Software Requirements Specification for Software Engineering: Code Plagiarism Detector

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

October 8, 2024

Contents

Revision History

Date	Version	Notes
October 9	1.0	First iteration of complete document

1 Purpose of the Project

1.1 User Business

Insert your content here.

1.2 Goals of the Project

Insert your content here.

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 in the competition, who is hoping the system does not misclassify their work with someone elses 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

Insert your content here.

3.2 Implementation Environment of the Current System

Insert your content here.

3.3 Partner or Collaborative Applications

Insert your content here.

3.4 Off-the-Shelf Software

Insert your content here.

3.5 Anticipated Workplace Environment

Insert your content here.

3.6 Schedule Constraints

Insert your content here.

3.7 Budget Constraints

Insert your content here.

3.8 Enterprise Constraints

Insert your content here.

4 Naming Conventions and Terminology

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

Insert your content here.

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 lacks the ability to 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, such as GDPR, by securing student data, anonymizing it when possible, and minimizing unnecessary data retention.

- 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 any 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 dtection 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 for 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 aparmount 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**: Conduct research on 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. The system ingests the uploaded code submissions.
2. The system processes each code snippet using the NLP model to generate semantic representations of the code.
3. The system compares the representations to detect plagiarism, taking into account code similarity beyond syntax or token matching.
4. The system outputs a similarity score for each comparison, with thresholds indicating whether plagiarism is suspected.
5. The professor reviews the similarity scores and flags any suspicious cases for further investigation.
6. 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

Insert your content here.

7.2 Data Dictionary

Insert your content here.

8 The Scope of the Product

8.1 Product Boundary

Insert your content here.

8.2 Product Use Case Table

Insert your content here.

8.3 Individual Product Use Cases (PUC's)

Insert your content here.

9 Functional Requirements

9.1 Functional Requirements

- The system will take in code snippets as inputs
- The system must parse the input into a format that can be fed into a model for interpretation
- The system will leverage known natural language processing techniques to handle inputs
- The system will process the formatted input to provide a verdict on the presence of plagiarism as output

- The system must generate clear and concise outputs to indicate plagiarism
- The system's outputs should provide explanations on how to interpret output

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 one minute

- The system must notify users if processing surpasses one minute

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.
- Detected plagiarism must be recorded for future use such as appeals.
- Detected plagiarism must be stored in our database for future use such as appeals.

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 60 seconds.
- The system should be able to store a large amount of reports for later review.

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 with 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 to new research in the field.

13 Operational and Environmental Requirements

13.1 Expected Physical Environment

Insert your content here.

13.2 Wider Environment Requirements

Insert your content here.

13.3 Requirements for Interfacing with Adjacent Systems

Insert your content here.

13.4 Productization Requirements

Insert your content here.

13.5 Release Requirements

Insert your content here.

14 Maintainability and Support Requirements

14.1 Maintenance Requirements

Insert your content here.

14.2 Supportability Requirements

Insert your content here.

14.3 Adaptability Requirements

Insert your content here.

15 Security Requirements

15.1 Access Requirements

Insert your content here.

15.2 Integrity Requirements

Insert your content here.

15.3 Privacy Requirements

Insert your content here.

15.4 Audit Requirements

Insert your content here.

15.5 Immunity Requirements

Insert your content here.

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 instituitions promote collaboration as well as 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 instituitions can chnage 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 Protec-*

tion of Privacy Act (FIPPA) for public institutions. Compliance with these laws requires:

- **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.
    - **Secure Database Storage**: All data will be securely stored in a trusted cloud database service (e.g., AWS), utilizing built-in security features such as encryption at rest and in transit, to protect sensitive user information.
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.
    - **Retention Limits**: Defining how long code submissions and related data will be stored, in compliance with PIPEDA, FIPPA, and institutional policies.
    - **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 ongoing 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

Insert your content here.

19 Off-the-Shelf Solutions

19.1 Ready-Made Products

There are many ready-made tools that 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 of 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 it's 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 because "AI" cause professors to not manually check results

20.2 Effects on the Installed Systems

- Users might host their own instance of the server for data privacy reasons

20.3 Potential User Problems

- Not knowing how to use the new UI - Incorrect interpretation / poor understanding of result signifigance

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

- Must follow data privacy laws regarding data retention ???

20.5 Follow-Up Problems

- Server maintenance and costs - Upgrading or training new models might cause performance regressions

21 Tasks

21.1 Project Planning

Insert your content here.

21.2 Planning of the Development Phases

Insert your content here.

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 that all data migrations comply with data protection regulations, such as Canada's Privacy Act. Secure student data and ensure that no sensitive information is exposed during migration.
- **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.

- **Plagiarism Reports**: Historical plagiarism reports and similarity scores from the legacy system need to be reformatted to align with the structure of the new system's reporting. This includes recalculating similarity scores if necessary.
- **User Data and Permissions**: Any existing user data, including professor, student, and administrator accounts, needs 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., 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 acuiring 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 the how much Google's cloud hardware is used.

24 User Documentation and Training

24.1 User Documentation Requirements

Insert your content here.

24.2 Training Requirements

Insert your content here.

25 Waiting Room

Insert your content here.

26 Ideas for Solution

Insert your content here.

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.

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?