MANGALAYATAN UNIVERSITY
Learn Today to Lead Tomorrow
NAAC GRADE A+ Accredited University

BACHELOR OF COMPUTER APPLICATION (BCA)

CSB-1111 (Management Information System)

Block 1: Introduction to Management Information Systems

Unit 4: Software Development Life Cycle (SDLC)

Structure	
4.0	Introduction
4.1	Objectives
4.2	Software Development Life Cycle
4.3	SDLC Models
4.3.1	Waterfall Model
4.3.2	RAD (Rapid Application Development) Model
4.3.3	Spiral Model
4.3.4	Incremental Model
4.3.5	Agile Model
4.3.6	Iterative Model
4.4	Summary
4.5	Questions and Answers
4.6	References

4.0 Introduction

In the rapidly evolving field of software engineering, the systematic approach to developing high-quality software is paramount. The Software Development Life Cycle (SDLC) serves as a comprehensive framework guiding the software development process from initial concept through deployment and maintenance. This structured approach ensures that software projects are well-planned, consistently executed, and rigorously tested, ultimately delivering solutions that meet user requirements and quality standards.

SDLC is a multi-phase process that includes planning, analysis, design, implementation, testing, deployment, and maintenance. Each phase has distinct objectives and deliverables that contribute to the overall success of the project. The importance of a well-defined SDLC lies in its ability to provide a clear roadmap for the development team, reduce the risk of project failures, and ensure efficient use of resources. By adhering to a systematic process, organizations can address potential issues early, manage project timelines effectively, and deliver robust software products.

Various SDLC models have been developed to address different project needs and organizational contexts. These models, such as the Waterfall model, Agile model, and Spiral model, offer diverse methodologies to navigate the complexities of software development. Understanding the strengths and limitations of each model allows development teams to choose the most appropriate approach based on project scope,

requirements, and risk factors. This chapter explores the fundamental concepts of SDLC, delves into the characteristics of different SDLC models, and highlights their practical applications in real-world software development projects.

4.1 Objectives

After completion of this unit, you will be able to understand,

- Gain knowledge of each phase in the Software Development Life Cycle and their roles in developing quality software.
- Identify and compare various SDLC models, including Waterfall, RAD, Spiral, Incremental, Agile, and Iterative.
- Learn to select and implement suitable SDLC models for different project requirements and contexts.
- Assess how SDLC models affect project success in terms of timelines, costs, and software quality.
- Explore how traditional SDLC integrates with modern methodologies like Agile and DevOps.

4.2 Software Development Life Cycle

The software industry uses the Software Development Life Cycle (SDLC) method to design, develop, and test high-quality software products. The goal of the SDLC is to create high-quality software that meets or beyond customer expectations and is completed on schedule and within budget.

A process model, often known as a software life cycle model, is a diagrammatic and graphical depiction of the software life cycle. Every technique needed to move a software product through its life cycle stages is represented by a life cycle model. It also encapsulates the framework within which these techniques should be used.

Stated differently, a life cycle model delineates the range of tasks executed on a software product from its conception to its eventual retirement. The required development tasks may be allocated to phases in a variety of ways by different life cycle models. Therefore, regardless of the life cycle model that is used, all important activities are included, even if they may be carried out in different sequences depending on the life cycle model. It is possible to engage in multiple activities at any point in the life cycle.

The term "Software Development Life Cycle" (SDLC) describes a process-oriented methodology for producing high-quality software. The SDLC technique, in detail, concentrates on the following stages of software development:

- Requirement analysis
- Planning
- Software design such as architectural design
- Software development
- Testing
- Deployment

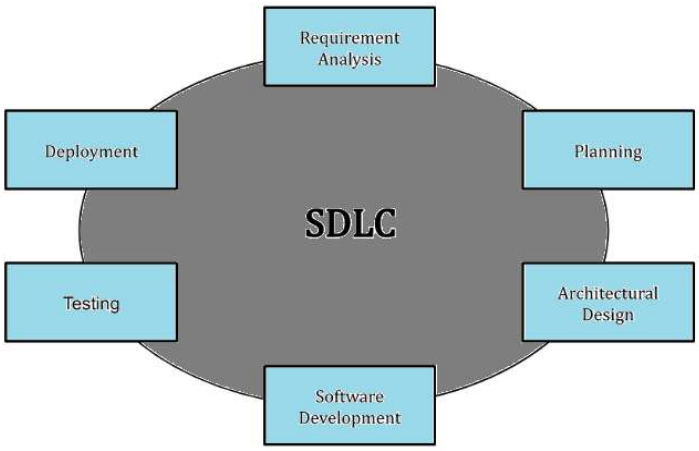
How the SDLC Works

The Software Development Life Cycle (SDLC) reduces software development costs while concurrently enhancing quality and speeding up output. By adhering to a strategy that eliminates the common dangers of software development projects, the SDLC accomplishes these seemingly incompatible objectives. The first step in such plan is to assess current systems for flaws.

It then goes on to specify the needs for the new system. The stages of analysis, planning, design, development, testing, and deployment are then used to produce the software. Anticipating costly errors such as neglecting to request input from the end-user or client might help SLDC avoid unnecessary rework and post-mortem repairs.

It's also critical to understand that the testing phase is given a lot of attention. Because the SDLC is a recurring approach, each cycle of the process requires code quality assurance. Many organizations have a tendency to put little effort into testing, despite the fact that doing so might save them a great deal of money, time, and rework. Write the appropriate tests with intelligence.

Let's now examine each of the Software Development Life Cycle's phases.



Phases and Optimal Methods:

1. **Identify the Current Problems:** "What issues are there right now?" Input from all stakeholders, including clients, salespeople, industry experts, and programmers, is required at this stage of the SDLC. Discover the advantages and disadvantages of the current system with the aim of making improvements.
2. **Plan:** "What are our goals?" The team ascertains the cost and resources needed to implement the requirements once they have been analyzed at this level of the SDLC. Additionally, it outlines the dangers and offers backup measures to lessen them. Stated differently, the team needs to assess the project's viability and figure out how to carry it out effectively while minimizing risk.
3. **Design:** "How can we make our vision a reality?" The software specifications are first transformed into a design plan known as the Design Specification during this stage of the SDLC. After reviewing this proposal, all stakeholders provide comments and recommendations. Having a strategy in place for gathering and adding stakeholder feedback to this document is essential. Failing now is nearly guaranteed to lead to cost overruns at best and the project's complete collapse at worst.
4. **Build "Let's Make What We Desire":** This is when the real development begins. It's critical that each developer follows the approved blueprint. Additionally, confirm that the appropriate policies about coding style and procedures are in place. Establish a file nomenclature or a variable name convention, such camelCase, as examples. This will assist your team in writing more logical, consistent code that will be simpler to test in the following stage and easier to understand.
5. **Code Test: "Did We Get What We Wanted?"** We check for flaws and shortcomings at this point. Until the product satisfies the original specifications, we address those problems. In other words, we

want to confirm that the code satisfies the specified requirements. Prefix is compatible with Java, PHP, Node.js, Ruby, Python, and.NET.

6. **Software Deployment:** "Let's get to work using what we have." Deploying the program to the production environment at this point will allow users to begin utilizing the product. Still, a lot of companies decide to run the product across many deployment environments, such a staging or testing environment.
   This enables everyone involved in the product to test it out safely before it goes on sale. Additionally, this makes it possible to catch any last errors before the product is released.

Software Maintenance: When the client begins utilizing the systems that were designed, the true problems arise and demands that need to be met periodically. Maintenance refers to the process wherein the developed product is cared for.

4.3 SDLC Models

The Software Development Life Cycle (SDLC) is a project management methodology that outlines the phases involved in developing an information system, starting with a feasibility assessment and ending with application maintenance.

During the software development phase, various software development life cycle models are specified and designed. Another name for these models is "Software Development Process Models." To guarantee success at each stage of the software development process, each process model adheres to a set of phases specific to its kind.

4.3.1 Waterfall Model

In 1970, Winston Royce presented the Waterfall Model. Requirements analysis and specification, design, implementation, unit testing, integration and system testing, and operation and maintenance are the five phases of this methodology. The steps never skip a beat and are always completed in this sequence. Every stage must be finished by the developer before moving on to the next. The reason this model is called the "Waterfall Model" is that a waterfall appears in its diagrammatic portrayal.

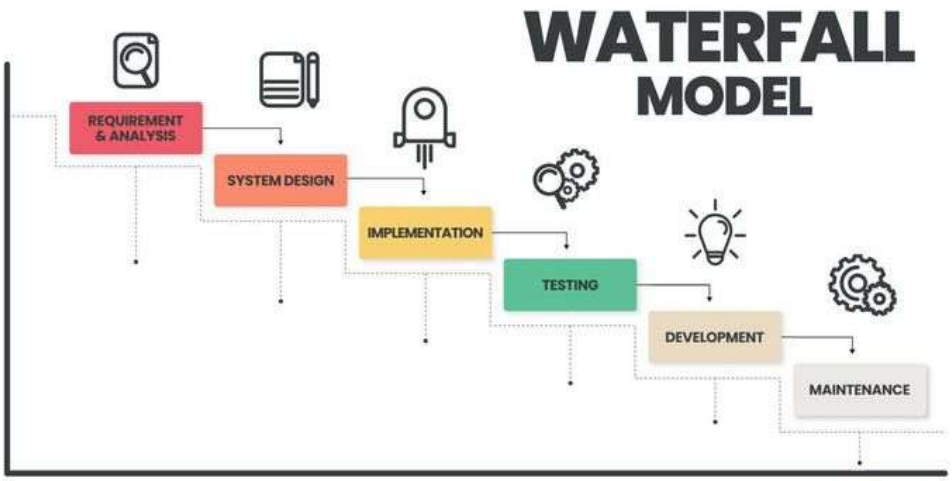


Image Source - Pinterest

1. **Requirements Analysis and Specification Phase:** This stage's goal is to accurately document the customer's precise needs by fully comprehending them. Together, the client and the software

developer document all of the features, functionality, and interface requirements of the program. The "what" of the system to be developed is described, not the "how. "During this stage, a sizable document known as the Software Requirement Specification (SRS) document is produced. It contains a comprehensive, common language description of the functions of the system.

2. **Design Phase:** The goal of this phase is to convert the requirements specified in the SRS into a format that can be further coded in a programming language. Along with high level and detailed design, it defines the entire software architecture. A Software Design Document (SDD) serves as the documentation for all of this effort.
3. **Implementation and Unit Testing**: In this stage, the design is put into practice. When the SDD is finished, software developers can easily move forward with the implementation or coding phase since it contains all the information they require. The code is carefully reviewed and adjusted as it is being tested. Initial testing is done on small modules alone. Subsequently, the interoperability of these modules and the flow of intermediate output are verified by implementing some overhead code.
4. **Integration and System Testing**: The success of the testing process determines the final product's quality; hence this stage is very important. Improved productivity will result in happier clients, cheaper upkeep, and precise findings. Unit testing establishes each module's level of efficiency. On the other hand, the modules' interactions with the system and with each other are tested during this phase.
5. **Operation and Maintenance Phase**: After the program has been installed, supplied to the customer, and made operational, maintenance is the work that each user must complete.

Use of Waterfall Model

Some Circumstances where the use of the Waterfall model is most suited are:

- When the requirements are constant and not changed regularly.
- A project is short
- The situation is calm
- Where the tools and technology used is consistent and is not changing
- When resources are well prepared and are available to use.

Advantages of Waterfall Model

- This model requires little in the way of resources and is easy to put into practice.
- The requirements are clear and uncomplicated, and they won't alter during the course of the project.
- It is simple to cover progress because each phase's beginning and ending are set.
- Before development, the final cost and release date of the entire product can be decided.
- Because of the stringent reporting system, it provides easy control and clarity for the customer.

Disadvantages of Waterfall Model

- Because of the higher risk component in this paradigm, it is not appropriate for larger, more intricate projects.
- During development, requirements modifications are not compatible with this paradigm.
- Returning to the phase gets harder. For instance, it becomes difficult to go back and modify the requirements if the program has now moved to the development phase and there is a change.

- It is challenging to design a risk reduction strategy because testing is done at a later stage, making it impossible to detect the risks and obstacles in the earlier phase.

4.3.2 RAD (Rapid Application Development) Model

The Rapid Application Development (RAD) process model is a sequential, linear software development cycle that prioritizes lean building with an element-based approach. A development team can produce a fully functional system in a short amount of time with the RAD approach if the requirements are properly understood and documented and the project scope is constrained.

According to the theory of RAD (Rapid Application Development), goods can be created more quickly and with greater quality by:

- Gathering requirements using workshops or focus groups
- Prototyping and early, reiterative user testing of designs
- The re-use of software components
- A rigidly paced schedule that refers design improvements to the next product version
- Less formality in reviews and other team communication

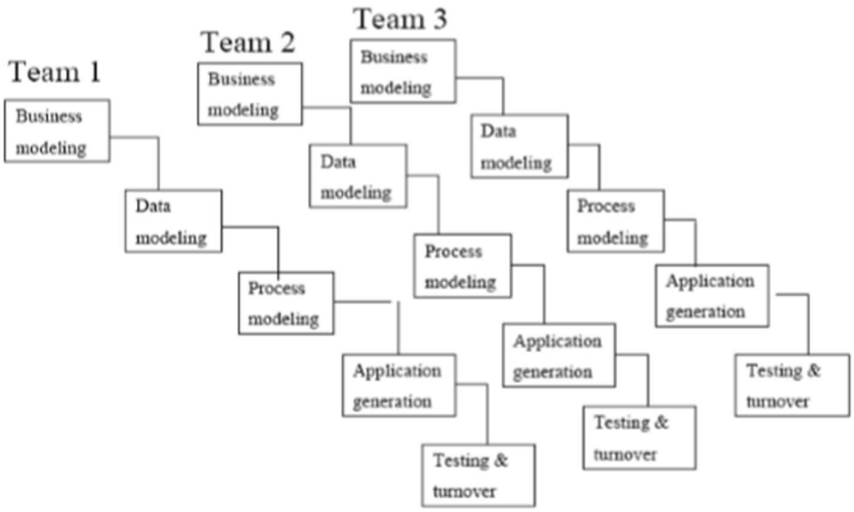


Figure: Rapid Application Development Model

Image Source: Viden.io

Phases of RAD:

1. **Business Modelling:** By providing answers to issues such as what data powers the business process, what data is generated, who generates it, where it goes, who processes it, and so on, the information flow between business units is established.
2. **Data Modelling:** A collection of data objects, or entities, that are required to support the company are derived from the data gathered from business modelling. Each entity's properties, or character, are identified, and the relationship between these data items, or entities, is established.
3. **Process Modelling:** To create the data flow required to carry out a business function, the information objects specified during the data modelling step are transformed. To add, change, remove, or retrieve a data object, processing descriptions are made.
4. **Application Generation:** To make the process of building software easier, automated methods are employed, even utilizing 4th GL approaches.

5. **Testing and Turnover:** Since RAD emphasizes reuse, a large number of the programming components have previously undergone testing. As a result, the total testing time is lowered. However, all interfaces need to be thoroughly used, and the new component needs to be tested.

Use of RAD Model:

- When the system should need to create the project that modularizes in a short span time (2-3 months).
- When the requirements are well-known.
- When the technical risk is limited.
- When there's a necessity to make a system, which modularized in 2-3 months of period.
- It should be used only if the budget allows the use of automatic code generating tools.

Advantage of RAD Model

- There is room for change with this model.
- Modifications are adoptable in this model.
- Every RAD step provides the customer with functionality that is of the utmost priority.
- It shortened the time needed for development.
- It improves a feature's capacity for reuse.

Disadvantage of RAD Model

- It needed really talented designers.
- Not every application is RAD compatible.
- We are unable to apply the RAD paradigm to smaller projects.
- Given the elevated technical risk, it is not appropriate.
- Necessary user participation.

4.3.3 Spiral Model

The spiral model, first put out by Boehm, is an evolutionary software process paradigm that combines the systematic and regulated elements of the linear sequential model with the iterative prototyping feature. It puts into practice the possibility of creating new software versions quickly. The program is built in a sequence of incremental releases according to the spiral approach. A paper model or prototype might be the supplemental release in the early iterations. Subsequent cycles yield increasingly comprehensive versions of the constructed system.

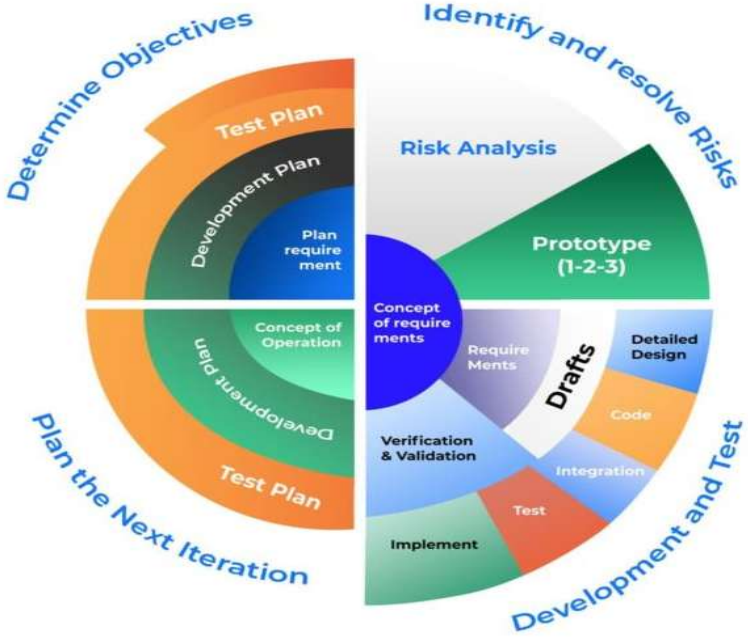


Image Source: Intellect Soft

Each cycle in the spiral is divided into four parts:

Objective setting: The purpose of each cycle in the spiral, the range of options available for reaching the goals, and the constraints that are present are all identified at the outset of the cycle.

Risk assessment and reduction: Based on the objectives and limitations, these different options are calculated in the following cycle phase. The project's perceived risk is the main area of review at this point.

Development and validation: Creating plans to address risks and uncertainties is the next stage. Activities like simulation, prototyping, and benchmarking may be a part of this process.

Planning: At last, the following action is organized. After evaluating the project, a decision is taken regarding whether to proceed with a subsequent spiral phase. Plans are created for the project's next phase if it is decided to keep.

The remaining hazards determine the development phase. The following stage might be an evolutionary development that involves creating a more thorough prototype for resolving the risks, for instance, if performance or user-interface concerns are deemed more important than program development risks.

The spiral model's risk-driven characteristic enables it to support any combination of a method that is specification-oriented, prototype-oriented, simulation-oriented, or of another kind. A key component of the approach is that all goods generated during a cycle, including plans for the following cycle, are reviewed at the end of each spiral phase. The spiral approach is applicable to both enhancement and development initiatives.

When to Use this Model

- When deliverance is required to be frequent.
- When the project is large
- When requirements are unclear and complex
- When changes may require at any time

- Large and high budget projects

Advantages

- High amount of risk analysis
- Useful for large and mission-critical projects.

Disadvantages

- Can be a costly model to use.
- Risk analysis needed highly particular expertise
- Doesn't work well for smaller projects.

4.3.4 Incremental Model

The incremental model is a software development process in which requirements are split up into several independent software development cycle modules. Every module in this paradigm goes through the phases of requirements, design, implementation, and testing. The module's functionality is increased with each new release. Until the entire system is achieved, the procedure is continued.

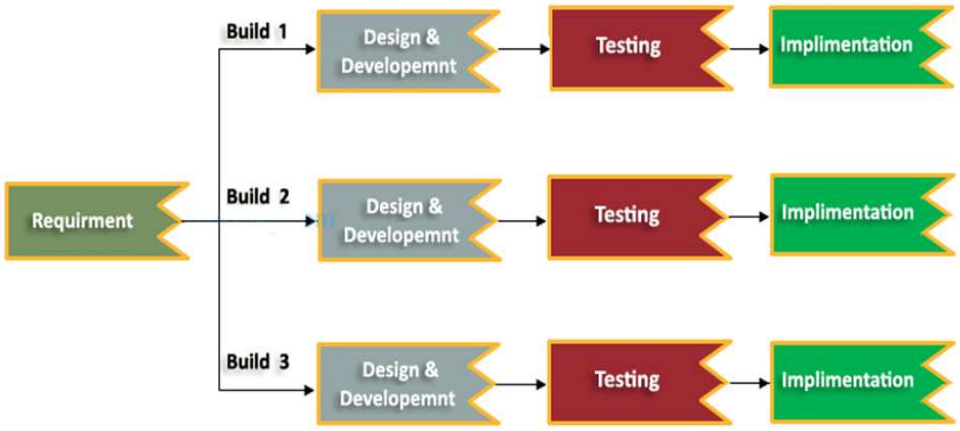


Fig: Incremental Model

Image Source: Sitesbay

Phases of Incremental Model

1. **Requirement analysis:** The product analysis expertise determines the requirements in the first stage of the incremental model. Additionally, the requirement analysis team is aware of the functional needs of the system. This stage is vital to the software development process under the incremental paradigm.
2. **Design & Development:** The development process and the design of the system's functionality are successfully completed in this stage of the incremental SDLC model. The incremental model makes use of style and development phase when software gains new practicality.
3. **Testing:** In the incremental paradigm, testing verifies that all current functions operate as intended as well as any new features. The behavior of every task is tested throughout the testing phase using a variety of techniques.
4. **Implementation:** The development system's coding phase is made possible by the implementation phase. In the designing and developing phase, it involves the final code, and in the testing phase, it involves testing the functionality. The number of products that are operational after this phase is finished is improved and upgraded up to the final system product.

When to Prefer Incremental Model

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Advantage of Incremental Model

- Errors are simple to identify.
- Flexible and easier to test and debug.
- Because risk was handled during its iteration, it was easy to control.
- Important functionality is delivered to the client early.

Disadvantage of Incremental Model

- The necessity of careful planning
- The entire cost is substantial.
- Module interfaces must be clearly defined.

4.3.5 Agile Model

Agile is defined as quick or adaptable. The term "agile process model" describes an iteratively based approach to software development. Agile techniques divide work into manageable chunks, or portions that don't directly require long-term planning. At the start of the development process, the requirements and scope of the project are established. Plans are outlined in advance with reference to the quantity, duration, and scope of each iteration.

In the Agile process model, an iteration is a small time "frame" that usually lasts one to four weeks. By breaking the project up into smaller components, the overall project delivery time requirements are decreased and project risk is minimized. Before a working product is shown to the client, a team goes through the entire software development life cycle during each iteration, including planning, requirements analysis, design, coding, and testing.

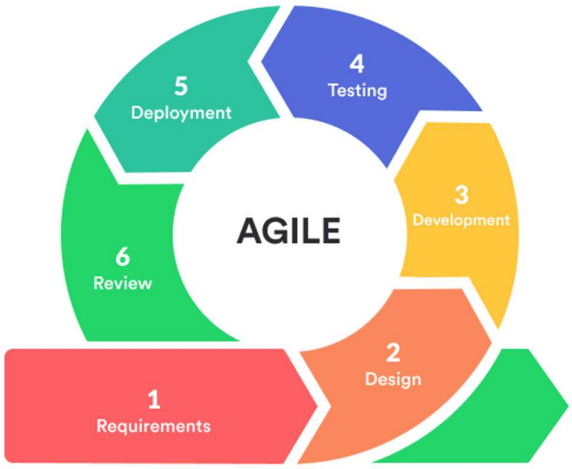


Image Source: Re10 School

Phases of Agile Model

1. **Requirements Gathering:** This stage involves defining the requirements. Along with outlining the time and effort required to complete the project, you should also discuss business potential. This data allows you to assess the viability from both a technical and financial standpoint.
2. **Design the Requirements:** After the project has been determined, collaborate with stakeholders to establish the specifications. The high-level UML diagram or the user flow diagram can be used to illustrate how new features will function and fit into your current system.
3. **Construction/Iteration:** Work starts as soon as the group decides what is needed. With the goal of releasing a functional product, designers and developers get to work on their project. The product has basic, minimal functionality and will go through several rounds of refinement.
4. **Testing:** The Quality Assurance team checks for bugs and assesses how well the product performs.
5. **Deployment:** During this stage, the team provides a product to be used in the workplace.
6. **Feedback:** Following product release, this is the final stage. In this process, the team gets input regarding the product and works through it.

When to use

- When frequent changes are required.
- When a highly qualified and experienced team is available.
- When a customer is ready to have a meeting with a software team all the time.
- When project size is small.

Advantage of Agile Method:

- Frequent Delivery
- Face-to-Face Communication with clients.
- Efficient design and fulfils the business requirement.
- Anytime changes are acceptable.
- It reduces total development time.

Disadvantages of Agile Model:

- Due to the shortage of formal documents, it creates confusion and crucial decisions taken throughout various phases can be misinterpreted at any time by different team members.
- Due to the lack of proper documentation, once the project completes and the developers allotted to another project, maintenance of the finished project can become a difficulty.

4.3.6 Iterative Model

You can create the initial software version using this model by starting with a few of the software specifications. If changes to the software are required after the initial version, a new version is made using a fresh iteration. The term "iteration" refers to the precise, set time that each release of the iterative model ends.

The Iterative Model makes it possible to access previous stages when changes were made. at the conclusion of the Software Development Life Cycle (SDLC) process, the project's ultimate output renewed.

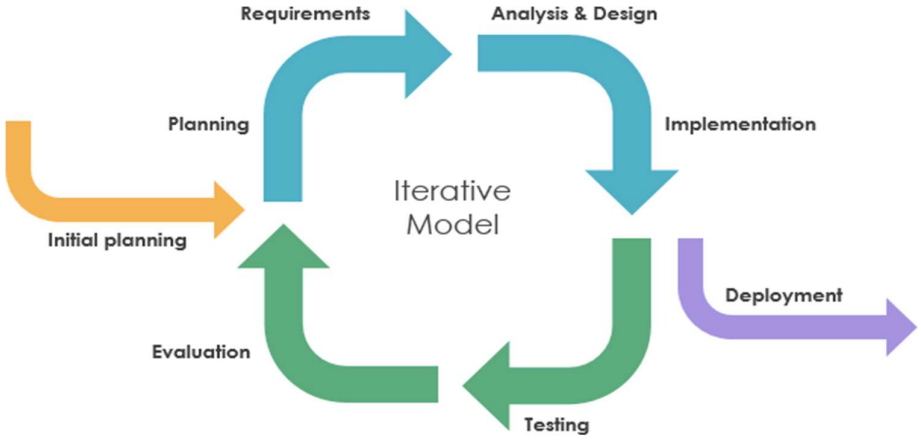


Image Source: Medium

Phases of Iterative Model

1. **Requirement gathering & analysis:** During this stage, analysts collect requirements from clients and determine whether or not they will be fulfilled. Analysts verify whether or not needs will be met within budget. The software team moves on to the next stage after all of this.
2. **Design:** Using various diagrams, such as state transition diagrams, activity diagrams, class diagrams, and data flow diagrams, the team designs the software during this phase.
3. **Implementation:** Requirements are expressed in a coding language and converted into software, or computer programs, during this phase.
4. **Testing:** Using various test techniques, software testing commences following the completion of the development process. Though there are other test methods as well, the most popular ones are the black, grey, and white box approaches.
5. **Deployment:** Software is put into its working environment once all the processes have been completed.
6. **Review:** Following product deployment, this phase is carried out to verify the behavior and applicability of the created product. And the procedure restarts at requirement gathering if any errors are discovered.
7. **Maintenance:** During the maintenance phase, defects, errors, or the need for new updates may arise after the program has been deployed in the working environment. Debugging and new additions are part of maintenance.

When to use this Model

- When requirements are defined clearly and easy to understand.
- When the software application is large.
- When there is a requirement of changes in future.

Advantage of Iterative Model

- Testing and debugging during smaller iteration is easy.
- A Parallel development can plan.

- It is easily acceptable to ever-changing needs of the project.
- Risks are identified and resolved during iteration.
- Limited time spent on documentation and extra time on designing.

Disadvantage of Iterative Model:

- It is not suitable for smaller projects.
- More Resources may be required.
- Design can be changed again and again because of imperfect requirements.
- Requirement changes can cause over budget.
- Project completion date not confirmed because of changing requirements.

4.4 Summary

In this chapter, we explored the Software Development Life Cycle (SDLC) and its essential role in the systematic development of software. The SDLC is a structured process that guides the development team through various phases, including planning, analysis, design, implementation, testing, deployment, and maintenance. Each phase has specific objectives and deliverables that ensure the creation of high-quality software.

We examined multiple SDLC models, such as the Waterfall model, Rapid Application Development (RAD) model, Spiral model, Incremental model, Agile model, and Iterative model. Each model offers distinct methodologies to address different project needs and organizational contexts. Understanding the strengths and limitations of these models helps in selecting the most appropriate approach for various software projects.

The chapter highlighted the importance of choosing the right SDLC model to influence project outcomes positively. It emphasized how the integration of traditional SDLC models with modern practices like Agile and DevOps can enhance iterative development, rapid prototyping, and customer feedback integration. Through this comprehensive overview, we provided a solid foundation for effectively managing and participating in software development projects using structured and systematic approaches.

4.5 Questions and Answers

1. **What are the main phases of the Software Development Life Cycle (SDLC)?**

   Answer: The main phases of the SDLC are:

   - **Planning**: Define the project scope, objectives, and feasibility.
   - **Analysis**: Gather and analyze user requirements.
   - **Design**: Create architectural and detailed design specifications.
   - **Implementation**: Write and compile the actual code.
   - **Testing**: Verify that the software meets all requirements and is free of defects.
   - **Deployment**: Release the software to users.
   - **Maintenance**: Perform ongoing support and enhancements.

2. **What is the Waterfall model, and what are its advantages and disadvantages?**

Answer: The Waterfall model is a linear sequential SDLC model where each phase must be completed before the next phase begins.

Advantages:

- Simple and easy to understand.
- Well-documented and structured.
- Easy to manage due to its rigidity.

Disadvantages:

- Inflexible to changes.
- Difficult to go back to any phase once completed.
- Not suitable for complex and long-term projects.

3. Explain the Agile model and its key principles.

Answer: The Agile model is an iterative and incremental SDLC approach that emphasizes flexibility, customer satisfaction, and rapid delivery.

Key Principles:

- **Customer Collaboration**: Continuous interaction with customers to gather feedback.
- **Adaptive Planning**: Adjust plans based on feedback and changing requirements.
- **Frequent Delivery**: Deliver working software frequently, typically in short iterations.
- **Cross-functional Teams**: Teams with various skills working together.
- **Continuous Improvement**: Regular reflection and adaptation to improve processes.

4. How does the Spiral model manage risks in software development?

Answer: The Spiral model combines iterative development with systematic risk management. It involves repeated cycles (spirals) through four phases: planning, risk analysis, engineering, and evaluation. At each spiral, risks are identified, analyzed, and mitigated through prototyping and user feedback. This approach allows for early identification and resolution of potential issues, making it suitable for complex and high-risk projects.

5. Compare and contrast the Incremental and Iterative models.

Answer:

Incremental Model:

- Develops software in small, manageable increments or pieces.
- Each increment adds functionality until the complete system is built.
- Focuses on delivering a working product early and adding features over time.

Iterative Model:

- Develops software through repeated cycles (iterations).
- Each iteration refines and improves the software based on feedback and testing.
- Allows revisiting and revising previous stages.

Comparison:

- Both models break down the development process into smaller, more manageable parts.
- The Incremental model emphasizes adding functionality in stages, while the Iterative model focuses on refining and improving the product through repeated cycles.
- Incremental development often results in partially complete products with full functionality, while iterative development improves the entire product incrementally.

6. What is the RAD (Rapid Application Development) model, and what are its primary advantages and disadvantages?

Answer: The RAD (Rapid Application Development) model is an SDLC approach that focuses on rapid prototyping and quick feedback over strict planning and testing. It emphasizes an adaptive process where prototypes are quickly developed and refined based on user feedback.

Primary Advantages:

- Speed: Allows for faster development and delivery of software.
- Flexibility: Adaptable to changes in requirements throughout the development process.
- User Involvement: Continuous user feedback ensures the final product closely aligns with user needs and expectations.
- Reduced Development Time: Iterative prototyping can quickly identify and fix issues, shortening the overall development time.

Primary Disadvantages:

- Scope Management: Projects can become chaotic if scope changes are not properly managed.
- Quality Risks: Due to the speed of development, there may be less emphasis on thorough testing and design, potentially impacting quality.
- Resource Intensive: Requires a team with diverse skills and significant user involvement, which can be resource-intensive.
- Not Suitable for All Projects: Best suited for projects with well-understood requirements and those that can afford rapid iterations; less effective for large, complex projects with unclear initial requirements.

4.6 References

- Laudon, K. C., & Laudon, J. P. (2020). Management Information Systems: Managing the Digital Firm (16th ed.). Pearson.
- O'Brien, J. A., & Marakas, G. M. (2011). Management Information Systems (10th ed.). McGraw-Hill/Irwin.
- Stair, R., & Reynolds, G. (2017). Fundamentals of Information Systems (9th ed.). Cengage Learning.
- Turban, E., Volonino, L., & Wood, G. R. (2015). Information Technology for Management: Digital Strategies for Insight, Action, and Sustainable Performance (10th ed.). Wiley.
- Haag, S., & Cummings, M. (2012). Management Information Systems for the Information Age (9th ed.). McGraw-Hill/Irwin.

***