria should be similar to that of the learners as various studies prove that people tend to learn more from people that are similar to themselves. We also integrated the ability to emotionally support and motivate the students into Maria to positively affect the learner's ability.

2.3 Design Constituents of Pedagogical Agents

Baylor [30] in her work proposed that there are seven design constituents to instructional pedagogical agents: (1) competency, (2) interaction type, (3) gender, (4) affect, (5) ethnicity, (6) multiplicity, and (7) feedback.

Competency: The designer has the flexibility of having various levels of competency of a pedagogical agent. It can vary from the pedagogical agent being an "expert" disseminating knowledge to the learner to playing a support role in certain learning tasks. According to Bandura [9], having a high competency agent might affect the cognitive development of a learner as they might rely highly on the agent. On the other hand, a pedagogical agent which has a low-to-moderate cognitive ability could affect the learner's cognitive conflict by

viewing the agent as a mere partner in the learning process. Baylor argues that competency of a pedagogical agent is the most researched topic and there is further scope for identifying how the competency of an agent affect the learning process.

Interaction Type: The interaction type has two dimensions: 1) the control of the interaction and 2) content of the interaction. The interaction control indicates the direction from which the interaction is initiated. As with the social learning theory we have seen before, the personal agent role can warrant the learner to initiate the interaction out of self-motivation to learn and the agent can just support the learner. Other type of interaction is where the agent itself initiates the contact in an effort to simulate social interaction. Regarding the content, the designer can use various techniques to disseminate information: confirmation, argument, suggestion and questioning to improve the inquisitiveness of the learner.

Gender: Pedagogical agents with friendly demeanors can help improve the learning process by reducing the anxiety and improving the motivation among learners. Kim [29] researched the effects of the gender of pedagogical agents on college student's social judgment, motivation and learning. In their study, they found that both male and female students perceived a male pedagogical agent more favorably than a female pedagogical agent. They also found that students were able to recall more after their interaction with a male pedagogical agent than a female pedagogical agent. Moreno's work [36] showed that undergraduate students applied gender stereotypes to pedagogical agents and it influenced their learning.

Affect: Pedagogical agents having an affective relationship with the students are perceived to be more interesting and engaging [38, 24]. A pedagogical agent displaying positive emotions can help the students have a positive affect as well. The affect of a pedagogical agent can be perceived in two ways: the agent recognizing the students affect and responding accordingly and the affect expressed by the agent itself. Since it is difficult to capture the affect of the students through a computer, we will not focus on this aspect of affect of the pedagogical agent. However, there is significant evidence that a pedagogical agent expressing a positive is well perceived by students and known to keep the students motivated in the

learning task.

Ethnicity: As we have seen before, pedagogical agents with similar attributes are able to relate well with the students. One attribute of similarity of agents with the student is the ethnicity and Bandura's [8] work shows that agents of same ethnicity appear to be credible and instill a lot of confidence among the students. Baylor and Kim [12] conducted a study with agents belonging to multiple ethnicities and found that students perceived agents belonging to their own ethnicity as more credible and more affable than agents belonging to other ethnicities. In contrast, Moreno [36] found that the ethnicity of an agent does not have any effect with the students.

Multiplicity: Multiplicity refers to use of multiple agents to induce a stronger sense of social interaction in a learning task. Bandura [8] in his work argues that using multiple agents performing multiple roles are better than using a single agent carrying out a wide variety of tasks. This provides a flexibility for students where they can interact with an agent of their choice and comfort.

Feedback: Effective feedback is an essential part in any learning process. Effective feedback can promote a growth mindset among students that will motivate the learners and promotes cognitive growth. Providing verbal messages assuring that students can do well in the learning task encourages them to spend more effort in the task and sustain them [8]. Multiple factors are to be considered when designing effective feedback--timing, scope, learning goals and feedback type.

2.4 Conclusion

In this chapter, we provided some background on the goal of this project, the need for a virtual teaching assistant and use of an animated pedagogical agent for the virtual teaching assistant. We also looked at the various learning theories used to develop these pedagogical agents and also the various attributes to keep in mind while designing a pedagogical agent.

Chapter 3

Literature Review

In this chapter, we will provide a literature review of the research surrounding the novice programmers and the difficulties they face dealing with programming errors. We will also provide an overview of current use of pedagogical agents in pedagogy and finally discuss about the existing use of pedagogical agents to provide motivation during the learning process.

3.1 Novice Programmer Difficulties

There has been significant research focused on students new to programming. A few topics of these researches include identifying the difficulties that a novice programmer (students) face, the types of errors their programs contain, the causes for these errors, common misconceptions about programming concepts and constructs etc.

One of the early works that provides a review of research relating to programming was done by Robins et. al [41]. They performed an educational study comparing the trends of novice and expert programmers. In their work, they find that the novice programmers find it very difficult to approach a programming task because they have many deficits in both knowledge and strategies. Although they learn these knowledge and strategies, they

are often very fragile. They identify specific language features such as loops, conditionals, arrays and recursion that are very problematic to students. A deeper problem is the issue of problem design, algorithm and actually coding the algorithm as a working program. Practical laboratory sessions and assignments can help students develop these traits but given the students deficit in understanding of language features, the instructors need to place emphasis on the course design and structure.

In another study conducted by Winslow [50], he outlined that novice programmers have a very shallow knowledge of the subject, and even when they have knowledge of the subject they fail to use it. They also use a line-by-line bottom up approach to solving a problem rather than using meaningful control structures. He claims that an instructor should teach basic paradigms of design very well in order for the student to be able to successfully combine operations into meaningful units of a program. He claims that to achieve the level of a proficient programmer, students should practice a lot mastering the basic facts, features and rules of programming.

Kurvinen et. al [32] conducted a study about how programming is difficult for students and how they encounter the same misconceptions about basic concepts in Java programming. They created a tool called ViLLE, which was used by students to complete programming exercises. These programs were then automatically assessed and students were provided feedback with the compiler output and error messages. Students participated in a 7-week long programming course and the tool collected all the students programs from the beginning to the end. They divided the students into three groups novice, average and advanced and they found that all the students made mistakes in all of the assignments and suffer from misconceptions in programming. Students who performed well were able to surmount these misconceptions with the help of just compiler and ViLLE output but not all students were able to, suggesting a need for external help to overcome the errors they encounter in their programs.

Pillay and Jugoo [40] aimed to identify the causes of errors made by novice programmers

in introductory Java courses at two South African tertiary universities. The course had a focus on variables, data types, simple input and output, selection, repetition and nested repetition. They collected the data from practical lab sessions and practical exams to analyze the student errors. They also analyzed the mental models of the programming constructs as these models give an insight into students functional knowledge of the construct. After carefully analyzing the errors and misconceptions of the students, they categorized the errors into 8 different categories. They established that the main causes of the errors observed were due to incorrect transfer of knowledge and lack of understanding of various concepts.

In his work, Traver [45] discussed about the difficulties with compiler error messages. He argues that compiler error messages are very cryptic, difficult to understand, resolve and prevent from occurring in the future. There is a lack of computer support to understand these errors and not a lot of work is done by the compiler developers to make it easy for the programmers to understand these errors. Research work done on the compilers include developing new compilers for new programming languages, optimizing the existing compilers for time and memory etc. but there is not a lot of work done on enhancing compiler error messages. They assert that such poor compiler error messages hinder the learning process of a novice programmer.

3.2 Pedagogical Agents in Learning Environments

Atkinson [7] studied the efficacy of a computer-based learning environment through the use of a pedagogical agent. They aimed to determine the efficiency of a learning environment that included an animated agent that can aurally deliver instructions and also if the image and characteristics of the agent itself can promote learning. The learning system used in the study was called worked example that used a step by step approach to solve particular problems. The study was conducted on thirty undergraduate students using three versions of the system--no agent, agent with text only and agent with text and instructions commu-

nicated aurally. The results indicated that students who used the fully embodied version of the agent performed slightly better than the students who used the other two systems. Also, students experienced less difficulty when the instructions were communicated aurally when compared to text based instructions.

Baylor [10] and her colleagues performed a study using their MIMIC (Multiple Intelligent Mentors Instructing Collaboratively), which is a pedagogical agent based learning system. The study incorporated three types of pedagogical agents--1) Expert agent provided a lot of information with subdued expressions, 2) Motivator agent provided encouragement and support along with a variety of expressions and 3) Mentor agent expressive agent that performed the roles of both the agents mentioned above. The study included 73 undergraduate students on educational technology course. They found that agents with sufficient knowledge (Expert and Mentor) facilitated better information transfer than the agent that provided only motivation and encouragement. Among the two agents that were effective, they found that the mentor agent was better than the expert agent which signifies that motivation aspect along with sufficient knowledge base for an agent is important in agent-based learning tasks.

Erin et. al [44] created an animated pedagogical agent called Adele which supported students with their problem solving exercises. Adele was used in a web-based system designed to teach medicine. It supported both single-user, single-system tutoring and multi-user, multi-system collaborative exercises. Given a case, students act as physicians and ask for the medical history and order medical tests etc. Adele monitored the students actions and provided feedback accordingly. Adele was implemented on a task-by-task basis based on the learning goals. The knowledge base behind Adele has a set of task steps, the dependencies between them, and their rationales. This architecture is very generic and can be easily extended to other domains as well. In this study, they found that students were receptive to the hints and the rationale provided by Adele. They also concluded that the agents behavior and appearance enhance the perception of expertise in the agent.

Johnson et. al [34] conducted a study with middle school students for a course on electric

circuits to obtain the voltage value. The students worked with a Cartesian graph where they could obtain the voltage value by changing the resistance and current values. The researchers used an animated pedagogical agent to point to clues in the graph that will help the students to calculate the voltage values. The results showed that the lower prior knowledge benefited from the pedagogical agent while people with high prior knowledge did not benefit a lot from the visual cues of the agent.

Carlotto and Jaques [15] integrated a pedagogical agent into a computer-assisted learning language (CALL) system used to teach English as a foreign language. The study divided the students into groups of four to use different versions of the agent: no agent, voice-only agent, a static agent and a fully embodied agent. The pedagogical agent used in the system was called Patti who resembled a wise and polite teacher. The study was mainly focused on verifying if the presence of an agent on the screen, agent simulated student-instructor relationship and the instructions delivered aurally had an impact on learning. They had a pretest and a posttest to evaluate the results of the study and found that the students that used the system with a pedagogical agent had a significantly higher gain in scores. To evaluate the effectiveness of content delivered aurally, they compared the scores of students with a voice-only agent with a no agent and found that students using the system with voice-only agent performed very well. They also studied the effect of the impact of expressions of the agent and found that a students working with fully embodied agent had a slightly less score gain than students working with a static agent.

Lester et. al [33] conducted a research to evaluate the contribution of pedagogical agents to student learning. They focused the study on the efficacy of pedagogical agents to improve the problem solving ability of students, the types of advices that the pedagogical agent can give to improve problem solving and the modality of the agent that can aid in problem solving process. They created five variants of Herman the bug and introduced it into their learning environment--DESIGN-A-PLANT. The five variants of the agent are: 1) a muted agent that provides no advice, 2) an agent providing task based verbal advice, 3) an agent providing principle based verbal advice, 4) an agent providing principle based verbal advice

coupled with animation and 5) a fully expressive agent. All the agents except the muted agent provided advice regarding the plant components that are operationalized by students in their learning task. The study compared the scores of students before and after the use of pedagogical agents and found that the use of any type of pedagogical agent improved the performance of the students. This means that careful advice provided by the agents are well received by the students and it helps them in their problem solving abilities.

3.3 Motivating Pedagogical Agents

There has been a lot of instances where a pedagogical agent was used to motivate a student in their learning environment. We will discuss about a few of them in this section.

van der Meij [47] conducted a study to check if pedagogical agents can improve the learning and motivation of students. The subjects of his study were 49 students from upper elementary school who were learning a tutorial of Microsoft Word's formatting options. They introduced an annotated photograph of a male who was about the same age as the students. During the tutorial, the agent was in constant communication with the students providing them information about the importance of the activities in the tutorial and motivating students to overcome any obstacles they may face. They performed a comparison study of students using a tutorial with and without the agent. The students who used the tutorial with the motivating agent had a significantly higher mean score than the students without the agent signifying that the presence of an agent can aid in the learning process and motivate the students.

In a later study, van der Meij et. al [48] evaluated if student motivation can be enhanced by a motivating agent in a science inquiry environment. Students participating in the study used three types of the same system—containing an agent with voice and image, system with voice added and a system with no voice or image. Their findings indicated that working with an agent with voice and image providing motivational feedback reported a rise in their

appraisal of self-efficacy. The scores of these students improved too, but only marginally better compared to the students using the other systems.

A lot of studies involving the pedagogical were not able to sufficiently determine if the learning gain was due to the extra help that the agent provided or if it is because of students having a learning companion. So Arroyo et. al [6] conducted a study where the pedagogical agent only motivated the students to continue working on the problems and also empathizing with the student's emotions. Such an agent was used in a system to solve math problems. It was reported that students using this learning companion performed well in all measures of their study and more importantly, students expressed higher confidence levels, excitement and interest suggesting that learning companions can be used to motivate students.

Student motivation is one of the key factors in self-regulated learning environments. Duffy et. al [17] conducted a study using a pedagogical agent in MetaTutor, a multi-agent hypermedia-based intelligent tutoring system. In their study, they wanted to examine achievement motivation through interaction with pedagogical agent scaffolding. They use five different types of agents in MetaTutor—each with a specific task, and they are all active throughout the session observing student's actions. In a system where the pedagogical agent intervened with prompts and feedback, students having a performance-approach performed better compared to students having a mastery-approach. Their conclusion is that prompts and feedback may impact students different depending on the motivational orientation of the students.

3.4 Conclusion

In this chapter, we established that novice programmers face a lot of difficulties understanding and dealing with errors. We also looked at the use of animated pedagogical agents in learning environments and their effectiveness. Finally we discussed about the effective use of pedagogical agents to motivate the students in learning systems. This literature survey provides necessary support to our idea of helping students through out the programming

exercises and this support can come from a virtual teaching assistant that is modeled after an animated pedagogical agent.

Chapter 4

Design Goals for Maria

As stated in the problem statement in Chapter 1, students have a general negative experience with programming. Students using Web-CAT receive an intelligent feedback about the failures of compilation, static analysis checks, evaluation of unit tests and instructor reference tests. Web-CAT feedback only enhances the negative experience but Web-CAT has no provisions to alleviate these negative emotions surrounding programming or influence what students think when using Web-CAT. The goal of this project is to continuously support students emotionally under the purview of an automated system. We believe we can achieve this overall goal by achieving smaller sub-goals. We will discuss about these sub-goals, the design decisions and how these sub-goals contribute to the overall goal in the further sections.

4.1 A Relatable Teaching Assistant

For our work to be successful, it is imperative that students perceive our Virtual Teaching Assistant well as someone they relate well to. This is important because for students to receive the support provided by the pedagogical agent, students will have to relate well to the agent. A careful design of the animated agent can make students perceive our agent as

Figure 4.1: Maria: relatable teaching assistant

a relatable teaching assistant.

We discussed the design constituents of pedagogical agents in Chapter 2. The design choices play an important role in perception of the agent by the students. When it comes to the appearance of the pedagogical agent, there are a lot of factors that needs to be considered. We decided to model the pedagogical agent as a female about the same age of undergraduate students. Patitsas et. al [39] showed that students find undergraduate TAs to be more effective that graduate TAs. Although there is no significant evidence supporting the choice of gender having an influence on students' learning process, there is support from Baylor [11] to suggest that impact on female students can be enhanced by using female agent who is attractive and young. So as seen in figure 4.1, our agent will look like a undergraduate TA. As seen in Chapter 2, agents of same ethnicity as the students are seen as more reliable and trustworthy by the students. Since our college consists of students belonging to various ethnicities, we chose an ethnically ambiguous agent to be our virtual teaching assistant. We named our virtual teaching assistant as Maria, which is also found to be a culturally

ambiguous name and used by people of all ethnicities. Competency is another essential factor in designing a pedagogical agent. We have modeled Maria to be as competent as a human teaching assistant is to provide explanations about Java errors. Students can quickly judge the competency of the agent by the appearance of the agent as well. So we selected an agent who is formally dressed but at the same time not very formal that students think the agent is an instructor or an expert. Also, Maria has the ability to orally communicate instructions as discussions in Chapter 2 identified voice to be one of the key factors to be considered while designing a pedagogical agent. Table 4.1 outlines few of the agents we considered and a brief argument about the selection criteria with respect to each of the agents.

When creating a virtual teaching assistant, there will be obvious comparisons to a human teaching assistant. Currently, Maria is not as good as a human teaching assistant but the goal is to become as good as a human teaching assistant who provides instructional support and emotional support to students. As mentioned, human TAs are not always available to students and we see Maria to be filling in for a human TA when they are unavailable. Few things that Maria can do better than a teaching assistant is to provide individual attention to students and automatically pick up on cues based on the student code submission to identify student effort, frustrations etc. For Maria to become as good as a human teaching assistant, she can be improved on various fronts as described in the future work of this thesis. Biggest failing currently in the direction of having Maria as good as a human teaching assistant is the inability to hold conversations with students. As a virtual teaching assistant, we think it is an important trait to posses and that will be the next work done.

4.2 Explain All Errors

As mentioned before, the errors in Web-CAT feedback contribute significantly to the negative experience surrounding programming because a many a times students do not understand these error messages increasing frustration. Compiler error messages are very cryptic and

Agent	Justification
	• Dressed too formal. Students may perceive to be like an instructor and never connect with it
	• Male agent. Some studies have shown female agent to be better in pedagogy than male agents
	• Appearance of the agent looks to be older than the students. Students connect better if the agents are the same age as the students
	• Agent is dressed too casual. We want to give the impression of an teaching assistant who is expert in Java errors.

Table 4.1: Comparison of agents

novice programmers often find it very difficult to understand them [37] and Web-CAT provides no support to students to understand and fix these errors In this section, we will describe how we aim to reduce the student's frustration by providing explanations for the various error messages seen by students in Web-CAT.

We have outlined the feedback that Web-CAT provides to students in the previous sections. As seen in Figure 4.2, the feedback is organized into 4 top level categories and each of these categories have subcategories [31]. This constitutes the summarized feedback of Web-CAT. At the lowest level of abstraction is the feedback of individual errors that appears in expanded boxes under each of the 4 boxes. The expanded boxes provide us a small subset of potential errors that student may ask Maria about. In such cases, students are most likely to ask Maria about the errors that they have encountered in their program. So we can provide small help links against each of these errors in the feedback which upon clicking provides explanation about the specific error that the student want to know about. This feature requires less effort than interacting with Maria or interacting with a human teaching assistant. This will enable the students to adapt our Maria support system even more.

When the design goal is to explain all errors, it is not enough to just explain the errors that the students see in the feedback. We need to provide a way for students to know about any error when they are working with Web-CAT. So we incorporated Maria also as a **chatbot style** teaching assistant in Web-CAT so that it is easily accessible to them the entire time in their working environment. Artificially Intelligent chatbots are pervasive these days and people use them all the time and are very familiar with them.

Chatbots have shifted the paradigm of research in Human Computer Interaction from work in designing graphical user interfaces to natural language user interfaces where users can communicate with the system through a string of texts [25]. Technology giants like Facebook and Google have already predicted the future of user interaction with digital applications will move from graphical user interfaces to messaging applications like Messenger and Allo. Even in pedagogy, many instructors use messenger apps like Slack in their classrooms that have a lot of chatbots designed specifically for students and are used widely. So there is a lot of support for the use of chatbot style communication to be effective and this chatbot is accessible all the time to the students.

To support the goal of explaining errors and for Maria to be perceived as a credible teaching

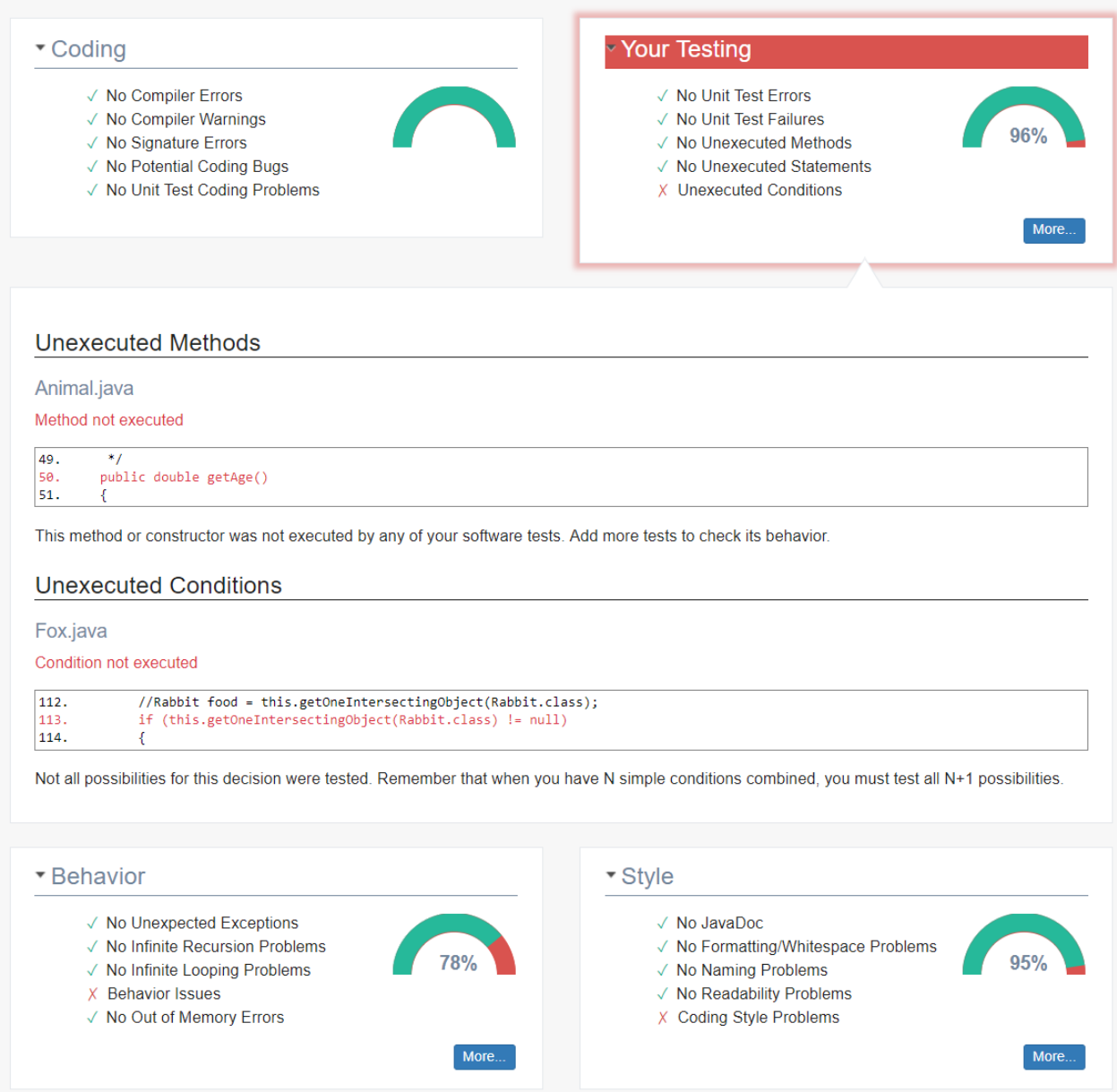

Figure 4.2: Web-CAT feedback page

assistant who has expertise in Java errors, we need to have a good **knowledge base** of errors and explanations that Maria can refer to. As mentioned before, Web-CAT compiles students' Java programs and if there are any compilation error, Web-CAT displays the compilation error observed to the students. It does not perform any additional processing of the program. If there are no compilation errors, Web-CAT evaluates the students unit tests for errors, failures and code coverage. Web-CAT also uses static analysis tools like CheckStyle and PMD to perform a static analysis of students' programs for adherence to coding standards. In addition to these tests, Web-CAT also runs instructor provided reference tests against the student program to check for correctness. The feedback to students can contain information about errors and failures from any of the checks performed above classified into various categories and sub-categories depending on the severity of the error. Few of these errors, like the code behavior errors, signature errors are dynamic and assignment specific and will require a human to look at the code and provide an explanation to the student. But for all other types of errors, there is usually a fixed explanation for the cause of the error and we can provide examples for fixing these errors.

In our work, we have identified the errors that can have standard explanations and collected information about these explanations and used it as the knowledge repository for Maria. This knowledge repository is quite expansive, so when a student interacts with Maria to learn about any of these errors, it will be able to provide an answer. This way, students will perceive Maria as an expert in answering questions about compiler errors, run time errors and static analysis errors, which will add to the reliability of Maria among students.

4.3 Promote a Growth Mindset

The difficulties surrounding programing courses are major causes of attrition rates in Computer Science department in many universities. The main reason for the attrition is believed to be the mindset of students [18] where they think programming is an innate skill and it

cannot be acquired. This is called a fixed mindset. The other alternative to this is the concept of a growth mindset where students understand the effort it takes to be successful. Individuals with a fixed mindset focus on scoring points in the assignment and not the learning process. They easily get discouraged by the feedback from automated grading systems, which can be negative and discouraging. Edwards and Li [19] proposed approaches to make the automated feedback more positive by identifying and recognizing hard work and effort from the students rather than just focus on the performance goals. They developed a set of twelve indicators that can identify student's effort. Using the combination of indicators to measure the student effort reliably, a positive feedback can be generated to foster a growth mindset among students.

Now a human teaching assistant when working with a student would be able to visually see and recognize the effort the students put in an assignment. We simulate this behavior for Maria as well, where the feedback mentioned above will be delivered as though it is coming a person who students perceive as a teaching assistant. This feedback coming from Maria will be of far greater importance than a block of text which is just statically placed in the feedback page.

Even when feedback is provided through Maria, this feedback appearing too often can quickly become noise and students tend to ignore it or the feedback appearing rarely can be ineffective to students. To solve this problem, we use the **Variable Interval Reinforcement** technique. It means that reinforcement is provided during varying intervals. For example, if the design is to provide a reinforcing feedback on an average of every 60 minutes, the first feedback can be provided in 30 minutes and next one after 90 minutes. This has shown to be a useful technique when providing reinforcing comments.

4.4 Providing Sympathetic Support

We have already discussed about the difficulty of programming for students and how the negative, objective oriented feedback of automated grading systems can be detrimental to their learning efforts. In this section, we discuss scenarios where students get frustrated with the feedback and think about giving up. This partly ties back to argument about students' mindset.

There are times when a student's code might not compile in Web-CAT. Sometimes student's code might not match the instructor's specifications—for example, the class name might not match what the instructor had specified. In such scenarios, students typically get a zero score on their assignment. There are other scenarios where a student might keep getting a recurring error, or fixing of errors do not improve their scores drastically. In all these scenarios, students with a fixed mindset get frustrated easily and they give up. Also, when students are frustrated with the feedback from Web-CAT, there is currently no way to influence what the student is thinking at that point in time. Providing sympathetic support can not only encourage the students to continue working on the assignment but can also positively influence the student's mindset. An experienced human teaching assistant would be easily able to identify that students are frustrated and they might empathize with them and help them resolve errors to get a good score. But in the absence of a human TA, when students encounter such scenarios while working with Web-CAT, Maria identifying such scenarios where they frustrated and providing encouraging comments will not only help the students to continue working on the assignment, but also perceive Maria to be more context-aware and hence human-like. As with providing growth mindset, we use the variable interval reinforcement technique to provide sympathetic support as well.

These encouraging comments and motivating feedback have been designed following Keller's ARCS model [28]. This model follows an intuitive and goal-oriented approach to identify the audiences that the system will work with and develop motivational strategies accordingly.

Chapter 5

Implementing Maria

In this chapter, we will discuss the implementation details of Maria. We will briefly discuss the pedagogical agent, the technology supporting the chatbot and its implementation. We will also discuss the implementation of the feedback used to promote growth mindset and the motivating feedback.

5.1 Pedagogical Agent

Pedagogical agents are by definition simulated human-like interface. So our implementation of a pedagogical agent is an animated avatar. These avatars are available widespread on the web. We selected our avatar from Bot Libre [1] which is a popular open source platform for developing and hosting bots. Bot Libre is a very popular site and supports bots for web, mobile, Slack, email, Facebook, Telegram, SMS, Skype, Twitter, WeChat etc.

For our work, we used the avatar shown in Figure 4.1. Bot Libre is very powerful and we can embed their avatars standalone on our own websites or use Bot Libre web apis to deploy the avatars. We have chosen to embed the Maria avatar standalone in Web-CAT.

Embedding a avatar from Bot Libre is easy. Bot Libre has a lot of flexibility while selecting

the characteristics of the avatar. For example, we can select the language that we want the bot to speak in, the type of voice, the type of emotion that the avatar should display by default etc. Once we select the required options, it generates a code which we can use to embed the avatar in Web-CAT. The avatar is a Javascript based implementation and the we can further modify the api provided by Bot Libre to customize our avatar. We encountered an issue with the rendering of the bot which we were able to fix by looking at the api and making the necessary changes to get the avatar working on Web-CAT.

5.2 Chat Bot Implementation

This is the most important work of this thesis. As described in Chapter 4, we decided to implement Maria in chat bot style too. The chat box we developed can be seen in Figure 5.1. There is a chat box with the animated avatar of Maria to the left and the chat history on the right. The chat box begins with an introduction from Maria. The chat box is implemented using front end technologies like HTML, CSS and JavaScript. The students can type their questions in the text box provided and press the Enter key. The user query is then displayed in the chat history followed by the response from Maria as seen in Figure 5.2. Typically for the questions to which Maria was able to provide a response, the reply from Maria would be a link upon the click on which will display the answer from Maria as seen in Figure 5.3.

5.2.1 Architecture

At this point, we will talk about the architecture we used to implement Maria as depicted in Figure 5.4. The front-end application is Web-CAT, so the chat bot is housed inside of Web-CAT. There is a middle layer that acts as a middleware used to support communication between the front-end and the back-end technologies. The back-end is an CentOS server running ChatScript, the technology that powers Maria. The middle layer is an apache server running a PHP script that will listen to AJAX requests made by Web-CAT and forward

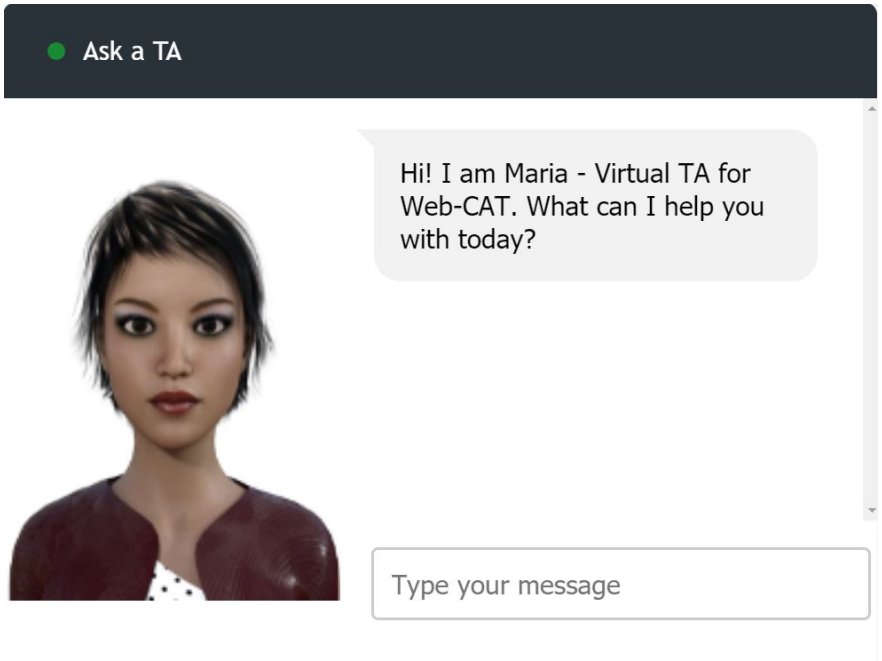

Figure 5.1: Chat box containing Maria as the virtual teaching assistant

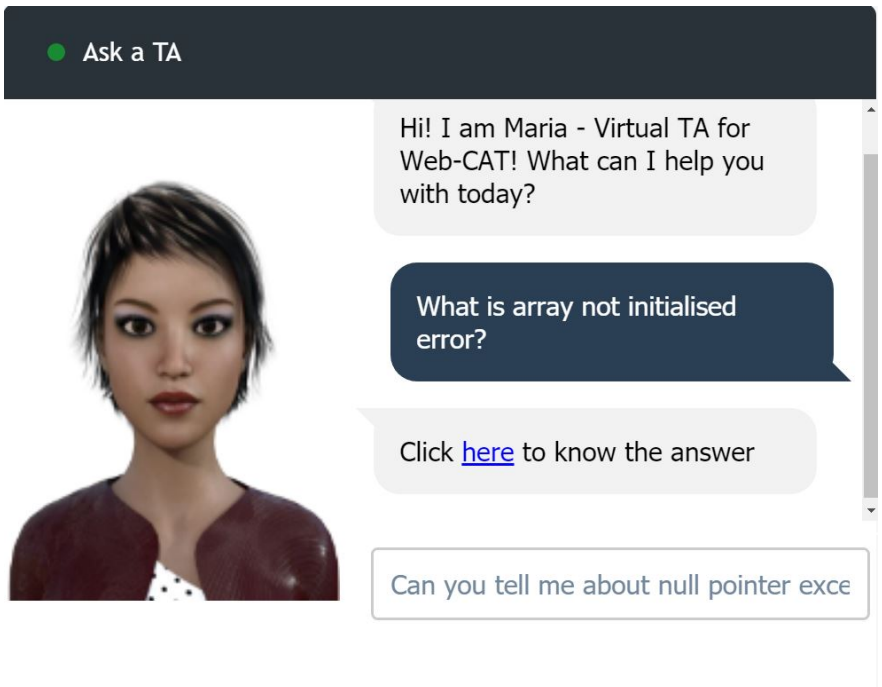

Figure 5.2: Student asking a question to Maria using the chatbox

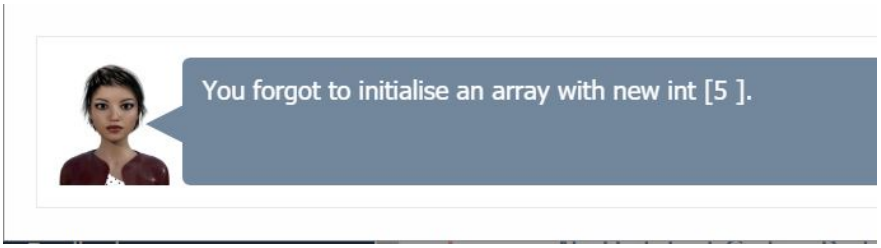

Figure 5.3: Maria's response to student's question displayed in a popup box

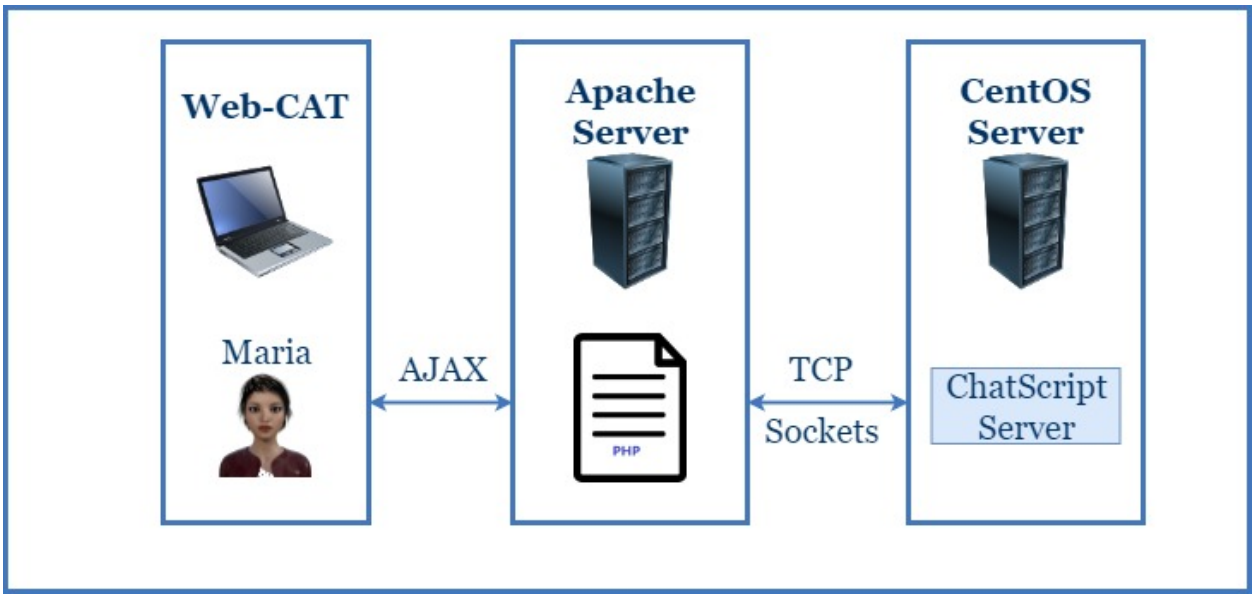

Figure 5.4: Chatbot's architecture

request to the back-end via raw sockets. Now, this middle layer was used to work around an issue with ChatScript where it can only listen to raw TCP socket communication. Since we use plain JavaScript for client side programming in Web-CAT, it does not support raw socket programming. But PHP is a powerful scripting language that supports raw socket programming. Hence, our implementation will make AJAX requests from Web-CAT to the apache server which will be received by the PHP script and then forward the request through raw socket programming to ChatScript. In the reverse direction, ChatScript will send its response to the PHP script via raw sockets and the script returns the response to Web-CAT via AJAX response.

5.2.2 ChatScript

There are many chat bot technologies that are very popular. Among them, we considered using AIML and ChatScript for our work. AIML stands for Artificial Markup Language and works based pattern matching or pattern recognition. It is an XML based application used to created artificially intelligent applications. It is easy to program, understand and extremely maintainable. But this also is the reason why we did not choose AIML as our chat bot scripting language. It is rule based, where it matches a sequence of words from the user input with the scripted rules and produces a response already mentioned in the matching rule. Few of the shortcomings of AIML are—it is very verbose and have a lot of redundant rules, the pattern matching is done using exact words from the user input, it is also not easy to handle to generic set-based sentences. In short, AIML has a lot of shortcomings and it is simply not a powerful enough chatbot engine. ChatScript on the other hand is a very powerful Chat bot tool.

ChatScript is a very powerful natural language processing and a dialog management tool implemented in C++[3]. It has won the prestigious Loebner's award 4 times and is considered to be the next-gen Chatbot engine. Like AIML, ChatScript is also a rule based engine but unlike AIML that uses XML to code up the rules, the rules can be written in natural language in ChatScript. A rule in ChatScript consists of 3 parts—kind, pattern and output. A kind indicates what type of input pattern (statement or question) will be matched with the rule, the pattern is the set of words used to match with the user input and output is the response. What makes ChatScript so powerful is its ability to understand the context of the user input and match it with the pattern and not just an exact word match. It has a strong natural language processing engine. A topic is a collection of rules. Topics may have a set of keywords that the ChatScript engine uses to look for the pattern matching for the input sentence. There is also a control script that is a topic by itself and this is first point of control entry for ChatScript. This control script allows the user to control the functioning of ChatScript. The control script receives the volleys (user inputs) and we can specify if we

need to look in specific topic files for a match, or we can let ChatScript to figure out the topic file to look into. We can also specify the default behavior after ChatScript is unable to find a match—options include start a random conversation, display a static text etc.

We will briefly discuss about how ChatScript functions here. There is a tokenizer that breaks the user input into multiple sentences and it performs pre-processing like spell correction, proper names and idioms substitution etc. It removes any trailing punctuations and from the user input it also tries to guess the nature of user input (sentence or question). After the pre-processing, it canonizes the sentence where it will generate a list of all alternate forms for words in the input. The next step is pattern matching. The matching happens one word at a time. It matches both the exact word from the user input and the canonical forms of it. If a match is found, it will continue matching with the rest of the patter. If not, the rule has failed and it will move on to the next rule. We have the ability to specify the searching techniques of the rules—if it has to random, linear etc. ChatScript looks to match the pattern in the current topic file it is working on. By default, if a rule is matched once, it will be marked as used and it will not be considered for further matching to avoid repeating the same answers.

In our implementation, we retained most of the out of the box properties of ChatScript. Although ChatScript is good in deciphering the keywords from the user input and use that to lookup the different topics, when it comes to Java errors, it was unable to match the topic to the user input accordingly. So we made changes to the control script to explicitly look in all the topic files until it found a match. Another important change is not to delete a rule once it is "used up". Students can have questions about the same error multiple times and this vanilla functionality will deter the performance of Maria.

5.3 Explaining All Errors

We explained in Chapter 4 about the accessibility of Maria to students. One way to easy accessibility is to provide help links besides the error messages displayed in the expanded section of the feedback as depicted in Figure 5.5. Clicking on the link would display a popup containing the explanation about the error. One way of implementing this would be to parse the DOM elements to find out the error adjacent to the explain links and use that to make a ChatScript request to elicit the response. But this method is very poor in terms of maintenance because every time the HTML is modified, the code needs to be modified as well.

We handled this implementation through the Perl plugin that generates the student feedback. We created keys to be associated with each of the errors. When the Perl plugin parses the errors to produce the feedback, it also populates the Key associated with the error in a hidden field in DOM. We have also created rulesets in ChatScript to have rules that have the patterns as keys. So now when clicks on the Explain link, we make a request to ChatScript with the key and get a response back. Now, sometimes the error messages can be specific to the program like inclusion of variable names in the error message etc. We have identified such error messages and created generic keys for them and part of processing in the Perl script involves making such messages generic so that we can simply perform a lookup to find the corresponding key value.

We described in Chapter 4 about the knowledge base required for Maria to be an expert of Java errors. For Maria to be a credible teaching assistant, it is important that Maria is able to answer all the questions related to Java programming errors. There are three types of errors that we would want Maria to respond to—Java compilation errors, run time errors and failures from static analysis tools like PMD and CheckStyle.

There are a lot of resources available on the web to provide explanations for errors and possible examples to fix compilation errors and run time errors. We used [2] as the source of

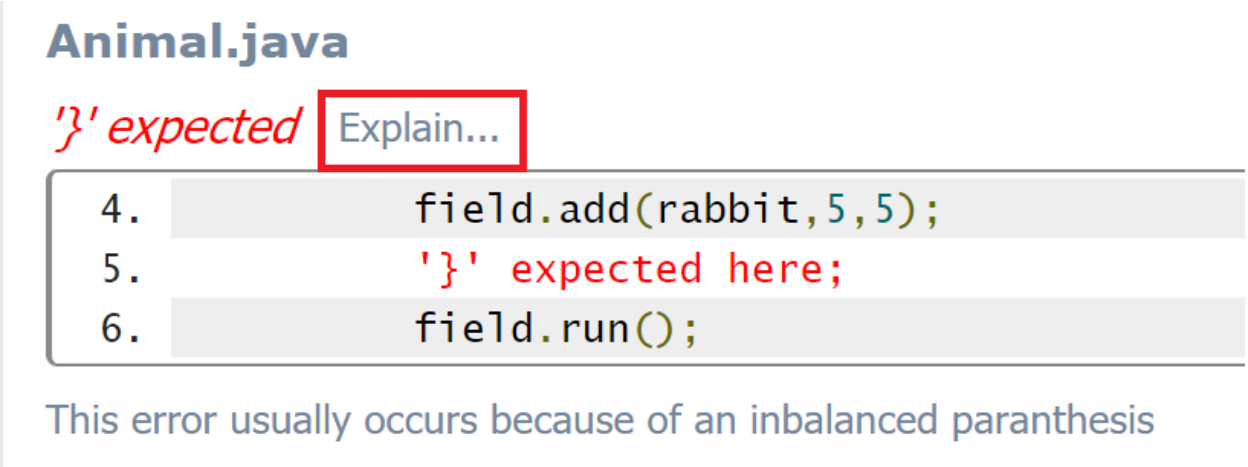

Figure 5.5: Feedback page containing the explain link adjacent to the error messages in the expanded section

explanations for errors that Maria provides. But PMD and CheckStyle error explanations are not very pervasive so we looked into the documentation of these tools to identify the different checks it performs and what causes failures in those checks and many of the checks also had examples of incorrect and correct code. We think this is a good explanation for students to understand why a particular static analysis check has failed. So we used the documentation as the information source for static analysis errors.

We have the information source required to create our knowledge base. It has to be translated to a form that ChatScript understands i.e. into rules. We coded a python script to generate the rule files from these websites for compile and runtime errors. The output of the script will generate rules with kind as u (responds to both statements and questions), the pattern as the actual compile or runtime error message and response as the explanation from the information source. Using this method, our rule set consists of explanations for 144 compiler errors and 88 run time errors. This is a very comprehensive list of all the compiler and runtime errors and we believe Maria would be able to answer any questions that the students may have regarding the errors. Likewise for PMD and CheckStyle failures, we have explanations for all the checks performed coded in the form of ruleset for ChatScript.

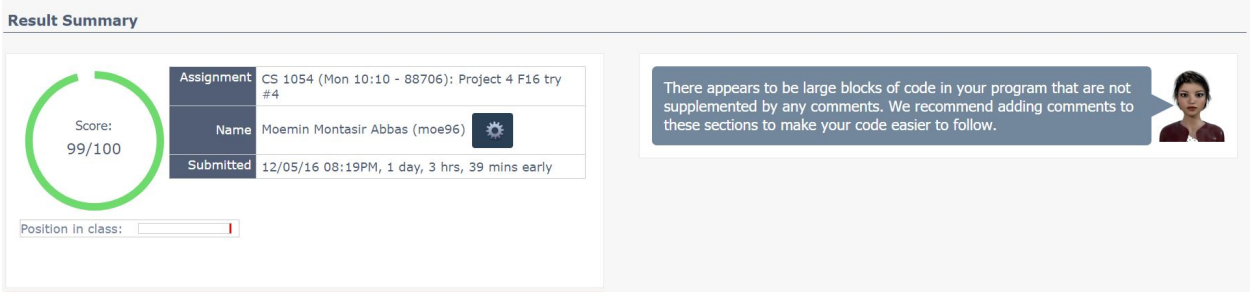

Figure 5.6: Maria providing reinforcing feedback about effort indicator

5.4 Promoting a Growth Mindset

Edwards and Li [19] have come up with strategies to identify the hard work and effort by students through a set of fifteen progress indicators. They have also come up with the set of encouraging comments and reinforcing comments for scenarios where these indicators are triggered. The scope of this project is model this feedback as though it is coming from Maria. We provide a placeholder in the Web-CAT feedback page with a picture of Maria and a small speech bubble that will contain the indicator feedback as shown in Figure 5.6. We have provided evidence through literature survey that pedagogical agents are effective in communicating such feedback to the students.

5.5 Providing Sympathetic Support

In this section, we will discuss the work we have done towards Maria providing motivating/encouraging comments to students. We explained the justification for Maria providing such comments in Chapter 4. We have identified the scenarios where we feel students feel lack of motivation and give up. Students with fixed mindset may feel indifferent if they get a zero on the assignment after working hard on it or if they unable make sufficient progress in scoring points even after fixing errors, or when an error is recurrent. But we realized that it difficult to identify scenarios when students are "stuck" because if significantly changing

scores is the criteria for marking students as stuck, then if students fix simple errors and not focus on the important errors that will actually boost their scores, they will never be identified as students being stuck. Other criteria we considered was to use the number of errors fixed between consecutive submissions as the criteria. But this also has a drawback where students can take a long break between submissions and when they get back to working on the assignment, they can submit partial code to Web-CAT just to assess where they are without fixing any errors. Such students will be incorrectly identified as being "stuck" when they are not. Even with students getting a zero score, we cannot provide the motivating comments every time students get a zero. An alternate would be to see if they get zero on few consecutive submissions but the problem is we do not have an clear idea if we should provide a motivating comment for getting a zero score or a comment for being stuck. Such difficulties prevented us from deploying the feedback comments we have generated to motivate students. The motivating comments were developed following the ARCS model as mentioned in Chapter 4. The motivating comments that we can provide to students when they get a zero is shown below:

- It is not uncommon that people get a low score on their first few submissions. If you keep working hard, you might soon see your score increase.
- This score might not be an indicator of the good effort you put in this assignment. If you keep working, you might see an improved score that matches your effort.
- The most important aspect of this assignment is the learning you get out of it. As you keep working, you will see an improved score to reflect the learning.
- I know you have it in you to complete this assignment. I am confident that as you work hard, your score will improve.

Likewise, the comments we have developed to motivate students when they are stuck are:

- Lets take it one step at a time. Focus on that fixing the behavior errors and you will find out what I have I known all along that you can do it.
- Alright! Looks like you have hit a roadblock. You can always visit a TA during the office hours to seek additional help.
- You have put in good effort to get this point in the assignment. You can get additional help from class discussion boards if youd like.
- It looks like you are not making sufficient progress in the assignment. Keep working hard, I am confident your score will improve.

One of the ways of making a chatbot more human-like is by making it context-aware. One of the ways of doing it is by not only providing motivating comments when they are not making progress but also by recognizing the fact that they were able to make progress. This will again help perceiving Maria as more human-like teaching assistant. We have again identified some comments that can be provided in such scenarios:

- Congrats, I can see that you are making progress again!
- I see that you have crossed the hurdle. Nice work!
- Kudos! It looks like you have crossed the hurdle!
- Congrats on overcoming the obstacle!

These comments will all be displayed in the same placeholder we use to display the feedback from indicators.

Chapter 6

Use Case Walk Through

In this chapter, we will navigate through few scenarios in which the students will interact with Maria. As described in the earlier chapters, Maria will be able to provide explanations for Java programming errors and provide feedback encouraging growth mindset among students.

6.1 Students Using Explain links

In this section, we will consider the scenario where students have finished working locally on their assignment and have submitted it to Web-CAT for grading. Students can get compile time and run time errors if they have not tested their programs locally. They can also receive a number of failures from the static analysis tools like PMD and CheckStyle. Once they submit their assignment, Web-CAT analyzes their programs and provides feedback. In this scenario, students can interact with Maria in the following way:

1. After student's submission of assignment, Web-CAT compiles the code and if successful runs the instructor reference tests, evaluate student unit tests and performs static analysis checks. After this, it provides an intelligent feedback to students.

2. If there are any errors or failures in student's program, at least one of the detail

feedback pages are expanded by default. Let us assume the student is having a few compilation errors. So the detailed feedback component corresponding to the coding section is expanded as shown in Figure 6.1

3. If there are any errors in the expanded section that the student does not understand and need more help, they can simple click the "Explain" link adjacent to it. If the student does not understand what the '} expected' error is and wanted more explanation about it, they can click the "Explain..." link next to it as highlighted in Figure 6.2

4. This will open a popup box that will contain the response annotated with a picture of Maria. This response will contain explanation about the '} expected' error asked by the student. This is shown in Figure 6.3

5. Students can peruse through other errors in this expanded box and click on the "Explain..." links for other errors that they do not understand. In this example, since there are compilation errors, Web-CAT would not even evaluate all the other sections. Had the code compiled successfully and there are other failures in different sections, students can repeat the same process for errors in other expanded boxes. Although there are a few errors like the Signature errors, behavior errors etc. for which Maria would not be able to provide an explanation and for such errors, the "Explain..." link would not be present.

6.2 Students Using the Chatbox

Typically, when students start working on an programming assignment, they code use an IDE of their own choice. They will write their code according to the specifications provided by the instructor. They do not get to see the feedback from Web-CAT until they submit the code to Web-CAT. Here we will look at a scenario where student submits a code to Web-CAT and "chats" to Maria about errors.

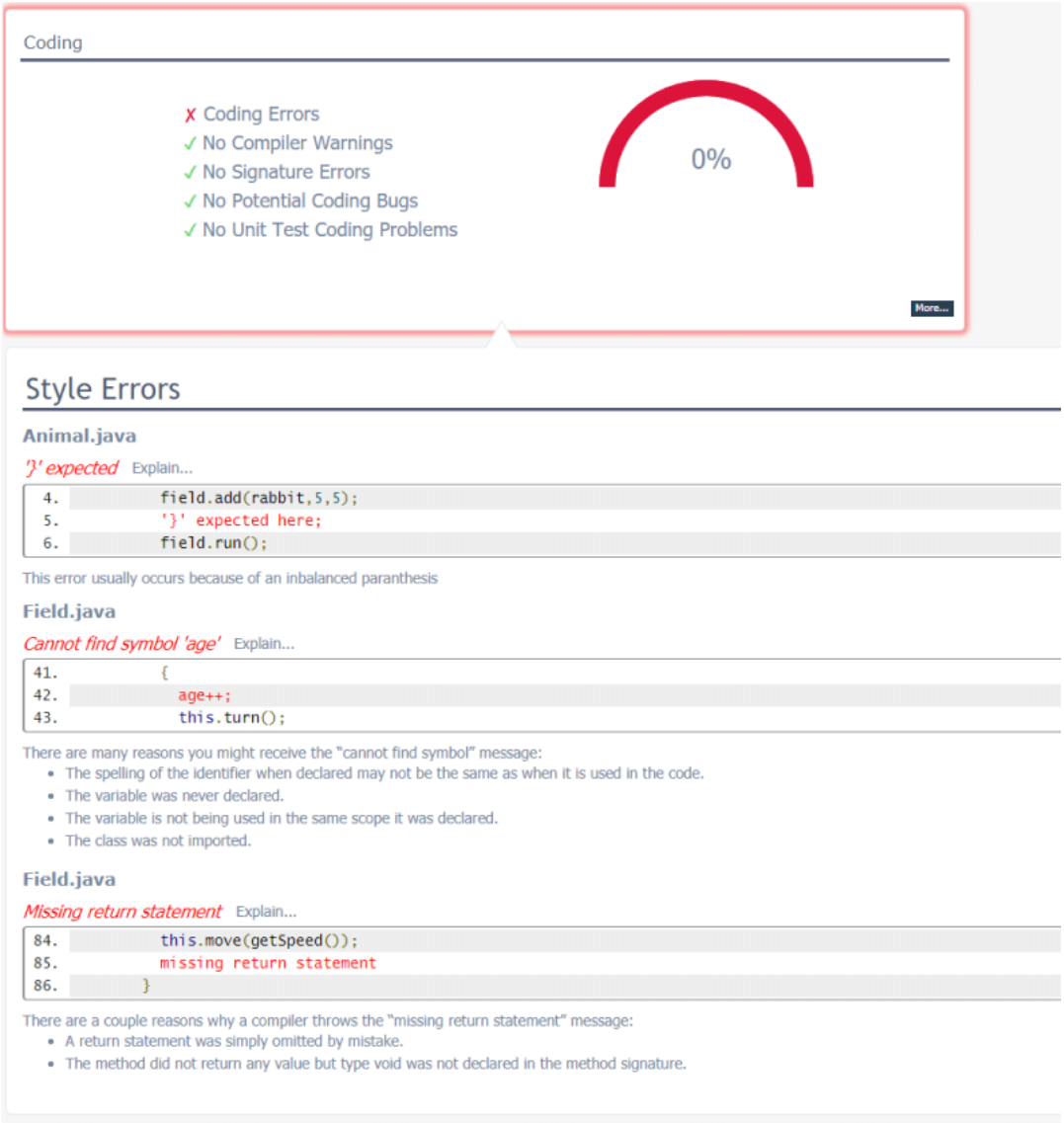

Figure 6.1: Student having a few compiler errors and the coding section expanded for detailed feedback

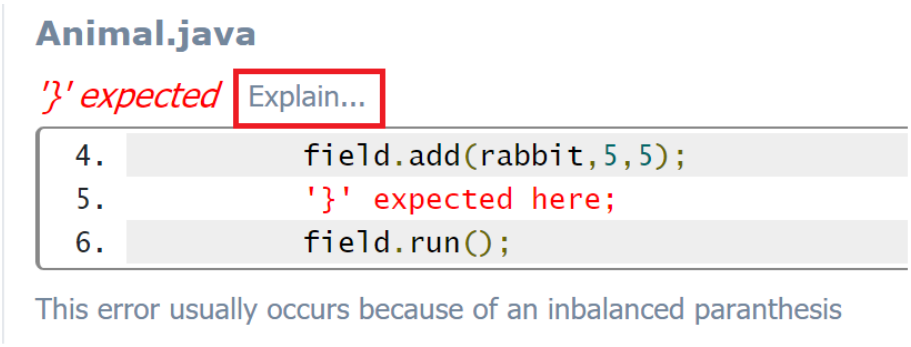

Figure 6.2: An error in the expanded section and the explain adjacent to it

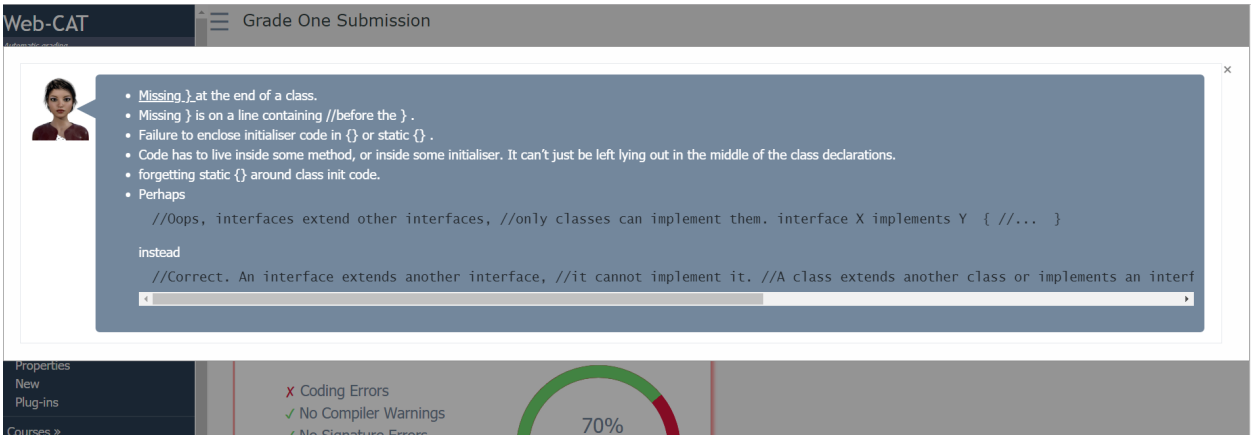

Figure 6.3: A popup box containing the response from Maria about the '} expected' error

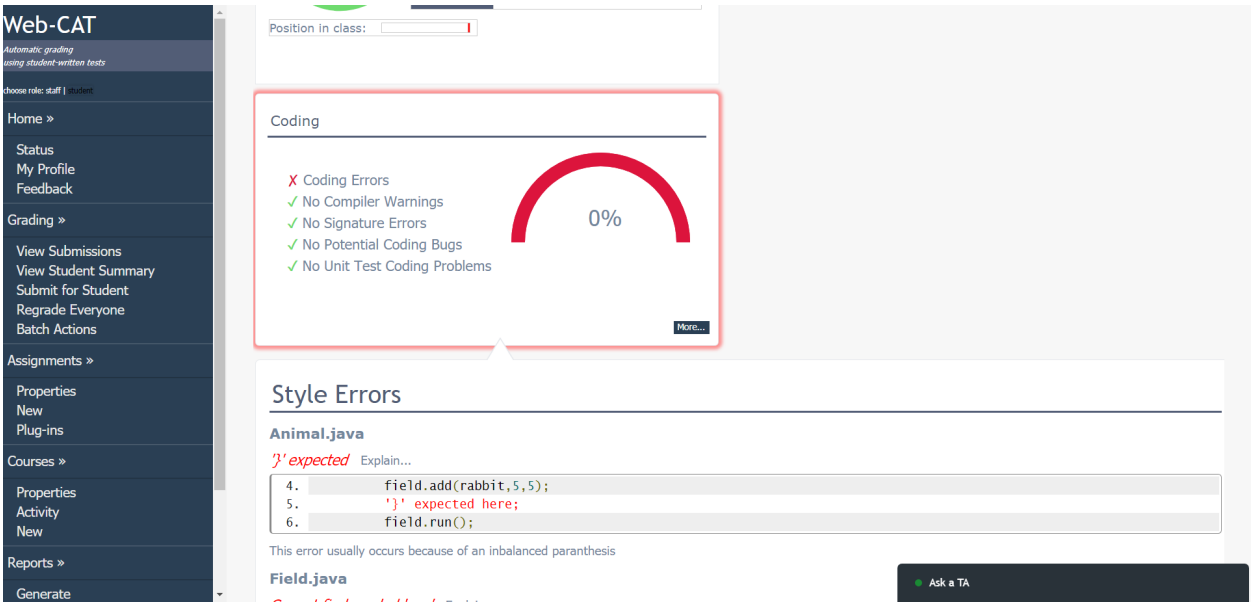

Figure 6.4: Web-CAT with the collapsed chat box for Maria

1. Student submits their code to Web-CAT through IDE or by doing it manually. Web-CAT shows a feedback about their work. Now students may want to know more about the errors displayed in the feedback page and chat to Maria about it.

2. Students can click on the chatbox header that is located at the bottom of the screen titled “Ask a TA”,as shown in Figure 6.4, to open the chatbox and begin conversation with Maria.

3. Clicking on the chat header opens the chat box as show in Figure 6.5.Suppose that student wants to ask Maria about ’missing return statement’ error. Student can use the text box available in the chatbox to type their question to Maria. This is illustrated in Figure 6.5.

4. Maria’s response will contain a highlighted link instructing the student to click on it to know the answer as depicted in Figure 6.6.

5. Upon clicking the link, Maria will display a popup box that will contain the response annotated with a picture of Maria. This response will contain explanation about the

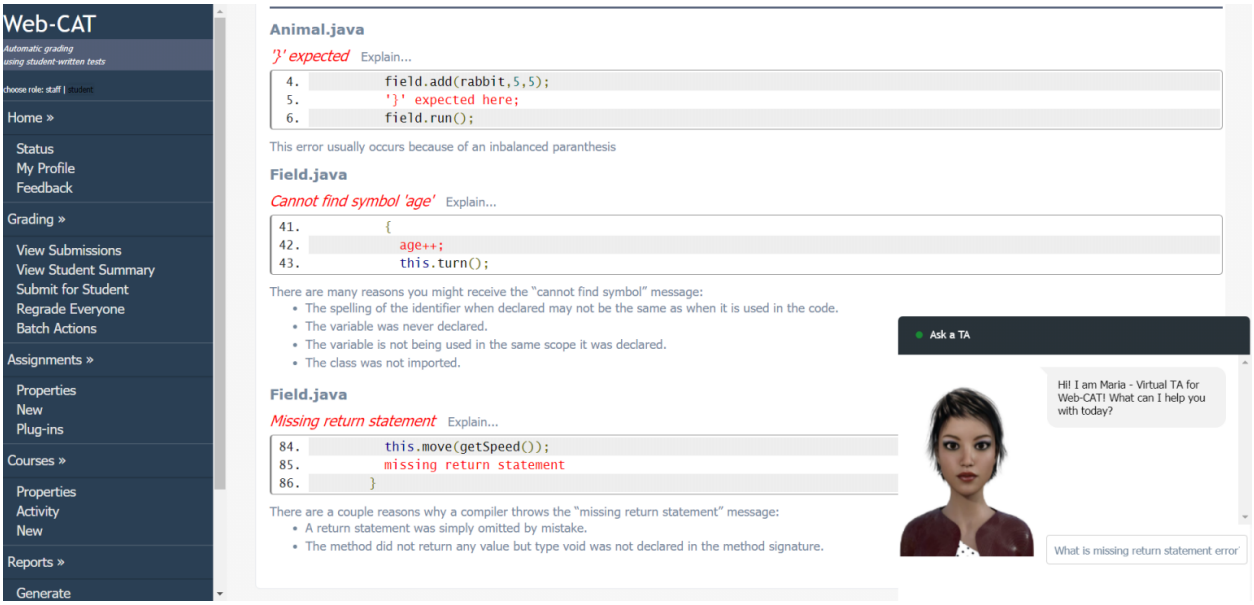

Figure 6.5: Web-CAT with Maria chat box expanded and student asking about 'missing return statement' error

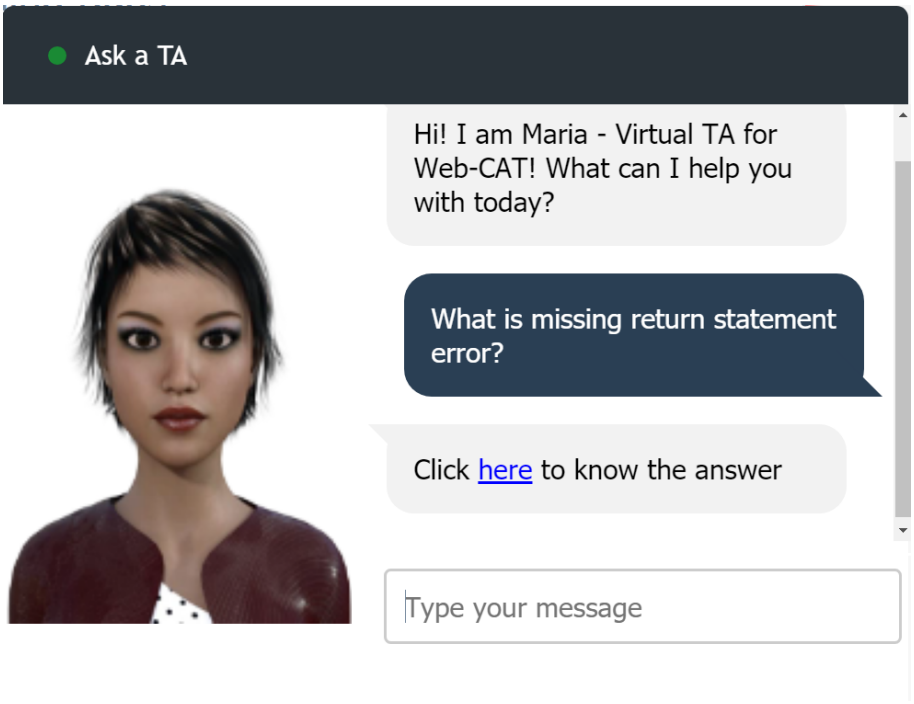

Figure 6.6: Chat history showing the student's question and Maria's response with a clickable link for the answer

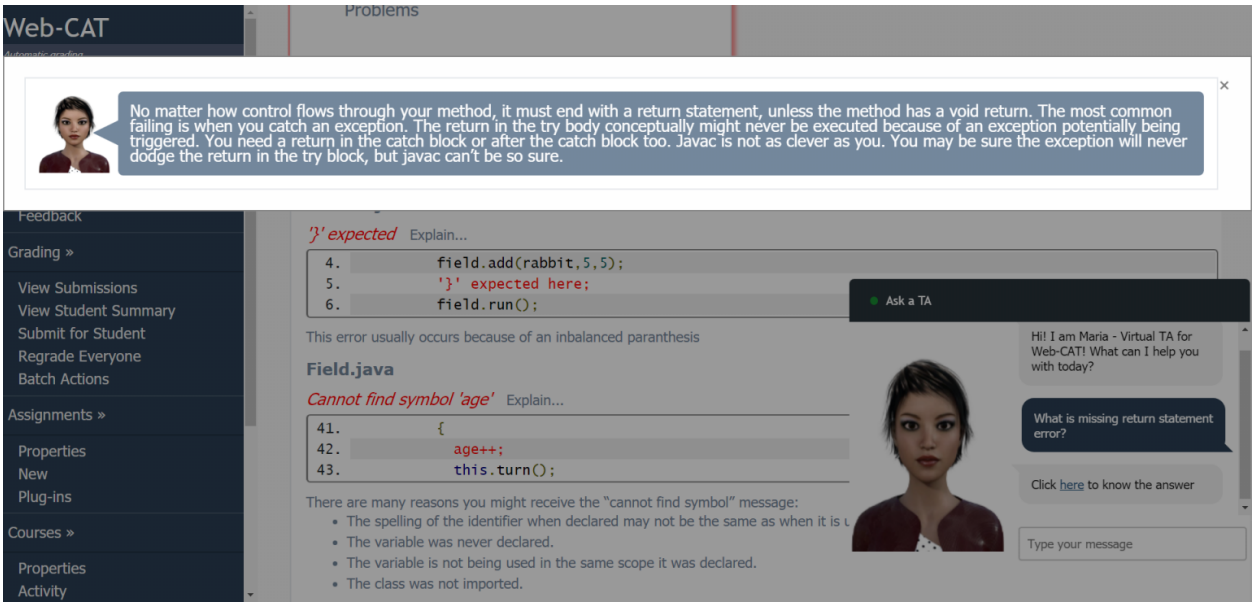

Figure 6.7: Popup showing an explanation for the question 'what is missing return statement error?'

'missing return statement' error asked by the student. This is illustrated in Figure 6.7

6.3 Providing Indicator Feedback

In this section, we will look at the interaction between Maria and the student while Maria provides feedback about one of the performance indicators. We will first look at a scenario where Maria will provide a reinforcing comment:

1. A student is working on an assignment and submits the code to Web-CAT for evaluation. Suppose that Web-CAT analyzes the code and provides feedback about unit testing coverage, styling and behavior errors in the first submission.

2. While working on fixing the feedback provided by Web-CAT, student also works on following certain best practices like increasing comments density, reducing the average method size to optimize the code, modularizing the code by adding more solution

Figure 6.8: Result summary details after 3 submissions

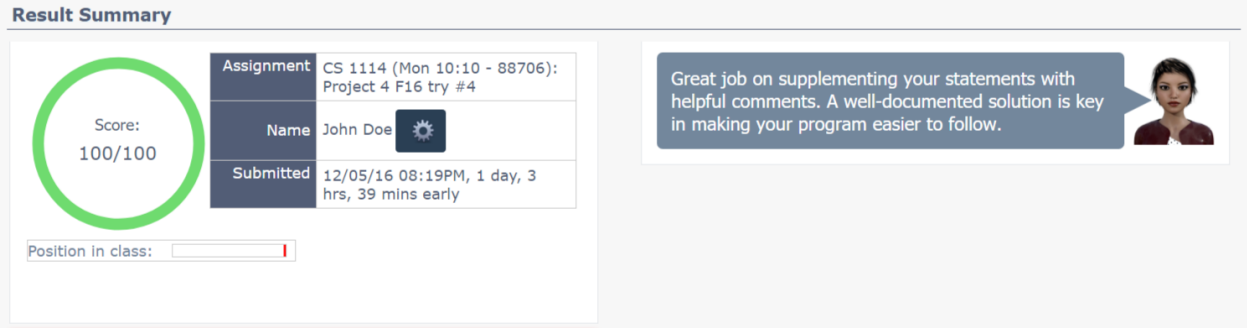

Figure 6.9: Maria providing a reinforcing comment about the effort student has made to add more comments to make the code more readable

methods. The student continues to do so for two submissions after the first one. After the student's third submission, Web-CAT's score part of the page will look like the Figure 6.8. Notice how the right hand side of the result summary section is empty.

3. Now for the fourth submission, student has fixed all the errors. In addition, lets say student has also added more comments and added more solution methods to make the code even more modular. Student fixing the unit testing coverage and static analysis errors from the initial feedbacks and the extra effort would have triggered at least 4 indicators. Now, since the student has worked above and beyond what was expected, Maria will provide a reinforcing comment about any of the indicators. In this case, Maria provided a comment about increasing comments density as shown in Figure 6.9

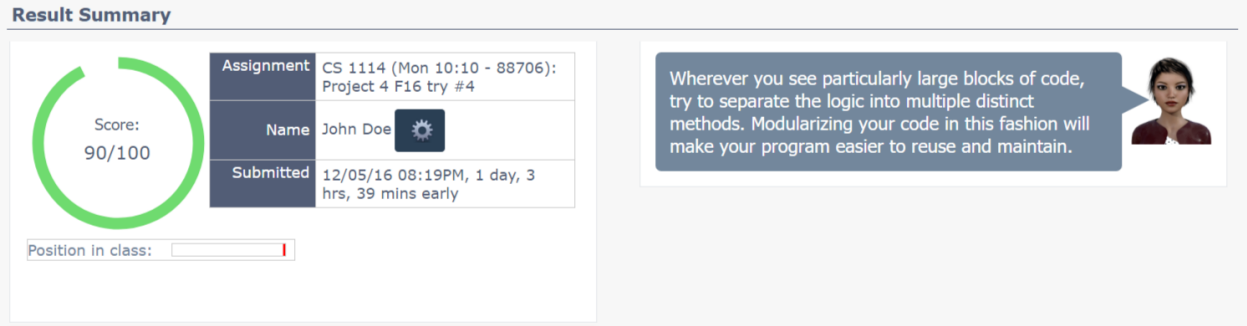

Figure 6.10: Maria providing an encouraging comment about adding more solution methods to modularize the code

Now, we will look at a scenario where Maria will provide a encouraging comment.

1. A student is working on an assignment and submits the code to Web-CAT for evaluation. Suppose that Web-CAT analyzes the code and provides feedback about unit testing coverage, styling and behavior errors in the first submission.

2. Student works on fixing all the testing coverage in the first attempt and submits the code to Web-CAT. Following that the student works on fixing all the static analysis errors in the third attempt and submits the code to Web-CAT. The student still will not get a full score in the assignment because of the behavior issues in the code.

3. In the fourth attempt, suppose that student fixes all but one behavior error and submits the code to Web-CAT. Student fixing the testing coverage issues, static analysis checks consistently over 3 submissions will have triggered a few indicators but not enough to recognize that the student is working hard since in this case student is only focused on fixing all the issues in the feedback of Web-CAT. So to encourage students to follow best practices, Maria provides them an encouraging comment about modularizing the code as shown in Figure 6.10

Figure 6.11: Result summary section of Web-CAT displaying a zero score to student due to failed compilation

6.4 Motivating Students

In this section, we will look at one of the scenarios when Maria will provide encouraging comments to motivate students who get a zero score on their submissions and improve their self-belief to keep working on the assignment.

1. Suppose that a student works on their assignment and without testing it locally submits the code to Web-CAT. Their code does not compile and Web-CAT provides the list of compiler errors and give a zero score. Result summary of the student evaluation is shown in Figure 6.11

2. Lets say the student does not fix all the compilation errors, and re-submit the code to Web-CAT. Student will get a zero score again. The result summary page will look like Figure 6.12. Students can easily get unmotivated if they see zero scores consecutive times, so Maria intervenes and provides an encouraging comment to help them continue working on the assignment.

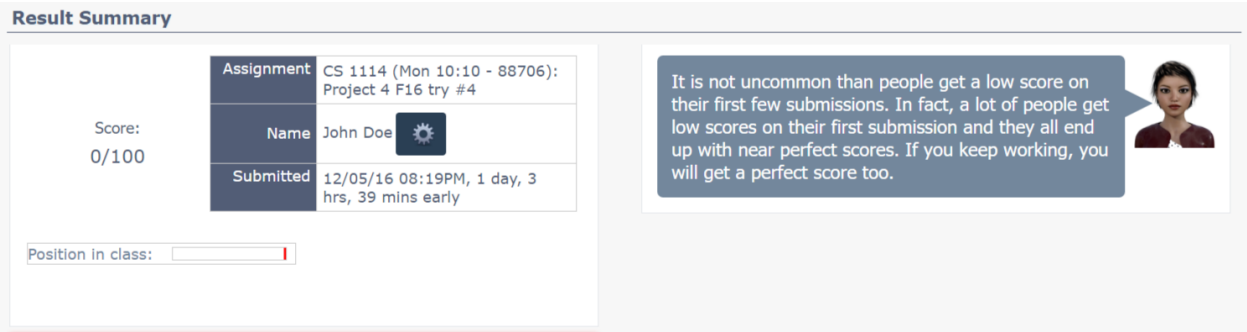

Figure 6.12: Maria providing an encouraging comment about adding more solution methods to modularize the code

Chapter 7

Evaluation

In this chapter, we will describe the evaluation methods we adopted to examine the effectiveness of Maria as a Chatbot and provide the results of the experiment. We will also describe the study conducted to review the messages provided by Maria. In another section, we outline ways to effectively measure the other aspects of virtual assistant.

7.1 Evaluation Method

Maria was designed to help provide emotional support to students while learning programming by providing explanations about Java errors, providing feedback about performance indicators and enhance motivation of students. Ideally we wanted to measure the success of Maria in achieving these goals by examining the interactions between Maria and collecting their feedback. Unfortunately, we were unable to perform the study involving students. To test one of the design goals of explaining all errors, we have devised an alternate strategy for our evaluation of the effectiveness of Maria in responding to student's questions. We have picked errors that we think students are mostly likely to ask Maria about and define certain criteria to evaluate Maria's response for those errors.

7.1.1 Data Selection

As seen in Chapter 3, there has been a lot of research performed around difficulties and misconceptions students have when programming and a number of researchers have attempted to identify the most frequently occurring errors to enable instructors to have special focus on them. One such work was performed by Brown et. al [14] as part of their Blackbox project. This project collected data from users of BlueJ (Java IDE) worldwide. BlueJ was designed and developed specifically for novice programmers and widely used for CS1 is many universities around the world. Blackbox is simply a data collection project and at the time of the study, the project had 150,000 users and over 10,00,000 compilation events. Analysis of such data enabled the researchers to find the most frequently occurring errors among novices. This data is outlined in Table 7.1 . Given the sheer volume of the data and the number of users in the study, we can confirm with a certain degree of confidence that this ordering of data is a good representation of all the novice programmers and by extension students at Virginia Tech as well.

Error	**% of all errors**
Unknown variable	17.7
Semicolon expected	9.5
Unknown Method	7.6
Bracket Expected	6.5
Unknown class	5.3
Incompatible types	4.5
Illegal start of expression	4.4
Method application error	3.7
Identifier expected	3.6
Not a statement	3.0

Table 7.1: Frequency of occurrence of Java compiler errors among students using BlueJ IDE

In addition to the compiler errors, we have also collected the most frequent run time errors.

The list is shown in Table 7.2

Run Time Errors
NullPointerException
ClassCastException
ArithmeticException
StackOverflowException
ArrayIndexOutOfBoundsException
StringIndexOutOfBoundsException
NoClassDefFoundError
OutofMemoryError
NumberFormatException
InputMismatchException

Table 7.2: Most frequently occurring run-time errors

Lastly, Edwards et. al [23] analyzed 9,913,817 PMD and CheckStyle static analysis errors occurring in 1,172,157 source files. Part of the study was to identify the most commonly occurring static analysis errors. The results are shown in Table 7.3.

7.1.2 Evaluation Criteria

Chatbots are generally evaluated by humans. There are no reliable automated mechanism to test the effectiveness of the response of a chatbot. There has been a lot of work surrounding chatbots for different needs and evaluated their effectiveness. We will look at one such study performed by Xu et. al [51] who developed a chatbot for customer service on social media because a lot of customers these days are reaching to companies via social media with inquiries. Through analysis of their subjects, they found that they received two types of requests—1) Emotional request where people want to express an opinion, emotion about a product and 2) Information Request where people want to know something about the

Static Analysis Errors Errors
JavaDoc Method
Indentation
Whitespace
Line Length
Author comment
Javadoc Variable
WhiteSpace After
Javadoc Type
Unused Imports
Singular Field

Table 7.3: Most frequently occurring static analysis errors

product. The authors performed a human evaluation of the response of the chatbot for each of these request types. The criteria they used for the evaluation are: 1) Appropriateness - is the response relevant to the topic, 2) Empathy - does the response make the customer feel valued, 3) Helpfulness - does the response address the user's request. Since empathy is very specific to the problem domain, we will ignore it and use the other two criteria in our evaluation. In addition, since Maria is a rule-based chatbot we include another criteria to check if Maria is able to match the question to a rule.

The main function of a chatbot is to comprehend the question posed by the user, lookup its rulebase and respond back to the user. If the question does not match with the rules in the rulebase, the chatbot will provide a coded default response. This will be first criteria to check if the chatbot understands a question regarding any of the top 10 errors mentioned above. If the chatbot is able to match the question to a rule, it will respond back to the user with the output. The second criteria is to validate if the output of the chatbot is relevant to the question. This will help us determine if the rule matching algorithm of the chatbot is as expected. The third criteria we will use is if the output provided by the chatbot is actually

able to answer user's question. In short, we will use the following criteria to evaluate Maria and her ability to answer student's questions:

- **Comprehension:** Is Maria able to understand the question and provide a response to it?
- **Relevance:** Is Maria's response relevant to the topic that the user has a question about?
- **Usefulness:** Is Maria's response useful in answering the user's question?

7.2 Experiments

The first experiment we conducted was to evaluate questions about the top 10 errors outline in Tables 7.1,7.2 and 7.3 based on the criteria mentioned. We devised a simple "Can you tell me about .." question for each of the compile time and run time errors for this experiment. For example, Can you tell me about incompatible types error?. We then asked Maria all these questions and recorded Maria's response and evaluated them.

Students may not always ask the same type of question. The phrasing of the questions can be different, not everyone will ask the "Can you tell me about" type of question. So in another experiment, we will try different ways of asking a question and evaluate Maria's response.

One of the main expectations of a chatbot is that it responds to user requests instantly. Hence speed is an important factor when using a chatbot and the response rate determines the speed at which the chatbot responds. In the last experiment, we will do a performance testing of Maria. We will simulate 1000 conversations at the same time and evaluate how well our chatbox implementation handles it and how long it takes to respond to those requests.

7.3 Results and Discussion

We conducted the experiment asking about the compile time errors first. The results are mentioned in Table 7.4. As we can see from the table, Maria was able to understand all but one error and providing a response. It is understandable that it is not able to understand the "; expected" error owing to the design and implementation of ChatScript as it strips off the punctuation marks typically out of place in a natural language conversation. We have to remember that ChatScript is an engine designed for natural language and we are using it for very specific purpose of answering Java errors. However, when we phrase the question as "Can you tell me about semicolon expected error?", Maria is able to provide a response as expected suggesting that an explanation will be provided for this error through the Explain links. All of the other errors for which Maria provided a response, the responses were about the specific errors in question. Also, among the errors that were relevant to the topic, we can see that most of the errors were useful to the students describing either the problem or the potential causes of the problem or ways to fix them. Two of the errors—cannot find symbol (variable) and cannot find symbol (method) are marked as partial because Maria provides the same response to both these questions that contains explanations for both these error types combined.

We continued the same experiment as compile time errors with run time errors as well. The results are documented in Table 7.5. As we can see from the results table, Maria was able to understand all of the top five questions. The response to these topics were also very relevant to the corresponding errors. Also these responses provided useful explanations about the error or ways of fixing them.

We continued the experiment for highest occurring static analysis errors as well. The results are shown in Table 7.6. As seen from the results, Maria was able to provide response to all of the questions we posed. Except for the line length failure, we received relevant and useful responses for all the other errors. Line length check is performed to see if any statement in the program exceeds 80 characters. Where as the response provided by Maria was an

Error	Comprehension	Relevance	Usefulness
Unknown variable	Yes	Yes	Partial
Semicolon expected	No	N/A	N/A
Unknown Method	Yes	Yes	Partial
Bracket Expected	Yes	Yes	Yes
Unknown class	Yes	Yes	Yes
Incompatible types	Yes	Yes	Yes
Illegal start of expression	Yes	Yes	Yes
Method application error	Yes	Yes	Yes
Identifier expected	Yes	Yes	Yes
Not a statement	Yes	Yes	Yes

Table 7.4: Results of evaluation of most frequent compiler errors

Run Time Errors	Comprehension	Relevance	Usefulness
NullPointerException	Yes	Yes	Yes
ClassCastException	Yes	Yes	Yes
ArithmeticException	Yes	Yes	Yes
StackOverflowException	Yes	Yes	Yes
ArrayIndexOutOfBoundsException	Yes	Yes	Yes
StringIndexOutOfBoundsException	Yes	Yes	Yes
NoClassDefFoundError	Yes	Yes	Yes
OutofMemoryError	Yes	Yes	Yes
NumberFormatException	Yes	Yes	Yes
InputMismatchException	Yes	Yes	Yes

Table 7.5: Results of evaluation of most frequent runtime errors

explanation for the length method of a string. The summary of this experiment is shown in Table 7.7. Maria as a chatbot was able to understand all but one of the questions and was able to provide a useful response to 90% of the questions. These results are an indication

that the chatbot will be able to understand the student's question and respond with an useful answer.

Static Analysis Errors	**Comprehension**	**Relevance**	**Usefulness**
JavaDoc Method	Yes	Yes	Yes
Indentation	Yes	Yes	Yes
Whitespace	Yes	Yes	Yes
Line Length	Yes	No	No
Author comment	Yes	Yes	Yes
Javadoc Variable	Yes	Yes	Yes
WhiteSpace After	Yes	Yes	Yes
Javadoc Type	Yes	Yes	Yes
Unused Imports	Yes	Yes	Yes
Singular Field	Yes	Yes	Yes

Table 7.6: Results of evaluation of most frequent static analysis errors

	Comprehension	**Relevance**	**Usefulness**
Total	29	28	27
Percentage	96.7%	93.3%	90%

Table 7.7: Summary of results of evaluation across topics

In the second type of experiment, we used different phrasings of the same question to test if Maria is able to respond to the different forms of the question and see if the response is relevant to the question. For example, what is a incompatible type error?, what causes incompatible type error? and how can I fix incompatible type error? are three different questions that asks about the incompatible error but have different meanings to it. We have conducted the experiment with the null pointer exception. Table 7.8 contains the different questions we asked Maria.

As we can see from Table 7.8, Maria was able to answer all the questions except for two.

Question	**Comprehension**	**Relevance**	**Usefulness**
What is a null pointer exception?	Yes	Yes	No
Can you tell me about nullpointerexception?	Yes	Yes	Yes
What causes null pointer exception?	Yes	Yes	Yes
How can I fix a null pointer exception?	Yes	Yes	Yes
Can you tell me about null pointer error?	Yes	Yes	Yes
What is null point exception?	No	N/A	N/A
What is a NPE?	No	N/A	N/A
Can you tell me about null pointer exception?	Yes	Yes	Yes
My program has a null pointer exception. How can I fix it?	Yes	Yes	Yes
Total	7	7	6
Percentage	77.8%	77.8%	66.7%

Table 7.8: Results of evaluation of different phrasings of questions

One of the questions that Maria was unable to answer had misspelled the word pointer and the other question used the abbreviation NPE that only a human subject matter expert would understand that it is NullPointerException. However, Maria is invariant to few forms of the same question. For example, it is able to provide a response to null pointer error and nullpointerexception instead of the actual "null pointer exception".

The response generated by Maria contained a one line explanation about null pointer exception as calling a method on an empty object followed by detailed explanation of the various scenarios that would cause it. So this is definitely related to the topic of null pointer exception and hence the relevance column has "Yes" for all the errors that Maria generated a

response. When it comes to usefulness, the one line explanation about the error does not provide an understanding of the error to the student. So it does not answer the "What is nullpointerexception" question sufficiently. Where as for all the other questions, since it provides the causes for the errors, it explains questions about the causes of the error and hence providing students sufficient knowledge to fix the errors as well.

As for the response speed of Maria, we simulated an event where we established 1000 individual TCP socket connections sequentially to ChatScript server and posted a request. This would simulate each user interaction with the ChatScript server. We recorded the time stamp at the before making the first request and after receiving the last response. We continued this experiment for 10 times and found that the mean time between the start of first request and the end of last response among all the experiments is 804 seconds. So on an average when a students asks a question to Maria, the student will receive a response back in 800 milliseconds which is very responsive for Chatbots.

Also in another experiment, we wanted to test the load handling capability of the ChatScript server. Since Web-CAT is used by many universities around the world, naturally the user load on the system is high and we wanted to make sure that our implementation is able to handle it. So we made 1000 asynchronous requests in parallel and the details about the number of concurrent requests and the response time are captured in Table 7.9. The result shows that our implementation is not only able to handle the load well but also provides a faster turnaround in response when there are multiple concurrent requests. We believe this speed up may be attributed to parallelizing of the handshaking process while establishing a TCP socket connection since this handshaking process involves the most overhead.

# of concurrent requests	**Total Time (ms)**	**Avg Time/Req (ms)**
10	756	75.6
100	7172	71.72
1000	43792	43.8

Table 7.9: Results of experiment testing server load handling capability

7.3.1 Limitations

This is only a representative study of the ability of Maria to respond to student questions. We picked the most frequently occurring errors that the student is most likely to ask Maria about. We also conducted an experiment with different types of phrasing of the question about the same error. But Students asking about different errors or phrasing the questions differently may change the change the result of this study. Also the biggest limitation with this study is it does not measure the effectiveness of the emotional support provided by the pedagogical agent. This is addressed in the next section where we describe a future study intended to measure the efficiency of Maria in providing emotional support.

7.4 Expert Review of Comments

As mentioned in the previous sections, Maria provides reinforcing/encouraging feedback to students about effort indicators and also provides sympathetic support in certain situations. Since we were unable to unable to get feedback from students regarding the effectiveness of these messages, we conducted an expert review (student who have taken the introductory programming courses and TAs for these courses) to carefully review these messages to see how students will perceive these messages and also their effectiveness in conveying the idea.

7.4.1 Review of Sympathetic Support Messages

We have identified 2 scenarios where Maria would provide sympathetic support to students and another scenario where Maria will recognize that students were able to overcome one of these scenarios. We had devised messages that Maria would provide for each of these scenarios. Few of the comments from experts about these messages are:

Do not make false promises: Few of the messages contained phrasings about students get a perfect score if they continued working on the assignment. However, there is no way

to guarantee that the student will get a perfect score if they keep working and that Maria should not be making false promises.

About effort required: There were a few messages that said only a little effort is required from the students to improve their scores. Again, Maria would not be able to measure the effort required by students to overcome the scenarios where they were stuck. Hence, Maria should not be making false claims.

Recognize the scenario: Earlier, when students were stuck, the message provided by Maria did not acknowledge that the students were stuck. This might confuse the students to see a message unexpectedly. So we made changes to acknowledge that the students were stuck in certain scenarios and proceed to provide the messages. Also in the scenarios where students were able to make progress, previously the messages did not provide any context as to why students would see them.

Remove phrases that can be taken in a negative connotation: One of the earlier messages had a phrase—"Keep working, champ! provided when students were continually getting zero scores in the assignment. In such scenario, students might take such messages to be sarcastic and will not receive such messages well in the future as well.

Table 7.10 shows the original messages for different scenarios and their corresponding modified versions after the expert review.

Original Message	**Modified Message**
Scenario: Students getting zero	
It is not uncommon that people get a low score on their first few submissions. In fact, a lot of people get low scores on their first submission and they all end up with near perfect scores. If you keep working, you will get a perfect score too.	It is not uncommon that people get a low score on their first few submissions. If you keep working hard, you might soon see your score increase.
The score is not an indicator of the good effort you put in this assignment. Only a little more effort is required to get a good score. Keep working, champ!	This score might not be an indicator of the good effort you put in this assignment. If you keep working, you might see an improved score that matches your effort.
The most important aspect of this assignment is the learning you get out of it. Keep focusing on learning, ask me about any questions you may need help with. I am confident you will get a good score.	The most important aspect of this assignment is the learning you get out of it. As you keep working, you will see an improved score to reflect the learning.
I know you have it in you to get a full score in this assignment. Only a little more effort is required. You can ask me about any errors you need help with and I can provide pointers to fixing it.	I know you have it in you to complete this assignment. I am confident that as you work hard, your score will improve.
Scenario: Students not making progress	
Alright! You have made it so far and you are almost done! A few more errors to fix and you will get a perfect score.	Alright! Looks like you have hit a roadblock. You can always visit a TA during the office hours to seek additional help.

Table 7.10: Original and modified messages for sympathetic support

Original Message	**Modified Message**
Lets take it one step at a time. Focus on that category of errors marked priority below and you will find out what I have I known all along that you can do it.	Alright! Looks like you have hit a roadblock. You can always visit a TA during the office hours to seek additional help.
You have put in great effort to get this far in the assignment. Take a small break and continue working on it and you will get a full score.	The most important aspect of this assignment is the learning you get out of it. As you keep working, you will see an improved score to reflect the learning.
I am confident you will definitely get a full score in this assignment. Keep working on it, champ!	It looks like you are not making sufficient progress in the assignment. Keep working hard, I am confident your score will improve.
Scenario: Students making progress after getting stuck	
I knew it all along that you had it in you to get a perfect score. Kudos!	Congrats, I can see that you are making progress again!
You worked hard to get this perfect score. Excellent work!	I see that you have crossed the hurdle. Nice work!
You are great for continuously working hard to get this score. Keep it up!	Kudos! It looks like you have crossed the hurdle!
Well done! You deserve this score for the effort you put in.	Congrats on overcoming the obstacle!

Table 7.10: Original and modified messages for sympathetic support

7.4.2 Review of Indicator Feedback

We also reviewed the different feedback messages that Maria will provide based on the triggering/untriggering of various effort indicators. Few of the comments about these feedback messages were:

Do not make comments about things not measured: Currently, the effort indicators are designed in a way that we would be able to tell that a indicator has been triggered but we do not have the mechanism to find the underlying cause of it. Simply speaking the indicators can be noisy. For example, there is an effort for increasing comments density. The previous messages assumed that the indicator was triggered because of students adding more comments to their code. But this indicator can also be triggered by students commenting out a part of their code. It would wrong to provide reinforcing feedback to students for something they did not achieve.

Maria should not express happiness: There were a lot of comments in which Maria expressed that she was happy some of the indicators were triggered in the student submissions. However, this would not help the students to any extent and it was decided that Maria should only observe and provide comments.

Do not assume scenarios: There were a few comments that were devised based on assumptions. For example, increasing solution methods could have been triggered by students more getter and setter methods. Many of the comments assumed that students modularized their code by splitting the longer methods into smaller methods. In case of getters and setters triggering this indicator, a comment about modularization of the code would be false and should be avoided.

Change noun: Many of the comments had first person plural noun—"we". Since the comments were coming from Maria, it was suggested that it was changed to first person singular noun.

Tables 7.11 to 7.21 shows the reinforcing comments that were designed earlier and the corresponding modified comments after the review.

Original Message	**Modified Message**
We see that you have modularized your solution by adding more methods to your classes. This is a great way to make your code more versatile and maintainable.	I see that you have added more methods to your solution classes. This is a good practice to make the code more versatile and maintainable.
Adding more methods to a solution helps to localize potential errors and identify bugs quickly. We are pleased to see a more modularized design in your submission.	I see more methods added to complete the solution. Adding more methods to a solution helps to localize potential errors and identify bugs quickly.
Excellent work distributing the functionality of your program among multiple methods. This will help to make your code easier to reuse and debug.	I notice that you have extra methods in your solution classes than before. This will help to make the code easier to reuse and debug.

Table 7.11: Original and modified reinforcing messages for adding new solution methods

Original Message	**Modified Message**
Great job on refactoring your solution by reducing the number of if-statements and loops. This is a great way to ensure that your methods work as expected.	I noticed that you have simplified some of the logic in your program. A code having less complexity is easy to test.
Removing excessive if-statements and loops is key in reducing the number of branches your program can take. Good job on simplifying your program.	I see that you have removed some of the logic in your program. Less complex code is always easier understand.
We are pleased to see that your solution is cleaner and less complex than before. Avoiding unnecessary branches and loop iterations is a great way to make your code more understandable.	I observe that your solution is less complex than before. Avoiding unnecessary branches and loop iterations is a great way to make your code more understandable.

Table 7.12: Original and modified reinforcing messages for reducing cyclomatic complexity

Original Message	**Modified Message**
Good job on reducing the average number of statements per method in your solution. This will make it easier to test your code.	I observe that you have smaller methods than before. This will make it easier to test your code.
We see that your have made your code more readable by reducing the average number of statements per method in your solution. Well done!	It appears that your methods contain fewer statements than before. This is good way to improve the readability of your code.
We are pleased to see that your methods generally contain fewer statements than before. This is a good practice to exercise because it prevents any one method from having too much capability.	I see that your methods generally contain fewer statements than before. This is a good practice to exercise because it prevents any one method from having too much capability.

Table 7.13: Original and modified reinforcing messages for reducing average method size

Original Message	**Modified Message**
Well done adding explanatory details to your code. This helps with comprehensibility of your program.	I noticed that your program has more comments than before. If used correctly, comments help with comprehensibility of your program.
We are glad to see that your submission contains more comments than before. Adding detailed comments helps people to better understand your program and its intended behavior.	I see that your submission contains more comments than before. Comments describing parts of your code helps people to better understand your program and its intended behavior.
Great job on supplementing your code with helpful comments. A well-documented solution is key in making your program easy to follow.	I observe that your code has more comments than before. Comments, when used properly, can make your program easier to follow

Table 7.14: Original and modified reinforcing messages for increasing comments density

Original Message	**Modified Message**
We see that you have added more classes to your solution, which is a key practice in object oriented programming.	I see that you have added more classes to your solution. Practicing writing more classes will help you make a better Object-Oriented programmer.
We see that you've taken a important step towards an ideal solution by dispersing the functionality of your code among multiple different classes. Well done!	I notice that there are more classes in your code than before. Dispersing the functionality of your code among multiple classes is always a good practice.
Great job on embracing the object-oriented spirit by adding more classes to your solution. Having more types of objects to use makes your program easier to conceptualize.	I observe that you have more classes in your solution than before. Having more types of objects to use makes your program easier to conceptualize.

Table 7.15: Original and modified reinforcing messages for increasing solution classes

Original Message	**Modified Message**
Good job on increasing your score by passing more instructor provided reference tests.	Good job on increasing your score by passing more instructor provided reference tests.
Your program is starting to conform more to the project's requirements. Great job!	Your program is starting to conform more to the project's requirements. Nice work!
We see that your program is producing more of the desired behavior stated in the project specification. Well done!	I see that your program is producing more of the desired behavior stated in the project specification. Well done!

Table 7.16: Original and modified reinforcing messages for increasing solution correctness

Original Message	**Modified Message**
Having many test methods reflects that you know how your program behaves given different conditions. Good job on adding new test methods.	Having many test methods reflects that you know how your program behaves given different conditions. Good job on adding new test methods.
Great work adding more methods to your test cases. Your tests are now less dependent on the behavior of your program, and so changing the behavior will require less maintenance.	I see you have added more methods to your test cases. Having more test methods independent on the behavior of your program will require less maintenance.
We see that you have broken up your tests into multiple additional test methods. This will make it easier for you to see which specific tests are passing or failing. Well done!	I see that you have more test methods than before. Having different methods to test different scenarios will help with identifying the failing scenarios easily.

Table 7.17: Original and modified reinforcing messages for adding new test method(s)

Original Message	**Modified Message**
We see that you have added more statements to your existing tests to cover the finer details of your program's behavior that are easy to overlook. Keep it up!	I see that you have added more statements to your existing tests. It is a good practice to test your code extensively.
It appears you are moving closer towards covering all possible conditions with your test methods. Well done!	I observe that you have more statements in your tests than before. Testing more scenarios is a good way to ensure that your program is working as expected.
Great job adding statements to your existing tests to hit more lines of code in your solution.	I notice that you have added more statements to your existing tests. Having a good code coverage by adding more tests helps build confidence that your program works as intended.

Table 7.18: Original and modified reinforcing messages for adding new statements to existing tests

Original Message	**Modified Message**
It appears that your tests are now executing more of your methods. Well done!	It appears that your tests are now executing more of your methods. Well done!
We can see that you constructed your tests more carefully so that they trigger more of your methods. This is a great practice!	I can see that you constructed your tests more carefully so that they trigger more of your methods. This is a good programming practice!
Good work enhancing your tests by triggering a larger number of your solution methods than before.	It appears that you have enhanced your tests by triggering a larger number of your solution methods than before.

Table 7.19: Original and modified reinforcing messages for increasing method coverage

Original Message	**Modified Message**
It is always smart to test every independent path of your program to identify potential bugs. We see that your software tests are covering more conditional branches than before. Well done!	It is always smart to test every independent path of your program to identify potential bugs. I see that your software tests are covering more conditional branches than before.
Great job identifying more branches that your program may potentially take. This is an important practice in unit testing.	Good job identifying and testing more branches that your program may potentially take. This is an important practice in unit testing.
Excellent work hitting a greater percentage of your solution branches with your software tests.	Nice work hitting a greater percentage of your solution branches with your software tests.

Table 7.20: Original and modified reinforcing messages for increasing conditional coverage

Original Message	**Modified Message**
Adding more assertions is a great way to test your code against various scenarios. We are glad to see that you have added more assertions.	I see you have added more assertions in your tests. Adding more assertions is a good way to test your code against various scenarios.
We can see that you have added more assertions to your tests to check the correctness for different inputs to your code. Keep it up!	I see that you have added more assertions to your tests to check the correctness for different inputs to your code.Good work!
Great job making your software tests more effective by adding more assertions.	Writing a thorough test to ensure your program is working in all scenarios is a good unit testing practice. I see you have made your software tests more effective by adding more assertions.

Table 7.21: Original and modified reinforcing messages for increasing assertion coverage

We repeated the same review for encouraging comments and Tables 7.22 to 7.32 display the original and modified messages for each of the indicators.

Original Message	**Modified Message**
A modularized solution will help you with code reuse and maintainability. We encourage you to refactor your code into more methods.	A modularized solution will help you with code reuse and maintainability. I encourage you to refactor your code into more methods.
There are instances in your submission in which large blocks of code are contained in a single method. This may lead to difficulty in identifying potential bugs; we recommend splitting your functionality among multiple methods.	Adding more methods to a solution helps to localize potential errors and identify bugs quickly. I recommend splitting your functionality among multiple methods.
Wherever you see particularly large blocks of code, try to separate the logic into multiple distinct methods. Modularizing your code in this fashion will make your program easier to reuse and maintain.	Wherever you see particularly large blocks of code, try to separate the logic into multiple distinct methods. Modularizing your code in this fashion will make your program easier to reuse and maintain.

Table 7.22: Original and modified encouraging messages for adding new solution methods

Original Message	**Modified Message**
Testing becomes easier if you have fewer paths to test. We recommend that you simplify your program to reduce the number of if-statements and loops used.	If your tests are taking too long to write, you might want to consider reducing the logic in your program that needs to be tested.
It appears that the logic of your solution can be implemented using fewer conditional branches and loops. Try to identify and condense these statements to make your code less complex.	I recommend you practice writing code with minimum logic necessary as it is a great way to improve your skill.
To minimize any unwanted behavior that your program may have, we recommend reducing the number of conditional branches and loops in your code.	If your logic in the code is overly complex, simplifying it will help with ease of testing.

Table 7.23: Original and modified encouraging messages for reducing cyclomatic complexity

Original Message	**Modified Message**
You can create a simpler solution than the one you have right now. Try to look at ways to reduce the number of lines per method while achieving the same functionality.	Having smaller methods can improve the readability of the code. See if you can make your methods smaller than they are now
It appears that some of your methods contain many lines of code. Making these methods smaller will make it easier to identify potential bugs.	Making the methods smaller will make it easier to identify potential bugs. I would recommend you to check if you can make your methods smaller.
Some of your methods have a large number of statements. Don't hesitate to create new methods to disperse the functionality covered by your existing methods.	Having smaller methods makes it easier to test your code. Check if there are ways to make your methods smaller.

Table 7.24: Original and modified encouraging messages for reducing average method size

Original Message	**Modified Message**
We would recommend that you add more comments to your program to make it easier to understand your logic.	I would recommend that you add more comments to your program to make it easier to understand your logic.
There appear to be large blocks of code in your program that are not supplemented by any comments. We recommend adding comments to these sections to make your code easier to follow.	There appear to be large blocks of code in your program that are not supplemented by any comments. I recommend adding comments to these sections to make your code easier to follow.
To make your program clearer to an outside reader, identify and explain blocks of code throughout your solution that are missing comments.	To make your program clearer to an outside reader, identify and explain blocks of code throughout your solution that are missing comments.

Table 7.25: Original and modified encouraging messages for increasing comments density

Original Message	**Modified Message**
Object oriented programming is a powerful paradigm in which real world objects are treated as separate classes. See if you can refactor to your code by adding more classes to match this real world behavior.	Object oriented programming is a powerful paradigm in which real world objects are treated as separate classes. See if you can refactor to your code by adding more classes to match this real world behavior.
To improve the modularity of your solution, we recommend encapsulating some of your program's functionality into additional classes.	To improve the modularity of your solution, I recommend encapsulating some of your program's functionality into additional classes.
It appears you could use a few more classes in your program to separate its various phases. This is a better design that lumping all your functionality into a small number of classes.	It appears you could use a few more classes in your program to separate its various phases. This is a better design than lumping all your functionality into a small number of classes.

Table 7.26: Original and modified encouraging messages for increasing solution classes

Original Message	**Modified Message**
It looks like your code is not passing a few of the instructor-provided tests. Pay closer attention to the requirements of the project to pinpoint where your program is not behaving as desired.	It looks like your code is not passing a few of the instructor-provided tests. Pay closer attention to the requirements of the project to pinpoint where your program is not behaving as desired.
To maximize the points you can earn, look at the reference tests you are currently failing and choose one of them to investigate. Once you figure out how to pass this test, move onto the others.	I see you are losing points for your code not passing instructor reference tests. Check the assignment requirements carefully to see how your program should behave.
It appears there are still some reference tests you are not passing. We recommend going through these tests and ensuring that your code can handle the various scenarios.	It appears there are still some reference tests you are not passing. You can visit a TA during the office hours to know more about the reference tests not passing.

Table 7.27: Original and modified encouraging messages for increasing solution correctness

Original Message	**Modified Message**
Adding more methods to your tests enables you to change the behavior of your program without making many modifications to your tests. See if you can modularize your tests by adding more methods.	Adding more methods to your tests enables you to change the behavior of your program without making many modifications to your tests. See if you can modularize your tests by adding more methods.
It appears that a lot of the branches in your code are being covered within single test methods. We recommend breaking up these tests into multiple additional methods to make it easier to see which specific tests are passing or failing.	Modularizing your test methods can be an effective way to quickly identify failing scenarios. See if you can add more test methods to break down the different scenarios.
To make the behavior of your program more clear, try adding more test methods to cover as many cases as possible.	To make the behavior of your program more clear, try adding more test methods to cover as many cases as possible.

Table 7.28: Original and modified encouraging messages for adding new test method(s)

Original Message	**Modified Message**
It looks like there is still scope for adding finer details to your existing tests. We would encourage you to look into these.	Writing tests that cover all the scenarios helps you improve as a programmer. I would recommend adding to your tests to account for the different scenarios.
It appears that your test methods do not quite account for all possible conditions your program may encounter. We recommend adding these missing cases to your test methods.	Having tests that cover your entire program is a good way to ensure your program is working as intended. See if you can add to your tests to improve coverage.
To cover more lines of code in your solution, look at the branches that are not currently being tested and add statements to your existing tests	It is a good practice to test all the branches in your code. See if you can add to your tests to cover more branches

Table 7.29: Original and modified encouraging messages for adding new statements to existing tests

Original Message	**Modified Message**
It looks like your tests are not covering all the methods in your program. We would recommend you to add more tests so that all of your solution methods are covered.	Writing tests to cover more methods in your program is a good practice to ensure your program is well tested. I would encourage you to write tests to cover more of your methods if not done currently.
It looks like there are several methods that are not being triggered by your tests. Try to set up scenarios that would force these methods to be executed.	If your tests are not covering all of your methods, I would recommend you to write more tests covering all of your methods.
To increase your code coverage, we encourage you to write tests that execute each of the different methods in your solution.	Writing tests to cover all your methods is a good way to improve your code coverage. See if you can write tests to cover all your solution methods.

Table 7.30: Original and modified encouraging messages for increasing method coverage

Original Message	**Modified Message**
A great way to discover hidden bugs is to write tests which cover all the statements and branches of your program. We would encourage you to write tests so that all the branches in your program are covered.	A great way to discover hidden bugs is to write tests which cover all the statements and branches of your program. I would encourage you to modify tests if they are not covering all the branches currently.
It appears there are some conditions that your tests do not account for. We suggest that you look at your tests and consider all possible branches your program can take.	A good unit testing practice is to cover all of your conditions in your program. If your tests are not covering all your conditions, see if you can increase your tests to cover all of your conditions.
Your tests are successfully hitting some of the branches in your solution, but not all of them. We encourage you to go through your if-statements, methods, and loops and ensure your tests are accounting for all possible inputs.	I encourage you to check if your tests are covering all your if-statements, methods, and loops and ensure your tests are accounting for all possible inputs. This is a good way to ensure your solution is working as expected.

Table 7.31: Original and modified encouraging messages for increasing conditional coverage

Original Message	**Modified Message**
Some of your methods are not being fully covered by your tests. To improve this, consider all different types of inputs for these methods and add assertions that execute them.	It is always a good practice to test your program for various input conditions. See if you can add more assertions to test different scenarios.
It appears there are some conditions that your test methods fail to account for. We suggest adding more assertions to these methods to trigger all possible branches.	I would suggest you to add more assertions to see if different flows in your program are triggered which were not triggering before.
To make your software tests more effective, we recommend adding more assertions to these tests that execute a larger percentage of your code branches.	To make your software tests more effective, I recommend adding more assertions to these tests that execute a larger percentage of your code branches.

Table 7.32: Original and modified reinforcing messages for increasing assertion coverage

As for the indicator for resolving static analysis errors, it was decided that the existing messages are not suitable for use as they are many types of static analysis errors that students can encounter and it would be incorrect to talk about a different static analysis error. So we enhanced the design to identify the different types of static analysis errors and also devised new messages to provide in each of these scenarios. Table 7.33 shows the different static analysis errors and the new feedback messages.

7.5 Evaluation of Emotional Support Goals

Providing emotional support to students is the main goal of this project. To evaluate this goal, we have to evaluate the effectiveness of Maria in promoting growth mindset, if sympathetic support helped reduce students frustration and enabled them to continue working on their assignment and if Maria was able to explain all Java errors. To evaluate Maria's ability to explain all errors, we can analyze the ChatScript detailed logs to see if all requests coming in generated a response and if the response was useful to the student. Since both the explain links and the chatbot use ChatScript to provide explanations, this would be a good method to evaluate the explanations. The other sub-goals involve assessing the student emotions during the specific scenarios. To evaluate this, we will conduct a carefully crafted user study. We will design assignments where students will encounter scenarios in which they will receive the sympathetic support and feedback about the performance indicators. We will capture the students' responses after their interaction with Maria to evaluate how effective Maria was in achieving these sub-goals.

Error Type	**Reinforcing Feedback**	**Encouraging Feedback**
JavaDoc	I see that you have more JavaDocs to your solution. Adding proper JavaDocs is a very good to help readers better understand your program	Having JavaDocs with a concise description of the class/method helps with understanding your code better. I recommend you to add more JavaDocs with meaningful description.
Formatting / WhiteSpace	Good work on fixing several of your styling errors. A program with proper indentation and spacing is more legible.	Did you know that code is read more than it is written? It would help improve the readability of your program if you added proper indentation and spacing.
Naming	It appears that you have used meaningful names to your variables in the program. Having meaningful names makes it easier to follow your program	Having meaningful names can help improve comprehensibility of your program. See if you can rename your variables more meaningfully.
Readability	I observe that you have added the optional braces to your code. It is a good practice to write code with braces as it avoids potential errors	Did you know missing optional braces can lead to potential errors in your program? I suggest you to add the optional braces as it is a good programming practice.
Coding Style	I see that you have tailored parts of your code to comply with best programming practices. It is usually a great way to make your solution more readable.	Following the best programming practices helps you improve as a programmer. See if you can modify your code to conform to best coding practices.

Table 7.33: Static analysis error types and feedback messages

Chapter 8

Conclusion

In this thesis, we have outlined and described the development and use of a virtual teaching assistant (Maria) in Web-CAT to emotionally support students in their programming assignments. We explained the existing problems that students face while working on assignments and provided the motivating factors for us to do this project. We conducted a literature study of how students struggle with programming, the effectiveness of pedagogical agents in learning environments and how they can be efficiently used to motivate the students. We had outlined the design goals that we strived to achieve in order to achieve the overall goal of the project. We discussed in length about the the different design choices we had to make to achieve those design goals. We provided a brief overview of the implementation of these design goals. We provided a User Interface walk through of how students in different scenarios will interact with Maria. We described our evaluation plan and performed an evaluation of the effectiveness of Maria. We also expressed our plan to conduct a user survey and user study.

8.1 Contribution

Our main contribution in this work is Maria—a virtual teaching assistant who will be able to alleviate some of the negative emotions associated with learning programming. Few of the issues, as highlighted in the problem statement, that contribute to this negative experience are lack of emotional support in automatic grading systems, student's inability to understand error messages, students having a fixed mindset and student frustrations with automated grading tools. By incorporating Maria into Web-CAT, who can provide error explanations for all the errors, provide feedback about effort indicators to promote a growth mindset and provide sympathetic support, we address the above mentioned issues thereby assuaging some of the negative experience with learning programming. As part of this work, we have introduced a highly researched and heavily used chatbot into Web-CAT. This paves way for a lot of further research work on how these chat bots can be designed to help students in computer science.

8.2 Future Work

As mentioned above, this project has implemented a chat bot style teaching assistant to Web-CAT with limited functionalities. We have a lot of scope for future work that we highlight below:

Fully Conversational Agent: We see this as the biggest failing currently. Maria does not have the ability to have a casual conversation with the student or the ability to initiate a conversation. Many successful chat bots have that feature and we would like that to be incorporated into Maria as well by adding information about a broad range of topics into ChatScript rules.

Support for other Programming Languages: Currently, Maria is only focused on answering questions related to Java only. Since, we use Web-CAT for programming courses

based on other languages also, we would want to extend Maria's capabilities to be able to answer questions related to those programming languages as well.

Implementation of Motivating Feedback: We have laid the ground work for Maria to provide comments to students when they feel unmotivated. In the future, we would want this to be expanded to chalk out the scenarios clearly in which Maria can motivate students.

Details about Assignment: It would be useful to students if Maria can provide more details about the assignment like specifications, due date and time etc. We also believe providing information such as mean score in the assignment, average number of submissions per student will also be useful in gauging themselves.

Intervention: One other feature that we think will be useful to students is when Maria initiates a conversation encouraging students to start working on assignments early reminding them about the due date.

Survey Plan: We plan to do a survey in the near future involving human subjects. We will deploy Maria to Web-CAT and have students of CS1114 use Maria and roll out a survey. The actual effectiveness and perception of Maria can only be measured by getting feedback from students. We feel carefully designed survey will help us understand how students perceive Maria, how they adapt to using Maria, how Maria is helpful etc. The survey was designed to assess our various design goals. It consisted of 12 questions and students can choose one of the five responses: Strongly Agree, Agree, Neither Agree not Disagree, Disagree and Strongly Disagree. To elicit reliable results, we have a mix of questions with both positive and negative connotations.

Bibliography

[1] Bot libre. https://www.botlibre.com/. Accessed: 2018-05-15.

[2] Canadian mind products java & internet glossary. http://mindprod.com/jgloss/jgloss.html. Accessed: 2018-05-15.

[3] Chatscript. https://github.com/bwilcox-1234/ChatScript. Accessed: 2018-05-15.

[4] Wikipedia pedagogical agent. https://en.wikipedia.org/wiki/Pedagogical_agent/. Accessed: 2018-05-15.

[5] S. Apostol, O. Soica, L. Manasia, and C. Stefan. Virtual pedagogical agents in the context of virtual learning environments: Framework and theoretical models. volume 2, pages 531–536, Bucharest, 2013. "Carol I" National Defence University.

[6] I. Arroyo, B. P. Woolf, J. M. Royer, and M. Tai. Affective gendered learning companions. In *Proceedings of the 2009 Conference on Artificial Intelligence in Education: Building Learning Systems That Care: From Knowledge Representation to Affective Modelling*, pages 41–48, Amsterdam, The Netherlands, The Netherlands, 2009. IOS Press.

[7] R. K. Atkinson. Optimizing learning from examples using animated pedagogical agents. 2002.

[8] A. Bandura. *Self-efficacy: The exercise of control.* New York: W.H. Freeman, 1997.

[9] A. Bandura. Social cognitive theory: An agentic perspective. 2:21 – 41, 12 2002.

[10] A. L. Baylor. The impact of three pedagogical agent roles. In *AAMAS*, 2003.

[11] A. L. Baylor. The design of motivational agents and avatars. *Educational Technology Research and Development*, 59(2):291–300, Apr 2011.

[12] A. L. Baylor and Y. Kim. Pedagogical agent design: The impact of agent realism, gender, ethnicity, and instructional role. In J. C. Lester, R. M. Vicari, and F. Paraguaçu, editors, *Intelligent Tutoring Systems*, pages 592–603, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[13] N. Bosch, S. D'Mello, and C. Mills. What emotions do novices experience during their first computer programming learning session? In H. C. Lane, K. Yacef, J. Mostow, and P. Pavlik, editors, *Artificial Intelligence in Education*, pages 11–20, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[14] N. C. C. Brown, M. Kölling, D. McCall, and I. Utting. Blackbox: A large scale repository of novice programmers' activity. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 223–228, New York, NY, USA, 2014. ACM.

[15] T. Carlotto and P. A. Jaques. The effects of animated pedagogical agents in an english-as-a-foreign-language learning environment. *Int. J. Hum.-Comput. Stud.*, 95(C):15–26, Nov. 2016.

[16] M. P. Driscoll. *Psychology of Learning for Instruction.* Boston : Allyn and Bacon, 2000.

[17] M. C. Duffy and R. Azevedo. Motivation matters: Interactions between achievement goals and agent scaffolding for self-regulated learning within an intelligent tutoring system. *Computers in Human Behavior*, 52:338 – 348, 2015.

[18] C. S. Dweck. *Mindset:The New Psychology Of Success.* New York : Ballantine Books, 2008.

[19] S. Edwards and Z. Li. Towards progress indicators for measuring student programming effort during solution development. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, Koli Calling '16, pages 31–40, New York, NY, USA, 2016. ACM.

[20] S. H. Edwards. Improving student performance by evaluating how well students test their own programs. *J. Educ. Resour. Comput.*, 3(3), Sept. 2003.

[21] S. H. Edwards. Using software testing to move students from trial-and-error to reflection-in-action. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '04, pages 26–30, New York, NY, USA, 2004. ACM.

[22] S. H. Edwards. Work-in-progress: Program grading and feedback generation with web-cat. In *Proceedings of the First ACM Conference on Learning @ Scale Conference*, L@S '14, pages 215–216, New York, NY, USA, 2014. ACM.

[23] S. H. Edwards, N. Kandru, and M. B. Rajagopal. Investigating static analysis errors in student java programs. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER '17, pages 65–73, New York, NY, USA, 2017. ACM.

[24] C. Elliott, J. Rickel, and J. Lester. *Lifelike Pedagogical Agents and Affective Computing: An Exploratory Synthesis*, pages 195–211. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[25] A. Følstad and P. B. Brandtzæg. Chatbots and the new world of hci. *Interactions*, 24(4):38–42, June 2017.

[26] P. Greenfield. A theory of the teacher in the learning activities of everyday life. 01 1984.

[27] M. Haungs, C. Clark, J. Clements, and D. Janzen. Improving first-year success and retention through interest-based cs0 courses. In *Proceedings of the 43rd ACM Technical*

Symposium on Computer Science Education, SIGCSE '12, pages 589–594, New York, NY, USA, 2012. ACM.

[28] J. M. Keller. Development and use of the arcs model of instructional design. *Journal of instructional development*, 10(3):2, Sep 1987.

[29] Y. Kim. Pedagogical agents as learning companions: Building social relations with learners. In *Proceedings of the 2005 Conference on Artificial Intelligence in Education: Supporting Learning Through Intelligent and Socially Informed Technology*, pages 362–369, Amsterdam, The Netherlands, The Netherlands, 2005. IOS Press.

[30] Y. Kim and A. L. Baylor. A social-cognitive framework for pedagogical agents as learning companions. *Educational Technology Research and Development*, 54(6):569–596, Dec 2006.

[31] N. Kndru. Intelligent Goal-Oriented Feedback for Java Programming Assignments. Master's thesis, VirginiaTech, Blacksburg, VA, 2018.

[32] E. Kurvinen, N. Hellgren, E. Kaila, M.-J. Laakso, and T. Salakoski. Programming misconceptions in an introductory level programming course exam. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '16, pages 308–313, New York, NY, USA, 2016. ACM.

[33] J. C. Lester, S. A. Converse, B. A. Stone, S. E. Kahler, and S. T. Barlow. Animated pedagogical agents and problem-solving effectiveness: A large-scale empirical evaluation. In *IN PROCEEDINGS OF EIGHTH WORLD CONFERENCE ON ARTI CIAL INTELLIGENCE IN EDUCATION*, pages 23–30. Press, 1997.

[34] A. M. Johnson, G. Ozogul, R. Moreno, and M. Reisslein. Pedagogical agent signaling of multiple visual engineering representations: The case of the young female agent. 102, 04 2013.

[35] E. Matusov and R. Hayes. Sociocultural critique of piaget and vygotsky. 18:215–239, 08 2000.

[36] K. N Moreno, R. Van Eck, N. Person, G. Jackson, A. Adcock, and J. C Marineau. Etiquette and efficacy in animated pedagogical agents: The role of stereotypes. 05 2002.

[37] M.-H. Nienaltowski, M. Pedroni, and B. Meyer. Compiler error messages: What can help novices? In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '08, pages 168–172, New York, NY, USA, 2008. ACM.

[38] C. Okonkwo. Affective pedagogical agents and user persuasion. 2003.

[39] E. Patitsas and P. Belleville. What can we learn from quantitative teaching assistant evaluations? In *Proceedings of the Seventeenth Western Canadian Conference on Computing Education*, WCCCE '12, pages 36–40, New York, NY, USA, 2012. ACM.

[40] N. Pillay, R. Jugoo, and Vikash. An analysis of the errors made by novice programmers in a first course in procedural programming in java. 2006.

[41] A. Robins, J. Rountree, and N. Rountree. Learning and teaching programming: A review and discussion. 13:137–, 06 2003.

[42] C. Scott, R. Harris, and A. R. Rothe. Embodied cognition through improvisation improves memory for a dramatic monologue. 31:293–305, 05 2001.

[43] C. Senko and J. M. Harackiewicz. Regulation of achievement goals: The role of competence feedback. In *Journal of Educational Psychology*, volume 97, pages 320–336, 2005.

[44] E. Shaw, W. L. Johnson, and R. Ganeshan. Pedagogical agents on the web. In *Proceedings of the Third Annual Conference on Autonomous Agents*, AGENTS '99, pages 283–290, New York, NY, USA, 1999. ACM.

[45] V. J. Traver. On compiler error messages: What they say and what they mean. *Adv. in Hum.-Comp. Int.*, 2010:3:1–3:26, Jan. 2010.

[46] B. Tversky and B. M. Hard. Embodied and disembodied cognition: Spatial perspective-taking. *Cognition*, 110(1):124 – 129, 2009.

[47] H. Van Der Meij. Motivating agents in software tutorials. *Comput. Hum. Behav.*, 29(3):845–857, May 2013.

[48] H. van der Meij, J. van der Meij, and R. Harmsen. Animated pedagogical agents effects on enhancing student motivation and learning in a science inquiry learning environment. *Educational Technology Research and Development*, 63(3):381–403, Jun 2015.

[49] L. S. Vygotsky, M. Cole, V. John-Steiner, S. Scribner, and E. Souberman. *Mind in Society*. Cambridge, MA: Harvard University Press, 1978.

[50] L. E. Winslow. Programming pedagogy - a psychological overview. *SIGCSE Bull.*, 28(3):17–22, Sept. 1996.

[51] A. Xu, Z. Liu, Y. Guo, V. Sinha, and R. Akkiraju. A new chatbot for customer service on social media. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 3506–3510, New York, NY, USA, 2017. ACM.

Appendix A

Text of Survey

Block 1

Survey on Web-CAT Feedback

This voluntary survey includes questions regarding your opinions on the automated feedback you received on your programming assignments. We will use this information to understand better how you view that feedback, and whether it is helpful to you.

The results from this survey will be used for research purposes and for improving the feedback you receive as you work on assignments. Please complete all items, even if you feel that some are redundant. This may require 5 minutes of your time. Usually it is best to respond with your first impression, without giving a question much thought. Your answers will remain confidential, and will not affect your grade in any way.

Your participation is voluntary. If you do not wish to participate, simply do not fill out the survey. You must be 18 or older to take part in this research.

Thank you for your participation.

Complete the following items. Select the option that best describes how strongly you **agree** or **disagree**.

Block 2

In some of your programming assignments, Web-CAT used a **new feedback presentation** to present the issues in your code in a different way:

In some of your programming assignments, **Maria the Virtual TA** was available on your feedback page.

Maria was able to describe details about specific compile-time and run-time errors and may have provided feedback about your work. Please indicate your agreement/disagreement with the following statements related to **Maria the Virtual TA**.

	Strongly agree	Agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Disagree	Strongly disagree
Maria's presence in my Web-CAT feedback is distracting	○	○	○	○	○	○	○
Maria provided helpful explanations of compiler errors and runtime exceptions	○	○	○	○	○	○	○
Maria's conversation is annoying	○	○	○	○	○	○	○
Answers from Maria were relevant to my questions	○	○	○	○	○	○	○
Maria helped me resolve errors	○	○	○	○	○	○	○
Maria was helpful in making progress on the assignment	○	○	○	○	○	○	○
Maria's audio voice was irritating during conversations	○	○	○	○	○	○	○
I found Maria's comments to be encouraging	○	○	○	○	○	○	○
Maria was easy to access and use	○	○	○	○	○	○	○
Maria's comments were relevant to my situation and the current state of my work on the assignment	○	○	○	○	○	○	○
I want a virtual TA who is more capable than Maria	○	○	○	○	○	○	○
I did not use Maria	○	○	○	○	○	○	○

How can the virtual TA be improved to help you better?

Appendix B

IRB Approval

Office of Research Compliance
Institutional Review Board
North End Center, Suite 4120
300 Turner Street NW
Blacksburg, Virginia 24061
540/231-3732 Fax 540/231-0959
email irb@vt.edu
website http://www.irb.vt.edu

MEMORANDUM

DATE: May 4, 2018

TO: Stephen H Edwards, Catherine Amelink, Bob Edmison, Michael Scott Irwin, Zhiyi Li, Nischel Kandru, Mukund Babu Manniam Rajagopal

FROM: Virginia Tech Institutional Review Board (FWA00000572, expires January 29, 2021)

PROTOCOL TITLE: Collaborative Research: Using Automated Feedback to Promote a Growth Mindset in Programming Assignments (NSF 1625425)

IRB NUMBER: **16-014**

Effective April 25, 2018, the Virginia Tech Institution Review Board (IRB) approved the Amendment request for the above-mentioned research protocol.

This approval provides permission to begin the human subject activities outlined in the IRB-approved protocol and supporting documents.

Plans to deviate from the approved protocol and/or supporting documents must be submitted to the IRB as an amendment request and approved by the IRB prior to the implementation of any changes, regardless of how minor, except where necessary to eliminate apparent immediate hazards to the subjects. Report within 5 business days to the IRB any injuries or other unanticipated or adverse events involving risks or harms to human research subjects or others.

All investigators (listed above) are required to comply with the researcher requirements outlined at:

http://www.irb.vt.edu/pages/responsibilities.htm

(Please review responsibilities before the commencement of your research.)

PROTOCOL INFORMATION:

Approved As:	**Expedited, under 45 CFR 46.110 category(ies) 5,7**
Protocol Approval Date:	**May 13, 2017**
Protocol Expiration Date:	**May 12, 2018**
Continuing Review Due Date*:	**April 28, 2018**

*Date a Continuing Review application is due to the IRB office if human subject activities covered under this protocol, including data analysis, are to continue beyond the Protocol Expiration Date.

FEDERALLY FUNDED RESEARCH REQUIREMENTS:

Per federal regulations, 45 CFR 46.103(f), the IRB is required to compare all federally funded grant proposals/work statements to the IRB protocol(s) which cover the human research activities included in the proposal / work statement before funds are released. Note that this requirement does not apply to Exempt and Interim IRB protocols, or grants for which VT is not the primary awardee.

The table on the following page indicates whether grant proposals are related to this IRB protocol, and which of the listed proposals, if any, have been compared to this IRB protocol, if required.

Date*	OSP Number	Sponsor	Grant Comparison Conducted?
06/01/2016	P534CP3I	National Science Foundation (Title: Collaborative Research: Using Automated Feedback to Promote a Growth Mindset in Programming Assignments)	Compared on 05/04/2016

* Date this proposal number was compared, assessed as not requiring comparison, or comparison information was revised.

If this IRB protocol is to cover any other grant proposals, please contact the IRB office (irbadmin@vt.edu) immediately.