Software Requirements Specification for Software Engineering: Code Plagiarism Detector

Team 2, SyntaxSentinals
Mohammad Mohsin Khan
Lucas Chen
Dennis Fong
Julian Cecchini
Luigi Quattrociocchi

October 14, 2024

Contents

Revision History

Date	Version	Notes
October 11th, 2024	0.0	First iteration of complete document

1 Purpose of the Project

1.1 User Business

The Measure of Software Similarity algorithm, or MOSS algorithm for short, is the current standard for plagiarism detection of code. This algorithm works by comparing tokenized code snippets and assigning a similarity score without any weighting based on the complexity of the line being examined. In other words, there is an inherent lack of semantic understanding of the code being examined. This gives rise to a major flaw in the MOSS algorithm, which is that benign lines of code can be added to a program that does not improve or change functionality but still serves to create an illusion of difference in the eyes of the algorithm.

This project seeks to create a plagiarism algorithm that relies on NLP techniques of present to account for semantics and prevent primitive circumvention of plagiarism detection, such as the addition of benign lines or variable name changes. The users of our product will primarily be those concerned with fairness and integrity of code submissions within a competitive environment, such as professors or code competition holders.

Users are intended to use the resulting product of our project by giving it code snippets and receiving a plagiarism report in return. This report will contain a set of similarity scores for inputted code snippets, which when assessed against an outputted threshold will indicate likelihood of plagiarsm having taken place. This will benefit the users by allowing them to more accurately assess the presence of plagiarized work, creating a fairer environment for competition and rewarding coders correctly. Ultimately, the project aims to help users achieve an environment that cycles merit instead of cheating, which is believed to be a primary interest of users too.

1.2 Goals of the Project

Goal	**Explanation**	**Reason**

Ease of Use	Detector has an intuitive way to insert data and obtain results	This application is expected to be used as a secondary tool for teachers/professors when administering assignments. It should not require in-depth learning or it will be too inconvenient as an assistant tool for detecting plagiarism. (Measured by actions to complete analysis)
Clarity of Output	Detector explains how to interpret outputs clearly, leaving no ambiguity in whether plagiarism is suspected	If the user does not comprehend the output, it may result in unjust accusations or undetected plagiarism. (Measured by lines of description or number of users who correctly interpret output)
Real-Time Processing	The detector computes results on a dataset of code snippets quickly, enabling professors to incorporate them into evaluations	Since multiple assignments are administered over several weeks, the detector must be fast enough to be realistic for daily use. (Measured by execution time)
False Positive Accuracy	The detector prioritizes minimizing false positives over false negatives	In this case, a false positive could cause harm to an innocent student, while a false negative allows a violation to go unnoticed. The focus is on protecting innocent students. (Measured by false positives and negatives using recall, precision, etc.)

Ethically Sourced Data	The detector uses only data that openly discloses its origins and all data used for the detector is stated for all to see.	In modern ML development, it has become a hot topic for how models get their data. This is because many modern models have used datasets that contain information that was taken from individuals without consent (such as art). It is important to our team to make it clear this is not precedent and that individuals should have clear consent to being used for training models. (Measured by having accreditation for datasets involved in training)

Table 2: Main Goals for the Plagiarism Detector

Stretch Goal	Explanation	Reason
Online Learning	Provide the ability for users to train models on their own datasets.	Without a sufficiently large amount of data to train on (more than will be seen over 8 months), the model will be biased to a degree and not as widely applicable to different sets of code. If the detector is given the ability to be trained by the user, they can better customize it for their own needs (measured by whether or not the user can conduct training)
Language Agnostic	The detector can analyze code from a multitude of languages	Having a detector that can draw patterns across different languages will make it adoptable by a wider set of professors who may conduct courses in less popular languages that our detector may have not dealt with at all before. If the detector is restrained to languages such as Python or Java, we will alienate some of our primary stakeholders.

Table 3: Strech Goals for the Plagiarism Detector

2 Stakeholders

2.1 Client

The primary clients for this project are the computing and software departments of academic institutions and code competition administrators. These clients seek an advanced plagiarism detection system that overcomes the limitations of existing tools like MOSS. In academic settings, the system will help maintain academic integrity, while in code competitions, it will ensure fair play by preventing plagiarism among participants.

2.2 Customer

The primary customers for this project are professors, course instructors, and code competition organizers. Professors and instructors will use the system to evaluate student submissions to identify plagiarism, while competition organizers will ensure that all participants submit original code.

2.3 Other Stakeholders

Other stakeholders include:

- Students: Indirectly affected, as their work will be evaluated by this system, and it is important to ensure that students don't get falsely accused of plagiarism.
- Competition Participants: In coding competitions, the participants rely on the system to ensure the competition they participate in is fair.

2.4 Hands-On Users of the Project

The hands-on users are the professors, teaching assistants, and code competition organizers who will directly interact with the system. They will use it to upload code submissions, compare entries, and review plagiarism reports.

2.5 Personas

- Professor: Dr. Onjama Wembo - A computer science professor who frequently assigns coding tasks and reviews student submissions.
- Student: John Johnson - An honest computer science student who expects the system to verify their work as valid and unplagiarised.
- Competition Organizer: Sung Yuhee - A competition organizer who uses the system to ensure participants submit original work to ensure fair play in the competition.
- Competition Participant: Giorno Capio - A competition participant aiming for a top score, hoping the system does not misclassify their work with someone else who may have similar code.

2.6 Priorities Assigned to Users

- High Priority: Professors, competition organizers, and instructors - They rely on the system to evaluate submissions to ensure integrity
- Low Priority: Students and competition participants - They are not the direct users of the system but rely on it for correct evaluations

2.7 User Participation

User participation is essential for the development and testing of the system. Professors will provide feedback during development and testing to ensure the system meets their needs. Regular feedback will help improve the system's accuracy and reliability.

2.8 Maintenance Users and Service Technicians

If possible, system administrators and IT staff will maintain the system. They will troubleshoot issues to keep the system functional, as well as oversee updates to the system.

3 Mandated Constraints

3.1 Solution Constraints

Constraint	Rationale
Our product shall follow a zero data retention policy.	We should comply with privacy laws regarding storage of sensitive information.

Table 4: Solution Constraints

3.2 Implementation Environment of the Current System

Constraint	Rationale
The application code should run on both Windows and Linux.	Users of our product who wish to host their own instance will be using a variety of server hardware providers, including Microsoft Azure and Amazon AWS.
The user interface should be accessible via a web browser on a computer with an internet connection.	Users of our product will access the tool on their own personal computers, which are not powerful enough to run the tool offline.

Table 5: Implementation Environment Constraints

3.3 Partner or Collaborative Applications

Our project will use Google Colab to train and tune our language models for the NLP component of our tool.

Constraint	**Rationale**
Usage of the Google Colab platform for training shall not exceed 100 compute units.	We will only purchase 100 compute units to stay within our project budget.
The training and tuning of our language models will be written in Python.	Google Colab only supports Python.
The number of training iterations of our language model will not exceed 1000.	The amount of computational power provided by Google Colab is the limiting factor in our training speed.

Table 6: Partner or Collaborative Applications Constraints

3.4 Off-the-Shelf Software

Constraint	**Rationale**
Any third-party code used in the project must be licensed under an open-source license that is compatible with the GPLv3 License.	Our project uses the GPLv3 license.

Table 7: Off-the-Shelf Software Constraints

3.5 Anticipated Workplace Environment

There are no constraints imposed upon us by the anticipated workplace environment.

3.6 Schedule Constraints

Constraint	Rationale
Training data for language models shall consist of pre-labelled datasets on the internet, or scraped from online programming contest results and labelled with an existing plagiarism detection tool.	Given the schedule of the project, there is not enough time to source and label datasets by hand.

Table 8: Schedule Constraints

3.7 Budget Constraints

Constraint	Rationale
The cost of sourcing training data and training NLP models using cloud computers shall not exceed CAD $150.	We are students and we don't have a lot of money.

Table 9: Budget Constraints

3.8 Enterprise Constraints

There are no enterprise constraints imposed upon our project.

4 Naming Conventions and Terminology

4.1 Glossary of All Terms, Including Acronyms, Used by Stakeholders involved in the Project

- MOSS - Measure of Software Similarity, commonly used code plagiarism detection algorithm
- NLP - Natural Language Processing, a type of model/algorithm that turns human-readable text into machine-readable text
- GPLv3 Licence - GNU General Public Licence v3.0
- API - Application programming interface
- AST - Abstract Syntax Tree, low-level representation of code execution
- AI - Artificial Intelligence
- UI - User Interface, the interface the user is met with
- PIPEDA - Personal Information Protection and Electronic Documents Act
- ZDR - Zero Data Retention

5 Relevant Facts And Assumptions

5.1 Relevant Facts

- The current standard for code plagiarism detection, MOSS, primarily relies on token matching and syntax-based comparison. This method cannot detect deeper semantic similarities in code.
- NLP techniques have advanced significantly in recent years, enabling more accurate natural language understanding. These techniques can be adapted to understand the structure and semantics of code, which could enhance plagiarism detection systems.

- There is a growing need for a plagiarism detection system that accounts for sophisticated plagiarism techniques, such as variable renaming, code restructuring, and adding non-functional code.
- Academic institutions are increasingly concerned with the fairness and accuracy of plagiarism detection systems to avoid penalizing students unfairly, especially with the rising prevalence of online and remote learning.

5.2 Business Rules

- The system must ensure compliance with data protection regulations by following a zero data retention policy.
- The similarity threshold for flagging plagiarism should be customizable by the institution or professor, allowing flexibility based on course policies.
- False positives (e.g., common code patterns) should be minimized, with options for professors to override flagged instances and manually validate the results.
- The system must be scalable to accommodate large datasets and multiple users submitting code for comparison at the same time.

5.3 Assumptions

- It is assumed that the academic institutions adopting this system have clear plagiarism policies and can provide a threshold score that reflects their definitions of plagiarism.
- It is assumed that the code samples provided for comparison are original and not previously processed by other plagiarism detection systems, ensuring that the results reflect real-time analysis.
- It is assumed that professors and administrators will review flagged cases manually to confirm plagiarism before taking disciplinary action.
- It is assumed that students will not have access to the internal workings of the plagiarism detection algorithm, preventing them from finding potential loopholes to bypass detection.

- Software will be used only in Canada, and the legal and ethical considerations of this country will be taken into account during development.

6 The Scope of the Work

6.1 The Current Situation

The current code plagiarism detection tools such as MOSS rely on tokenization and syntax-level comparisons. Although effective in detecting direct copies or slight variations, they struggle when faced with techniques such as adding redundant code which allows the user to completely bypass detection while still plagiarizing the underlying logic and structure of the code.

Additionally, MOSS does not take into account the complexity or intent behind the code, leading to issues such as false positives for common programming patterns. This creates a gap for more advanced tools capable of understanding the semantic meaning of code to more accurately detect plagiarism.

6.2 The Context of the Work

Our project aims to address these gaps by incorporating Natural Language Processing (NLP) and machine learning techniques which will be leveraged to improve the accuracy of detecting copied code. The context of the work is within academic institutions, where the integrity of student work is paramount and our tool will be used by professors to ensure a fair grading process while also supporting students in understanding the ethical use of code.

6.3 Work Partitioning

- **Research and Design**: Research current plagiarism detection systems and state-of-the-art NLP techniques applicable to code plagiarism.
- **Data Collection**: Gather a dataset of code snippets, including both plagiarized and original works, to train and test the model.
- **Model Development**: Develop the NLP-based model capable of understanding the semantic meaning of code. This may involve exploring

techniques like abstract syntax trees (ASTs), vector embeddings, or other representations of code that retain semantic meaning.

- **System Integration**: Build the system to take code as input, run through the developed model, and output a similarity score with appropriate thresholds.
- **Testing and Validation**: Test the system with various code samples to validate its performance and accuracy compared to traditional systems like MOSS. This will also test whether our method produces any false positives.
- **Documentation and Deployment**: Document the system architecture, the model, and the results. Deploy the system for use within academic settings.

6.4 Specifying a Business Use Case (BUC)

Business Use Case: Automated Code Plagiarism Detection for Academic Institutions

- **Actors:** Professors, Students, System Administrators
- **Trigger:** A professor or system administrator uploads multiple code submissions for plagiarism detection in a course assignment.

Main Success Scenario

1. user logs into the system using their credentials.
2. user uploads code submissions for plagiarism detection.
3. The system ingests the uploaded code submissions.
4. The system processes each code snippet using the NLP model to generate semantic representations of the code.
5. The system compares the representations to detect plagiarism, taking into account code similarity beyond syntax or token matching.
6. The system outputs a similarity score for each comparison, with thresholds indicating whether plagiarism is suspected.

7. The professor reviews the similarity scores and flags any suspicious cases for further investigation.

8. The system generates a report summarizing the findings for the professor's review.

Extensions

- If the system detects false positives (common programming patterns being flagged as plagiarism), the professor can override the result.
- If new sophisticated plagiarism techniques are detected, the system can update its learning algorithms to improve accuracy over time.

7 Business Data Model and Data Dictionary

7.1 Business Data Model

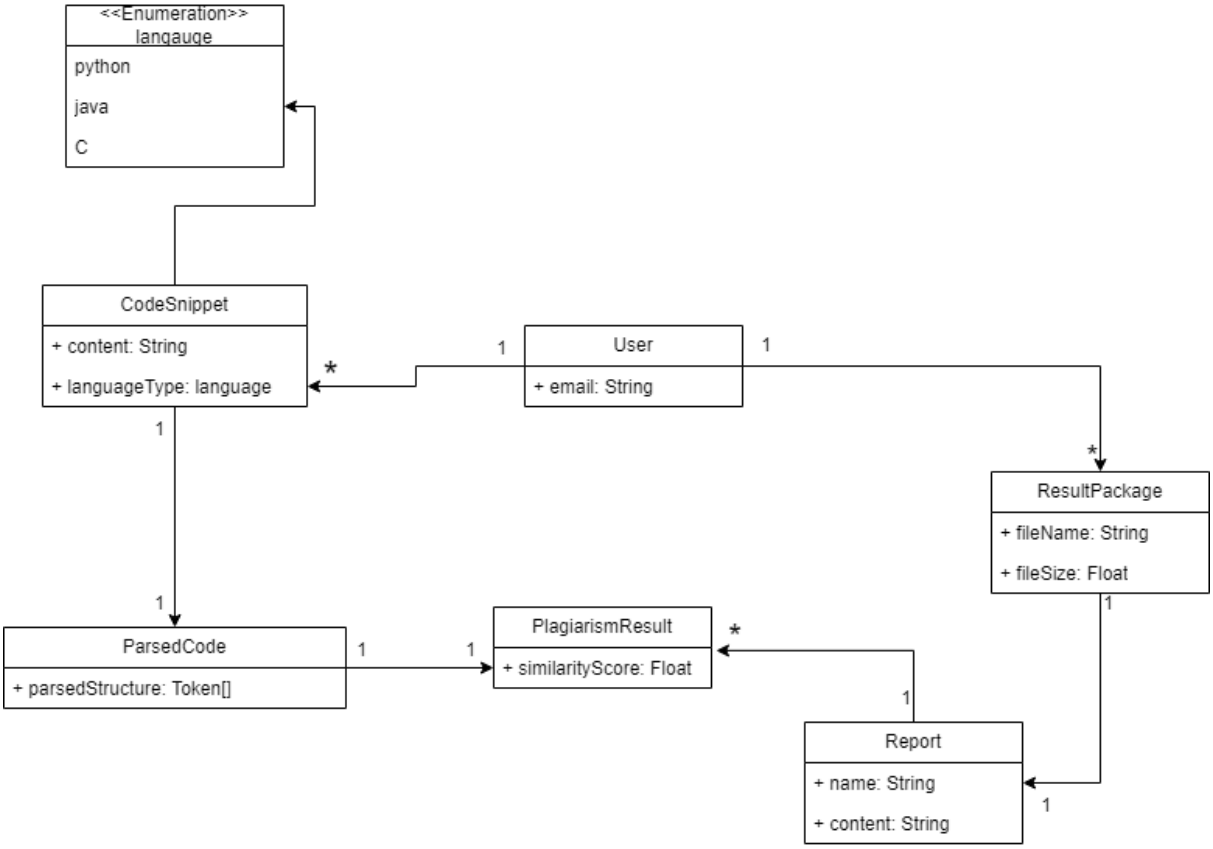

Figure 1: Business Data Model

7.2 Data Dictionary

Enums
language: Represents the programming language used in a code snippet.

- **Values**:
  - `Python`
  - `Java`
  - `C`

Classes
CodeSnippet: Represents the code snippet submitted by a user.

- **Attributes**:
  - `content: String` – The actual content of the code.
  - `languageType: language` – The programming language used, selected from the `language` enum.
  - `submissionTime: String` – The timestamp when the code was submitted.
- **Relationships**:
  - A `User` can submit multiple `CodeSnippet` objects (1-to-many relationship).
  - Each `CodeSnippet` generates exactly one `ParsedCode` (1-to-1 relationship).

ParsedCode: Represents the parsed structure of a code snippet for analysis.

- **Attributes**:
  - `parsedStructure: Token[]` – An array of tokens that represents the parsed structure of the code. Note: the actual structure of tokens hasn't been determined yet.
- **Relationships**:

- Each `CodeSnippet` is associated with exactly one `ParsedCode`.
- Each `ParsedCode` generates exactly one `PlagiarismResult` (1-to-1 relationship).

PlagiarismResult: Represents the outcome of the plagiarism detection process.

- **Attributes**:
  - `timeStamp: String` – The timestamp of when the plagiarism result was generated.
  - `similarityScore: Float` – The similarity score between the code snippet and other submissions.
- **Relationships**:
  - A `ParsedCode` generates exactly one `PlagiarismResult` (1-to-1 relationship).
  - A `PlagiarismResult` can be part of one `ResultPackage` (many-to-one relationship).

ResultPackage: Represents a zipped file that contains one or more plagiarism results, which can be sent via email.

- **Attributes**:
  - `fileName: String` – The name of the result package file.
  - `fileSize: Float` – The size of the result package file.
  - `timeStamp: String` – The timestamp of when the result package was generated.
- **Relationships**:
  - A `User` can receive multiple `ResultPackages` (1-to-many relationship).
  - Each `ResultPackage` can include multiple `PlagiarismResult` objects.

User: Represents a user of the system, such as a professor or student.

- **Attributes**:
  - `email: String` – The email address of the user.
- **Relationships**:
  - A `User` can submit multiple `CodeSnippet` objects (1-to-many relationship).
  - A `User` can receive multiple `ResultPackages` (1-to-many relationship).

Relationships Overview

- `User` submits multiple `CodeSnippets` (1-to-many).
- `CodeSnippet` is parsed into exactly one `ParsedCode` (1-to-1).
- `ParsedCode` generates exactly one `PlagiarismResult` (1-to-1).
- `PlagiarismResult` can be part of one `ResultPackage` (many-to-one).
- `User` receives multiple `ResultPackages` (1-to-many).

8 The Scope of the Product

8.1 Product Boundary

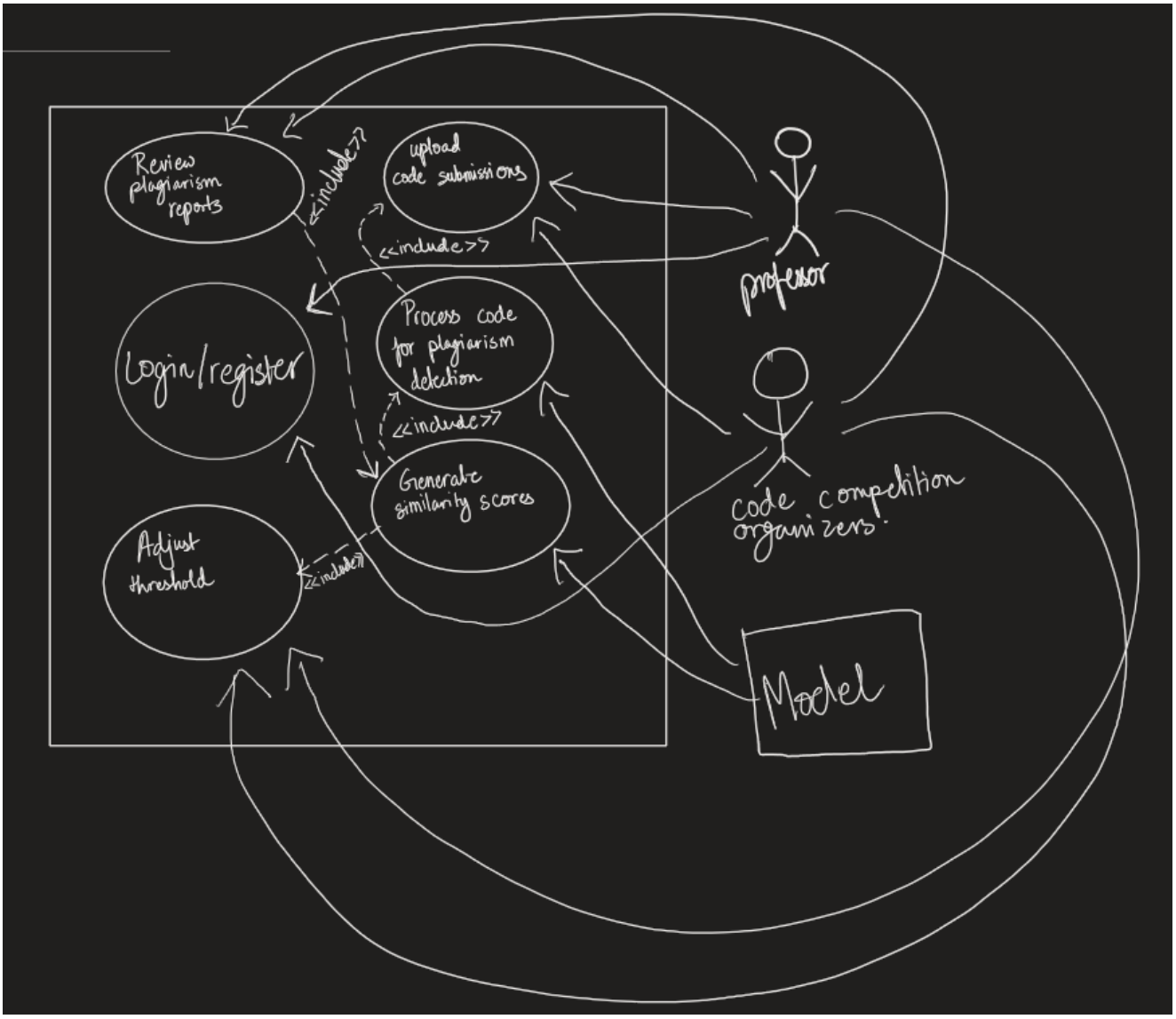

Figure 2: Use Case Diagram
Note: The Model here represents our NLP model, which is the core of our product.

8.2 Product Use Case Table

PUC No	PUC Name	Actor/s	Input/Output
1	Upload Code Submissions	Professor, Competition Organizers	Code Submission (in) / Uploaded Code (out)
2	Process Code for Plagiarism Detection	Model	Uploaded Code (in) / Processed Code (out)
3	Generate Similarity Scores	Model	Processed Code (in) / Similarity Scores (out)
4	Review Plagiarism Reports	Professor, Competition Organizers	Similarity Scores (in) / Plagiarism Report (out)
5	Adjust Plagiarism Detection Threshold	Professor, Competition Organizers	Threshold Input (in) / Updated Threshold (out)
6	Login/Register	Professor, Competition Organizers	Login Credentials (in) / Access to System (out)

Table 10: Product Use Case Table

8.3 Individual Product Use Cases (PUC's)

PUC 1: Upload Code Submissions
Actors: Professor, Competition Organizers
Preconditions: The user is logged in and has the necessary permissions to upload submissions.
Main Scenario:

1. The system prompts the user to upload code files.
2. The user uploads the required code files and submits them.
3. The system stores the submitted files for plagiarism detection processing.

4. The system notifies the user that the upload was successful.

Postconditions: The code submissions are available for plagiarism detection.

PUC 2: Process Code for Plagiarism Detection
Actors: Model, Professor
Preconditions: Code submissions have been uploaded and are ready for processing.
Main Scenario:

1. The system retrieves the submitted code for processing.
2. The system parses the code to generate structured representations (e.g., Abstract Syntax Tree or AST, still TBD for our use).
3. The system compares the parsed code against other uploaded submissions and datasets using the plagiarism detection model.
4. The system stores the processed results for similarity scoring.

Postconditions: The code is fully processed and ready for generating similarity scores.

PUC 3: Generate Similarity Scores
Actors: Model
Preconditions: The code has been processed and structured.
Main Scenario:

1. The system calculates similarity scores by comparing the structured code.
2. The system generates detailed similarity reports with scores for each code submission.
3. The system emails the similarity reports in a .zip file to the user.

Postconditions: Similarity scores are available for review.

PUC 4: Review Plagiarism Reports
Actors: Professor, Competition Organizers
Preconditions: Similarity scores have been generated and are ready for review.
Main Scenario:

1. The professor/organizer logs into the system and uploads the .zip file containing the plagiarism reports.
2. The system displays the plagiarism reports, including similarity scores and flagged submissions.
3. The professor/organizer reviews the results and flags any cases for investigation or further action.
4. The system logs any flags or notes made by the professor/organizer.

Postconditions: The professor/organizer has reviewed the plagiarism reports.

PUC 5: Adjust Plagiarism Detection Threshold
Actors: Professor, Competition Organizers
Preconditions: The plagiarism detection system has been configured with default thresholds.
Main Scenario:

1. The professor/organizer navigates to the plagiarism detection settings.
2. The system displays the current thresholds for flagging plagiarism.
3. The professor/organizer adjusts the thresholds based on the desired sensitivity level.
4. The system applies the new threshold settings to future plagiarism checks.

Postconditions: The new threshold settings are saved and used for subsequent plagiarism checks.

PUC 6: Login/Register
Actors: Professor, Competition Organizers

Preconditions: The user has an account or needs to register.
Main Scenario:

1. The user navigates to the login or registration page.
2. The system prompts the user to enter their credentials (or create a new account).
3. The user provides the necessary information (email, password, etc.) for login or registration.
4. The system verifies the credentials or creates a new account.
5. The system logs the user in or confirms successful registration.

Postconditions: The user is logged into the system and can access the functionalities based on their role.

9 Functional Requirements

9.1 Functional Requirements

- The system shall take in code snippets as inputs
- The system shall output similarity scores alongside thresholds that indicate whether or not plagiarism is present within the input
- The system shall be able to output documentation explaining how to interpret any other output (guide document)
- The system shall allow code snippets to be flagged after similarity scores and thresholds are generated
- The system shall provide a set of similarity scores for all code snippet pairings
- The system shall allow a report to be created after scores and thresholds are generated that summarize all system outputs
- The system will allow users to create an account with an email and password

- The system will allow users to sign into a created account with their email and password
- The system shall be able to email all outputs as a .zip file to an email selected by the user
- System shall be able to visualize a .zip file containing its outputs uploaded by user the on the user interface

10 Look and Feel Requirements

10.1 Appearance Requirements

The user interface (UI) must adhere to the following appearance guidelines:

- **Consistency**: The UI should maintain a uniform color palette, font, and layout across all screens and elements. For instance, the primary color is defined as blue (#0047AB) for buttons, links, and headers, and white (#FFFFFF) for background areas.
- **Clarity**: All icons, buttons, and menus should be intuitive and clearly identifiable. Hovering over icons will provide tooltips with a brief description of their functionality.
- **Responsiveness**: The layout should adjust according to screen size, ensuring the interface remains usable on a range of devices, including desktops, tablets, and mobile phones.

10.2 Style Requirements

The style guidelines for the interface are as follows:

- **Typography**: The font family used across the UI will be *Roboto*. Headers should use a 24px font size, body text should be 14px and button text should be 16px.
- **Color Scheme**: Buttons and interactive elements should use the primary color #0047AB. Background areas will use #F0F0F0, and error messages will be highlighted in #FF0000.

- **Button Styles**: All buttons should have rounded corners with a radius of 5px. On hover, the button background will lighten by 20%.
- **Spacing and Padding**: There should be at least 10px of padding between elements and a margin of 20px around each section to maintain a clean layout.

11 Usability and Humanity Requirements

11.1 Ease of Use Requirements

The system should be intuitive and simple to use for the target audience.

- **Minimal Learning Curve**: Users should be able to complete tasks with minimal instruction or training, taking at most 5 minutes from start to finish. The interface must guide users intuitively through this process.
- **Task Efficiency**: Common tasks should be achievable in no more than three clicks or interactions from the main screen.
- **Clear Navigation**: All navigation elements should be labeled clearly and positioned consistently across different pages to avoid confusion.

11.2 Personalization and Internationalization Requirements

The system will support a customizable experience in the future, but users will not be able to modify key settings such as themes or layout preferences at launch.

- **Future Customization Support**: Future updates will allow users to modify settings such as themes and layouts to suit their personal preferences.
- **English Language Support**: The system will operate exclusively in English, with all dates, currency, and numeric formats adhering to English (US) standards.

11.3 Learning Requirements

The system should provide clear documentation and onboarding materials.

- **Onboarding**: A guided onboarding process should be available for new users, helping them understand the main features of the system in under 5 minutes.
- **Help Documentation**: Detailed help documentation and tooltips should be available for key features to reduce the need for external assistance.

11.4 Understandability and Politeness Requirements

The system should use clear language and maintain a polite tone in all interactions with users.

- **Clear Language**: All messages and labels should be in simple, everyday language to ensure clarity for users of different levels of technical expertise.
- **Politeness**: Error messages and prompts should be worded politely and offer constructive guidance. Example error messages include:
  - *"Oops! Something went wrong. Please try again or contact support if the issue persists."*
  - *"We're sorry, but the file you uploaded is not supported. Please upload a .py file."*

11.5 Accessibility Requirements

Due to time constraints, the system will not be fully accessible at launch. But this can be a future goal.

12 Performance Requirements

12.1 Speed and Latency Requirements

- The system should process the inputs and provide results in under ten minutes

- The system must notify users if processing surpasses ten minutes

12.2 Safety-Critical Requirements

- The system must avoid false positives to protect students from wrongful accusations.
- A manual override option for professors must exist to prevent automatic accusations solely based on the model.

12.3 Precision or Accuracy Requirements

- The system should have an accuracy rate of at least 90% for identifying plagiarized content.
- The false positive rate (incorrectly flagged plagiarism cases) must be kept below 5%.

12.4 Robustness or Fault-Tolerance Requirements

- The system will not crash in the case of a malformed input, and will instead issue an error message to the user.

12.5 Capacity Requirements

- The system should be able to handle batches of inputs without significant delays.
- The system should be able to handle batches of inputs without the total processing time exceeding 600 seconds.

12.6 Scalability or Extensibility Requirements

- The system should be designed in a way such that adding support for new file types or coding languages should not impede current functionality

12.7 Longevity Requirements

- Backwards compatibility must be maintained when new features are introduced
- The system should be designed such that the natural language processing part can be changed in accordance with new research in the field.

13 Operational and Environmental Requirements

13.1 Expected Physical Environment

There are no specific physical environment requirements for our product since it is entirely software-based. Users can access the tool from any location with an internet connection via a web browser on any major operating system.

13.2 Wider Environment Requirements

There are no wider environment requirements for our product since it operates entirely within a virtual environment. Because of this, environmental factors are outside of the scope of our project.

13.3 Requirements for Interfacing with Adjacent Systems

- The system will provide the option to interface with cloud computing (AWS, Azure)
- The system will deploy the frontend and backend code in a free hosting Service
- The system will use external services for user authentication

13.4 Productization Requirements

For the tool to be ready for widespread use, it should meet several productization requirements:

- **User-friendly interfaces** for professors and code contest administrators, allowing easy input and clear, actionable outputs.
- **Documentation and training materials** to help users understand how to utilize the tool effectively, including explanations of how the similarity scores are calculated and how to interpret them.
- **Scalability**, ensuring that the tool can handle many code submissions simultaneously without performance degradation.

13.5 Release Requirements

To release the tool effectively, certain conditions must be met:

- **Beta testing** with a select group of professors and students to gather feedback on usability, performance, and false positive/negative rates.
- **Clear versioning** and change logs to track improvements over time, particularly as new programming languages or detection features are added.
- **Ongoing support**, including the ability to update the plagiarism detector with new models or features as they are developed.

14 Maintainability and Support Requirements

14.1 Maintenance Requirements

- **Versioning**: Keep track of and store model versions to allow switches between experimental models and previous versions. A verifiable metric is that version history exists.

  Justification: Gives the ability to work on separate model architectures to try improvements in different directions as well as rollback on model releases if something goes wrong as well as provide experimental releases.

- **Metric Reports**: Every model release shall be accompanied by a report of metrics decided upon to track model performance. A verifiable metric is that the report exists.

  Justification: Gives clarity on whether the model is being improved or not and what specific changes may be damaging the model. This will allow model health and performance to be tracked and maintained over time.

- **Issue Tracking**: Every bug should be registered in an issue-tracking system. A verifiable metric is bugs existing as issues.

  Justification: Brings attention to what bugs have been resolved and which ones still require work.

14.2 Supportability Requirements

- **Documentation**: provide concise (max 30 lines) comments on all APIs and algorithms used in the code base. This will allow others to comprehend what sections may need updates or bug fixes A verifiable metric is comment exists at every piece of API code or algorithm.

  Justification: leaves no obfuscation in where somebody should adjust code or update a component by clarifying what responsibility each area of code holds.

- **Logging** : provides a report of code components that executed leading up to a crash whenever a crash occurs, so bugs can be identified and fixes can be staged. A verifiable metric is a log with relevant information about the crash, such as a stack trace, is generated for the user to view when a crash occurs.

  Justification: Almost 100% necessary for effectively debugging issues in the code and giving the ability to support the model in the future for people who aren't familiar with the code base.

- **Community Acknowledgement**: provide a pathway for users to post or vote for requests/issues that should be addressed. A verifiable metric is the existence of said pathway, like email or GitHub issues or forum.

Justification: Will highlight to the community what are current pain points that should be prioritized and communication about how it is desired to address them.

14.3 Adaptability Requirements

- **Dataset Compatibility**: provide compatibility for at least two data formats when it comes to training datasets. A verifiable metric is that data the formats that are accepted.

  Justification: will allow the model to accommodate a wider amount of datasets and provides it better longevity in case either one of its data formats becomes less popular.

- **Template Adherence**: Every model layer/component follows a template. A verifiable metric is that a template exists for all model layers/components and is upheld by said layers/components.

  Justification - if somebody wishes to update the model with new layers/components or modify existing ones, there will be a guide to make the process straightforward and encourage those conducting the update to follow through.

- **Modularity**: every model layer/component can be treated as an individual function. A verifiable metric is that model layers/components can be fed an input and produce an output which reflects only operations conducted within the layer/component.

  Justification - enables removal or substitution of individual model layers/components which may be problematic or less efficient, similar to how one may remove or substitute a function in a program, without having to overhaul a larger/more significant portion of the model architecture.

15 Security Requirements

15.1 Access Requirements

- **Authentication**: The system shall require user authentication before allowing access to the tool. Only authenticated users can upload code

for comparison.

Justification - Users need to be identifiable so that they can receive their results via email and so that usage limits can be imposed upon them.

15.2 Integrity Requirements

Integrity requirements are outside the scope of this project. The system is designed to avoid storing any user data. Since no data is retained, data integrity is not a concern.

15.3 Privacy Requirements

- **Zero Data Retention**: The system shall retain no user-uploaded data beyond the immediate need of the task. Refer to section 17.2.5 for more information.

  Justification - This ensures user privacy and protects sensitive academic work from being mishandled.

- **In Transit Encryption**: All data (including code snippets) shall be encrypted while being uploaded to or downloaded from the system using secure encryption protocols.

  Justification - Protecting user data during transmission ensures that it can't be intercepted by unauthorized third parties.

15.4 Audit Requirements

- **In Transit Encryption**: The system shall log all user interactions and actions (such as authentication events, code uploads, and plagiarism checks) anonymously, without storing user-uploaded code or sensitive data.

  Justification - While no user data is retained, logging system events and user actions provide a transparent way to track system usage and ensure accountability without violating the zero data retention policy.

15.5 Immunity Requirements

Immunity requirements are beyond the scope of this project. The system should not need to defend against external attacks beyond the security measures already in place (e.g., authentication, encryption). The focus of the project is on code plagiarism detection, and immunity against malicious actors is not a primary concern.

16 Cultural Requirements

16.1 Cultural Requirements

No major cultural requirements are identified for this project but some that could be taken into consideration are:

Data Privacy and Ethical Use
Student Privacy: In some cultures and institutions, the handling of student work and data is highly regulated. Laws like the FIPPA mandate strict data privacy standards. The system should ensure that student data, including their code submissions, is securely handled, anonymized where possible, and not stored unnecessarily.

Differences in Academic Integrity Norms
Varying Definitions of Plagiarism: Some cultures and institutions promote collaboration and code borrowing so it is essential to define what plagiarism is in the context of this project. The tool should also be modifiable in its threshold for detecting plagiarism so institutions can change it to their needs.

17 Compliance Requirements

In developing the enhanced plagiarism detection tool, it is imperative to address various compliance requirements to ensure the tool operates legally, ethically, and in alignment with industry standards. These requirements encompass legal obligations related to data protection, intellectual property rights, and adherence to educational policies, as well as compliance with established software development and data security standards.

17.1 Legal Requirements

1. **Data Protection and Privacy Laws**: The tool will process sensitive information, including students' code submissions, which may be considered personal data under Canadian privacy laws such as the *Personal Information Protection and Electronic Documents Act* (PIPEDA) at the federal level, and Ontario's *Freedom of Information and Protection of Privacy Act* (FIPPA) for public institutions. Compliance with these laws require:

    - **Lawful Basis for Data Processing**: Ensuring that the collection and use of personal information is authorized under PIPEDA or FIPPA, typically requiring consent from students before processing their code or ensuring that processing is necessary for educational purposes.
    - **Data Minimization and Purpose Limitation**: Collecting only the data necessary for plagiarism detection and using it solely for that purpose.
    - **Transparency and Information Rights**: Informing students about how their data will be used, stored, and protected, and respecting their rights to access, correct, or withdraw their personal information.
    - **Security Measures**: Implementing appropriate technical and organizational measures to safeguard personal data against unauthorized access, loss, or disclosure, as required under PIPEDA and FIPPA.

2. **Intellectual Property Rights**: Under the *Copyright Act* of Canada, students typically hold the intellectual property rights to their original code. The tool must:

    - **Respect Ownership**: Use students' code exclusively for plagiarism detection without unauthorized distribution or reproduction.
    - **Establish Clear Terms**: Provide clear terms of service or agreements outlining how the code will be used, ensuring students are aware and consent to these terms.
    - **Avoid Infringement**: Ensure that any storage or processing of code does not violate the *Copyright Act* or institutional policies.

3. **Academic Integrity Policies**: The tool must align with the academic integrity and misconduct policies of Canadian educational institutions by:

    - **Supporting Fair Evaluation**: Assisting educators in identifying potential plagiarism accurately without bias.
    - **Due Process**: Ensuring that students have the opportunity to respond to plagiarism accusations, with results from the tool serving as part of a broader investigation rather than definitive proof.
    - **Confidentiality**: Maintaining the confidentiality of students' work and any findings related to plagiarism investigations.

17.2 Standards Compliance Requirements

1. **Software Development Standards**: Adherence to recognized software development practices is essential for ensuring quality and maintainability.

    - **SOLID Principles**: The project will follow SOLID principles—Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion—to promote clean, maintainable, and scalable code.
    - **Documentation and Testing**: Thorough documentation will be maintained throughout development, and rigorous testing will be conducted to validate the tool's performance and reliability.

2. **Data Security Standards**: Protecting sensitive data is a priority, and although full compliance with industry security standards such as ISO/IEC 27001 may not be feasible for a student capstone project, we will take measures to ensure data security.

    - **OWASP Guidelines**: Basic security measures will be implemented in line with the Open Web Application Security Project (OWASP) guidelines to mitigate common vulnerabilities such as injection attacks and unauthorized access.

3. **Accessibility Standards**: The tool will be designed to support basic accessibility features.

- **Screen Reader Support**: Key elements of the user interface will be made compatible with screen readers to assist visually impaired users.
- **Alt Text for Images**: All images and non-text content will include descriptive alt text to improve accessibility for users relying on assistive technologies.
- **Text Color Contrast**: The tool will ensure sufficient contrast between text and background colors to improve readability for users with visual impairments or color blindness.

4. **Ethical AI and Machine Learning Standards**: As the tool leverages AI technologies, it must adhere to ethical standards in AI development.
    - **Transparency and Explainability**: Ensuring that the AI models used are transparent in their operation and that their decision-making processes can be explained to users.
    - **Fairness and Non-Discrimination**: Preventing biases in the AI models that could unfairly target or disadvantage any group of students. Ensuring that names and other potentially discriminatory information are not used in the detection process.
5. **Data Handling and Retention Policies**: Establishing clear policies for how data is managed throughout its lifecycle.
    - **Zero Data Retention**: No user data will be retained beyond the immediate needs of the task. All user-uploaded data will be immediately loaded into memory and subsequently removed from disk as soon as the task is finished.
    - **Secure Disposal**: Implementing procedures for the secure deletion or anonymization of data that is no longer needed.
    - **Audit and Compliance**: Regularly auditing data handling practices to ensure compliance with all relevant laws and standards.

By meticulously addressing these legal and standards compliance requirements, the project not only safeguards the rights and interests of all stakeholders but also enhances the credibility and trustworthiness of the plagiarism detection tool. Ensuring compliance is fundamental to the tool's success and its acceptance by educational institutions, educators, and students alike.

18 Open Issues

18.1 System Design Uncertainty

- How the integration of the Natural Language Processing (NLP) model with the code submission system will work. Such as choosing what NLP model to use, how to train it, and where to host it.
- Selecting the most appropriate techniques for semantic code analysis, such as Abstract Syntax Trees (ASTs) or vector embeddings.
- Deciding on how the model will handle false positives, especially common coding patterns, to avoid misclassification: for instance, students using the same code snippets from a tutorial.

18.2 Data Collection Challenges

- While the project mentions scraping datasets from online programming contests and using pre-labeled datasets, there could be issues with data availability, quality, or ensuring that the data accurately reflects real-world usage.

18.3 Scalability Concerns

- Ensuring the system remains performant when processing large batches of submissions within the 10-minute processing time goal. Handling larger academic institutions with potentially thousands of submissions may challenge the current system's capacity requirements
- Given the limited budget for cloud compute units, it may be challenging to balance performance improvements with cost constraints. If training processes requires more resources than anticipated, costs could quickly escalate.

19 Off-the-Shelf Solutions

19.1 Ready-Made Products

Many ready-made tools aim to detect source-code level plagiarism, the most well-known of which being MOSS. MOSS is most commonly used by professors. There are also a handful of open-source alternatives, including: JPlag, SIM, Sherlock, and Plaggie. Most of them support checking multiple languages and use a variety of techniques to improve detection rates. These tools are relatively old and are not all actively developed.

19.2 Reusable Components

JPlag uses ANTLR 4 as a parser generator for many of its supported languages. For cross-language support, our tool can reuse JPlag's ANTLR grammar files to create language frontend parsers for each language we choose to support. Using a parser generator with pre-existing grammar files would reduce development time significantly since the alternative would entail writing a custom parser for each supported language. By using ANTLR and JPlag's grammar files, we could feasibly support many source languages as opposed to just one, which would most likely be Python (our product will be written in Python, and Python is capable of parsing its own syntax tree).

19.3 Products That Can Be Copied

The primary inspiration for our product is MOSS. Our product, similar to many others, will copy the general data pipeline of input source code. Specifically, after reading an input source code file, plagiarism checkers typically have parsing, tokenizing, and normalizing steps. This is followed by some analysis on the normalized text - MOSS uses "Winnowing", an algorithm that produces local fingerprints in a piece of text.
To reduce development time, we plan to copy the first step (the text preprocessing as described above) of the data pipeline implemented in MOSS. This kind of text normalization is commonly studied and there are many resources that explain implementation details.

20 New Problems

20.1 Effects on the Current Environment

- **Blind Trust in the Tool**: One potential issue is that users may develop a sense of over-reliance on the tool based solely on the fact that it is AI-based. This can lead to blind trust in its results. This could cause professors to forego manual checks, resulting in incorrect classifications.

20.2 Effects on the Installed Systems

There should be no new problems for users of this tool since they don't need to install any software on their local machines. The tool's interface is accessed by a web browser, and the processing happens on the server side.

20.3 Potential User Problems

- **Difficulty with the New UI**: Some users may struggle to navigate the new UI, resulting in misuse of the tool. Additionally, some settings may not be known by the user and left at their default value, which may cause confusion or result in a worse detection rate.

- **Misinterpretation of Results**: Users may incorrectly interpret the significance of the results provided by the tool. For instance, a high similarity score might be misinterpreted as definitive proof of plagiarism when there could be legitimate reasons for the similarity, such as the use of common libraries or design patterns.

20.4 Limitations in the Anticipated Implementation Environment That May Inhibit the New Product

- **Limited Compute Power Causing Delays**: If the system is deployed in environments with limited computational resources, such as local servers with insufficient processing power or memory, users may experience longer waiting times for results. Additionally, high computational demands may overload the system during peak usage times,

also resulting in longer waiting times.

20.5 Follow-Up Problems

- **Server Maintenance and Costs**: Long-term maintenance of the server infrastructure could present challenges, including server downtime, performance degradation, and increasing operational costs as more users adopt the system. Costs related to server hosting, cloud processing, and ensuring uptime can grow as demand increases.
- **Model Upgrades and Performance Regressions**: Updating or training new models may introduce performance regressions or unexpected behaviour in the plagiarism detection system. For example, a newer model may detect plagiarism more accurately but at the cost of higher processing times or an increase in false positives.

21 Tasks

21.1 Project Planning

1. **Hazard Analysis (October 12 - October 23)**
    - **High-Level Description:** This document identifies potential risks or hazards in the project, assesses their severity, and proposes mitigations. It is important to ensure safety and compliance with standards.
    - **Story Points:** 12 days
    - **Tasks:** Brainstorming hazards, analysis, and review.
2. **V&V Plan (Revision 0) (October 24 - November 1)**
    - **High-Level Description:** The Verification and Validation (V&V) plan outlines the strategy for ensuring that the system is both built correctly and performs as intended. It covers the methods and resources required for testing.
    - **Story Points:** 7 days
    - **Tasks:** Define tests, acceptance criteria, and procedures for verification and validation.

3. **Prep for Proof of Concept Demonstration (November 2 – November 11)**

    - **High-Level Description:** Preparation for a demonstration showing that the core concepts of the system are feasible. It typically involves implementing and testing essential features and preparing to showcase them to stakeholders.
    - **Story Points:** 10 day
    - **Tasks:** Development of key features and preparation for the demo.

4. **Proof of Concept Demonstration (TBD)**

    - **High-Level Description:** A demonstration showing that the core concepts of the system are feasible. This involves implementing and testing essential features and demonstrating them to stakeholders.
    - **Story Points:** 1 day
    - **Tasks:** Execution and presentation of the demo.

5. **Design Document (Revision 0) (November 23 - January 15)**

    - **High-Level Description:** This document outlines the architecture, components, data flow, and design decisions for the system. It serves as a blueprint for the development team.
    - **Story Points:** 7 days
    - **Tasks:** Detailed system design, architecture planning, and review.

6. **Prep for Revision 0 Demonstration (January 16 – February 3)**

    - **High-Level Description:** Preparation for the Revision 0 Demo, which demonstrates functionality that meets the initial design and requirements.
    - **Story Points:** 8 days
    - **Tasks:** Prepare, refine, and finalize core product features.

7. **Revision 0 Demonstration (Date: TBD)**

- **High-Level Description:** A demonstration of the system after the first iteration, showing functionality that meets the initial design and requirements.
- **Story Points:** 1 day
- **Tasks:** Present the system, and showcase its core features.

8. **V&V Report (Revision 0) (February 15 - March 7)**
    - **High-Level Description:** This documents the process of verifying that the system meets its functional and nonfunctional requirements and validating that the system performs as intended. It includes the evaluation of both functional and non-functional requirements, details of unit testing, automated testing, and code coverage metrics. The report also highlights any changes made due to testing and traces these changes back to both requirements and modules. Comparisons to existing implementations are provided, where applicable.
    - **Story Points:** 5 days
    - **Tasks:** Collect results, analyze tests, and create the report.
9. **Final Demonstration (Revision 1) (Date: TBD)**
    - **High-Level Description:** A final demonstration showcasing the complete system. By this stage, the system should be fully functional, meeting all requirements and passing validation.
    - **Story Points:** 10 days
    - **Tasks:** Ensure all components work seamlessly and prepare the final demo.
10. **EXPO Demonstration (April TBD)**
    - **High-Level Description:** This is a public or stakeholder-facing presentation of the system, potentially at an event or exhibition. The focus is on clear communication and showcasing the system's capabilities.
    - **Story Points:** 7 days
    - **Tasks:** Practice, set up, and run the demo smoothly.

11. **Final Documentation (Revision 1) (Due: April 2)**

    - **High-Level Description:** This is the comprehensive final documentation, which includes all deliverables like the problem statement, development plan, POC plan, requirements document, design, and more.
    - **Story Points:** 12 days
    - **Tasks:** Complete, revise, and polish all components of the documentation.

21.2 Planning of the Development Phases

Phase 1: Proof of Concept (Oct 12 - Nov 11)

- **Model Development:**
  - Story Points: 30 days
  - Concentrate on training and evaluating the plagiarism detection model. Experiment with various algorithms and techniques.
  - Weekly Iterations: Regularly assess model performance, making adjustments based on results. Document findings and refine methods to enhance the model's accuracy.

Phase 2: Initial System Design and UI Development (Nov 12 - Feb 3)

- **Model Development:**
  - Story Points: 30 days
  - Continue refining the model based on insights from the POC. Prepare the model for integration with the overall system architecture.
  - Mid-January: Finalize the model for production use.
- **UI Development:**

- Begin designing the user interface. Focus on mockups and user experience workflows based on expected model outputs.
- Weekly Check-Ins: Collaborate with the model development team to ensure UI designs align with the model's capabilities and outputs.
- **Tasks and Duration:**

Task	Estimated Days	Notes
Account Creation UI	2	Interface for user registration
Comparison UI	20	Displaying similarity results
Report Generation UI	10	Generating and downloading reports

Table 11: UI Development Tasks

- **Unit and Integration Testing:**
  - Story points: 5 days
  - Conduct unit and integration tests between the finalized model and the backend components. Ensure data flow and output processing are functioning correctly.

Phase 3: Final Completion (Feb 4 - Mar 24)

- **Final Model Tweaks:**
  - Make any last-minute adjustments to the model based on integration testing feedback.
- **Backend Development:**
  - Develop the backend infrastructure, including user account management and result delivery systems.
  - **Tasks and Duration:**

Task	Estimated Days	Notes
Account Management	1	Handling user registrations and logins
Similarity Processing	14	Processing model outputs for similarity scores
Report Generation	7	Generating reports based on model output
Report Delivery	4	Sending reports via email

Table 12: Backend Development Tasks

- **UI Finalization:**
  - Story points: 10 days
  - Finalize the UI based on integration testing results and feedback from the model. Ensure all user functionalities are ready for testing.
- **Final Testing and Debugging:**
  - Story points: 10 days
  - Conduct comprehensive testing of the entire system, focusing on functionality, performance, and user experience. Address any bugs and finalize documentation.

22 Migration to the New Product

22.1 Requirements for Migration to the New Product

The migration to the new code plagiarism detection system must be carefully planned and executed to ensure smooth adoption by academic institutions. The following requirements should be addressed:

- **User Training and Support**: Provide comprehensive documentation and training materials for professors, system administrators, and students to familiarize them with the new system. This includes tutorials on how to upload code, interpret results, and resolve flagged cases.

- **Phased Rollout**: Implement a phased migration plan, starting with pilot tests in a controlled environment (e.g., one course or department) before full-scale implementation across the institution.
- **Data Security Compliance**: Ensure data migrations comply with data protection regulations, such as Canada's Privacy Act.
- **System Downtime Minimization**: Plan the migration to minimize downtime and disruption to academic workflows. Ideally, the transition should occur during a break period, when student and faculty activity is low.

22.2 Data That Has to be Modified or Translated for the New System

For the migration to the new plagiarism detection system, certain data from the legacy systems must be modified or translated to ensure compatibility:

- **Code Submissions**: Legacy code submissions must be translated into a format that the new system can process, especially if the old system uses proprietary formats or different programming language encoding.
- **User Permissions**: Any user and administrator accounts need to be transferred to the new system. This includes roles, permissions, and access levels.
- **Configuration Data**: Settings from the old system, such as threshold scores, course configurations, and institution-specific policies, must be mapped and adjusted to fit the new system's configuration parameters.
- **Metadata and Logs**: Metadata (e.g., anonymous submission timestamps, course IDs) and system logs related to prior plagiarism checks should be preserved and transferred, ensuring transparency and continuity.

23 Costs

The costs associated with this project come from several different parts of the project.

- Data must be acquired before training the model. The data required to train the model can cost money. However, the team intends to automate processes to acquire data, and thus there is no charge incurred. An unknown amount of money will be needed for data if the team's method of acquiring data fails.
- In the training and testing phase, the model will require hardware to be trained on. The team intends to use Google Co-lab and leverage the hardware provided by their cloud platform. This will cost approximately 30 dollars for the required computation. However, if more training and testing is required, more Google compute units will be used, costing more money. An upper limit of 150 dollars is set, which is 5 times more than the current guess.
- The front end will need to be hosted somewhere. However, free alternatives exist, thus this will have no cost incurred.

In total, the project should only cost approximately 30 dollars. The cost is subject to change, and can increase/decrease depending on the amount of data needed, and how much Google's cloud hardware is used.

24 User Documentation and Training

24.1 User Documentation Requirements

The user documentation must be comprehensive, detailing all functionalities of the system. It should include step-by-step guides for professors, system administrators, and competition organizers, ensuring they can:

- Upload code submissions for plagiarism checks.
- Understand the process behind similarity score generation.
- Review and interpret similarity reports.
- Adjust plagiarism detection thresholds to suit specific needs.
- Administer the system, including managing users, system settings, and troubleshooting common issues.

The documentation should also include a frequently asked questions (FAQ) section, covering common user issues, and should be available in both written and video tutorial formats.

24.2 Training Requirements

The system requires training sessions for all primary users, including professors, competition organizers, and system administrators. Training should focus on:

- How to upload and submit code for plagiarism checking.
- How to review and interpret similarity scores and plagiarism reports.
- Techniques for flagging plagiarism cases and overriding false positives.
- Administrative training for system administrators, covering user management, and system maintenance.
- Ongoing support for users, including refresher training and updates on new features as the system evolves.

Training materials should be available in multiple formats: in-person workshops, webinars, and self-paced online courses.

25 Waiting Room

It is too early in the project to have a waiting room. This section will be completed as the project progresses and new features are considered.

26 Ideas for Solution

It is too early in the project to have ideas for solutions. This section will be completed as the project progresses.

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

1. What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project? Examples of possible knowledge to acquire include domain-specific knowledge from the domain of your application, or software engineering knowledge, mechatronics knowledge or computer science knowledge. Skills may be related to technology, or writing, or presentation, or team management, etc. You should look to identify at least one item for each team member.
   Knowledge Areas:

   - **Language normalization**: Understanding source-code level processing techniques necessary for transforming input text into a format usable by NLP models.
   - **NLP Techniques**: Understanding and applying Natural Language Processing (NLP) techniques to analyze code submissions. Necessary to develop algorithm/model project will rely on at its core.
   - **Server Deployment**: Understanding how to deploy algorithm/model in conjunction with UI to receive requests for analyzing plagiarism from a non-local user. Necessary to be able to host our model outside of a local host.

   Skills:

   - **Writing**: The ability to convey thoughts in a formalized manner. This is necessary to write documentation for projects such as SRS.
   - **Team Management**: The ability to delegate responsibilities amongst members in a respectful manner while working off feedback from teammates about their enjoyment of given tasks to retain a cohesive team dynamic. This is necessary to keep work on track and maintain an enthusiastic team spirit.

- **Proactive Communication**: Letting others know in advance the direction you want to head in for a given task. This is necessary to allow strong planning that will not crumble under roadblocks in code development, which are bound to happen.

Julian: Server Deployment, Proactive Communication
Lucas: NLP Techniques, Team Management
Dennis: NLP Techniques, Team Management
Mohammad: NLP Techniques, Writing, Proactive Communication

2. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

   - **Server Deployment**: YouTube Videos, Online Articles, Building miniproject
   - **Proactive Communication**: Scheduling Meetings for direction, Notebook Reminders
   - **Language normalization**: YouTube Videos, Research Papers, Open source code on GitHub
   - **NLP Techniques**: YouTube Videos, Online Articles, Research Papers
   - **Team Management**: YouTube Videos, Online Articles, GitHub Project Management Tools
   - **Writing**: Writing Documentation, Online Articles
   - **Proactive Communication**: Notebook Reminders, Scheduling Meetings

   Which team member chose to pursue which approach, and why?

   - **Lucas Chen:**
     - **NLP Techniques**: Lucas will pursue YouTube videos and online reading to understand NLP techniques. This is because he learns best through visual and auditory methods.

  - **Team Management**: Lucas will pursue learning more about GitHub project management tools and techniques. This is because he has taken on the role of project manager and needs to understand how to manage the team effectively.

- **Dennis Fong:**
  - **NLP Techniques**: Dennis will look for research papers on NLP, especially ones that pertain to the topic of this project. This is because he seeks a deeper understanding of the inner workings of a NLP model from both a math and code perspective.
  - **Team Management**: Dennis will learn about team management through the use of GitHub and the many features it supports for managing a project for the team. He is not as familiar with GitHub as his team members, and this will serve as a strong learning experience.

- **Julian Cecchini:**
  - **Server Deployment**: Julian will pursue online articles to learn this. He finds maintaining attention for long YouTube videos can be cumbersome and if he's interrupted, it's easier to pick up off online articles than YouTube videos. Also, he doesn't feel he has time to trial server deployment in a side project. He lacks experience in server deployment compared to the rest of the team, so he seeks to close the knowledge gap to be able to contribute and gain experience in this area.
  - **Proactive Communication**: Julian will schedule meetings for thought direction on given tasks. He feels the commitment imposed by a meeting where he must announce his thoughts will have him try more consistently to announce his direction than a notebook reminder that suggests he should bring some thoughts forward. He feels its essential he masters this while dealing with multiple extracurriculars on top of capstone so he doesn't lose touch with the team.

- **Mohammad Mohsin Khan:**
  - **NLP Techniques**: Mohammad will pursue online articles and research papers to understand NLP techniques. This is

because he prefers to read and analyze information at his own pace.

  - **Writing**: Mohammad will practice writing by creating documentation for the project. This hands-on approach will help him improve his writing skills through practical application.
  - **Proactive Communication**: Mohammad will use notebook reminders to prompt him to communicate proactively. This method aligns with his preference for written reminders and will help him stay organized and on track.

- **Luigi Quattrociocchi:**
  - **Language normalization**: Luigi will study open-source repositories on GitHub and read research papers relating to source-code-level language processing. This knowledge will be critical to transforming user input into a format that is usable for NLP analysis. This choice was made because Luigi has many ideas for the implementation of these techniques and he believes that formalizing this will help the project proceed.
  - **Server Deployment**: Luigi aims to improve his knowledge of server deployment and general web development best practices and techniques by watching YouTube videos and participating in discussions with the team about how we plan to deploy the system. Luigi finds that he lacks expertise in this area and would like to contribute to this aspect of the project.
  - **Writing**: Luigi will pursue improvement in writing by continuing to write documentation and enforcing strict linter rules about comments and function docstrings in the project source code. This will help him improve his communication with his team and improve documentation for any other maintainers of the project.