System Verification and Validation Plan for MES-ERP

Team #26, Ethical Pals
Sufyan Motala
Rachid Khneisser
Housam Alamour
Omar Muhammad
Taaha Atif

November 4, 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

Abbreviation	Description
MES	McMaster Engineering Society
V&V	Verification and Validation
SRS	Software Requirements Specification
UI	User Interface
API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Deployment

This section defines the symbols, abbreviations, and acronyms used throughout the document to ensure clarity and consistency.

This document outlines the Verification and Validation (V&V) Plan for the McMaster Engineering Society Custom Financial Expense Reporting Platform. The V&V plan will ensure that the software performs as expected, meets user requirements, is secure, and reliable. The primary goals of this document are to define the objectives, methods, and scope of the V&V process, covering aspects such as requirements validation, design verification, and system testing.

2 General Information

2.1 Summary

The MES Custom Financial Expense Reporting Platform is a web-based application designed to streamline financial operations for McMaster Engineering Society and its associated student groups. It enables users to submit, review, and track reimbursement requests, create budgets, and manage expenses in a transparent, auditable, and efficient manner. This V&V Plan outlines the verification and validation processes for ensuring the accuracy, functionality, and security of the software.

2.2 Objectives

The primary objective of this V&V Plan is to confirm that the MES Custom Financial Expense Reporting Platform meets its requirements and operates as intended. The plan aims to:

- Build confidence in the software's functional correctness and ensure it meets all outlined requirements.
- Validate usability by demonstrating a clear, intuitive, and user-friendly interface for all key user roles.
- Ensure system security and data integrity, especially regarding sensitive financial data and authorization levels.

The plan does not prioritize nonfunctional aspects such as aesthetic design consistency or performance optimizations beyond standard system requirements, as these are not critical to initial functionality and budget constraints.

2.3 Challenge Level and Extras

The challenge level for this project is classified as **General**, aligning with the expectations outlined in the problem statement. The team has chosen to address the critical needs of verification of functionality, security, and usability. No additional extras beyond the required project scope are included in this V&V Plan.

2.4 Relevant Documentation

The following documents are relevant to this V&V plan and provide a basis for verification activities:

- **Software Requirements Specification (SRS)** (Atif et al., 2024d): Defines the functional and nonfunctional requirements of the system, serving as a primary source for validation and test case development.
- **Hazard Analysis**(Atif et al., 2024b): Outlines potential system risks and serves as a guide for security testing and fault tolerance validation.
- **Problem Statement**(Atif et al., 2024c): Provides an overview of the project's purpose, goals, and core functionality requirements, detailing the needs of the McMaster Engineering Society for a streamlined financial reporting platform. It identifies key problems in the current system, justifying the need for an automated, digital solution to enhance efficiency and reliability in financial processes.
- **Development Plan**(Atif et al., 2024a): Outlines the project's development lifecycle, including phases for requirements gathering, design, implementation, and testing. It establishes milestones, assigns roles, and defines the timeline, while ensuring that the team follows best practices in Agile development, with iterative check-ins for verification and validation of functional requirements.

These documents establish the foundation for testing activities, allowing each verification and validation activity to target specific functional areas, requirements, or potential risk zones.

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

The implementation of the verification plan will be accomplished through a multi faceted approach:

Code Reviews

- All code changes will undergo peer review through GitHub Pull Requests before being merged into the main branch.
- At least one team member has to approve the pull request before it can be merged.
- Code reviews will use a standardized checklist focusing on:
  - Implementation correctness against requirements.
  - Security considerations for financial data handling.
  - Test coverage verification.
  - Error handling completeness.

Test Execution

- System tests as detailed in Section 4, with particular focus on:
  - Functional tests for reimbursement submission (Section 4.1).
  - Security and data integrity tests (Section 4.2).
  - Performance and scalability tests (Section 4.2).
- Unit tests as outlined in Section 5, covering:
  - Individual module testing.
  - Integration points between components.
  - Edge cases and error conditions.

Static Analysis

- ESLint will be used for static code analysis.
- Type checking through TypeScript's compiler.
- SonarQube will be used for code quality and security analysis.

Review Sessions

- Weekly review sessions will be held with the team to discuss progress, issues, and verification results. This meeting will additionally be held with the project supervisor on a monthly basis.
- Review sessions will include a demonstration of new features and verification results.
- Occasionally review sessions will include a demonstration of the system to stakeholders.
- The checkpoint meetings with the TA will also serve as a review session.
- The final presentation will serve as a comprehensive code walkthrough.

3.6 Automated Testing and Verification Tools

The following tools will be utilized for automated testing and verification, aligned with our development technology stack:

Testing Frameworks

- Jest: Primary testing framework for unit and integration testing.
- React Testing Library: Component testing focusing on user interactions and UI behavior.
- Cypress: End-to-end testing for user flows and system integration.

Code Quality Tools

- ESLint: Configured with specific rules for TypeScript and Next.js development.
- Prettier: Automated code formatting on commit to maintain consistent style.
- TypeScript: Static type checking with strict mode enabled.
- Git pre-commit hooks: Automated formatting and linting checks before commits are allowed.

Continuous Integration

- GitHub Actions for:
  - Automated test execution on each commit.
  - Code quality checks through ESLint.
  - Type checking through TypeScript compiler.
  - Automated deployment to staging environment for pull requests.
- Test coverage reporting through Jest with:
  - Minimum 90% coverage requirement as specified in the Development Plan.
  - Coverage reports generated for each pull request.
  - Blocking of merges if coverage thresholds aren't met.

Performance Monitoring

- Lighthouse: Performance, accessibility, and best practices monitoring.
- Chrome DevTools: Memory usage and rendering performance analysis.
- Next.js Analytics: Built-in performance monitoring for production deployments.

Security Testing

- npm audit: Regular dependency vulnerability scanning.
- Automated vulnerability scanning integrated into CI pipeline.

All tools have been selected to integrate seamlessly with our Next.js and TypeScript stack while supporting our coding standards and quality requirements. The automation of these tools through our CI/CD pipeline ensures consistent verification across all stages of development.

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 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

Taaha Atif, Omar Muhammad, Housam Alamour, Sufyan Motala, and Rachid Khneisser. System requirements specification. https://github.com/Housam2020/MES-ERP/blob/main/docs/DevelopmentPlan/DevelopmentPlan.pdf, 2024a.

Taaha Atif, Omar Muhammad, Housam Alamour, Sufyan Motala, and Rachid Khneisser. System requirements specification. https://github.com/Housam2020/MES-ERP/blob/main/docs/HazardAnalysis/HazardAnalysis.pdf, 2024b.

Taaha Atif, Omar Muhammad, Housam Alamour, Sufyan Motala, and Rachid Khneisser. System requirements specification. https://github.com/Housam2020/MES-ERP/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf, 2024c.

Taaha Atif, Omar Muhammad, Housam Alamour, Sufyan Motala, and Rachid Khneisser. System requirements specification. https://github.com/Housam2020/MES-ERP/blob/main/docs/SRS/SRS.pdf, 2024d.

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?