Deliverable 1

Msc Computer science

Software Analysis and Design

Birmingham City University

**Case Study: MSc Supervision Management System (MSMS)**

25105436

Table of Contents

[1.1 Phases of the Software engineering life cycle 3](#_Toc194002657)

[1.1.1 Requirements Analysis 3](#_Toc194002658)

[1.1.1.1 Requirements Elicitation 4](#_Toc194002659)

[a. Requirements discovery: 5](#_Toc194002660)

[c. Requirements prioritization and negotiation 5](#_Toc194002661)

[d. Requirements specification 5](#_Toc194002662)

[1.1.1.2. Requirements Analysis 5](#_Toc194002663)

[1.1.1.3. Requirements Validation 5](#_Toc194002664)

[1.1.1.4. Requirements Management 5](#_Toc194002665)

[1.1.2 System Design 6](#_Toc194002666)

[1.1.3 Implementation 7](#_Toc194002667)

[1.1.4 Testing 7](#_Toc194002668)

[1.1.5 Deployment 8](#_Toc194002669)

[1.1.6 Maintenance and Support 8](#_Toc194002670)

[1.2 Software engineering methodologies 8](#_Toc194002671)

[1.2.1 V-model 8](#_Toc194002672)

[1.2.2 Prototype model 9](#_Toc194002673)

[1.2.3 The waterfall model 10](#_Toc194002674)

[1.2.4 The agile methodology 12](#_Toc194002675)

[1.2.5 The spiral model 13](#_Toc194002676)

[1.2.6 The Scrum Model 13](#_Toc194002677)

[1.2.7Kanban 14](#_Toc194002678)

[1.2.8 Lean Software Development 15](#_Toc194002679)

[1.2.9 RAD (Rapid Application Development) 17](#_Toc194002680)

[1.2.10 DevOps 17](#_Toc194002681)

[1.3 Technological Tools for Each Life Cycle Phase in MSMS Development 18](#_Toc194002682)

[1. Planning Phase 18](#_Toc194002683)

[3. System Design. 19](#_Toc194002684)

[5. Testing Phase. 19](#_Toc194002685)

[6. Deployment Phase 19](#_Toc194002686)

[1.4 Ethical and Legal Issues in System Analysis and Design 19](#_Toc194002687)

[PART 2 21](#_Toc194002688)

[Why Agile? 21](#_Toc194002689)

[Scrum Framework for MSMS 21](#_Toc194002690)

[Main Activities Conducted During the Requirements Phase 22](#_Toc194002691)

[Requirements Table 24](#_Toc194002692)

[Foldable concept mapping of MSMS 27](#_Toc194002693)

[References 31](#_Toc194002694)

1.1 Phases of the Software engineering life cycle

The **Software Engineering Life Cycle (SELC)** refers to the structured process followed to develop, deploy, and maintain software systems. It ensures software is designed efficiently, meets user requirements, and remains adaptable to changes over time.

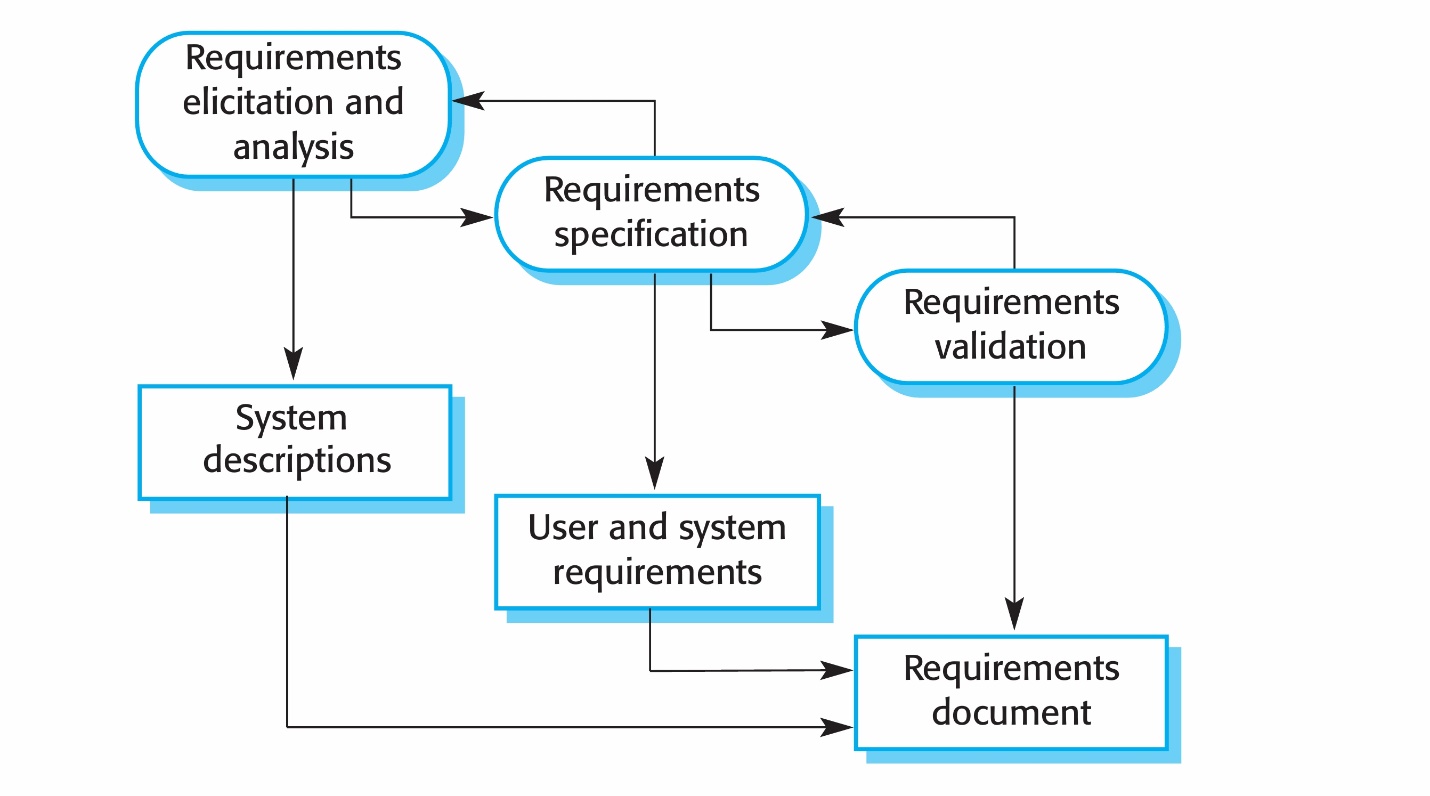
The software development life cycle (SDLC) consists of several critical phases, each playing a significant role in ensuring a software system is developed effectively and meets user needs (Sommerville, 2019)

### 1.1.1 Requirements Analysis

The first phase, Requirement Analysis, involves understanding the needs of stakeholders, including students, supervisors, and course leads, to define system functionalities. Through interviews, surveys, and document analysis, requirements are gathered and categorized into functional and non-functional specifications. Functional requirements include supervisor allocation, progress tracking, and meeting scheduling, while non-functional requirements focus on scalability, security, and usability.

However, there are a number of generic activities common to all processes

* Requirements elicitation
* Requirements analysis
* Requirements validation
* Requirements management.

**Fig 1.1** Requirement analysis process

## **1.1.1.1 Requirements Elicitation**

Requirements elicitation is the process of gathering and identifying the needs of stakeholders (users, clients, developers) for a software system. This phase is crucial as incorrect or incomplete requirements can lead to project failure (Sommerville, 2019).

1. stages of Requirement Elicitation
2. Requirements discovery
3. Requirement Classification
4. Requirement Prioritization
5. Requirement Specification

### Requirements discovery:

The process of gathering information about the required and existing systems and distilling the user and system requirements from this information. Interaction is with system stakeholders from managers to external regulators. ″Systems normally have a range of stakeholders. Examples are **Interviews**, ethnographys**urveys & Questionnaires**

1. Requirements classification and organization,

Once requirements are gathered through elicitation techniques such as interviews, observations, and ethnography, they need to be classified and organized to ensure clarity, consistency, and feasibility. Proper classification helps stakeholders understand different types of requirements and their impact on the system, while organization ensures that dependencies and priorities are clearly defined (Sommerville, 2019).

### Requirements prioritization and negotiation

Prioritization helps development teams focus on essential features while deferring less critical ones. Common prioritization techniques include: MoSCoW Method, Kano Model, 100 point Method.

### Requirements specification

The process of writing the user and system requirements in a requirements document. User requirements have to be understandable by end users and customers who do not have a technical background. System requirements are more detailed requirements and may include more technical information. The requirements may be part of a contract for the system development . It is therefore important that these are as complete as possible

## **1.1.1.2. Requirements Analysis**

Requirements analysis involves refining and structuring the gathered requirements to ensure they are clear, consistent, and feasible for implementation (Pressman & Maxim, 2020).

## **1.1.1.3. Requirements Validation**

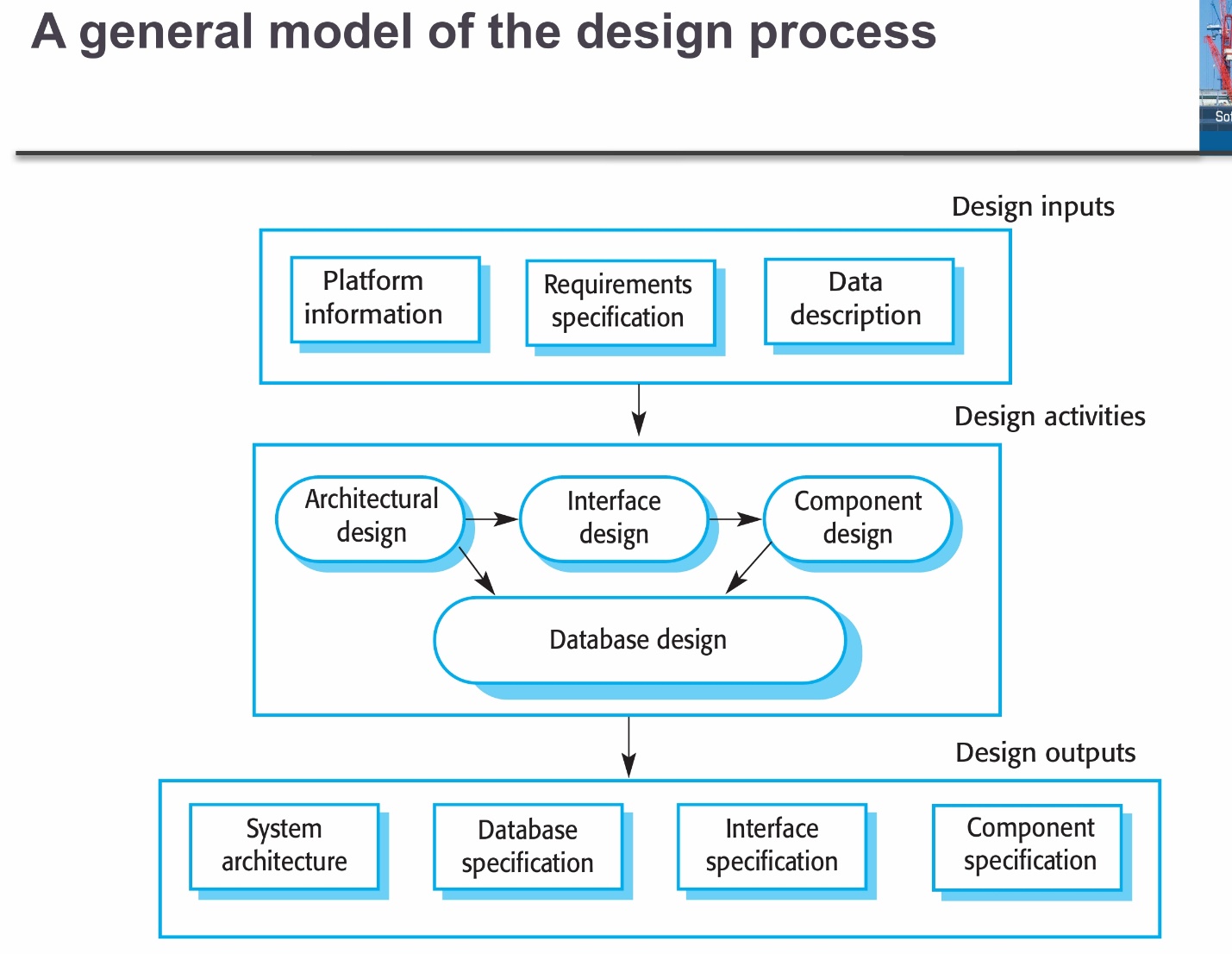
Requirements validation ensures that the collected requirements accurately reflect the stakeholder’s needs and are free from inconsistencies or errors (Pfleeger & Atlee, 2019).

## **1.1.1.4. Requirements Management**

Requirements management involves tracking, updating, and controlling changes to requirements throughout the software development lifecycle (Sommerville, 2019). Since requirements often evolve due to business or technological changes, effective management is crucial.

### 1.1.2 System Design

The next phase, System Design, translates these requirements into a blueprint for the system. Architectural design choices, such as a client-server model with a web and mobile interface, are established. Design decisions affect performance, scalability, and security. Database design is critical at this stage, ensuring relational database structures are developed for storing student records, meeting logs, and feedback. User interface design also plays a vital role in crafting an intuitive dashboard that facilitates seamless user interactions. The architectural blueprint of the system is created, including user interface, database structure, and software components (Pfleeger & Atlee, 2019).

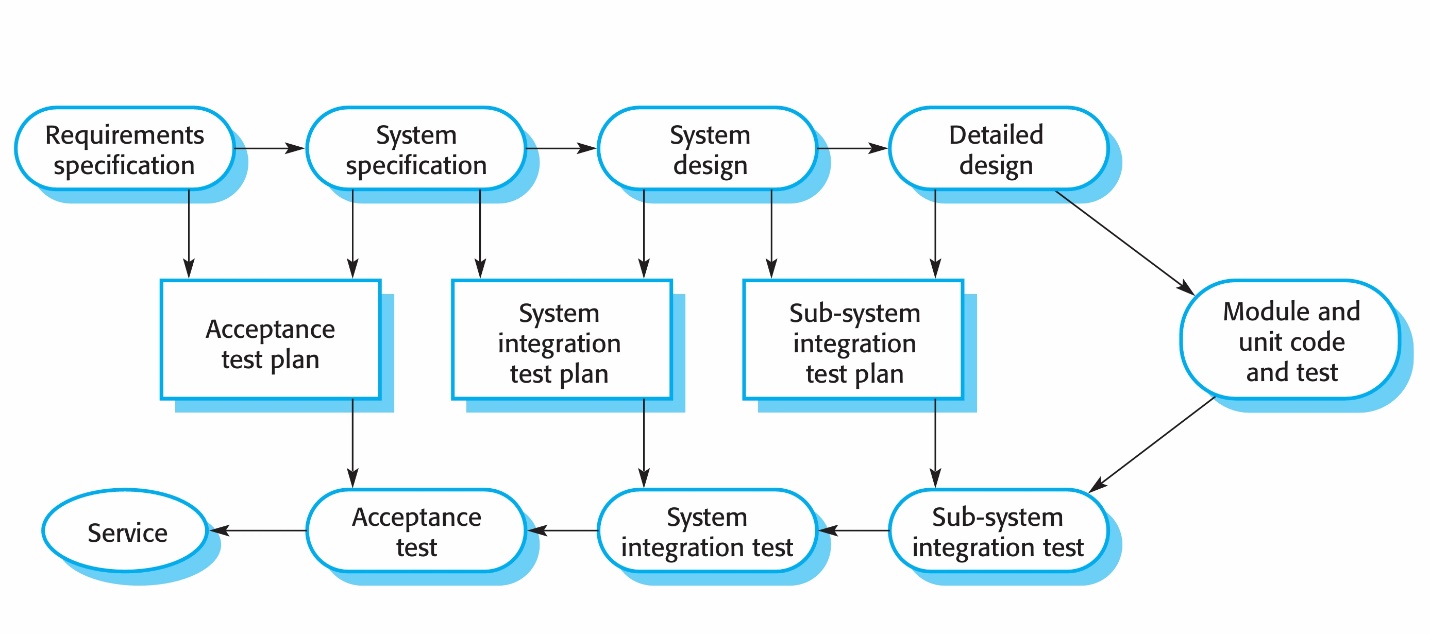
**Fig 1.2** General model of design process

### 1.1.3 Implementation

Implementation follows, where the actual coding and development of the system take place. Agile methodologies enable iterative development, allowing incremental releases of system features. Developers work on the backend using Python/Django, the frontend with React.js, and PostgreSQL for efficient database management. This phase ensures that all core functionalities, such as automated supervisor allocation and document submission, are integrated effectively. The actual software development takes place based on the system design (Pressman & Maxim, 2020).

### 1.1.4 Testing

Once implementation is complete, the Testing phase ensures the system functions as intended. Unit testing verifies individual modules, integration testing ensures different components work together, and user acceptance testing involves students and supervisors providing feedback on system usability. Security testing is also crucial to protect sensitive academic data. Software is evaluated to ensure it functions correctly, with various testing methods such as unit testing, integration testing, and user acceptance testing (Sommerville, 2019).



**Fig 1.3** Stages of testing

### 1.1.5 Deployment

The Deployment phase involves making the system live and available for users. A staged deployment strategy is followed, where beta testing is conducted with a select group before full-scale rollout. Continuous Integration and Continuous Deployment (CI/CD) mechanisms automate the deployment process, ensuring minimal downtime and frequent updates. The software is made available to users, often in a phased rollout with beta testing before full deployment (Pressman & Maxim, 2020).

### 

### 1.1.6 Maintenance and Support

Finally, the Maintenance phase ensures the system remains functional and efficient over time. Periodic updates are applied to fix bugs, introduce new features, and address security vulnerabilities. Data backups and performance monitoring tools such as New Relic are used to ensure system reliability. Updates, bug fixes, and security patches are regularly applied to keep the system functional (Voigt & Bussche, 2017).

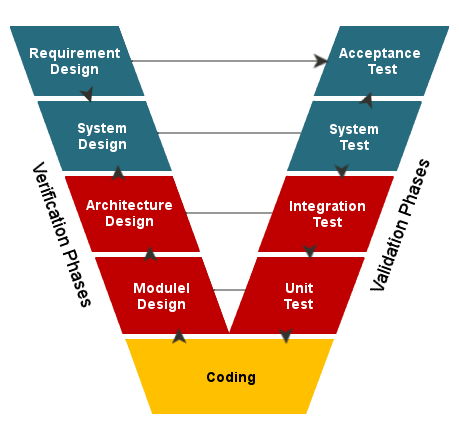
### 1.2 Software engineering methodologies

### 1.2.1 V-model

The V-Model, also known as the Verification and Validation model, is a linear SDLC approach that emphasizes testing and validation at every stage of the development process. The model gets its name from its "V" shape, which represents the parallel development and testing phases. This makes it more preferable and a faster software development model when compared to the traditional waterfall model. On the other hand, just like the waterfall model, it is less flexible, and there is little room for iterations. The entire product will only be ready at the implementation phase (Stephen and Oriaku, 2014).

Best For:

1. Critical systems where early testing and validation are essential (e.g., embedded systems, medical devices).
2. Projects with stable and well-documented requirements where verification at each stage reduces errors.
3. Projects requiring heavy testing at each phase to prevent costly mistakes later.



**Fig 1.4** V-model

Advantages of the V-Model

* 1. Strong emphasis on testing: The V-Model prioritizes testing, ensuring that defects are identified and addressed early in the development process.
  2. Clear structure: The V-Model's linear approach and well-defined stages make it easy to understand and follow for both developers and stakeholders.
  3. Early detection of issues: By incorporating testing at each stage, the V-Model can identify problems early on, reducing the likelihood of costly rework later in the development process.

### 1.2.2 Prototype model

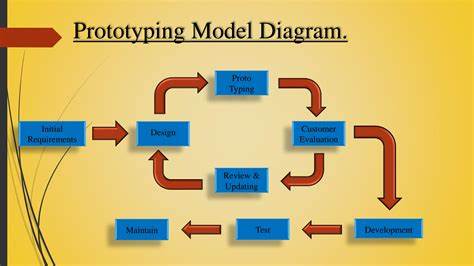
The Prototype Model focuses on building an initial version of the system to gather user feedback before full development. This methodology is advantageous when requirements are not well-defined, as it allows stakeholders to refine the system iteratively.

The prototyping model is lucrative in software development. It involves making a working replica of the software to be produced and allowing users to access it (geeksforgeeks, 2024).

Best For:

1. Projects with unclear or evolving requirements where user feedback is needed early.
2. New product development where stakeholders need to visualize the system before full development.
3. UX/UI-focused applications that require iterative design and validation.

Example: Building a new e-commerce platform where users need to test and refine the interface before full deployment.



**Fig 1.5** Prototyping model

Advantages of prototype model

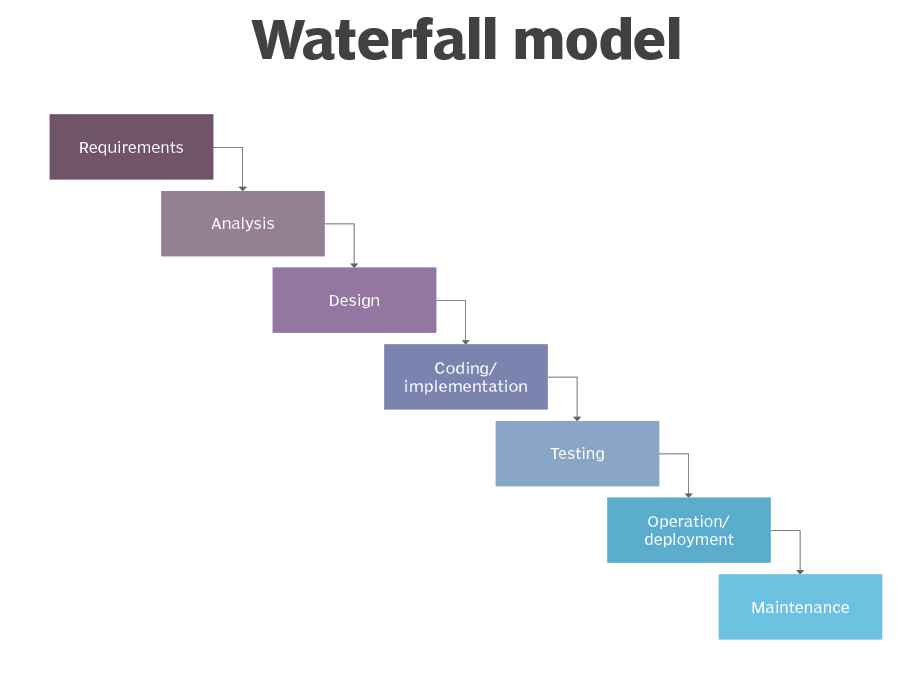
* + 1. Early User Feedback
    2. Improved Requirement Clarity
    3. Reduced Development Risk
    4. Faster Problem Identification

### 1.2.3 The waterfall model

The Waterfall Model follows a sequential approach where each phase must be completed before moving to the next. This method is beneficial for projects with well-defined requirements and minimal expected changes. However, it lacks flexibility, making it less ideal for dynamic projects like MSMS, where evolving needs require frequent updates.

Best For:

1. Well-defined projects with clear requirements and minimal expected changes.
2. Regulated industries (e.g., healthcare, aerospace, government) where documentation and approvals are essential.
3. Short-term projects with fixed budgets and deadlines.



**Fig 1.6** Waterfall model

Advantages of the Waterfall Model

Some of the benefits of the Waterfall model include:

* 1. Simplicity: The Waterfall model is easy to understand and follow, making it suitable for small projects with well-defined requirements. The Waterfall model is very easy to understand and implement, making it ideal for smaller projects with precise requirements.
  2. Documentation: The Waterfall model places a strong emphasis on documentation. Documentation ensures that every step of the process is recorded, which can be helpful for future reference, troubleshooting, and understanding how and why someone made any particulate decisions throughout the project.
  3. Manageability: Due to its sequential nature, the Waterfall model allows for easier project management. Each phase has specific deliverables and deadlines, making monitoring progress and allocating resources efficiently simple.

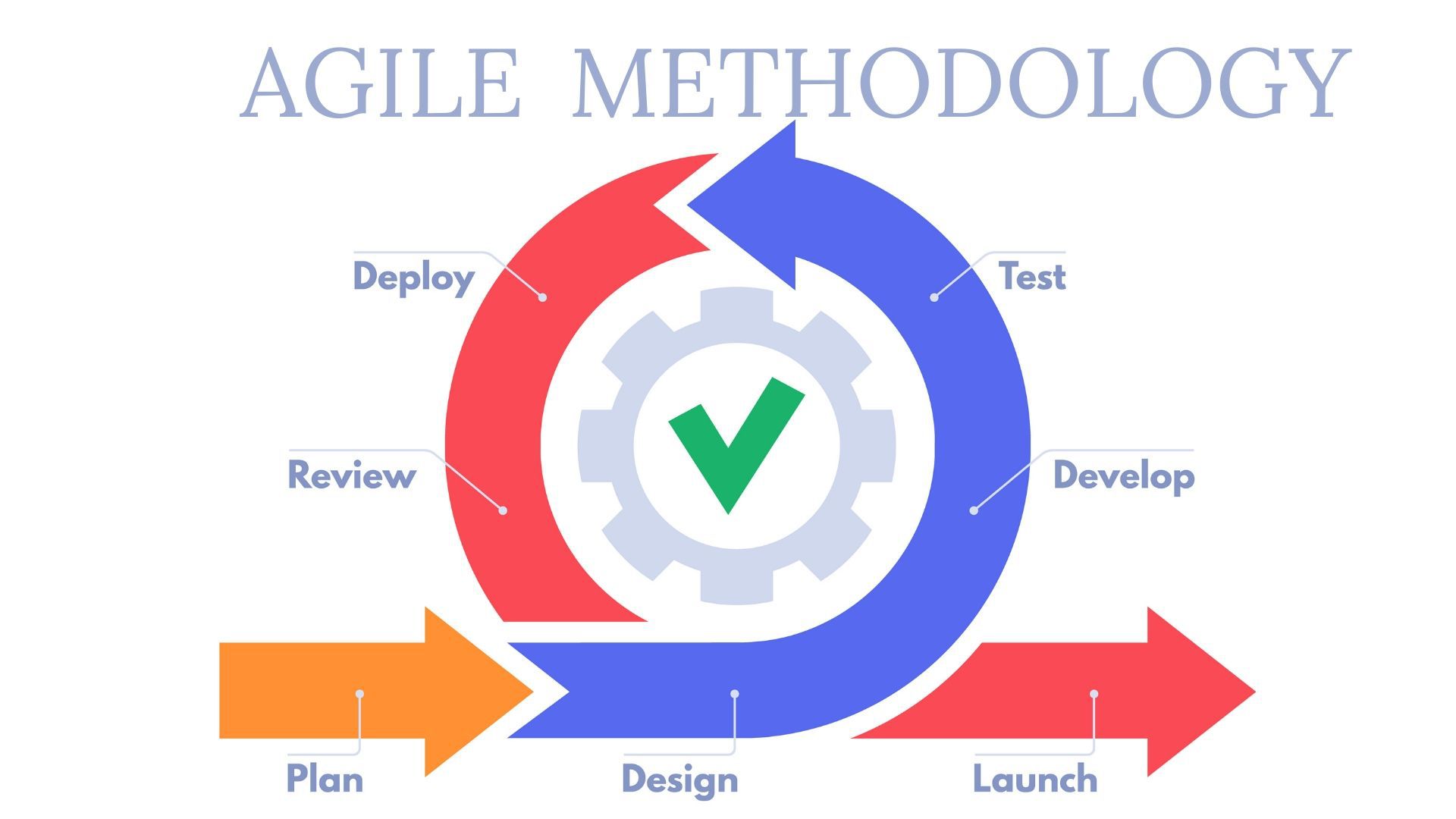
### 1.2.4 The agile methodology

The Agile Methodology is a highly flexible and iterative approach that emphasizes collaboration, adaptability, and rapid delivery of functional software. Agile is ideal for MSMS since it allows stakeholders to provide continuous feedback, ensuring the system evolves according to their needs. Teams work in sprints, regularly testing and refining functionalities, making it easier to address challenges early in the development cycle. Agile methodology is a project management framework that breaks projects down into several dynamic phases, commonly known as sprints. (Asana 2025).

Best For:

1. Projects requiring flexibility where requirements evolve frequently.
2. Fast-paced environments where continuous development and user feedback are crucial.

Example: Developing a social media app where user preferences and trends change rapidly.



**Fig 1.7** Agile Methodology

Advantages of Agile Methodology

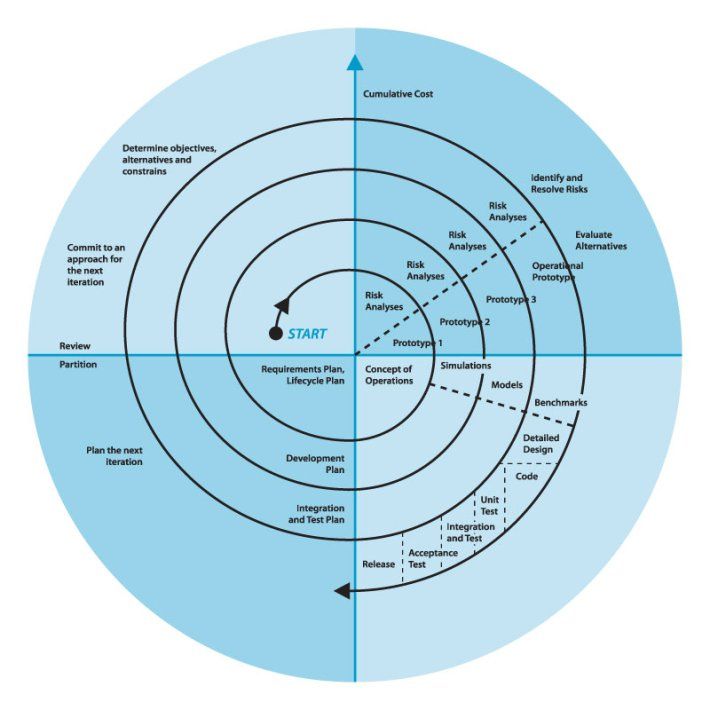
* 1. Focus on Customer Value: Agile places a high priority on providing customers with value by attending to their requirements and preferences.
  2. Enhanced Team Morale and Motivation: Agile gives teams the freedom to own their work and decide together..
  3. Stakeholder Collaboration: Throughout the development process, agile promotes strong coordination between product owners, developers, and other stakeholders.
  4. Early and Continuous Delivery: Agile encourages the tiny, incremental releases of functional software.

### 1.2.5 The spiral model

The Spiral Model incorporates iterative development while also focusing on risk assessment and mitigation. It is ideal for high-risk projects that require frequent evaluations and adjustments. However, due to its complexity and cost, it is better suited for large-scale enterprise systems rather than a university management system like MSMS.(Abhey 2025).

Best For:

1. Large and high-risk projects where iterative risk assessment is needed.
2. Projects with frequent requirement changes where flexibility is necessary.



**Fig 1.8** Spiral model

Advantages of Spiral model

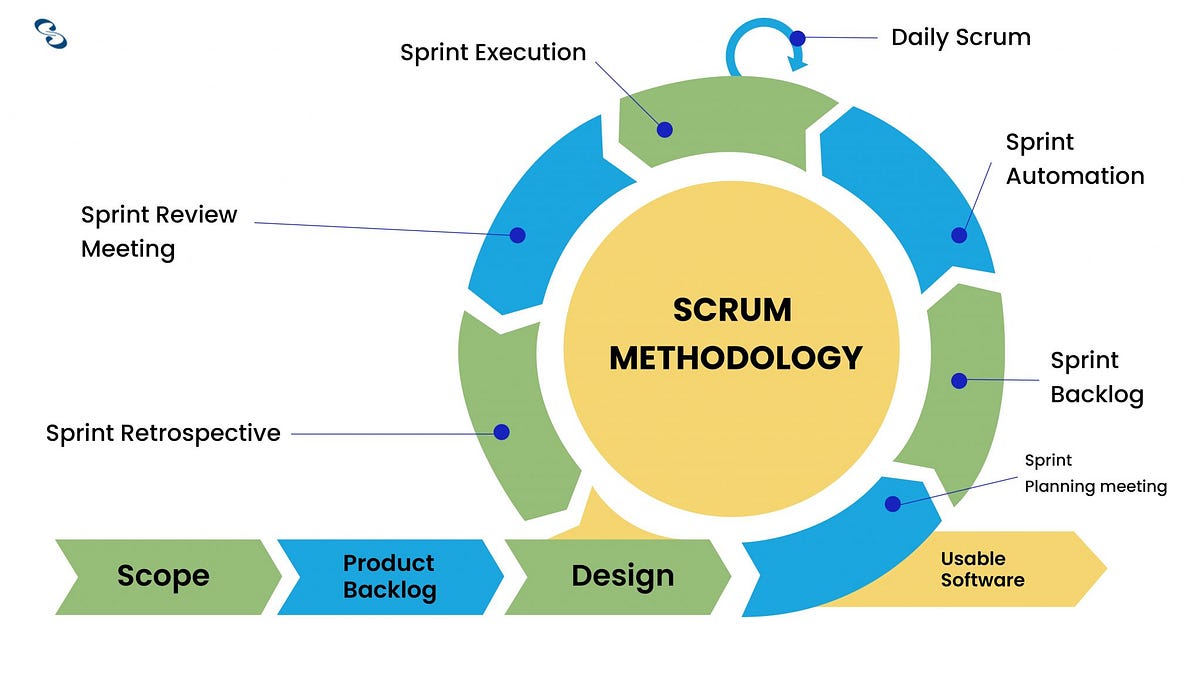
* 1. Risk Management: Continuous risk assessment throughout the development process allows teams to address potential challenges before they escalate into critical issues.
  2. Flexibility: The model's inherent iterative nature supports modifications and enhancements at every stage of development, facilitating an adaptive approach.

1.2.6 The Scrum Model

A subset of Agile that organizes work into sprints (short development cycles) and uses roles like Product Owner, Scrum Master, and Development Team. Scrum is a flexible Agile approach that helps teams move quickly and deliver results in short cycles. Scrum “is an Agile Project Management framework that organizes projects in iterations called Sprints. The Scrum Team is responsible for delivering value, while the Product Owner communicates with the stakeholders and prioritizes the work. This iterative approach allows Scrum to produce work and test it quickly.”(Teamhood 2021).

Best For:

1. Complex and fast-moving projects where frequent updates are required.
2. Software requiring continuous integration and rapid releases in short development cycles (Sprints).



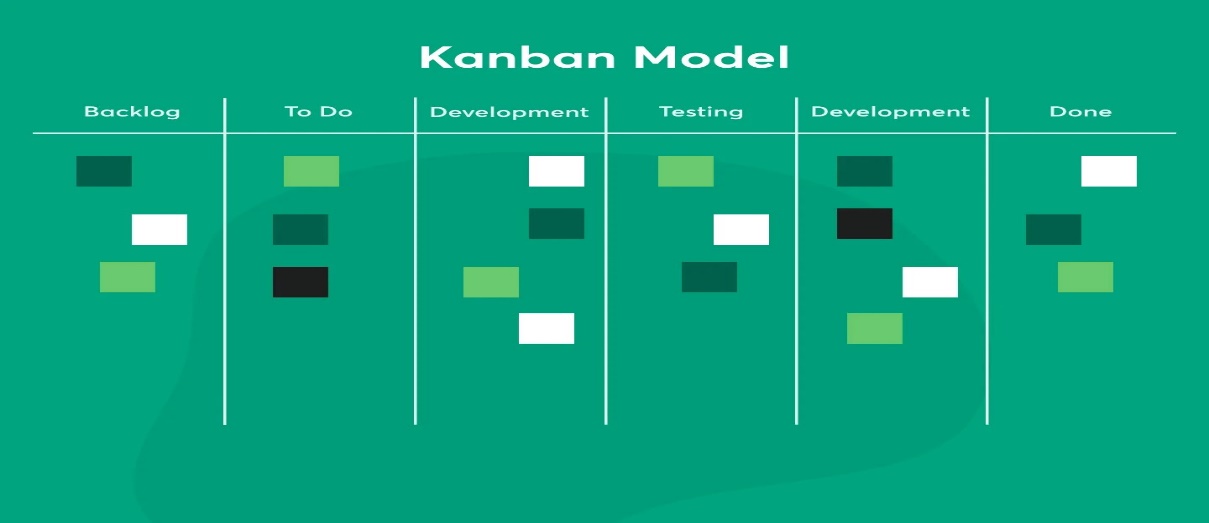
**Fig 1.9** Scrum methodology

Advantages of Scrum Model

1. Adaptability and flexibility: Scrum is itself a flexible and adaptable practice and is also suitable for a wide variety of environments and situations that require a flexible approach for example, when it can be difficult to identify clear requirements.
2. Creativity and innovation: Scrum emphasizes creativity and new ideas. Because Scrum teams work closely together and analyze ideas from all members, this encourages new and innovative solutions.
3. Time to market: Scrum often results in faster delivery than other approaches, speeding time to market for new products or features.

### 1.2.7 **Kanban**

A visual workflow management system that helps teams manage work in progress by using a board with columns for different stages of the development process. Kanban is a visual project management system originating from Toyota's lean manufacturing model. Unlike other Agile frameworks that rely on time-based iterations, Kanban focuses on continuous flow, allowing teams to deliver value steadily and rapidly.(Abhey 2025)



**Fig 1.10** Kanban

The Kanban system brings multiple benefits:

* 1. Enhances visibility of workflow
  2. Facilitates smooth workflow
  3. Reduces waste
  4. Enables faster delivery
  5. Provides flexibility in prioritization

### 1.2.8 **Lean Software Development**

Inspired by lean manufacturing, it aims to eliminate waste, improve efficiency, and deliver value to customers faster. The lean approach presents a very strategic approach toward software development as it makes it easier for teams to ship different functionalities and features at different periods while working on other features (Maritz and Hamdulay, 2018).

Best For:

1. Projects needing cost and resource efficiency (eliminating unnecessary processes).



**Fig 1.11** Lean Software Development

Advantages of Lean Software Development

1. Faster Time-to-Market: Lean methodologies prioritize delivering value quickly, allowing products to reach the market faster than traditional development approaches.
2. Improved Efficiency: By eliminating waste and focusing on essential features, Lean Software Development streamlines the development process, making it more efficient and productive.
3. Enhanced Quality: Building quality into the development process from the start ensures that products meet high standards and customer expectations, reducing defects and rework.

### 1.2.9 **RAD (Rapid Application Development)**

Unlike the Waterfall methodology, which dictates that software teams need a detailed roadmap for their project, RAD does not rely on heavy planning. On the contrary, its goal is to release a working application (or a small piece thereof!) faster, present it to stakeholders, test the application on real-world data and tasks, and tweak the software based on what’s been discovered so far.(Appery 2024)

Advantages of RAD

1. The major advantage of rapid application development is that software teams can change the key features and functionality of an application at any given moment.
2. Stakeholders communicate frequently, solving issues together and creating a product that best meets the business/technology needs of a company
3. With RAD, you can manage your resources appropriately and avoid running over budget without seeing a glimpse of real value from your new technology

### **1.2.10 DevOps**

DevOps Integrates development and IT operations to improve collaboration, automate deployment, and ensure continuous integration and delivery (CI/CD). By fostering collaboration, automation, and continuous integration, DevOps seeks to improve the efficiency and quality of the software development process. In this article, we'll explore the DevOps approach in detail, discuss its advantages and disadvantages, and understand how it addresses the triple constraint.(Abhey 2025)



**Fig 1.12** DevOps

Advantages of DevOps

1. Faster Delivery: By automating processes and integrating development and operations teams, DevOps helps shorten the software development lifecycle and enables faster delivery of software updates.
2. Improved Collaboration: DevOps fosters a culture of collaboration and shared responsibility, breaking down silos between teams and promoting better communication.
3. Higher Quality: With continuous integration and automated testing, DevOps helps ensure that software updates are of high quality and less prone to errors.
4. Increased Efficiency: Automation and streamlined workflows reduce manual tasks, allowing team members to focus on more valuable work.

### 1.3 Technological Tools for Each Life Cycle Phase in MSMS Development

### **1. Planning Phase**

This phase involves defining project scope, requirements, and feasibility.

**Tools Used:**  
 1. **Trello** – Task and project management tools for organizing activities.  
 2. **Microsoft Teams** – For team communication and collaboration.  
 3. **Lucid chart** – Diagramming tools for system architecture and workflow planning.

1. **Requirements Analysis**.

This phase focuses on gathering functional and non-functional requirements

**Tools Used:**   
1. **Google Forms** – For conducting surveys and collecting feedback from stakeholders.  
2. **Microsoft Word** – For documenting requirement specifications.

## **System Design**.

Here, the system architecture, database schema, and UI/UX are defined

**Tools Used:**  
1. **Lucid chart** – For designing ERDs, flowcharts, and system diagrams.  
2. **Figma** – For designing user interfaces and prototypes.

1. **Development Phase**

This is where the system is coded and implemented.

**Tools Used:**  
 1. **Visual Studio Code** – IDEs for coding (Node.js, Python, or Java-based backend).  
 2. **React.js** – Frontend frameworks for building an interactive UI.

### **Testing Phase**

This phase ensures that the system functions as expected.

**Tools Used:**

1. JUnit – Automated testing for functional and UI testing.

2. Postman – For API testing and backend validation

### **Deployment Phase**

**Tools Used:**

1. Docker – For containerization and scalable deployments.

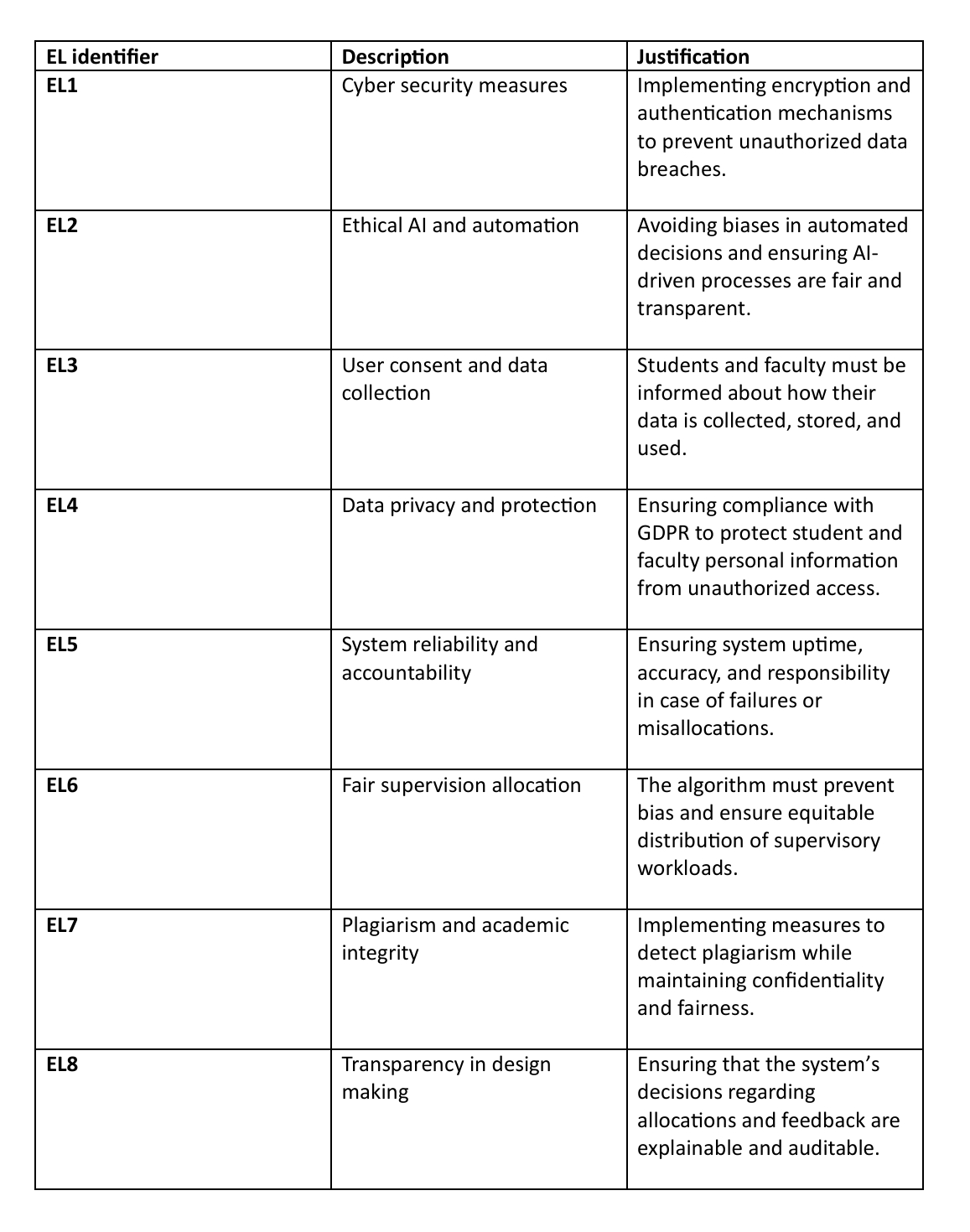
2. AWS – Cloud hosting services for the system.

1. **Maintenance & Monitoring Phase**

**Tools Used:**  
1. **Google Analytics**– For tracking system usage and performance.

### 1.4 Ethical and Legal Issues in System Analysis and Design

When analysing and designing the MSc Supervision Management System (MSMS), several ethical and legal challenges must be considered. These concerns include data privacy, fairness in supervisor allocation, accessibility, intellectual property rights, and security.



### PART 2

The MSc Supervision Management System's (MSMS) most suitable methodology

The Agile technique more especially, the Scrum framework is the most suitable methodology for the MSMS. Agile is perfect for this system as it is user-driven, flexible, and iterative, providing that the system changes in response to input from stakeholders.

### **Why Agile?**

1. **Frequent Feedback & Iterative Development**

* The MSMS serves multiple stakeholders, including **students, supervisors, and course leads,** each with evolving requirements.
* Agile allows the system to be developed **incrementally**, with continuous feedback ensuring the final product meets user expectations.

1. **Flexibility & Adaptability**
   * Dissertation supervision involves **dynamic processes**, such as changes in supervisor allocation, scheduling conflicts, and evolving academic policies.
   * Agile allows for **quick adjustments** to accommodate these changes.
2. **Risk Reduction**
   * Agile breaks development into small **manageable iterations** (sprints), reducing the risk of major failures by identifying issues early.
   * This minimizes wasted time and resources while ensuring critical functionalities are tested throughout the development cycle.
3. **User-Centric Development**
   * With **students and supervisors as primary users**, Agile ensures that their feedback is incorporated at every stage.
   * Features like **progress tracking, document management, and meeting scheduling** are refined based on real user input.
4. **Improved Collaboration**
   * Agile promotes strong **collaboration between developers, IT support, and course leads** through **daily stand-up meetings and sprint reviews**.

### **Scrum Framework for MSMS**

The **Scrum methodology,** a subset of Agile, is best suited for MSMS because it offers a structured yet flexible approach.

**Key Components of Scrum for MSMS:**

1. **Product Backlog** – A list of prioritized MSMS features such as supervisor allocation, document tracking, meeting scheduling, and automated notifications.
2. **Sprints** – Short development cycles (2–4 weeks) focusing on delivering small, functional parts of the system.
3. **Daily Standups** – Short meetings to discuss progress, challenges, and next steps.
4. **Sprint Reviews & Retrospectives** – Continuous feedback sessions with stakeholders to improve the system.

### **Main Activities Conducted During the Requirements Phase**

The **requirements phase** is crucial for defining the functionalities and scope of MSMS. The following key activities were conducted:

#### **1. Requirements Elicitation**

To ensure a comprehensive understanding of stakeholder needs, multiple elicitation techniques were used:

* **Stakeholder Interviews:** I conducted discussions with **students, supervisors, and course leads** to understand their challenges in the supervision process.
* **Surveys & Questionnaires:** I gathered structured feedback on current supervision inefficiencies and user expectations.
* **Document Analysis:** I reviewed existing systems like Moodle, email-based tracking, spreadsheets to identify gaps and pain points.
* **Stories & Scenarios:** I developed user scenarios to understand potential use cases, such as:
  1. A student struggling to schedule a meeting with a busy supervisor.
  2. A supervisor managing multiple students without a centralized progress tracking tool.
  3. A course lead reallocating students due to an overwhelmed supervisor.

#### **2. Requirements Analysis**

* **Functional Requirements Identified:** Core features such as **supervisor allocation, document submission, meeting scheduling, student dashboard, lecturer dashboard and notifications** were documented.
* **Non-Functional Requirements Defined:** We focused on **security, scalability, usability, and system performance** to ensure a robust platform.

#### **3. Requirements Validation**

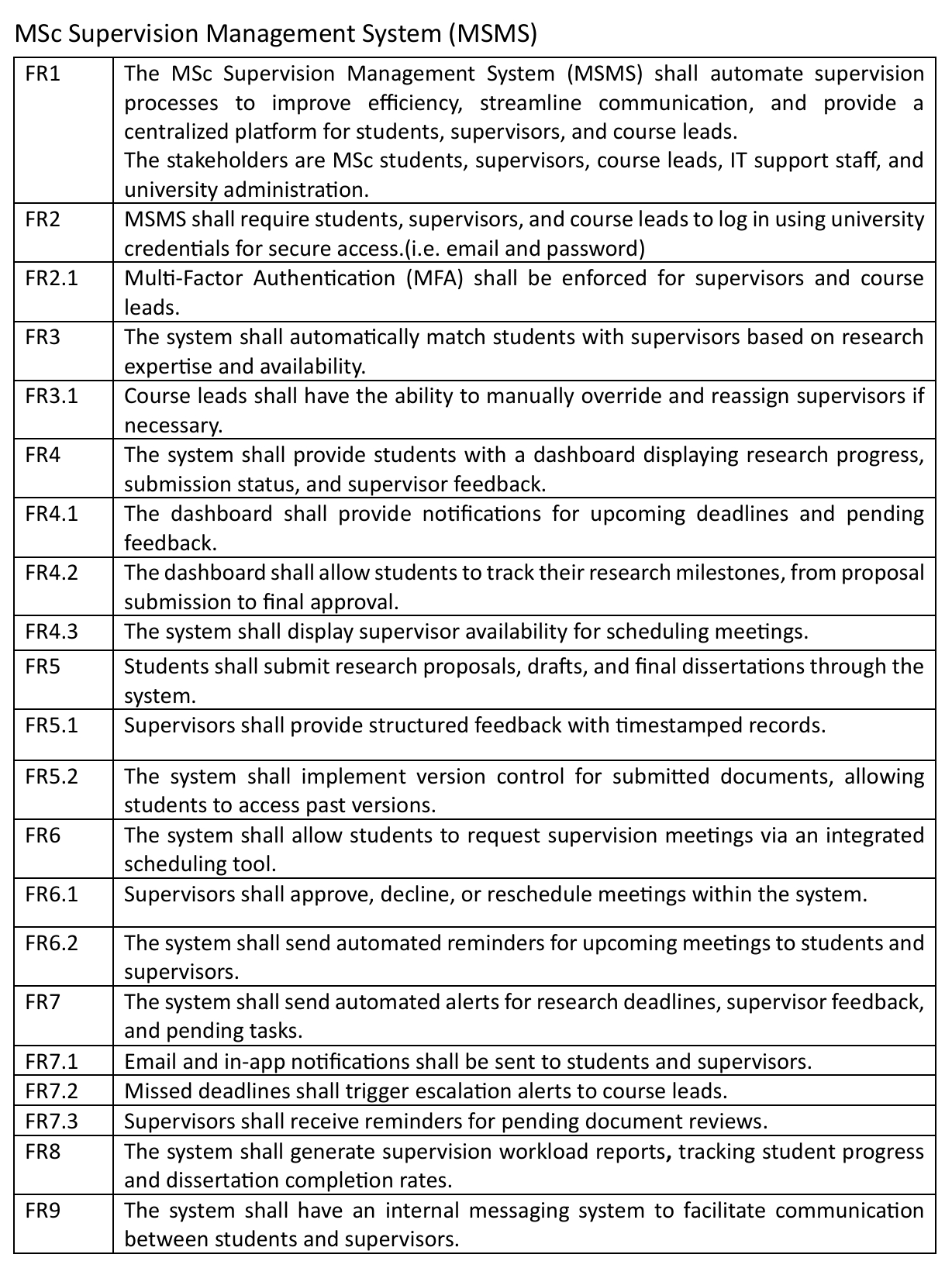
* **Stakeholder Feedback Sessions:** Gathered feedback from potential users to refine system requirements.
* **Conflict Resolution:** Addressed differing expectations among stakeholders (e.g., balancing supervisor workloads).

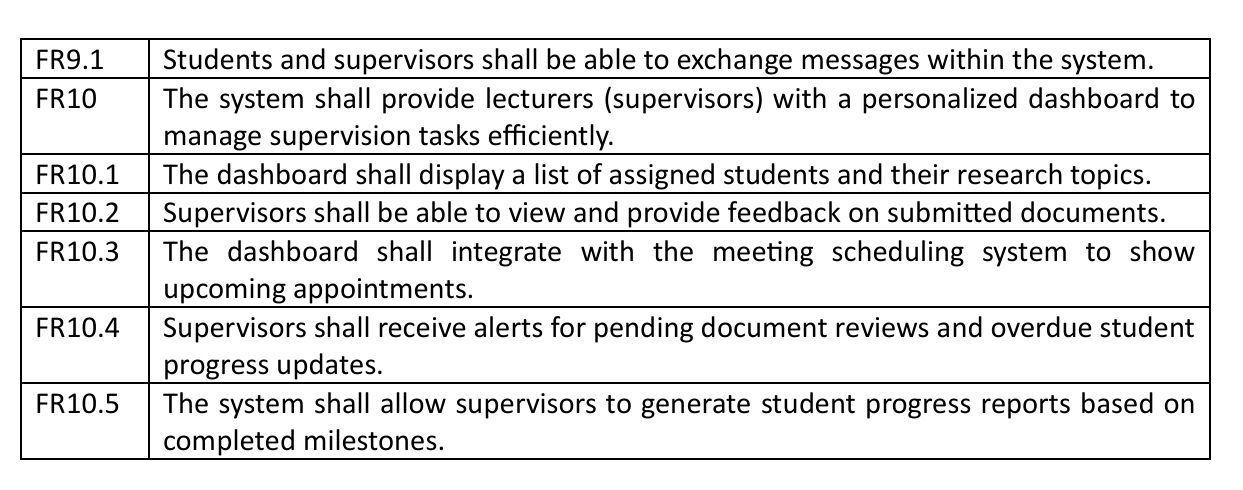
#### **4. Requirements Documentation & Management**

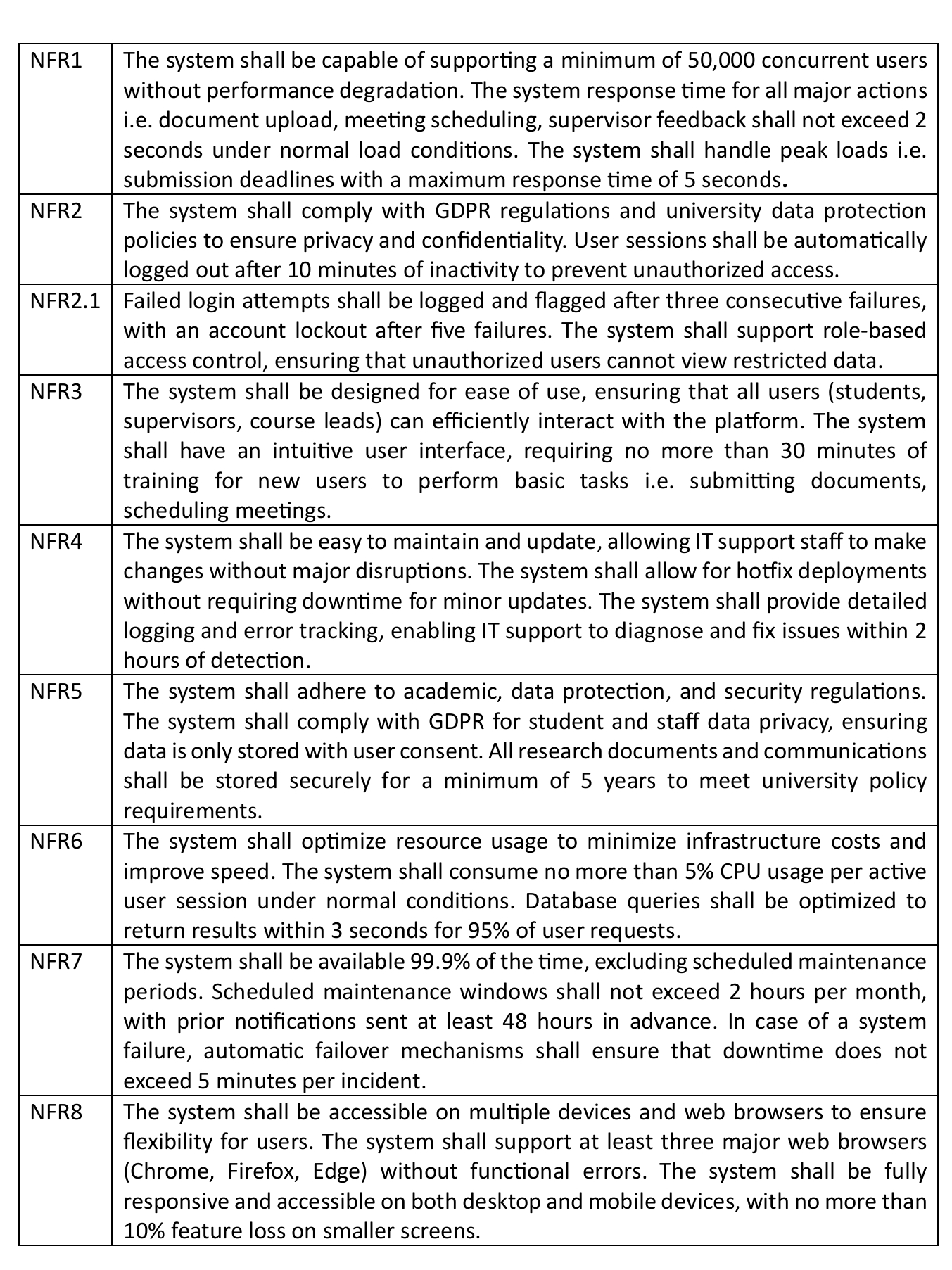
* Created a **Software Requirements Specification (SRS)** document.
* Established **version control** to track requirement changes.
* Used **JIRA** for task tracking and backlog management.

By following these structured activities, the MSMS was designed to be **user-centric, efficient, and adaptable to evolving academic needs.**

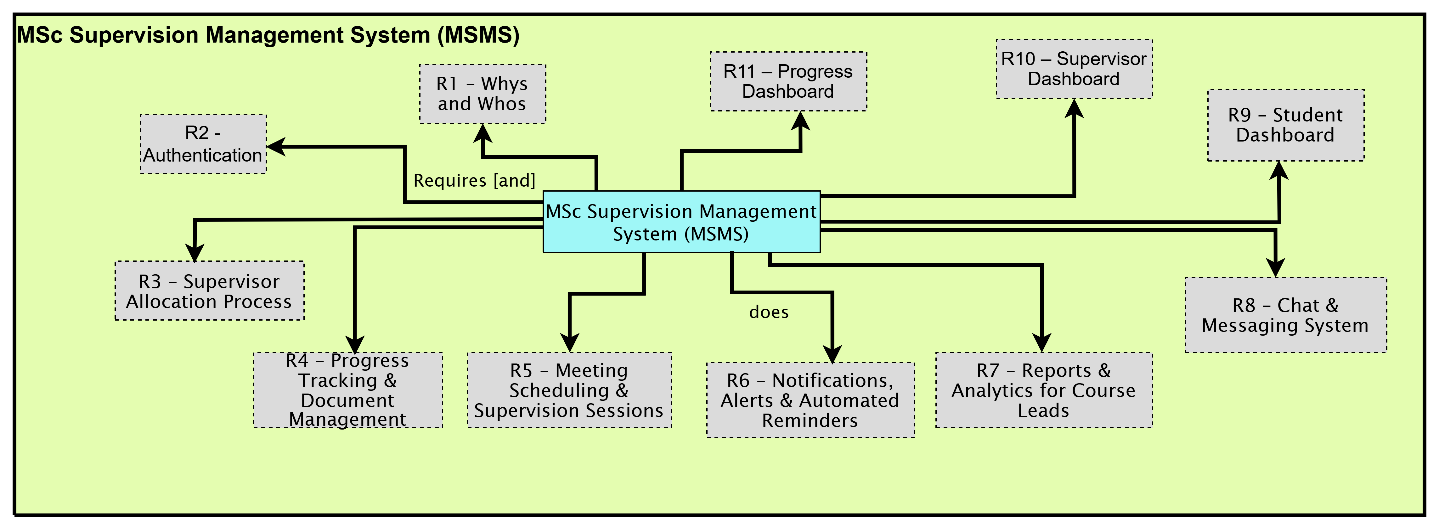
Requirements Table

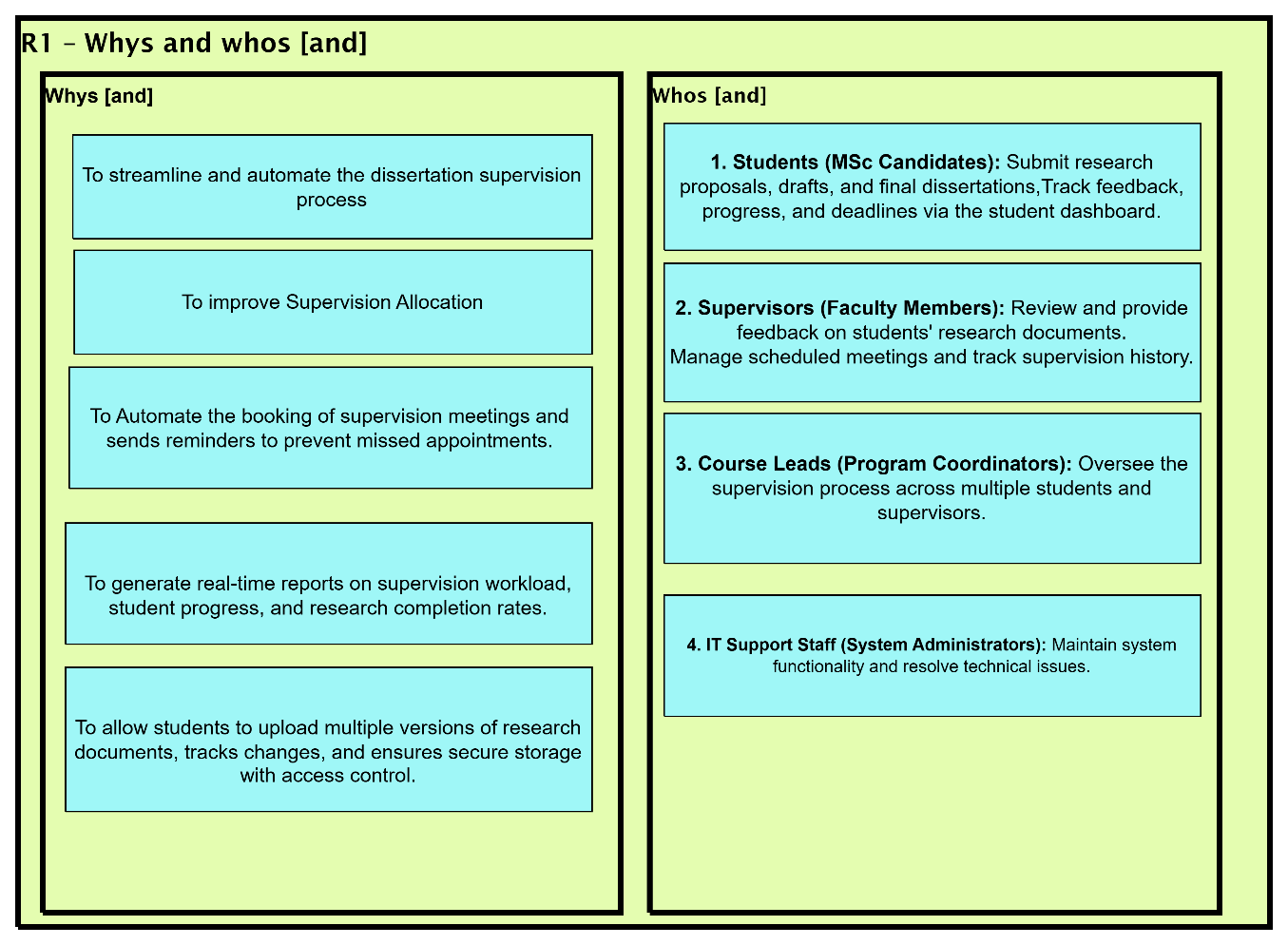


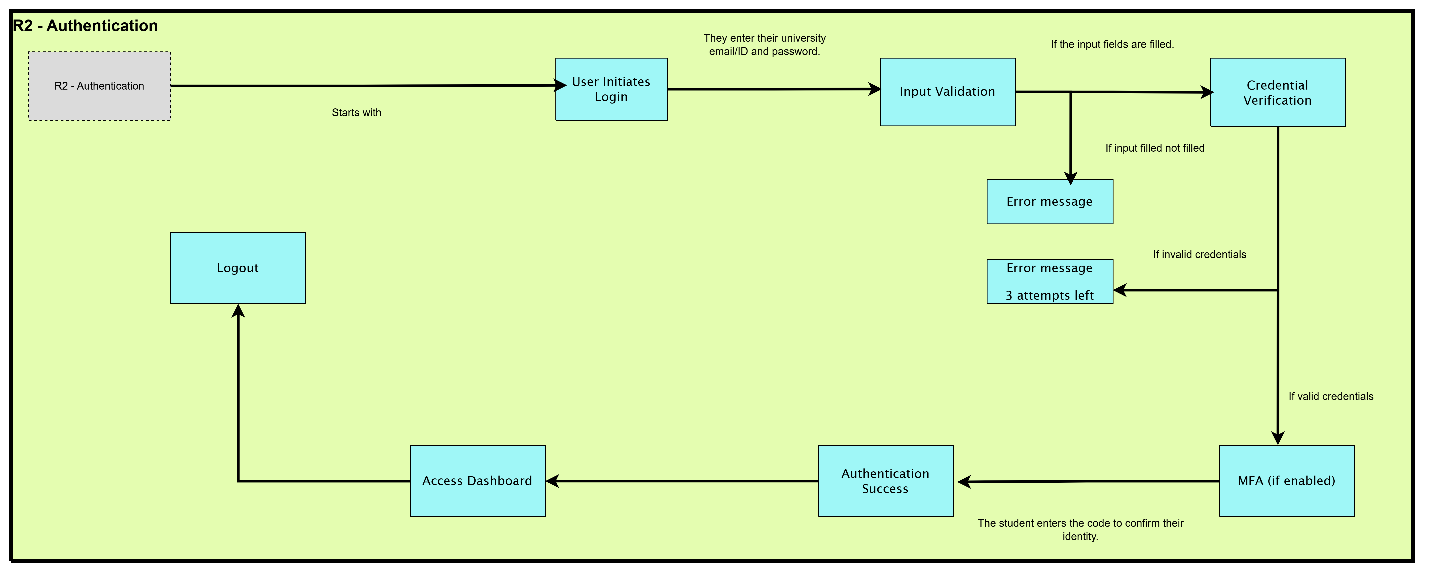


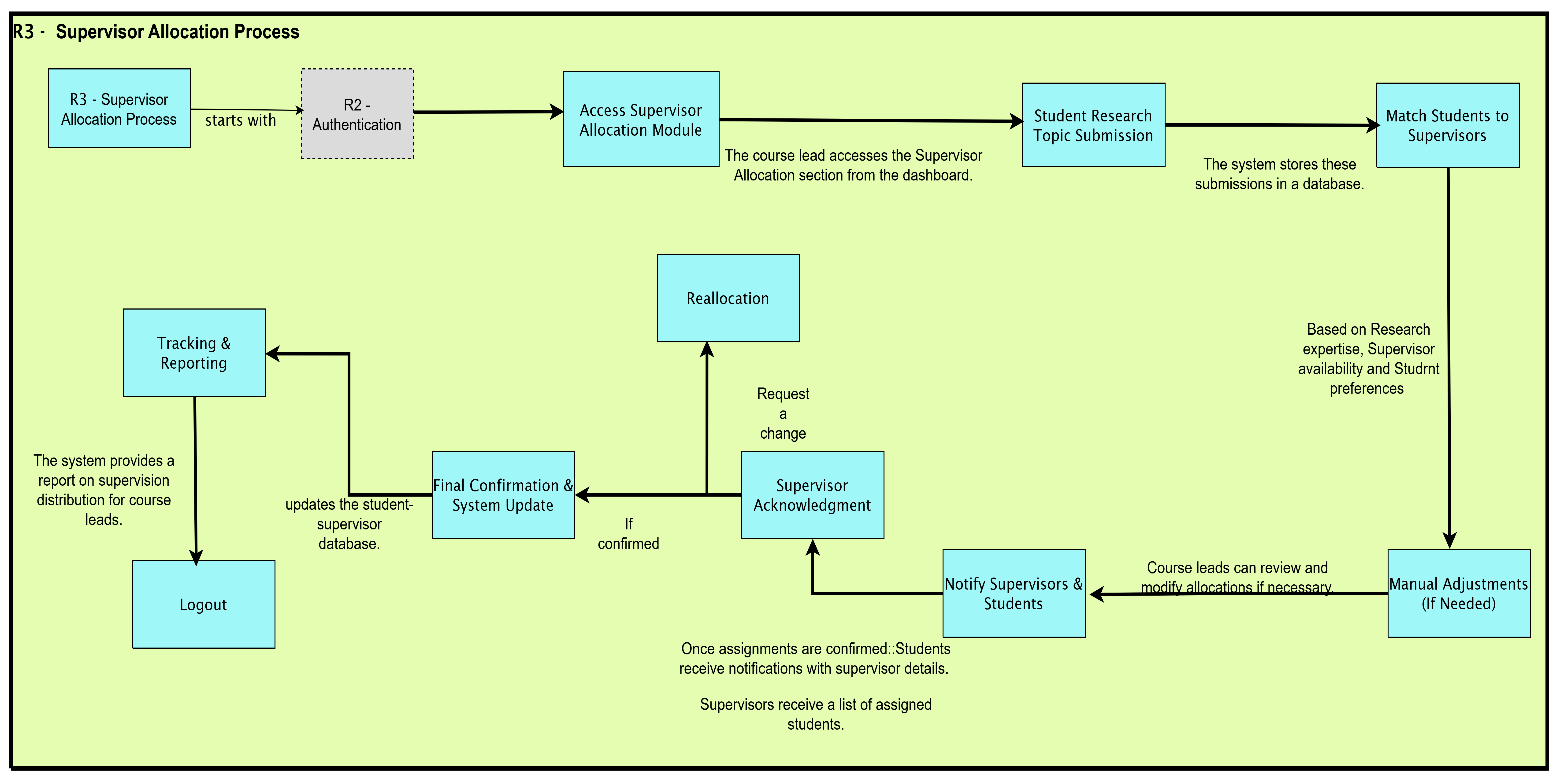


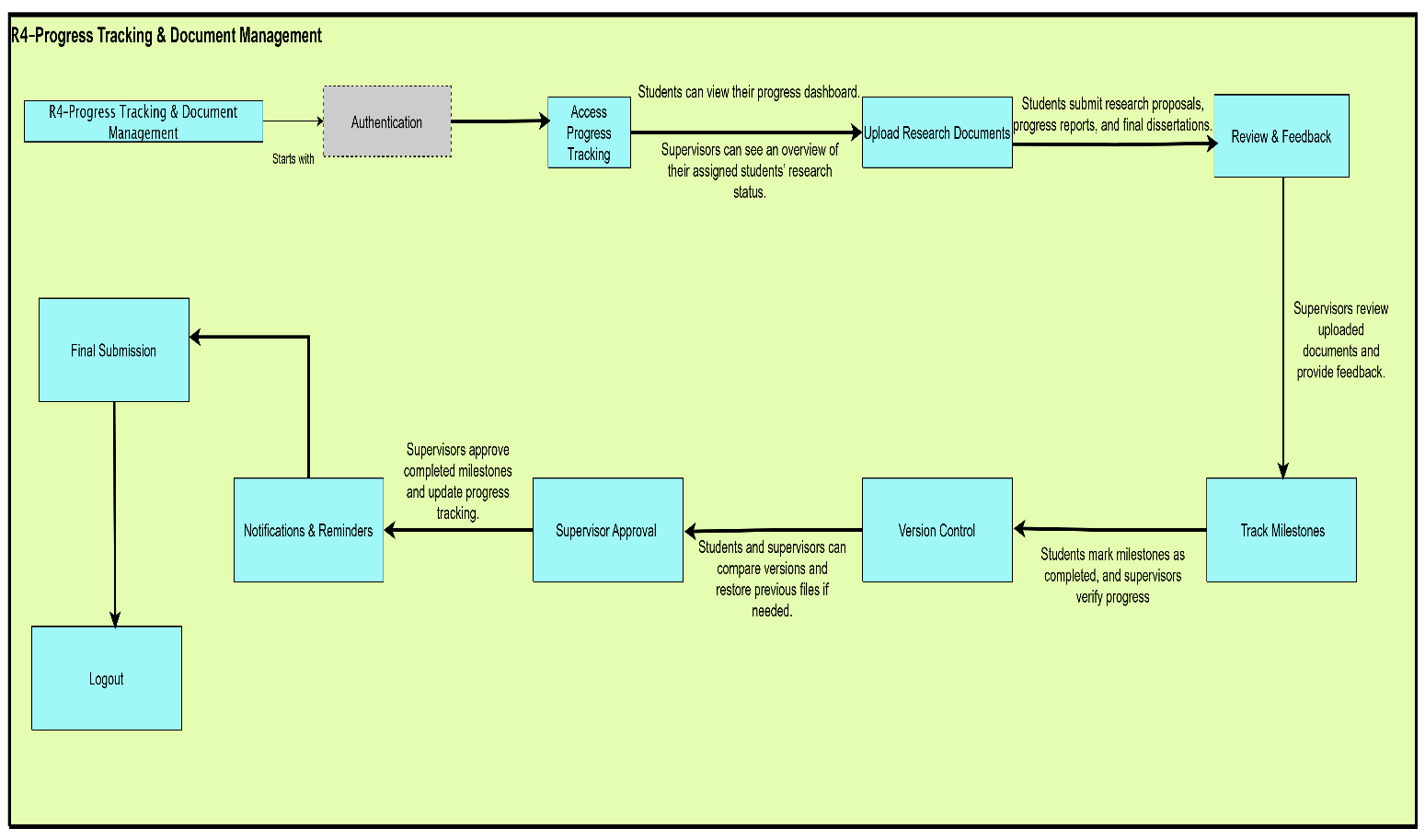
### Foldable concept mapping of MSMS

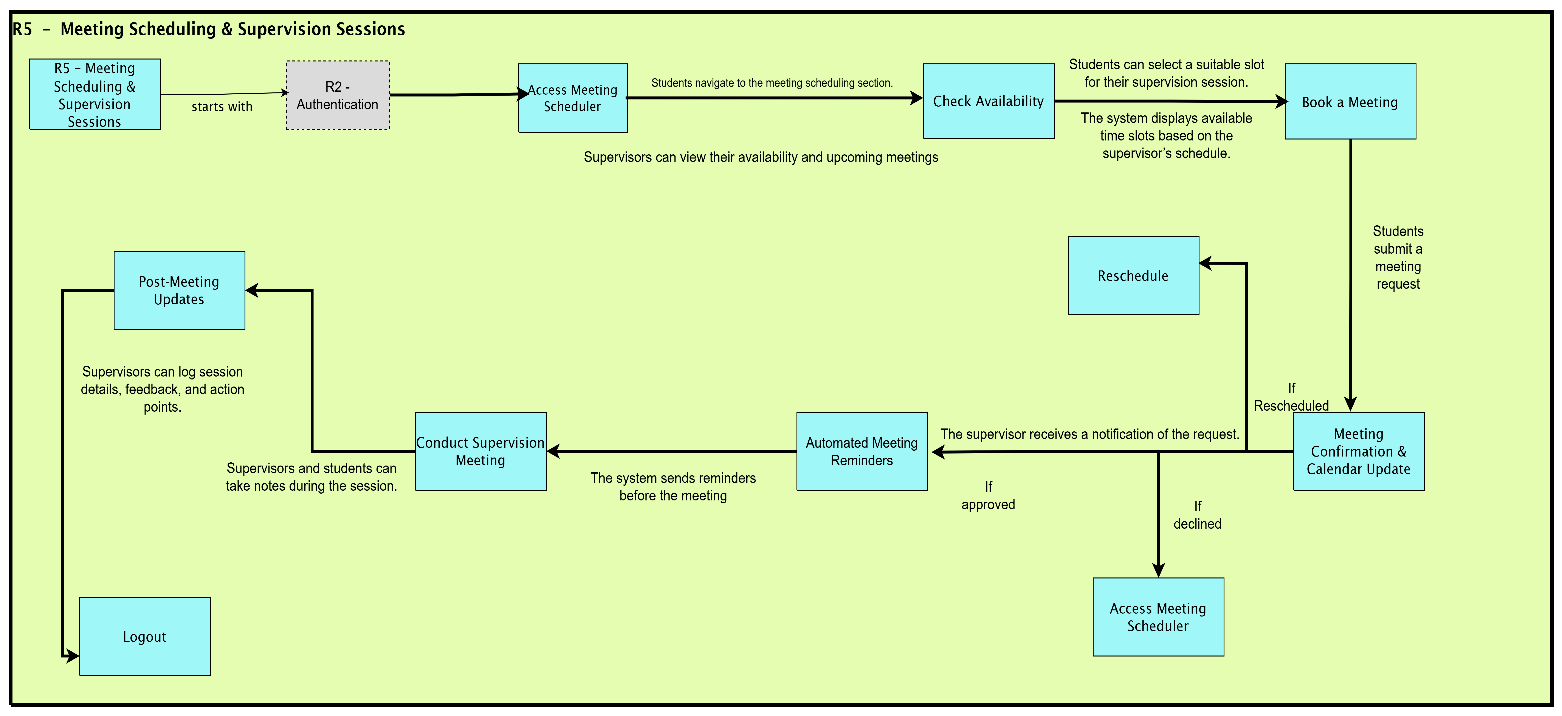


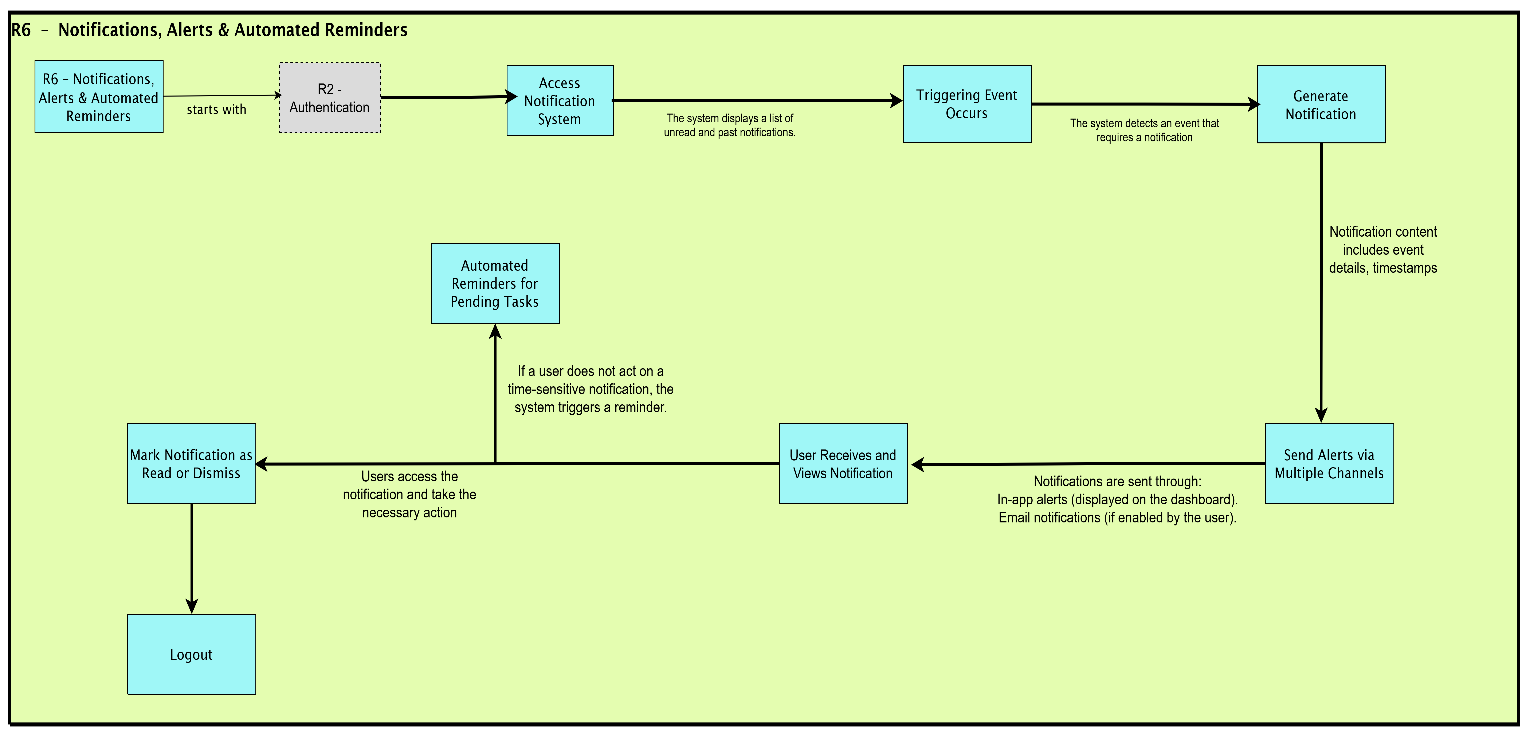


****

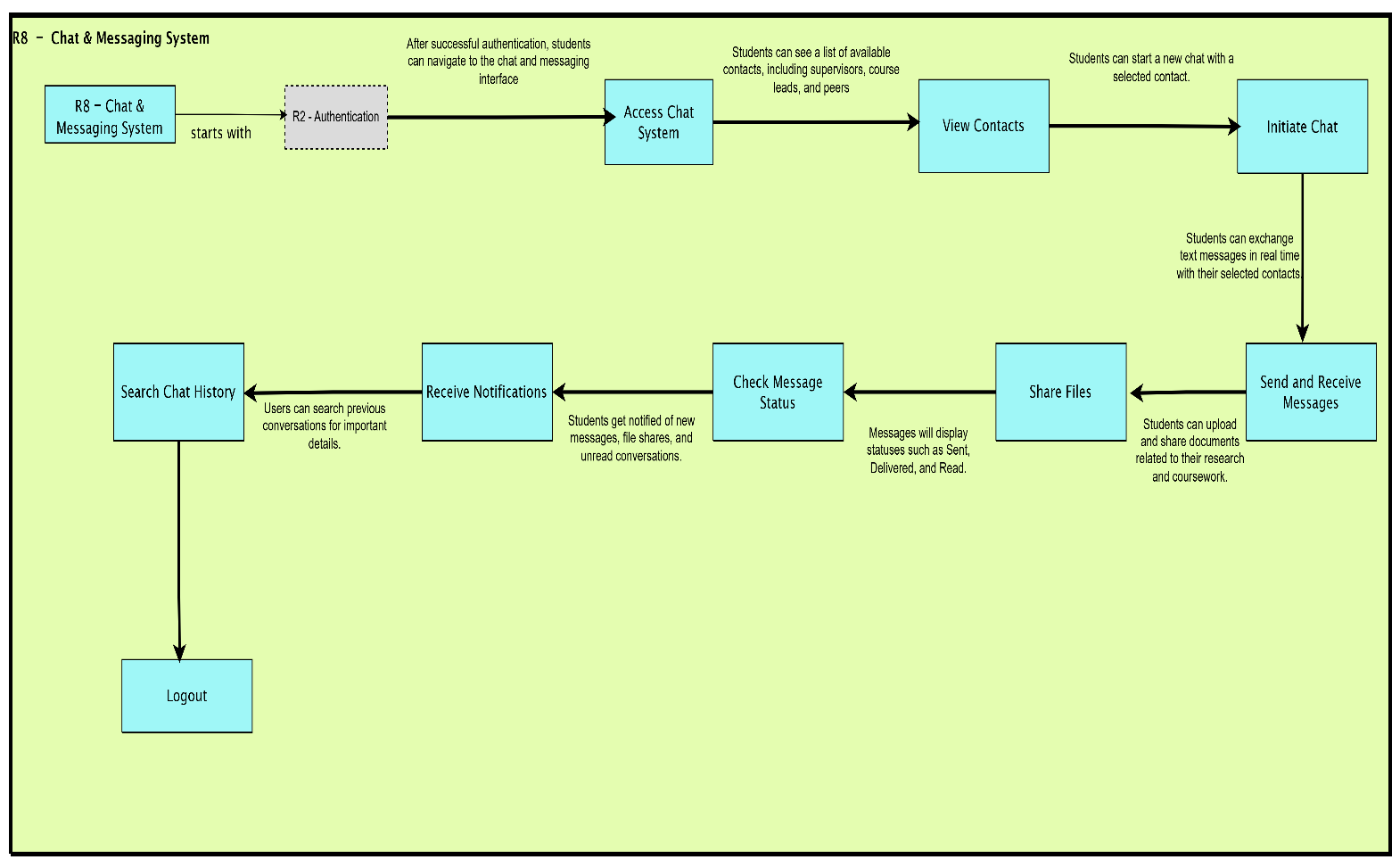


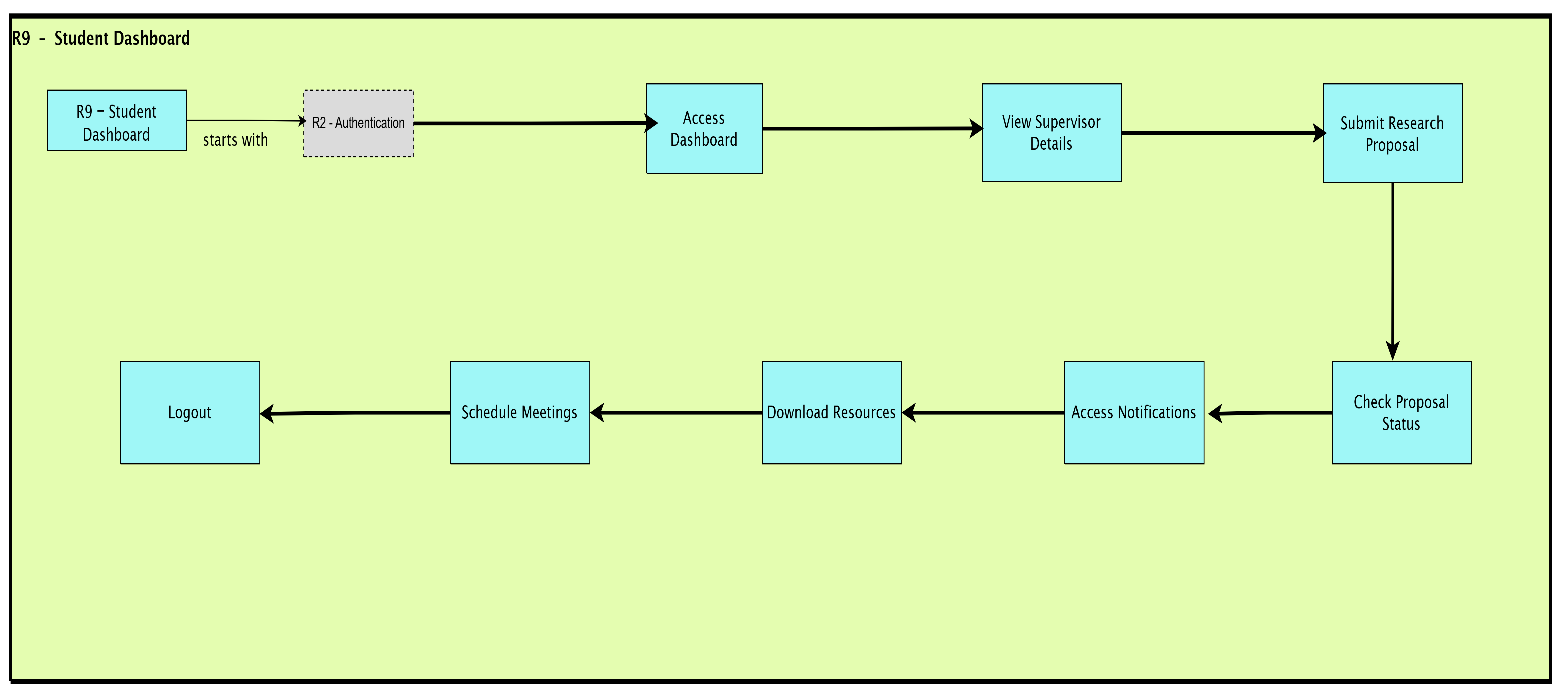












### References

**GeeksforGeeks (2024)** Prototyping Model in Software Development*.* Available at: https://www.geeksforgeeks.org/prototyping-model-in-software-development

**Maritz, J. and Hamdulay, A. (2018)** Lean Software Development: Principles and Benefits*.*

Pfleeger, S. L., and Atlee, J. M. (2019) Software Engineering: Theory and Practice. 5th edn. Pearson.

Pressman, R. S., and Maxim, B. R. (2020) Software Engineering: A Practitioner’s Approach. 9th edn. McGraw-Hill.

Sommerville, I. (2019) Software Engineering. 10th edn. Pearson.

Stephen, J. and Oriaku, R. (2014) ‘A Comparative Analysis of Software Development Models’, International Journal of Software Engineering*,* 12(3), pp. 45-60.

Stephen, J. and Oriaku, R. (2014) Software Development Models: Principles and Practices. London: Springer.

Voigt, P., and Bussche, A. V. D. (2017) The EU General Data Protection Regulation (GDPR). Springer.