ASSIGNMENT 1

Soft Computing Lab

Member	Enrollment No
Ramesh Chandra Soren	2022CSB086
Moodu Roopa	2022CSB087
Jitendra Kumar Dodwadiya	2022CSB089
Deep Sutariya	2022CSB090

Part I: 1 week

>> We know that a software lifecycle consists of many different phases like feasibility study, requirement analysis, design, coding, testing and maintenance.

>> Explore online sample charts/tables/sheets used for formal documentation in each of these phases.

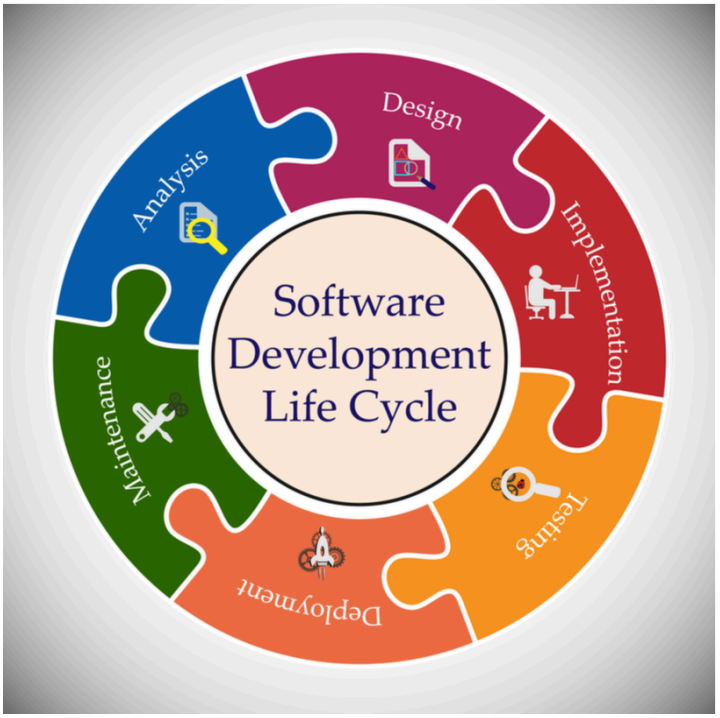
Part II: 1 week

>> Hence briefly describe your collected samples in the form of a case study (word limit for each sample:100).

>> Since you are guaranteed to come up with many pieces of text, figures, tables, notations etc. in discrete form, you are advised to submit a single doc/pdf file for this entire assignment.

>> In your descriptions don't forget to refer your sources (url/book/journal/conference etc.) of the samples.

Software Development Life Cycle



>> The software development lifecycle (SDLC) is a step-by-step process that helps development teams efficiently build the highest quality software at the lowest cost.

>> Teams follow the SDLC to help them plan, analyze, design, test, deploy, and maintain software.

>> The SDLC also helps teams ensure that the software meets stakeholder requirements and adheres to their organization's standards for quality, security, and Compliance.

Phase	Purpose	Key Activities	Output/Deliverables
Planning	Define project goals and assess feasibility.	- Conduct feasibility study. - Identify project scope and objectives. - Develop timelines.	- Feasibility Study Report. - Project Plan.

Requirement Analysis	Gather and analyze user and system requirements.	- Interact with stakeholders. - Create use cases. - Document functional and non-functional requirements.	- Software Requirement Specification (SRS). - Use Case Diagrams.
Design	Create the architecture and design of the system.	- Develop system and database design. - Design UI/UX interfaces. - Select technologies.	- Design Document. - System Architecture Diagrams.
Development (Coding)	Implement the design into working software.	- Write code for the system. - Perform unit testing. - Use version control tools.	- Source Code. - Code Documentation.
Testing	Verify that the software meets requirements and is defect-free.	- Execute test cases. - Perform functional and non-functional testing. - Log defects and retest fixes.	- Test Plan. - Test Cases and Reports.
Deployment	Deliver the product to users and set up the production environment.	- Install software in production. - Train users. - Monitor for early issues.	- Deployed Software. - Deployment Plan.
Maintenance	Ensure the software remains functional and updated post-deployment.	- Fix bugs. - Update software for new features. - Monitor performance.	- Maintenance Logs. - Updated Software Versions.

Formal documentation is essential in each phase of the Software Development Life Cycle (SDLC) to ensure clarity, consistency, and quality. Below is an overview of each phase, along with sample charts, tables, and templates commonly used for documentation:

1. Feasibility Study

- **Feasibility Analysis Report**: Assesses the practicality of the proposed project, including technical, economic, legal, operational, and scheduling considerations.
    - *Sample Template*: The California Department of Technology provides SDLC plans and tools, including templates for feasibility analysis.
      Project Resources

2. Requirement Analysis

- **Requirements Specification Document (RSD)**: Details functional and non-functional requirements, serving as a foundation for system design.
    - *Sample Template*: The Maryland Department of Information Technology offers a comprehensive SDLC Template Library, which includes templates for requirements specification.
      Do It Maryland

3. Design

- **System Design Document (SDD)**: Outlines the system architecture, components, interfaces, and data flow.
    - *Sample Template*: SDLCforms provides fillable templates for various SDLC phases, including system design.
      SDLCforms
- **Entity-Relationship Diagrams (ERDs)**: Visual representations of data entities and their relationships.
    - *Sample Tool*: Miro offers an SDLC template that can be adapted to create ERDs.
      https://miro.com/

4. Coding

- **Code Review Checklists**: Ensure adherence to coding standards and identify potential issues.
    - *Sample Template*: The University of Illinois provides an SDLC Life Cycle Document that includes guidelines for code reviews.
      Home

5. Testing

- **Test Plan Document**: Describes the scope, approach, resources, and schedule for testing activities.
    - *Sample Template*: Qulix Systems discusses the importance of test plans in their SDLC template overview.
      Qulix
- **Defect Tracking Sheets**: Log and track defects identified during testing phases.
    - *Sample Tool*: Notion provides a Software Development Lifecycle Template that can be customized for defect tracking.
      Notion

6. Maintenance

- **Maintenance Plan**: Outlines procedures for updates, bug fixes, and enhancements post-deployment.
    - *Sample Template*: Cannon Quality Group offers a Software Development Lifecycle Template that includes maintenance planning.
      Cannon Quality Group, LLC

Utilizing these templates and tools can streamline the documentation process, ensuring comprehensive coverage of all aspects of the SDLC.

Feasibility Study:

>> A feasibility study is an important phase in the Software Development Life Cycle (SDLC) that helps determine whether a proposed software project is viable and worth pursuing.
>> It involves evaluating various aspects, such as technical, operational, economic, legal, and scheduling feasibility.
>> The goal is to assess the project's potential success and identify any potential risks or challenges before investing significant resources.

Purpose: To determine if a project is viable (technically, economically, legally, operationally, and schedule-wise).

- Technical: Can the system be built with the available technology?
- Economic: Is the project cost-effective?
- Legal: Does it comply with laws and regulations?
- Operational: Will the system work effectively in the intended environment?
- Schedule: Can the project be completed on time?

Feasibility study of some case studies:

1. E-Commerce Platform for Small Businesses
   > This feasibility study evaluates the development of an e-commerce platform targeted at small businesses.
   > The platform aims to simplify product listing, order management, and online payment integration while being affordable and easy to use.
2. Existing Systems
   + Shopify: Expensive and complex for small-scale businesses.
   + Wix: Limited scalability for growing businesses.
   + Custom Websites: Costly to develop and maintain.
3. Proposed Solutions
   + Solution 1: Develop a custom platform using open-source technologies like WordPress WooCommerce.
   + Solution 2: Partner with an existing SaaS platform and offer a customized version.
   + Solution 3: Build a new platform from scratch using modern web frameworks (e.g., React, Node.js).
4. Cost-Benefit Analysis
   The Actual cost might look like

	Small	Medium	Large
Design	<$1,000	$1,000-$5,000	$10,000-$80,000
Functionality	<$2,000	$5,000 - $12,000	$20,000 - $100,000
Fulfillment	Free	$30 - $400	$5,000
Marketing	$0 - $500	$500 - $2,000	$5,000
Maintenance	$0 - $500	$500 - $1,250	$3,600 - $12,000
Essentials	$300	$6,000	$10,000

Category	Estimated Cost	Benefits
Custom Platform	$50,000 (initial)	Scalable, fully tailored.
SaaS Customization	$30,000 (initial)	Faster to market, reduced maintenance.
From-Scratch Build	$70,000 (initial)	Fully customizable, long-term ownership.

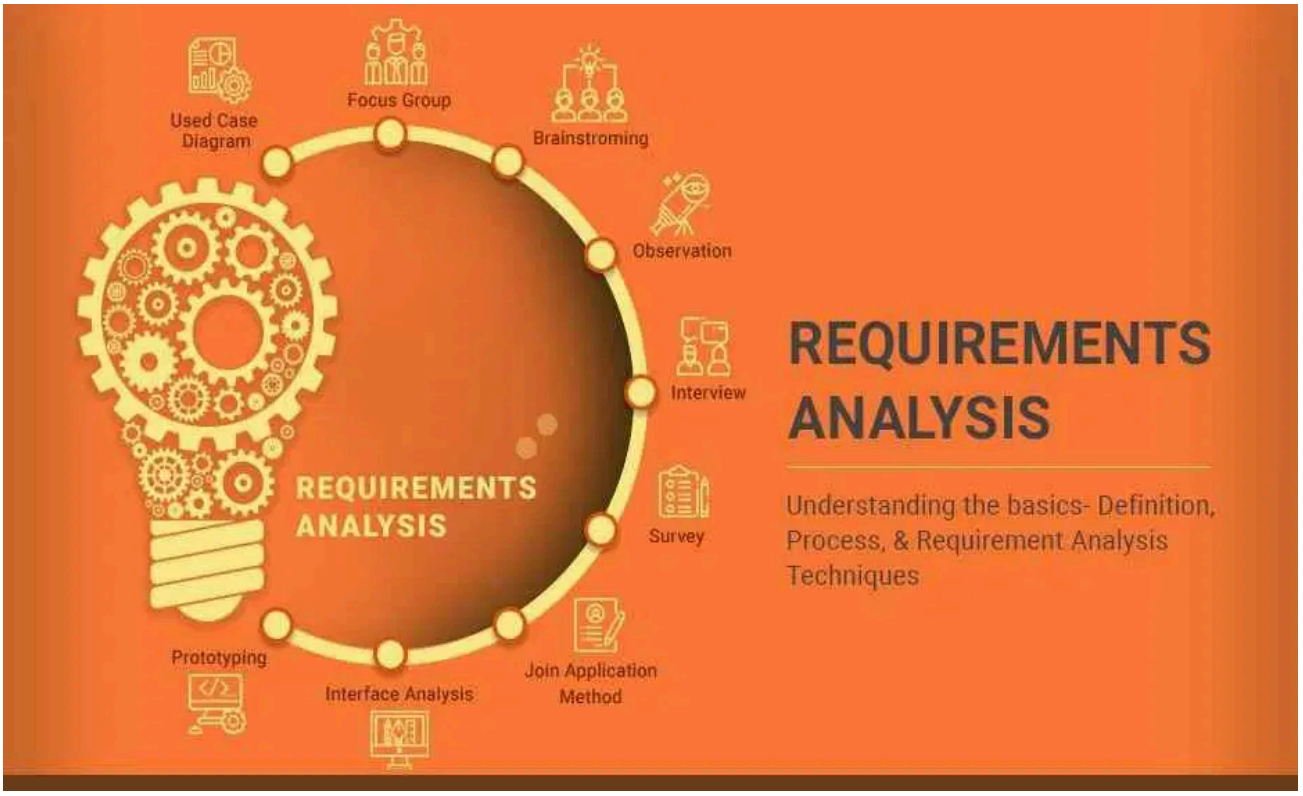
- **Recommendation:** Solution 2 (SaaS customization) offers the best balance of cost, speed, and functionality.

5. Risk Assessment

Risk	Impact	Mitigation
High initial cost	Medium	Secure funding through grants.
Limited technical staff	High	Hire freelancers or contractors.
Changing business needs	High	Use modular design principles.

6. Recommendations
   - Proceed with Solution 2 (SaaS customization).
   - Engage an experienced SaaS development team.
   - Pilot the platform with a small group

Requirements Analysis



Software requirement means a requirement that is needed by software to increase the quality of software products.
These requirements are generally a type of expectation of the user from a software product that is important and needs to be fulfilled by software.
Analysis means to examine something in an organized and specific manner to know complete details about it.
Therefore, Software requirement analysis simply means complete study, analyzing, describing software requirements so that requirements that are genuine and needed can be fulfilled to solve problems.

There are several activities involved in analyzing Software requirements. Some of them are given below :

1.Problem Recognition

2.Evaluation and Synthesis

3.Modeling

4.Specification

Requirement analysis of some case studies:

1. Submission System:

- The system supports students, lecturers, and course administrators.
- All users must log in to access features. Students can view and manage their data, enroll in courses, upload papers, and access their grades.
- Lecturers create tasks, assess assigned papers, provide feedback, and assign grades.
- Administrators manage courses, assign lecturers, allocate papers, and access all course data.
- Courses can be created, updated, or deleted, with at least one administrator assigned.
- Grades are based on points from submissions. User data is imported from an external system, eliminating manual entry. Administrators finalize grades and issue certificates for course completion.

2. Automated Teller Machine (ATM) Software

Background:

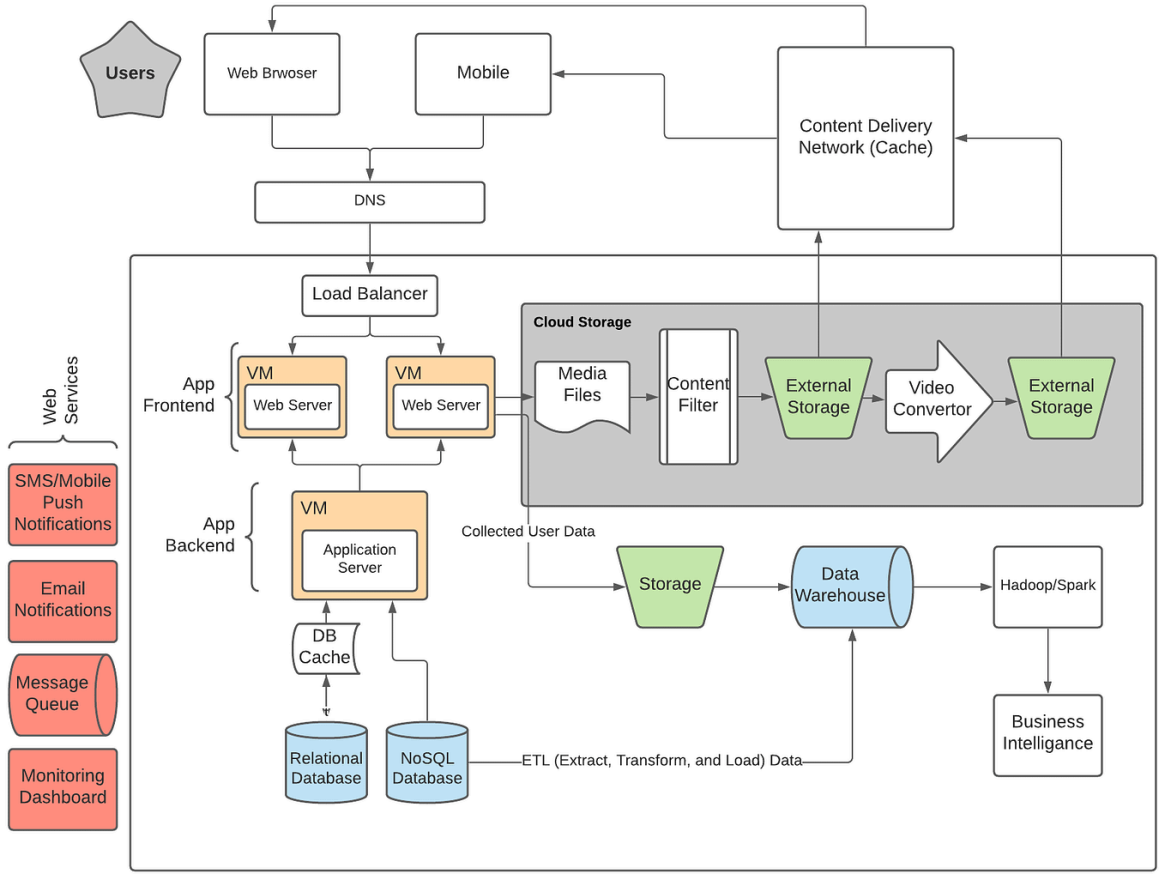
A bank aimed to develop software for its ATM network to improve customer experience and streamline operations.

Requirement Analysis Process:

- **Stakeholder Identification:** Included bank managers, IT staff, and customers.
- **Elicitation Techniques:** Interviews, surveys, and focus groups with stakeholders.
- **Functional Requirements:**
    - Enable cash withdrawal, deposit, and balance inquiries.
    - Support multi-language interfaces.
- **Non-Functional Requirements:**
    - 99.9% system uptime.
    - Maximum transaction response time of 3 seconds.

Outcome: The software successfully met customer needs, reduced transaction errors, and increased system reliability.

Design:



- **Interface Design**: Plans user interaction with UI wireframes, UX flows, and API specifications.

- **Security Design**: Includes authentication, encryption, and secure communication protocols.

- **Data Design**: Focuses on data storage, organization, and access using ERDs, schemas, DFDs, and validation mechanisms.

- **Performance and Scalability Design**: Ensures load balancing, failover strategies, and system optimization.

Development:

- **Code Implementation**: Translating design specifications into actual code using chosen programming languages and frameworks.

- **Integration**: Combining developed services and ensuring they communicate and work together seamlessly.

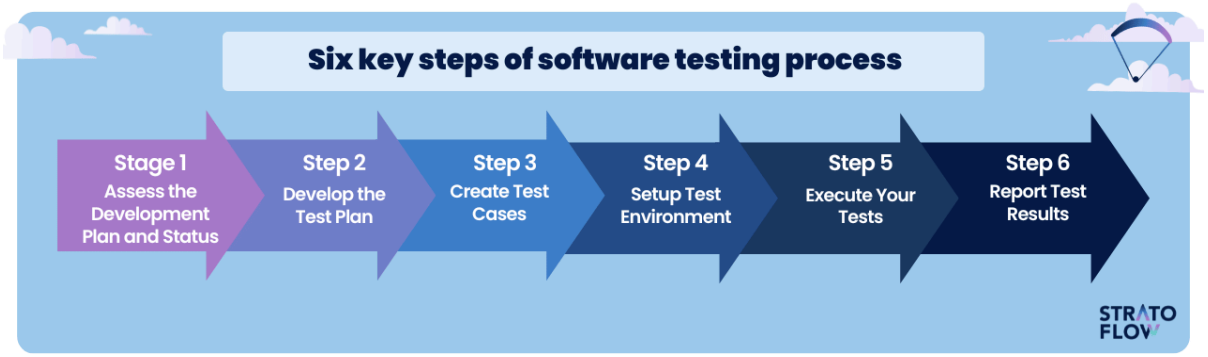
- **Error Handling**: Implementing mechanisms to detect, log, and manage errors gracefully.

- **Version Control**: Using tools like Git to track changes, manage code versions, and collaborate effectively.

- **Documentation**: Creating code-level documentation for maintainability, including comments, readme files, and API references.

TESTING

- The Testing Phase in the Software Development Life Cycle (SDLC) ensures the software meets quality standards and functions as expected. It identifies and fixes defects, ensuring the system aligns with requirements.



The Testing Phase in the Software Development Life Cycle (SDLC) ensures the software meets quality standards and functions as expected. It identifies and fixes defects, ensuring the system aligns with requirements.

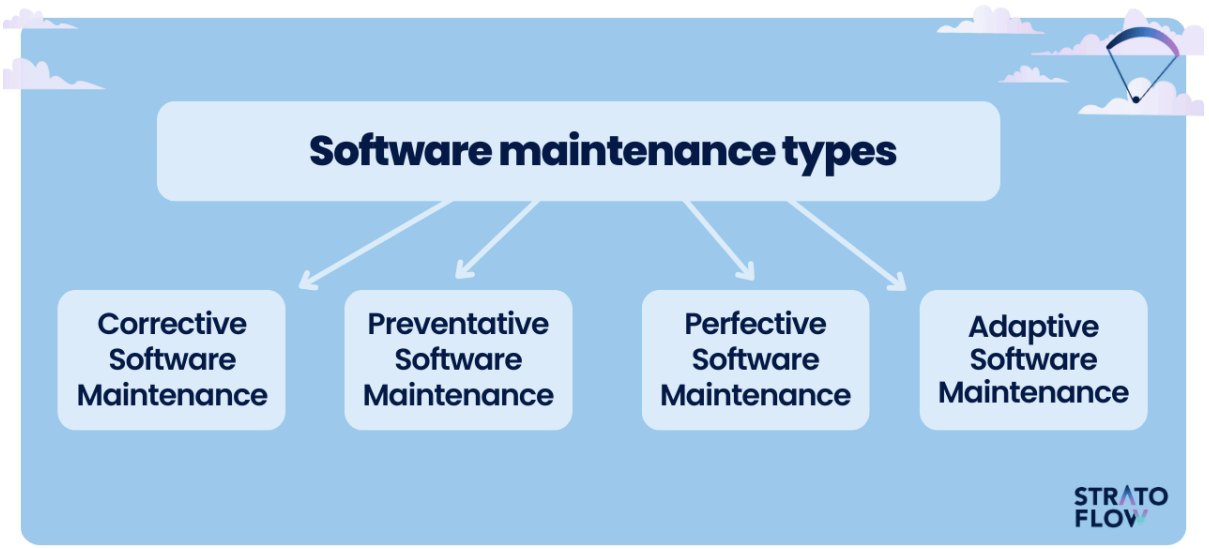
Key Activities

1. Test Planning:

    - Define test objectives, scope, and strategies.
    - Prepare test plans and test cases.
2. Types of Testing:
    - Unit Testing: Validate individual components or modules.
    - Integration Testing: Verify interactions between integrated modules.
    - System Testing: Assess the system's compliance with requirements.
    - Acceptance Testing: Confirm the system meets end-user needs.
    - Performance Testing: Evaluate speed, stability, and scalability.
    - Security Testing: Check for vulnerabilities and threats.
3. Test Execution:
    - Execute test cases and document results.
    - Identify and report bugs or issues.
4. Bug Fixing and Retesting:
    - Developers resolve issues, and testers verify fixes.

The testing phase ensures the software is reliable, secure, and ready for deployment.

MAINTENANCE



The Maintenance Phase in the Software Development Life Cycle (SDLC) is the final stage and is critical to ensuring the long-term functionality and performance of a software system.

After the software has been deployed and is operational, this phase addresses post-deployment activities to keep the software running smoothly, adapt to changing requirements, and resolve any issues that arise.

Key Activities in the Maintenance Phase:

1. Bug Fixing

    - Address defects or errors discovered after deployment.
    - Resolve issues reported by users or identified during monitoring.
2. **Performance Enhancements**
    - Optimize the software to improve speed, scalability, or resource efficiency.
    - Upgrade components to improve system performance.
3. **Adaptation to New Requirements**
    - Modify the software to accommodate new business or technical requirements.
    - Ensure compatibility with updated operating systems, hardware, or third-party tools.
4. Preventive Maintenance
    - Perform proactive updates to prevent potential issues.
    - Refactor code to improve maintainability and reduce technical debt.
5. Corrective Maintenance
    - Fix issues caused by bugs or errors in the software that affect functionality.
    - Ensure the system operates as intended.
6. Perfective Maintenance
    - Introduce new features or enhance existing ones based on user feedback.
    - Adapt to evolving user expectations or industry standards.

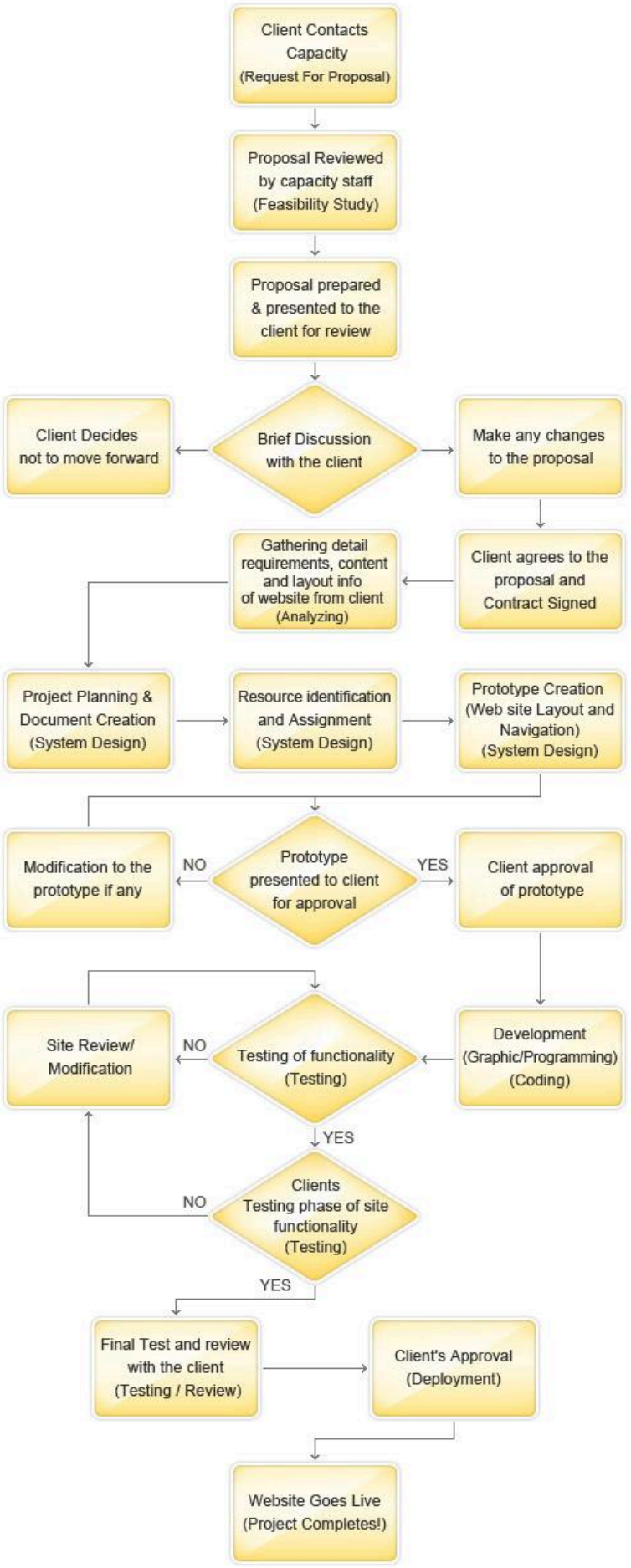
7. Monitoring and Feedback
    - Continuously monitor the system for errors, crashes, or performance bottlenecks.
    - Gather user feedback to identify areas for improvement.
8. Security Updates
    - Apply patches to address security vulnerabilities.
    - Strengthen defenses against potential threats and cyberattacks.
9. Documentation Updates
    - Keep technical and user documentation up to date with any changes or enhancements.
    - Ensure accurate records for future maintenance efforts.

Importance of the Maintenance Phase:

- Ensures the software remains relevant and functional over time.
- Helps adapt to changes in user needs, technology, and market conditions.
- Improves user satisfaction by addressing issues promptly.
- Extends the software's lifecycle by keeping it updated and secure.

The maintenance phase often represents a significant portion of the software's total lifecycle cost, emphasizing the

need for proper planning and efficient processes during this stage.


Client Contacts Capacity (Request For Proposal)
Proposal Reviewed by capacity staff (Feasibility Study)
Proposal prepared & presented to the client for review
Brief Discussion with the client
Client Decides not to move forward
Make any changes to the proposal
Client agrees to the proposal and Contract Signed
Gathering detail requirements, content and layout info of website from client (Analyzing)
Project Planning & Document Creation (System Design)
Resource identification and Assignment (System Design)
Prototype Creation (Web site Layout and Navigation) (System Design)
Prototype presented to client for approval
NO
YES
Modification to the prototype if any
Client approval of prototype
Development (Graphic/Programming) (Coding)
Testing of functionality (Testing)
NO
Site Review/ Modification
YES
Clients Testing phase of site functionality (Testing)
NO
YES
Final Test and review with the client (Testing / Review)
Client's Approval (Deployment)
Website Goes Live (Project Completes!)

Part II: Brief Descriptions of Collected Samples

Below are the descriptions of various sample charts, tables, and sheets used in different phases of the SDLC.

Each description includes its relevance, purpose, and source of reference.

1. Feasibility Study: Cost-Benefit Analysis Table

A cost-benefit analysis table outlines the estimated costs and benefits associated with a proposed project.

This table typically includes columns for cost categories (e.g., development, operations), benefit categories (e.g., revenue growth, risk reduction), and a comparison to evaluate feasibility.

Example Source: Retrieved from https://www.projectmanagementdocs.com.

2. Requirement Analysis: Use Case Diagram

A use case diagram visually represents the interactions between actors (users or systems) and the software system.

It helps in identifying functional requirements by showcasing user actions and system responses.

Example Source: "Unified Modeling Language User Guide" by Grady Booch et al.

3. Design Phase: Entity-Relationship Diagram (ERD)

An ERD is used to model the database design by representing entities, their attributes, and relationships.

It simplifies database planning by visually organizing data.
Example

Source: Sample diagrams from https://www.visual-paradigm.com.

4. Coding Phase: Pseudocode Template

Pseudocode provides a structured representation of algorithms in plain language before actual coding begins.

This template includes sections for initialization, condition checks, loops, and error handling.

Example Source: "Introduction to Algorithms" by Cormen et al.

5. Testing Phase: Test Case Template

A test case template includes fields such as test case ID, description, preconditions, test steps, expected results, and actual results.

It ensures systematic testing of the software. Example Source: Example templates from https://www.softwaretestinghelp.com.

6. Maintenance Phase: Bug Tracking Sheet

A bug tracking sheet records details about reported bugs, such as ID, description, priority, severity, status, and resolution.

It helps manage and resolve issues efficiently.

Example Source: "JIRA Software Documentation" by Atlassian.

Case Study: Online Retail Management System

The case study demonstrates how the formal documentation samples are applied in a real-world project: the development of an Online Retail Management System.

Feasibility Study

The cost-benefit analysis identified initial development costs of $50,000, with anticipated revenue growth of $120,000 in the first year.

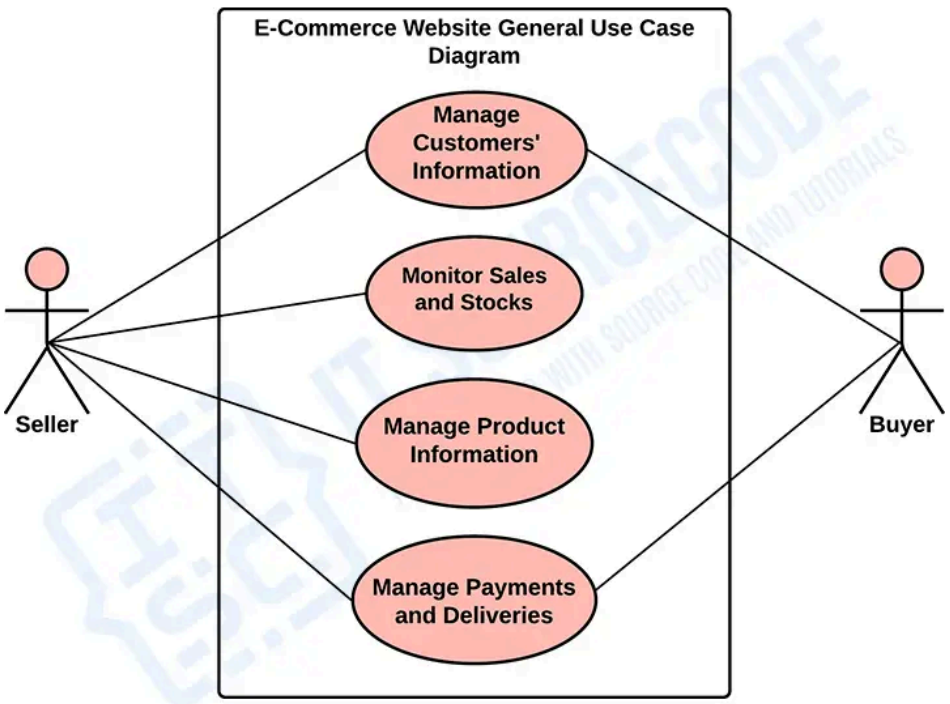
Risks of overrun were mitigated by adopting Agile.

Source: Adapted from https://www.businesscasepro.com.

Requirement Analysis

Use case diagrams captured scenarios like customer login, product search, and order placement.

E-COMMERCE WEBSITE

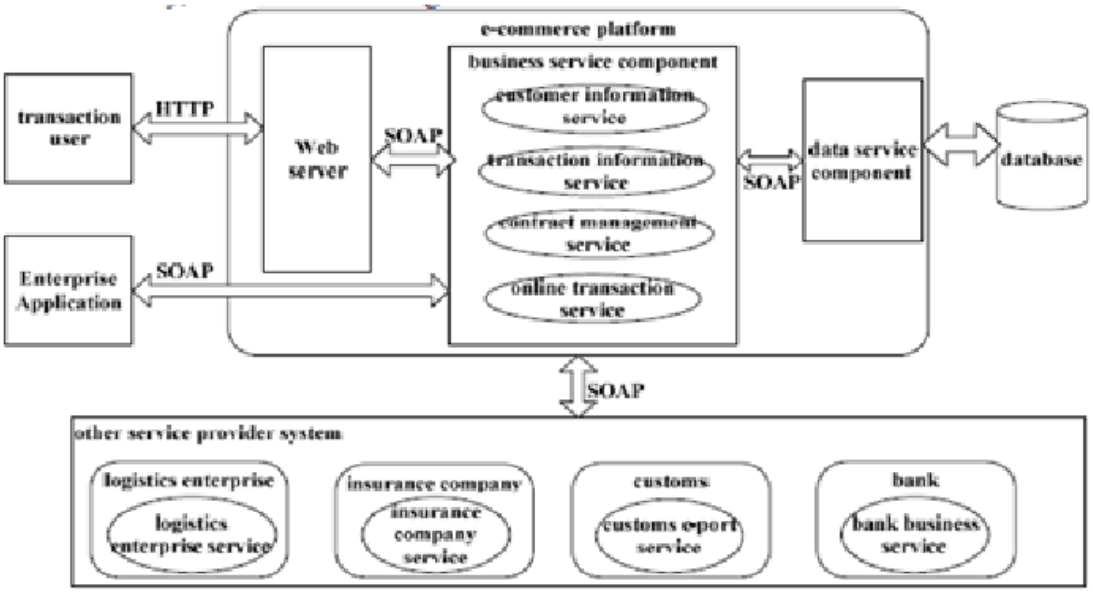


GENERAL USE CASE DIAGRAM

Key actors included customers, administrators, and suppliers. Source: Created using guidelines from "UML Distilled" by Martin Fowler.

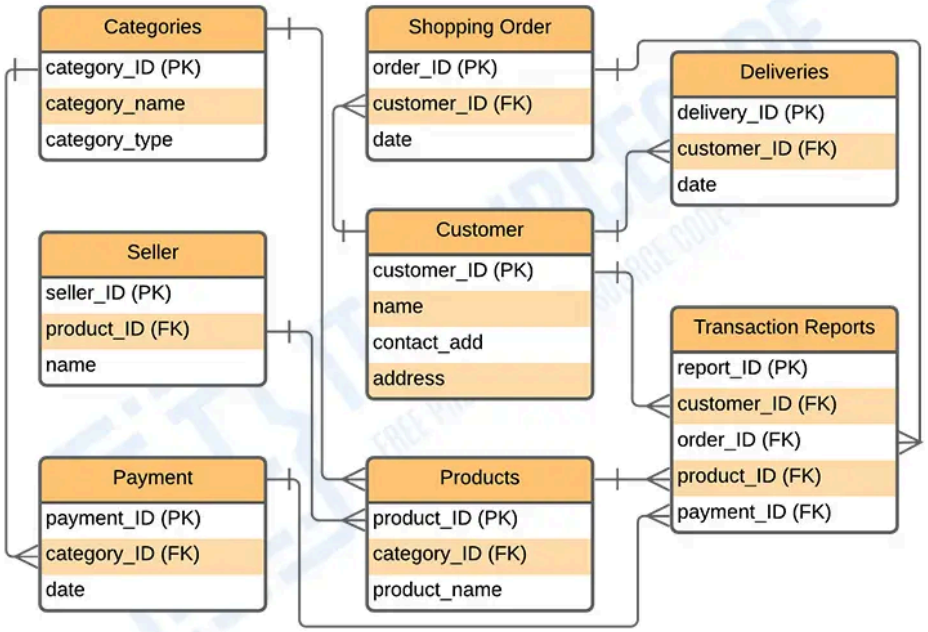
Design Phase:

- System Architecture Design



The ERD diagram

E-COMMERCE WEBSITE SYSTEM



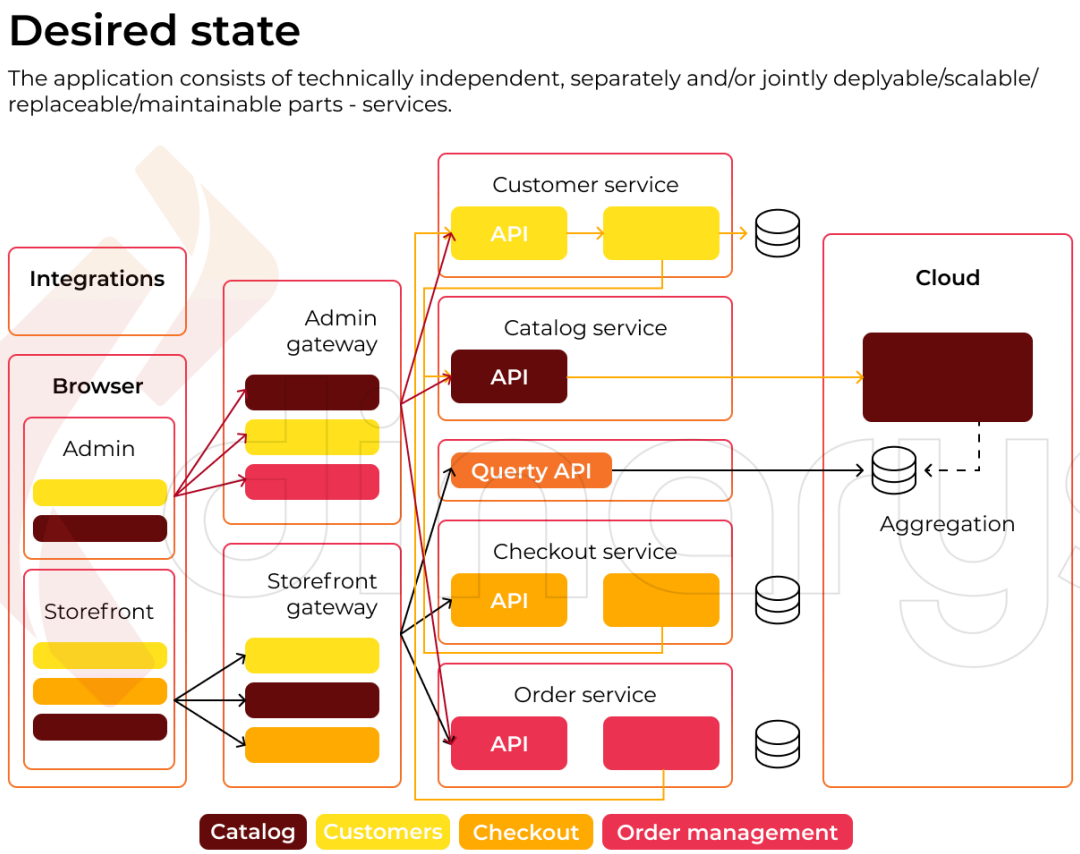
ENTITY RELATIONSHIP DIAGRAM

Relationships like "One Customer can place Many Orders" were mapped clearly.

Source: https://lucidchart.com.

Coding Phase

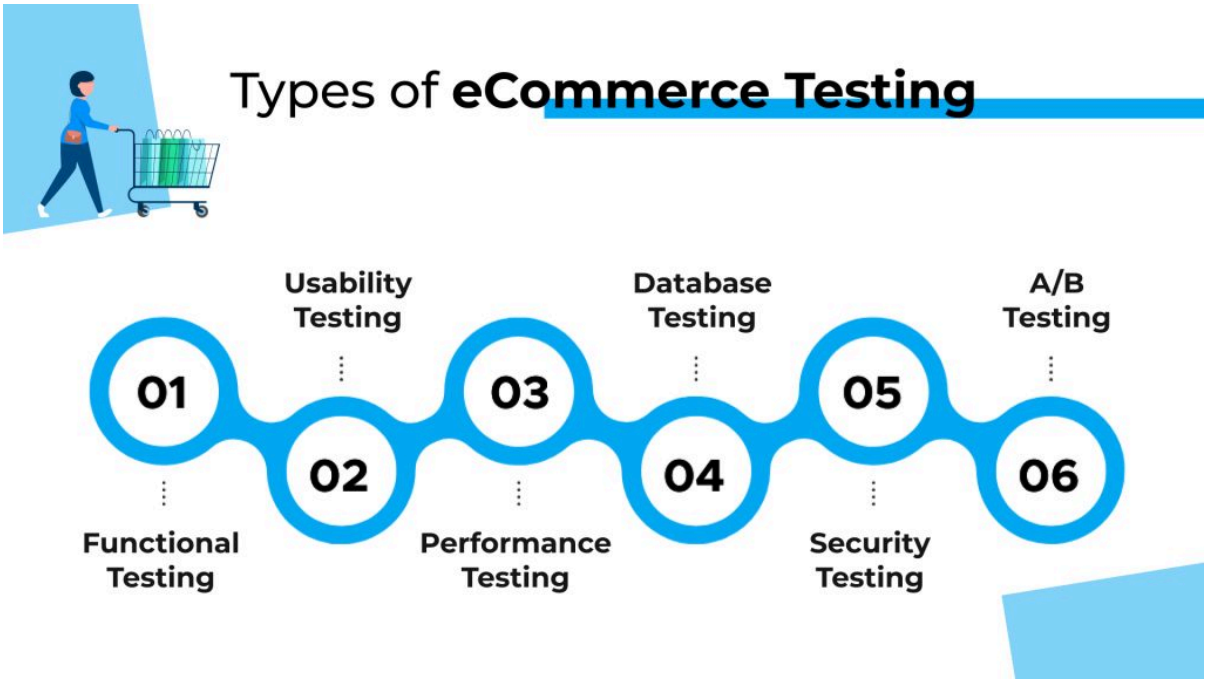
Pseudocode for the "Order Placement" module included steps for validating customer details, checking product availability, and updating the database.



Testing Phase

A test case template tested the login module.

Test steps included entering valid/invalid credentials, with expected results ensuring correct responses.



Source: Sample tests from https://www.guru99.com.

Maintenance Phase

A bug tracking sheet recorded issues like "Payment Gateway Error" and "Product Image Not Loading."

Severity levels guided the priority of fixes.

a. Corrective Maintenance

- Purpose: Fix errors or bugs discovered after the system is live.
- Examples:
    - Resolving broken features (e.g., product search not working).
    - Fixing payment gateway issues.
    - Addressing crashes during high traffic periods.

b. Adaptive Maintenance

- Purpose: Modify the system to accommodate changes in the environment or requirements.
- Examples:
    - Adapting the system to new taxation rules or regulations.
    - Integrating new payment methods (e.g., cryptocurrencies).
    - Supporting additional languages or currencies for international expansion.

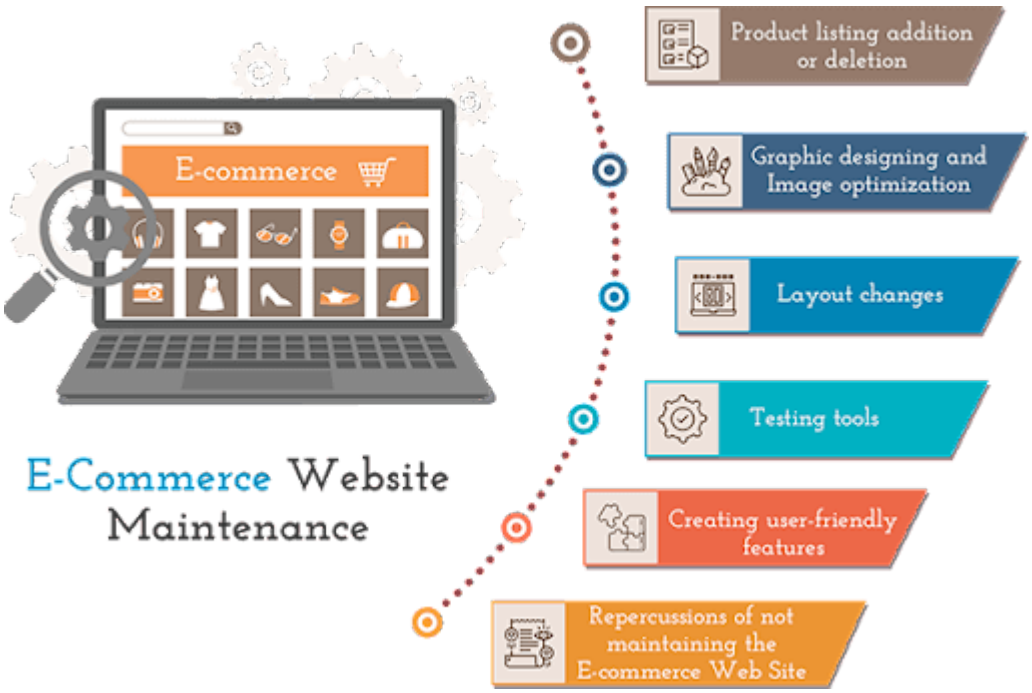
c. Perfective Maintenance

- Purpose: Improve existing features or add new functionalities based on user feedback or market trends.
- Examples:
    - Adding advanced product filtering options.
    - Enhancing the user interface (UI) for better navigation.
    - Introducing personalized recommendations using machine learning.

d. Preventive Maintenance

- Purpose: Anticipate and prevent potential problems to ensure system reliability.
- Examples:
    - Updating software libraries and dependencies.
    - Performing regular backups to avoid data loss.
    - Monitoring server performance to avoid downtime.

Source: Real-world examples from JIRA documentation.



Conclusion

Formal documentation is indispensable in the software lifecycle, ensuring clarity, consistency, and efficiency at each phase.

The examples provided demonstrate how structured documentation supports successful project outcomes.

SDLC Report: Software Acquisition for General Hospital Home Health Division

1. Introduction

- This report details the application of the Systems Development Life Cycle (SDLC) methodology in a real-world scenario involving the acquisition of a software package for the Home Health division of General Hospital, a medium-sized regional hospital.
- The project aimed to modernize the hospital's Home Health operations and enhance patient care by replacing an outdated system.
- This report outlines the stages of the SDLC utilized in this project, highlighting the key activities and outcomes of each phase.
- The case study demonstrates the continued relevance and adaptability of the SDLC in contemporary IT projects, particularly in software acquisition within the healthcare sector.

2. Analysis Phase

- The Analysis phase of the SDLC for this project encompassed three critical components:
    + Problem Definition, Feasibility Study, and Requirements Analysis.

2.1 Problem Definition

- The primary problem was the inadequacy of the existing Home Health software system at General Hospital.

- The system, approximately seven years old, was struggling to keep pace with evolving billing practices, Medicare requirements, and the expanding needs of the Home Health division, which had recently become a separate subsidiary unit.
- The core issue was the need to modernize operations with up-to-date technology and improve patient care in the Home Health arena.

2.2 Feasibility Study

Prior to the formal project initiation, a Feasibility Study was conducted by the hospital administration, addressing three key areas:

- Technical Feasibility: The hospital's IT infrastructure was assessed and deemed capable of supporting new software.
- Economic Feasibility: A budget of $250,000 was allocated for software procurement, ensuring financial viability.
- Operational Feasibility: Staff were considered trained and enthusiastic about adopting new technology.

The positive outcomes of the Feasibility Study paved the way for proceeding with the software acquisition project.

2.3 Requirements Analysis

This stage involved detailed consultations with stakeholders, including the Director of the Home Care facility and end-users, to identify the essential requirements for the new software system.

Key requirements were meticulously documented and categorized.

Critical requirements included:

- **MEDITECH Compatibility**: Seamless integration with the hospital's existing MEDITECH system was paramount.
- **Point of Care Documentation**: The system needed to support electronic medical record (EMR) point-of-care (POC) documentation.
- **OASIS Analyzer**: Integration of an OASIS Analyzer module was essential for Medicare and Medicaid reimbursement compliance and outcome analysis.
- **Physician Portal**: A dedicated portal for physician access to patient data was required to enhance care coordination.
- **Essential Features**: Home Health-specific billing and accounts receivable modules, real-time reporting with digital dashboards, schedule optimization, and user-friendliness were also mandatory.

Desirable but not essential features included advanced security and a trial period.

Further considerations during this phase included vendor experience, available modules, demonstration options, and vendor support capabilities.

3. Design Phase (Vendor-Driven)

- In this software acquisition project, the Design phase was effectively pre-existing, having been completed by the various software vendors considered.
- The researchers and hospital staff did not engage in system design in the traditional SDLC sense.

- Instead, they evaluated the designs inherent in the available software packages offered by different vendors against the **identified requirements.**

4. Implementation Phase

General Hospital chose Vendor B's software and implemented it using the Parallel Installation method, a strategy aligned with SDLC best practices.

- **Parallel Installation**:
  - For approximately 60 days, clinicians simultaneously used both the old and new systems.
  - This ensured data migration to the new system while maintaining operational continuity with the legacy system.
  - While longer than initially anticipated, this method minimises disruption during the transition.
- **Training:** Staff training commenced two weeks prior to the "go-live" date. Approximately 25 users were trained in two cohorts.
  - Hands-on training included live patient visits using the new system to ensure practical preparedness.

The Parallel Installation approach and comprehensive training facilitated a relatively smooth transition to the new software system.

5. Maintenance/Support Phase:

Post-implementation, the software vendor provides ongoing maintenance and support.

- **Regular Upgrades**: Software upgrades ("code loads") are implemented every six weeks to incorporate

improvements and address evolving healthcare industry requirements, particularly Medicare and billing changes. These upgrades are reported to be non-disruptive to daily operations.

- **User Satisfaction**: Feedback from end-users (nurses, physical therapists, physicians, and other staff) indicates high satisfaction with the new system and a lack of major complaints.

General Hospital anticipates continued use of the software for the foreseeable future, indicating a successful and sustainable solution.

Source:-
https://quod.lib.umich.edu/j/jsais/11880084.0001.103/--case-study-of-the-application-of-the-systems-development?rgn=main;view=fulltext

References

1. "Unified Modeling Language User Guide" by Grady Booch et al.
2. "Introduction to Algorithms" by Cormen et al.
3. https://www.projectmanagementdocs.com
4. https://www.visual-paradigm.com
5. https://www.softwaretestinghelp.com
6. https://lucidchart.com
7. https://www.guru99.com
8. https://www.businesscasepro.com
9. "JIRA Software Documentation" by Atlassian
10. https://quod.lib.umich.edu/j/jsais/11880084.0001.103/--case-study-of-the-application-of-the-systems-development?rgn=main;view=fulltext
11. https://intersog.com/blog/development/software-development-life-cycle-case-study/
12. https://www.studocu.com/row/document/university-of-gujrat/software-requirement-specification/software-engineering-case-studies/49147117
13. IEEE Explore articles on ATM system design.