# System Verification and Validation Plan for Software Engineering

Team #18, Gouda Engineers
Aidan Goodyer
Jeremy Orr
Leo Vugert
Nathan Perry
Tim Pokanai

October 25, 2025

# 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

The following section will outline a verification and validation plan to follow throughout the project timeline. Since our development of the project is executed in multiple sequential steps, this section will outline the verification and validation plans, as well as the tools and methods used for each milestone component leading up to our final software solution. This section will also list the entities that will participate in the verification and validation steps. First we will outline who will be involved in the verification and validation team. Then we will discuss the verification plans of the SRS, design, VnV Plan, implementation, and the automatic tools we use for testing and verification. Lastly, we will discuss how we will perform the validation of our software.

[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]

- **Aidan Goodyer**,**Jeremy Orr**,**Nathan Perry**,**Timothy Pokanai**,**Leo Vugert**:

  The five members of the capstone team will split the testing evenly among themselves. All members will be equally expected to contribute to the synthesis of both unit and system tests as well as setting up automated tests down the line. Since the author of the System Testing plan was **Jeremy Orr** and the author of the Unit Testing plan will be **Nathan Perry**, they will be considered the head of system testing and unit testing respectively. Although this does not require them to necessarily produce more tests, they will be expected to be prepared to provide guidance to other group members if they need it.

- **Dr. Henry Szechtman**, **Dr. Anna Dvorkin-Gheva**:

  The two supervisors of this project will be involved in the testing process as if they were early access users. Given that they represent a demographic very similar to what our actual users will look like (Albeit with a much better understanding of the data), their feedback will be extremely useful in understanding how well our system will be recieved by users. They will perform both User Acceptance Testing and and Usability testing on our system; trying out the various functionalities and commenting on if they think it is satisfactory and understandable for the general user base.

## 3.2 SRS Verification

### 3.2.1 SRS Verification Approaches

- **Peer Reviewers:** Our first approach for SRS verification is reviewing the peer review issues created by our classmates. This is not the most rigorous or systematic but allows us to evaluate feedback on our SRS and based on these comments, determine if the SRS is consistent, correct, feasible and complete or if changes are needed.

- **Unstructured Supervisor SRS Review:** Our second approach for SRS verification is to provide our SRS document to our supervisor, Dr. Henry Szechtman and allow him to make comments on it. This approach is also quite unstructured but is still useful in our verification process. By giving our supervisor the time to look over the SRS, it will elicit any large gaps between our understanding of his goals for the system and our interpretation of these goals during our meetings to elicit said requirements. In other words, this will show us any glaring issues in our requirements from the perspective of our supervisor.

- **Structured Supervisor SRS Review:** Our third approach for SRS verification also involves our supervisor but in a much more structured manner. Our approach to this review would be a meeting in which we present to him the key aspects of the SRS and ask him if he sees any issues with these key sections. The main sections (note these do not necessarily refer to specific sections in the SRS template itself) to be covered, in descending order of priority are: Functional Requirements and Use Cases, Project Constraints and Scope, Non-Functional Requirements, Project Drivers and Project Issues. During this review we would ask him things like: Is this section complete? Is there anything missing? Does the way we laid out these requirements seem correct to you? Do these requirements seem feasible to you? Is this in accordance with your conception of what the system will look like?

- **Structured Team SRS Review:** Our final approach for SRS verification will be a review in a much similar format to the structured supervisor review but only with the five members of our team. We would go over the same key aspects of the SRS and ask mostly the same questions regarding these sections as we would in our supervisor review. Doing an internal review like this will be helpful as although the SRS was generated collectively, the actual writing of the requirements was done in a very individual and compartmentalized way and this gives each member an opportunity to closely examine the key parts of our requirements and provide and criticisms, issues or additions that they feel are necessary to each section.

### 3.2.2 SRS Verification Checklist

Below is a general checklist of questions that our SRS should satisfy (i.e. the

answer should be yes):

- Are all of the functional requirements necessary, feasible, complete and provide unique value to the system?
- Do the functional requirements cover every expected function of the system?
- Are all of the use cases necessary, feasible and complete?
- Do the existing use cases cover every expected function of the system?
- Does the SRS cover every relevant project constraint such that there is no unaccounted for aspect of the environment that may prevent us from implementing some of the system's functionality?
- Are all project constraints accurately described such that it is clear what exactly is constrained and it is understandable how this may affect the system?
- Is the scope properly defined such that all project components have been accounted for in the scope?
- Have all relevant out of scope components been listed and rationale provided for why they are out of scope?
- Are all non-functional requirements necessary, feasible, complete and provide unique value to the system?
- For non-functional requirements that are not completely necessary, is their value large enough to justify their existence?
- Have all obligatory non-functional requirements been accounted for, specifically those relating to legality, security or compliance that are imposed by entities outside of the system?

## 3.3 Design Verification

[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

[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

[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

[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 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?