SYSTEM ANALYSIS AND DESIGN

CSC 307

DEPARTMENT OF COMPUTER SCIENCE
FEDERAL COLLEGE OF EDUCATION, ZARIA

 COVER

Table of Contents

1.0 Introduction

A system is a collection of interconnected parts or components that work together to achieve a common goal. It can be physical or conceptual and can be found in various fields, including engineering, biology, ecology, economics, and social sciences. In a physical sense, a system might be a machine made up of interconnected mechanical components that work together to perform a specific task, such as an engine or a computer. In an abstract or conceptual sense, a system might be a set of rules or procedures that govern how people interact within a given context, such as a legal system or an economic system.

A system can be defined as a set of elements or components that are interdependent and function as a whole. These elements could be physical objects, people, processes, or even ideas that interact with one another in a specific way. For example, an ecosystem is a natural system comprising various living and non-living components, such as plants, animals, water, air, soil, and sunlight, that interact with each other to maintain a balance of life. A computer system, on the other hand, is made up of hardware components, such as the central processing unit (CPU), memory, and input/output devices, as well as software components, such as the operating system and applications. These components are interconnected and work together to perform specific functions, such as processing data or running software programs.

1.1 Information System

An information system is a set of interconnected components that work together to collect, store, process, and distribute information to support decision-making,

coordination, control, analysis, and visualization of activities within an organization or across organizations.

An information system can be composed of hardware, software, data, people, and processes that interact with each other to create, manage, and use information. The systems are designed to process, store, retrieve, and disseminate information for a variety of purposes. Here are some examples of information systems:

a. Management Information Systems (MIS): These systems are used to provide information to managers for decision-making purposes. They collect and analyze data from various sources to produce reports that provide insights into business performance.

b. Transaction Processing Systems (TPS): These systems are used to process business transactions, such as sales, purchases, and inventory updates. They are designed to be fast and accurate, and can handle large volumes of data.

c. Customer Relationship Management (CRM) systems: These systems are used to manage interactions with customers, including sales, marketing, and customer service. They provide insights into customer behavior and preferences, and help organizations develop targeted marketing campaigns.

d. Geographic Information Systems (GIS): These systems are used to capture, store, manipulate, analyze, and display geospatial data. They are used in a variety of fields, including urban planning, natural resource management, and environmental science.

e. Decision Support Systems (DSS): These systems are used to help people make decisions by analyzing data and providing insights. They are often

used in business and government settings to help managers make strategic decisions.

These are just a few examples of information systems. There are many other types of information systems that are used in a variety of industries and settings.

1.2 System Analysis and Design?

System Analysis and Design (SAD) is defined as a structured approach used to develop or improve an information system. It involves a detailed study of the current system, identifying the user requirements, analyzing the system, and designing the new system. The goal of SAD is to develop an efficient and effective system that meets the user's requirements and solves the business problem.

1.3 The Importance of Conducting System Analysis and Design

System analysis and design are crucial in the software development process as they ensure that the software being developed meets the needs of the end-users and the organization that will be using it. The following are some of the key reasons why system analysis and design are essential in developing software:

i. *Identify and define requirements*: System analysis and design help to identify and define the requirements of the software. This process ensures that the software being developed will meet the specific needs of the users and the organization.

ii. *Ensure cost-effectiveness*: Proper analysis and design of a system can identify areas where costs can be reduced, while still achieving the desired outcome. This process can help organizations save money by eliminating unnecessary features or components.

iii. *Optimize performance*: System analysis and design ensure that the software is designed to perform optimally, with a focus on speed, accuracy, and reliability. This process helps to minimize downtime, reduce errors, and improve overall productivity.

iv. *Enhance usability*: Analysis and design also ensure that the software is designed to be user-friendly, with an interface that is intuitive and easy to use. This process helps to ensure that users can operate the software without requiring extensive training.

v. *Improve scalability*: Analysis and design ensure that the software can be scaled up or down easily as the needs of the organization change. This flexibility is essential for organizations that experience rapid growth or need to downsize.

vi. *Minimize risk*: Analysis and design help to identify potential risks and issues with the software before it is developed. This process helps to ensure that any problems are addressed before they become costly or time-consuming to fix.

In summary, system analysis and design are critical in the software development process as they ensure that the software being developed meets the specific needs of the users and the organization, is cost-effective, performs optimally, is user-friendly, is scalable, and minimizes risk.

2.0 System Analyst

A system analyst is a professional who analyzes, designs, and implements information systems to help organizations achieve their goals. They typically work with stakeholders to understand their requirements and then translate those requirements into technical specifications for developers to implement.

2.1 Role of a System Analyst

System Analysts play a vital role in bridging the gap between business needs and technical solutions. Some of their core duties include:

i. Requirement Gathering: They interact with stakeholders to gather and understand their requirements for a new system or an improvement to an existing one.

ii. Analysis and Design: They analyze the gathered requirements and design a system that meets the needs of the stakeholders. This involves creating system models, such as data flow diagrams or entity-relationship diagrams, to visualize the system's structure and behavior.

iii. Communication: They act as a liaison between technical teams and stakeholders, ensuring that the technical solution aligns with the business requirements.

iv. Implementation Support: They may assist in the implementation of the system, providing guidance to developers and ensuring that the final product meets the specified requirements.

v. Testing and Quality Assurance: They participate in testing activities to verify that the system functions correctly and meets the stakeholders' expectations.

vi. Documentation: They document the system requirements, design specifications, and other relevant information to facilitate future maintenance and support.

2.2 Tasks of a System Analyst

To effectively bridge the gap between business needs and technical solutions, system analysts perform a variety of crucial tasks, including:

a. Understanding Business Needs: System analysts must understand the business processes and goals of the organization to effectively design information systems that support those objectives.

b. Requirement Elicitation: They gather requirements from stakeholders through interviews, surveys, and other techniques to understand what the system needs to accomplish.

c. System Design: They create detailed specifications for the system, including its architecture, data structures, and functionality.

d. Collaboration: They collaborate with developers, designers (UI/UX designers), and other stakeholders to ensure that the system meets the agreed-upon requirements and specifications.

e. Problem-Solving: They troubleshoot issues that arise during the development and implementation of the system, finding solutions that meet both technical and business needs.

2.3 Attributes of a System Analyst

To thrive in this role, system analysts possess a unique blend of attributes that bridge the gap between business needs and technical solutions. Here's what makes a great system analyst:

i. Analytical Thinking: System analysts must be able to analyze complex problems and break them down into manageable components.

ii. Communication Skills: Effective communication is essential for gathering requirements, explaining technical concepts to non-technical stakeholders, and collaborating with team members.

iii. Attention to Detail: System analysts need to pay close attention to detail to ensure that system requirements are accurately documented and implemented.

iv. Problem-Solving Ability: They should be able to identify and solve problems efficiently, especially when faced with unexpected challenges during system development.

v. Adaptability: Technology and business requirements can change rapidly, so system analysts must be able to adapt to new situations and technologies.

2.4 Skills of a System Analyst

Beyond technical knowledge, system analysts need a blend of soft and hard skills set to excel. The following are some of the skills a system analyst should posses:

i. Technical Skills: System analysts should have a strong understanding of information technology, including programming languages, databases, and system architectures.

ii. Business Acumen: They need to understand the business processes and goals of the organization to design systems that effectively support those objectives.

iii. Project Management: Knowledge of project management methodologies and tools helps system analysts coordinate with teams and ensure projects are completed on time and within budget.

iv. Interpersonal Skills: Building rapport with stakeholders and collaborating effectively with team members are essential skills for system analysts.

v. Documentation Skills: They should be proficient in documenting system requirements, specifications, and other project-related information in a clear and organized manner.

3.0 The System Development Life Cycle

The System Development Life Cycle (SDLC) is a framework for software development that outlines the stages involved in the development of information systems. The SDLC process provides a structured and standardized approach to software development, which helps to ensure that software is developed efficiently, meets user needs, and is of high quality. It is a process followed by software development teams to design, develop, test, and maintain software products. The SDLC consists of seven stages, which are:

i. Planning
ii. Analysis
iii. Design
iv. Development
v. Testing
vi. Deployment
vii. Maintenance

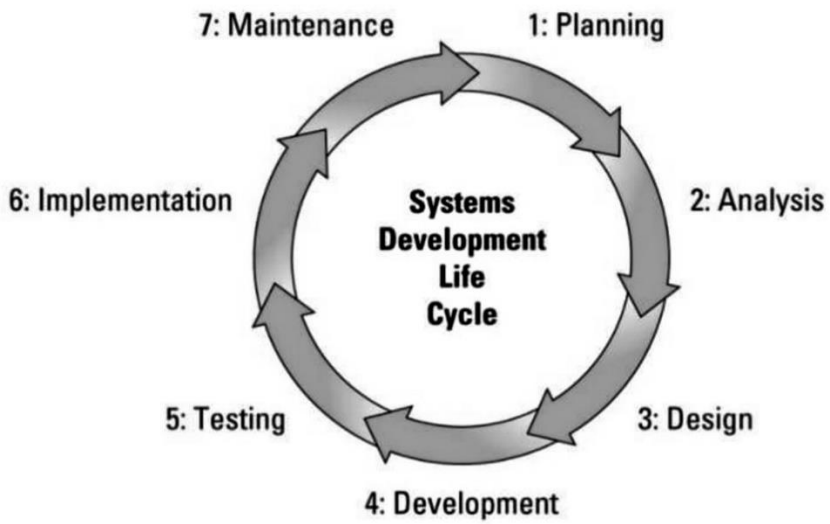

Figure 3.1 Software Development Life Cycle

3.1 Planning Phase

Planning is the first stage of the Software Development Life Cycle (SDLC) and is a critical phase that sets the foundation for the entire software development process. During the planning stage, the project team identifies and defines the software development project's goals, objectives, requirements, and constraints. The following are some of the key activities that are typically performed during the planning phase of the SDLC:

i. *Defining the project scope*: The project scope refers to the specific tasks, deliverables, features, and objectives that the software development project will encompass. It is essential to establish a clear and concise project scope to ensure that everyone involved in the project has a shared understanding of the project's goals.

ii. *Assessing feasibility*: The feasibility assessment involves evaluating the project's technical, economic, and operational viability. The team examines

the technical feasibility of the project, the financial resources required to complete the project, and the project's impact on the organization's operations.

iii. *Defining project objectives*: During the planning phase, the project team sets the project objectives, which are specific, measurable, achievable, relevant, and time-bound. Objectives should be SMART to ensure that they are achievable and contribute to the overall success of the project.

iv. *Creating a project plan*: The project plan is a document that outlines the project's timelines, budgets, and resource requirements. The project plan serves as a roadmap for the project team and helps to ensure that the project stays on track.

v. *Creating a project team*: During the planning stage, the project team is identified and assembled. The team may include developers, designers, testers, project managers, and other stakeholders who will be involved in the project.

Overall, the planning stage is a critical step in the SDLC that sets the foundation for the rest of the software development process. By carefully defining the project scope, assessing feasibility, establishing project objectives, creating a project plan, and assembling a project team, the team can increase the chances of success and ensure that the project is completed on time and within budget.

3.2 Analysis Phase

Analysis is the second stage of the Software Development Life Cycle (SDLC), which follows the planning stage. During the analysis phase, the project team gathers and documents the software project's requirements, including functional and non-

functional requirements. The analysis phase is crucial because it helps ensure that the software project meets the stakeholders' needs and requirements. The following are some of the key activities that are typically performed during the analysis phase of the SDLC:

i. *Identifying stakeholders*: The first step in the analysis phase is to identify the stakeholders who will use the software product. These stakeholders may include end-users, customers, management, and other stakeholders who have an interest in the project.

ii. *Gathering requirements*: Once the stakeholders have been identified, the project team will work to gather requirements from them. Requirements may be functional, such as specific features and functionality that the software should provide, or non-functional, such as performance, security, and usability requirements.

iii. *Analyzing requirements*: After the requirements have been gathered, the project team will analyze them to ensure that they are complete, accurate, and consistent. This analysis helps to identify any gaps or inconsistencies in the requirements and ensure that they are aligned with the project objectives.

iv. *Creating a requirements document*: Once the requirements have been analyzed, the project team will create a requirements document that outlines the software's functional and non-functional requirements. The requirements document serves as a blueprint for the development team, providing a clear understanding of what needs to be built and tested.

v. *Reviewing the requirements document*: The requirements document is reviewed by stakeholders and the development team to ensure that all requirements

are complete, accurate, and consistent. Any discrepancies or gaps in the requirements are addressed before proceeding to the next stage of the SDLC.

In a nutshell, the analysis phase is a critical step in the SDLC, as it helps to ensure that the software project meets the stakeholders' needs and requirements. By carefully gathering, analyzing, and documenting requirements, the development team can create a software product that meets the end-users' needs and provides value to the organization.

3.3 Design Phase

The Design stage is the third phase of the Software Development Life Cycle (SDLC) in which the requirements gathered during the previous stages are analyzed and transformed into a complete system design. In this stage, the technical details of the project are defined, including the system architecture, software modules, hardware and network requirements, and user interface. During the Design stage, the development team, including designers, architects, and developers, collaborate to create a detailed plan for the software project. This plan includes the following steps:

i. *Defining the System Architecture*: This step involves determining the overall structure of the system, including the hardware and software components, their inter-relationships, and how they interact with each other.

ii. *Creating the Functional Specification*: In this step, the software requirements gathered in the previous stages are analyzed and transformed into a detailed functional specification. This document describes the software's functionality and the expected results, which serve as the basis for the software development process.

iii. *Designing the User Interface*: The design team creates the user interface for the software, defining the layout, controls, and interaction methods that users will use to interact with the system.

iv. *Defining the Software Modules*: In this step, the system's functionality is broken down into individual modules, and each module is designed to perform a specific function.

v. *Defining Data Structures*: The design team defines the data structures used by the software and how they will be stored and accessed.

vi. *Defining the Algorithms*: The algorithms used by the software to process data are defined in this step. This includes the logic used for calculations, decision-making, and other complex operations.

vii. *Creating the Technical Specification*: The technical specification defines the software's technical details, including the programming languages, tools, and libraries used in development, the development environment, and any third-party integrations.

The Design stage serves as a blueprint for the software development process, providing the development team with a clear direction and plan for creating the software. The result of the Design stage is a comprehensive design document that serves as a reference for the development team throughout the rest of the SDLC.

3.4 Development Phase

The development stage is the fourth stage of the Software Development Life Cycle (SDLC), following the planning, analysis, and design stages. The development stage, also known as the implementation stage, is where the actual coding and

programming of the software takes place. During this stage, the software design is turned into a working product through the creation of source code, testing, debugging, and integration with other components of the software. This stage involves collaboration between developers, quality assurance personnel, and other stakeholders to ensure that the software is developed according to the project requirements and specifications. The development stage typically consists of the following sub-stages:

i. *Code development*: This stage involves the actual creation of the source code that implements the software design.
ii. *Code testing*: Once the code is developed, it must be tested to ensure that it works as expected and meets the project requirements.
iii. *Debugging*: This stage involves identifying and fixing any issues or bugs in the code.
iv. *Integration*: The individual components of the software must be integrated with each other to create a working product.

3.5 Testing Phase

The testing stage is the fifth stage of the Software Development Life Cycle (SDLC), following the development stage. During the testing stage, the software is thoroughly tested to ensure that it meets the specified requirements and is free from defects before it is released to the end-users. The testing stage is a critical part of the SDLC because it helps identify and fix any issues with the software before it is deployed. Testing can help prevent costly errors and ensure that the software functions correctly, efficiently, and securely. The testing stage can be divided into several sub-stages, including:

i. *Unit testing*: This is the first level of testing, where individual components or modules of the software are tested in isolation. Unit testing verifies that each module works as expected and meets the requirements.

ii. *Integration testing*: Once the individual modules have been tested, they are combined and tested as a group to ensure that they work together seamlessly. Integration testing verifies that the software functions correctly as a whole and that there are no conflicts or errors between different modules.

iii. *System testing*: Once the software has been integrated, it is tested as a complete system. System testing verifies that the software meets all the requirements and performs all the intended functions correctly. This can involve testing different scenarios and configurations to ensure that the software works under various conditions.

iv. *User acceptance testing (UAT)*: This is the final level of testing, where the software is tested by the end-users to ensure that it meets their needs and expectations. UAT verifies that the software is usable, reliable, and performs all the required functions. UAT also helps identify any final issues or defects that need to be addressed before the software is released.

Throughout the testing stage, it is important to keep track of progress and report any issues or defects that are found. Testing can be done manually or using automated testing tools, depending on the complexity of the software and the available resources. It also requires careful planning, coordination, and monitoring to ensure that the software functions correctly, efficiently, and securely.

3.6 Implementation Phase

Implementation is the stage of the Software Development Life Cycle (SDLC) that involves delivering the software product to the end-users after successful testing, integration, and quality assurance. The main objective of implementation is to ensure that the software is installed, configured, and operational in the production environment. Here are the various steps involved in the implementation phase of the SDLC:

i. Planning: The implementation plan is created, detailing the activities that need to be performed during the implementation phase. This includes selecting the implementation environment, defining the roles and responsibilities, and outlining the timelines.
ii. Installation: The software is installed on the target hardware and operating system. This involves setting up the software components, configuring the system parameters, and installing any required third-party software.
iii. Configuration: The software is configured to meet the specific needs of the end-users. This includes setting up user accounts, configuring access controls, and configuring system parameters.
iv. Testing: The implemented software is tested to ensure that it is functioning as expected. This includes both functional and non-functional testing, such as performance testing, security testing, and usability testing.
v. Training: The end-users are trained on how to use the software. This includes training on the features and functionalities of the software, as well as on any new processes or workflows that may be introduced.

In a nutshell, a well-planned and executed implementation process can help ensure that the software is delivered on time, within budget, and meets the needs of the end-users.

3.7 Maintenance Phase

Maintenance is the final stage of the Software Development Life Cycle (SDLC) and involves ongoing support and maintenance of the software product after it has been deployed to the production environment. The main objective of maintenance is to ensure that the software continues to function correctly and meets the evolving needs of the end-users. There are four different types of maintenance activities:

i. *Corrective Maintenance*: This type of maintenance involves fixing any bugs or errors that are discovered after the software has been deployed. It may also involve addressing any issues that were not identified during the testing phase or that arise due to changes in the production environment.

ii. *Adaptive Maintenance*: This type of maintenance involves modifying the software to accommodate changes in the user requirements or the external environment. This may include updating the software to work with new hardware or software platforms, integrating with new third-party applications, or modifying the user interface to improve usability.

iii. *Perfective Maintenance*: This type of maintenance involves improving the software's functionality or performance to meet the changing needs of the users. It may involve enhancing existing features, improving the response time of the software, or optimizing the code to reduce resource utilization.

iv. *Preventive Maintenance*: This type of maintenance involves proactively identifying and addressing potential issues before they become problems.

This may involve monitoring the software to identify areas where performance can be improved or updating the software to prevent known security vulnerabilities.

The maintenance phase is essential to ensure that the software remains relevant and useful to the end-users. It involves ongoing support and maintenance, including bug fixes, updates, and upgrades. Maintenance activities are usually carried out by a dedicated support team, who work closely with the end-users to identify and resolve issues.

4.0 Methodologies of System Development

The System Development Life Cycle (SDLC) outlines the stages involved in developing software, from initial planning to deployment and maintenance. However, while SDLC defines the general steps, there are different methodologies for implementing these stages based on project requirements and complexity, budget, team structure and risk management. These methodologies provide distinct approaches to executing SDLC, each with its own advantages and best use scenarios. Understanding these methodologies helps in selecting the most suitable approach for successful software development. Some common system development methodologies include the following:

4.1 Waterfall Model

The Waterfall Model was the first software development process model introduced and is also known as the linear sequential life cycle model. It is a structured and systematic approach to software development, where the entire process is divided into distinct phases, and each phase must be completed before moving to the next.

There is no overlapping between phases, and progress flows in one direction, like a waterfall.

4.1.1 Phases of the Waterfall Model

The Waterfall Model follows a structured sequence of phases, each depending on the completion of the previous one:

i. **Requirement Analysis** – Gathering and defining project requirements in detail.

ii. **System Design** – Creating architecture and system specifications based on requirements.

iii. **Implementation (Coding)** – Writing and developing the actual software.

iv. **Testing** – Identifying and fixing errors to ensure the system functions correctly.

v. **Deployment** – Releasing the final software for users.

vi. **Maintenance** – Providing updates and fixing issues after deployment.

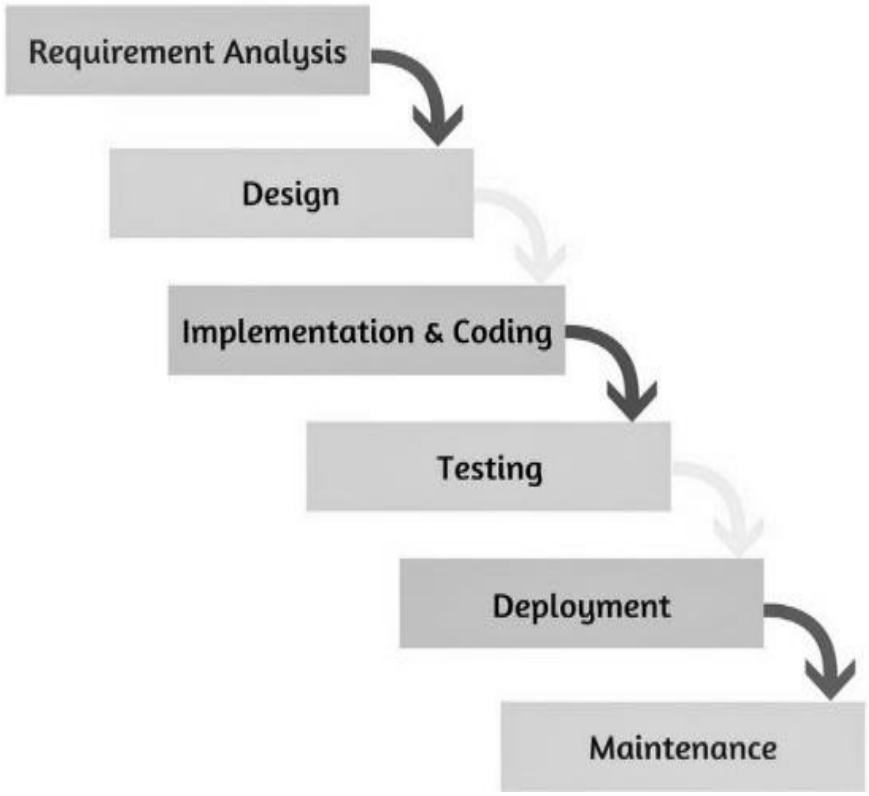

Figure 4.1 The Waterfall Model

Advantages of the Waterfall Model

i. **Simple and easy to understand**: Suitable for teams with limited experience.

ii. **Clear structure**: Well-defined phases make progress easy to track.

iii. **Strong documentation**: Provides a clear reference for future maintenance and development.

Disadvantages of the Waterfall Model

i. **Lack of flexibility**: Difficult to make changes once development has started.

ii. **Late issue discovery**: Problems may only be found during testing, making them expensive to fix.

iii. **Slow process**: Every phase must be completed before moving to the next, which can delay development.

iv. **Limited user involvement**: Users only see the final product, which may not fully meet their expectations.

4.2 Iterative Model

The Iterative Model is a software development approach where a system is built gradually through repeated cycles. Instead of developing the entire system at once, a basic version is created first, and new features are added in subsequent iterations until the final system is complete. Each cycle includes requirement analysis, design, implementation, and testing. This approach allows for continuous improvement based on feedback.

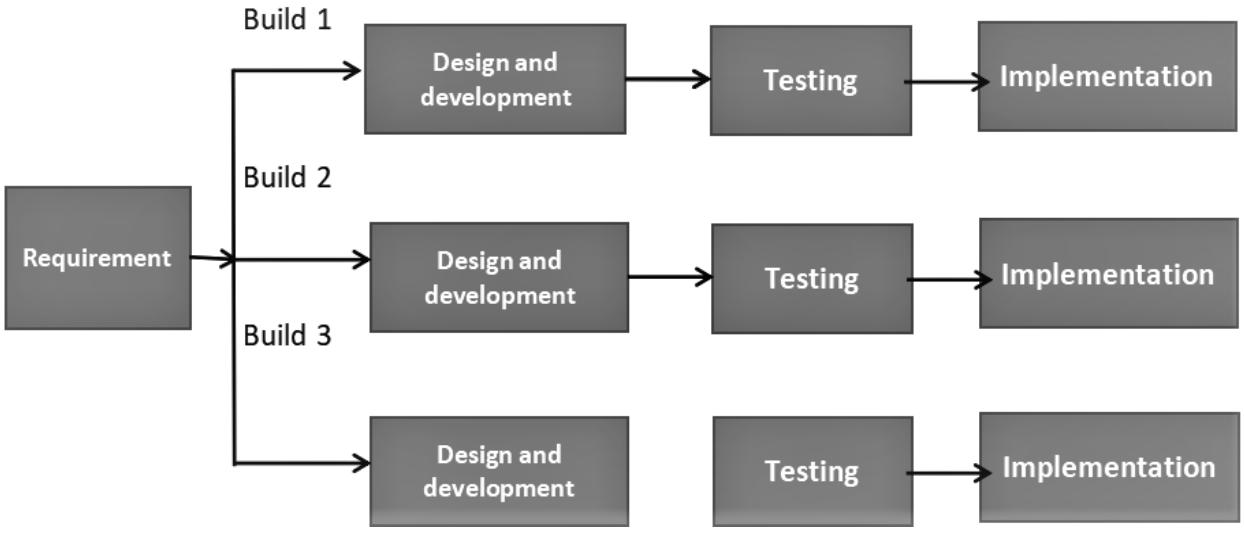

Figure 4.2 Iterative Model

Advantages of the Iterative Model

i. **Early system functionality**: A working version is available early for testing and feedback.
ii. **Better risk management**: High-risk parts are developed first to identify and solve issues early.
iii. **Flexibility in requirements:** Changes can be made in later iterations with minimal impact on cost.
iv. **Continuous testing and debugging:** Each iteration ensures quality improvements.
v. **Faster problem detection**: Design flaws and functional issues are discovered earlier.
vi. **Supports large projects**: Well-suited for **complex and mission-critical** systems.

Disadvantages of the Iterative Model

i. **Resource-intensive**: Requires skilled personnel for **design, testing, and risk analysis**.
ii. **Higher management effort**: Close monitoring is needed to track progress and risks.
iii. **Undefined end-point**: It may be difficult to predict when the project will be fully completed.
iv. **System design challenges**: Since not all requirements are gathered upfront, architectural issues may arise later.
v. **Not ideal for small projects**: Breaking down a simple system into multiple iterations can be inefficient.

4.3 Incremental Model

The Incremental Model is a software development approach where the system is built and delivered in small, manageable parts (increments). Each increment adds new functionality to the system until the final product is complete. This allows for gradual implementation, continuous testing, and learning from previous stages to improve future increments.

Key Features of the Incremental Model

- The system is divided into multiple independent modules (increments).
- Each increment undergoes requirement analysis, design, implementation, and testing before integration.
- Users can interact with early versions of the system and provide feedback.
- Reduces development risks by addressing high-priority features first.
- The final system is achieved through progressive enhancement over multiple releases.

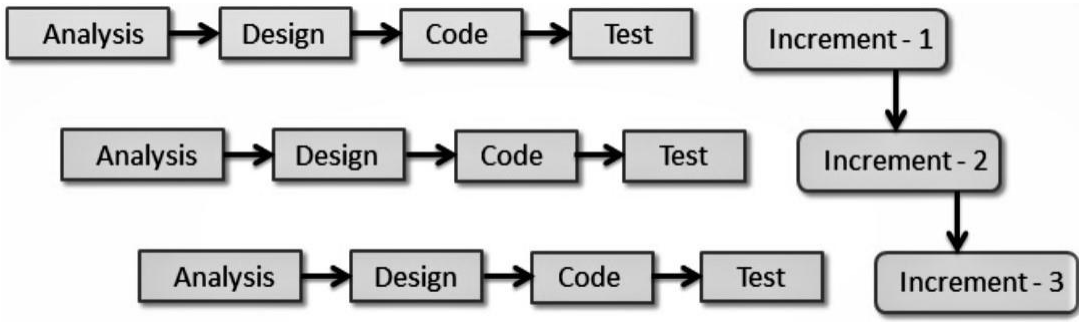

Figure 4.3 Incremental Model

Advantages of the Incremental Model

i. **Step-by-step progress:** Allows controlled development, reducing overall project risk.
ii. **Early feedback:** Users can test functional parts early, improving system alignment with needs.
iii. **Easier testing and debugging:** Smaller modules are tested independently, simplifying issue resolution.
iv. **Better resource management:** Work can be divided among teams efficiently.
v. **Flexibility in requirements:** Changes can be incorporated in later increments with minimal disruption.

Disadvantages of the Incremental Model

i. **Postponed problem-solving:** Some complex issues may be left for later, leading to challenges in future increments.
ii. **Potential loss of project vision:** Focusing too much on individual increments may lead to inconsistent system integration.
iii. **More management effort:** Requires careful planning and coordination to ensure all increments fit together properly.

4.4 Prototyping Model

The Prototyping Model is a software development approach that focuses on building an initial working version (prototype) of the system before full scale development. This prototype is used to gather user feedback, refine requirements,

and improve the final product. The goal is to reduce misunderstandings between developers and users by allowing early testing of system functionality.

Key Features of the Prototyping Model

- A prototype is developed based on initial requirements.
- Users evaluate the prototype and provide feedback.
- The system is refined and improved based on user suggestions.
- Once the prototype is approved, full development begins.
- Testing ensures the final product meets user expectations.

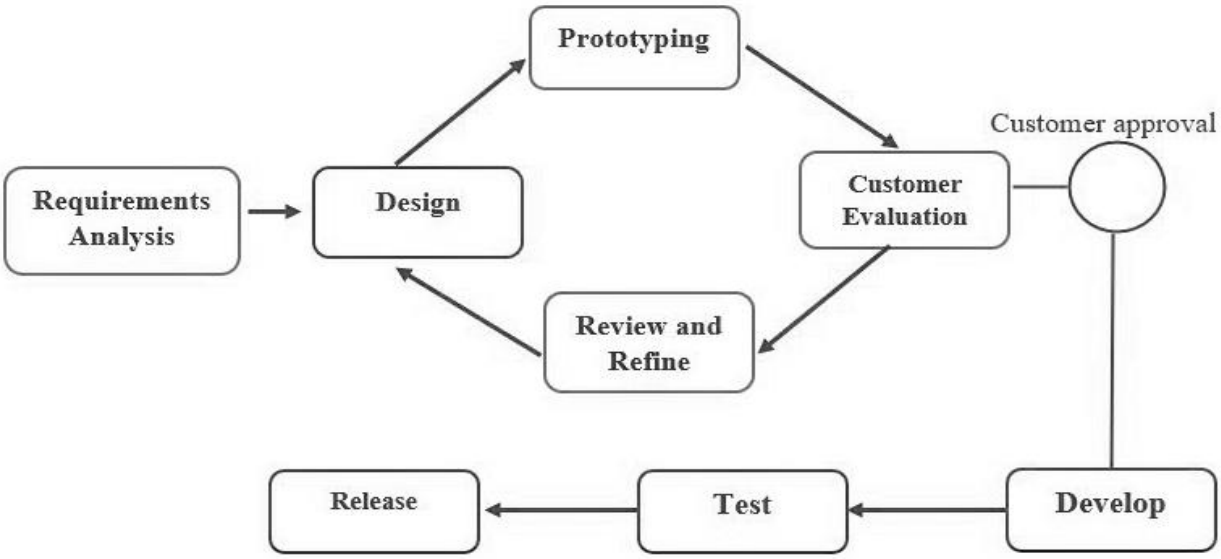

Figure 4.4 Prototyping Model

Advantages of the Prototyping Model

i. **Early user involvement**: Ensures the system aligns with user needs.
ii. **Reduces misunderstandings**: Users can visualize and refine requirements.
iii. **Flexible design**: Changes can be made before final development.

iv. **Faster problem detection**: Issues are identified early, reducing costs.
v. **Improved system quality**: Ensures better user satisfaction.

Disadvantages of the Prototyping Model

i. **Time-consuming**: Repeated refinement cycles may delay development.
ii. **Higher costs**: Creating and modifying multiple prototypes requires resources.
iii. **Unclear project scope**: Frequent changes can lead to scope creep.
iv. **Not suitable for complex systems**: Large-scale projects may require more structured methodologies.

4.5 Agile Model

The Agile Software Development Life Cycle (SDLC) model combines iterative and incremental approaches, emphasizing adaptability and customer satisfaction through the rapid delivery of functional software. Development occurs in small, incremental builds within short iterations, typically lasting one to three weeks. Throughout each iteration, cross-functional teams collaborate on planning, requirement analysis, design, coding, unit testing, and acceptance testing, ensuring continuous progress. At the end of each iteration, a functional product is demonstrated to stakeholders for feedback, allowing for necessary adjustments. Development tasks are organized into time-boxed phases, each focusing on delivering particular features, while the software undergoes continuous enhancement through iterative builds. The final version integrates all required features, ensuring it meets customer expectations efficiently.

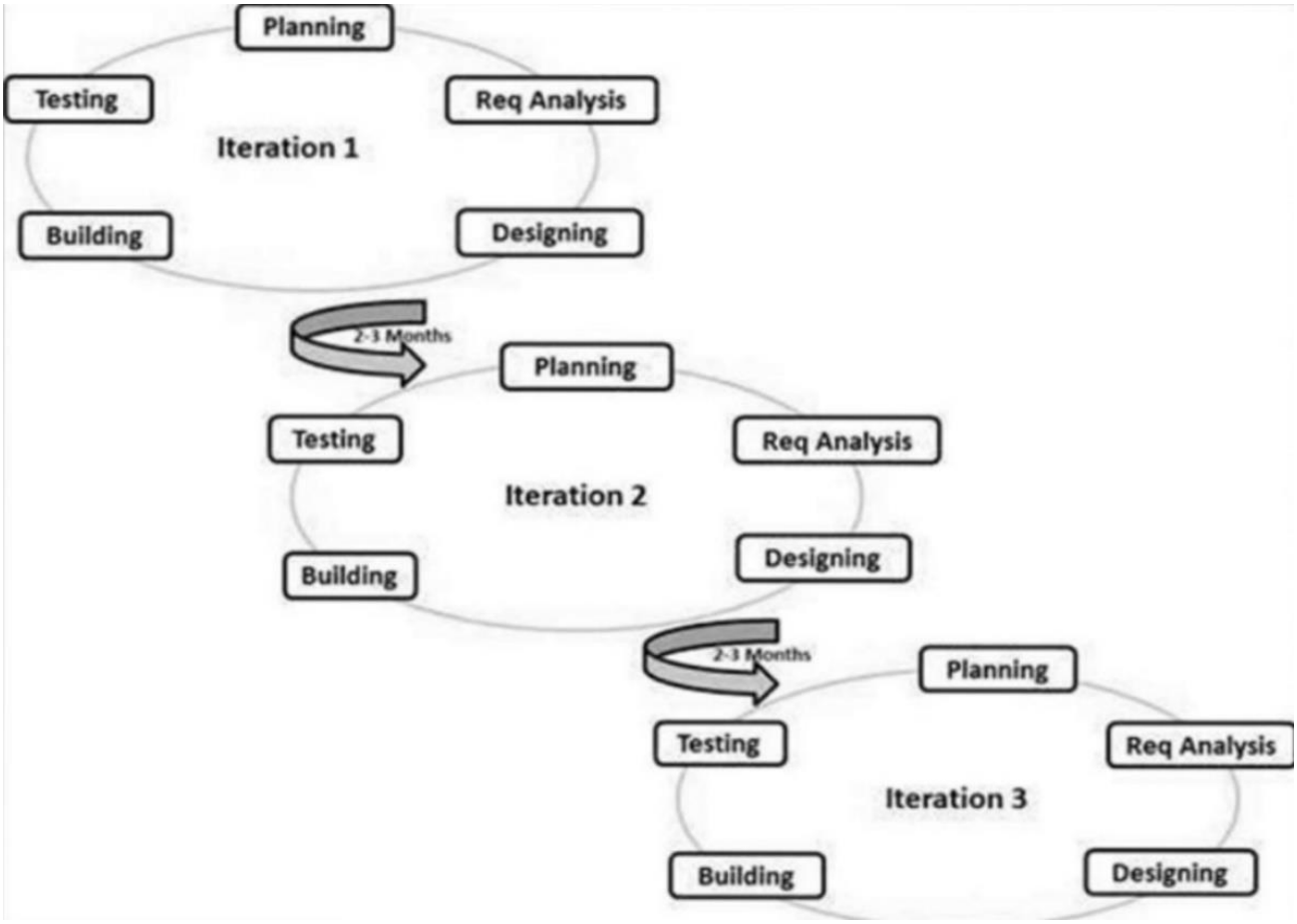

Figure 4.5 Agile Model

4.5.1 Popular Agile Methodologies

Agile is a broad approach to software development that emphasizes flexibility, collaboration, and continuous improvement. Within this approach, different methodologies provide structured ways to implement Agile principles which includes:

a) **Scrum (1995)**: This is one of the most widely used Agile frameworks. It organizes work into short, time-boxed development cycles called sprints

(typically 1–4 weeks). A Scrum Master ensures the team follows Agile principles, while daily stand-up meetings keep progress on track. At the end of each sprint, the team delivers a functional product increment, collects feedback, and adjusts accordingly.

b) **Extreme Programming (XP, 1996)**: XP enhances Agile by focusing on high-quality code and customer satisfaction. It relies on continuous testing, pair programming, and frequent small releases to ensure the software remains reliable and adaptable to changes.

c) **Feature-Driven Development (FDD)**: This method structures Agile development around customer-prioritized features. Instead of focusing on entire projects, FDD breaks work into small, functional features that are developed, reviewed, and delivered incrementally.

d) **Dynamic Systems Development Method (DSDM, 1995)**: DSDM provides a more structured Agile approach while maintaining flexibility. It ensures rapid application development (RAD) by enforcing strict time constraints and prioritizing business needs. It integrates continuous stakeholder involvement to refine the project as it progresses.

Advantages of Agile Model

i. **Rapid Functionality Development**: Allows quick development and demonstration of working features.

ii. **Team Collaboration**: Encourages teamwork, cross training, and shared knowledge.

iii. **Minimal Resource Requirements**: Can be implemented with relatively low resource needs.

iv. **Adaptability to Change**: Suitable for projects with fixed or changing requirements.

v. **Early Delivery of Working Solutions**: Ensures customers see usable features early in development.

vi. **Ideal for Dynamic Environments**: Works well in industries where requirements frequently evolve.

vii. **Lightweight Documentation**: Uses minimal documentation while maintaining flexibility.

viii. **Simplified Management**: Follows an easy-to-manage approach with minimal bureaucracy.

ix. **Developer Flexibility**: Allows developers to adapt quickly and make necessary modifications.

Disadvantages of Agile Model

i. **Challenges in Maintenance & Scalability**: Long-term sustainability, maintainability, and extensibility can be difficult.

ii. **Requires Strong Leadership & Planning**: Needs an Agile leader and structured Agile Project Management (PM) to function effectively.

iii. **Strict Delivery Management**: Scope and features must be carefully managed to meet deadlines.

iv. **Heavy Customer Involvement**: Success depends on continuous customer feedback. Unclear requirements can misdirect development.

v. **High Individual Dependency**: Minimal documentation makes knowledge transfer difficult, increasing reliance on specific team members.

4.6 Rapid Application Development (RAD) Model

Rapid Application Development (RAD) is a software development methodology that prioritizes speed and flexibility over extensive planning. It focuses on rapid prototyping, where functional components are developed iteratively, allowing for quick modifications and faster product delivery. In the RAD model, development occurs in parallel, with small cross-functional teams comprising developers, domain experts, and customer representatives working on independent prototypes that are later integrated. Since there is minimal preplanning, RAD allows for easy incorporation of changes throughout the development process.

4.6.1 Phases of the RAD Model

a) **Business Modeling**: This phase identifies the flow of information within the system by analyzing how data is obtained, processed, and distributed across various business units. It also defines key factors that drive efficient business processes.

b) **Data Modeling**: The collected business information is organized into structured data sets, where attributes and relationships between data objects are clearly defined. Additionally, data integrity rules are established to ensure alignment with business requirements.

c) **Process Modeling**: In this phase, structured data is converted into business workflows by defining operations such as data creation, modification,

retrieval, and deletion. It also specifies any necessary process enhancements to meet business objectives effectively.

d) **Application Generation**: Automated tools and code generators are used to transform process models into functional prototypes. This phase ensures rapid development and seamless integration of various software components.

e) **Testing and Turnover**: Prototypes are independently verified in each iteration, reducing overall testing time. Comprehensive testing of data flow and component interfaces is conducted to ensure the seamless integration of all modules into a fully functional product.

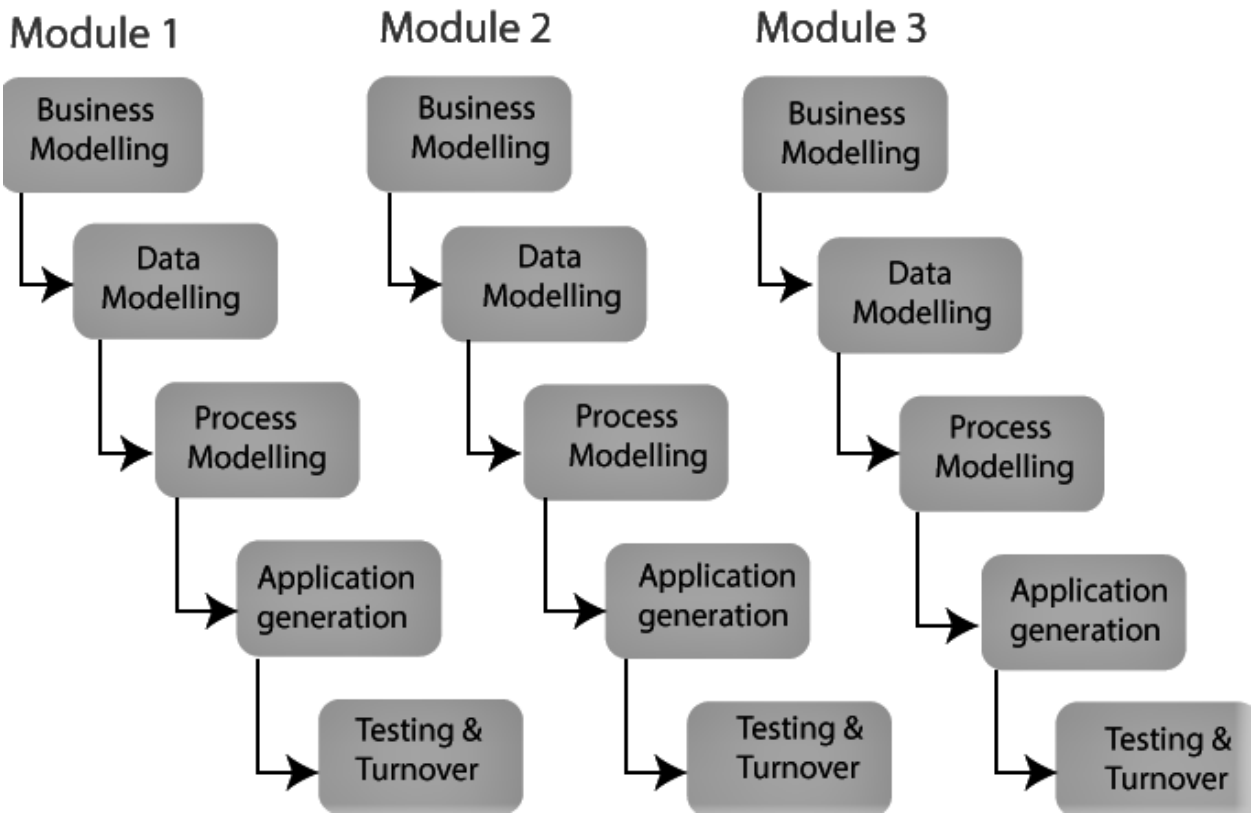

Figure 4.6 RAD Model

Advantages of the RAD Model

The **RAD (Rapid Application Development) model** offers several advantages, particularly in environments where quick development and adaptability are essential.

i. **Accommodates Changing Requirements**: RAD allows modifications and refinements at any stage without significantly disrupting the development process.

ii. **Measurable Progress**: Continuous iterations and prototype evaluations provide clear insights into development progress.

iii. **Short Iteration Time**: The use of powerful RAD tools enables rapid iterations, reducing overall development time.

iv. **High Productivity**: Requires fewer developers while maintaining a fast-paced development cycle.

v. **Reduced Development Time**: Accelerates software delivery by focusing on reusable components and automation.

vi. **Component Reusability**: Promotes the reuse of software modules, improving efficiency and consistency.

vii. **Quick Initial Reviews**: Early-stage evaluations help identify and resolve potential issues before full-scale development.

viii. **Encourages Customer Feedback**: Frequent interactions with stakeholders ensure that the product aligns with user expectations.

ix. **Early Integration**: Integration from the beginning minimizes compatibility issues and improves system stability.

Disadvantages of the RAD Model

i. **Requires Skilled Team Members**: Success depends on developers with strong technical and analytical expertise.

ii. **Limited to Modular Systems**: The RAD model is only effective for projects that can be broken down into independent modules.

iii. **High Dependency on Expertise**: Skilled designers and developers are essential to ensure successful implementation.

iv. **Relies on Strong Modeling Skills**: Development heavily depends on accurate system modeling and design.

v. **High Cost of Modeling and Automation**: Not suitable for low-budget projects due to expensive modeling and automated code generation tools.

vi. **Increased Management Complexity**: Continuous iterations and customer involvement add to project management challenges.

vii. **Best for Component-Based and Scalable Systems**: Not ideal for monolithic systems with tightly integrated components.

viii. **Requires Active User Involvement**: Success depends on continuous customer participation throughout the development cycle.

ix. **Optimized for Shorter Development Timelines**: Works best for projects with tight deadlines that require rapid deployment.

4.7 V-Model

The V-Model, also known as the Validation and Verification Model, is a structured approach in the Software Development Life Cycle (SDLC) where each development stage is directly linked to a corresponding testing phase. It follows a V-shaped sequence, ensuring early defect detection and maintaining high software quality. This model is derived from the Waterfall Model and emphasizes a disciplined, sequential process where each phase must be completed before proceeding to the next.

4.7.1 Structure of the V-Model

The V-Model consists of two parallel tracks: the left side represents Verification Phases, where planning and designing take place, while the right side covers Validation Phases, focusing on testing and evaluation. The coding phase sits at the base of the "V," bridging both sides by converting the design into a working system.

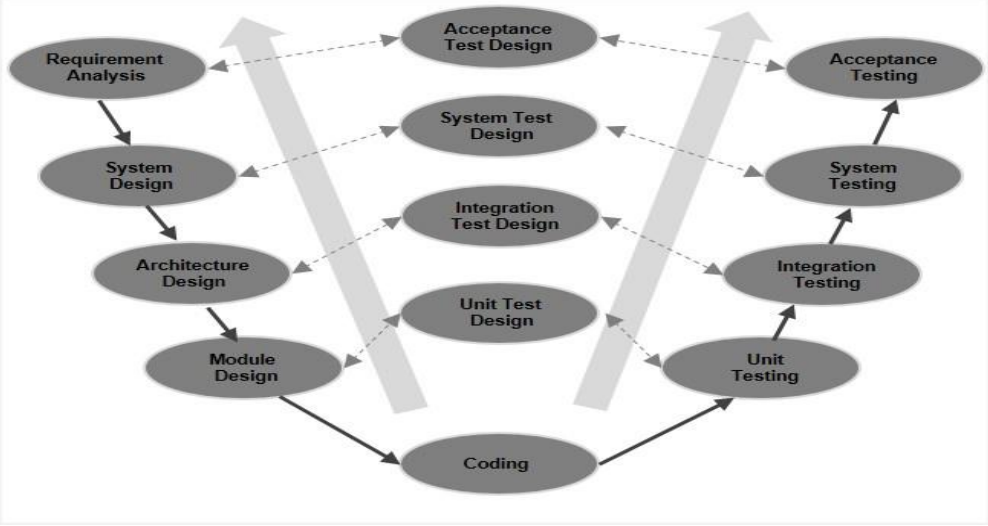

Figure 4.7 V-Model Structure

4.7.2 Verification Phases

a) **Business Requirement Analysis**: This initial phase involves gathering requirements from the client's perspective, ensuring a clear understanding of business needs. Since customers may not always articulate their requirements precisely, thorough discussions are conducted. Additionally, this phase includes acceptance test planning to validate the final product against business expectations.

b) **System Design**: After defining business requirements, the system's overall structure is planned, covering hardware, software, and communication aspects. Early system test planning is also initiated at this stage, providing ample time for thorough validation.

c) **Architectural Design (High-Level Design - HLD)**: This phase defines the system architecture, breaking it down into functional modules and specifying the communication flow. Multiple technical approaches may be evaluated, with the final choice determined based on feasibility and cost-effectiveness. Integration test plans are also formulated to ensure seamless interaction between components.

d) **Module Design (Low-Level Design - LLD)**: The internal details of each module are designed in this phase, ensuring that all components align with the overall system architecture. Unit test planning is conducted to facilitate early identification of defects within individual modules.

4.7.3 Coding Phase

The coding phase involves translating design specifications into executable system modules using appropriate programming languages. Development follows coding standards and best practices, with rigorous code reviews and performance optimizations before final integration.

4.7.4 Validation Phases

1. **Unit Testing**: The unit tests planned during module design are executed in this phase. These tests validate individual components, helping to identify and fix defects early in the development cycle.
2. **Integration Testing**: This phase ensures smooth interaction and data exchange between integrated modules, verifying that components function together correctly.
3. **System Testing**: Conducted after integration testing, this stage evaluates the entire system's functionality, ensuring that both software and hardware components operate as expected.
4. **Acceptance Testing**: The final testing phase assesses the software in a real-world environment, validating whether it meets business requirements. This phase also identifies potential performance and compatibility issues before deployment.

Advantages of the V-Model

i. **Structured and Disciplined Approach**: Each phase follows a sequential order, ensuring clarity and well-defined deliverables.

ii. **Best for Well-Defined Requirements**: Works effectively for small projects where the requirements are stable and clearly understood.

iii. **Easy to Manage**: Due to its rigid structure, progress is easily tracked, and each phase has specific review processes.

iv. **Straightforward Implementation**: Simple to understand and apply, making it accessible even for teams with minimal Agile experience.

Disadvantages of the V-Model

i. **Limited Flexibility**: Once the project moves into later stages, making changes to earlier phases is costly and time-consuming.

ii. **High Risk in Dynamic Environments**: Not suitable for projects with evolving requirements, as modifications can be expensive.

iii. **Not Ideal for Complex Systems**: Struggles with object-oriented or large-scale, ongoing projects due to its linear approach.

iv. **Late Software Availability**: No functional software is produced until the later stages, making early user feedback difficult.

4.8 Spiral Model

The Spiral Model is a software development approach that blends elements of the Iterative Development Model with the Waterfall Model, emphasizing risk management at every stage. This method enables incremental development, where each iteration refines and enhances the product before moving forward.

4.8.1 Phases of the Spiral Model

The Spiral Model is structured around four key phases, which are repeated in cycles known as spirals throughout the software lifecycle.

a) **Requirement Identification**: The process begins with defining business requirements in the initial spiral. As development progresses, system, sub-system, and module-level requirements are determined in subsequent spirals. Continuous interaction between the customer and system analyst helps refine these requirements, ensuring alignment with user needs. At the end of the spiral, the product is prepared for release in the target market.

b) **System Design**: The design phase starts with conceptual planning in the first spiral. As iterations continue, more detailed architectural, logical, physical, and final design refinements are made. This evolving approach ensures that design modifications are integrated effectively based on feedback and system requirements.

c) **Development (Build Phase)**: In this stage, the actual software product is created. The initial spiral typically produces a Proof of Concept (POC) to gather early customer input. As requirements and design details become clearer, working versions of the software, known as builds, are developed and assigned version numbers. These builds are shared with customers for review and validation.

d) **Evaluation and Risk Management**: This phase focuses on identifying and assessing technical and project-related risks, such as potential delays or budget constraints. After testing, the customer reviews the software and

provides feedback. Any identified issues are addressed in the next iteration to improve the product.

4.8.2 Iterative Development in the Spiral Model

Each cycle incorporates customer feedback, allowing for continuous enhancements while following a structured, phase-by-phase approach. This repetitive cycle of evaluation and improvement ensures that risks are mitigated early, and the software evolves efficiently to meet project requirements.

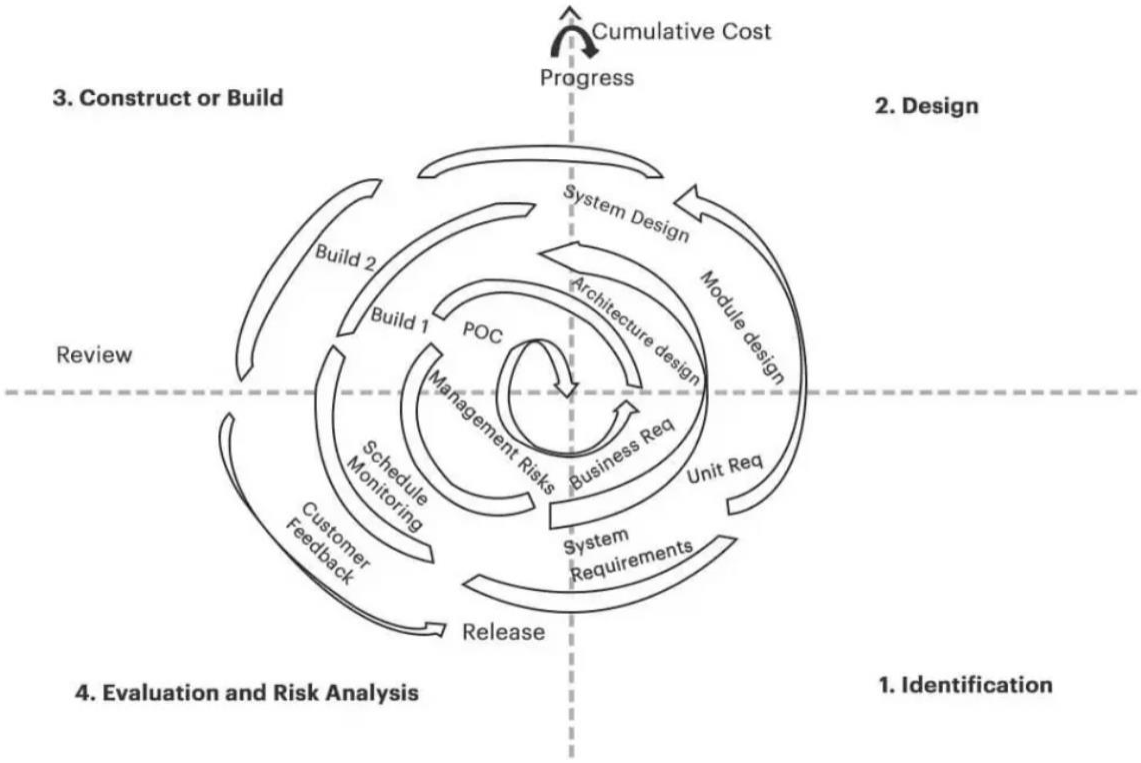

Figure 4.8 Spiral Model

Advantages of the Spiral Model

i. **Accommodates changing requirements:** The model is highly adaptable, making it suitable for projects with evolving needs.

ii. **Extensive use of prototypes:** Prototyping allows stakeholders to visualize and interact with the system at different stages.

iii. **Accurate requirement capture:** Continuous feedback ensures that user requirements are well understood and implemented correctly.

iv. **Early user involvement:** Users can interact with the system in the early stages, leading to better usability insights and improvements.

v. **Better risk management:** High-risk components can be developed and tested earlier, reducing the chances of project failure.

Disadvantages of the Spiral Model

i. **Complex management:** The iterative nature of the model makes project tracking and coordination more challenging.

ii. **Uncertain project timeline:** The end date may not be clearly defined due to continuous iterations and refinements.

iii. **Not suitable for small projects:** The model is expensive and inefficient for low-risk or small-scale developments.

iv. **Process complexity:** The development structure is intricate, requiring careful planning and execution.

v. **Possibility of infinite loops:** Without clear objectives and scope, the spiral process can continue indefinitely.

vi. **Excessive documentation:** Each phase requires extensive documentation, increasing the overall project workload.

4.9 Big Bang Model

The Big Bang Model is a software development approach that does not adhere to a structured process or predefined methodology. Development begins by utilizing the available resources such as time, money, and effort without extensive planning or requirement analysis. The final product is the developed software, which may or may not fully meet customer expectations.

This model is highly adaptable, allowing requirements to be incorporated as they emerge rather than being strictly defined from the start. It is particularly useful in projects where clients are unsure about their needs, making it well-suited for exploratory or experimental development.

4.9.1 Structure and Practical Use of the Big Bang Model

In this approach, most resources are directed toward software development and coding, with minimal emphasis on structured phases or pre-defined planning. Since requirements are often unclear at the outset, they are dynamically implemented throughout the project. As a result, there is a high likelihood of revisions or even complete redevelopment if major changes arise during development.

The Big Bang Model is best suited for small-scale projects, especially those handled by individual developers or small teams. It is commonly used in academic environments, experimental software projects, and development scenarios where rigid requirements and strict deadlines are not a priority. This model is ideal when

project goals are uncertain, allowing developers the freedom to experiment, iterate, and refine the software as needed.

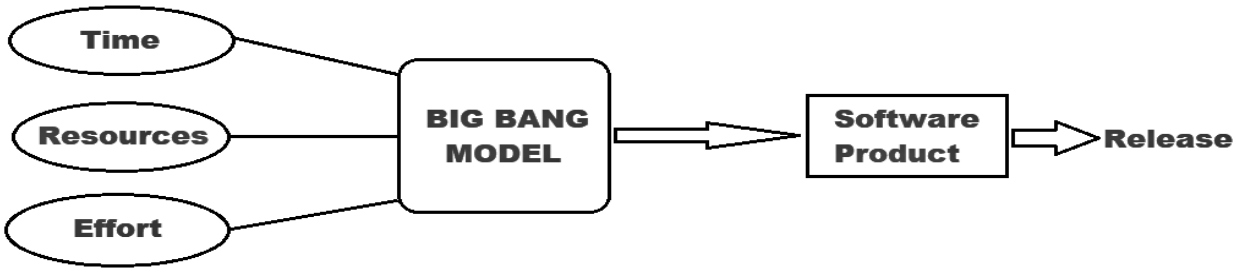

Figure 4.9 The Big Bang Model

Advantages of the Big Bang Model

1. **Simplicity:** This model is straightforward and easy to implement.
2. **Minimal Planning:** Requires little to no initial planning before development begins.
3. **Ease of Management:** Since it lacks structured phases, it is simple to oversee.
4. **Low Resource Requirement:** Can be executed with minimal resources, making it cost-effective for small projects.
5. **Flexibility for Developers:** Developers have the freedom to make changes and experiment without rigid constraints.
6. **Ideal for Learning:** Serves as a good learning tool for beginners or students practicing software development.

Disadvantages of the Big Bang Model

1. **High Risk and Uncertainty:** The lack of planning increases the chances of project failure.
2. **Not Suitable for Complex Systems:** Ineffective for large-scale, object-oriented, or structured projects.
3. **Unsuitable for Long-Term Projects:** Poor choice for projects that require ongoing development and maintenance.
4. **Potentially Expensive:** Misunderstood or evolving requirements can lead to excessive costs due to rework and inefficiencies.

5.0 Requirement Determination

Requirement determination is a crucial step in system development, as it defines what a system must do to meet user and business needs. Without proper requirements gathering, a system may fail to function as expected, leading to inefficiencies, financial losses, or even total system failure. This process involves engaging with stakeholders to gather, analyze, and refine their expectations. The goal is to understand their needs, identify existing challenges, and propose solutions that align with the objectives of the organization.

For example, if a university wants to develop a Student Information System, the system analyst must determine how students will register for courses, how lecturers will upload results, and how administrators will generate reports. Similarly, if a bank plans to introduce an Online Loan Application System, it must define how customers will apply for loans, how eligibility will be checked, and how approvals will be processed.

5.1 Types of Requirements

Requirements are broadly classified into two categories: functional requirements and non-functional requirements.

5.1.1 Functional Requirements

Functional requirements define the essential features, operations, and tasks that a system must perform to meet user needs. These requirements specify what the system should do and how it should respond to different inputs, ensuring it functions as intended. They outline the core functionalities that enable users to interact with the system effectively, making it an integral part of software and system development. Whether it's processing transactions, managing records, or facilitating communication, functional requirements set clear expectations for system behavior and performance. For instance, in a university system, a Course Registration System should allow students to register for courses, pay fees, and receive confirmation via email or SMS, while a Student Result Management System should enable lecturers to upload grades, compute GPAs, and generate transcripts. In banking, an Automated Teller Machine (ATM) must allow customers to check balances, withdraw cash, and print receipts, whereas a Mobile Banking App should support money transfers, bill payments, and account statement generation. Clearly defining these requirements ensures the system delivers expected functionalities, improving user experience and operational efficiency.

5.1.2 Non-Functional Requirements

Non-functional requirements define the quality attributes of a system, focusing on how well it performs rather than what it does. These requirements encompass aspects like speed, security, scalability, reliability, and usability, all of which

contribute to the system's overall efficiency and user experience. Unlike functional requirements that specify tasks the system must perform, non-functional requirements ensure that these tasks are executed smoothly, securely, and reliably. For example, in a university system, performance requirements dictate that a student registration system should process course enrollments in under 10 seconds, even during peak periods, while security measures ensure that only authorized lecturers can modify students grades. In banking, scalability is critical for a mobile banking app to support at least 100,000 concurrent users without crashing, and ATMs must maintain high availability, operating 24/7 except during maintenance. These non-functional aspects are crucial for optimizing system performance, ensuring security, and delivering a seamless user experience.

5.2 Importance of Requirement Determination

Proper requirement determination helps in:

i. **Ensuring the system meets user expectations**: Clearly defining requirements ensures that students, bank customers, government agencies, and hospital staff get a system that works for them.

ii. **Reducing development costs**: Identifying requirements early prevents costly redesigns or total system failure.

iii. **Improving system usability**: A well-defined system is easier to use, reducing the need for excessive training.

iv. **Enhancing efficiency**: By automating processes, banks can process loans faster, universities can manage student records better, and hospitals can improve patient care.

v. **Supporting decision-making**: With well-organized data, university administrators, bank managers, and government agencies can make better decisions.

5.3 Techniques for Gathering Requirements

To develop a successful system, system analysts must gather accurate and complete requirements from users and stakeholders. Various techniques can be used to ensure that all necessary information is collected efficiently. The choice of technique depends on factors such as the nature of the project, the availability of users, and the complexity of the system being developed. Below are the most commonly used techniques for gathering requirements:

5.3.1 Interviews

Interviews are one of the most effective methods for gathering requirements because they allow system analysts to engage directly with users and stakeholders. This technique involves asking structured or unstructured questions to understand their needs, expectations, and concerns about the system.

5.3.1.1 Types of Interviews:

a. **Structured Interviews:** The analyst prepares a fixed set of questions that all interviewees must answer. This ensures consistency and makes it easier to compare responses.

b. **Unstructured Interviews:** The analyst has a general idea of what to ask but allows the conversation to flow naturally, encouraging users to provide more detailed feedback.

c. **Semi-Structured Interviews:** A mix of structured and unstructured questions, allowing flexibility while maintaining focus on key topics.

Advantages of Interviews

i. Provides in-depth information about user needs.

ii. Allows direct clarification of misunderstandings.

iii. Helps build good relationships with stakeholders.

Disadvantages of Interviews

i. Time consuming, especially in large organizations.

ii. Requires skilled interviewers who can ask the right questions.

iii. Some users may not fully express their needs due to fear or uncertainty.

5.3.2 Questionnaires

A questionnaire is a structured set of written questions distributed to multiple users to collect information about their needs and expectations. This method is particularly useful when dealing with a large number of users or when responses need to be quantified.

*5.3.2.1*Types of Questions:

a. **Closed-Ended Questions:** Provide predefined answer choices (e.g., Yes/No, multiple-choice). Useful for gathering measurable data.

b. **Open-Ended Questions:** Allow users to provide detailed responses in their own words, giving deeper insights into their needs.

Advantages of Questionnaires

i. Can reach a large number of users quickly.

ii. Responses can be easily analyzed and compared.

iii. Users can complete them at their convenience.

Disadvantages of Questionnaires:

i. Users may ignore or misinterpret questions.

ii. Responses may lack depth compared to interviews.

iii. Difficult to follow up on unclear answers.

5.3.3 Observation

Observation involves watching how users interact with an existing system or carry out their daily tasks. This technique helps analysts understand real-life challenges and identify requirements that users might not mention in interviews or questionnaires.

5.3.3.1 Types of Observation:

a. **Passive Observation:** The analyst silently watches the user without interfering.

b. **Active Observation:** The analyst interacts with the user, asking questions while observing.

Advantages of Observation

i. Helps identify actual challenges users face, rather than relying on what they say.

ii. Uncovers inefficiencies and workarounds users may have developed.

iii. Provides real-time feedback on system performance.

Disadvantages of Observation

i. Users may alter their behavior when they know they are being observed.

ii. Some tasks are complex and require additional clarification from users.

iii. Time consuming, especially for large scale systems.

5.3.4 Document Analysis

Document analysis involves reviewing existing records, reports, policy documents, system manuals, or historical data related to the system being developed. This technique helps analysts understand how current processes work and what improvements are needed.

Advantages of Document Analysis:

i. Provides a historical perspective on how the system has evolved.

ii. Helps identify gaps in existing processes.

iii. Useful for regulatory and compliance related requirements.

Disadvantages of Document Analysis:

i. Documents may be outdated or incomplete.

ii. May require additional clarification from users.

iii. Can be time consuming if there is a large volume of documents.

5.3.5 Joint Application Development (JAD)

JAD is a collaborative approach where system analysts, developers, and key stakeholders come together in structured workshops to discuss and define system requirements. These sessions help ensure that everyone's input is considered before development begins.

Steps in a JAD Session

a. **Preparation:** Identify key participants and define objectives.

b. **Workshop Sessions:** Stakeholders discuss system requirements, processes, and potential improvements.

c. **Documentation:** The analyst records agreed upon requirements for approval.

d. **Review & Refinement:** Participants verify the documented requirements and suggest modifications if needed.

Advantages of JAD:

i. Encourages direct collaboration between users and developers.

ii. Reduces misunderstandings and speeds up decision making.

iii. Ensures that all stakeholders' views are represented.

Disadvantages of JAD:

i. Requires commitment from all participants.

ii. Scheduling can be difficult, especially in large organizations.

iii. Sessions may become unproductive if not properly managed.