

System Verification and Validation Plan for MES-ERP

Team #26, Ethical Pals

Sufyan Motala

Rachid Khneisser

Housam Alamour

Omar Muhammad

Taaha Atif

November 5, 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

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	5
3.7	Software Validation Plan	6
4	System Tests	9
4.1	Tests for Functional Requirements	9
4.1.1	Reimbursement Submissions	9
4.1.2	Budget Management	10
4.2	Tests for Nonfunctional Requirements	10
4.2.1	Performance Testing	11
4.2.2	Security Testing	11
4.3	Traceability Between Test Cases and Requirements (above) . .	12
5	Unit Test Description	12
5.1	Unit Testing Scope	12
5.2	Tests for Functional Requirements	13
5.2.1	Module 1	13
5.2.2	Module 2	14
5.3	Tests for Nonfunctional Requirements	14
5.3.1	Module ?	14
5.3.2	Module ?	14
5.4	Traceability Between Test Cases and Modules	15

6	Appendix	16
6.1	Symbolic Parameters	16
6.2	Usability Survey Questions?	16

List of Tables

1	Traceability Between Test Cases and Requirements	12
	[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

The validation plan ensures the software meets the needs of the McMaster Engineering Society and its 60 student groups through a comprehensive approach:

External Data Validation

- Historical reimbursement data from the MES's previous Excel-based system will be used to validate:
 - Accuracy of financial calculations.
 - Correct budget categorization.
 - Processing time improvements.
- Sample receipts provided by the MES will be used to validate the receipt processing system.

Stakeholder Requirements Review

- Task-based inspection sessions with MES VP Finance to verify SRS completeness:
 - Walkthrough of common reimbursement scenarios.
 - Review of financial approval workflows.
 - Verification of audit requirements.
- Requirements validation through prototype demonstrations:
 - Interactive sessions with student group leaders.
 - Feedback collection on user interface design.
 - Verification of notification preferences.

Rev 0 Demo Validation

- Scheduled demonstration with external supervisor (MES VP Finance).
- Focus areas for supervisor feedback:
 - Validation of implemented requirements from Section 9.1 of the SRS.
 - Verification of financial workflow accuracy.
 - Assessment of security measures.
 - User interface evaluation.

- This feedback will be documented and incorporated into subsequent development iterations.

User Acceptance Testing

- Beta testing program with selected student groups:
 - Testing with real reimbursement scenarios.
 - Validation of notification systems.
 - Performance under typical usage patterns.
- Specific validation metrics:
 - Reimbursement request completion time.
 - Budget categorization accuracy (target: 90%).
 - System response times under load.
 - User error rates during submission process.

Requirements Validation

- Regular requirement review sessions to ensure:
 - Alignment with MES financial policies.
 - Compliance with security requirements.
 - Meeting of performance targets.
- Validation of specific SRS requirements:
 - Custom budget creation functionality.
 - Real-time ledger tracking.
 - Multi-level approval workflows.
 - Automated notifications.

All validation activities will be documented and tracked through GitHub. Feedback will be incorporated into the development cycle. There will be particular attention to the core requirements of reducing wait times and preventing loss of reimbursement requests. Any changes to requirements identified through validation will be reviewed and updated in the SRS document accordingly.

4 System Tests

This section outlines the testing strategies for the functional and nonfunctional requirements of the MES Custom Financial Expense Reporting Platform. Each test case is linked to the requirements specified in the SRS to ensure thorough validation of the application.

4.1 Tests for Functional Requirements

These tests validate core features of the platform, such as reimbursement processing, budget management, and approval workflows. The subsections break down tests into areas that align with key functionalities outlined in the SRS.

4.1.1 Reimbursement Submissions

Test 1: Submission Confirmation

- **Test ID:** test-id1
- **Control:** Automatic
- **Initial State:** User logged in, valid reimbursement document available
- **Input:** A completed reimbursement form with attached receipt
- **Output:** Confirmation message and ledger update
- **Test Case Derivation:** Ensures the form submission logic updates the ledger correctly, as described in the SRS
- **How Test Will Be Performed:** Submit the form and verify through the UI and database that the ledger is updated

Test 2: Approval Workflow

- **Test ID:** test-id2
- **Control:** Manual
- **Initial State:** Admin logged in, pending request available

- **Input:** An admin approval action on a pending reimbursement request
- **Output:** Status changes to approved, and notifications are sent
- **Test Case Derivation:** Derived from multi-level approval requirements to ensure workflow consistency
- **How Test Will Be Performed:** Approve the request and check the status change, notification delivery, and database integrity

4.1.2 Budget Management

Test 3: Expense Addition

- **Test ID:** test-id3
 - **Control:** Automatic
 - **Initial State:** Initialized budget with no transactions
 - **Input:** An expense entry into the system
 - **Output:** Budget overview reflects the new expense
 - **Test Case Derivation:** Ensures real-time updates to the budget, as required by the SRS
 - **How Test Will Be Performed:** Add an expense and verify updates in the UI and database
-

4.2 Tests for Nonfunctional Requirements

These tests address performance, usability, and security aspects of the application. They ensure the system is efficient and meets user expectations.

4.2.1 Performance Testing

Test 4: Load Handling

- **Test ID:** test-id10
- **Type:** Dynamic
- **Initial State:** User dashboard loaded with multiple active sessions
- **Input/Condition:** Simultaneous form submissions from multiple users
- **Output/Result:** System responds within 5 seconds under load
- **How Test Will Be Performed:** Manually simulate multiple users submitting forms at the same time by having multiple team members perform form submissions in parallel on different devices or browsers.

4.2.2 Security Testing

Test 5: Data Integrity Check

- **Test ID:** test-id11
- **Type:** Static and Dynamic
- **Initial State:** System ready, database connected
- **Input:** Attempt to modify sensitive data without proper authorization
- **Output:** Unauthorized modification attempt is blocked
- **How Test Will Be Performed:** Manually attempt to access and modify sensitive data (e.g., through the application interface or API) without proper authorization. This could involve using invalid user credentials, altering URL parameters, or testing with non-privileged user accounts. The system should reject these unauthorized actions and prevent any changes to sensitive data.

Requirement	Test Case(s)
Reimbursement Submission	test-id1, test-id2
Budget Management	test-id3
Performance Under Load	test-id10
Security and Data Integrity	test-id11

Table 1: Traceability Between Test Cases and Requirements

4.3 Traceability Between Test Cases and Requirements (above)

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, your 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?

Sufyan: What went well for me this assignment was getting github actions to build the pdf on commit. I also enjoyed learning about all the tooling we could use for the project to ensure code quality, test coverage and security.

Taaha What went well while writing this deliverable was the amount of support I had when asking teammates questions. Was extremely helpful whenever I came across an issue I was unsure of. **Omar:** What went well was the clarity of this document, we understand our project so we were able to devise this document with little to no questions asked to the TA.

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

Sufyan: The pain points I experienced were getting the github actions to work properly. I had to do a lot of research and trial and error to get it to work. I also had some issues with the formatting of the document. I resolved these issues by asking for help from my team members.

Taaha The pain points I experienced were mostly related to formatting the document correctly. In order to resolve this, I got help from teammates and looked online. **Omar:** I got really sick during this deliverable which slowed down my progress a lot, in addition, it was harder to manage this deliverable since I had a bunch of other deliverables due near the same time.

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?