Reflection and Traceability Report on MES-ERP

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

1 Changes in Response to Feedback

1.1 Changes in Response to Usability Testing

The results concluded from the usability testing were thoroughly analyzed and taken into consideration when adding new changes to the final product. Various points of improvement were taken note of as well as numerous key changes to be made. These results can be seen in the Usability Testing document.

1.2 SRS and Hazard Analysis

1.2.1 Changes Made to the SRS

1. **Added diagrams**
   Why? To provide visual representations of complex processes, increase clarity, and stay consistent with the rest of the documentation standards.

2. **Fixed spelling**
   Why? To maintain professionalism, ensure correctness, and keep the document free of typographical errors.

3. **Added references**
   Why? To align with formal SRS practices, give credit to external sources where applicable, and enhance document credibility.

4. **Added Reflection**
   Why? To evaluate and discuss key decisions and modifications, ensuring transparency and consistency with reflective sections in similar documents.

5. **Looked over the missing context in part 7**
   Why? To fill gaps and ensure that all sections have the necessary background and details for clarity.

6. **Rationales were looked over but no citations were used (hence no citations were added)**
   Why? The rationales were internally based, and no external sources were needed, so citations were not applicable.

7. **Fixed table formatting**
   Why? To ensure uniform presentation across the document and improve readability and consistency.

8. **Added additional context to unclear sections**
   Why? To clarify ambiguities, make the requirements more precise, and maintain cohesion with the overall web application focus.

9. **Removed solution-based requirements**
   Why? To keep the SRS strictly requirement-focused, ensuring it does not prescribe specific implementations and remains consistent with standard SRS principles.

10. **Updated all requirements based on comments and feedback from rubrics**
    Why? To address identified issues, remain compliant with rubric standards, and ensure the SRS meets its intended quality and completeness.

1.2.2 Changes Made to the Hazard Analysis

The document has been thoroughly revised to address feedback from a rubric. Here are the key changes:

1.2.3 Document Structure Improvements

- Added Table of Contents, List of Tables, and List of Figures
- Improved document organization with appropriate page breaks
- Updated revision history table with current changes
- Added proper labels to Safety/Security Requirements (SSR-1, SSR-2, SSR-3)

1.2.4 Content Enhancements

1. **Introduction & Scope**:
   - Refined hazard definitions specifically for MES-ERP financial operations
   - Expanded scope to clearly include all system components (frontend, backend, database, auth)
   - Added clearer document roadmap

2. **System Boundaries & Components**:
    - Added technology specifics (Supabase/PostgreSQL, Next.js/React)
    - Enhanced hazard descriptions with concrete examples
    - Added new hazards (race conditions, XSS vulnerabilities, session hijacking)
3. **Critical Assumptions**:
    - Clarified existing assumptions
    - Added a fifth assumption about security of underlying infrastructure
4. **FMEA Implementation**:
    - Created a complete FMEA table with severity, occurrence, and detection ratings
    - Calculated Risk Priority Numbers (RPN)
    - Added specific mitigation strategies for each failure mode
    - Cross-referenced Safety/Security Requirements
5. **Safety Requirements**:
    - Added formal labels and improved descriptions
    - Enhanced rationales for implementation
6. **Roadmap**:
    - Improved immediate implementation items with technical specifics
    - Added cross-references to SSRs
    - Added new future implementation item (Security Penetration Testing)

The revisions make the document more precise, technically detailed, and better aligned with software engineering best practices for hazard analysis.

All changes were made to address the rubric feedback, with special attention to improving the technical specificity of hazard descriptions, creating a proper FMEA analysis, and ensuring consistent cross-referencing between identified hazards and safety requirements. The revisions significantly enhanced the document's precision and alignment with software engineering best practices for hazard analysis of financial systems.

1.3 Design and Design Documentation

1.4 VnV Plan and Report

2 Challenge Level and Extras

2.1 Challenge Level: General

The general challenge level comes from the integration of several complex components: a multi-group RBAC system, database interactions for real-time budget/request tracking, form handling with file uploads and OCR, automated notifications, and data visualization, all within a modern web framework. Ensuring data consistency, security, and a usable interface across different user roles presents a significant challenge. This must be done while maintaining stability and performance.

2.2 Extras

1. **Usability Testing** Conducted formal usability testing with 8 stakeholders (4 student leaders, 2 MES administrators, and 2 regular club members) to evaluate platform usability. Testing was performed remotely via screen sharing using a think-aloud protocol, with participants completing four core tasks: submitting a reimbursement request, tracking request status, approving requests (administrators only), and updating account information. Both quantitative metrics (task completion rates, time, ease-of-use ratings) and qualitative feedback were collected. The testing achieved an overall 96.9% task completion rate with an average satisfaction score of 4.3/5. Key improvements implemented based on testing results included enhanced OCR feedback, improved payment method interface clarity, and identification of future enhancements for form layout optimization.

2. **User Documentation** Created comprehensive user documentation in the form of written guides and video tutorials that guide end-users through key tasks like submitting requests, managing budgets (for admins), and navigating the platform. This documentation serves different user roles within the system and provides step-by-step instructions for all critical workflows.

3. **User Guide Page** Implemented a dedicated help section directly within the application to provide contextual assistance and guidance for users as they navigate through different features of the platform. This integrated support system helps users understand complex processes and reduces the learning curve for new users.

4. **Demo/Instructional Video** Developed a comprehensive video demonstration of the system's functionality that serves both as a promotional tool and an instructional guide for new users to understand the workflow and features of the MES-ERP platform. The video covers all major

user journeys and highlights the efficiency gains compared to the previous manual process.

3 Design Iteration (LO11 (PrototypeIterate))

4 Design Decisions (LO12)

Our design process aimed to create a robust, maintainable, and secure platform tailored to the MES's specific needs, while working within the constraints of a capstone project. Key decisions included:

- **Technology Stack (Next.js, TypeScript, Supabase):** The choice of Next.js and TypeScript was initially guided by supervisor constraints for consistency with other MES projects, as noted in the Development Plan. However, this stack proved advantageous. Next.js provided a powerful framework for both frontend (React) and backend (API Routes), simplifying development. TypeScript offered static typing, crucial for catching errors early in a complex application involving financial data and intricate permissions. Supabase was selected as the Backend-as-a-Service (BaaS) platform. This significantly reduced the backend development overhead (database setup, authentication management, storage) allowing the team to focus on core application logic, fitting well within the project's time and budget constraints. The trade-off was a dependency on Supabase's infrastructure and potential limitations of its free/pro tiers.

- **Modular Architecture (Information Hiding - MG/MIS):** We adopted the principles of information hiding, decomposing the system into Hardware-Hiding (Database Interaction), Behaviour-Hiding (Core Features like Expense Submission, Approval Workflow), and Software Decision (Validation, GUI) modules, as documented in the Module Guide (MG) and specified in the Module Interface Specification (MIS). This decision was driven by the need for maintainability and adaptability. Anticipated changes, such as supporting new input formats or notification methods, were encapsulated as module secrets, aiming to localize the impact of future modifications. This structured approach, while requiring upfront design effort, was deemed essential for a system intended for long-term use by the MES.

- **Role-Based Access Control (RBAC) Implementation:** The decision to implement a relatively complex RBAC system, supporting users belonging to multiple groups with potentially different roles in each, stemmed directly from the MES's organizational structure (many clubs with varying leadership roles). A simpler model (e.g., one user, one role) would not have accurately reflected the real-world requirements. We used junction tables (`user_roles`, `group_roles`, `role_permissions`) in the database schema, a standard approach for modeling these many-to-many relationships. This design provided the necessary flexibility but increased implementation complexity, particularly in permission checking logic within the middleware, API routes, and frontend components (`usePermissions` hook). This complexity was a necessary trade-off for accurately modeling the domain.

- **Database Schema Design:** The schema, documented in `src/docs/DATABASE.md`, prioritized clear relationships between entities (users, groups, roles, requests, budgets). The use of junction tables for RBAC was crucial. The `operating_budget_lines` table provided a flexible structure for managing detailed budgets per group, separating allocated amounts from actual expenses derived from `payment_requests`. This separation supported accurate tracking and reporting.
- **API Route Structure:** We utilized Next.js API routes for specific backend operations like sending notifications (`send-email-notif`, `send-sms-notif`) and updating user roles/groups (`update-role`, `update-group`). This kept backend logic separate from the frontend rendering, promoting cleaner code and easier testing of backend functionality.
- **External Service Integration (SendGrid, Tesseract.js):** Integrating external services like SendGrid for email notifications and Tesseract.js for OCR simplified development compared to building these features from scratch. This aligned with the project's constraints, allowing focus on core MES-specific workflows. The limitation is reliance on these services' availability and potential costs.
- **Assumption/Limitation Impact:** The assumption of reliable internet meant little effort was put into offline capabilities. The budget constraint influenced the choice of Supabase (generous free tier) and limited the scope of "stretch goals". The capstone timeline necessitated prioritizing core features over extensive performance optimization or advanced features like ML-based categorization initially. The design explicitly tried to accommodate anticipated changes (like reporting formats) by isolating related logic in specific modules.

Overall, design decisions prioritized fulfilling core MES requirements, ensuring security and data integrity for financial operations, and promoting maintainability, while leveraging existing tools and services to meet project constraints.

5 Economic Considerations (LO23)

As a capstone project developed specifically for the McMaster Engineering Society (MES), the initial "market" is internal. However, the underlying problem – managing finances for multiple student clubs within a larger organization – is common across universities.

- **Market Potential:**

  - **Internal (MES):** The immediate user base consists of roughly 60 MES-affiliated clubs, MES executives, and financial administrators. Success here relies on adoption and usability.
  - **McMaster Expansion:** Other student societies within McMaster (e.g., MSU clubs, Faculty societies) face similar challenges and could be potential users if the platform is adaptable.
  - **External Universities:** Student unions/societies at other universities represent a larger potential market, though adaptation to different financial policies would be necessary.

- **Marketing and User Attraction:**
  - **MES Internal:** Promotion via MES communication channels, training sessions for club treasurers, and endorsements from MES executives. Ease of use and clear benefits over the old system (faster reimbursements, better tracking) are key selling points.
  - **External/Open Source:** If generalized or open-sourced, marketing could involve presentations at student government conferences, articles in university tech forums, presence on GitHub, and potentially partnerships with university IT departments. Attracting users would depend on demonstrating clear advantages over generic tools (like spreadsheets) or commercial alternatives, ease of deployment, and good documentation.

- **Production and Ongoing Costs:**
  - **Initial Development Cost:** Primarily student labor within the capstone structure. The stated project budget was $750, likely covering minor expenses (perhaps specific software tools or small service tiers). Estimating a commercial equivalent would involve multiplying development hours (estimated 400-500 in the VnV Report) by a junior/intermediate developer rate, resulting in tens of thousands of dollars.
  - **Hosting & Services:** Supabase offers a free tier, but scaling might require paid plans ($25+/month). Vercel (for Next.js hosting) also has free/pro tiers. SendGrid and Twilio have usage-based pricing, likely low initially but scaling with notification volume. Domain name registration (around $15/year). Total ongoing costs could range from $0 (on free tiers with low usage) to $50-$100+/month depending on scale and features used.
  - **Maintenance:** Requires ongoing effort for bug fixes, security updates, adapting to policy changes, and user support. This could be handled by future student volunteers, MES IT staff, or potentially paid contractors, representing a significant hidden cost.

- **Pricing Model (If Commercialized/Generalized):**

- **Subscription:** A tiered annual subscription based on the number of clubs or users seems most plausible for university societies (e.g., \$500-\$2000/year per society).
- **Open Source + Support:** Offer the core platform as open source, charging for setup assistance, hosting, custom modifications, or premium support contracts.

- **Sales Estimate / Break-Even:** Assuming ongoing costs of \$600/year (\$50/month) and needing to recoup an estimated \$30,000 commercial development cost over 3 years (\$10,000/year), the platform would need to generate \$10,600 annually. At a \$1000/year subscription, this requires roughly 11 paying student societies. Given the niche market, achieving significant profit seems challenging without broad adoption or a very lean maintenance model.
- **Potential Users:** Initially 60 MES clubs + MES staff. Potentially hundreds of clubs within McMaster. Thousands across Canadian universities.

In conclusion, while MES-ERP addresses a real need for MES, its direct commercial viability is uncertain due to the niche market and the likely need for customization for other institutions. An open-source model with optional paid support or focusing on adoption within McMaster might be more realistic paths for broader impact beyond the initial MES deployment. The primary economic benefit for MES is the significant saving in administrative time and improved financial control compared to the previous manual system.

6 Reflection on Project Management (LO24)

6.1 How Does Your Project Management Compare to Your Development Plan

Our project management generally followed the Development Plan, but with some natural adaptations:

- **Team Meeting Plan:** We adhered to the plan for regular team meetings (twice weekly initially, likely adjusting frequency later) and scheduled supervisor meetings (though one was missed by a member). Attendance, as tracked in contribution reports, was generally high for team meetings.
- **Team Communication Plan:** Discord was used as the primary informal communication channel, as planned. GitHub Issues were used for tracking tasks and feedback, although perhaps not as consistently for *all* communication or discussion as initially intended. Merge conflicts and urgent issues were likely handled via Discord/calls.
- **Team Member Roles:** Initial roles (Meeting Chair, Notetaker, Reviewer, Leader, Helper) were defined. In practice, roles in small student

teams often become more fluid, with members contributing across different areas based on need and expertise. The commit history suggests varying levels of contribution, which might indicate roles shifted or workloads weren't perfectly balanced despite the initial plan. The plan did include monthly reviews to adjust roles, but it's unclear if these formal reviews occurred.

- **Workflow Plan (Git, Issues, CI/CD):**
  - **Git Usage:** The use of branches for features/bugfixes and Pull Requests (PRs) with peer review seems to have been implemented, based on standard practice and the presence of multiple contributors. Commit messages aimed for descriptiveness (mentioning Conventional Commits). Tags for releases may or may not have been consistently used.
  - **Issue Management:** GitHub Issues were used, and templates were defined. The team contribution reports mention issues authored and assigned, indicating usage. The effectiveness depended on issue granularity and consistent tracking/closing.
  - **CI/CD Implementation:** The planned CI/CD pipeline (Lint, Format, PDF Build, Unit Tests via GitHub Actions) was successfully implemented, as evidenced by the workflow files. This automated quality checks and documentation builds. Test coverage goals (90%) were set, but the VnV report indicates actual automated coverage was lower, suggesting challenges in achieving this target.
- **Technology:** The core technology stack (Next.js, TypeScript, Supabase, Shadcn UI, Tailwind, Jest) defined in the Development Plan was indeed used in the implementation.

6.2 What Went Well?

- **Clear Initial Planning:** The Development Plan provided a good roadmap for roles, communication, and workflow.
- **Technology Stack Adoption:** The team successfully implemented the application using the chosen modern web technologies.
- **Version Control:** Effective use of Git and GitHub facilitated collaboration, code sharing, and tracking changes.
- **CI/CD Automation:** Implementing workflows for linting, formatting, testing, and PDF building automated crucial quality checks and saved manual effort.
- **Communication Channels:** Utilizing Discord for quick communication and GitHub Issues for task tracking generally worked well.

- **Regular Meetings:** Consistent team meetings helped maintain alignment and address roadblocks.

6.3 What Went Wrong?

- **Achieving Test Coverage Goals:** The ambitious 90% automated test coverage target was not met, indicating challenges in writing comprehensive tests alongside feature development, possibly due to time constraints or complexity.
- **Initial Setup Friction:** Reflections mentioned initial difficulties with setting up development environments (e.g., LaTeX), causing early delays.
- **Strict Role Adherence:** Maintaining the initially defined, distinct roles might have been impractical; tasks likely required more cross-functional collaboration than the roles implied.
- **Meeting Scheduling:** Some scheduling conflicts occurred, leading to missed supervisor meetings or team building activities.
- **Issue Tracker Discipline:** While used, the consistency of using GitHub Issues for *all* task tracking and discussion might have varied.

6.4 What Would you Do Differently Next Time?

- **Proactive Workload Management:** Implement more explicit task breakdown and assignment during sprint planning. Use pair programming or focused work sessions to help members who might be falling behind or need assistance. Regularly check in on individual progress and offer support.
- **Integrate Testing Earlier and Continuously:** Write unit tests concurrently with feature development rather than potentially leaving it towards the end. Set more realistic, incremental coverage goals throughout the project lifecycle.
- **Improve Onboarding/Setup:** Create a more streamlined setup guide or utilize tools like Dev Containers to minimize initial environment configuration issues.
- **Flexible Roles:** Define roles more flexibly from the start, acknowledging that members will likely contribute across different areas. Emphasize shared responsibility.
- **Enhanced Issue Tracking:** Encourage more disciplined use of the issue tracker for all tasks, discussions, and decisions to improve traceability and transparency.
- **Better Time Estimation:** Factor in more buffer time for complex features (like RBAC) and for writing thorough tests.

7 Reflection on Capstone

Throughout the duration of the capstone, we gained valuable experience in several key areas. We improved our project management skills by organizing and prioritizing tasks effectively, ensuring deadlines were met, and coordinating with team members. Our coding abilities expanded as we learned to apply software engineering principles to real-world problems. Teamwork and collaboration became critical, as we frequently had to communicate with others, delegate responsibilities, and integrate diverse skill sets into a unified product. Additionally, reflecting on each stage of the project helped us develop critical thinking skills, making it easier to identify and resolve issues early on. Finally, we grew more confident in working with external teams, stakeholders, and mentors, which taught us to remain consistent with the established design processes and maintain clear, structured communication.

7.1 Which Courses Were Relevant

Several courses directly informed our approach to this project. For instance, Databases proved essential for designing and maintaining efficient data storage, while Software Architecture and Software Design helped us structure the system in a modular and scalable way. Software Testing guided us in writing test suites and performing systematic debugging, ensuring reliable functionality. Computer Interfaces provided insights into user experience considerations, which was valuable for front-end design and usability testing. Finally, the Engineering Design courses taught us to follow a structured design process, from requirement gathering to implementation and validation.

7.2 Knowledge/Skills Outside of Courses

Beyond the formal coursework, we had to develop additional project management skills such as scheduling, risk assessment, and resource allocation. We also needed to improve our web development capabilities, particularly in frameworks and libraries not covered in class. Another area of growth was image processing and algorithms, including crafting and optimizing regular expressions for data validation and analysis. Moreover, we honed our soft skills, from effective communication and negotiation to conflict resolution and team coordination, all of which proved vital for successful collaboration and stakeholder engagement throughout the project.