Verification and Validation Report: MES-ERP

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

March 10, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 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.

Contents

List of Tables

List of Figures

This document ...

3 Functional Requirements Evaluation

4 Nonfunctional Requirements Evaluation

4.1 Usability

4.2 Performance

4.3 etc.

5 Comparison to Existing Implementation

The MES-ERP system represents a significant advancement over the previous implementation, which relied on Google Forms for expense submissions and spreadsheets for budget tracking and reimbursement processing. This section compares the two implementations across several key dimensions.

5.1 Process Efficiency

- **Previous Implementation:** The Google Forms and spreadsheets approach required manual data transfer between systems. Staff had to copy information from form submissions into tracking spreadsheets, manually update status changes, and communicate updates via separate email threads.
- **MES-ERP:** The new system provides an end-to-end digital workflow where data flows automatically through the system. Once a request is submitted, it remains in the database and can be tracked, updated, and processed without manual data re-entry.

5.2 User Experience

- **Previous Implementation:** Users had to navigate to different Google Forms for different types of requests. After submission, they had limited visibility into the status of their requests and often had to follow up via email.

- **MES-ERP:** Users now have a centralized dashboard where they can submit different types of requests, track the status of all their submissions, and receive automatic email notifications when their request status changes.

5.3 Role-Based Access Control

- **Previous Implementation:** Access control was limited to basic Google permissions. Different spreadsheets were shared with different stakeholders, creating information silos and making it difficult to maintain a comprehensive view of financial activities.

- **MES-ERP:** The system implements a sophisticated role-based access control system that allows for fine-grained permission management. Users can be assigned specific roles (e.g., club admin, MES executive) with appropriate permissions, ensuring they can access only the information and functions relevant to their responsibilities.

5.4 Budget Management

- **Previous Implementation:** Budget tracking was done in separate spreadsheets, making it difficult to get real-time insights into spending patterns or remaining budgets. Annual budget planning was a separate process with limited integration to actual spending.

- **MES-ERP:** The system integrates budget planning and expense tracking in a single platform. The operating budget module allows for detailed budget planning, while the analytics dashboard provides real-time visibility into spending patterns, budget utilization, and financial trends.

5.5 Notification System

- **Previous Implementation:** Status updates and approvals were communicated manually via email, leading to delays and sometimes missed communications.

- **MES-ERP:** The system includes an automated notification system that sends emails to users when their request status changes, providing clear information about the current status and expected next steps.

5.6 Group Management

- **Previous Implementation:** Group affiliations were tracked manually in spreadsheets, making it difficult to enforce group-specific budget limits or permissions.
- **MES-ERP:** The system includes dedicated group management functionality, allowing administrators to create and manage groups, assign users to groups, and enforce group-specific budget allocations and permissions.

5.7 Analytics and Reporting

- **Previous Implementation:** Generating reports or analytics required manual data extraction and processing from multiple spreadsheets.
- **MES-ERP:** The system includes a built-in analytics dashboard that provides visualizations of spending patterns, budget utilization, request volumes, and other key metrics, enabling data-driven decision making.

The MES-ERP system has successfully addressed the key limitations of the previous implementation, providing a more efficient, transparent, and user-friendly platform for managing the MES financial processes. The integration of reimbursement requests, budget planning, user role management, and automated notifications into a single system has significantly improved the overall experience for both users submitting requests and administrators managing the financial processes.

6 Unit Testing

Unit testing was conducted using a combination of automated and manual testing to ensure key functionalities worked as expected. The primary objectives were:

- Verify that critical features, such as reimbursement submissions and approvals, group creation, and user management work correctly.
- Identify and resolve potential errors early in the development cycle.
- Ensure that the system remains stable after modifications.

6.1 Testing Approach

The team used **Jest** for automated testing of frontend logic and API calls, and **manual testing** for real-world scenario validation. The approach included:

- Writing unit tests for key functions such as group creation, deletion, and authentication handling.
- Conducting manual test runs where team members acted as users submitting and approving reimbursement requests.
- Gathering feedback from a small group of MES student leaders who tested the platform and provided insights.
- Running scenario-based testing sessions to simulate real-world usage, such as handling multiple reimbursement requests at once.

6.2 Unit Test Cases

The test cases covered core functionalities, including validation checks and API interactions:

Table 1: Sample Unit Test Cases

Test Case	**Expected Output**
Create a group with a unique name	Group is successfully created
Attempt to create a group with an existing name	System displays an error message
Unauthorized user attempts to access group management	Access is denied with error message

Continued on next page

Table 1 – *Continued from previous page*

Test Case	**Expected Output**
Delete a group with no assigned users	Group is successfully deleted
Attempt to delete a group with assigned users	System prevents deletion and notifies the user
Handle failed database connection during group fetch	Error is logged, and fallback UI is displayed
Register a new user with valid credentials	User is successfully created and logged in
Attempt to register with an existing email	System displays an error message
Login with correct credentials	User is authenticated and redirected to the dashboard
Login with incorrect credentials	System displays an error message
Assign a role to a user	User is successfully assigned the role
Remove a role from a user	User no longer has the role permissions
Attempt to assign an invalid role	System displays an error message
Submit a payment request with valid details	Request is successfully submitted
Submit a payment request missing required fields	System displays an error message
View all payment requests as an admin	Admin sees all requests
View only personal payment requests as a user	User sees only their own requests
Attempt to approve a request without proper permissions	Access is denied with an error message
Update request status successfully	Request status is updated

Continued on next page

Table 1 – *Continued from previous page*

Test Case	**Expected Output**
Handle failed database connection when fetching requests	Error is logged, and fallback UI is displayed

6.3 Edge Cases Tested

- Submitting a reimbursement request without required fields.
- Uploading incorrect file formats for receipts.
- Attempting to create a duplicate group.
- Handling multiple users modifying the same data simultaneously.
- Ensuring database rollback occurs when a transaction fails.
- Registering a user with an invalid email format.
- Assigning multiple roles to a user and verifying access control.
- Submitting a payment request with an incorrect amount format.
- Preventing unauthorized users from updating request statuses.
- Ensuring bulk request approvals work correctly.

7 Changes Due to Testing

Based on the testing process and feedback from users, the following modifications were made to the system:

7.1 Bug Fixes

- Fixed an issue where reimbursements submitted without attachments were still being processed.
- Addressed a bug where users could submit duplicate requests.
- Resolved a login issue where incorrect error messages were displayed.

- Fixed a bug where request status updates were not properly reflected in the UI.
- Added the ability for users to be in multiple groups.

7.2 User Feedback and Enhancements

- Improved the **receipt upload system** by making the file upload process clearer.
- Enhanced **audit logging** for better tracking of request status changes.
- Added a clearer role management interface to reduce confusion when assigning roles.
- Improved user experience by adding filters to search for specific payment requests.

7.3 Performance and Security Improvements

- Implemented basic input validation to prevent incorrect data submissions.
- Reduced system load times by optimizing database queries.
- Strengthened authentication by enforcing stricter password requirements.
- Enhanced role-based access control to prevent unauthorized modifications.

8 Automated Testing

9 Trace to Requirements

10 Trace to Modules

11 Code Coverage Metrics

To ensure the robustness and reliability of the MES-ERP system, we will use a combination of code coverage metrics to evaluate the effectiveness of our testing. The goal of code coverage analysis is to measure the extent to which the source code is tested, helping identify untested paths and potential vulnerabilities.

11.1 Code Coverage Measurement Approach

The following strategies were used to assess the coverage of our codebase:

- **Statement Coverage**: Verifies that each executable statement in the code has been executed at least once.
- **Branch Coverage**: Ensures that both the expected inputs and the expected alerts for wrong inputs are executed (if/else).
- **Function Coverage**: Confirms that all functions and methods in the system have been called at least once.
- **Line Coverage**: Measures the percentage of total lines of code executed during testing.

References

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Reflection.

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?
   Omar: What went well while writing this deliverable was the structured approach we took from the beginning. By clearly defining our Verification and Validation (VnV) strategy early on, we were able to efficiently organize the document and ensure all necessary components were included.

2. What pain points did you experience during this deliverable, and how did you resolve them?
   Omar: One of the main pain points I experienced during this deliverable was ensuring that the VnV plan aligned with our actual testing process and our original VnV plan. To accomodate for this I just went back and forth between documents and talked to my team to ensure I did not mix anything up.

3. Which parts of this document stemmed from speaking to your client(s) or a proxy (e.g. your peers)? Which ones were not, and why?
   Team: The majority of this document stemmed from discussions with my peers, as we collectively determined the best approach for verification and validation. Specific sections, such as requirements and changes due to testing all stem from our stakeholders. We built upon what we

know from our stakeholders for those sections. Other sections were built upon feedback received from stakeholders as well as the TA/Professor, we made sure to account for all information provided to us to ensure a complete report.

4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)
   Team: The actual VnV activities differed slightly from the original plan, primarily due to adjustments made during the development cycle. Some test cases had to be modified or expanded as we encountered new scenarios that were not initially anticipated, such as specific security vulnerabilities and data integrity concerns. The biggest change was in prioritizing additional integration tests over certain lower-priority unit tests, since we found that issues were more likely to arise in cross-module interactions. These changes occurred because real-world implementation often reveals gaps in planning, and some verification methods turned out to be less effective than expected. In future projects, I would anticipate these changes by leaving more flexibility in the VnV plan and incorporating iterative updates based on early testing feedback.