applied system innovation

MDPI

Article

Empowering Healthcare: A Comprehensive Guide to Implementing a Robust Medical Information System—Components, Benefits, Objectives, Evaluation Criteria, and Seamless Deployment Strategies

Ana-Maria Ștefan [1,*], Nicu-Răzvan Rusu [2], Elena Ovreiu [1] and Mihai Ciuc [1]

[1] Faculty of Electronics, Telecommunications and Information Technology, 060042 Bucharest, Romania; elena.ovreiu@upb.ro (E.O.); mihai.ciuc@upb.ro (M.C.)
[2] Independent Researcher, 050688 Bucharest, Romania; rusu.nrazvan@gmail.com
* Correspondence: ana_maria.stefan@stud.fim.upb.ro

Abstract: In the ever-evolving landscape of healthcare, the implementation of a robust medical information system stands as a transformative endeavor. This article serves as a comprehensive guide, delineating the intricate steps involved in deploying an effective medical information system. Delving into the main components that constitute this innovative system, we explore its fundamental architecture and how each element contributes to seamless information flow. The benefits of adopting a medical information system are highlighted, emphasizing improved patient care, streamlined processes, and enhanced decision making for healthcare professionals.

Keywords: medical information system; healthcare technology; digital transformation

Citation: Ștefan, A.-M.; Rusu, N.-R.; Ovreiu, E.; Ciuc, M. Empowering Healthcare: A Comprehensive Guide to Implementing a Robust Medical Information System—Components, Benefits, Objectives, Evaluation Criteria, and Seamless Deployment Strategies. *Appl. Syst. Innov.* **2024**, *7*, 51. https://doi.org/10.3390/asi7030051

Academic Editors: Po-Lei Lee, Chun-Yen Chang, Teen-Hang Meen, Charles Tijus and Kuei-Shu Hsu

Received: 18 April 2024
Revised: 5 June 2024
Accepted: 11 June 2024
Published: 14 June 2024

1. Introduction

In the dynamic landscape of modern healthcare, the intersection of technology and patient care is evolving at an unprecedented pace. At the heart of this transformative journey lies the intricate web of medical information systems (MIS)—a comprehensive framework designed to revolutionize the way medical information is implemented, shared, and utilized. MIS is a term used to describe a comprehensive application that can manage all the information gathered from many sources at different levels of the healthcare organization. This guide embarks on an exploration of MIS, dissecting its modules, deployment strategies, and the pivotal role it plays in shaping the future of healthcare services.

At the forefront of this technological revolution stands the implementation of a robust MIS, a transformative endeavor that holds the potential to reshape the way healthcare is delivered and managed from the implementation of an information system point of view. As healthcare organizations increasingly recognize the pivotal role of information systems in enhancing patient care and operational efficacy, the need for a comprehensive guide to navigate the intricacies of MIS implementation becomes paramount. This article aims to serve as an indispensable resource for healthcare professionals, administrators, and IT specialists embarking on the journey of deploying an MIS. With a focus on providing a holistic understanding, we delve into the main components that constitute an MIS, exploring their fundamental architecture and elucidating how each element contributes to the seamless flow of critical information. Beyond the technical aspects, we underscore the tangible benefits that adopting an MIS brings, emphasizing its role in not only optimizing processes but also empowering healthcare professionals with enhanced decision-making capabilities.

Meticulously outlining the objectives of an MIS within a healthcare setting, we navigate through the key milestones of implementation, from optimizing data management to fostering interoperability. The article also delves into the essential criteria for evaluating

the success of such systems, ensuring alignment with organizational objectives, regulatory standards, and the satisfaction of end-users. As we progress, we will explore the deployment phase, shedding light on the strategies that facilitate the smooth integration of an MIS into the existing healthcare infrastructures. Through a thorough examination of integration considerations, including interoperability with electronic health records and existing systems, this guide aims to equip stakeholders with the knowledge needed to pave the way for a technologically advanced and interconnected healthcare ecosystem.

2. Medical Information System

MIS represents a system that manages information about the health status of patients, including the storage of various types of data, transmission, the monitoring of electronic patient records, and the monitoring of in-hospital operations. In addition to patient information, MIS includes specific activities in the workflows between healthcare service providers and patients to improve patient life, and influence policy development and decision making. Implementing an MIS is no different than implementing an IS. The key is understanding and adapting to the unique workflows of healthcare professionals, which is crucial. MIS should enhance clinical processes, ensuring efficiency and accuracy in patient care.

2.1. Modules of a Medical Information System

A medical information system is typically constructed from multiple interconnected modules that serve specific functions within a medical organization. These modules facilitate the management of patient information, optimize processes, and improve the delivery of medical services. Here, the integration part of development plays a bigger role in implementing MIS vs. IS. While specific modules may vary based on system design and organizational needs, the common modules are presented in Figure 1.

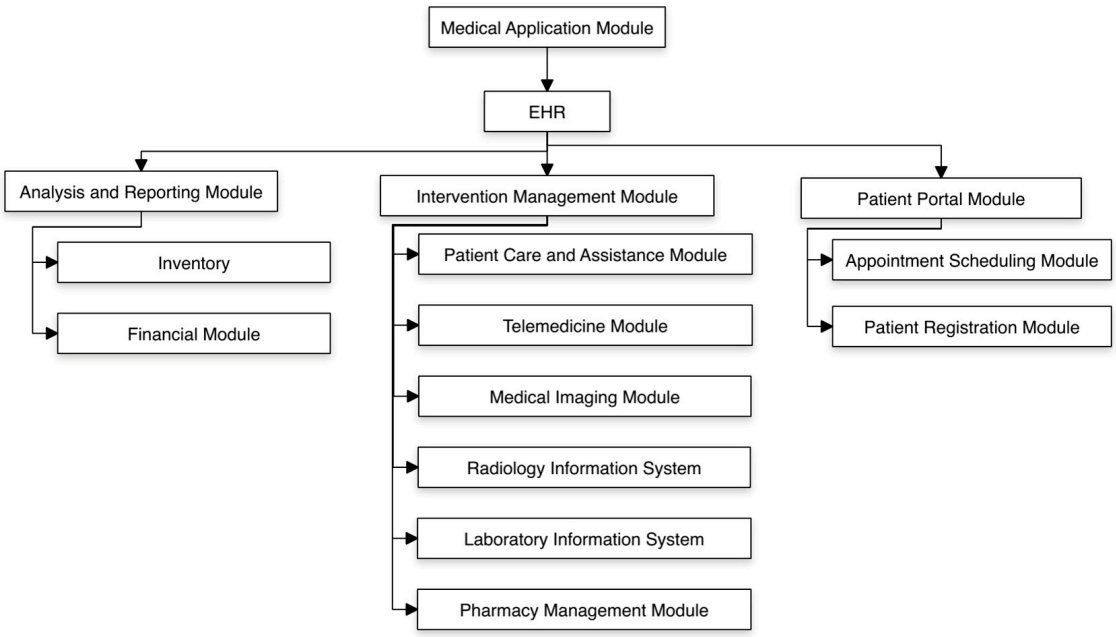

Figure 1. Structure of a medical information system.

- Electronic Health Record: This is the central component, digitally storing and managing patients' medical records. It includes information such as medical history, diagnoses, treatments, medications, laboratory test results, and more.
- Patient Registration Module: Manages patient registration, demographic details, and insurance information.

- Appointment Scheduling Module: Allows the management of patient appointments with healthcare providers, including alerts and notifications.
- Financial Module: Facilitates financial and accounting processes, insurance claims, and financial transactions related to provided services.
- Laboratory Information System: Manages laboratory testing activities, their results, and reporting. Integrates with diagnostic equipment and automates laboratory processes.
- Radiology Information System: Manages radiology and imaging activities, procedures, and results, often integrated with Picture Archiving and Communication Systems (PACS).
- Pharmacy Management Module: Manages medication ordering, dispensing, and inventory control. May include checks for drug interactions and their history.
- Telemedicine Module: Facilitates remote consultations, virtual visits, and patient monitoring through telemetry.
- Patient Care and Medical Assistance Module: Supports healthcare assistance activities, patient care plans, and their documentation.
- Intervention Management Module: Assists in scheduling interventions, preoperative assessments, and coordinating postoperative care.
- Medical Imaging Module: Facilitates the management of medical images, integrates with radiology labs, and enables image visualization and sharing.
- Inventory Module: Facilitates inventory management, expiration date tracking, and supply chain management.
- Analysis and Reporting Module: Facilitates data analysis, reporting, and performance metrics for informed decision making and process evaluation in healthcare.
- Patient Portal Module: Provides patients with access to medical records, appointment scheduling, communication with healthcare providers, and educational resources.
- Mobile Applications Module: Allows access to patient information, appointments, and communication through mobile devices [1,2].

In an MIS, the modules function synergistically to elevate patient care and operational efficiency. A variety of health organizations are using information technology to build a strategic comprehensive information system that can manage information for various levels of decision making to improve the quality of care and efficiency of services. The strategic information system should aim to control resources and enhance clinical and administrative decision making. This information system is designed to provide information in the form of output reports and decision support for the various levels and different functions in the organization. High-quality information is expected to result in a high-quality decision-making process. MIS is crucial for health institutions such as hospitals to provide the best result of care to patients.

Having a computer-based information system has become very important in the health field due to the large amount of data needing to be processed. A computer-based MIS can handle large amounts of data and generate information which can easily be accessed. This can be very helpful for diagnosis and treatments in patient care. MIS can have many functions. One of the most common functionalities is an information system for clinical decision support.

2.1.1. Benefits and Objectives of a Medical Information System

In modern healthcare, technology integration has revolutionized medical information management. At the forefront is the MIS, a framework designed to streamline processes, optimize patient care, and ensure regulatory compliance. This chapter highlights the benefits and objectives of MIS, including enhancing patient safety, improving the quality of care, and promoting seamless communication among healthcare professionals. MIS fosters efficiency, effectiveness, and patient-centeredness in healthcare. Our research [3–10] and implementation projects [11–14] which have identified the primary advantages of MIS for healthcare organizations are presented in Table 1.

Table 1. Advantages of MIS for healthcare organizations.

Category	Details
Efficient Information Management	- The centralized storage and organization of patient records, medical histories, treatment plans, and diagnostic reports. - Reduces paperwork, manual data entry errors, and the duplication of efforts. - Streamlines administrative processes.
Enhanced Patient Care	- Facilitates comprehensive patient data access for informed decisions and personalized care. - Enables faster diagnosis, treatment planning, and coordination among healthcare professionals. - Leads to improved patient outcomes.
Improved Communication and Collaboration	- Enables seamless communication among healthcare teams, including physicians, nurses, specialists, and support staff. - Enhances care coordination and reduces communication gaps. - Fosters interdisciplinary collaboration for better patient care.
Data-Driven Decision Making	- Provides access to real-time and historical patient data for evidence-based decisions and clinical analytics. - Supports clinical decision tools, alerts, and reminders for timely decisions.
Enhanced Patient Safety and Quality of Care	- Reduces medication errors, adverse drug interactions, and medical errors through electronic prescribing and medication reconciliation. - Improves patient safety with access to allergy information, medication history, and clinical guidelines at the point of care.
Cost Reduction and Resource Optimization	- Minimizes redundant tests and administrative overhead, leading to cost savings. - Enables the proactive management of chronic conditions, preventive care, and population health initiatives, reducing long-term healthcare costs.
Patient Engagement and Empowerment	- Provides patient portals, telemedicine, and remote monitoring capabilities. - Empowers patients to access their health information, schedule appointments, request prescription refills, and communicate with healthcare providers online.

Table 1. *Cont.*

Category	Details
Scalability and Flexibility	- Offers scalable and flexible solutions that adapt to evolving healthcare needs, technological advancements, and regulatory changes. - Supports interoperability with other healthcare systems, devices, and data sources for seamless integration and data exchange.
Continuous Improvement and Innovation	- Supports data analytics, research studies, and population health management. - Identifies trends, patterns, and opportunities for improvement in healthcare outcomes.

These objectives collectively contribute to a more effective and impactful healthcare environment. Overall, the benefits of any information system are numerous and contribute to the strategic, operational, and financial success of organizations. They play a vital role in the modern operations of businesses and are essential for maintaining competitiveness in today's digital era [15–17].

2.1.2. Criteria for Evaluating a Medical Information System

The evaluation criteria for an MIS are specifically tailored to the healthcare domain, playing a pivotal role in gauging its effectiveness, efficiency, and overall quality. With a plethora of options available, evaluating and selecting the most suitable MIS requires a thorough understanding of the system's capabilities, functionalities, and alignment with organizational goals. This chapter delves into the essential criteria for evaluating an MIS, providing healthcare stakeholders with a structured framework to assess the effectiveness, reliability, and suitability of potential solutions. From usability and interoperability to security, performance, and cost-effectiveness, each criterion plays a pivotal role in determining the system's ability to meet the diverse needs of healthcare organizations and support the delivery of high-quality patient care. Join us as we explore the key criteria for evaluating an MIS, empowering healthcare professionals to make informed decisions, and select the optimal solution to enhance healthcare delivery and outcomes. Basic and more important fundamental considerations include the following:

- The intuitiveness of the user interface, ease of navigation and task completion for healthcare professionals, and accessibility features for users with diverse needs.
- The comprehensive coverage of essential features such as patient management, scheduling, and clinical documentation. Integration with medical devices, laboratory systems, and other healthcare systems.
- Support for clinical decision support tools and alerts.
- Compatibility with industry standards such as HL7 and FHIR for data exchange. Ability to integrate with EHRs, Health Information Exchanges (HIEs), and other external systems. Seamless interoperability with medical devices and telemedicine platforms.
- The implementation of robust security measures to safeguard patient data and ensure compliance with privacy regulations (e.g., HIPAA and GDPR). The encryption of sensitive information in transit and at rest. Audit trails and access controls to monitor and track user activities.
- The responsiveness and reliability of the system under varying loads. Minimal downtime and fast response times during peak usage periods. Scalability to accommodate growing data volumes and user base.

- Alignment with clinical workflows and practices to minimize disruptions. Seamless integration with existing healthcare processes and systems. Customization options to adapt to specific organizational needs and workflows.
- The accuracy, completeness, and consistency of patient data stored within the system. Data validation mechanisms to prevent the entry of erroneous or duplicate information. Data governance policies and procedures to maintain data integrity over time.
- The availability of comprehensive training resources and user support channels. Timely technical support and assistance for troubleshooting and issue resolution. Continuous updates and enhancements to address user feedback and evolving needs.
- The total cost of ownership, including initial implementation, maintenance, and support. Value proposition in terms of improved efficiency, reduced errors, and enhanced patient outcomes. Return on investment (ROI) in terms of tangible benefits and cost savings over time.
- Feedback from healthcare professionals, administrators, and end-users regarding system usability, functionality, and performance. User satisfaction surveys, focus groups, and usability testing to gather insights for system improvement. Iterative refinement based on user feedback to ensure ongoing alignment with user needs and preferences [18–21].

Evaluation based on these criteria provides a comprehensive assessment of an MIS's effectiveness, suitability, and value in supporting healthcare delivery and improving patient care.

3. Overview of Information Systems

An information system is a combination of software, hardware, and communication networks designed to collect, process, store, and distribute data for use in workflows and to facilitate decision making and process optimization. Data become information when processed to support decision making. Information systems can vary widely in complexity and functionality, ranging from simple systems like personal spreadsheets to complex enterprise systems like Customer Relationship Management (CRM) software or Enterprise Resource Planning (ERP) systems.

3.1. Components of an Information System

From a socio-technical perspective, information systems are composed of several components presented in Figure 2.

- Human Resources—encompassing end-users and anyone else interacting with the system, providing inputs, utilizing system outcomes, and making decisions based on the provided information.
- Equipment—physical devices used for inputting, processing, storing, and extracting data within the system, such as computers, servers, network equipment, storage devices, and peripherals.
- Software—programs, applications, and operating systems enabling users to interact with and manipulate data.
- Data—information collected, processed, and stored by the computer system, which can be structured (organized in a specific format like a database) or unstructured (such as text documents or images).
- Procedures—methodologies and rules governing system usage, management, and information processing, including rules, policies, workflows, and processes dictating activities within an organization.
- Networks—connecting different system components, enabling data transmission between computers and devices, either locally (LAN) or globally (WAN).
- Security—protecting the system against unauthorized access, including authentication, authorization, encryption, firewalls, and other security measures ensuring data confidentiality, integrity, and availability.

- Analysis—mechanisms gathering information about system performance, usage, and effectiveness, aiding in identifying improvement opportunities and adapting the organization to evolving technological needs.
- Control—mechanisms ensuring system functionality efficiency, including monitoring, error detection and resolution, and data consistency maintenance.
- Environment—the context in which the information system operates, influenced by external factors like economic conditions, industry trends, and social factors.

All these components collaborate to create a coherent and functional information system supporting the objectives and goals of an organization. The design and integration of these components are crucial for the system's success in providing accurate, timely, and relevant information to support decision making and operational processes [22].

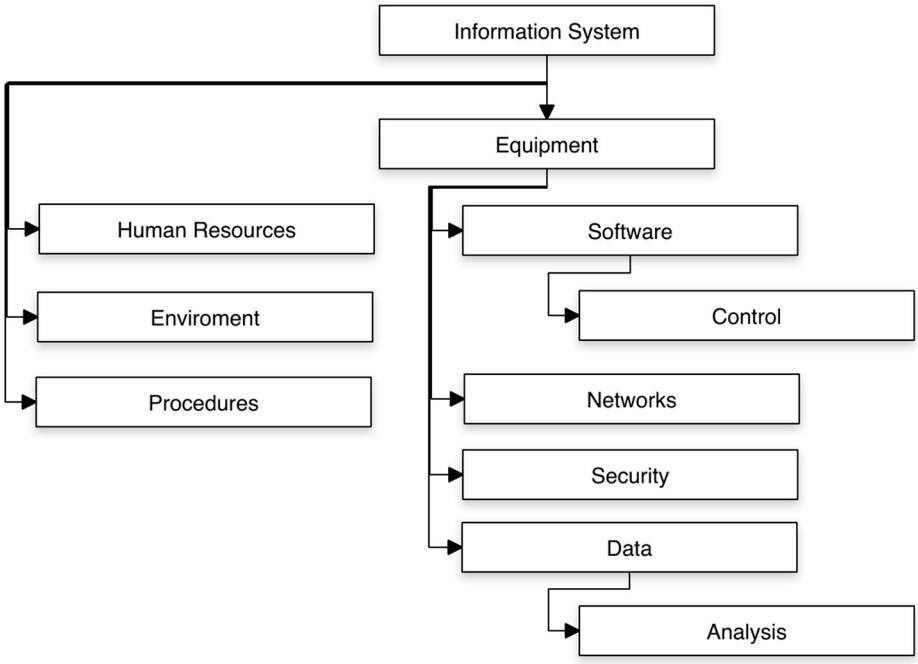

Figure 2. Components of an information system.

3.2. Stages of Implementing an Information System

An information system goes through a series of stages in its implementation to serve its intended objectives. These stages are part of a methodology developed in the 1960s for managing the implementation of complex software projects—the System Development Life Cycle (SDLC). SDLC is a structured and systematic approach to designing, developing, testing, implementing, and maintaining applications or software systems. It outlines the various stages and activities involved in creating and managing software, from the initial concept to the final implementation and ongoing maintenance. SDLC provides a framework that helps ensure the efficient completion of software projects on time and within budget, while also meeting quality and performance requirements. There are different methods and models for implementing SDLC, including the Waterfall model, agile methodologies (such as Scrum and Kanban), and iterative models. Each approach has its own set of principles, practices, and benefits, and organizations can choose the one that best fits the project requirements, team dynamics, and organizational culture. Overall, SDLC provides a structured framework that guides software development projects through a series of well-defined stages, ensuring the systematic and controlled development, testing, and implementation of software. The main stages or phases of SDLC are presented in Figure 4.

I. Planning: This stage involves defining the project scope, objectives, and requirements; identifying resources, budget, and deadlines; creating a plan; and defining roles and responsibilities within the project management team. Understand who

will interact with the system and what their needs and requirements are. This can include end-users, administrators, managers, and other stakeholders.

II. Analysis: In this stage, the requirements and user expectations are collected and documented. It includes project feasibility, technology, resource, and budget analysis, identifying potential risks and challenges, creating usage scenarios, and prioritizing requirements. This document is structured based on UML (Unified Modeling Language), which simplifies the way information systems are modeled. UML identifies various diagrams that need to be elaborated for the implementation of a system and can be grouped according to the system development stages [23,24]. In different stages of the analysis, there are diagrams made to evidentiate the process that is being developed in the information system. The point is that the external activities, information, process, and business flow are all reflected in the information system in the order that is logically performed by the end users. Here are the stages of the analysis process and some simple examples of diagrams drawn to reflect users' interaction with the information system.

a. Analysis:

- Use case diagram represents the application's use cases by the end user. In Figure 3 is a basic example.

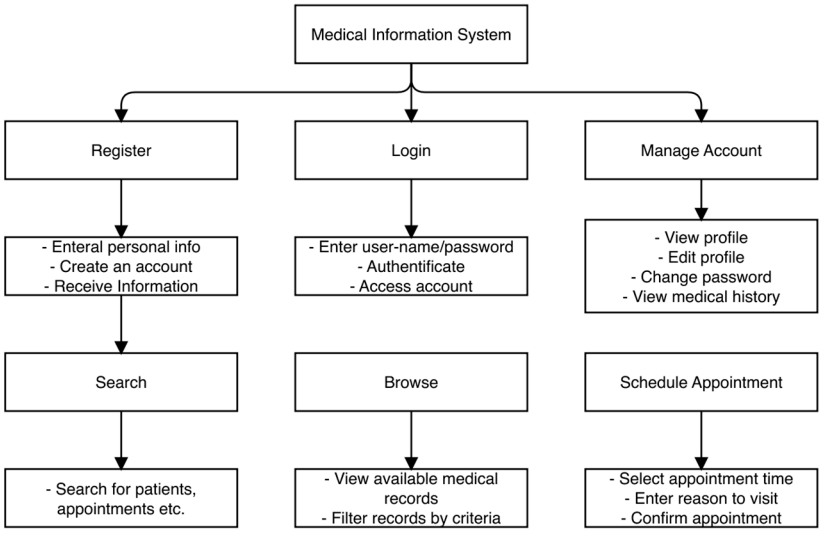

Figure 3. Use case diagram for a medical information system.

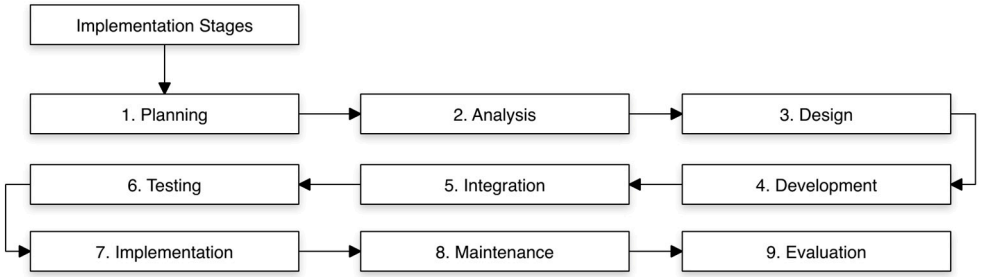

Figure 4. Phases of implementing an information system.

The main actors, including Register, Login, Search, Browse, Manage Account, and Schedule Appointment, are depicted in the diagram. Each actor corresponds to a specific use case, representing the distinct actions or functionalities offered by the system. The Register actor can perform actions such as registering and receiving confirmation. The Login actor can perform actions such as logging in and accessing the account. The Search

actor can perform actions such as searching for patients or appointments and the Browse actor can view available medical records and filter records. The Manage Account actor can view and edit profiles, change passwords, and view medical history. Lastly, The Schedule Appointment actor can select appointment times, enter reasons for visits, and confirm appointments. This diagram presents an overview of the medical information system's functionality and the interactions among various actors and use cases.

- Activity diagram illustrates the activities undertaken in a workflow. In Figure 5 is a basic example of a medical appointment booking system.

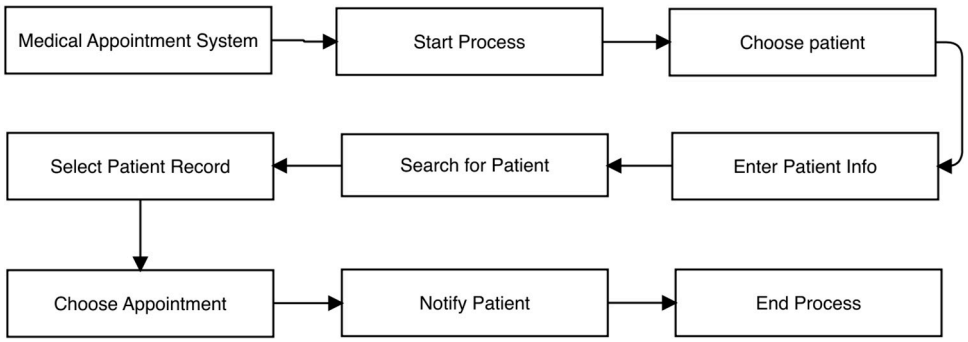

Figure 5. Activity diagram for medical information system.

In this activity diagram:

- The process starts with the "Start Process" activity.
- The user chooses the patient for whom they want to book an appointment.
- The user enters patient information or searches for an existing patient record.
- The system searches for the patient's record.
- The user selects the patient's record from the search results.
- The user chooses the appointment date and time.
- The user confirms the appointment.
- The system notifies the patient about the appointment.
- The process ends.

Each activity represents a specific task or action within the system, and the arrows show the flow of control from one activity to the next. This activity diagram provides a visual representation of the steps involved in booking a medical appointment within the system.

- Data flow diagram represents the flow of data within the system (as presented in Figure 6), illustrating how data move from input to processing and output. DFDs can be used to model both high-level and detailed data processes.

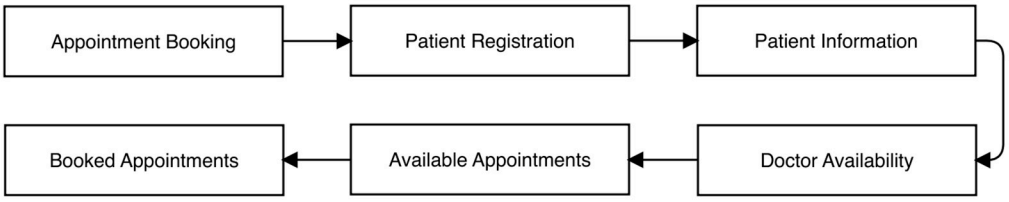

Figure 6. Data flow diagram for an appointment booking in a medical information system.

In this DFD:

- Appointment Booking—the process represents the main function of the system, where users can book appointments.
- Patient Registration—the process handles the registration of new patients.

- Patient Information—the process manages patient information, including updating and retrieving patient records.
- Doctor Availability—the process deals with the availability of doctors and their schedules.
- Available Appointments—the process determines the available appointments based on doctor availability and existing bookings.
- Booked Appointments—the process manages the booked appointments, including storing and updating appointment details.

The arrows represent the flow of data between the processes. For example, data flow from the "Appointment Booking" process to the "Patient Registration" process when a new patient registers for an appointment. Similarly, data flow from the "Available Appointments" process to the "Booked Appointments" process when an appointment is booked. This DFD provides a high-level overview of the data flow and the processes involved in the medical appointment booking system, helping to understand how information moves through the system.

b. Design:

- Class diagram identifies the objects to be used and their interaction. Example of this kind of diagram is presented in Figure 7.

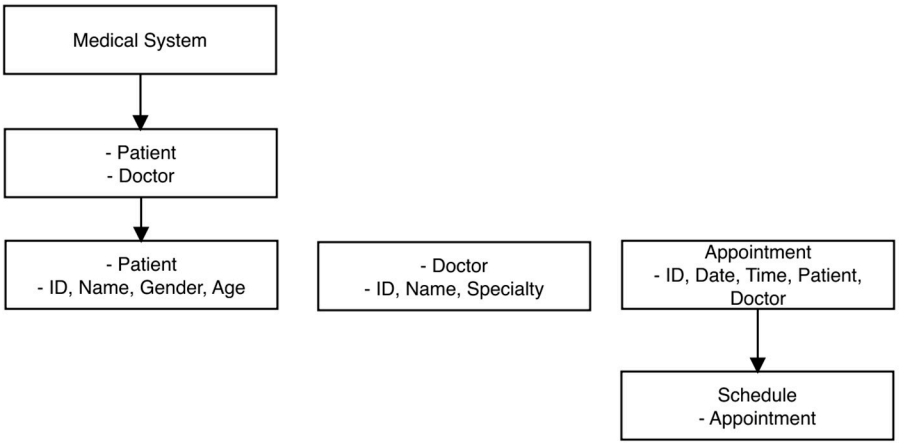

Figure 7. Class diagram for an appointment booking in a medical information system.

In this class diagram:

- Medical system represents the main class that holds the collections of patients and doctors.
- Patient represents a patient with attributes such as id, name, gender, and age.
- Doctor represents a doctor with attributes such as id, name, and specialty.
- Appointment represents an appointment with attributes such as id, date, time, patient, and doctor.
- Schedule represents a schedule that holds a collection of appointments.

The arrows indicate associations between classes. For example:

- Each medical system can have multiple patients and doctors.
- Each appointment is associated with one patient and one doctor.
- The schedule class aggregates a collection of appointments.

This class diagram provides a basic representation of the main classes and their associations in the medical appointment booking system. Depending on the requirements and complexity of the system, additional classes and relationships may be needed.

- Package diagram—groups multiple classes creating subsystems that can be developed in parallel. Diagram example is presented in Figure 8.

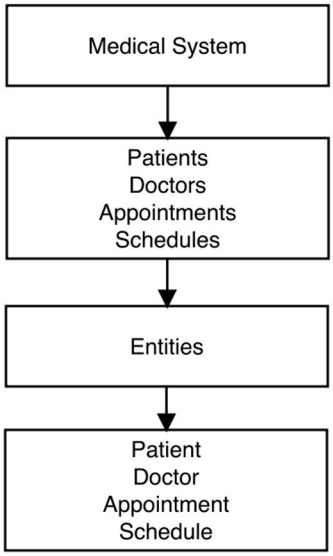

Figure 8. Package diagram for an appointment booking in a medical information system.

In this package diagram:

- Medical system represents the main package or module of the system. It contains components related to the overall functionality of the system, such as patients, doctors, appointments, and schedules.
- Entities represents a sub-package within the medical system package. It contains classes representing the main entities or domain objects of the system, such as patient, doctor, appointment, and schedule.

The arrows indicate dependencies between packages, with components inside the medical system package depending on the entities package. This package diagram provides a high-level view of how the components of the medical appointment booking system are organized into packages or modules, helping to manage the complexity of the system and facilitate maintenance and development.

- State diagram describes the states through which a particular object passes. Diagram example is presented in Figure 9.

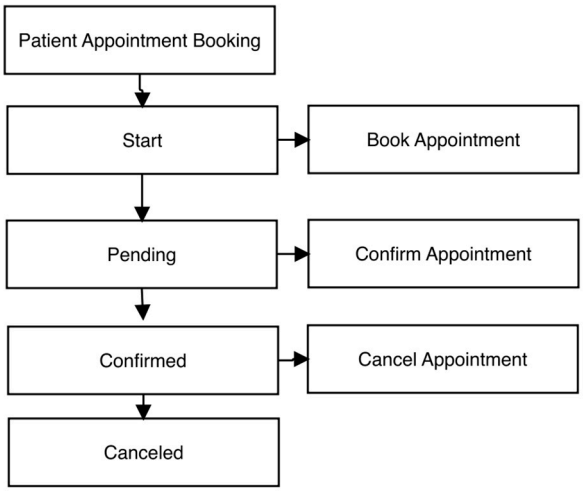

Figure 9. State diagram for an appointment booking in a medical information system.

In this state diagram:

- Patient Appointment Booking represents the overall process of a patient booking an appointment.

- Start represents the initial state when the booking process begins.
- Pending represents the state when the patient has selected an appointment slot but has not confirmed it yet.
- Confirmed represents the state when the patient has confirmed the appointment.
- Canceled represents the state when the patient cancels the appointment.

Transitions:

- Book Appointment: Transition from the start state to the Pending state when the patient selects an appointment slot.
- Confirm Appointment: Transition from the Pending state to the confirmed state when the patient confirms the appointment.
- Cancel Appointment: Transition from the Pending or confirmed state to the Canceled state when the patient cancels the appointment.

This state diagram provides a visual representation of the different states a patient's appointment booking can be in and the transitions between those states. It helps to understand the lifecycle of an appointment booking process within the system.

- Entity Relationship diagram presents entities (such as tables in a database), their attributes, and the relationships between entities. ERD is commonly used in database design and data modeling. Diagram example is presented in Figure 10.

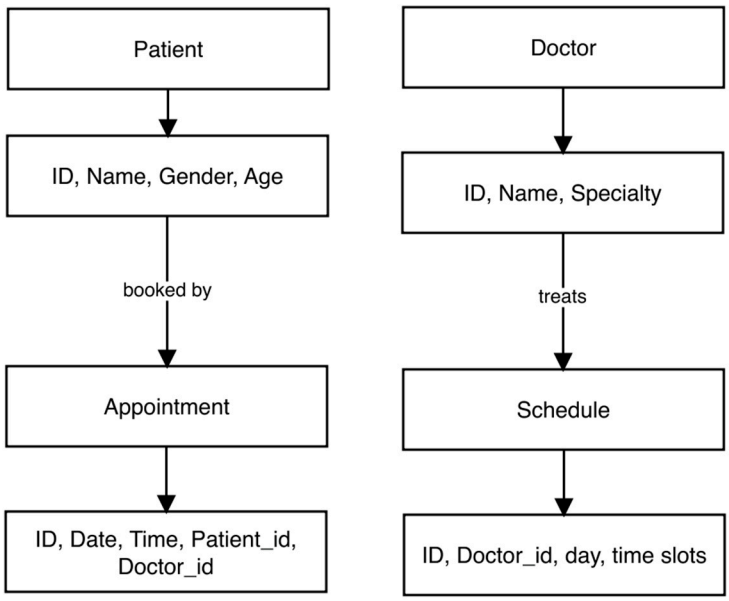

Figure 10. ERD for an appointment booking in a medical information system.

In this ERD:

- Patient and doctor are entities representing patients and doctors, respectively. They have attributes such as id, name, gender, age, and specialty.
- Appointment represents appointments booked by patients with doctors. It has attributes such as id, date, time, and foreign keys referencing the patient and doctor entities.
- Schedule represents the schedules of doctors. It has attributes such as id, doctor id, day, and time slots.

Relationships:

- Appointment is associated with patient and doctor entities through the patient id and doctor id, representing the patient who booked the appointment and the doctor involved in the appointment.
- Doctor is associated with schedule through the doctor id, representing the schedule of each doctor.

This ERD provides a visual representation of the entities involved in the medical appointment booking system and their relationships. It helps to understand the structure of the database schema and how the different entities are connected.

c. Implementation:

- Component diagram illustrates the system's modules and their interaction. Diagram example is presented in Figure 11.

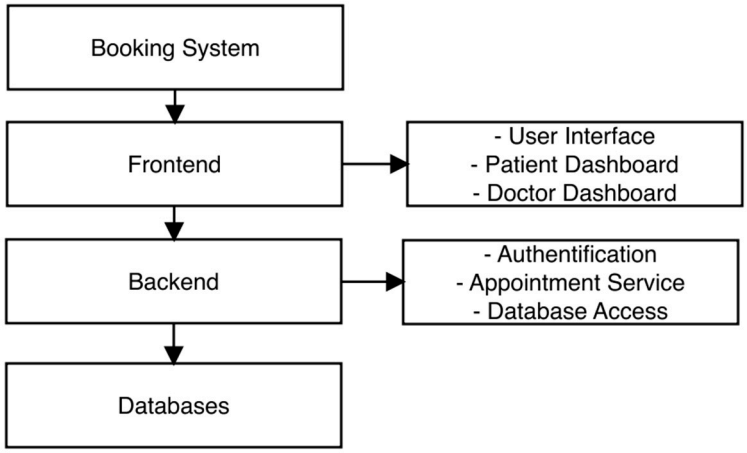

Figure 11. Component diagram for an appointment booking in a medical information system.

In this component diagram:

- Medical appointment booking system represents the main system.
- Frontend represents the user-facing components of the system. It includes modules such as user interface, patient dashboard, and doctor dashboard.
- Backend represents the server-side components of the system. It includes modules such as authentication, appointment service, and database access.
- Database represents the database used by the system to store data related to patients, doctors, appointments, etc.

Relationships:

- Frontend communicates with backend to request and receive data, perform authentication, and manage appointments.
- Backend interacts with the database to store and retrieve data related to patients, doctors, appointments, etc.

This component diagram provides a high-level overview of the main components and their interactions in the medical appointment booking system. It helps to understand the system's architecture and how different parts of the system work together to achieve its functionality.

- Sequence diagram shows interactions and messages exchanged between different objects or components in a system over time. They are often used to model system behavior and communication. Diagram example is presented in Figure 12.

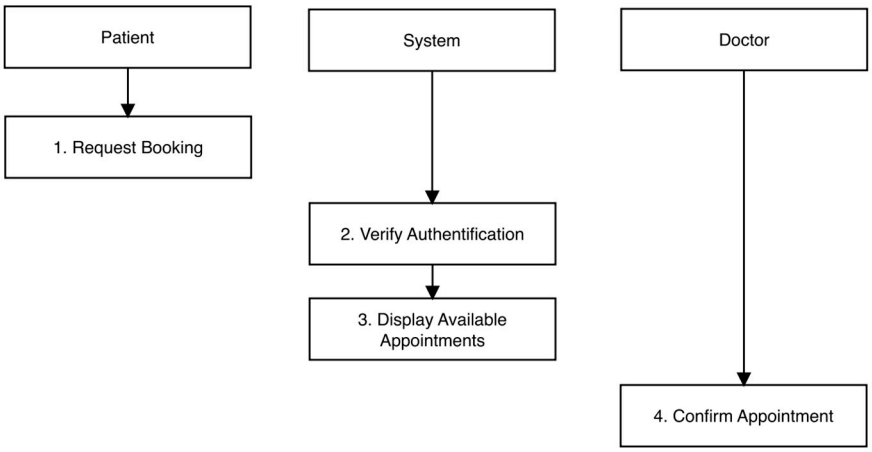

Figure 12. Sequence diagram for an appointment booking in a medical information system.

In this sequence diagram:

- Patient—initiates the booking process by sending a request for booking.
- System—represents the medical appointment booking system.
- Doctor—the doctor object in the system.

Sequence of Events:

1. The patient initiates the booking process by sending a request for booking to the system.
2. The system verifies the authentication of the Patient.
3. The system displays available appointments to the Patient.
4. The patient confirms the appointment.
5. The system confirms the appointment with the Doctor.

This sequence diagram illustrates the interaction between the patient, system, and doctor objects during the appointment booking process in a medical appointment booking system.

- Design—this stage involves creating a detailed architecture of the system based on requirements. It includes developing system architecture, technical specifications, and system components, designing the user interface, designing the database, establishing technical infrastructure, and workflow design. Ensuring that a medical information system fulfills the requirements of healthcare professionals, upholds patient confidentiality, and adheres to regulatory standards is paramount. During our extensive research, we came across some key design points to take into consideration in implementing information systems:
    - During the design phase, the foremost consideration should be the needs and preferences of both the end users and healthcare professionals. Interfaces should be intuitive, clear, and efficient to facilitate ease of use.
    - Creating a navigational structure that is logical and user-friendly is paramount, enabling users to swiftly locate and access relevant features and information. A hierarchical organization of data aids in navigation.
    - Developing a medical information system necessitates adherence to healthcare privacy regulations such as HIPAA or GDPR to safeguard against unauthorized access to sensitive data. Robust security measures like approval processes, access limitations, multi-factor authentication, and encryption are essential.
    - Seamless integration with other medical equipment, systems, and data sources is crucial for a medical information system, fostering interoperability and smooth data exchange through standardized formats and protocols.
    - A medical information system must possess scalability to accommodate increasing data volumes, users, and organizational changes, ensuring a responsive,

dependable, and efficient system that can be easily tailored and adjusted as required.
    - Incorporating features and technologies for clinical decision support aids medical professionals in making well-informed decisions by providing access to the medical literature, guidelines, databases, and decision support algorithms.
    - Facilitating system use and improving accessibility for users can be achieved through mobile access via browsers or mobile applications, enabling remote and on-demand access with interfaces optimized for various devices and screen resolutions.
    - Allowing users to customize the system to suit diverse workflows, organizational needs, and preferences encourages system adoption. Features like customizable dashboards and interface displays enhance usability.
    - Offering thorough documentation through detailed materials, user manuals, and video tutorials maximizes system utility and streamlines the onboarding process for new team members. Specialized expertise and support aid users in familiarizing themselves with the system through periodic training sessions on both existing and new features.
    - Leveraging feedback from users, stakeholders, and usability tests is vital for iteratively enhancing the system during the design phase. Monitoring system performance, usage patterns, and user feedback helps pinpoint areas for improvement.
- Development: This stage involves writing, testing, and optimizing code based on design specifications, developing databases and creating functionalities, and developing and integrating different software modules and components together while ensuring compliance with standards and best practices [19]. Development plays a critical role in ensuring the creation of a reliable, secure, and efficient software solution tailored to the needs of healthcare professionals while adhering to industry regulations. The agile development methodology presents an ideal approach, employing frameworks such as Scrum or Kanban in the developmental phase to facilitate iterative progress, thereby accommodating changes in requirements. Establishing conventions and naming standards, as well as ensuring uniform code formatting, is imperative for streamlining the process, enhancing code comprehension, and simplifying updates. Conducting regular code reviews serves as another means to uphold code quality and mitigate potential errors. Additionally, employing logging and monitoring tools enables the real-time tracking of system activities, facilitating the prompt identification of errors and performance issues.
- Integration: From the perspective of information systems, integration refers to the process of connecting, synchronizing, and coordinating various components, modules, applications, or information systems to work together as a unified and efficient entity. The goal of integration is to ensure the uninterrupted and coherent flow of data and information between the various components of an information ecosystem. We established a checklist to ensure seamless communication and interoperability between different components, systems, and data sources within the healthcare environment:
    - Define standardized interfaces (e.g., RESTful APIs and SOAP services) for communication between different components of the system. This promotes interoperability and allows for seamless integration with external systems.
    - Adopt a service-oriented architecture approach, where functionalities are modularized into independent services that communicate through well-defined interfaces. This facilitates reusability, scalability, and flexibility in integrating new features or external systems.
    - Utilize asynchronous communication patterns (e.g., message queues and publish/subscribe) for integrating components that perform long-running tasks or need to handle high volumes of requests. This helps decouple components and improves system responsiveness.

- Consider implementing an event-driven architecture where the components react to events or messages asynchronously. This enables loosely coupled integration and allows for real-time processing and reaction to changes in the system.
- Define clear data integration strategies for synchronizing data between different systems or databases. Use tools and techniques such as extract/transform/load processes, data replication, and data synchronization to ensure the consistency and integrity of data across the system.
- Implement robust error-handling mechanisms and resilience patterns (e.g., retries and circuit breakers) to handle failures gracefully during integration. This ensures fault tolerance and maintains system availability and reliability.
- Ensure that integration points are secured using appropriate authentication, authorization, and encryption mechanisms. Apply security best practices such as HTTPS, OAuth, and JWT to protect sensitive data and prevent unauthorized access.
- Implement API management solutions to manage and monitor APIs exposed by the system, including versioning, rate limiting, and analytics. This provides visibility and control over API usage and ensures performance and compliance with service level agreements (SLAs).
- Develop comprehensive integration tests to verify the correctness and reliability of integration points. Test various scenarios, including success cases, error conditions, and edge cases, to ensure robustness and compatibility across different environments.
- Document integration points, protocols, and data formats comprehensively to guide developers and maintainers in integrating with the system. Establish governance policies and processes to manage integration contracts, versioning, and change management effectively.
- Implement monitoring and logging mechanisms to track integration performance, detect anomalies, and troubleshoot issues in real-time. Monitor key metrics such as response times, error rates, and throughput to ensure optimal system performance.
- Design well-defined and documented APIs to enable communication and data exchange between the medical information system and other systems. Document API endpoints, request/response formats, authentication mechanisms, and usage guidelines comprehensively to facilitate integration for developers.
- Ensure compatibility and integration with existing EHR systems to facilitate the seamless exchange of patient data and clinical information.
- Support integration with medical devices such as monitors, sensors, and wearable devices to capture and transmit patient data directly into the medical information system. Implement standard communication protocols (e.g., DICOM for medical imaging) and device-specific interfaces to ensure compatibility and data accuracy.
- Implement robust authentication and authorization mechanisms to control access to sensitive healthcare data and ensure data privacy and security. Utilize industry-standard authentication protocols (e.g., OAuth 2.0) and access control mechanisms to authenticate users and authorize access to protected resources.
- Perform data mapping and transformation to harmonize data formats and schemas between different systems and sources during integration. Define mappings between data elements, fields, and terminology standards to ensure consistency and interoperability across integrated systems.
- Implement error handling mechanisms to capture and manage integration errors, ensuring the reliability and resilience of integration processes. Log integration events, messages, and exceptions for auditing, troubleshooting, and monitoring purposes, enabling timely resolution of issues.
- Conduct thorough integration testing to validate the functionality, performance, and reliability of integration interfaces and processes. Test data exchange, mes-

sage formats, error handling, and security controls to verify interoperability and compliance with integration requirements.
- Document integration workflows, protocols, and configurations to guide developers, administrators, and users in integrating and utilizing the medical information system. Provide training and resources to stakeholders involved in integration activities to ensure the effective implementation and operation of integrated solutions.

There are several aspects of integration from the perspective of information systems that would be a good practice to be treated separately like data integration, application integration, technological, and hardware integration. We will delve into describing the methods to create proficient procedures to ensure seamless integration between the different types of systems and data.

Data integration involves combining and synchronizing data from different sources to provide a complete and accurate overview. Data integration may involve format conversions, data cleaning, duplicate removal, and error correction, ensuring that the information is consistent and useful for users. Data integration in a medical information system involves combining and synchronizing data from various sources such as electronic health records, laboratory systems, imaging systems, and administrative systems. In our research and practice with MIS systems, we encountered that is very important to establish procedures to ensure a robust integration like the following:

- Establish data governance policies and procedures to ensure data quality, consistency, and security throughout the integration process. Define the data ownership, access controls, and data stewardship roles to govern data across the system.
- Standardize data formats, vocabularies, and terminologies to ensure consistency and interoperability across different systems and data sources. Adopt industry standards such as HL7, FHIR, DICOM, and LOINC for healthcare data exchange.
- Implement real-time data integration mechanisms to capture and process data updates as they occur. Utilize message queues, event-driven architectures, or change data capture techniques to propagate data changes in real-time across the system.
- Employ batch data processing techniques for integrating large volumes of data from disparate sources. Schedule regular batch jobs or data pipelines to extract, transform, and load data into the target system incrementally.
- Implement data validation and cleansing routines to ensure the accuracy, completeness, and consistency of integrated data. Validate incoming data against predefined rules, perform data deduplication, and handle data quality issues using data quality tools.
- Establish a master data management strategy to maintain a single, authoritative source of truth for critical data entities such as patients, providers, and procedures. Implement tools and processes to manage master data across the organization.
- Implement robust data security and privacy measures to protect sensitive health information during integration. Encrypt data in transit and at rest, enforce access controls, and comply with regulatory requirements such as HIPAA to ensure patient privacy and confidentiality.
- Implement data auditing and logging mechanisms to track data lineage, changes, and access history for compliance and accountability purposes. Log data integration activities and monitor data access to detect unauthorized or suspicious activities.
- Optimize data integration processes for performance and scalability to handle large volumes of data efficiently. Tune database configurations, optimize query performance, and leverage caching mechanisms to improve data processing speed and throughput.

Application integration focuses on connecting and interoperating various software applications; involves connecting and interoperating various software applications to streamline workflows, share data, and improve efficiency across the healthcare ecosystem; and enables the sharing of data and functionalities between applications, making it possible to achieve complex workflows that span multiple applications. In our latest study [12], we introduce an application integration aimed at streamlining the workflow of medical

professionals and providing support in the diagnostic process. This is achieved through the integration of two platforms. There are several steps to follow to ensure a good integration between multiple applications, starting with identifying integration points—the key applications and systems within the medical information system landscape that need to be integrated. This may include EHR systems, LIS, PACS, pharmacy systems, billing systems, and HIE or middleware solutions to facilitate application integration. These tools provide pre-built connectors, message routing, and transformation capabilities to simplify integration tasks and reduce development effort.

Technological integration refers to connecting different technologies, platforms, and infrastructures to work together. For example, technological integration may involve connecting a local system with a cloud computing service or integrating a mobile system with a centralized system. Technological integration in a medical information system involves incorporating various technologies to create a comprehensive and interoperable healthcare solution:

- Implement an API-driven architecture to expose and consume services, enabling flexible integration with internal and external systems. APIs should be well-documented, versioned, and secured to ensure smooth integration with third-party applications and devices.
- Leverage cloud computing platforms such as AWS (Amazon Web Services), Azure, or Google Cloud Platform to host and scale your medical information system. Cloud-based solutions offer scalability, flexibility, and cost-effectiveness, allowing for easy integration with other cloud services and applications. In our recent studies, we tried leveraging cloud computing platforms in the integration process and also made a comparison evaluating some of the available tools to create a model for image classification [12].
- Adopt a microservice architecture, where the system is composed of loosely coupled, independently deployable services. Microservices promote agility, scalability, and resilience, facilitating integration with third-party services and enabling the rapid development and deployment of new features.
- Integrate IoT (Internet of Things) devices such as wearable sensors, remote monitoring devices, and smart medical devices into the medical information system. IoT integration enables real-time data collection, remote patient monitoring, and proactive healthcare interventions, enhancing patient care and treatment outcomes.
- Incorporate big data analytics capabilities into the medical information system to analyze large volumes of healthcare data and derive actionable insights. Use technologies such as Hadoop, Spark, and Elasticsearch to process, analyze, and visualize healthcare data, enabling data-driven decision making and predictive analytics.
- Integrate AI and ML algorithms into the medical information system to automate tasks, improve diagnostic accuracy, and personalize patient care. AI-powered applications can assist healthcare professionals in medical image analysis, clinical decision support, and patient risk stratification, enhancing overall system capabilities.
- Explore the use of blockchain technology to enhance data security, integrity, and privacy in the medical information system. Blockchain can be used to secure patient health records, track pharmaceutical supply chains, and facilitate the secure sharing of sensitive healthcare data between stakeholders while ensuring data immutability and auditability.
- Integrate NLP technologies into the medical information system to extract insights from unstructured clinical notes, patient records, and the medical literature. NLP-based applications can assist in clinical documentation, medical coding, and information retrieval, improving efficiency and accuracy in healthcare delivery.
- Integrate telemedicine and telehealth platforms into the medical information system to enable virtual consultations, remote monitoring, and telemedicine-enabled care delivery. Telehealth integration enhances access to healthcare services, improves patient engagement, and supports remote collaboration among healthcare providers.

Hardware integration focuses on connecting different hardware devices, such as servers, networks, sensors, and communication equipment, to ensure the smooth operation of the information system. You should take into consideration the following:

- Ensure seamless connectivity between medical devices and the information system by supporting industry-standard communication protocols such as Bluetooth, Wi-Fi, USB, and serial interfaces. Implement device drivers and protocols necessary to interface with a wide range of medical devices, including patient monitors, infusion pumps, and diagnostic equipment.
- Adhere to interoperability standards such as IEEE 11073 (health informatics—personal health device communication), DICOM Communications in Medicine, and HL7 to ensure compatibility and interoperability between medical devices and the information system. Implement standard data formats and protocols to facilitate seamless data exchange between devices and the system.
- Deploy IoT gateways or edge computing devices to aggregate, preprocess, and transmit data from medical devices to the information system. IoT gateways provide local intelligence and connectivity capabilities, allowing for real-time data processing, filtering, and analysis at the network edge before transmitting data to the central system.
- Develop software modules or middleware components to acquire, process, and normalize data from different types of medical devices. Implement data parsing, validation, and transformation routines to convert raw device data into standardized formats suitable for storage and analysis within the information system.
- Design the hardware integration architecture to be scalable and resilient to accommodate a large number of connected devices and handle fluctuations in data volume and traffic. Implement load balancing, fault tolerance, and redundancy mechanisms to ensure the high availability and reliability of the hardware integration infrastructure.
- Enable the remote monitoring and management of connected medical devices through centralized management consoles or dashboards. Implement remote configuration, firmware updates, and diagnostic tools to facilitate proactive maintenance and troubleshooting of devices deployed across multiple locations.
- Integrate medical device data seamlessly with EHR systems to provide clinicians with comprehensive patient information and enable data-driven decision making. Implement bidirectional data exchange capabilities to synchronize patient data between medical devices and EHRs in real-time.
- Testing: In this stage, various types of tests are conducted, issues are identified and rectified, and it is verified that the solution or functionality meets the specified requirements. Testing is conducted based on the initial design document, which may contain scenarios in which the system must respond correctly and reliably to achieve the final goal. This stage is considered critical in the software development cycle. Testing comes in various forms:
    - VI.1 Unit testing is a method of testing individual code units, components, or modules to verify their functionality. Unit testing in a medical information system is crucial for ensuring the correctness, reliability, and robustness of individual software components.
    - VI.2 Integration testing refers to the interface between two systems or modules and how they function individually and together. Integration testing ensures that the individual components of a system work together as expected when integrated.
    - VI.3 System testing is the testing of the fully integrated system to evaluate its conformity with requirements. System testing ensures that the entire medical information system behaves as expected when all the components are integrated and tested together. Usually, the overall testing of the system is performed based on the analysis documentation.

- VI.4 Performance and stress testing evaluates the speed, responsiveness, scalability, and stability of the system under various conditions, such as different loads or user counts. It evaluates the responsiveness, scalability, and stability of the medical information system under various load conditions.
- VI.5 Security testing aims to identify vulnerabilities and weaknesses in the system's security mechanisms, including data protection and access control. It is essential for ensuring the confidentiality, integrity, and availability of the medical information system and safeguarding sensitive healthcare data.
- VI.6 Usability testing focuses on user experience, ensuring that the system is user-friendly and intuitive. Usability testing evaluates the medical information system's ease of use, efficiency, and user satisfaction.
- VI.7 Compatibility testing verifies that the system functions correctly on different devices, operating systems, browsers, and network environments. Compatibility testing ensures that the medical information system functions correctly across different environments, platforms, and configurations.
- VI.8 Acceptance testing, also known as user acceptance testing (UAT), involves testing the system with real users to ensure it meets their requirements and expectations. Acceptance testing ensures that the medical information system meets the specified requirements and satisfies user needs before deployment.
- VI.9 Exploratory testing explores the system, often without predefined test cases, to discover defects that cannot be captured by predefined tests. It is a more non-conventional testing approach but is a dynamic and interactive approach to software testing where testers explore the medical information system, learn its functionalities, and design tests on the fly based on their observations and intuition.
- VI.10 Alpha testing is conducted by the internal development team before releasing the system to a selected group of external users.
- VI.11 Beta testing is conducted by a larger group of external users testing the system in a real environment before the official release. Beta testing involves releasing the medical information system to a limited group of external users for real-world testing in a controlled environment [25–28].

The in-depth explanations of the various testing strategies to consider can be found in Appendix A.

- Implementation: Once testing is completed, and everything operates according to requirements, the system can be put into use. Developments are transferred to the working environment that end users will utilize, often referred to as "Production". Configuration testing ensures that the medical information system functions correctly across different configuration settings and scenarios. In the system implementation, besides technical steps, there are also the following:
  a. Data Analysis
     - Gain a comprehensive understanding of the data sources available within the medical information system, including patient records, clinical notes, diagnostic reports, laboratory results, and administrative data. Explore data structures, formats, and quality characteristics to inform analysis approaches.
     - Preprocess and clean the data to ensure accuracy, consistency, and completeness before analysis. Handle missing values, outliers, and inconsistencies appropriately using data-cleaning techniques such as imputation, normalization, and outlier detection.
     - Evaluate the performance of predictive models using evaluation metrics such as accuracy, precision, recall, F1-score, ROC curves, and confusion matrices. Assess model generalization, robustness, and reliability through cross-validation and validation techniques [12].

- Interpret model results, findings, and insights in the context of clinical practice, healthcare policies, and patient outcomes. Communicate analysis results effectively using visualizations, charts, dashboards, and reports to facilitate understanding and decision making.

b. Data Import (often involving data processing)

- Define a clear mapping between the source data and the target data structure within the medical information system. Identify corresponding fields, attributes, and data formats to ensure accurate data transfer.
- Implement data validation checks to verify the integrity, consistency, and quality of imported data. Validate the data against predefined rules, constraints, and standards to detect errors and inconsistencies before importing.
- Preprocess and clean the data before importing to remove duplicates, errors, and inconsistencies. Handle missing values, format discrepancies, and data anomalies using data-cleansing techniques such as deduplication, normalization, and error correction.
- Transform the source data into the required format, structure, and encoding for import into the medical information system. Convert data types, adjust field lengths, and apply necessary transformations to align the source data with the system requirements.
- Import the data in batches or chunks to manage large volumes of data efficiently. Implement batch-processing techniques to streamline data import, monitor progress, and handle errors or exceptions gracefully.
- Support incremental data import to update the existing data with new or modified records. Identify changes since the last import, such as additions, updates, or deletions, and synchronize the data accordingly to maintain data consistency.
- Automate the data import process using scripts, workflows, or integration tools to reduce manual effort and improve efficiency. Schedule regular import tasks, monitor import jobs, and automate error handling to streamline data management tasks.
- Log import activities, errors, and status updates to track import progress and troubleshoot issues effectively. Maintain detailed import logs containing timestamps, import parameters, error messages, and corrective actions taken during the import process.
- Test data import procedures thoroughly in a controlled environment before production deployment. Conduct unit tests, integration tests, and end-to-end tests to validate data import functionality, performance, and reliability.

c. Report Generation is crucial for presenting data, insights, and findings derived from the medical information system in a clear, concise, and actionable format. During theprocess of implementing a system, we created a checklist in order not to miss important information that end-users will need after Go-Live:

- Clearly define the purpose, audience, and content requirements for each report. Identify the key stakeholders, their information needs, and the specific insights or metrics they require from the report.
- Develop standardized report templates with consistent layouts, formats, and styles. Define the report sections, headers, footers, and visual elements to maintain a cohesive look and feel across all the reports.
- Choose appropriate data visualization techniques such as charts, graphs, tables, and dashboards to convey information effectively. Select the visualizations that best represent the data and insights being presented in the report.

- Verify the accuracy, completeness, and integrity of the data used in report generation. Validate the data sources, perform data cleansing, and conduct quality checks to ensure that the reported information is reliable and trustworthy.
- Provide options for customizing the reports based on user preferences, including date ranges, filters, and parameters. Allow users to tailor the reports to their specific needs and requirements for personalized insights.
- Include relevant metrics, KPIs, and performance indicators in the report to track progress, monitor trends, and assess outcomes. Highlight key findings and actionable insights to guide decision making and inform strategic initiatives.
- Organize the report content logically and hierarchically to facilitate comprehension and navigation. Structure the reports with clear headings, subheadings, and sections to guide readers through the information presented.
- Provide context and interpretation for the data presented in the report to help the users understand its significance and implications. Offer explanations, commentary, and insights to clarify complex concepts and draw meaningful conclusions.
- Use clear, concise, and jargon-free language in the report text to ensure readability and comprehension. Avoid technical terms or acronyms that may be unfamiliar to the target audience.
- Ensure that the reports are accessible to all users, including those with disabilities or special needs. Use accessible design principles, provide alternative text for images, and support assistive technologies to enhance accessibility.
- Review the reports thoroughly for accuracy, consistency, and relevance before distribution. Validate the report content against the source data, conduct peer reviews, and solicit feedback from stakeholders to ensure quality and reliability.
- Secure report distribution channels to protect sensitive information and maintain confidentiality. Implement access controls, encryption, and authentication mechanisms to restrict access to authorized users only.
- Document report generation processes, methodologies, and assumptions in detailed documentation. Maintain version control of the report templates, data sources, and configurations to track changes and ensure consistency over time.

d. User training is crucial for ensuring that individuals who interact with the medical information system have the necessary knowledge and skills to use it effectively.

- Identify the training needs of different user groups, including healthcare professionals, administrators, support staff, and end-users. Determine their proficiency levels, learning preferences, and specific training requirements.
- Clearly define the objectives, goals, and learning outcomes of the training program. Align the training objectives with system functionalities, user roles, and organizational goals to ensure relevance and effectiveness.
- Develop training content tailored to the needs and roles of different user groups. Customize training materials, modules, and exercises to address specific job responsibilities, workflows, and usage scenarios within the medical information system.
- Use interactive training methods such as hands-on exercises, simulations, case studies, and role-playing activities to engage the participants actively. Encourage active participation, collaboration, and knowledge sharing during training sessions.

- Incorporate multimedia resources such as videos, tutorials, demonstrations, and interactive e-learning modules into the training program. Provide diverse learning resources to accommodate different learning styles and preferences.
- Organize the training sessions into a structured curriculum with clear learning objectives, session agendas, and learning pathways. Sequence the training modules logically, starting with basic concepts and progressing to more advanced topics.
- Using Train-the-Trainer approach implies to train designated trainers or super-users within the organization to deliver training sessions to end-users. Empower the trainers with in-depth knowledge of the medical information system, effective teaching techniques, and facilitation skills to ensure consistent and quality training delivery.
- Provide opportunities for hands-on practice and experimentation with the medical information system in a simulated or training environment. Allow users to explore system functionalities, practice workflows, and perform common tasks under guidance and supervision.
- Solicit feedback from the participants during and after the training sessions to assess learning effectiveness and identify areas for improvement. Conduct knowledge assessments, quizzes, or surveys to evaluate learning outcomes and measure training impact.
- Offer ongoing support and resources to facilitate continuous learning and skill development beyond initial training. Provide access to user manuals, online help resources, knowledge bases, and user forums to address questions, troubleshoot issues, and reinforce learning.
- Schedule refresher training sessions periodically to reinforce learning, update users on system changes, and introduce new features or functionalities. Keep users informed about system updates, enhancements, and best practices through regular communication channels.
- Provide comprehensive documentation, reference materials, and job aids to supplement the training sessions. Develop user manuals, quick reference guides, FAQs, and troubleshooting tips to assist users in navigating the medical information system independently.
- Establish a feedback loop to gather input from users about their training experiences, challenges, and suggestions for improvement. Use feedback to iterate on training content, methods, and delivery approaches to enhance effectiveness and user satisfaction.

e. The creation of technical documentation for development and application use—based on the technical specifications document and the testing report, a manual detailing both technical aspects and workflow addressed to the end-user is created. Creating technical documentation for development and application use is essential for ensuring the clarity, consistency, and usability of the medical information system. The key to implementing a scalable information system is the documentation process and these are the main things to take into consideration:

- Identify the audience for the documentation, including developers, system administrators, end-users, and other stakeholders. Understand their knowledge level, roles, and information needs to tailor the documentation accordingly.
- Define a clear and logical structure for the documentation, including sections, chapters, and headings. Organize the content hierarchically to facilitate the navigation and retrieval of information.
- Maintain consistency in formatting, styling, and layout throughout the documentation. Use standardized templates, fonts, colors, and styles to enhance readability and visual appeal.

- Start with an overview of the documentation and the scope of the medical information system. Describe the purpose, objectives, and intended audience of the documentation to set expectations.
- Document the development process, methodologies, and best practices followed during system development. Include information on coding standards, version control, testing procedures, and deployment strategies.
- Provide detailed technical specifications for the medical information system, including architecture, components, modules, interfaces, and dependencies. Describe the system requirements, hardware specifications, and software dependencies.
- If applicable, document APIs used by the developers to interact with the medical information system. Provide clear and comprehensive API documentation, including endpoints, methods, parameters, and authentication mechanisms.
- Document configuration settings, installation procedures, and deployment instructions for system administrators. Provide step-by-step guides, screenshots, and troubleshooting tips to assist with system setup and deployment.
- Develop user guides and manuals for end-users, covering system functionalities, features, and usage instructions. Provide task-based guides, FAQs, and troubleshooting tips to help users navigate the system effectively.
- Document common error messages, warning signs, and troubleshooting procedures for developers, administrators, and end-users. Include error codes, descriptions, and recommended solutions for resolving issues.
- Document security guidelines, best practices, and recommendations for securing the medical information system. Provide information on access controls, data encryption, vulnerability management, and compliance with security standards.
- Maintain version control of the documentation and update it regularly to reflect changes in the medical information system. Document release notes, change logs, and version history to track updates and revisions.
- Include a glossary of terms and definitions used in the documentation to clarify technical terminology and acronyms. Define the key concepts, terms, and abbreviations to improve understanding and reduce ambiguity.
- Provide a mechanism for users to provide feedback on the documentation, such as comments, surveys, or feedback forms. Use the feedback to identify areas for improvement and update the documentation accordingly.
- Create manuals and user guides to illustrate the steps of the targeted workflow and how the transition of activities from outside the information system is performed, how it is used as a working tool, and the information obtained as a result and how it is subsequently used in the workflow.

- Maintenance (monitoring and optimization) is the final stage in the development of an information system following its implementation. In this stage, various errors encountered during the system's use are resolved, ways to optimize certain system functionalities are identified, and workflows are monitored through reports. Maintenance, monitoring, and optimization are critical aspects of ensuring the continued performance, stability, and efficiency of the medical information system. Define clear procedures and workflows for ongoing maintenance tasks, including monitoring, troubleshooting, patching, and optimization activities. Document maintenance schedules, responsibilities, and escalation procedures for handling issues as follows:
  - Implement monitoring tools and systems to continuously monitor the performance of the medical information system. Monitor KPIs such as response times, resource utilization, and system availability to detect issues and identify areas for optimization.

- Adopt a proactive approach to monitoring by setting up alerts, thresholds, and notifications for abnormal system behavior or performance degradation. Implement automated monitoring checks to detect and address issues before they impact users.
- Perform regular maintenance tasks such as database cleanup, log rotation, and system updates to keep the medical information system running smoothly. Schedule routine maintenance activities during off-peak hours to minimize disruption to users.
- Establish a patch management process to apply security patches, updates, and bug fixes to the system in a timely manner. Monitor vendor releases, security advisories, and software vulnerabilities to prioritize and schedule patching activities accordingly.
- Implement robust backup and disaster recovery mechanisms to protect against data loss and system downtime. Regularly backup critical data, databases, and configurations, and test backup and restore procedures to ensure data integrity and availability.
- Identify opportunities for optimizing system performance, efficiency, and resource utilization. Implement optimization strategies such as database indexing, query optimization, caching mechanisms, and resource allocation adjustments to improve system responsiveness and scalability.
- Ensure ongoing security maintenance by applying security best practices, updates, and patches to protect against security threats and vulnerabilities. Conduct regular security assessments, audits, and penetration tests to identify and mitigate security risks.
- Document maintenance procedures, troubleshooting steps, and optimization techniques in a centralized knowledge base or repository. Share best practices, lessons learned, and troubleshooting tips with the IT team and stakeholders to foster knowledge sharing and collaboration.
- Continuously monitor and tune system performance based on feedback, user experience, and performance metrics. Identify bottlenecks, optimize resource allocation, and fine-tune system configurations to enhance performance and user satisfaction.
- Solicit feedback from users and stakeholders regarding system performance, reliability, and usability. Establish channels for users to report issues, request assistance, and provide feedback on system maintenance and optimization efforts.
- Embrace a culture of continuous improvement by regularly reviewing and refining maintenance processes, monitoring strategies, and optimization techniques. Identify areas for enhancement, implement improvements, and iterate on maintenance practices to adapt to evolving system requirements and user needs.

- Evaluation: This stage involves assessing the software's performance, user satisfaction, and overall success. It includes gathering feedback from users and identifying improvement opportunities [29,30]. Evaluation is crucial for assessing the effectiveness, efficiency, and impact of the medical information system. For the results of the evaluation process to benefit the monitoring phase, we established some points to take into consideration in order not to confuse end-users and make them reluctant to this step:
  - Clearly define the objectives and goals of the evaluation, including what aspects of the medical information system will be evaluated and what outcomes are expected from the evaluation process.
  - Identify relevant evaluation criteria and performance metrics based on the goals and objectives of the evaluation. Consider factors such as system functionality, usability, performance, reliability, security, and user satisfaction.
  - Employ a mixed-methods approach to evaluation, combining qualitative and quantitative research methods. Use a variety of data collection techniques, such

as surveys, interviews, observations, and system logs, to gather comprehensive insights.

- Involve stakeholders, including healthcare professionals, administrators, IT staff, and end-users, in the evaluation process. Seek input from diverse perspectives to ensure a comprehensive understanding of the system's strengths, weaknesses, and impact.
- Develop a detailed plan for conducting evaluation activities, including data collection methods, timelines, and responsibilities. Allocate resources, establish evaluation protocols, and define roles and responsibilities for evaluation team members.
- Collect relevant data and information to assess the performance and effectiveness of the medical information system. Use a combination of quantitative metrics (e.g., system usage statistics and error rates) and qualitative insights (e.g., user feedback and observations) to provide a holistic view of system performance.
- Analyze the collected data using appropriate analytical techniques and tools. Identify trends, patterns, and correlations in the data to draw meaningful conclusions about the system's performance, usability, and impact on patient care.
- Interpret the evaluation findings in the context of the system's goals, objectives, and user needs. Analyze strengths, weaknesses, opportunities, and threats (SWOT analysis) to identify areas for improvement and strategic recommendations for enhancement.
- Based on the evaluation findings, provide actionable recommendations for optimizing system performance, addressing identified issues, and enhancing user satisfaction. Prioritize recommendations based on their potential impact and feasibility of implementation.
- Document evaluation results, findings, and recommendations in a comprehensive evaluation report. Present findings in a clear, concise, and visually appealing format, using charts, graphs, and tables to illustrate key insights.

Engaging various stakeholders in the process of developing key performance indicators (KPIs) is crucial for the successful assessment and adoption of digital health interventions. This collaborative approach ensures that the KPIs are comprehensive, relevant, and aligned with the specific needs and goals of all the parties involved. Stakeholders in healthcare, including clinicians, administrators, patients, and policymakers, bring diverse perspectives and expertise. By involving them in the development of KPIs, we can ensure that the indicators are not only scientifically valid but also practically applicable and meaningful across different contexts of healthcare delivery. This engagement helps in the following:

- Identifying relevant metrics as different stakeholders can identify what metrics are most relevant to their roles and responsibilities, ensuring a balanced and holistic set of KPIs.
- Ensuring buy-in and acceptance based on the fact that when stakeholders are part of the KPI development process, they are more likely to support and adhere to the assessment framework.
- Enhancing usability and impact by taking into consideration that stakeholder input can enhance the usability of digital health interventions by aligning KPIs with user needs and expectations, thus maximizing the intervention's impact.

The process of identifying and defining KPIs should be systematic and inclusive. The following methods can be employed to ensure effective stakeholder engagement:

- Workshops and focus groups with interactive sessions with stakeholders to brainstorm and discuss potential KPIs. This method fosters a collaborative environment where ideas can be shared and refined collectively.
- Surveys and questionnaires to gather quantitative and qualitative data on what stakeholders consider important metrics for evaluating digital health interventions.

- Interviews with key stakeholders to gain in-depth insights into their specific needs, challenges, and expectations.

In conclusion, the engagement of various stakeholders in the development of KPIs for digital health interventions is a fundamental step towards ensuring their effectiveness and acceptance. By employing structured methods for identifying and defining KPIs, we can create robust, relevant, and impactful metrics that drive the successful assessment and adoption of digital health solutions [31]. KPIs for assessing and adopting digital health interventions can vary depending on the specific intervention and its goals. However, some common KPIs are presented in Table 2.

Table 2. KPIs for assessing and adopting digital health interventions.

KPI	Details
Adoption Rate	The percentage of target users or stakeholders who have adopted the digital health intervention.
Engagement Metrics	Metrics such as user activity, the frequency of use, and the duration of use can indicate the level of engagement with the intervention.
Health Outcomes	The measures of health outcomes such as improvements in patient health, reduction in symptoms, or the achievement of health goals.
User Satisfaction	Feedback from users about their satisfaction with the intervention, including the ease of use, usefulness, and overall experience.
Cost-effectiveness	The evaluation of the cost-effectiveness of the intervention in terms of the resources required compared to the benefits achieved.
Interoperability	The ability of the intervention to integrate and exchange data with other systems and platforms.
Data Security and Privacy	The assessment of the intervention's compliance with data security and privacy regulations, as well as its effectiveness in safeguarding sensitive information.
Workflow Efficiency	The measurement of the impact of the intervention on workflow efficiency, such as reduction in time spent on administrative tasks or improvement in communication between healthcare providers.
Technology Performance	The evaluation of the reliability, availability, and performance of the technology infrastructure supporting the intervention.
Long-term Impact	The assessment of the long-term impact of the intervention on patient outcomes, healthcare delivery, and overall healthcare system performance.

To ensure project success and process optimization, the team must possess extensive expertise. Any delays and miscommunication, along with a shortage of expertise, vague objectives, and the neglect of best practices, could potentially harm the entire institution and its personnel. During the implementation of an information system, end-users encounter challenges, including reluctance toward new systems, which necessitates attention from both the team and management. Factors to take into consideration include the following:

- The fear of change—Humans are creatures of habit, and change can be unsettling. Introducing a new system means disrupting familiar workflows and routines, which can evoke fear and resistance among users who prefer the status quo.
- The lack of familiarity with the new system entails learning to use a new system, which requires time and effort. Users may feel overwhelmed or intimidated by unfamiliar

interfaces, features, and functionalities, especially if they lack adequate training or support.

- The perceived loss of control by using a new system as imposing constraints or limitations on their autonomy and freedom. They may fear losing control over their work processes or being micromanaged by the system.
- Uncertainty about benefits means users may be skeptical about the benefits and advantages of the new system, particularly if they perceive it as unnecessary or inferior to existing solutions. Without a clear understanding of how the system will improve their work or simplify tasks, they may resist adoption.
- Introducing new technology can evoke concerns about job security, especially if users fear that the system will automate or replace their roles. This fear of obsolescence can lead to resistance and reluctance to embrace change.
- Past negative encounters with poorly implemented or malfunctioning systems can create skepticism and reluctance to trust new technology. Users may harbor lingering doubts and reservations based on past negative experiences.
- Organizational culture, norms, and values can influence attitudes toward change and innovation. Resistance to new systems may be reinforced by cultural norms that prioritize stability, tradition, or hierarchy over innovation and experimentation.

Addressing reluctance to adopt a new system requires proactive communication, training, and support to help users overcome fears, build confidence, and recognize the benefits of the change. Providing clear explanations of the reasons for the change, offering comprehensive training programs, soliciting user feedback, and involving users in the decision-making process can help mitigate resistance and foster a smoother transition to the new system [32–37].

3.3. Case Study

Case Study: "Advancements in Healthcare: Development of a Comprehensive Medical Information System with Automated Classification for Ocular and Skin Pathologies—Structure, Functionalities, and Innovative Development Methods" published in *Applied System Innovation* Journal, MDPI, 2024 [12].

Background: The scope of the article was to present the development and implementation of a medical information system with an automated pathology detection module.

Implementation: The implementation platform was Salesforce, which featured an automated classification module for ocular and skin pathologies using Google Teachable Machine. The project had two primary objectives: first, to create a classification module that could integrate with the platform where the MIS was developed, and second, to develop the MIS itself. These two aspects were combined by embedding a medical image classification system directly into the information system. This integration optimized the workflow, benefiting both the medical team and patients by providing real-time, instant information about the presence of a pathology from an acquired image or video. The arhitecture of the MIS is presented in Figure 13.

Outcome:

- Enhanced Patient Care: The system allowed for seamless access to patient records across different departments, reducing errors and improving diagnosis and treatment accuracy.
- Operational Efficiency: Automated workflows and integrated systems reduced administrative burdens and improved resource allocation.
- Cost Savings: Improved data management and streamlined processes resulted in significant cost reductions over time.

This article is the first in a series that will be followed by several papers emphasizing the results of the pilot project evaluating the developed medical system in real-life scenarios in a medical institution. This pilot is crucial for gathering authentic user feedback and practical case studies. We anticipate publishing multiple papers in the near future that will elaborate on these findings.

Although detailed case studies and comprehensive user feedback will be included in future publications, preliminary feedback from the pilot users has been positive. The users have highlighted the system's intuitive interface, improved data management capabilities, and enhanced workflow efficiency as key benefits.

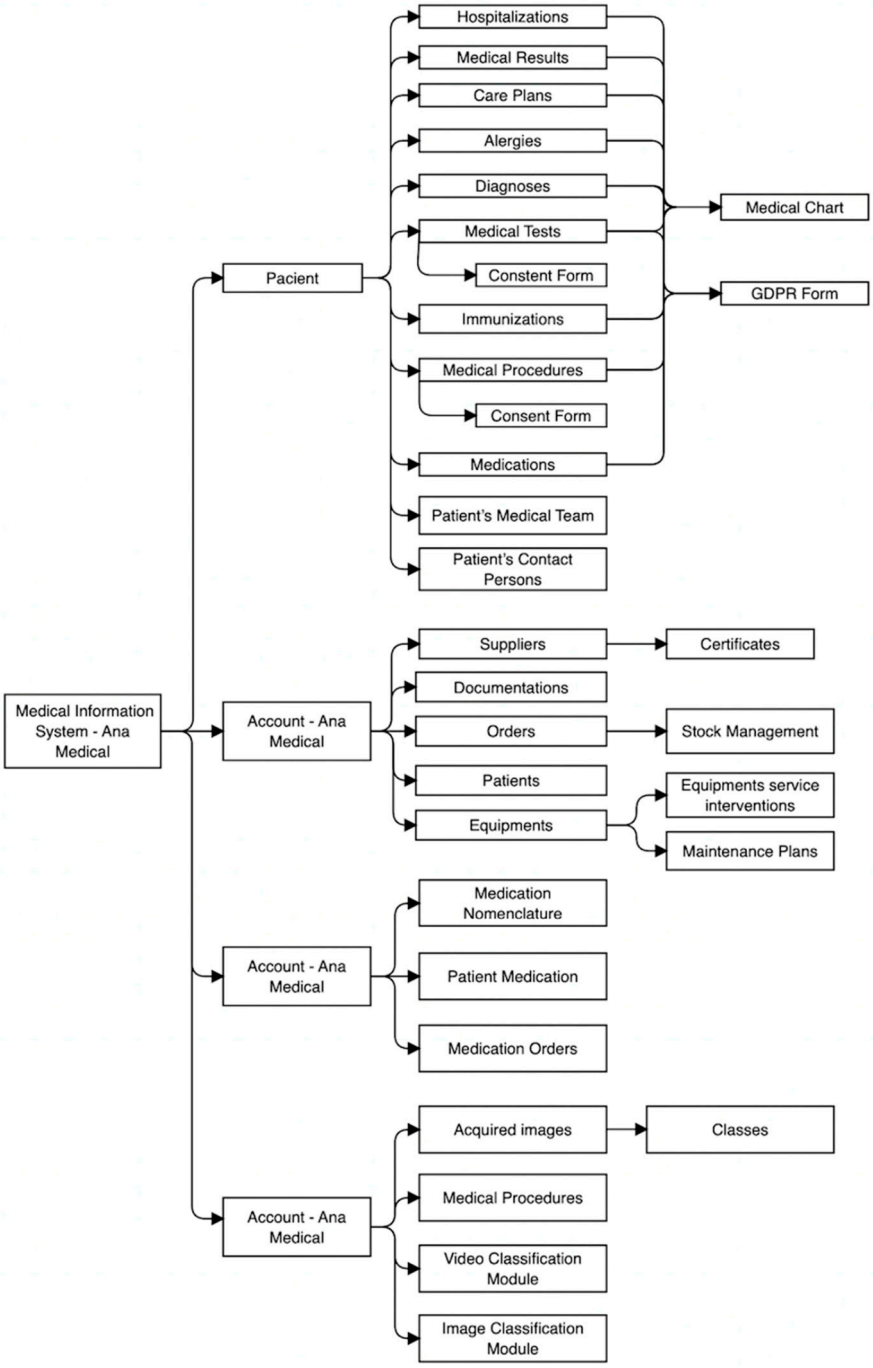

Figure 13. Example of a MIS implemented for a medical institution [12].

4. Deployment of Information Systems

An information system can be developed in-house or by engaging an outsourcing firm (a practice that outsources certain operations to a third party). After establishing the desired system structure, the necessary functionalities are developed, and following testing, these functionalities are deployed into the production environment.

In-house development refers to the practice of creating an IS internally within the organization. This approach involves assembling an in-house team of developers, designers, and other IT professionals who work exclusively for the healthcare institution. The organization retains complete control over the development process, allowing for a high degree of customization to meet specific requirements. In-house development provides the advantage of direct communication and collaboration between IT teams and healthcare professionals, fostering a deep understanding of the organization's unique needs. While it offers greater control and customization, in-house development can require significant time, resources, and expertise.

Outsourcing involves contracting external vendors or development teams to design, develop, and maintain an IS on behalf of the institution. Outsourcing offers several benefits, including cost savings, access to specialized expertise, and faster development timelines. The external vendors may bring industry-specific knowledge and experience to the project, ensuring the incorporation of best practices and compliance with healthcare regulations. However, outsourcing also introduces challenges such as potential communication barriers, dependency on external timelines, and the need for effective project management to ensure alignment with the organization's goals. Careful vendor selection and clear communication are crucial for successful outsourcing in the realm of HIS. The choice between in-house development and outsourcing for IS depends on various factors such as the organization's size, budget, timeline, and internal capabilities. In-house development offers greater control and adaptability but may require extensive resources. Outsourcing can be more cost-effective and time-efficient but demands careful vendor management. Some organizations adopt a hybrid approach, combining in-house and outsourced efforts to leverage the benefits of both models. Ultimately, the decision hinges on the specific needs, goals, and constraints of the healthcare institution seeking to implement or enhance its IS.

The process described in the previous chapter is made possible usually by a team of professionals with diverse skills and expertise to ensure the successful development, deployment, and maintenance of the system. In successfully implementing an information system, the team should be formed of the following:

- Project Manager: Is responsible for overall project planning, coordination, and execution. They ensure that the project stays on schedule, within budget, and meets the defined objectives.
- Product managers/owners represent the voice of the customer and are responsible for defining product requirements, prioritizing features, and ensuring that the development team delivers value to users.
- Business Analyst: Works closely with stakeholders to gather and analyze requirements, identify business needs, and translate them into functional specifications for the information system.
- Systems analyst analyzes existing systems, processes, and workflows to identify areas for improvement and develop system requirements. They bridge the gap between business needs and technical solutions.
- Software developers/Engineers: Responsible for designing, coding, testing, and implementing software applications or modules that make up the information system. They may specialize in frontend, backend, or full-stack development.
- Database administrators (DBAs) manage and maintain the databases that store the organization's data. They are responsible for ensuring data integrity, security, and performance.
- Network administrators manage the organization's network infrastructure, including servers, routers, switches, and other networking components. They ensure network reliability, security, and performance.
- UX/UI Designers: User experience (UX) and user interface (UI) designers focus on designing intuitive and visually appealing interfaces that enhance the usability and user satisfaction of the software.

- Quality assurance/testers are responsible for testing the system to identify and fix defects, ensure functionality meets requirements, and validate system performance and reliability.
- Scrum master/agile coach facilitates the agile development process, removes impediments, and ensures that the team follows agile principles and practices.
- Security Specialists ensure that the information system is secure against unauthorized access, data breaches, and other cybersecurity threats. They implement security measures such as encryption, access controls, and firewalls.
- Architects are responsible for designing the overall structure and architecture of the software, making high-level design decisions, and ensuring that the system meets technical requirements and standards.
- DevOps (development and operations) engineers focus on automating and streamlining the software delivery process, managing infrastructure, and ensuring the reliability and scalability of the software. DevOps is a set of practices that combines software development (Dev) with IT operations (Ops) to shorten the software development lifecycle and deliver high-quality software more rapidly. It emphasizes collaboration, automation, and integration between software developers and IT operations professionals.
- Training and support staff provide training to users on how to use the information system effectively and offer ongoing support to address any issues or questions that arise post-implementation.
- Release managers are responsible for planning and coordinating software releases, managing version control, and ensuring that releases are delivered on time and meet quality standards.

Depending on the size and complexity of the project, some roles may be combined or expanded, and additional roles may be required to address specific needs or challenges. Collaboration and communication among team members are essential for the success of the information system implementation.

Deployment means copying/transferring metadata from one environment to another (from a test environment—Sandbox to a production environment) [28]. Metadata represent the developments in the application (objects, classes, and automations) in file format for easy management and deployment. Data refer to the information contained in specific records, while metadata encompass schemas, processes, and general configurations of the information system, defining how the system operates. For example, metadata describe what happens when a button is pressed on a specific object or how certain information is displayed at the interface level. Through the metadata of a component or a page, it specifies security or a user's access to certain fields through the metadata of a profile or a permission.

Metadata are represented by XML files (Extensible Markup Language—a file format for storing, transmitting, and reconstructing arbitrary data) and can be moved by invoking deploy and retrieve to move data models containing newly developed functionalities. These metadata can be moved from one environment to another or locally on a computer. Once the XML files are stored locally, changes can be made to the source code, configurations can be modified, copied, and set, and at the component level and other similar modifications. Subsequently, the changes can be moved to the test or production environment.

4.1. Continuous Integration and Continuous Deployment (CI/CD)

The methodology for carrying out deployment follows the concept of CI/CD (continuous integration/continuous delivery). Continuous integration (or "CI") involves the automated movement of metadata through pipelines across environments for the merging and testing of packages before bringing them into production. This way, the testing and validation of new functionalities are optimized to reduce the time required for review [30]. Continuous deployment refers to the instant implementation of changes in an environment as soon as the changes have been merged into the version control. Continuous delivery is the ultimate goal and involves a workflow to bring new functionalities to end-users as

quickly as possible. This approach reduces manual work, thus decreasing the likelihood of errors. File deployment is achieved using an application such as Visual Studio Code—a source code editor that runs locally on a computer. It supports JavaScript, TypeScript, and Node.js, along with extensions for other programming languages (C++, C#, Java, Python, PHP, Go, and .NET).

Automate the process of building, testing, and deploying software artifacts to streamline development workflows. Use build automation tools such as Jenkins, Travis CI, or GitLab CI to automate the compilation, packaging, and distribution of software components. Here are some key technical points for the IT team to take into consideration:

- Use a version control system (e.g., Git) to manage source code, track changes, and facilitate collaboration among development teams. Adopt branching strategies such as GitFlow to organize feature development, bug fixes, and release management in a structured manner.
- Implement CI practices to integrate code changes into a shared repository frequently, typically several times a day. Configure CI pipelines to trigger automated builds, tests, and code quality checks whenever changes are pushed to the version control system. Use CI tools to run unit tests, integration tests, and other automated checks to validate code changes and identify issues early in the development process.
- Extend CI practices to include CD, enabling the automated deployment of code changes to production or staging environments. Implement deployment pipelines to automate the deployment process, including provisioning infrastructure, configuring environments, and deploying application updates. Use deployment orchestration tools such as Kubernetes, Docker Swarm, or AWS CodeDeploy to automate deployment tasks and manage the application lifecycle.
- Integrate monitoring and feedback mechanisms into the CI/CD pipelines to track system performance, health, and reliability. Monitor key metrics such as build success rate, test coverage, deployment frequency, and mean time to recovery to assess pipeline effectiveness and identify areas for improvement.
- Implement security best practices throughout the CI/CD pipeline to ensure code integrity, data confidentiality, and compliance with regulatory requirements. Incorporate security scanning tools, vulnerability assessments, and static code analysis into the CI/CD workflows to identify and remediate security vulnerabilities early in the development lifecycle.
- Enable incremental updates and rollback capabilities in the CI/CD pipelines to minimize the impact of failed deployments and ensure system reliability. Implement blue/green deployments, canary releases, or feature toggles to gradually roll out changes and mitigate risks associated with new releases.
- Document the CI/CD processes, procedures, and best practices to facilitate knowledge sharing and the onboarding of new team members. Provide training and resources to developers, testers, and operations teams to ensure they understand and adhere to the CI/CD principles and workflows.
- Foster a culture of continuous improvement by regularly reviewing and optimizing the CI/CD pipelines, tools, and practices. Collect feedback from development teams, stakeholders, and end-users to identify opportunities for enhancing automation, efficiency, and reliability in the CI/CD process.

4.2. Benefits of Continuous Deployment

First of all, instant implementation, often referred to as rapid deployment or quick deployment, involves the swift and immediate execution of a system, software, or process. This approach offers several advantages. Firstly, it significantly reduces the time required to launch a system or deploy a solution, allowing organizations to promptly respond to emerging needs or changing business conditions. Secondly, instant implementation facilitates a faster return on investment, enabling organizations to realize the benefits of their investment in a much shorter timeframe. Thirdly, it enhances organizational agility,

empowering businesses to adapt swiftly to market demands, technological changes, or competitive pressures—a crucial aspect in dynamic and fast-paced environments. Lastly, rapid deployment enables the quick testing of new features or functionalities, fostering an iterative approach that allows for prompt adjustments and improvements based on real-world usage and feedback.

Second, reducing manual work involves the automation of processes to diminish reliance on manual labor, presenting various advantages. Firstly, it enhances overall efficiency by eliminating the need for manual, repetitive tasks, allowing employees to redirect their focus towards more value-added activities. Secondly, automated processes ensure a higher level of consistency and accuracy compared to manual execution, particularly in tasks demanding precision and attention to detail. Thirdly, time savings are realized as employees are freed from routine, manual tasks, enabling them to invest their time in more strategic and creative aspects of their roles, ultimately contributing to heightened productivity. Moreover, this approach often leads to reduced operational costs by streamlining processes and minimizing required labor, thereby positively impacting the organization's bottom line. Simultaneously, minimizing errors is imperative for maintaining data integrity, ensuring quality outcomes, and improving overall organizational performance. The benefits encompass improved data quality, enhanced decision making through access to accurate information, increased customer satisfaction resulting from reliable interactions, and compliance with industry regulations and standards, thereby mitigating risks associated with incorrect data or flawed processes.

4.3. Version Control and Code Management

Version control and code management are integral aspects of software development, ensuring the organized and collaborative evolution of codebases. In the dynamic realm of software engineering, where multiple developers contribute to a project, version control becomes paramount. It enables teams to track changes, manage the different iterations of their code, and collaborate seamlessly. This process ensures that developers can work concurrently on various features without disrupting the overall stability of the project. Code management goes hand in hand with version control, extending its scope to encompass the broader strategies and practices employed in handling the complete lifecycle of software development. From initial coding to testing, deployment, and maintenance, effective code management involves creating structured workflows, implementing development best practices, and utilizing tools that enhance collaboration and productivity.

Together, version control and code management serve as the backbone for successful and streamlined software development, providing the necessary framework for developers to work cohesively, track changes systematically, and deliver high-quality, scalable solutions. This introduction sets the stage for exploring the nuanced strategies, tools, and methodologies that drive efficient version control and code management in the ever-evolving landscape of software engineering.

4.4. Versioning Strategies

Version control strategies and branching strategies offer structure and guidance on how code changes are managed, integrated, and deployed within a development team or organization. The choice of the method depends on factors such as team size, project complexity, implementation frequency, and collaboration requirements. The management of repositories can be conducted in the following way:

- Version control systems like Git, Mercurial, and Subversion are tools that manage changes in source code over time. They enable multiple developers to collaborate on a project by tracking changes, providing version history, branching, merging, and more.
- Hosting platforms are online platforms that offer hosting for version-controlled repositories. Some popular options include GitHub, GitLab, Bitbucket, and Azure DevOps. These platforms provide features like repository management, collaboration tools, issue tracking, and continuous integration.

- Access control managing a repository often involves controlling who has access to the repository and what they can do with it. This includes defining user roles (read, write, and administer) and establishing permissions at different levels.
- Branching strategies refer to how branches are organized and used in a repository and are an important aspect of management. Different branching strategies, such as Gitflow or trunk-based development, dictate how code is developed, integrated, and deployed.
- Code review is a process for code review which is essential for maintaining code quality. This involves reviewing and approving code changes before they are merged into the main codebase.
- Documentation—the proper documentation of the repository, including project purpose, setup instructions, coding standards, and contribution guidelines, helps in the efficient management of the repository.
- Keeping track of issues and bugs is crucial for monitoring and addressing problems. Many hosting platforms offer integrated issue-tracking systems.
- How and when code changes are released to users is another aspect of repository management. Proper release management ensures that the stable and tested versions of the software are made available to users.
- Implementing backups and having disaster recovery plans are important for protecting the repository and its data.
- Git hooks are scripts that automatically run in response to specific events, such as push or commit. They can be used to enforce coding standards, run tests, and perform other actions [33,34].

4.5. GitHub and Code Review

GitHub plays a pivotal role in modern version control, providing a collaborative platform that facilitates efficient code management and seamless collaboration among software developers. As a web-based hosting service, GitHub integrates the power of Git, a distributed version control system, to track changes, manage revisions, and coordinate the work of multiple contributors in a software project. The platform enables developers to host repositories, manage branches, and initiate pull requests to propose changes and enhancements. GitHub's user-friendly interface simplifies the process of code review, ensuring that team members can collaboratively assess modifications before incorporating them into the main codebase. Additionally, GitHub offers features like issue tracking, project management, and wikis, enhancing the overall organization and communication within development teams. With its emphasis on transparency, accessibility, and collaboration, GitHub has become a cornerstone in modern software development workflows. Whether for open-source projects or private repositories, GitHub's robust set of tools and functionalities empowers developers to streamline version control processes, fostering a more efficient and collaborative development environment. Key aspects to take into consideration include the following:

- Organize the repository logically, with separate directories for different components (e.g., frontend, backend, and documentation).
- Adopt a branching strategy such as Gitflow, where features are developed in feature branches, and stable releases are kept in the master branch.
- Create a pull request template that includes sections for describing the changes, listing related issues, and specifying any necessary testing steps.
- Encourage team members to review each other's code before merging PRs into the main branch.
- Set up CI pipelines using tools like GitHub Actions or Travis CI to automatically run tests and checks on every PR.
- Use GitHub Issues or a similar system to track bugs, feature requests, and other tasks. Link PRs to relevant issues to provide context.

- Enable GitHub's security features to receive alerts about vulnerabilities in dependencies and take appropriate action to address them.

Code reviews play a pivotal role in software development by offering a multifaceted approach to enhancing code quality. They serve as an effective error detection and bug prevention mechanism, leveraging multiple sets of eyes to identify issues early in the development process, thereby minimizing the chances of defects making their way into the production environment. Furthermore, code reviews facilitate knowledge sharing and onboarding within development teams, acting as a valuable mechanism for familiarizing new members with coding styles, best practices, and project-specific conventions. Enforcing consistency in coding styles is another significant outcome of code reviews, ensuring readability and maintainability across the entire codebase. Additionally, these reviews provide a platform for improving code design and architecture, allowing team members to offer constructive feedback on alignment with project requirements and scalability needs. Participation in code reviews contributes to a culture of continuous learning, enabling developers to enhance their skills through feedback and collaboration. The discussions and comments during code reviews serve as living documentation, capturing decision-making processes and rationale for future reference. Finally, code reviews act as a quality assurance checkpoint, ensuring compliance with coding standards and best practices, ultimately delivering reliable and maintainable software [34,38–42].

When developing any information system, several considerations must be taken into account from the standpoint of code review. It is crucial to ensure that the code is clear, concise, and adheres to consistent naming conventions. This involves declaring clear and descriptive variable names, utilizing detailed comments, maintaining consistent formatting and indentation, breaking down complex logic into smaller functions, and employing easily understandable function and method names. Additionally, it is imperative to verify that the code follows best practices for security, encompassing proper input validation, the encryption of sensitive data, access control, authorization, and secure authentication mechanisms.

From a performance perspective, it is essential to review the code for any potential performance bottlenecks or inefficient algorithms, particularly given the potentially large volume of data managed by medical systems. The areas to avoid include inefficient algorithms for data processing, excessive database queries, inefficient string concatenation, suboptimal sorting algorithms, and memory-intensive data structures. Scalability is another crucial consideration to accommodate the system's ability to handle a large volume of data and numerous users effectively.

5. Conclusions

In conclusion, information systems, fundamental to modern organizations, encompass a diverse set of components working synergistically to achieve efficiency and effectiveness. Defined by their socio-technical nature, these systems include human resources, equipment, software, data, procedures, networks, security, analysis, control, and environment. Their successful integration is pivotal for supporting organizational goals and decision making. Medical information systems, with their specialized modules, play a crucial role in enhancing patient care and operational efficiency in healthcare settings.

Implementation stages, deployment strategies, and continuous integration techniques, such as CI/CD, contribute to the systematic and agile development of information systems. Emphasizing rapid and error-minimizing strategies, CI/CD and version control mechanisms, including GitHub, facilitate collaborative coding environments. Code reviews ensure the quality of the codebase through error detection, knowledge sharing, and maintaining consistency.

Furthermore, the evaluation criteria for information systems, including functionality, the ease of use, performance, security, and flexibility, guide organizations in assessing the system's effectiveness. The advantages of instant implementation, reduction in manual work, and minimizing errors are pivotal for achieving operational efficiency and maintain-

ing code quality. In summary, information systems are dynamic entities that evolve through various stages of development, deployment, and continuous integration. Their strategic implementation and effective management contribute to organizational success, offering benefits in decision making, efficiency, and adaptability. The integration of specialized medical information systems and robust coding practices exemplifies the commitment to excellence in the evolving landscape of technology and healthcare.

Author Contributions: Conceptualization, methodology, and formal analysis, A.-M.Ș. and N.-R.R.; investigation, resources, data curation, writing—original draft preparation, writing—review and editing, visualization, and project administration, A.-M.Ș.; supervision, E.O. and M.C.; funding acquisition, M.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by POLITEHNICA Bucharest National University of Science and Technology through the PubArt doctoral program.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: This comprehensive guide has been thoughtfully developed, drawing on the implementation experiences of several information systems by Ana-Maria Ștefan and Nicu-Răzvan Rusu, along with their distinct personalized workflows and knowledge.

Conflicts of Interest: Author Nicu-Răzvan Rusu was an independent researcher. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

AI	Artificial intelligence
API	Application Programming Interface
AWS	Amazon Web Services
CD	Continuous deployment
CI	Continuous integration
CRM	Customer Relationship Management
DFD	Data flow diagram
DICOM	DICOM
EHR	Electronic health records
ERD	Entity Relationship diagram
ERP	Enterprise Resource Planning
FHIR	Fast Healthcare Interoperability Resources
GDPR	General Data Protection Regulation
HIE	Health Information Exchanges
HIPAA	Health Insurance Portability and Accountability
HL7	Health Level Seven International
IoT	Internet of Things
IS	Information system
KPI	Key performance indicators
LAN	Local Area Network
LOINC	Logical Observation Identifiers Names and Codes
MIS	Medical information system
ML	Machine Learning
NLP	Natural Language Processing
PACS	Picture Archiving and Communication System
SDLC	System Development Life Cycle
UAT	User acceptance testing
UML	Unified Modeling Language
VCS	Version control systems
WAN	Wide-Area Network
XML	Extensible Markup Language

Glossary

Term	Definition
Return on Investment (ROI)	A measure used to evaluate the efficiency or profitability of an investment, such as in Customer Relationship Management (CRM) software or Enterprise Resource Planning (ERP) systems.
Customer Relationship Management (CRM)	A software that helps organizations manage interactions with current and potential customers.
Enterprise Resource Planning (ERP)	Systems that integrate various business processes and functions into a single comprehensive system.
System Development Life Cycle (SDLC)	A process used in software development that outlines the stages from initial planning to deployment and maintenance.
Unified Modeling Language (UML)	A standardized modeling language used to visualize the design of a software system.
Deployment	The process of making a software application available for use in a live environment.
Backend	The server-side part of a software application that manages data and business logic.
Frontend	The client-side part of a software application that interacts with the user.
Database	A structured set of data held in a computer, typically managed by a database management system (DBMS).
RESTful APIs	Application Programming Interfaces (APIs) that adhere to the principles of Representational State Transfer (REST) for communication between systems.
SOAP services	Simple Object Access Protocol (SOAP) services that use XML-based messaging for communication between applications over the internet.
Asynchronous communication patterns	Methods such as message queues and publish/subscribe systems used to integrate components that handle long-running tasks or high volumes of requests, improving system responsiveness by decoupling components.
HTTPS	HyperText Transfer Protocol Secure, a protocol for secure communication over a computer network.
OAuth	An open standard for access delegation commonly used to grant websites or applications limited access to user information without exposing passwords.
JWT	JSON Web Token, a compact, URL-safe means of representing claims to be transferred between two parties.
Service level agreements (SLAs)	Contracts between service providers and customers that define the level of service expected from the service provider.
DICOM	Digital Imaging and Communications in Medicine, a standard for handling, storing, and transmitting information in medical imaging.
LOINC	Logical Observation Identifiers Names and Codes, a universal standard for identifying medical laboratory observations.
LIS	Laboratory information system, a software that manages data and workflows in a laboratory setting.
Cloud	A collection of remote servers and software networks that provide various computing services over the internet.
Hadoop	An open-source framework that allows for the distributed processing of large data sets across clusters of computers.
Spark	An open-source distributed computing system used for big data processing and analytics.
Elasticsearch	A search engine based on the Lucene library, used for full-text search, analytics, and more.
Production	The live environment where software applications are run and used by end-users.
Accuracy	A measure of how often a model correctly predicts the actual value.
Precision	The proportion of positive identifications that are actually correct.
Recall	The proportion of actual positives that are correctly identified by a model.

F1-score	A measure that combines precision and recall, providing a single metric for model performance.
ROC curves	Receiver Operating Characteristic curves, graphical plots that illustrate the diagnostic ability of a binary classifier system.
Confusion matrices	A table used to describe the performance of a classification model by comparing actual versus predicted values.
Dashboards	The visual displays of key metrics and data points, often used for monitoring and analysis.
Batches	The groups of transactions or data processed together at one time.
Go-Live	The point at which a new system or software application is put into production and becomes operational.
SWOT	A strategic planning technique used to identify strengths, weaknesses, opportunities, and threats related to a business or project.
Metadata	Data that provide information about other data, such as descriptions, context, or structure.
Packages	Bundles of software or modules that can be easily distributed and installed.
Repository	A central location where data, often source code, are stored and managed.
Pull Request (PR)	A method of submitting contributions to a software project, where changes are proposed and reviewed before being merged into the main codebase.

Appendix A

In-depth explanations of the various testing strategies to consider.
Unit Testing:

- Test coverage aims for high test coverage by designing test cases that exercise all critical paths and edge cases within the codebase. Cover both positive and negative scenarios to validate the behavior of components under various conditions.
- Isolation ensures that unit tests are isolated from external dependencies such as databases, network services, and external APIs. Use mocking frameworks or stubs to simulate external dependencies and control their behavior during testing.
- Test data management uses appropriate test data to validate the behavior of components. Generate realistic test data representative of real-world scenarios, including typical and boundary cases. Avoid using production data in unit tests to maintain data integrity and privacy.
- Develop testable design software components with testability in mind. Use principles such as dependency injection, the inversion of control, and the separation of concerns to decouple dependencies and facilitate unit testing. Design components should be modular, cohesive, and loosely coupled for easier testing.
- Write clear and concise assertions to verify the expected behavior of components. Ensure that assertions cover both the expected outcomes and side effects of component execution. Use descriptive assertion messages to provide meaningful feedback in case of test failures.
- Follow consistent and descriptive naming conventions for unit tests to make them self-explanatory and easy to understand. Use naming patterns such as <MethodName>_Should<ExpectedBehavior> or <ComponentUnderTest> _<Scenario>_Should<ExpectedOutcome> for clarity.
- Organize unit tests logically within test suites or test classes based on the functionality or module being tested. Group related tests together and use test fixtures or setup methods to prepare the test environment consistently across multiple tests. This step should contain the following:
    - A comprehensive test plan outlining the scope, objectives, resources, and schedule for testing activities. Define the test cases, test scenarios, and acceptance criteria to ensure a thorough test coverage.

- A dedicated test environment that mirrors the production environment as closely as possible. Configure hardware, software, and network settings to simulate real-world conditions for testing.
- Detailed test cases covering various aspects of the information system, including functionality, usability, performance, security, and compatibility. Ensure that the test cases are well-documented, executable, and traceable back to requirements.
- The execution of test cases according to the test plan, documenting test results, and tracking defects. Conduct functional testing, regression testing, integration testing, and other types of testing as required to validate system behavior.

○ Automate the execution of unit tests as part of the build and continuous integration (CI) process. Use CI/CD tools such as Jenkins, Travis CI, or GitHub Actions to automatically run the unit tests whenever code changes are made. Monitor the test results and integrate them with code review systems for feedback.

○ Continuously refactor and improve the unit tests as the codebase evolves. Refactor the test code to maintain readability, remove duplication, and improve maintainability. Ensure that the unit tests remain aligned with the latest changes in the codebase to provide accurate feedback. Optimize database schema and queries to improve data access performance and minimize data redundancy. Refactor database indexes, table structures, and SQL queries to ensure efficient data retrieval and storage. Consistently detect and address code issues like lengthy methods, oversized classes, and repeated code segments. Utilize refactoring methodologies like Method Extraction, Class Extraction, and Polymorphism Application to enhance code clarity and comprehensibility.

○ Document the unit tests effectively to provide context, rationale, and usage instructions for developers and maintainers. Use comments, annotations, or documentation tools to explain the purpose of the tests, assumptions made, and any known limitations or constraints.

○ Includes the unit tests in regression testing suites to detect regressions or unintended side effects introduced by the code changes. Re-run the unit tests frequently to ensure that existing functionality remains intact after modifications or enhancements, software updates, configuration changes, or database changes.

Integration Testing:

○ Establish a dedicated test environment that mirrors the production environment as closely as possible. This environment should include all necessary hardware, software, and network configurations to simulate real-world conditions.

○ Define a clear integration strategy that outlines the order in which components will be integrated and tested. Consider integration points, dependencies, and critical paths to determine the optimal testing approach.

○ Adopt an incremental integration approach, starting with testing individual components in isolation and gradually integrating and testing larger subsystems or modules. This allows for the early detection and resolution of integration issues and reduces the complexity of testing.

○ Identify integration points between system components, including APIs, databases, messaging systems, and external services. Clearly define the inputs, outputs, and expected behavior at each integration point to guide testing efforts.

○ Verify that the interfaces between components adhere to specifications and exchange data correctly. Test data formats, protocols, and communication channels to ensure seamless integration and interoperability.

○ Test boundary conditions and edge cases at the integration points to verify robustness and error handling. Include tests for maximum and minimum input values, boundary transitions, and exceptional scenarios to uncover potential vulnerabilities.

- Test the system's behavior under concurrent and parallel execution scenarios. Verify that multiple components can interact and execute concurrently without interference or data corruption.
- Verify data integrity across integrated components by comparing the input and output data at the integration points. Perform data reconciliation checks to ensure the consistency and accuracy of the data exchanged between components.
- Manage dependencies between components effectively during integration testing. Use stubs, mocks, or simulators to simulate external dependencies and isolate components for testing purposes.

System Testing:

- Have clearly defined the scope of system testing, including the functionalities, features, and user scenarios to be tested. Consider both functional and non-functional requirements, regulatory compliance, and user acceptance criteria.
- Develop a dedicated test environment that closely resembles the production environment, including hardware, software, network configurations, and test data. Ensure that the test environment is stable, isolated, and representative of real-world conditions.
- Establish comprehensive test cases that cover all aspects of the system's functionality, including positive and negative scenarios, boundary conditions, and edge cases. Test cases should be clear, unambiguous, and executable, with predefined inputs, expected outcomes, and validation criteria.
- Integrate end-to-end testing to validate the system's behavior across all user interactions, workflows, and integration points. Test the entire patient journey, from appointment booking and registration to diagnosis, treatment, and discharge, to ensure the seamless continuity of care.
- Verify that the system meets all functional requirements specified in the requirements documentation. Test individual features, modules, and workflows to validate their correctness, completeness, and compliance with user expectations.
- Perform non-functional testing to assess the system's performance, scalability, reliability, security, and usability. Test response times, throughput, system stability, data integrity, authentication mechanisms, and user interface elements to ensure optimal system behavior.
- Conduct security testing to identify and mitigate vulnerabilities in the system's architecture, design, and implementation.
- Validate interoperability with external systems, devices, and services to ensure seamless data exchange and integration.
- Document system test plans, test cases, and test results comprehensively. Provide clear and detailed reports on test coverage, defects found, and resolutions to stakeholders. Document any deviations from the expected behavior and proposed corrective actions.
- Continuously evaluate and refine the system testing process based on feedback, lessons learned, and evolving requirements. Incorporate the feedback from users, stakeholders, and testing teams to improve testing practices, automation, and overall system quality.

Performance and Stress Testing:

- Identify and prioritize performance scenarios representing typical user interactions, peak usage scenarios, and critical workflows within the medical information system. Consider scenarios such as patient record retrieval, appointment scheduling, and the real-time monitoring of vital signs.
- Set up a dedicated performance testing environment that closely resembles the production environment in terms of hardware, software, network configuration, and database size. Ensure that the test environment is isolated, stable, and scalable to accommodate load testing requirements. Generate realistic load profiles representative of expected usage patterns and peak traffic conditions.

- Conduct stress testing to evaluate the system's behavior under extreme load conditions beyond normal operating limits. Gradually increase the load, volume, and concurrency levels to identify performance bottlenecks, resource constraints, and system failure points.
- Monitor performance metrics such as response time, throughput, CPU utilization, memory usage, and network traffic during performance tests. Use performance monitoring tools and dashboards to collect, analyze, and visualize performance data in real-time.
- Optimize system performance based on performance test results and recommendations. Implement performance-tuning techniques such as caching, indexing, query optimization, and load balancing to improve system responsiveness and scalability.
- Perform long-running stress tests to assess the system's stability and reliability over extended periods of sustained stress. Run stress tests for hours or days to identify memory leaks, resource exhaustion, and performance degradation over time.
- Test failure scenarios such as server crashes, network outages, or database failures to evaluate the system's fault tolerance and resilience. Introduce failures deliberately during stress testing to observe how the system recovers and handles failures gracefully.
- Test the system's recovery mechanisms and resilience under stress conditions. Measure recovery time, data integrity, and system stability after the stress load is removed to ensure that the system can recover gracefully from stress-induced failures.

Security Testing:

- Perform threat modeling to identify potential security threats, vulnerabilities, and attack vectors specific to the medical information system. Analyze system architecture, data flow, access controls, and potential attack surfaces to prioritize security testing efforts.
- Review security requirements specified in the system's requirements documentation, regulatory standards, and industry best practices. Ensure that security requirements are clearly defined, measurable, and testable to guide security testing efforts.
- Test authentication mechanisms such as login processes, password policies, multi-factor authentication, and session management to ensure secure access control. Verify the proper enforcement of access rights, role-based permissions, and least privilege principles.
- Evaluate data security controls to ensure the confidentiality, integrity, and availability of sensitive healthcare data. Test encryption algorithms, data masking, data-at-rest protection, and secure transmission protocols to protect data in transit and at rest.
- Test APIs for security vulnerabilities such as improper authentication, authorization bypass, and data exposure risks. Verify secure API design, input validation, access controls, and the encryption of sensitive data transmitted over APIs.

Usability Testing:

- Establish clear usability goals and objectives based on user needs, expectations, and system requirements. Define the measurable usability metrics such as task completion time, error rates, and user satisfaction scores to evaluate system usability.
- Identify target user personas representing different user roles, demographics, and skill levels within the healthcare context. Tailor usability testing scenarios and tasks to the needs, preferences, and workflows of each user persona.
- Develop a usability test plan outlining the scope, objectives, methodologies, and test scenarios for usability testing. Define the usability test tasks, scenarios, and success criteria to assess system usability effectively.
- Recruit representative participants from the target user population to participate in usability testing sessions. Ensure diversity in participant demographics, roles, and experience levels to capture a broad range of user perspectives.

- Set up a dedicated usability testing environment that closely resembles the production environment in terms of hardware, software, and user interfaces. Provide necessary equipment, facilities, and tools to support usability testing activities.
- Design usability test scenarios and tasks that reflect common user workflows, tasks, and goals within the medical information system. Include tasks such as patient registration, appointment scheduling, record retrieval, and data entry to evaluate system usability.
- Encourage the participants to verbalize their thoughts, feelings, and actions while performing usability test tasks using the think-aloud protocol. Capture user feedback, observations, and insights in real-time to understand user behavior and interaction patterns.
- Analyze usability test results and user feedback to identify usability issues, pain points, and areas for improvement. Prioritize usability issues based on severity, impact on user experience, and the frequency of occurrence for remediation.
- Document usability testing procedures, findings, and recommendations in detailed usability test reports. Provide clear and actionable recommendations for addressing identified usability issues, enhancing system usability, and improving user satisfaction.

Compatibility Testing:

- Identify the target environments, platforms, devices, browsers, and operating systems that the medical information system needs to support. Define the compatibility requirements based on user preferences, organizational standards, and industry best practices.
- Test the medical information system across popular web browsers such as Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, and Opera. Verify that the system's features and functionalities work consistently across different browser versions and configurations.
- Test the medical information system on various operating systems such as Windows, macOS, Linux, iOS, and Android. Validate compatibility with different operating system versions, editions, and configurations to ensure consistent performance and functionality.
- Test the medical information system on different devices such as desktop computers, laptops, tablets, and smartphones. Verify compatibility with various screen sizes, resolutions, input methods, and device capabilities to provide a seamless user experience across devices.
- Validate compatibility with different hardware configurations, peripherals, and accessories commonly used in healthcare settings. Test integration with printers, scanners, barcode readers, and medical devices to ensure interoperability and functionality.
- Test the medical information system under different network conditions such as wired, wireless, and mobile networks. Verify performance, responsiveness, and data transmission reliability across varying network speeds, latency levels, and connectivity conditions.
- Verify support for different languages, locales, and regional settings to ensure internationalization and localization readiness. Test language localization, date and time formats, currency symbols, and cultural conventions to accommodate users from diverse regions and cultures.
- Test the medical information system for accessibility compliance and compatibility with assistive technologies such as screen readers, magnifiers, and voice recognition software.
- Validate compatibility with third-party systems, databases, APIs, and services commonly integrated with the medical information system. Test data exchange formats, communication protocols, and interoperability to ensure seamless integration with external systems.
- Test the backward and forward compatibility of the medical information system with different software versions, updates, and patches. Verify that the system remains

compatible with the older and newer versions of dependencies, libraries, and software components.

Acceptance Testing:

- Involve end-users, stakeholders, and domain experts in the acceptance testing process. Collaborate with healthcare professionals, administrators, and patients to validate system functionality, usability, and suitability for real-world use.
- Develop a comprehensive acceptance test plan outlining the scope, objectives, methodologies, and test scenarios for acceptance testing. Define the acceptance criteria, test cases, and success criteria to ensure a thorough evaluation of system readiness for deployment.
- Evaluate the user interface for usability, intuitiveness, and adherence to design standards during acceptance testing. Solicit user feedback on interface design, navigation, layout, and interaction patterns to identify areas for improvement and optimization.
- Validate user acceptance criteria against actual system behavior and performance. Ensure that the system meets user expectations, addresses user needs, and delivers value to stakeholders before proceeding with deployment.

Exploratory Testing

- Conduct exploratory testing sessions with an open and curious mindset, exploring different areas of the system without predefined test scripts or scenarios. Experiment with various inputs, interactions, and usage patterns to uncover unexpected behaviors and defects.
- Embrace change and adaptability during the exploratory testing, responding dynamically to new information, feedback, and observations. Adjust testing strategies, priorities, and focus areas based on emerging insights and evolving testing objectives.
- Prioritize testing efforts based on risk, focusing on critical functionalities, high-impact areas, and potential vulnerabilities within the medical information system. Identify the areas prone to defects, errors, and usability issues for deeper exploration and testing.
- Apply testing heuristics, mnemonics, and models to guide exploratory testing activities effectively. Use heuristics such as "POSH (Place, Order, Size, and Hidden)" to identify testing priorities and generate test ideas based on observable system characteristics [43].
- Generate test ideas spontaneously based on intuition, experience, and domain knowledge. Brainstorm creative test scenarios, edge cases, and negative test cases to challenge the system's behavior and uncover hidden defects.
- Take detailed notes, screenshots, and recordings during the exploratory testing sessions to document observations, findings, and insights. Capture the test scenarios, test data, steps performed, and observed behaviors for later analysis and reporting.

Alpha Testing:

- Clearly define the objectives, scope, and criteria for alpha testing. Determine the goals of the testing phase, including functionality validation, usability assessment, and defect identification.
- Identify a diverse group of alpha testers representing the target user population, including healthcare professionals, administrators, and end-users. Ensure that the alpha testers have the necessary domain expertise and are willing to actively participate in the testing process.
- Develop a comprehensive test plan outlining the test objectives, test scenarios, test cases, and testing procedures for alpha testing. Define the acceptance criteria, success metrics, and timelines for completing the alpha testing activities.
- Configure the test environment with the latest version of the medical information system and relevant test data. Ensure that the test environment mirrors the production environment as closely as possible to simulate real-world conditions.

- Provide training and orientation sessions for the alpha testers to familiarize them with the medical information system's functionalities, features, and usage guidelines. Educate the testers on how to perform tests, report issues, and provide feedback effectively.
- Execute test cases according to the predefined test plan, scenarios, and acceptance criteria. Encourage the alpha testers to explore different areas of the system, perform common tasks, and interact with various functionalities to uncover defects and usability issues.
- Document defects, issues, and observations encountered during alpha testing in a centralized defect-tracking system. Include detailed information such as steps to reproduce, expected behavior, actual results, and severity level for each reported issue.
- Evaluate alpha testing results against the predefined acceptance criteria and success metrics. Analyze test coverage, defect trends, and user feedback to assess the system's readiness for beta testing and eventual release to production.
- Document the alpha testing procedures, findings, and recommendations in detailed test reports. Provide clear and comprehensive reports on the test coverage, test results, defect metrics, and action items for stakeholders' review and decision making.

Beta Testing:

- Clearly define the objectives, scope, and criteria for beta testing. Determine the goals of the testing phase, including functionality validation, user feedback collection, and readiness assessment for production release.
- Recruit a diverse group of beta testers representing the target user population, including healthcare professionals, patients, and other stakeholders. Ensure that the beta testers have varying levels of expertise, backgrounds, and usage scenarios to provide comprehensive feedback.
- Identify and recruit the beta testers through user forums, community groups, and direct invitations. Provide clear instructions and guidelines for participating in beta testing, including how to access the system, report issues, and provide feedback.
- Plan the release of the medical information system to the beta testers, including the release schedule, deployment process, and version control. Coordinate with development teams, IT administrators, and stakeholders to ensure a smooth and timely release.
- Configure the beta test environment with the latest version of the medical information system and relevant test data. Ensure that the test environment is stable, secure, and accessible to the beta testers for testing purposes.
- Establish communication channels and support mechanisms for the beta testers to report issues, ask questions, and provide feedback. Provide clear instructions on how to contact support, submit bug reports, and participate in feedback discussions.
- Encourage the beta testers to explore different areas of the system, perform common tasks, and interact with various functionalities. Request the testers to provide feedback on usability, performance, reliability, and overall user experience during the testing period.
- Collect feedback from the beta testers through surveys, feedback forms, interviews, and discussion forums. Solicit qualitative feedback on usability, feature requests, enhancement suggestions, and bug reports to improve the system's quality and user satisfaction.
- Document and prioritize defects reported by the beta testers in a centralized defect-tracking system. Analyze reported issues, triage defects, and collaborate with development teams to address and resolve issues promptly.

References

1. Hassling, L.; Babic, A.; Lönn, U.; Casimir-Ahn, H. A Web-Based Patient Information System-Identification of Patients' Information Needs. *J. Med. Syst.* **2003**, *27*, 247–257. [CrossRef] [PubMed]
2. Tatarkanov, A.; Umyskov, A.A.; Tekeev, R.; Kuklin, V. Model Development of Universal Hardware and Software Module for Medical Information System. *Int. J. Emerg. Technol. Adv. Eng.* **2022**, *12*, 136–146. [CrossRef] [PubMed]
3. Epizitone, A.; Moyane, S.P.; Agbehadji, I.E. A Systematic Literature Review of Health Information Systems for Healthcare. *Healthcare* **2023**, *11*, 959. [CrossRef] [PubMed] [PubMed Central]
4. Khubone, T.; Tlou, B.; Mashamba-Thompson, T.P. Electronic Health Information Systems to Improve Disease Diagnosis and Management at Point-of-Care in Low and Middle Income Countries: A Narrative Review. *Diagnostics* **2020**, *10*, 327. [CrossRef] [PubMed] [PubMed Central]
5. Malliarou, M. Advantages of Information Systems in Health Services. *Choregia.* **2009**, *5*, 43.
6. Karimi, J. Strategic Planning for Information Systems: Requirements and Information Engineering Methods. *J. Manag. Inf. Syst.* **1988**, *4*, 5–24. [CrossRef]
7. Khalifa, M. Perceived Benefits of Implementing and Using Hospital Information Systems and Electronic Medical Records. *Stud. Health Technol. Inform.* **2017**, *238*, 165. [CrossRef] [PubMed]
8. Abrego Almazán, D.; Sánchez Tovar, Y.; Medina Quintero, J.M. Influence of information systems on organizational results. *Contad. Adm.* **2017**, *62*, 321–338. [CrossRef]
9. Ragowsky, A.; Ahituv, N.; Neumann, S. The benefits of using information systems. *Commun. ACM* **2000**, *43*, 13-es. [CrossRef]
10. Alsalman, D.; Alumran, A.; Alrayes, S.; Althumairi, A.; Almubarak, S.; Alrawiai, S.; Alakrawi, Z.; Hariri, B.; Alanzi, T. Implementation status of health information systems in hospitals in the eastern province of Saudi Arabia. *Inform. Med. Unlocked* **2021**, *22*, 100499. [CrossRef]
11. El-Khatib, H.; Ștefan, A.-M.; Popescu, D. Performance Improvement of Melanoma Detection Using a Multi-Network System Based on Decision Fusion. *Appl. Sci.* **2023**, *13*, 10536. [CrossRef]
12. Ștefan, A.-M.; Rusu, N.-R.; Ovreiu, E.; Ciuc, M. Advancements in Healthcare: Development of a Comprehensive Medical Information System with Automated Classification for Ocular and Skin Pathologies—Structure, Functionalities, and Innovative Development Methods. *Appl. Syst. Innov.* **2024**, *7*, 28. [CrossRef]
13. El-khatib, H.; Ștefan, A.-M.; Popescu, D. Melanoma Automated Detection System Integrated with an EHR Platform. *UPB Sci. Bull. Ser. C Electr. Eng. Comput. Sci.* **2024**, *86*, 2024.
14. Ștefan, A.-M.; Ovreiu, E.; Ciuc, M. Comparative analysis of web-based machine learning models. *Rom. J. Inf. Technol. Autom. Control.* **2024**, *34*.
15. Sittig, D.F.; Singh, H. A New Socio-technical Model for Studying Health Information Technology in Complex Adaptive Healthcare Systems. In *Cognitive Informatics for Biomedicine. 59–80 Health Informatics*; Patel, V.L., Kannampallil, T.G., Kaufman, D.R., Eds.; Springer: Cham, Switzerland, 2015. [CrossRef]
16. Or, C.; Dohan, M.; Tan, J. Understanding Critical Barriers to Implementing a Clinical Information System in a Nursing Home Through the Lens of a Socio-Technical Perspective. *J. Med. Syst.* **2014**, *38*, 99. [CrossRef] [PubMed]
17. Novak, L.L.; Holden, R.J.; Anders, S.H.; Hong, J.Y.; Karsh, B.T. Using a sociotechnical framework to understand adaptations in health IT implementation. *Int. J. Med. Inform.* **2013**, *82*, e331–e344. [CrossRef] [PubMed] [PubMed Central]
18. Noël, R.; Taramasco, C.; Márquez, G. Standards, Processes, and Tools Used to Evaluate the Quality of Health Information Systems: Systematic Literature Review. *J. Med. Internet Res.* **2022**, *24*, e26577. [CrossRef] [PubMed] [PubMed Central]
19. Siau, K.; Tan, X. Evaluation criteria for information systems development methodologies. *Am. Conf. Inf. Syst.* **2005**, *16*, 509.
20. Sharifi Kia, A.; Beheshti, M.; Shahmoradi, L. Health Information Systems Evaluation Criteria: Overview of Systematic Reviews. *Front. Health Inform.* **2022**, *11*, 120. [CrossRef]
21. Hanmer, L. Criteria for the evaluation of district health information systems. *Int. J. Med. Inform.* **1999**, *56*, 161–168. [CrossRef]
22. Stair, R.M.; Reynolds, G.W. *Principles of Information Systems: A Managerial Approach*; Course Technology Cengage Learning: Boston, MA, USA, 2010.
23. Pecoraro, F.; Luzi, D. Using Unified Modeling Language to Analyze Business Processes in the Delivery of Child Health Services. *Int. J. Environ. Res. Public Health* **2022**, *19*, 13456. [CrossRef] [PubMed]
24. Modha, J.; Gwinnett, A.; Bruce, M. A review of information systems development methodology (ISDM) selection techniques. *Omega* **1990**, *18*, 473–490. [CrossRef]
25. Anwar, N.; Kar, S. Review Paper on Various Software Testing Techniques & Strategies. *Glob. J. Comput. Sci. Technol.* **2019**, *19*, 43–49. [CrossRef]
26. Jugulum, R. Information System Testing. In *Competing with High Quality Data: Concepts, Tools, and Techniques for Building A Successful Approach to Data Quality*; Wiley: Hoboken, NJ, USA, 2014; pp. 121–138. [CrossRef]
27. Tanuska, P.; Vlkovic, O.; Spendla, L. The Usage of Performance Testing for Information Systems. *Int. J. Comput. Theory Eng.* **2012**, *4*, 144–147. [CrossRef]
28. Neto, P.S.; Resende, R.; Pádua, C. A Method for Information Systems Testing Automation. In *Advanced Information Systems Engineering*; Notes on Numerical Fluid Mechanics and Multidisciplinary Design; Springer: Berlin/Heidelberg, Germany, 2005; pp. 504–518. [CrossRef]

29. Kruger, R.; Brosens, J.; Hattingh, M. A Methodology to Compare the Usability of Information Systems. In *Responsible Design, Implementation and Use of Information and Communication Technology, Proceedings of the 19th IFIP WG 6.11 Conference on e-Business, e-Services, and e-Society, I3E 2020, Skukuza, South Africa, 6–8 April 2020*; Springer Nature: Berlin, Germany, 2020; Volume 12067, pp. 452–463. [CrossRef]
30. Rocha, A.; Correia, A.M.; Tan, F.B.; Stroetmann, K.A. *New Perspectives in Information Systems and Technologies*; Springer International Publishing: Cham, Switzerland, 2014; Volume 1.
31. Brenner, M.; Weir, A.; McCann, M.; Doyle, C.; Hughes, M.; Moen, A.; Ingvar, M.; Nauwelaerts, K.; Turk, E.; McCabe, C. Development of the key performance indicators for digital health interventions: A scoping review. *Digit. Health* **2023**, *9*, 20552076231152160. [CrossRef]
32. Denić, N.; Živić, N.; Siljković, B. Management of The Information Systems Implementation Project. *Ann. Oradea Univ. Fascicle Manag. Technol. Eng.* **2013**, *XXII*. [CrossRef]
33. Ska, Y.; Mohammad, S. A Study and Analysis of Continuous Delivery, Continuous Integration In Software Development Environment. *SSRN Electron. J.* **2019**, *6*, 96–107.
34. Available online: https://code.visualstudio.com/docs (accessed on 20 February 2022).
35. Or, C.; Chan, A.H.S. Inspection Methods for Usability Evaluation. In *User Experience Methods and Tools in Human-Computer Interaction*, 1st ed.; Stephanidis, C., Salvendy, G., Eds.; CRC Press: Boca Raton, FL, USA, 2024.
36. Kushniruk, A.; Monkman, H.; Borycki, E.; Kannry, J. User-Centered Design and Evaluation of Clinical Information Systems: A Usability Engineering Perspective. In *Cognitive Informatics for Biomedicine*; Springer: Berlin/Heidelberg, Germany, 2015. [CrossRef]
37. Karsh, B.-T.; Holden, R.; Or, C.K. Human Factors and ergonomics of health information technology implementation. In *Handbook of Human Factors and Ergonomics in Health Care and Patient Safety*, 2nd ed.; Carayon, P., Ed.; CRC Press: Boca Raton, FL, USA, 2012; pp. 249–264. Available online: https://hub.hku.hk/handle/10722/185234 (accessed on 24 July 2023).
38. Available online: https://www.atlassian.com/git/tutorials/what-is-version-control (accessed on 20 February 2022).
39. Available online: https://github.com (accessed on 20 February 2022).
40. Zolkifli, N.N.; Ngah, A.; Deraman, A. Version Control System: A Review. *Procedia Comput. Sci.* **2018**, *135*, 408–415. [CrossRef]
41. Available online: https://www.spiceworks.com/tech/devops/articles/what-is-version-control/ (accessed on 20 February 2022).
42. Available online: https://www.accuwebhosting.com/blog/best-version-control-systems/ (accessed on 21 February 2022).
43. Liu, W.; Tuck, J.; Ceze, L.; Strauss, K.; Renau, J.; Torrellas, J. POSH: A Profiler-Enhanced TLS Compiler That Leverages Program Structure. 2005. Available online: https://iacoma.cs.uiuc.edu/iacoma-papers/pacs05_1.pdf (accessed on 15 April 2024).