System Verification and Validation Plan for Software Engineering

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

October 29, 2024

Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

[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

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 (Author, 2019) tables, if appropriate —SS]

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

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

2 General Information

2.1 Summary

[Say what software is being tested. Give its name and a brief overview of its general functions. —SS]

2.2 Objectives

[State what is intended to be accomplished. The objective will be around the qualities that are most important for your project. You might have something like: "build confidence in the software correctness," "demonstrate adequate usability." etc. You won't list all of the qualities, just those that are most important. —SS]

[You should also list the objectives that are out of scope. You don't have the resources to do everything, so what will you be leaving out. For instance, if you are not going to verify the quality of usability, state this. It is also worthwhile to justify why the objectives are left out. —SS]

[The objectives are important because they highlight that you are aware of limitations in your resources for verification and validation. You can't do everything, so what are you going to prioritize? As an example, if your system depends on an external library, you can explicitly state that you will assume that external library has already been verified by its implementation team. —SS]

2.3 Challenge Level and Extras

[State the challenge level (advanced, general, basic) for your project. Your challenge level should exactly match what is included in your problem statement. This should be the challenge level agreed on between you and the course instructor. You can use a pull request to update your challenge level (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

[Summarize the extras (if any) that were tackled by this project. Extras can include usability testing, code walkthroughs, user documentation, formal proof, GenderMag personas, Design Thinking, etc. Extras should have already been approved by the course instructor as included in your problem statement. You can use a pull request to update your extras (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

2.4 Relevant Documentation

[Reference relevant documentation. This will definitely include your SRS and your other project documents (design documents, like MG, MIS, etc). You can include these even before they are written, since by the time the project is done, they will be written. You can create BibTeX entries for your documents and within those entries include a hyperlink to the documents. —SS]

Author (2019)

[Don't just list the other documents. You should explain why they are relevant and how they relate to your VnV efforts. —SS]

3 Plan

This section outlines the structured plan for verification and validation of the project. It defines team roles, describes approaches for verifying different phases of the development process, and specifies tools to be used in ensuring the project meets quality standards. The subsections cover SRS verification, design verification, implementation verification, automated testing tools, and software validation.

3.1 Verification and Validation Team

The following table lists team members and their respective roles in the verification and validation process. Each member will be responsible for writing test cases, executing tests, and documenting results for their assigned areas. The team will meet regularly to discuss progress, address issues, and ensure alignment with project goals.

Team Member	Role in Verification and Validation
Mohammad Mohsin Khan	Oversees system architecture verification and leads checklist creation.
Lucas Chen	Security verification, focusing on access control and data flow assessments.
Dennis Fong	Responsible for interface and compatibility reviews between components.
Julian Cecchini	Ensuring that individual modules or components integrate smoothly with one another.
Luigi Quattrociocchi	Tracks checklist items and maintains verification documentation.

Table 1: Verification and Validation Team Roles

3.2 SRS Verification Plan

The SRS (Software Requirements Specification) verification will ensure that all functional and non-functional requirements are accurately documented and align with the project goals. The verification plan includes:

- **Structured Reviews**: Team members will perform a structured review of the SRS document, verifying that all requirements are feasible and testable.
- **Checklist-Based Verification**: An SRS checklist will be used to ensure all critical elements, such as functional completeness and clarity, are covered.
- **Reviewer Feedback Sessions**: We will gather feedback from peer reviewers and our project supervisor, meeting to discuss any discrepancies or ambiguous requirements. These meetings will include task-based inspections, where reviewers are asked to analyze requirements based on specific scenarios.

Item	Description
1. Completeness	
1.1 Purpose and Scope	Document states the purpose and scope of the project.
1.2 Stakeholders	Key stakeholders (clients, users) are defined.
1.3 Functional Requirements	All primary functions (e.g., plagiarism detection, reporting) are covered.
1.4 Non-Functional Requirements	Includes performance, usability, and security requirements.
2. Clarity	
2.1 Unambiguous Terminology	Each requirement is clearly stated, terms are defined (e.g., MOSS, NLP).
2.2 Glossary Completeness	All acronyms and terms are included in the glossary.
3. Consistency	
3.1 Consistent Terminology	Terminology and references are consistent throughout.
3.2 No Conflicting Requirements	No contradictory requirements (e.g., conflicting performance vs. security).
4. Verifiability	
4.1 Testable Requirements	Each requirement is testable (e.g., accuracy metrics, response times).
4.2 Acceptance Criteria	Clear acceptance criteria for each requirement.
5. Traceability	
5.1 Unique Identifiers	Each requirement has a unique identifier.
5.2 Source of Requirements	Requirements link to stakeholder needs or project goals.
6. Feasibility	
6.1 Technical Feasibility	Requirements are achievable within project constraints.
6.2 Practical Constraints	Constraints such as budget and timeline are realistic.

Item	Description
7. Security and Privacy	
7.1 Data Retention Policy	Compliance with data privacy laws (e.g., PIPEDA).
7.2 Access Control Requirements	Requirements for user authentication and authorization are clear.
8. Modifiability	
8.1 Organized Structure	Requirements are logically organized for easy updates.
8.2 No Redundancies	No duplicate requirements to avoid confusion.
9. Compliance and Ethics	
9.1 Legal and Ethical Standards	Legal (e.g., Copyright) and ethical considerations are addressed.

Table 2: Checklist for SRS Verification

3.3 Design Verification Plan

The design verification plan aims to ensure that the design accurately implements the requirements and adheres to best practices. This plan includes:

- **Checklist-Based Design Review**: A checklist will be used to guide reviews, focusing on key aspects such as modularity, scalability, and security.
- **Peer Design Reviews**: Peer reviews by classmates and team members will provide feedback on the design, highlighting potential areas of improvement.
- **Regular Team Reviews**: Scheduled meetings will allow the team to discuss any design modifications, verify alignment with the SRS, and ensure consistency across components.

The checklist for design review will cover:

- Modularity and Scalability
- Security of data flows and access control
- Component interfaces
- Error handling and fault tolerance

3.4 Verification and Validation Plan Verification Plan

The verification and validation plan itself will also undergo verification to ensure its effectiveness. This will be achieved by:

- **Peer Reviews**: Classmates will review the plan to provide feedback on its clarity, feasibility, and alignment with project requirements.
- **Mutation Testing**: We will apply mutation testing to validate that the plan can effectively catch errors and discrepancies in project requirements and implementation.
- **Checklist-Based Verification**: A checklist specific to the verification and validation plan will guide reviewers in assessing all critical aspects of the plan.

3.5 Implementation Verification Plan

The implementation verification plan focuses on ensuring the correctness and quality of the implementation phase. The plan includes:

- **Unit Testing**: Each function and module will undergo unit testing to ensure they meet functional requirements.
- **Static Code Analysis**: We will use static analysis tools to verify code quality, adherence to coding standards, and security practices.
- **Code Walkthroughs**: Code walkthroughs will be held in team meetings and during the final presentation, allowing team members to inspect each other's code for issues in logic, structure, and readability.

3.6 Automated Testing and Verification Tools

To streamline the verification process, the following automated testing and verification tools will be used:

- **Unit Testing Framework**: A unit testing framework (such as JUnit for Java or Pytest for Python) will be used to automate testing of individual functions and modules.
- **Static Analysis Tools**: Tools such as SonarQube or Valgrind will be used for static analysis to detect code issues, potential security vulnerabilities, and memory leaks.
- **Continuous Integration (CI) Tool**: GitHub Actions or Jenkins will be set up for continuous integration to ensure that tests are automatically run on new code submissions.
- **Code Coverage Tool**: Code coverage tools will track the extent to which the codebase has been tested, ensuring that all critical paths are covered.
- **Linters**: Linters appropriate to the project's programming language will enforce coding standards, improving code readability and maintainability.

3.7 Software Validation Plan

The software validation plan outlines the strategies for validating that the software meets the intended requirements. This includes:

- **User Review Sessions**: Review sessions with stakeholders and user representatives will be conducted to validate that the system meets user needs and expectations.
- **Rev 0 Demonstration**: Shortly after the scheduled Rev 0 demo, we will seek feedback from stakeholders and supervisors, if applicable, to confirm that the design and initial implementation align with project goals.
- **Usability Testing**: We will plan user testing sessions to ensure the software is intuitive and meets usability expectations.

The validation process will involve gathering external data, where possible, to test the system's accuracy and performance under realistic scenarios.

Summary

This comprehensive plan for verification and validation addresses each phase of the project lifecycle, from requirements to implementation. By following a structured approach with well-defined roles, checklists, and automated tools, we aim to ensure a high level of quality, accuracy, and security in the final product.

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 should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

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:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   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:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   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:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Author Author. System requirements specification. https://github.com/..., 2019.

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?