# TM470 TMA01 Answers Refined and improved

## Revised Project Scope (Prototype Approach)

This project focuses on developing a proof-of-concept prototype to demonstrate the benefits of digitalising routine maintenance inspections within the company. The aim is to explore the feasibility of replacing the current paper-based system with a digital solution, improving efficiency, record-keeping, and compliance.  
  
The prototype will be designed as a self-contained system with its own internal database, hosted on-premises, in line with the company's security policies, but it will not directly integrate with the company's existing VB.NET-based system. Instead, it will serve as a demonstration model, showcasing how a digital system could streamline maintenance processes and reduce administrative overheads.  
  
As this is a prototype, the focus is on usability, process improvement, and security considerations, rather than delivering a fully integrated enterprise solution. The IT department may later develop their own production-ready version based on the insights and findings from this project.  
  
To ensure the prototype meets real-world needs, user feedback from engineers and managers will be incorporated throughout development. The final evaluation will assess efficiency gains, ease of use, and potential challenges that may arise in a future full-scale implementation.  
  
While the technical approach remains flexible, the prototype will likely be a web or hybrid application, ensuring accessibility across different devices used within the company. However, all data storage and processing will remain internal and separate from cloud-based services, adhering to the company’s security policies.  
  
This approach allows for academic exploration and practical validation without being restricted by the company’s IT policies for production environments. The project will conclude with recommendations for future enhancements, should the company wish to proceed with a full deployment.

## Account of Related Literature

### Database Systems: Relational vs. NoSQL Approaches

Harrington (2016) provides a foundational understanding of relational database design, covering key concepts such as normalisation, entity relationships, and data integrity. Relational databases are widely used for structured data storage, particularly in environments where data consistency and referential integrity are priorities. This is particularly relevant for the maintenance application, as inspection records must be stored securely with a clear structure linking engineers, equipment, and historical maintenance logs.  
  
However, as noted by Stonebraker (2020), not all data models fit relational structures. The rise of NoSQL databases, such as MongoDB and CouchDB, offers flexibility in handling semi-structured data, which could be beneficial if inspection forms require frequent schema modifications or unstructured attachments (e.g., images, notes). Furthermore, graph databases like Neo4j provide efficient ways to model relationships dynamically, which could be useful if the system needs to track equipment dependencies and fault histories (Angles & Gutierrez, 2008).  
  
Given the company’s strict on-premises data policy, an SQL-based database (such as PostgreSQL or Microsoft SQL Server) is likely more appropriate, as these solutions align with existing infrastructure and offer robust security and access control mechanisms (Elmasri & Navathe, 2021). However, further evaluation is needed to determine whether hybrid approaches (e.g., combining SQL for structured data and NoSQL for form attachments) could offer a more adaptable solution.

### Application Development: Native vs. Web-Based Solutions

Initial research considered cross-platform mobile development frameworks, including React Native, Flutter, and Xamarin (Windmill, 2020). These technologies allow developers to build mobile applications that run on both Android and iOS from a single codebase, making them attractive for projects requiring broad device compatibility. However, given recent findings that the company will not permit external applications to integrate with its internal database, a web-based or hybrid approach may be more appropriate.  
  
As highlighted by Resig (2022), Progressive Web Applications (PWAs) offer a lightweight and secure alternative to native mobile apps while allowing access to device capabilities such as offline storage and push notifications. Blazor (Microsoft, 2021) provides another option for developing server-hosted web applications using C# and .NET technologies, which would align better with the company’s existing VB.NET-based ecosystem.  
  
An analysis by Shneiderman et al. (2020) on usability in enterprise applications suggests that web-based solutions often lead to faster adoption and easier maintenance than platform-specific mobile apps, particularly in companies with strict IT governance policies. Given that engineers will access the system through company-provided tablets via a remote desktop or web interface, a Blazor-based or PWA solution may be preferable to a fully native mobile app.

### System Deployment Considerations

Since the company enforces on-premises data storage and prohibits cloud-based solutions, the deployment model must adhere to strict security and compliance measures. AWS, Firebase, and other cloud platforms were initially considered for backend hosting but were ruled out due to these security constraints.  
  
Tanenbaum & Wetherall (2021) highlight that on-premises hosting ensures full control over data security and network access but introduces higher maintenance overheads compared to cloud solutions. The proposed system must therefore be designed for local server deployment while maintaining scalability and performance comparable to cloud-hosted applications.  
  
Research from Davis & Patterson (2020) suggests that containerisation (e.g., Docker, Kubernetes) can improve deployment efficiency in on-premises environments, offering advantages such as faster updates, modular scalability, and enhanced security isolation. This could be a viable approach for ensuring the system remains manageable while meeting IT security policies.

### Conclusion and Next Steps

This literature review highlights the importance of carefully evaluating database structures, application frameworks, and deployment strategies to ensure the maintenance inspection prototype aligns with user needs and company restrictions. While relational databases remain the most viable storage option, alternative models such as NoSQL for unstructured data should not be dismissed outright. Similarly, while mobile development frameworks were initially considered, a shift toward web-based approaches may provide a more practical and compliant solution.  
  
Future research will focus on validating these choices through prototyping and stakeholder feedback, ensuring that the final approach remains technically feasible, user-friendly, and aligned with corporate IT governance standards.

## Resources and Planning

The success of this project depends on several key resources, including hardware, software, technical expertise, and software engineering skills. These resources ensure the prototype is feasible, user-friendly, and aligns with the company’s IT policies.

### 1. Hardware Resources

|  |  |  |
| --- | --- | --- |
| Resource | Purpose | Justification |
| Tablets (Android & iOS) | Testing application usability across company devices | Ensures compatibility with maintenance team’s Android tablets and production team’s iPads. |
| Development Machine | Running development environments (React Native, .NET, or hybrid frameworks) | Required for writing, testing, and debugging software efficiently. |
| Server Infrastructure | Hosting application database and backend services | The system must be on-premises due to security restrictions, requiring access to company servers. |

### 2. Software Resources

|  |  |  |
| --- | --- | --- |
| Resource | Purpose | Justification |
| Visual Studio Code | Development environment for coding | Supports JavaScript-based frameworks (React, Node.js) and .NET integration. |
| PostgreSQL / MySQL | Backend database for storing inspection records | SQL databases offer structured storage, aligning with IT security policies. |
| Git & GitHub | Version control and collaboration | Enables structured project development, tracking changes efficiently. |
| Figma / Adobe XD | UI/UX prototyping | Supports usability testing before implementation. |

### 3. Software Engineering Skills Needed

|  |  |  |
| --- | --- | --- |
| Skill | Purpose | Why It’s Important? |
| User Requirements Elicitation | Collect feedback from maintenance engineers | Ensures the prototype aligns with real-world workflows and user expectations. |
| Usability Testing & Evaluation | Assess ease of use and efficiency | Determines if the digital system improves upon paper-based processes. |
| Prototyping & Proof-of-Concept Development | Create a test version before final deployment | Ensures feasibility before IT invests in full development. |
| Security & Compliance | Implement authentication and role-based access | Aligns with IT policies for data protection and restricted database access. |

### Discussion: Why These Resources Contribute to Project Success

The combination of hardware, software, and development tools ensures that the prototype can be built, tested, and evaluated effectively. By aligning backend storage with the company's security policies and using tablets for testing, the project remains technically feasible within the company’s ecosystem.

A prototype is only successful if it meets user needs. Research from Preece et al. (2019) highlights that usability testing and user feedback are critical for refining digital solutions. By conducting structured requirements gathering and evaluating the prototype with engineers, the project will ensure a practical, user-friendly solution rather than just a technical implementation.

Using an iterative development approach, this project will test and refine features before finalising a full-scale solution. Rogers et al. (2020) emphasise the importance of prototyping in software development to reduce risks and ensure that system requirements are well-defined before committing to large-scale deployment.

### Risk Assessment and Mitigation Strategies

Effective risk management is crucial for ensuring software projects remain viable and adaptable. Sommerville (2020) highlights that early risk identification reduces delays, scope creep, and resource misallocation. This is particularly relevant for this project, where technical constraints, compliance requirements, and stakeholder engagement are critical to success.

### Key Risks and Mitigation Strategies

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Risk | Likelihood | Impact | Justification | Mitigation Strategy |
| Lack of experience with React Native | Medium | High | As a first-time React Native developer, debugging and performance optimisation may take longer than expected. | Allocate structured learning time, complete online tutorials, and seek mentorship from experienced developers. |
| Integration challenges with the company’s existing software | High | High | The company restricts external applications, requiring a standalone system with strict security policies. | Engage with a senior software engineer early to clarify technical constraints and explore integration alternatives. |
| Lack of access to testing devices | Medium | Medium | Testing on company-provided tablets is required, but access may be limited. | Use emulators for initial development and request company devices for final testing. |
| Data security concerns and compliance | High | High | The company enforces strict on-premises storage and role-based access control policies. | Ensure compliance with security protocols, use encrypted data storage, and implement role-based authentication. |
| Scope Creep (Uncontrolled feature expansion) | Medium | High | Without strict control, additional feature requests could delay project completion. | Define a strict project scope, prioritise core functionalities, and document additional feature requests for future iterations. |
| Time constraints due to workload | High | High | Managing university deadlines and work commitments simultaneously may lead to rushed development phases. | Establish clear project milestones, allocate buffer time for unexpected delays, and follow an Agile approach. |
| Lack of user requirements elicitation | High | High | Without stakeholder engagement, the prototype may not meet real-world needs. | Conduct interviews with engineers & managers, use early user testing, and document feedback iteratively. |
| Lack of exploration of alternative technical tools | Medium | High | Selecting the wrong development framework could lead to inefficiencies or compatibility issues. | Research and compare different development approaches, evaluating them based on feasibility and company IT policies. |
| Lack of evaluation criteria for the technical solution | Medium | High | Without clear evaluation metrics, it will be difficult to assess the prototype’s success. | Define technical success criteria (e.g., usability, security compliance, efficiency improvements) before development begins. |
| Changes in company IT policy | Medium | High | If IT policies change mid-project, the chosen technology stack may no longer be valid. | Maintain regular communication with IT managers, document alternative approaches, and ensure flexibility in development. |

### Discussion: The Importance of Risk Management in Software Prototyping

Security risks must be carefully managed due to company policies restricting external applications and requiring strict data controls. ISO 27001 security standards recommend encryption, multi-factor authentication, and controlled access to mitigate compliance-related risks (Whitman & Mattord, 2020).

Additionally, Agile methodologies suggest mitigating scope creep by using a strict prioritisation framework (Beck et al., 2001). Regular stakeholder feedback cycles will help ensure the prototype remains aligned with user needs while keeping development focused on core requirements.

### Project Lifecycle Selection

Selecting an appropriate Software Development Lifecycle (SDLC) model is crucial for ensuring that the project is delivered efficiently and effectively. Different lifecycle models provide varying degrees of structure and flexibility, impacting the way requirements, development, and testing are approached.  
  
The Iterative Waterfall Model was chosen for this project due to its balance between structured development and iterative refinements. However, alternative models, such as Agile and the Spiral Model, were also considered to ensure the most suitable methodology is selected.

### Comparison of SDLC Models

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Characteristics | Advantages | Disadvantages |
| Iterative Waterfall | Follows a structured phase-by-phase approach but allows refinements at each stage. | - Clear milestones and documentation. - Easier to track progress. - Works well for projects requiring a structured database-first approach. | - Less flexible for changing user requirements. - Requires detailed planning upfront. |
| Agile (Scrum) | Iterative and incremental approach with frequent user feedback. | - Highly flexible, allows continuous requirement refinement. - Encourages user involvement. | - May not align with strict company IT policies. - More difficult to apply when security and compliance constraints limit frequent deployment. |
| Spiral Model | Focuses on iterative risk assessment and phased refinements. | - Ideal for high-risk projects where uncertainties exist. - Allows controlled iterations with structured feedback. | - Can be complex and time-consuming. - Requires risk assessment expertise. |

### Justification for Choosing Iterative Waterfall

1. Structured Development with Iterative Refinements   
 - This approach allows a clear, step-by-step process, ensuring that foundational elements, such as database design, are established before moving on to UI development.   
 - While the traditional Waterfall model is rigid, the Iterative Waterfall approach introduces incremental flexibility, allowing refinements at each stage.  
  
2. Progress Tracking and Project Management   
 - This model aligns well with academic project deadlines, as it allows structured progress tracking through distinct milestones.   
 - Agile would introduce frequent requirement changes, making it difficult to meet structured university submission requirements.  
  
3. Security and Compliance Considerations   
 - Since the company has strict IT policies (requiring on-premises storage and controlled application deployment), Agile’s frequent deployments and continuous updates may not be feasible.   
 - Iterative Waterfall ensures security and compliance are addressed early in development rather than adjusted retroactively.  
  
4. Handling Uncertain Requirements   
 - While Agile is traditionally better for evolving requirements, the Iterative Waterfall approach can still accommodate adjustments through review and refinement stages.   
 - This means that once user requirements are fully established, refinements can still be made within the structured framework.

### Conclusion

The Iterative Waterfall Model was selected because it provides a structured approach while allowing controlled iterations. It balances clear project milestones with incremental flexibility, making it ideal for a proof-of-concept prototype where security and structured development are key concerns.  
  
However, future considerations may involve transitioning to a more Agile-like approach once the IT department takes over development, particularly if continuous improvements are required after initial implementation.

### Project Schedule and Milestones

The following schedule outlines key project milestones and expected completion timeframes. This refined timeline ensures a structured approach while allowing flexibility for review and refinements at each stage.

### Refined Project Schedule & Milestones

|  |  |  |  |
| --- | --- | --- | --- |
| Phase | Tasks | Deliverables | Timeline |
| Phase 1: Analysis & Requirements Gathering | Conduct stakeholder interviews, define user needs, analyse database structure, assess technical requirements for security & photo uploads. | Stakeholder meeting notes, survey results, database draft, user requirements document. | Weeks 1-4 |
| Phase 2: Design | Design UI/UX, create database schema, define security measures, structure form workflows. | UI wireframes, finalised database schema, initial system architecture. | Weeks 5-7 |
| Phase 3: Implementation (Development & Coding) | Develop authentication, implement main screens, build form structures, integrate photo uploads, implement security features. | Functional prototype with authentication, form submission, photo upload. | Weeks 7-13 |
| Phase 4: Testing & Refinements | Conduct unit testing, security testing, gather user feedback, refine features, and fix bugs. | Testing reports, user feedback analysis, bug fixes applied. | Weeks 13-15 |
| Phase 5: Final Adjustments & Documentation | Implement necessary improvements based on evaluation, prepare final documentation and reports. | Final prototype, project documentation, user guide, evaluation report. | Weeks 15-17 |

## Project Work

This project builds on principles from TM354 Software Engineering and TM352 Web, Mobile, and Cloud Technologies, integrating structured software development methodologies with mobile application design to digitize routine maintenance processes.

### Current Maintenance Process & Challenges

Maintenance engineers currently complete safety inspections using paper forms, which must be scanned and manually stored for compliance with HSE 1988 regulations. This introduces several inefficiencies:

* Data Loss & Misplacement – paper records can be lost, affecting compliance.
* Manual Processing Delays – forms require scanning, uploading, and filing, consuming management time.
* Lack of Real-Time Access – engineers cannot easily retrieve historical maintenance records, affecting decision-making.

### Application of TM354 & TM352 Concepts

**Software Engineering Principles (TM354):**

- Use Case Diagrams – Defining system interactions between engineers, managers, and the application.

- Functional & Non-Functional Requirements – Ensuring the application meets performance, security, and usability standards.

- UML Diagrams – Using activity and sequence diagrams to model workflow automation for maintenance inspections.

- Test-Driven Development (TDD) – Writing test cases before implementation, ensuring software quality.

- Enterprise Architectures – Evaluating architectural patterns to design a scalable and maintainable system.

**Web, Mobile & Cloud Development (TM352):**

- React Native for Cross-Platform Compatibility – Allowing a single codebase for Android and iOS tablets.

- Secure Authentication & Data Storage – Implementing encryption and access control for sensitive maintenance records.

- Cloud Synchronisation (Limited by Company IT Policy) – Exploring secure on-premises alternatives to manage data storage.

### Proposed Application Workflow

Main Screen: Engineers log in and choose between Facility Checks or Machine Monthly Safety Checks.

Facility Checks: Each inspection type has structured input fields to ensure compliance:  
- Bio-Waste Treatment – Logs foliage conditions, odor levels, and fluid levels to prevent contamination risks.  
- Emergency Lights – Requires zone, light ID, and test results for compliance with fire safety regulations.

Machine Monthly Safety Checks: Engineers select a machine type (e.g., Die Cutter, Printer, Finisher).  
Dynamic Checklists are automatically assigned, including:  
- Guard & Light Guard Tests – Ensures machines comply with operator safety standards.  
- Emergency Stop Function Tests – Verifies that emergency shutdown features work as intended.

By structuring the application this way, the system ensures compliance, reduces administrative workload, and enhances data accessibility for engineers and management.

## Legal, Social, Ethical, and Professional Issues & EDI

This project involves several Legal, Social, Ethical, and Professional Issues (LSEPI) and Equality, Diversity, and Inclusion (EDI) considerations that impact its design, implementation, and adoption. The following analysis outlines key considerations and how they will be addressed.

### 1. Legal & Data Protection Considerations

#### Data Protection & GDPR Compliance

Since the system stores inspection records and staff-submitted reports, it must comply with General Data Protection Regulation (GDPR) requirements, affecting:

* Design: The database must be structured to limit data collection to essential information only.
* Security Measures: Role-based access and encryption must prevent unauthorized access to sensitive records.
* Deployment: A data retention policy must be in place, ensuring data is not stored indefinitely.

Impact on the Project:

Not considering GDPR could lead to non-compliance leading to legal issues, data breaches, or loss of trust in the system.

Mitigation Strategy:

- Implement role-based access. For example, engineers can access their reports, but managers have broader access.  
- Encrypt sensitive data to prevent unauthorized access.  
- Define data retention and deletion policies following GDPR guidelines.

### 2. Social & Ethical Considerations

#### Stakeholder Interests & Ethics

The maintenance team and Plant and Services Manager are direct stakeholders. To ensure ethical implementation, the system must:

* Be fit for purpose - intuitive and easy to adopt for non-technical users.
* Minimise disruption, aligning the digital workflow with existing maintenance practices.
* Ensure transparency, providing clear documentation and training on how data is stored and accessed.

Not considering stakeholder interests could result with system not being implemented, making the digital transition unsuccessful.

Mitigation Strategy:   
- Conduct user interviews and usability testing during development.  
- Provide training sessions and quick-reference guides.

#### Job Impact & Change Management

Digitising paper forms will change workflows and could impact:  
- Workload distribution: The project could lead to overhead reduce for managers but would require training for engineers.  
- Resistance to change: Engineers accustomed to paper forms may hesitate to adopt digital systems.  
  
Change management may be difficult for some engineers, not considering the job impact could result with engineers bypassing the system and continuing to use paper forms.

Mitigation Strategy:  
- Run user tests before full implementation.  
- Offer technical support and step-by-step onboarding. Since my project would form the prototype for a full-scale enterprise application further training could be later provided by the IT department.

### 3. Equality, Diversity, and Inclusion (EDI) Considerations

#### Accessibility & Usability

The application must be inclusive and accommodate engineers with different levels of technical proficiency. Key accessibility features include:  
- Simple and intuitive navigation—reducing cognitive load.  
- Readable text & high-contrast designs supporting users with visual impairments.  
- Offline functionality—ensuring engineers can complete inspections even in areas without Wi-Fi.

Although there is no disabled team members within the maintenance team, if accessibility and usability factors are not considered the application may exclude certain users, leading to inconsistent system use.

Mitigation Strategy:  
- Follow WCAG (Web Content Accessibility Guidelines) in UI design.  
- Conduct user testing with diverse engineers to identify usability issues.

#### Inclusive Language & Design

- The interface must be language-inclusive, avoiding overly technical terms.  
- The system should be designed with input from a diverse user base, ensuring it is user-friendly across different demographic groups.

Confusing terminology could discourage users, reducing application adoption.

Mitigation Strategy:  
- Use plain, intuitive language in buttons and instructions.  
- Conduct user feedback sessions to refine text and UI layout.

### 4. Ethical Approval & Compliance

To address ethical concerns, I have:  
- Completed a Project Ethics Questionnaire (approved by my tutor).  
- Assessed data privacy, user consent, and accessibility challenges.  
- Included ethical approval documentation in Appendix 1.

## Personal development, review and reflection

### What have I done this week?

I have conducted meetings with several stakeholders, including **maintenance engineers, managers, and an IT Manager**, to refine the requirements for my project. These discussions have provided **valuable insights** that shaped both the **technical feasibility and alignment with company policies**.

Key insights from stakeholder meetings:

- User Preferences: Engineers preferred a simple checklist-style form rather than free-text fields for inspections. Managers requested a dashboard summary of completed reports rather than raw data exports.

- Workflow Adjustments: Engineers emphasized the need for offline functionality, as some inspection areas like Plant Room or Welding Bay have poor Wi-Fi coverage.

Image uploads should allow multiple photos per report rather than a single image attachment.

- Security & Access Considerations: Managers stressed the importance of role-based permissions, restricting access to completed reports based on user roles.

Key insights from Technical Discussions with the IT Manager

During technical discussions with the former IT Manager, it became clear that the company is enforcing strict security and operational policies regarding software integration. The IT department is focused on standardizing all applications under the company’s internal software system. As a result, standalone applications like mine that would use the server data would not be adopted in their current form (React Native or Flutter).

The company has made a strategic decision not to allow fragmented applications but instead integrate all tools with internal software system.

The IT Manager highlighted that my proposed solution was using:

* A **separate database structure** that does not align with internal sofware data model.
* A **different interface** from company-wide applications, affecting usability and supportability.
* An **incompatible operational architecture**, making maintenance and long-term support difficult.

While I have flexibility in designing my prototype as a web or mobile app for academic purposes, **for company-wide adoption, the application would need significant modifications** to comply with company IT policies.

### What Went Well & Why?

The meetings were successful because:

- Stakeholders were enthusiastic about moving from paper-based inspections to a digital system. They provided detailed feedback, making refinements easier.

- Discussions helped finalise key functional requirements, such as checklist structures, offline access, and security levels.

- The insights gained enhanced the project scope and feasibility, ensuring alignment with user needs.

### What Didn’t Go Well & How to Improve

I underestimated the workload of balancing TM470, TM352, my job, child care, and job searching. Managing time effectively has been a challenge.

- Going forward I must allocate fixed weekly study hours (e.g., Monday & Thursday mornings/evenings for TM470, Tuesdays, Wednesdays and Fridays for TM352).

- Additionally, I could use task breakdown techniques (e.g., Trello or Notion boards) to track progress.

- I should extend analysis and design phases slightly to allow more time for requirements refinement especially considering new findings about the company’s systems integration future plans.

The above strategies will help me stay on track and manage my workload effectively.

### What Do I Plan to Do Next Week?

Next week, I will:

- Organize insights from stakeholder meetings into a structured requirements document. Validate requirements with a few sample users to confirm accuracy. Have a discussion with one of my colleagues who is responsible for completing most of the facility checks.

- Evaluate whether a relational database (SQL) or an alternative model (NoSQL, document-based) best fits the project. Compare relational vs. non-relational storage based on:  
 - Data structure: Are form records best stored in structured tables, or would a more flexible schema be beneficial?  
 - Scalability: How much data growth is expected over time?  
 - Query complexity: Will queries require complex joins and relationships, or is a simpler retrieval model better?

- Identify the best way to translate form fields into database entities. Look at existing database models for maintenance record systems.