TED UNIVERSITY

CMPE491/SENG491 Senior Project

SyntaxSavior Project Analysis Report

22.11.2024

Team Members:

İrem BEŞİROĞLU Software Engineering

Yiğit Oğuzhan KÖKTEN Computer Engineering

Yüksel Çağlar BAYPINAR Computer Engineering

Table of Contents

1 Introduction

This report aims to provide a comprehensive analysis of the proposed SyntaxSavior system, designed to enhance the learning experience for students enrolled in CMPE 113: Introduction to Programming at TED University. The primary objective of SyntaxSavior is to bridge the gap between theoretical programming knowledge and its practical application by offering real-time, context-aware support during laboratory sessions.

SyntaxSavior seeks to address common challenges faced by novice programmers, such as deciphering seemingly complex error messages, navigating the intricate interface of the Eclipse IDE, and avoiding repeated logical mistakes. By employing a dual-component structure—comprising an IDE plugin and a main analysis program—it provides tailored assistance that promotes conceptual understanding while preserving the integrity of the learning process. This document outlines the conceptual foundation of the project, clarifying its goals, scope, and potential impact on both students and lab instructors.

Through this analysis, we aim to refine the system's design, identify key challenges, and establish a roadmap for its development and implementation.

2 Current System

Students are currently provided with a step-by-step guide to using Eclipse for creating projects, defining classes, writing Java code, and executing their programs. They receive a concise description of their task requirements, along with samples of expected input and output. The objective is to replicate these samples by solving the given task, uploading their code to the Virtual Programming Lab (VPL) system in the TEDU Learning Management System (LMS), and passing the automated checks in place.

The laboratory sessions are divided into two parts:

- **First Hour (Optional, Ungraded):**
    - Students can receive tutor assistance upon request.
    - They are free to use any external resources to support their learning process.

    - This hour serves as a preparatory phase where students can explore solutions and clarify concepts.

- **Second Hour (Graded):**
    - Students are given a similar programming task.
    - They are restricted to using their lecture notes as the sole resource.
    - Tutor assistance is only provided for technical issues unrelated to the task itself.

2.1 Survey Results

Our survey of 59 students taking CMPE 113 in a laboratory setting revealed key insights into their experiences and preferences for programming assistance tools. A significant number of students (21 out of 59) reported needing assistance during every lab session, while 22 students indicated they frequently encounter syntax errors in their Java programming. Logical errors emerged as a major challenge, with many students (21 out of 59) typically seeking help from peers or instructors when stuck. While 32 students found teaching assistants and tutors sufficiently helpful, there remains a clear interest in supplementary resources to enhance their learning experience.

When asked about desirable features in a programming assistance tool, students expressed strong interest in functionalities such as detailed explanations of coding errors, hints for solving logical issues, topic-specific descriptions aligned with lab exercises, and precise identification and feedback on erroneous code. Many students (39 out of 59) felt lab exercises often help them better understand theoretical concepts, with perfect ratings of 5/5 for how well labs enhance their understanding of lecture material and their application of theoretical concepts in hands-on contexts.

Furthermore, students showed interest in using a tool like **SyntaxSavior**, with several indicating they may need it at some point and feeling encouraged to use it if it provides guidance rather than direct solutions. These findings suggest a promising role for tools like SyntaxSavior in addressing common challenges, fostering problem-solving skills, and further supporting the high educational value of laboratory sessions.

2.2 Upsides of the Current System

- The current system fosters independence by encouraging students to solve problems using their existing knowledge and limited resources.
- It simulates scenarios like technical interviews and exams where access to external support may be constrained.

- The structured division of the session allows students to prepare and practice before undertaking graded tasks.
- Automated checks ensure objective evaluation of the code's correctness.

2.3 Downsides of the Current System

- The complexity and presentation of error messages in Java and the unfriendly interface of Eclipse can overwhelm novice programmers, hindering their ability to troubleshoot effectively.
- Students often struggle with basic navigation and debugging in Eclipse, detracting from their focus on learning core programming concepts.
- The lack of contextual feedback or guidance in real time may lead to frustration and repeated errors.
- Limited tutor availability during the graded hour can leave students without support for conceptual misunderstandings, impacting their confidence and performance.

SyntaxSavior is not intended to replace this system but to enhance it by addressing these downsides. However, it is not a drop-in replacement; its implementation will require cooperation and adaptation on the part of both students and instructors. This tool aims to supplement the current framework by improving error comprehension, reducing technical barriers, and fostering a deeper understanding of programming concepts.

3 Proposed system

3.1 Overview

SyntaxSavior is a hybrid system consisting of an IDE plugin and a centralized program performing analysis. The tool presents guidance and feedback to students interactively in the programming environment, encouraging learning by problem-solving guidance without allowing them to rely too much on automated solutions. It addresses common challenges, such as deciphering Java error codes and navigating the complex user interface of the Eclipse IDE, by focusing on live code analysis tailored to the task at hand. The system also aims to broaden the understanding of key concepts by providing summaries and examples of subject materials regarding the current task and help create an intuitive understanding of programming in general for the students.

The implementation of SyntaxSavior comprises two core components: an IDE plugin and a main analysis program. Each of these components is designed to address specific aspects of the student learning process while ensuring seamless integration with existing workflows.

3.1.1 IDE Plugin

The IDE plugin is designed to provide immediate, context-aware assistance by:

Enhancing Eclipse's Accessibility:

- The plugin simplifies and streamlines Eclipse's user interface for novice users.
- Features like linting, code formatting, file browsing, and project structuring are made more intuitive for beginners by providing guidance on specific tools commonly used by the students.
- Navigation aids and context-sensitive tips minimize the overwhelming nature of Eclipse's extensive feature set, allowing students to focus on essential functionalities.

Providing Immediate Feedback:

- The plugin offers pop-up notifications and alerts for common syntax issues such as missing semicolons or parentheses, addressing them at an early stage to prevent frustration.
- It helps clarify potential false positives and assists students in understanding the relevance of specific errors in their code.

Communicating with the Main Program:

- Relevant information, such as live code snippets, project status, and student progress, is transmitted to the main program for deeper analysis.

3.1.2 Main Analysis Program

The main analysis program operates as the central engine for SyntaxSavior, offering in-depth, dynamic, and contextual support to students while they work on their programming tasks. Its primary functions include:

Live Feedback and Learning Support

- The program continuously analyzes student submissions for logical errors, stylistic issues, and recurring patterns of mistakes.

- Feedback is tailored to both the specific laboratory task and the lecture topics covered up to that point, ensuring alignment with the curriculum.
- Instead of providing ready-to-use code solutions, the program offers practical hints, step-by-step guidance, and conceptual clarifications to encourage independent problem-solving.

Dynamic Subject-Related Guidance

- To help students build a foundational understanding of programming, the system provides concise summaries and relevant examples of key concepts tied to their current progress.
- For instance, if a task involves reading user input and performing mathematical calculations, the system will dynamically display explanations of related topics, such as:
    - The purpose and usage of variables.
    - Defining functions and their role in organizing code.
    - Working with the Scanner class for user input.
- This dynamic guidance evolves as students' progress through their tasks, ensuring that the information presented remains pertinent and actionable.

Transforming Mistakes into Learning Opportunities

- By identifying areas where students struggle, the program provides targeted feedback that encourages reflection and correction.
- It aims to create an environment where mistakes are no longer roadblocks but stepping stones for deeper understanding.

Maintaining Contextual Relevance

- The feedback and guidance offered are carefully tailored to the current task and the course objectives, preventing information overload or confusion.
- The system avoids delving into advanced or unrelated topics, which can often distract students when they turn to external resources for help.

Encouraging Conceptual Understanding

- Beyond task-specific feedback, the program fosters a broader, more intuitive grasp of programming by reinforcing underlying principles.
- Through examples, hints, and concise explanations, students can better connect individual programming concepts to the larger context of software development.

3.2 Functional Requirements

Based on observations and feedback from students and tutors in CMPE 113, the key features for SyntaxSavior are:

- **Step-by-Step Guidance:**
    - Emphasis on providing hints and pointers instead of direct solutions to promote problem-solving skills.
    - Task-specific assistance tailored to the students' progress, such as contextual explanations of Java concepts like Scanner or Math.round.
- **Real-Time Error Detection and Categorization:**
    - Identify and categorize syntax errors, runtime exceptions, and logical errors.
    - Simplify error messages into beginner-friendly explanations with actionable suggestions.
- **Lab-Specific Concept Reinforcement:**
    - Dynamic concept explanations displayed in a side panel, based on task progress and topics covered in lectures.
    - Examples, brief summaries, and potential pitfalls for concepts related to the task at hand.
- **Enhanced Eclipse UI Integration:**
    - Non-intrusive design that integrates seamlessly into the Eclipse IDE.
    - Features like navigation hints for locating closed panels, managing project structures, and highlighting useful IDE tools for beginners.

3.3 Nonfunctional Requirements

- **Performance:**
    - The system should have near-instantaneous response times to provide live feedback without disrupting the student workflow.
- **Usability:**
    - Designed with first-year students in mind, including those with no prior programming experience.
    - Clear and intuitive UI that does not overwhelm users with unnecessary options.
- **Scalability:**
    - Capable of handling large classroom sizes and multiple simultaneous users without degrading performance.
- **Reliability:**
    - The system should operate smoothly during extended lab sessions without crashes or significant downtime.
- **Security and Fine Tuning:**
    - Implement safeguards to prevent misuse or abuse of the system, such as cheating or exploiting features to bypass learning objectives.
    - Provide administrative controls for instructors to configure system features as needed.
- **Accessibility:**
    - Ensure compatibility with accessibility tools and standards to accommodate diverse student needs. This may include language selection and translation tools.
- **Privacy Compliance:**
    - Adhere to data privacy regulations, ensuring that student data is stored and analyzed securely and used only for educational purposes.

3.4 Pseudo requirements

The following features are not essential for the initial implementation but could greatly enhance the system's utility:

- **Expanded IDE Support:**
    - Add compatibility with additional IDEs like IntelliJ IDEA or support for IDE-less environments.
- **Safe Exam Browser Integration:**
    - Implement a feature to block external unmoderated tools (e.g., ChatGPT) during exam or assessment sessions while still allowing access to SyntaxSavior.
- **Customizable Feedback Models:**
    - Allow instructors to adjust the level of feedback detail or focus on specific concepts, tailoring the system to their teaching style.
- **Gamified Elements:**
    - Add optional gamification features, such as achievements or progress tracking, to engage students and encourage independent learning.
- **Collaboration Tools:**
    - Include features for peer collaboration, such as shared troubleshooting spaces or discussion threads moderated by tutors.

3.5 System models

3.5.1 Scenarios

3.5.1.1 Tutor Overload

During peak periods, particularly in the initial weeks or when tackling challenging topics, tutors face a deluge of questions from students. Many queries are repetitive and stem from either a lack of familiarity with the tools (e.g., Eclipse) or misunderstandings of lecture material. This overload forces tutors to prioritize speed over depth, providing quick fixes rather than thorough explanations.

- **Expected Impact of SyntaxSavior:**
    - By handling common issues such as technical difficulties, basic syntax errors, and misunderstandings of simple concepts, SyntaxSavior will reduce the overall volume of questions directed at tutors.
    - Tutors can dedicate more time to complex, nuanced problems, ensuring students receive higher-quality assistance when needed.

3.5.1.2 Lab Task Clarification

Students frequently struggle with interpreting lab task requirements, particularly edge cases or ambiguous instructions. The recurring question of "How do I begin?" or "What are the steps to solve this?" often leaves them feeling lost, delaying their progress and undermining their confidence.

- **Expected Impact of SyntaxSavior:**
    - The tool can provide task-related guidance in the form of pointers or hints, helping students break down tasks into manageable steps without spoon-feeding them solutions.
    - Clearer understanding of tasks will encourage independent thinking and foster problem-solving skills.

3.5.1.3 Lack of Understanding: “How Do I Use This?”

Students often struggle with vague instructions like "use Math.round" or "implement this function," as they lack a foundational understanding of what it means to "use" a specific feature or concept. This points to a more general gap in understanding how to translate instructions into functional code.

- **Expected Impact of SyntaxSavior:**
    - The system will provide short, actionable explanations and context-sensitive examples for concepts like using methods or implementing a given requirement.
    - For example, when instructed to use Math.round, the tool can display:
        - A brief explanation of what Math.round does.

        - A simple, non-task-specific example of its usage.
        - Common pitfalls to avoid when using similar methods.

3.5.1.4 Navigating the Eclipse Interface

A frequent technical issue involves students accidentally closing essential UI elements, such as the project explorer, and then struggling to locate or restore them. Tutors regularly repeat instructions for these situations, taking up valuable time.

- **Expected Impact of SyntaxSavior:**
    - SyntaxSavior's plugin can offer interactive navigation hints, such as:
        - Pop-ups directing students on how to reopen closed panels like the project explorer.
        - Context-sensitive tips when students encounter common UI hurdles.

3.5.1.5 Overwhelming Error Messages

Java's error messages, particularly for beginners, can be verbose and intimidating. Misunderstanding these errors often leads to prolonged struggles, as students misinterpret or overlook the actual problem.

- **Expected Impact of SyntaxSavior:**
    - The tool will parse and simplify Java error messages, providing beginner-friendly explanations and actionable suggestions for resolution.
    - For instance, instead of displaying a raw "NullPointerException," SyntaxSavior can explain:
        - "A variable is being used without being initialized. Check if all variables have a value before accessing them."

3.5.1.6 Over-Reliance on Generative AI

Some students turn to generative AI tools like ChatGPT or external online resources for assistance. While these tools can be helpful, they often provide information or solutions that are either too advanced or out of context, leaving students more confused.

- **Expected Impact of SyntaxSavior:**

- By offering curated, task-specific guidance aligned with the curriculum, SyntaxSavior will reduce students' reliance on external, unmoderated tools.
- This ensures students stay focused on the intended learning outcomes without veering into unrelated or overly advanced topics.

3.5.1.7 Submission Issues

Common lab-specific struggles include difficulties uploading code to the LMS/VPL system, incorrectly formatted files, or unanticipated errors during submissions.

- **Expected Impact of SyntaxSavior:**
  - The tool can preemptively check for common submission issues, such as file naming conventions, correct folder structures, or unhandled edge cases.
  - Students will receive warnings and instructions to address these issues before submission, reducing stress and frustration.

3.5.2 Use case model

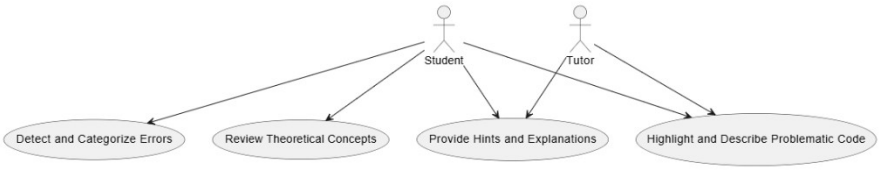

Figure 1 : Use Case Diagram

This use case diagram depicts interactions between students, assistants and the system of SyntaxSavior. Students rely on the system to discover and categorize coding faults, study theoretical ideas, obtain pointers and explanations, and identify faulty code with descriptions. Assistants utilize the approach to improve their assistance for students by emphasizing explanations and feedback.

3.5.3 Object and Class Model

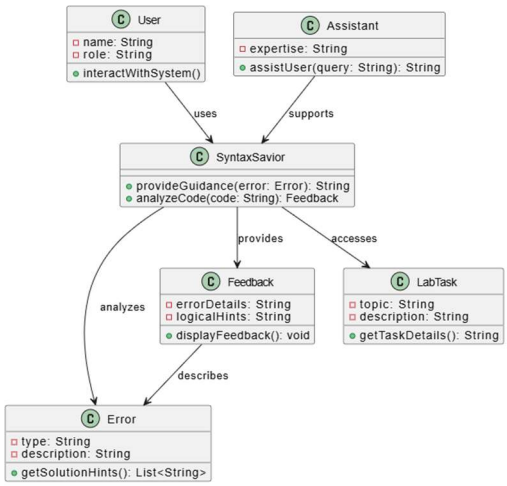

Figure 2 : Object and Class Diagram

This class diagram represents the SyntaxSavior system, where a User interacts with the system, and Assistant supports it by providing expertise. The central SyntaxSavior class provides guidance and analyze code, generating Feedback that includes errorDetails and logicalHints. Errors, describedby their type and description, are analyzed to generate solution hints and feedback. The system also accesses LabTask details to assist students in completing tasks. The relationships emphasize how the system processes errors, provides feedback, and integrates student and assistant roles for effective functionality. This diagram is entirely representative and created for demo purposes only.

3.5.4 Dynamic Models

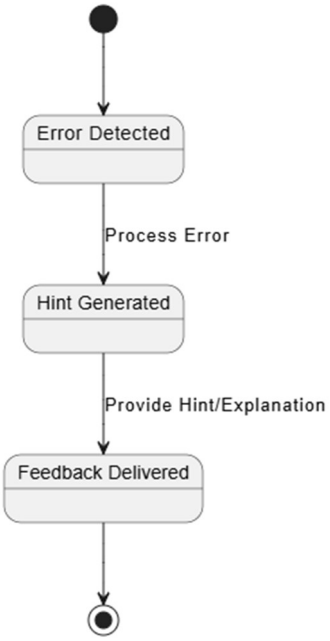

Figure 3 : State Diagram

The state diagram depicts the SyntaxSavior system's error handling procedure. It starts by identifying an issue in the code (issue Detected), then processes the error to generate appropriate hints (Hint Generated). The system subsequently gives the user specific explanations or feedback (Feedback Delivered), which completes the procedure. This offers a disciplined approach to assisting users with code challenges.

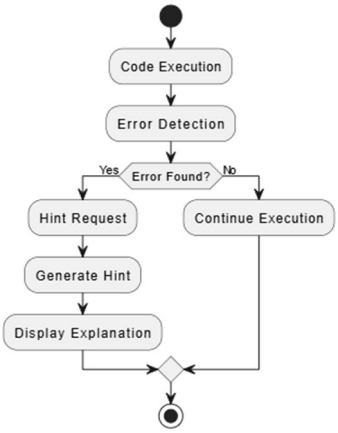

Figure 4 : Activity Diagram

The activity diagram depicts a method for code execution and error resolution. It begins with code execution, followed by error detection. If a mistake is discovered, the procedure proceeds to request a hint, generate the hint, and show an explanation to assist the user in correcting the issue. If no errors are identified, the execution will proceed uninterrupted. The procedure concludes when the error is resolved or the execution is successful. This procedure guarantees a user-friendly approach to debugging and mistake remediation.

3.5.5 User Interface - Navigational Paths and Screen Mock-ups

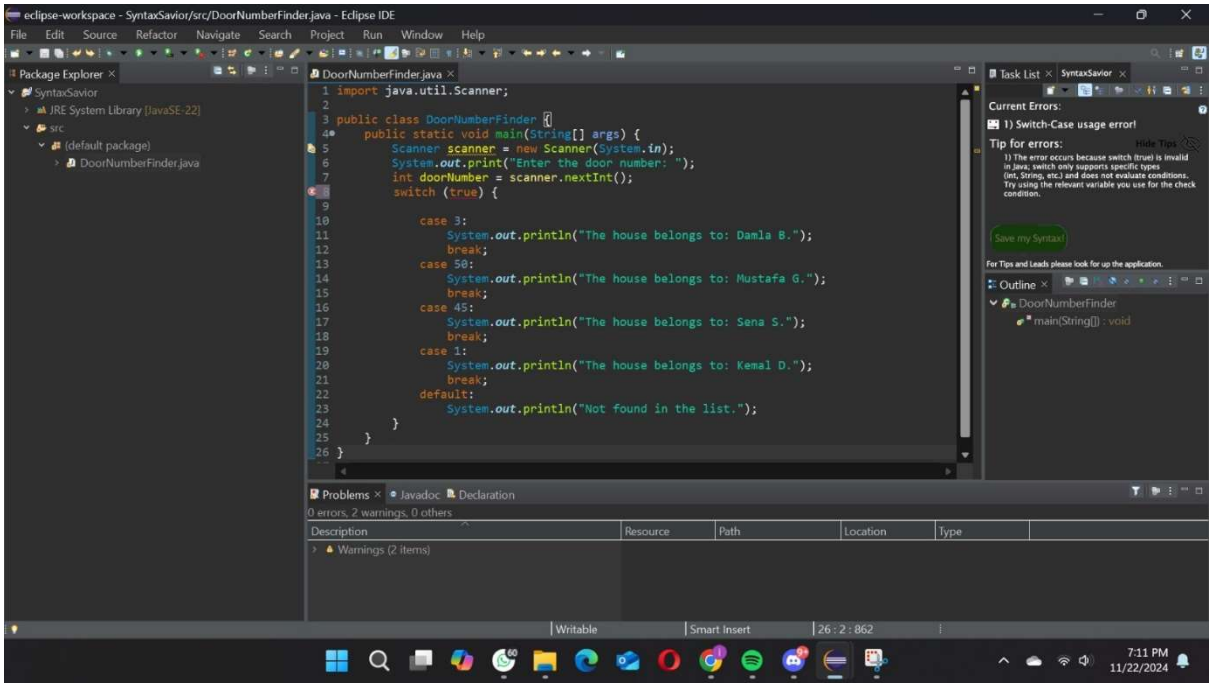

Figure 5: Mock Up of Plugin

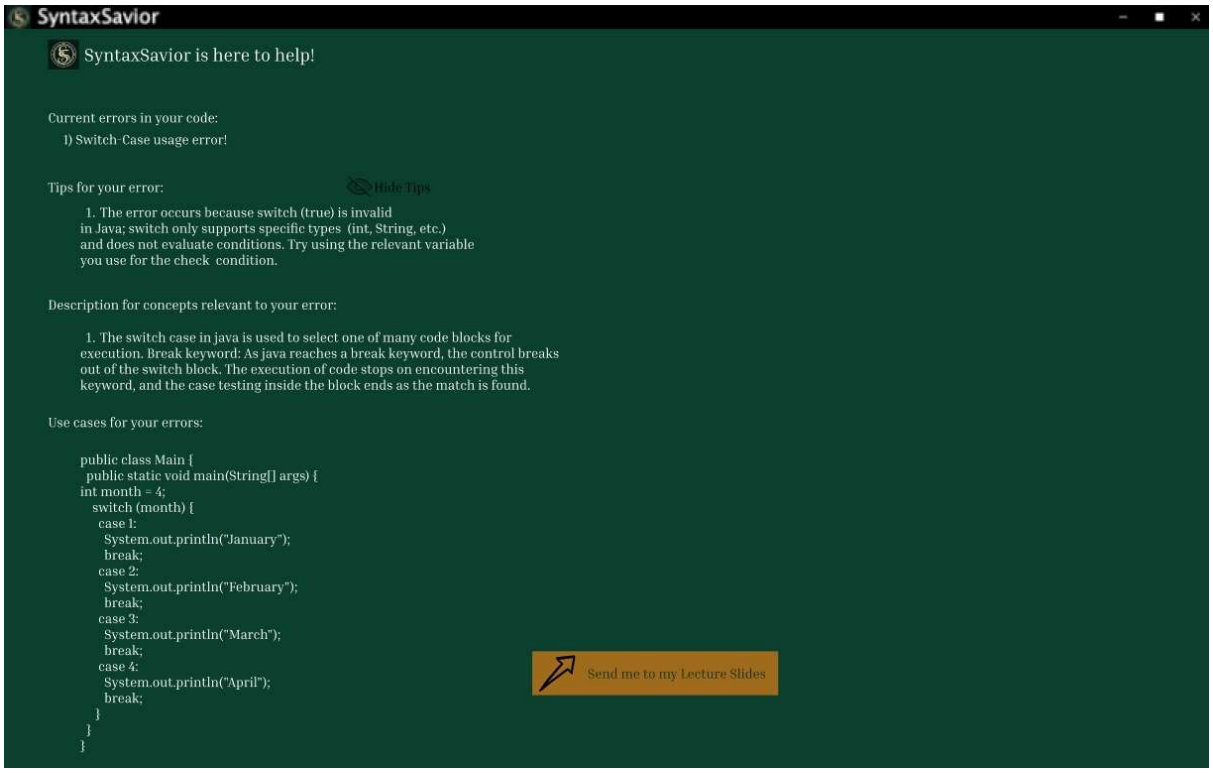

Figure 6: Mock-Up of Main Program

4 Glossary

- **Analytics for Tutors**
  Anonymized insights into common student mistakes, helping tutors identify areas needing more focus.
- **Automated Checks**
  System-generated checks that flag issues like incorrect file naming, missing files, or edge cases before submission.
- **Cross-Platform Compatibility**
  Ability of the system to work across multiple IDEs (e.g., Eclipse, IntelliJ IDEA).
- **Error Detection and Categorization**
  Real-time identification of syntax, runtime, and logical errors in student code.
- **Fine-Tuning**
  Adjusting system settings to optimize performance and prevent misuse.
- **Functional Requirements**
  Key system features such as real-time error detection, guidance, and lab-specific concept explanations.
- **Lab-Specific Concept Explanations**
  Contextual explanations of programming concepts relevant to the current task.
- **Non-Intrusive UI**
  A user-friendly interface integrated within Eclipse, providing helpful hints without cluttering the screen.
- **Nonfunctional Requirements**
  Criteria describing system performance, reliability, security, and scalability.
- **Pre-Submission Checks**
  Automated checks to catch common submission errors before final submission.
- **Privacy Compliance**
  Ensuring that student data is securely handled and compliant with data privacy regulations.
- **Real-Time Feedback**
  Instant feedback on student progress, offering guidance as they work through tasks.
- **Reliability**
  System's ability to consistently function during lab sessions without downtime.
- **Security**
  Protection against misuse, including prevention of cheating or unauthorized resource usage.
- **Scalability**
  System's ability to handle large numbers of students and users without performance loss.
- **Step-by-Step Guidance**
  Guidance focused on hints and suggestions rather than direct solutions to encourage independent learning.
- **Subject-Related Information**
  Task-specific explanations that provide context for concepts as students work.
- **Usability**
  Ease of use, especially for first-year students, with a simple and intuitive interface.

- **Version Control**
  Tracking of code changes to monitor progress and assist with debugging.
- **UI (User Interface)**
  The system's interface for interaction, integrated into Eclipse with minimal distractions.
- **Zero-Tolerance for Misuse**
  Prevention of cheating or misuse by detecting inappropriate behavior during tasks.

5 References

1. CMPE 113: Introduction to Programming Syllabus. TED University, [2024].
   Course syllabus for the CMPE 113 Introduction to Programming class, providing foundational information on course structure, objectives, and expectations.

2. CMPE 491: Senior Project 1 Syllabus. TED University, [2024].
   Course syllabus for the CMPE 491 Senior Project 1 class, outlining guidelines and expectations for senior-level project work, relevant to the development of the SyntaxSavior tool.

3. SyntaxSavior User Needs Form. TED University, [2024].
   Survey conducted with students to gather insights into their needs, preferences, and challenges in the introductory programming course CMPE 113, to inform the design and features of the SyntaxSavior tool.

4. Diagrams :
   https://drive.google.com/file/d/1oaY8Gwv7wXEXcHPpunhAtsIc9eleeRjn/view?usp=sharing