System Verification and Validation Plan for Software Engineering

Team 4, EvENGage
Virochaan Ravichandran Gowri
Omar Al-Asfar
Rayyan Suhail
Ibrahim Quraishi
Mohammad Mahdi Mahboob

October 27, 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 (**?**) 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]

?

[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

[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

[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

[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

- **Vitest:** Used as the primary unit and integration testing framework for the frontend. It integrates seamlessly with the Vite development environment and allows for efficient automated testing of components, hooks, and application logic. Vitest supports mocking and snapshot

testing, which ensures that UI and logic remain consistent after code changes.

- **Playwright:** Utilized for automated end-to-end testing to simulate user interactions across different browsers and devices. This tool can be used to validate complete user workflows such as event registration and form creation and submissions. This will help ensure system functionality and reliability from the user's perspective.
- **Postman and Newman:** Employed for API-level testing and verification. Postman allows the team to design and execute test cases for RESTful endpoints, while Newman enables command-line execution of these tests within the CI/CD pipeline. Together, they ensure backend consistency and correct response handling.
- **GitHub Actions:** Serves as the continuous integration (CI) platform that automates testing and verification tasks. It will execute unit tests, end-to-end tests, and linters automatically on each push or pull request. This will ensure that all commits meet quality and functionality and don't break the system before merging.
- **pgTAP:** Used for database testing in PostgreSQL. It can help ensure database integrity and ensure that that database schema and back-end logic is implemented correctly through unit testing.

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

This section outlines the the system wide tests to validate both functional and non-functional requirements defined in our Software Requirements Specification.

4.1 Tests for Functional Requirements

This section verifies that the implemented system fulfills all **Functional Requirements (FR-1 through FR-11)** defined in the SRS. The tests are grouped into three **Areas of Testing**: *Admin Portal (AP)*, *Attendee Application (AA)*, and *System Integration and Security (SI)*.

4.1.1 Area of Testing 1 – Admin Portal

Test-FR-AP-1 Create Form

Type: Manual
Initial State: Admin is logged in with form creation privileges.
Input/Condition: Admin enters a new form name and adds multiple field types (text, checkbox, rating).
Expected Output: The new form module is created and is visible in the database and in the admin dashboard.
Test Case Derivation: This test verifies the system's ability to support dynamic form creation and storage as required by FR-1 and FR-7.
How the test will be performed: The tester will create a new form and confirm it appears correctly in the interface and database.

Test-FR-AP-2 View and Analyze Form Data

Type: Manual
Initial State: At least two forms with collected responses exist.
Input/Condition: Admin selects multiple forms and clicks button to view report analytics

Expected Output: Aggregated analytics are displayed correctly and match backend data.
Test Case Derivation: This test confirms data organization and analytical visualization functionalities described in FR-2 and FR-8, ensuring accurate reporting and analytics.
How the test will be performed: The tester will generate combined analytics and confirm that the displayed charts match backend results.

Test-FR-AP-3 Manage Events via Dashboard
Type: Manual
Initial State: Several events with participant data exist.
Input/Condition: Admin accesses the dashboard and applies filters (e.g., pending payment).
Expected Output: Filtered results display accurately with correct status counts.
Test Case Derivation: This test validates event management, filtering, and data persistence mechanisms described in FR-5 and FR-6.
How the test will be performed: The tester will apply filters and verify that displayed data matches expected event records.

Test-FR-AP-4 View and Export Event Analytics
Type: Manual
Initial State: Events and surveys contain collected feedback.
Input/Condition: Admin selects an event and exports analytics.
Expected Output: Correct analytics display and downloadable report matches system data.
Test Case Derivation: This test ensures that analytics generation and report export satisfy the requirements for event statistics and reporting defined in FR-8 and FR-10.
How the test will be performed: The tester will view analytics for an event, export the report, and compare the output to dashboard data.

4.1.2 Area of Testing 2 – Attendee Application

Test-FR-AA-1 Register for Event

Type: Automatic
Initial State: An event is open and attendee is authenticated.
Input/Condition: Attendee submits an event registration form.
Expected Output: Registration is saved and a confirmation message is displayed and sent to attendee and a QR ticket is generated.
Test Case Derivation: This test validates the registration workflow and QR ticket generation process as required by FR-3, FR-4, and FR-6.
How the test will be performed: An automated test will submit registration data and verify the confirmation message and QR code has been created. Will confirm that the event data has also been updated in the database.

Test-FR-AA-2 Access Event Listings

Type: Manual
Initial State: The system has multiple upcoming and past events.
Input/Condition: Attendee views "Upcoming" and "Registered Events" tabs.
Expected Output: Each tab displays correct events for that category.
Test Case Derivation: This test confirms accurate retrieval and filtering of events for authenticated users as required by FR-3 and ensures user can view event info.
How the test will be performed: The tester will view event lists and verify they match stored registration records. Tester will also ensure event info shows up correctly and matches database.

Test-FR-AA-3 Fill out Form

Type: Automatic
Initial State: A survey is available and attendee is authenticated.
Input/Condition: Attendee completes and submits the survey.
Expected Output: Survey responses are stored and a confirmation message appears.
Test Case Derivation: This test ensures that survey submission and

feedback processing meet the specifications in FR-7, FR-9.
How the test will be performed: An automated test will submit survey responses and confirm that results are stored and submission is found in the centralized database.

4.1.3 Area of Testing 3 – System Integration and Security

Test-FR-SI-1 Real-Time Data Synchronization
Type: Automatic
Initial State: Admin and attendee interfaces are active.
Input/Condition: Admin updates event details (e.g., venue or date).
Expected Output: Attendee view updates automatically without manual refresh.
Test Case Derivation: This test validates the real-time synchronization and data propagation features required by FR-6 and FR-7 and ensures that both databases are connected to the interface.
How the test will be performed: An automated test will modify event data and verify that the attendee view reflects the change immediately.

Test-FR-SI-2 Role-Based Access Verification
Type: Manual
Initial State: Admin and attendee accounts exist with distinct permissions.
Input/Condition: Each user attempts to access admin-only features.
Expected Output: Admin gains access but attendee is denied access with proper error handling.
Test Case Derivation: This test checks role-based access control and authorization mechanisms as defined in FR-11 ensuring only specific users have access to admin controls.
How the test will be performed: The tester will log in as both user types and attempt to access admin features to confirm proper restriction.

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

Test ID	Functional Requirements Covered
Test-FR-AP-1	FR-1, FR-7
Test-FR-AP-2	FR-2, FR-8
Test-FR-AP-3	FR-5, FR-6
Test-FR-AP-4	FR-8, FR-10
Test-FR-AA-1	FR-3, FR-4, FR-6
Test-FR-AA-2	FR-3
Test-FR-AA-3	FR-7, FR-9
Test-FR-SI-1	FR-6, FR-7
Test-FR-SI-2	FR-11

Table 1: Mapping of System Tests to Functional Requirements

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]

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:

Virochaan Ravichandran Gowri Reflection

1. **What went well while writing this deliverable?** Since we had already completed the SRS deriving test cases to ensure that the different requirements were covered was quite easy and straightforward. We used some of the diagrams and components identified within the SRS to breakdown the functional system tests and for the NFR system tests

2. **What pain points did you experience during this deliverable, and how did you resolve them?** One pain point that was more of an annoyance was ensuring traceability and that we kept a good track of what test cases were covering which requirements and not having too much overlap on the requirements we were validating. We also wanted to ensure that the test were actually feasible and did not put too high of a burden on us. This meant that we had to consider both manual and automatic tests that could be accomplished.

Group Reflection

1. **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.**

   (a) **Automated Testing with CI/CD**: We need to ensure we understand how test pipelines work and how some of these automated testing tools such as ViteTest and Playwright will integrate within our workflows. As mentioned in the SRS we also need to build upon our knowledge on GitHub Actions to ensure that we utilize it properly to have a proper testing workflow.

   (b) **Static Testing and Code Quality Assurance**: Understand and implement static testing methods such as code inspections and code reviews. It will help us find more tools to help perform static analysis to find defects early in the development lifecycle.

   (c) **Dynamic Testing**: Build upon pre-existing dynamic testing skills and find ways to use it in our use case. This includes developing better system and unit testing skills and developing skills in creating test cases.

   (d) **Usability Testing**: Though this might relate more to the Usability report, we need to develop this skill to ensure that we can create a positive user experience.

   (e) **ViteTest and Playwright**: These two tools are the most important tools that we will be using throughout our testing phase of the project so it is important we gain a good understanding of these tools to utilize them to the highest level.

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?**

(a) **Automated Testing with CI/CD**: To acquire this knowledge we can study the official documentation and tutorials for GitHub Actions as well as related automated testing tools. This will help us understand configuration and workflow syntax. We can also analyze open-source repositories that employ CI/CD with similar front-end stacks to observe best practices and reusable workflow patterns.

(b) **Static Testing and Code Quality Assurance**: This knowledge can be developed by completing structured online modules or guides on static code analysis, focusing on tools that are easy to use for our use case. We can also conduct regular peer-led code reviews and checklist-based inspections to ensure we hold each other accountable and figure out our code patterns.

(c) **Dynamic Testing**: The team can strengthen this skill by taking short online courses or tutorials on unit, integration, and system testing methodologies and find videos on testcase design. We can also look back to our SFWRENG 3S03 where we worked first hand on developing test cases to ensure we can find ways to maximize code coverage.

(d) **Usability Testing**: To develop this skill, the team could review Human–Computer Interaction (HCI) principles and usability evaluation methods through our course work. We can also look into tutorials online about effective UX design to and link usability testing to front-end design. We can also discuss with our stakeholders and supervisors their expectations to develop our own testing framework.

(e) **ViteTest and Playwright**: The team can gain proficiency by following official tutorials and documentation to build example test suites and understand configuration options for both tools. We can also look to open-source projects or community-driven examples to learn advanced testing features that these tools offer.

What each member will pursue:

(a) **Virochaan**: Virochaan will focus on mastering **Automated Testing with CI/CD**, particularly the integration of GitHub Actions

with Vitest and Playwright. Since he was specified as the DevOps manager it will be important to know to create proper Github workflows. From this he will also need to know the details of **Vitest and Playwright** to ensure that we develop the proper testing protocol for this. He will primarily focus on looking at tutorials and find open source projects with workflows that we can look to understand to develop our own.

(b) **Mohammad**: Mohammad has worked with static coding tools in the past so he will look to develop those skills further by focusing on **Static Testing**. He will do this by researching static testing tools for our usecase and looking at tutorials and examples where static testing has been done for full-stack apps.

(c) **Omar**: Omar has experience developing mobile apps so he will focus on **Usability Testing** to ensure that he can develop the best user experience possible. He will do this by primarily looking at video tutorials on usability testing and in our supervisor meetings develop a testing framework for our UI.

(d) **Ibrahim**: Ibrahim will look to focus on **Dynamic Testing** with **Vitest and Playwright**. These tools also offer dynamic testing capabilities and it is important we figure out how to connect the two effectively. Ibrahim will focus on looking through tutorials for the two technologies and watching videos on testing methodologies so we can implement the methodologies for our usecase.

(e) **Rayyan**: Rayyan has experience working with testing through his coop experiences. To build upon this he will focus on **Dynamic Testing** and **Static Testing** and will look to be well rounded on this. In our roles we also identified him as a web app tester so this will allow him to develop good testcases to accomplish this. He will primarily focus on looking at tutorials on dynamic and static testing and how we can connect the two together to make a better testing framework.