System Verification and Validation Plan for Software Engineering

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

October 30, 2024

Revision History

Date	Version	Notes
November 1	1.0	Initial documentation

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

Contents

1 Symbols, Abbreviations, and Acronyms **iv**

2 General Information **1**
- 2.1 Summary . . . 1
- 2.2 Objectives . . . 1
- 2.3 Challenge Level and Extras . . . 2
- 2.4 Relevant Documentation . . . 2

3 Plan **3**
- 3.1 Verification and Validation Team . . . 3
- 3.2 SRS Verification Plan . . . 3
- 3.3 Design Verification Plan . . . 3
- 3.4 Verification and Validation Plan Verification Plan . . . 3
- 3.5 Implementation Verification Plan . . . 4
- 3.6 Automated Testing and Verification Tools . . . 4
- 3.7 Software Validation Plan . . . 4

4 System Tests **5**
- 4.1 Tests for Functional Requirements . . . 5
  - 4.1.1 Area of Testing1 . . . 5
  - 4.1.2 Area of Testing2 . . . 6
- 4.2 Tests for Nonfunctional Requirements . . . 6
  - 4.2.1 Area of Testing1 . . . 7
  - 4.2.2 Area of Testing2 . . . 7
- 4.3 Traceability Between Test Cases and Requirements . . . 7

5 Unit Test Description **7**

6 Appendix **9**
- 6.1 Symbolic Parameters . . . 9
- 6.2 Usability Survey Questions? . . . 9

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test

[symbols, abbreviations, or acronyms — you can simply reference the SRS (Khan et al., 2024d) tables, if appropriate —SS]

[Remove this section if it isn't needed —SS]

This document outlines the strategies and processes used to ensure that the plagiarism detection system developed by the SyntaxSentinals team meets all functional and non-functional requirements. The primary goal of this plan is to build confidence in the correctness, usability, and performance of the system. It also focuses on identifying and mitigating potential risks, ensuring that the final product aligns with academic and competition standards. This document is organized as follows. The general information section provides an overview of the objectives, challenges, and relevant project documents used throughout the V&V process. The plan section describes the roles and responsibilities of the team members and the tools used for automated testing and verification. The system tests section lists the tests performed for both functional and non-functional requirements, with traceability to the SRS. The unit test description section details the unit testing scope, the modules tested, and the strategies for covering edge cases. Finally, the appendix contains symbolic parameters, survey questions (if applicable), and any other relevant information to support the V&V process. This plan will evolve as the project progresses, with updates following the completion of the detailed design and implementation phases.

2 General Information

2.1 Summary

The software being tested is a plagiarism detection system designed to identify similarities in Python code submissions, called SyntaxSentinels. This system utilizes natural language processing (NLP) techniques to analyze code semantics, preventing common circumvention methods like adding benign lines or altering variable names. Its core function is to allow users to input code snippets and receive a plagiarism report containing similarity scores, which, when compared to a threshold, indicate the likelihood of plagiarism. This tool is primarily intended for use in academic and competitive environments to promote fairness and integrity in code submissions.

2.2 Objectives

The primary objectives of this V&V plan are to:

- Build confidence in the programs correctness by ensuring alignment with the SRS requirements.
- Show that we have met the documented safety and security requirements (SR-SAF1- SR-SAF5) in the Hazard Analysis document.
- Demonstrate adequate usability in the program by conducting functional and non-functional tests mentioned in this document.

Out of Scope:

- Validation of any external libraries will be assumed to be handled by their maintainers.

2.3 Challenge Level and Extras

This project has been classified as having a General difficulty level, as agreed with the course instructor. Planned extras include:

- A user manual for instructors and administrators.
- Benchmarking of the tool's effectiveness compared to MOSS.

2.4 Relevant Documentation

The following documents are critical to the development and V&V efforts for this project.

- **Software Requirements Specification (SRS)**: Defines the project's requirements, guiding both verification and validation. (Khan et al., 2024d).
- **User Guide**: Provides operational instructions, relevant for usability testing. (Khan et al., 2024e).
- **Module Guide (MG)**: Outlines the system's architecture, essential for design verification. (Khan et al., 2024b).
- **Module Interface Specification (MIS)**: Details the internal modules and interfaces, critical for unit testing. (Khan et al., 2024c).
- **Hazard Analysis**: Identifies potential risks, guiding validation efforts for safety and security. (Khan et al., 2024a).

3 Plan

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

3.1 Verification and Validation Team

[Your teammates. Maybe your supervisor. You should do more than list names. You should say what each person's role is for the project's verification. A table is a good way to summarize this information. —SS]

3.2 SRS Verification Plan

[List any approaches you intend to use for SRS verification. This may include ad hoc feedback from reviewers, like your classmates (like your primary reviewer), or you may plan for something more rigorous/systematic. —SS]

[If you have a supervisor for the project, you shouldn't just say they will read over the SRS. You should explain your structured approach to the review. Will you have a meeting? What will you present? What questions will you ask? Will you give them instructions for a task-based inspection? Will you use your issue tracker? —SS]

[Maybe create an SRS checklist? —SS]

3.3 Design Verification Plan

[Plans for design verification —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

3.4 Verification and Validation Plan Verification Plan

[The verification and validation plan is an artifact that should also be verified. Techniques for this include review and mutation testing. —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

3.5 Implementation Verification Plan

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walkthroughs, code inspection, static analyzers, etc. —SS]

[The final class presentation in CAS 741 could be used as a code walkthrough. There is also a possibility of using the final presentation (in CAS741) for a partial usability survey. —SS]

3.6 Automated Testing and Verification Tools

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

[If you have already done this in the development plan, you can point to that document. —SS]

[The details of this section will likely evolve as you get closer to the implementation. —SS]

3.7 Software Validation Plan

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

[You might want to use review sessions with the stakeholder to check that the requirements document captures the right requirements. Maybe task based inspection? —SS]

[For those capstone teams with an external supervisor, the Rev 0 demo should be used as an opportunity to validate the requirements. You should plan on demonstrating your project to your supervisor shortly after the scheduled Rev 0 demo. The feedback from your supervisor will be very useful for improving your project. —SS]

[For teams without an external supervisor, user testing can serve the same purpose as a Rev 0 demo for the supervisor. —SS]

[This section might reference back to the SRS verification section. —SS]

4 System Tests

[There should be text between all headings, even if it is just a roadmap of the contents of the subsections. —SS]

4.1 Tests for Functional Requirements

[Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]

[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

4.1.1 Area of Testing1

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Title for Test

1. test-id1

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs. Output is not how you are going to return the results of the test. The output is the expected result. —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

2. test-id2

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

4.1.2 Area of Testing2

...

4.2 Tests for Nonfunctional Requirements

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. Not all projects will necessarily have nonfunctional requirements related to accuracy. —SS]

[For some nonfunctional tests, you won't be setting a target threshold for passing the test, but rather describing the experiment you will do to measure the quality for different inputs. For instance, you could measure speed versus the problem size. The output of the test isn't pass/fail, but rather a summary table or graph. —SS]

[Tests related to usability could include conducting a usability test and survey. The survey will be in the Appendix. —SS]

[Static tests, review, inspections, and walkthroughs, will not follow the format for the tests given below. —SS]

[If you introduce static tests in your plan, you need to provide details. How will they be done? In cases like code (or document) walkthroughs, who will be involved? Be specific. —SS]

4.2.1 Area of Testing1

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

4.2.2 Area of Testing2

...

4.3 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

5 Unit Test Description

This section will not be filled in until after the MIS document has been completed.

References

Mohammad Mohsin Khan, Lucas Chen, Dennis Fong, Julian Cecchini, and Luigi Quattrociocchi. Hazard analysis. https://github.com/lilweege/SyntaxSentinels/blob/main/docs/HazardAnalysis/HazardAnalysis.pdf, November 2024a.

Mohammad Mohsin Khan, Lucas Chen, Dennis Fong, Julian Cecchini, and Luigi Quattrociocchi. Module guide. https://github.com/lilweege/SyntaxSentinels/blob/main/docs/Design/SoftArchitecture/MG.pdf, November 2024b.

Mohammad Mohsin Khan, Lucas Chen, Dennis Fong, Julian Cecchini, and Luigi Quattrociocchi. Module interface system. https://github.com/lilweege/SyntaxSentinels/blob/main/docs/Design/SoftDetailedDes/MIS.pdf, November 2024c.

Mohammad Mohsin Khan, Lucas Chen, Dennis Fong, Julian Cecchini, and Luigi Quattrociocchi. System requirements specification. https://github.com/lilweege/SyntaxSentinels/blob/main/docs/SRS-Volere/SRS.pdf, November 2024d.

Mohammad Mohsin Khan, Lucas Chen, Dennis Fong, Julian Cecchini, and Luigi Quattrociocchi. User guide. https://github.com/lilweege/SyntaxSentinels/blob/main/docs/UserGuide/UserGuide.pdf, November 2024e.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

[This section is not required for CAS 741 —SS]

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

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

4. 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?