Semester: V	SOFTWARE ENGINEERING (Professional Elective - I) Common to CSE-AIML,CSE-DS	L	T	P	C
Code:		3	0	0	3

Course Objectives:

- Software life cycle models, Software requirements and SRS document.
- Project Planning, quality control and ensuring good quality software.
- Software Testing strategies, use of CASE tools, Implementation issues, validation &verification procedures.

Course Outcomes (COs):

At the end of the course students will be able to:

CO1	Understand software engineering evolution and apply life cycle models
CO2	Analyze and prepare SRS, project estimation & risk management techniques.
CO3	Design software using modularity, cohesion, coupling, agile methods, and UI design principles.
CO4	Implement and test software with coding standards, testing strategies, ISO 9000, CMM
CO5	Apply CASE tools, software maintenance, reuse, reverse engineering

Unit- I

Introduction: Evolution, Software development projects, Exploratory style of software developments, Emergence of software engineering, Notable changes in software development practices, Computer system engineering.

Software Life Cycle Models: Basic concepts, Waterfall model and its extensions, Rapid application development, Agile development model, Spiral model.

Unit– II

Software Project Management: Software project management complexities, Responsibilities of a software project manager, Metrics for project size estimation, Project estimation techniques, Empirical Estimation techniques, COCOMO, Halstead's software science, risk management.

Requirements Analysis and Specification: Requirements gathering and analysis, Software Requirements Specification (SRS), Formal system specification, Axiomatic specification, Algebraic specification, Executable specification and 4GL.

Unit–III

Software Design: Overview of the design process, How to characterize a good software design? Layered arrangement of modules, Cohesion and Coupling. approaches to software design.

Agility: Agility and the Cost of Change, Agile Process, Extreme Programming (XP), Other Agile Process Models, Tool Set for the Agile Process

Function-Oriented Software Design: Overview of SA/SD methodology, Structured analysis, Developing the DFD model of a system, Structured design, Detailed design, and Design Review.

User Interface Design: Characteristics of a good user interface, Basic concepts, Types of user interfaces, Fundamentals of component-based GUI development, and user interface design methodology.

Unit– IV

Coding And Testing: Coding, Code review, Software documentation, Testing, Black-box testing, White-Box testing, Debugging, Program analysis tools, Integration testing, Testing object-oriented programs, Smoke testing, and Some general issues associated with testing.

Software Reliability and Quality Management: Software reliability. Statistical testing, Software quality, Software quality management system, ISO 9000.SEI Capability maturity model. Few other important quality standards and Six Sigma.

Unit– V

Computer-Aided Software Engineering (Case): CASE and its scope, CASE environment, CASE support in the software life cycle, other characteristics of CASE tools, Towards second generation CASE Tool, and Architecture of a CASE Environment.

Software Maintenance: Characteristics of software maintenance, Software reverse engineering, Software maintenance process models and Estimation of maintenance cost. **Software Reuse:** reuse- definition, introduction, reason behind no reuse so far, Basic issues in any reuse program, A reuse approach, and Reuse at organization level.

Text Books:

1. Fundamentals of Software Engineering, RajibMall,5th Edition, PHI.
2. Software Engineering A Practitioner's Approach, Roger S. Pressman, 9thEdition, McGraw Hill International Edition.

Reference Books:

1. Software Engineering, IanSommerville,10thEdition, Pearson.
2. Software Engineering, Principles and Practices, Deepak Jain, Oxford University Press.

e-Resources:

1) https://nptel.ac.in/courses/106/105/106105182/
2) https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_01260589506387148827_shared/overview
3) https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_01338269041100 3904735_shared/overview
4) https://www.linkedin.com/learning/agile-software-development?u=285479924

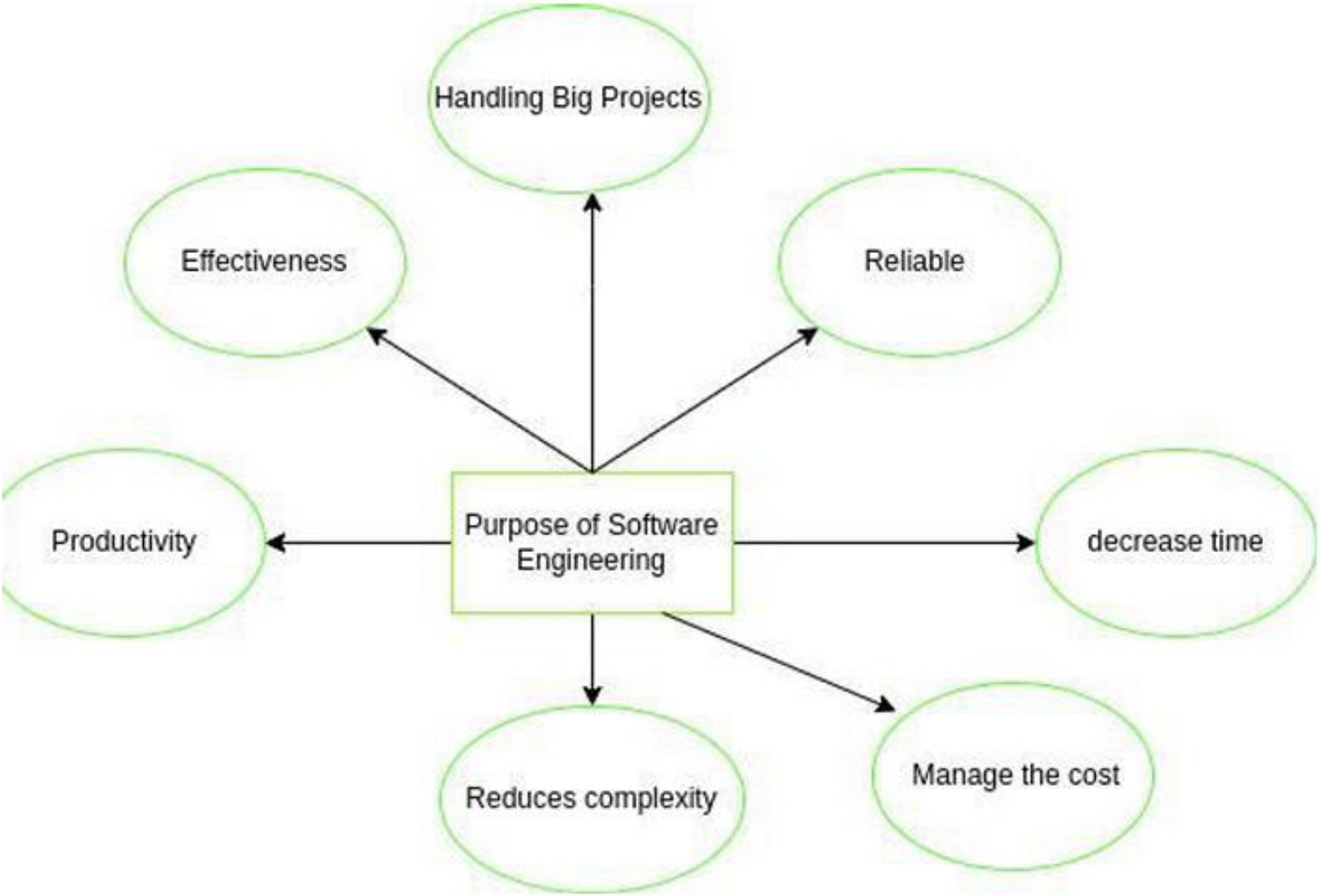
Software Engineering ?

Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.

(or)

Software engineering is a field that focuses on the design, development, testing, and maintenance of software applications. It combines principles from engineering and computer science to create software that is efficient, reliable, and meets user needs.

Purpose/Need of software Engineering:



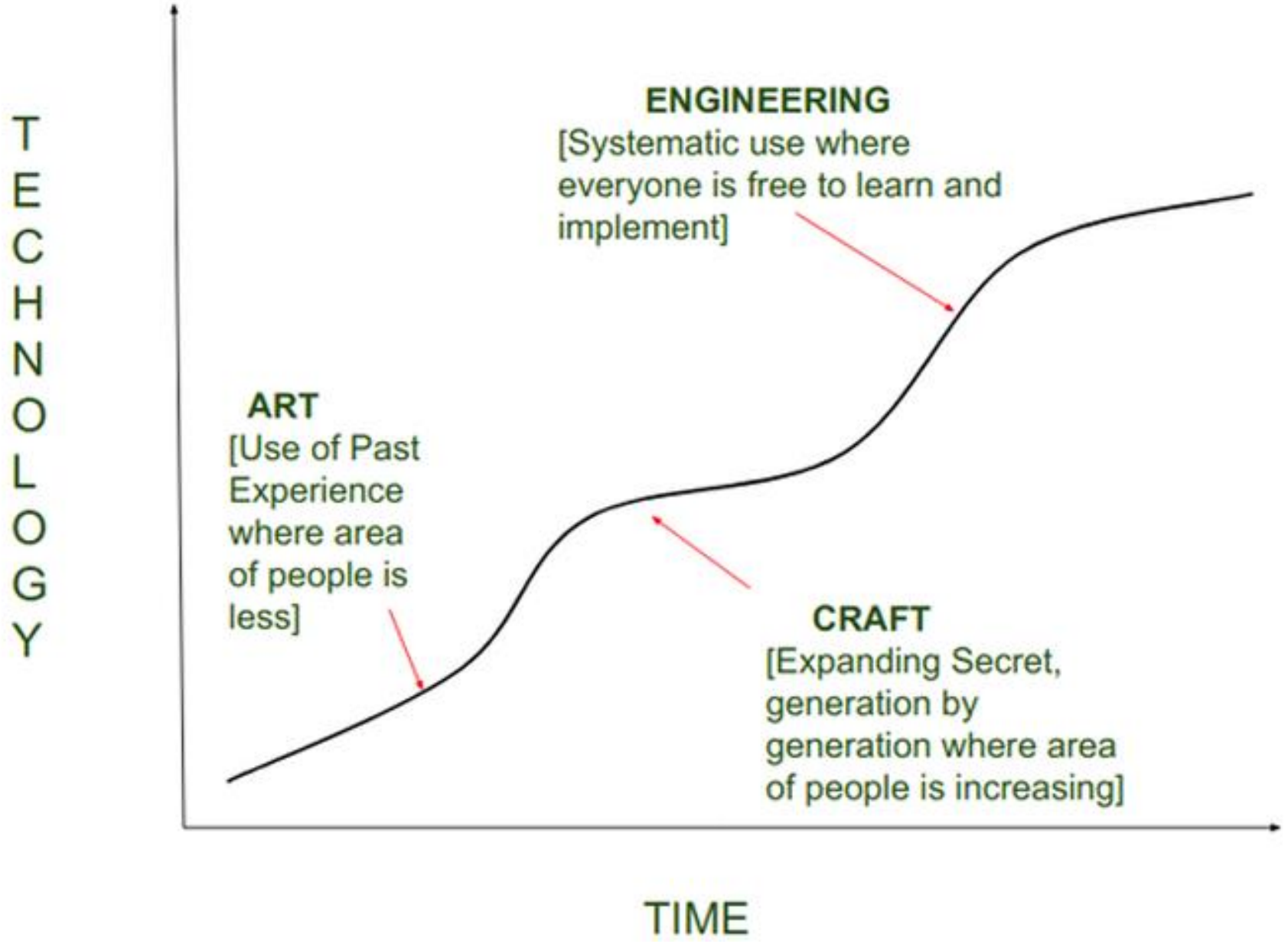
EVOLUTION- FROM AN ART FORM TO AN ENGINEERING DISCIPLINE

The detailed step-by-step process of the evolution of software engineering.

1.Software engineering as an art

2.Software engineering transition from art to craft

3.Software engineering transition from craft to an engineering discipline


TECHNOLOGY
ENGINEERING
[Systematic use where everyone is free to learn and implement]
ART
[Use of Past Experience where area of people is less]
CRAFT
[Expanding Secret, generation by generation where area of people is increasing]
TIME

Software Engineering as an Art

- In the early days of software development, many programmers considered software development as an art, as it was more about creativity and intuition than about following a set of established processes and methodologies.
- Programmers would often write code in an ad-hoc manner, without much structure or planning, and the final product was often the result of their personal artistic expression.

Software Engineering as a Craft

- Software systems have become more complex and important, it has become clear that the traditional ad-hoc approach to software development is not sufficient.
- This led to the emergence of the field of software engineering, which focused on bringing more structure and discipline to the software development process.
- The focus shifted from the artistic expression of individual programmers to the use of established processes and methodologies to ensure the quality and reliability of software systems.
- This led to the development of new methodologies and techniques such as the Waterfall and Agile methodologies, as well as the development of new tools and technologies to support software development

Software Engineering Transition from Craft to Engineering Discipline

- With the growing recognition of software engineering as an engineering discipline, the focus has shifted from the use of established processes and methodologies to ensure the quality and reliability of software systems, to a more scientific and systematic approach to software development.

- This has led to the development of new techniques and methodologies such as formal methods, model-driven development, and software architecture, which are based on sound scientific principles and are designed to ensure the quality and reliability of software systems.

The Pioneering Days (1940s-1950s)

In the early days of computing, software development was a manual and highly technical process. Computer programmers wrote machine-level instructions, dealing directly with the hardware.

Key Points:

•**Manual Coding**: In the beginning, software was crafted through manual coding, where programmers wrote machine-level instructions by hand.

•**Limited Hardware**: Hardware limitations forced developers to write efficient and compact code.

•**Use**: Software development was in its infancy, primarily used for scientific and military purposes.

Applications:

•Scientific calculations and simulations.

•Military and defense systems.

•Business data processing.

The Birth of High-Level Languages (1950s-1960s)

The introduction of high-level programming languages like Fortran, COBOL, and LISP revolutionized software development.

Key Points:

•**High-Level Languages**: The introduction of high-level programming languages like Fortran, COBOL, and BASIC made coding more accessible.

•**Compiler and Interpreter**: Compilers and interpreters translated high-level code into machine code, simplifying the coding process.

•**Use**: Business applications and database management systems gained prominence.

Applications:

•Commercial data processing.

•Early database management systems.

•Development of operating systems.

The Personal Computer Revolution (1970s-1980s)

The advent of personal computers brought software development to a broader audience. This era witnessed:

Key Points:

Personal Computers: The advent of personal computers brought software development to a broader audience.

Graphical User Interfaces (GUI): Graphical interfaces like Windows and Macintosh OS improved user experience.

Use: Expansion into home computing, gaming, and word processing.

Applications:

Word processing software (e.g., MS Word).

Early PC games (e.g., Pong and Pac-Man).

Development of GUI-based operating systems.

The Internet Age (1990s-2000s)

The World Wide Web transformed software into a global, interconnected entity. Key developments included:

Key Points:

World Wide Web: The birth of the World Wide Web transformed software into a global, interconnected entity.

Client-Server Architecture: Client-server models allowed users to interact with web applications.

Use: E-commerce, online communication, and web-based applications.

Applications:

Development of web browsers (e.g., Netscape Navigator).

E-commerce platforms (e.g., Amazon and eBay).

Email and instant messaging services.

The Rise of Mobile and Apps (2000s-Present)

The proliferation of smartphones and app stores introduced a new era of software development. Significant developments included:

Key Points:

Mobile Devices: The rise of smartphones and tablets led to a new era of software development.

App Stores: App stores, such as the Apple App Store and Google Play, centralized distribution.

Use: Mobile apps for various purposes, from social networking to navigation.

Applications:

Mobile gaming apps (e.g., Angry Birds).

Social media applications (e.g., Facebook and Instagram).

Navigation and productivity apps (e.g., Google Maps and Microsoft Office).

Cloud Computing and AI (Present and Future)

The present era is characterized by cloud computing and the integration of artificial intelligence (AI) into software development:

Key Points:

Cloud Computing: Cloud platforms offer scalable and accessible resources for software development.

Artificial Intelligence: AI and machine learning are integrated into software, enabling automation and intelligent decision-making.

Use: Cloud-based services, AI-driven applications, and IoT.

Applications:

Cloud-based storage and computing (e.g., Amazon Web Services).

AI-powered virtual assistants (e.g., Siri and Alexa).

Internet of Things (IoT) applications for smart homes and cities.

Programs *versus* Products

The software do not have any supporting documents such as users' manual, maintenance manual, design document, test documents, etc., we call these software as *programs*.

Professional software usually have multiple users and, therefore, have good user-interface, proper users' manuals, and good documentation support. Since, a software product has a large number of users, it is systematically designed, carefully implemented, and thoroughly tested.

A professionally written software usually consists not only of the program code but also of all associated documents such as requirements spécification document, design document, test document, users' manuals, etc.

Types of Software Development Projects

- A software development company typically has a large number of on-going projects.
- Each of these projects may be classified into software product development projects or services type of projects.
- These two broad classes of software projects can be further classified into subclasses

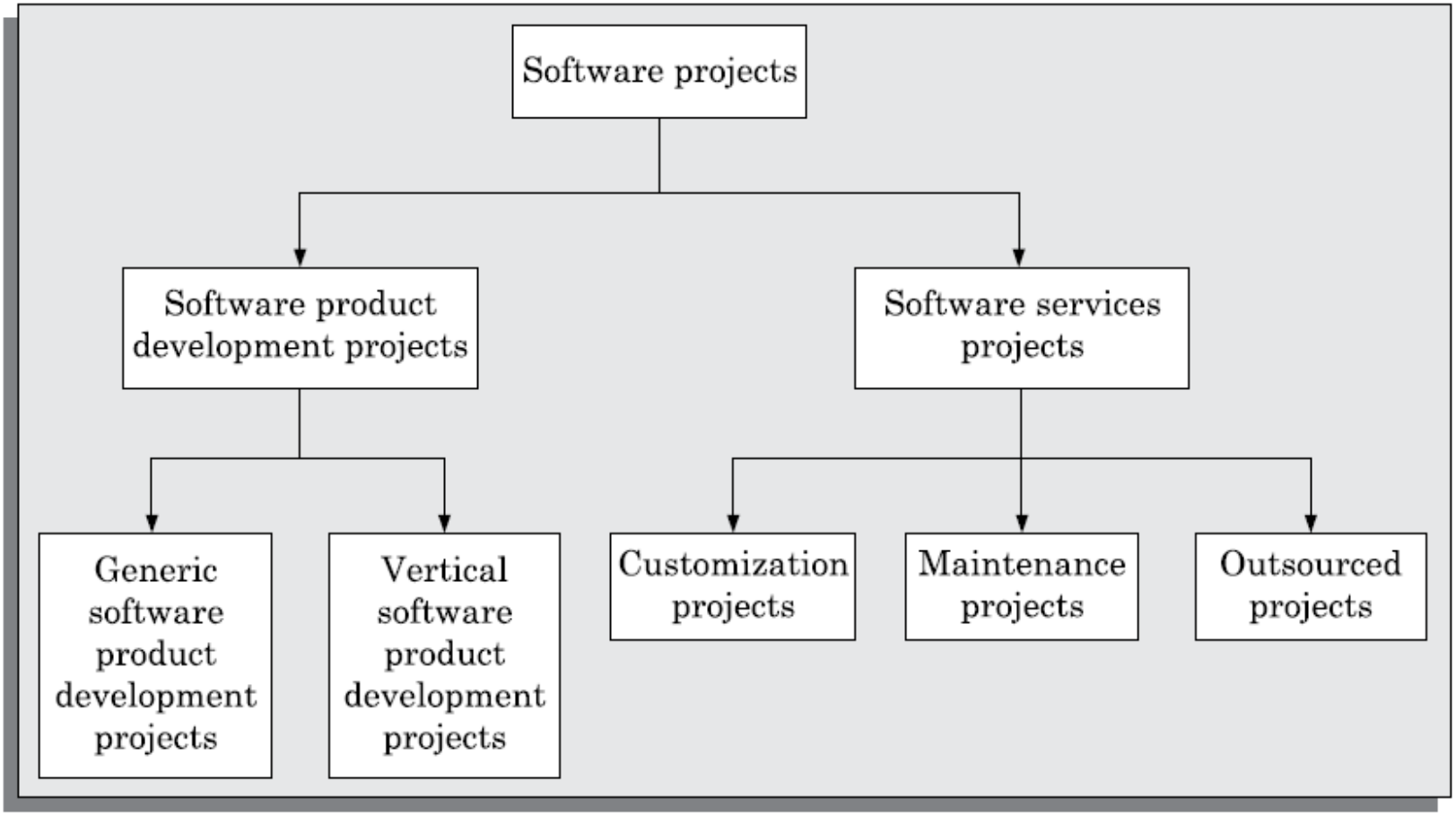


FIGURE 1.3 A classification of software projects.

EXPLORATORY STYLE OF SOFTWARE DEVELOPMENT

❖The exploratory program development style refers to an informal development style where the programmer makes use of his own intuition to develop a program rather than making use of the systematic body of knowledge categorized under the software engineering discipline.

❖The exploratory development style gives complete freedom to the programmer to choose the activities using which to develop software.

❖Though the exploratory style imposes no rules a typical development starts after an initial briefing from the customer.

❖The developers start coding to develop a working program.

❖The software is tested and the bugs found are fixed.

❖This cycle of testing and bug fixing continues till the software works satisfactorily for the customer.

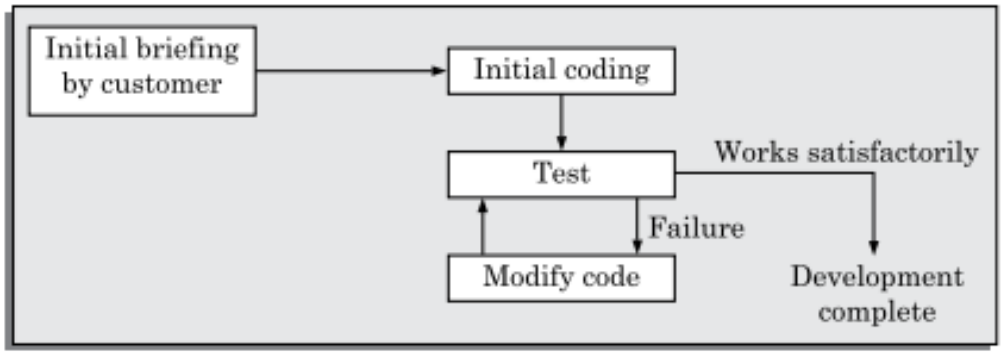


FIGURE 1.4 Exploratory program development.

What is wrong with the exploratory style of software development?

- In an exploratory development scenario, the effort and time required to develop professional software increase with an increase in program size.
- The **thick line** plots represent the case in which the exploratory style is used to develop a program. As program size increases, required effort and time increase almost exponentially.
- The **thin solid line** is used to represent a case when development is carried out using software engineering principles. In this case, it becomes possible to solve a problem with effort and time that is almost linear in program size.
- The **dotted line** is used to represent a case when development is carried out by an automated machine. In this case, an increase in effort and time with size would be even closer to a linear increase with size.

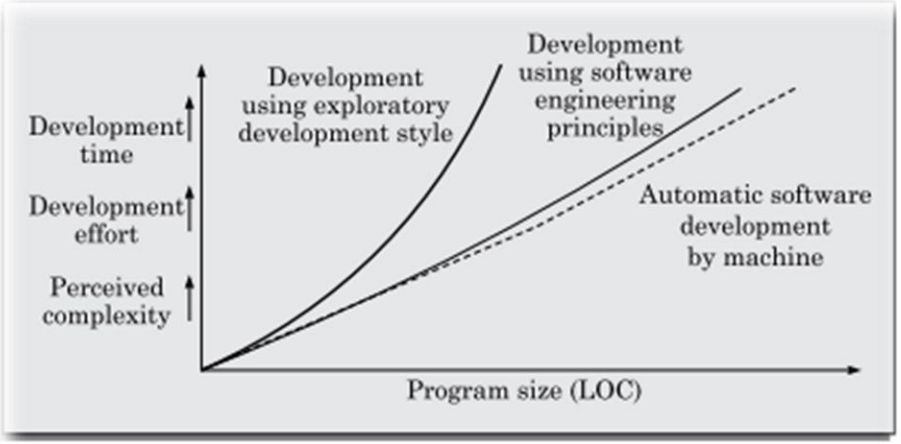


FIGURE 1.5 Increase in development time and effort with problem size.

Emergence of software engineering

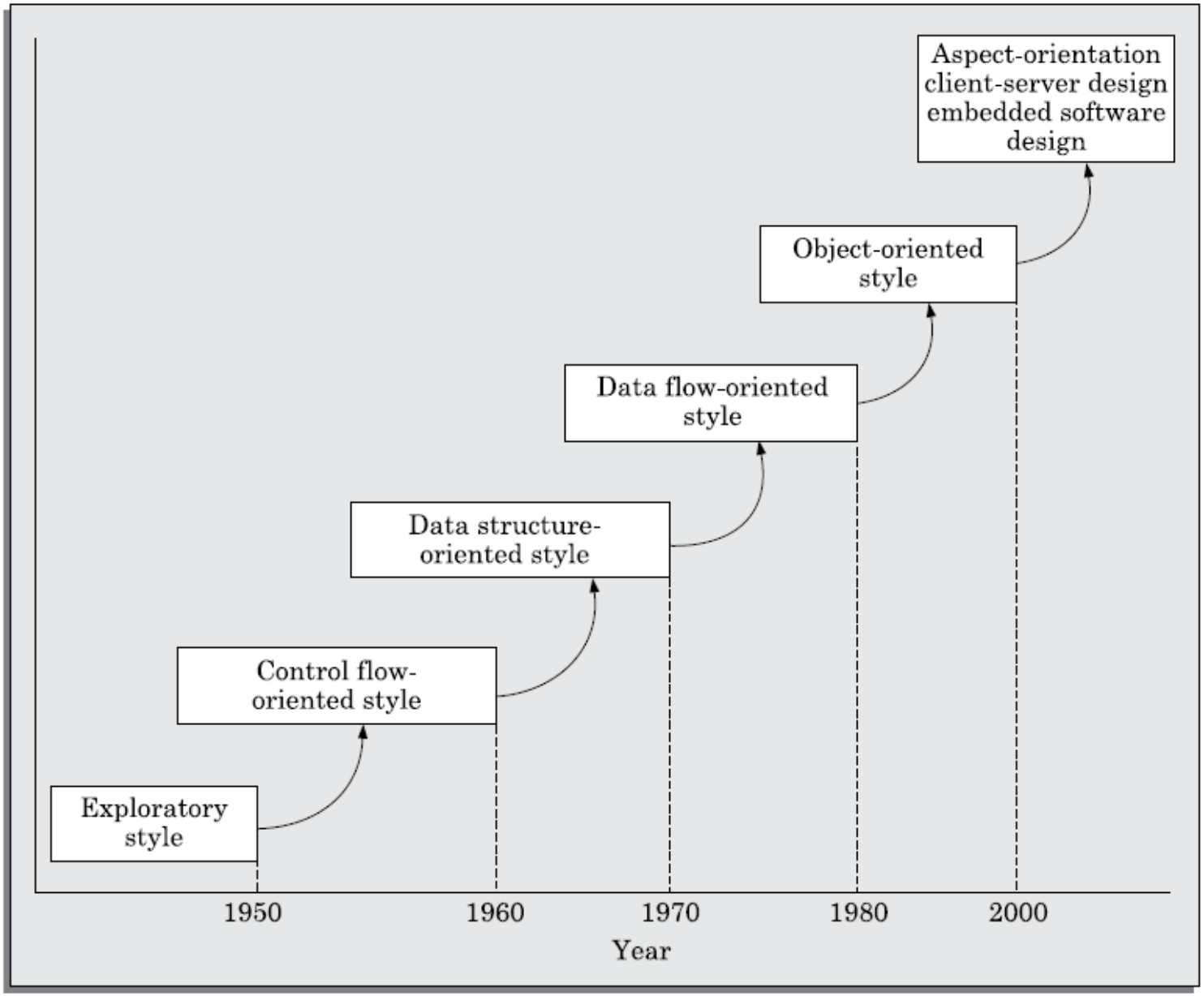


FIGURE 1.13 Evolution of software design techniques.

Control Flow-based Design

```
1       if(customer_savings_balance>withdrawal_request) {
2  100:     issue_money=TRUE;
3           GOTO 110;
        }
4       else if(privileged_customer==TRUE)
5           GOTO 100;
6       else GOTO 120;
7  110: activate_cash_dispenser(withdrawal_request);
8       GOTO 130;
9  120:     print(error);
10 130:     end–transaction();
```

(a) Unstructured program

```
1  if(privileged_customer||(customer_savings_balance>withdrawal_request)){
2          activate_cash_dispenser(withdrawal_request);
   }
3  else print(error);
4  end–transaction();
```

(b) Corresponding structured program

A program is called *structured* when it uses only the sequence, selection, and iteration types of constructs and is modular.

FIGURE 1.9 An example of (a) Unstructured program (b) Corresponding structured program.

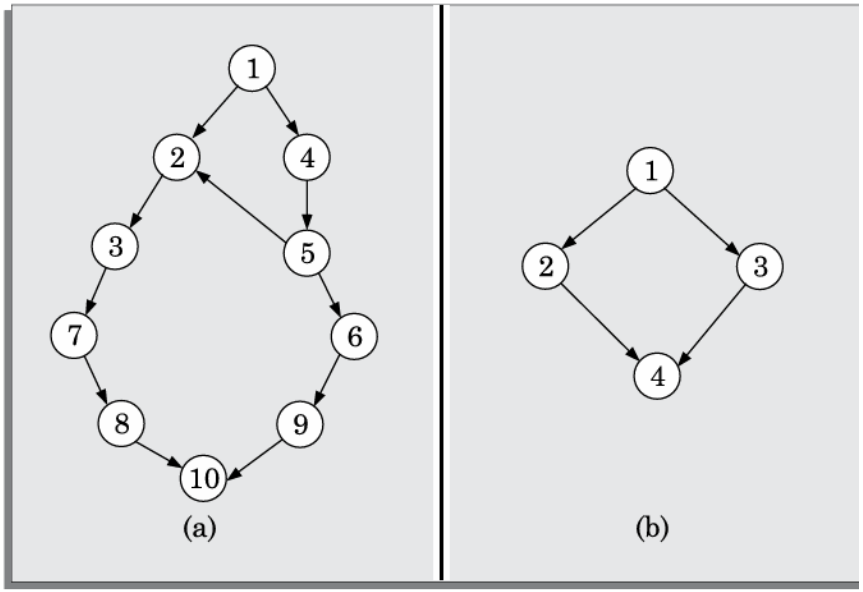


FIGURE 1.10 Control flow graphs of the programs of Figures 1.9(a) and (b).

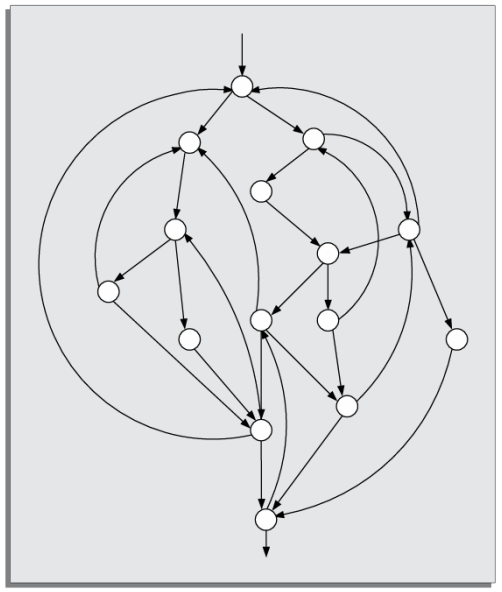


FIGURE 1.11 CFG of a program having too many GO TO statements.

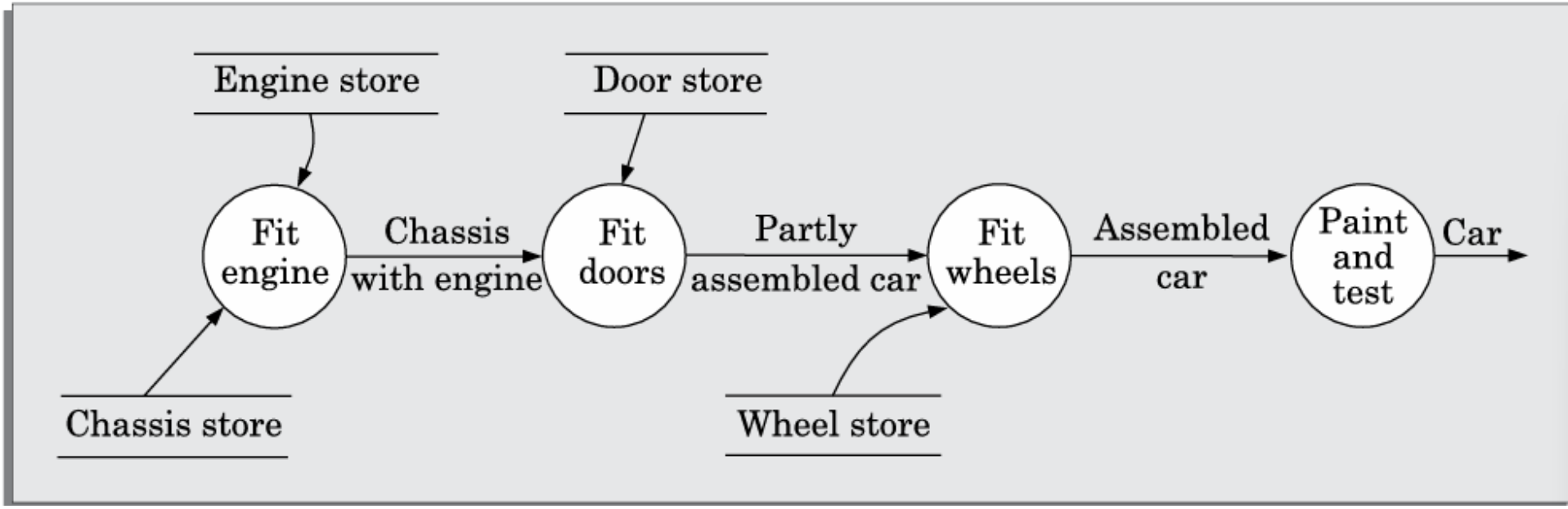
Data Structure-oriented Design

- Computers became even more powerful with the advent of integrated circuits (ICs) in the early seventies.
- These could now be used to solve more complex problems.
- Software developers were tasked to develop larger and more complicated software.
- This often required writing in excess of several tens of thousands of lines of source code.
- The control flow-based program development techniques could not be used satisfactorily any more to write those programs, and more effective program development techniques were needed.

Using data structure-oriented design techniques, first a program's data structures are designed. The code structure is designed based on the data structure.

Data Flow-oriented Design

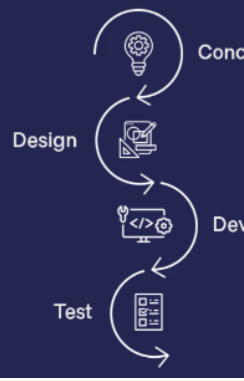
- Computers became still faster and more powerful with the introduction of very large scale integrated (VLSI) Circuits and some new architectural concepts, more complex and sophisticated software were needed to solve further challenging problems.
- Software developers looked out for more effective techniques for designing software and soon data flow-oriented techniques were proposed.
- The functions (also called as processes) and the data items that are exchanged between the different functions are represented in a diagram known as a data flow diagram (DFD).
- The program structure can be designed from the DFD representation of the problem.

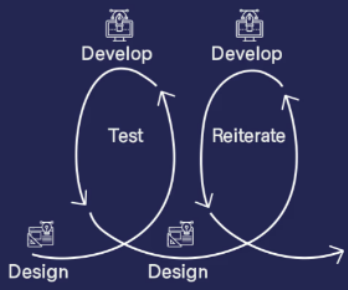


NOTABLE CHANGES IN SOFTWARE DEVELOPMENT PRACTICES

Proprietary to Open Source Software

Waterfall to Agile Methodology





On-Premise to Cloud Computing

In-house to Outsourcing

1. From Waterfall to Agile

Old Way (Waterfall): Developers used to plan everything in advance, like building a house step-by-step. Once a step was done, they couldn't go back.

New Way (Agile): Now, developers work in small steps, like building a Lego set. They keep improving the software bit by bit and can change things as they go. This makes it faster and more flexible.

2. Collaboration is Key

- **Old Way:** Developers worked alone or in small teams, and communication was limited.
- **New Way:** Developers, designers, and testers work together closely. Tools like Slack, Zoom, and GitHub help them share ideas and solve problems as a team.

3. Automation is Everywhere

- **Old Way:** Testing and deploying software was done manually, which took a lot of time.
- **New Way:** Now, robots (automated tools) do repetitive tasks like testing code or releasing updates. This saves time and reduces mistakes.

4. Open Source is Popular

- **Old Way:** Software code was kept secret, and only a few people could work on it.
- **New Way:** Many developers share their code openly (called open source). This allows everyone to learn, improve, and use the code for free. Examples: Linux, Android.

5. Cloud Computing

- **Old Way:** Software was stored on physical computers or servers.
- **New Way:** Now, software is stored in the "cloud" (online servers). This means you can access it from anywhere, like how you use Google Drive or Netflix.

6. Focus on User Experience (UX)

- **Old Way:** Software was built mainly to work, even if it wasn't easy or fun to use.
- **New Way:** Developers now focus on making software easy, enjoyable, and visually appealing for users. Think of how apps like Instagram or TikTok are designed.

7. Continuous Updates

- **Old Way:** Software was released once and rarely updated.
- **New Way:** Apps and programs are updated frequently to fix bugs, add new features, and improve performance. For example, your phone apps update automatically.

8. DevOps: Developers + Operations

- **Old Way:** Developers wrote code, and a separate team handled its deployment and maintenance.
- **New Way:** Developers and operations teams work together (called DevOps). This makes the process faster and smoother, like a well-oiled machine.

9. Mobile-First Approach

- **Old Way:** Software was designed mainly for computers.
- **New Way:** With smartphones everywhere, developers now design apps for mobile first, then adapt them for computers.

10. AI and Machine Learning

- **Old Way:** Software followed strict rules and couldn't learn or adapt.
- **New Way:** Now, software can learn from data and improve itself. For example, Netflix recommends shows based on what you watch.

COMPUTER SYSTEMS ENGINEERING

1. **Designing Systems**: Computer system engineers design the overall structure of computer systems, including how different hardware components (like processors, memory, and storage) connect and communicate with each other.
2. **Building Hardware**: They select and assemble the physical parts of computers, such as circuit boards, processors, and other components, to create functional systems.
3. **Developing Software**: Engineers also write software that controls the hardware and allows users to interact with the system. This includes operating systems and applications.
4. **Testing and Maintenance**: After building a system, engineers test it to ensure everything works correctly. They also provide ongoing maintenance to fix any issues that arise and to update the system as needed.
5. **Integration**: A crucial part of their job is making sure that all components—both hardware and software—work together seamlessly. This often involves troubleshooting problems when things don't work as expected.

COMPUTER SYSTEMS ENGINEERING

The software being developed would run on some general-purpose hardware platform such as a desktop computer or a server.

But, in several situations it may be necessary to develop special hardware on which the software would run.

Examples of such systems are numerous, and include a robot, a factory automation system, and a cell phone.

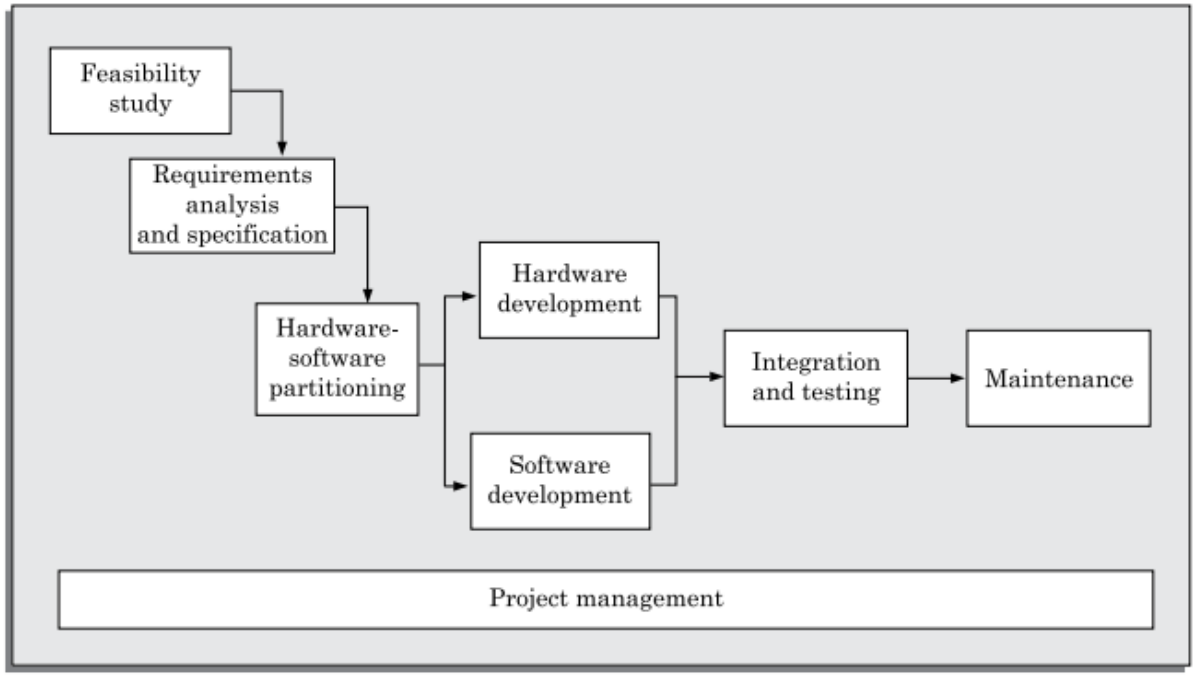


FIGURE 1.14 Computer systems engineering.

What is SDLC?

- The Software Development Life Cycle (SDLC) is a structured process followed in software engineering to design, develop, test, and maintain high-quality software.
- It provides a systematic approach to software development, ensuring that the final product meets user expectations, is cost-effective, and is delivered on time.

Need of a SDLC Model

1. Ensures a Systematic Development Process

- The SDLC model **breaks down software development** into well-defined phases, ensuring a clear workflow.
- Developers and stakeholders **know what to expect at each stage** of development.

2. Reduces Project Risks

- Helps in **early identification of risks** and ensures that potential issues are addressed before they escalate.
- **Prevents software failures** by following a structured approach.

3. Improves Software Quality

- Testing and validation are **integrated at every stage**, ensuring a bug-free, high-quality product.
- **Ensures user requirements are met** and software functions as expected.

4. Cost and Time Efficiency

- A well-defined SDLC model **reduces unnecessary rework** and ensures optimal use of resources.
- **Helps in better budgeting** by estimating costs and timelines in advance.

5. Better Project Management and Planning

- Helps teams **track progress, allocate resources efficiently, and manage time effectively**.
- Provides **clear documentation** and guidelines for developers, testers, and stakeholders.

6. Enhances Communication and Collaboration

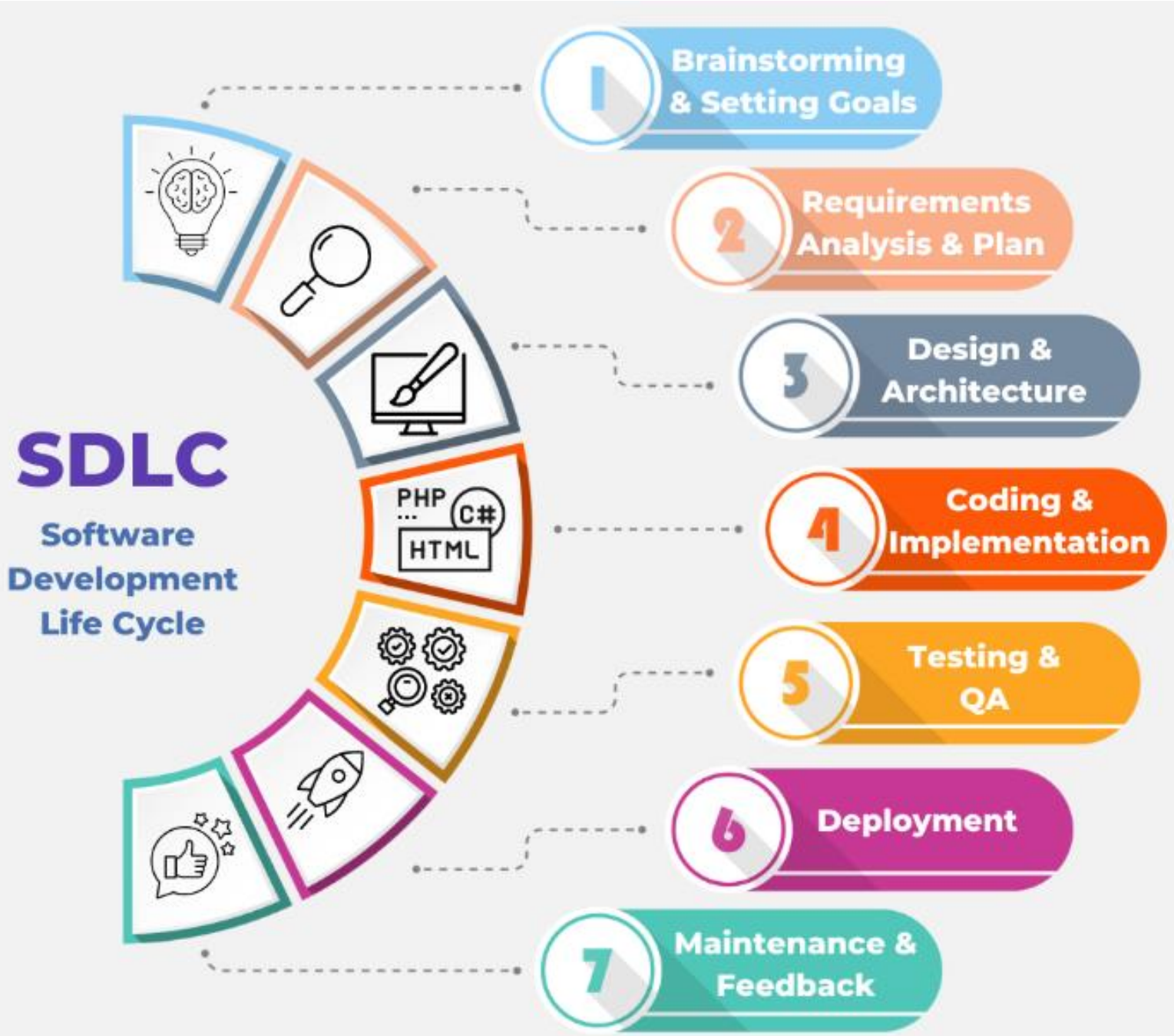
- SDLC encourages **better communication between developers, testers, project managers, and clients**.
- Ensures **transparency** throughout the development process.

7. Facilitates Maintenance and Upgrades

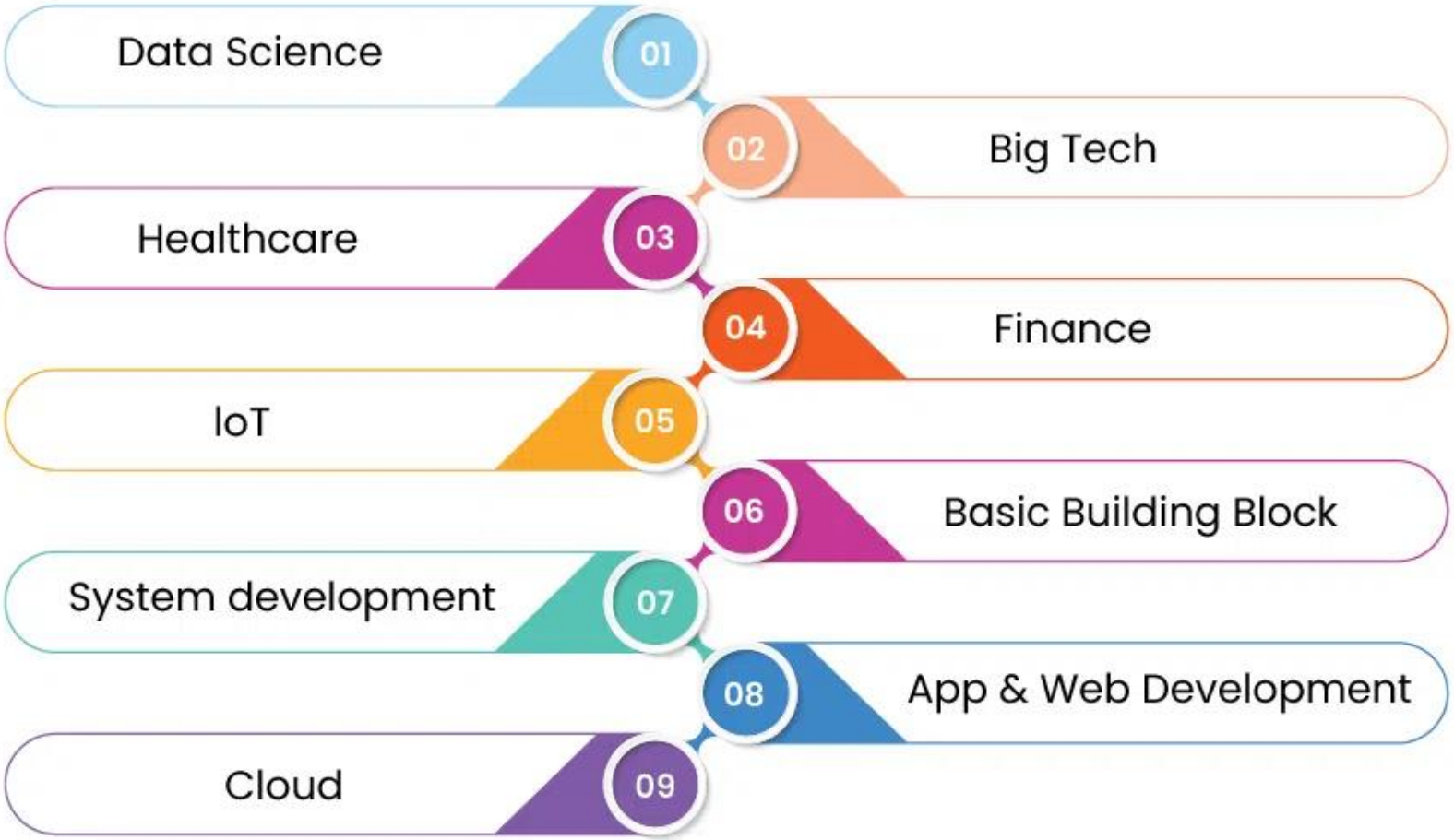
- **Provides a clear framework for software maintenance and future updates.**
- **Ensures software remains scalable, secure, and efficient over time.**

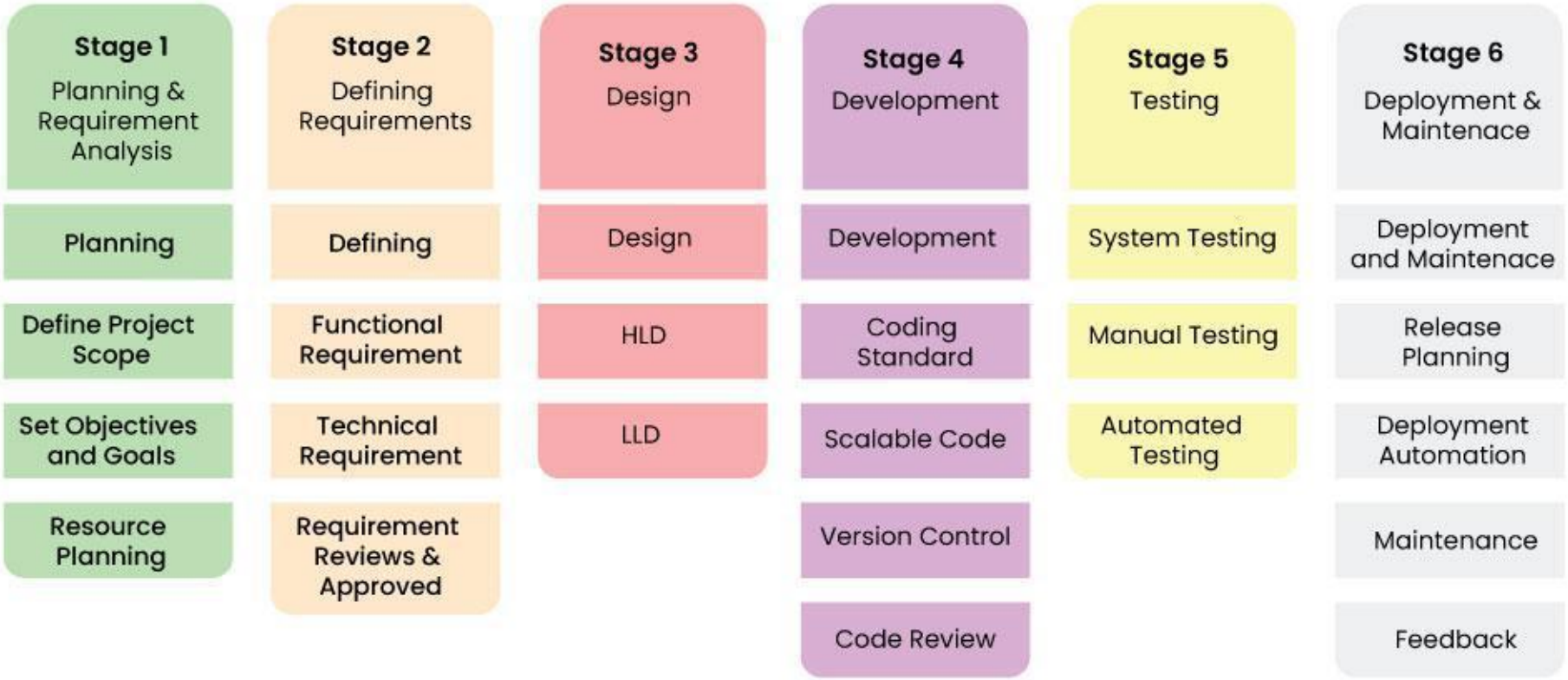
8. Supports Different Development Needs

- **Different SDLC models (Waterfall, Agile, Spiral, etc.) can be chosen based on project complexity, requirements, and risks.**


SDLC
Software Development Life Cycle
PHP
C#
HTML
1 Brainstorming & Setting Goals
2 Requirements Analysis & Plan
3 Design & Architecture
4 Coding & Implementation
5 Testing & QA
6 Deployment
7 Maintenance & Feedback

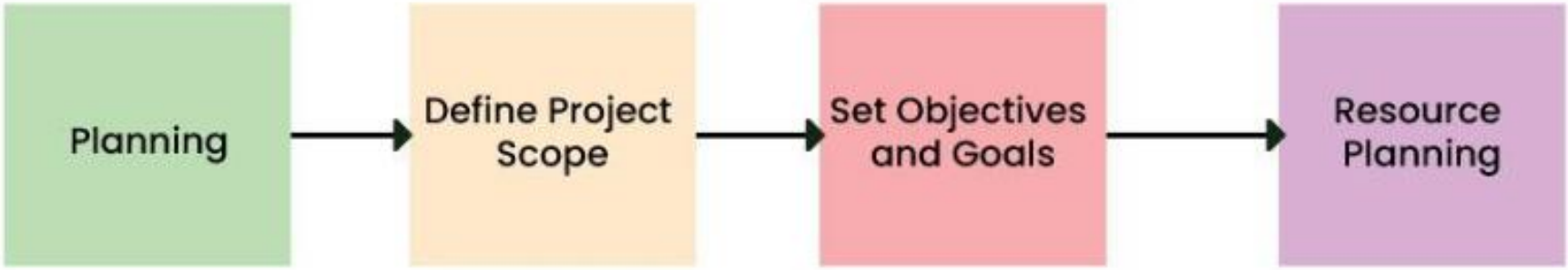
Real world applications of SDLC
(Software Development Life Cycle)





6 Stages of Software Development Life Cycle

Stage-1: Planning and Requirement Analysis



Planning is a crucial step in everything, just as in software development.

In this same stage, requirement analysis is also performed by the developers of the organization.

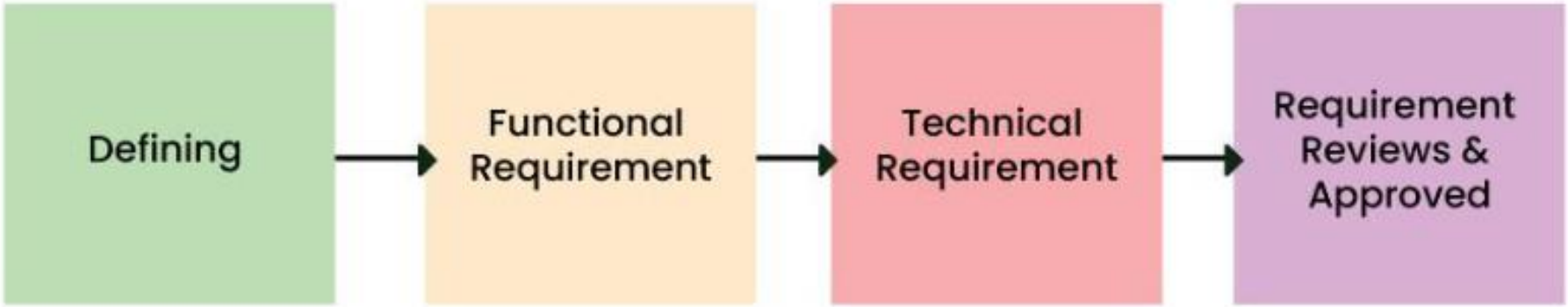
This is attained from customer inputs, and sales department/market surveys.

The information from this analysis forms the building blocks of a basic project.

The quality of the project is a result of planning.

The basic project is designed with all the available information.

Stage-2: Defining Requirements



In this stage, all the requirements for the target software are specified.

These requirements get approval from customers, market analysts, and stakeholders.

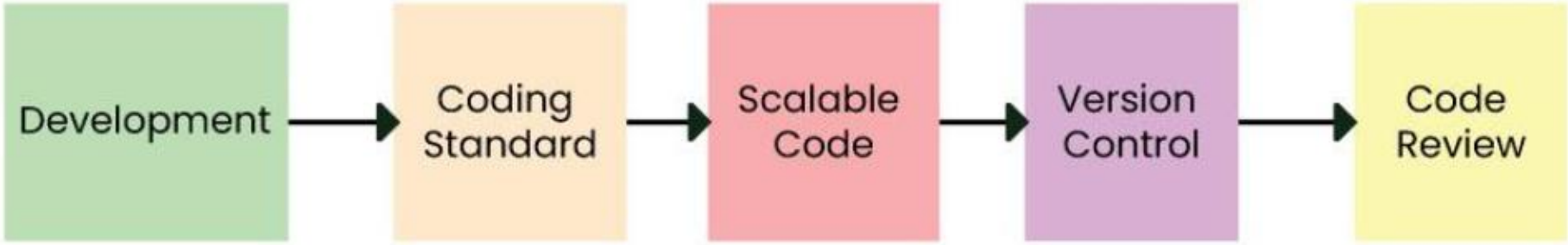
This is fulfilled by utilizing SRS (Software Requirement Specification).

This is a sort of document that specifies all those things that need to be defined and created during the entire project cycle.

Stage-3: Designing Architecture

- SRS is a reference for software designers to come up with the best architecture for the software.
- The requirements defined in SRS, multiple designs for the product architecture are present in the Design Document Specification (DDS).
- This DDS is assessed by market analysts and stakeholders.
- After evaluating all the possible factors, the most practical and logical design is chosen for development.

Stage-4: Developing Product

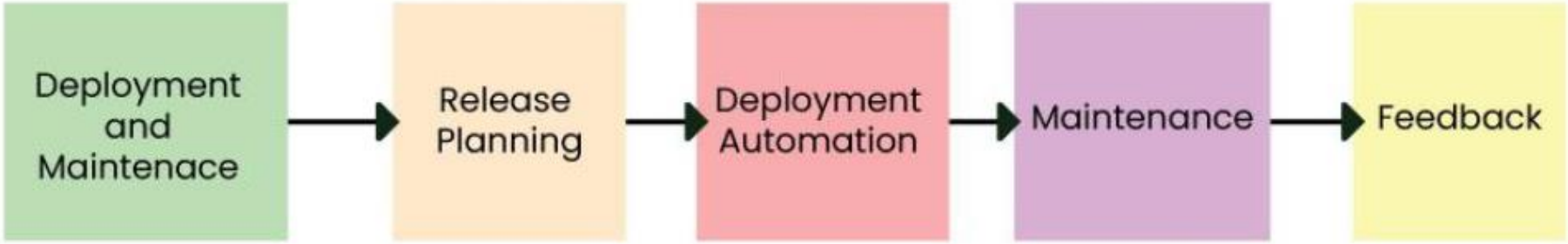


- At this stage, the fundamental development of the product starts.
- Developers use a specific programming code as per the design in the DDS.
- It is important for the coders to follow the protocols set by the association.
- Conventional programming tools like compilers, interpreters, debuggers, etc. are also put into use at this stage.
- Some popular languages like C/C++, Python, Java, etc. are put into use as per the software regulations.

Stage-5: Product Testing and Integration

- After the development of the product, testing of the software is necessary to ensure its smooth execution.
- Although, minimal testing is conducted at every stage of SDLC.
- Therefore, at this stage, all the probable flaws are tracked, fixed, and retested.
- This ensures that the product confronts the quality requirements of SRS.

Stage 6: Deployment and Maintenance of Products



WATERFALL MODEL AND ITS EXTENSIONS

- The waterfall model and its derivatives were extremely popular in the 1970s and still are heavily being used across many development projects.
- The waterfall model is possibly the most obvious and intuitive way in which software can be developed through team effort.
- We can think of the waterfall model as a generic model that has been extended in many ways for catering to specific software development situations.
- This has yielded all other software life cycle models.

Classical Waterfall Model

- Classical waterfall model is intuitively the most obvious way to develop software.
- It is simple but idealistic.
- It is hard to put this model into use in any non-trivial software development project, since developers do commit many mistakes during various development activities many of which are noticed only during a later phase.
- This requires to revisit the work of a previous phase to correct the mistakes, but the classical waterfall model has no provision to go back to modify the artifacts produced during an earlier phase.
- The phases starting from the *feasibility study to the integration and system testing* phase are known as the *development phases*.
- The last phase is also known as the *maintenance phase* of the life cycle.

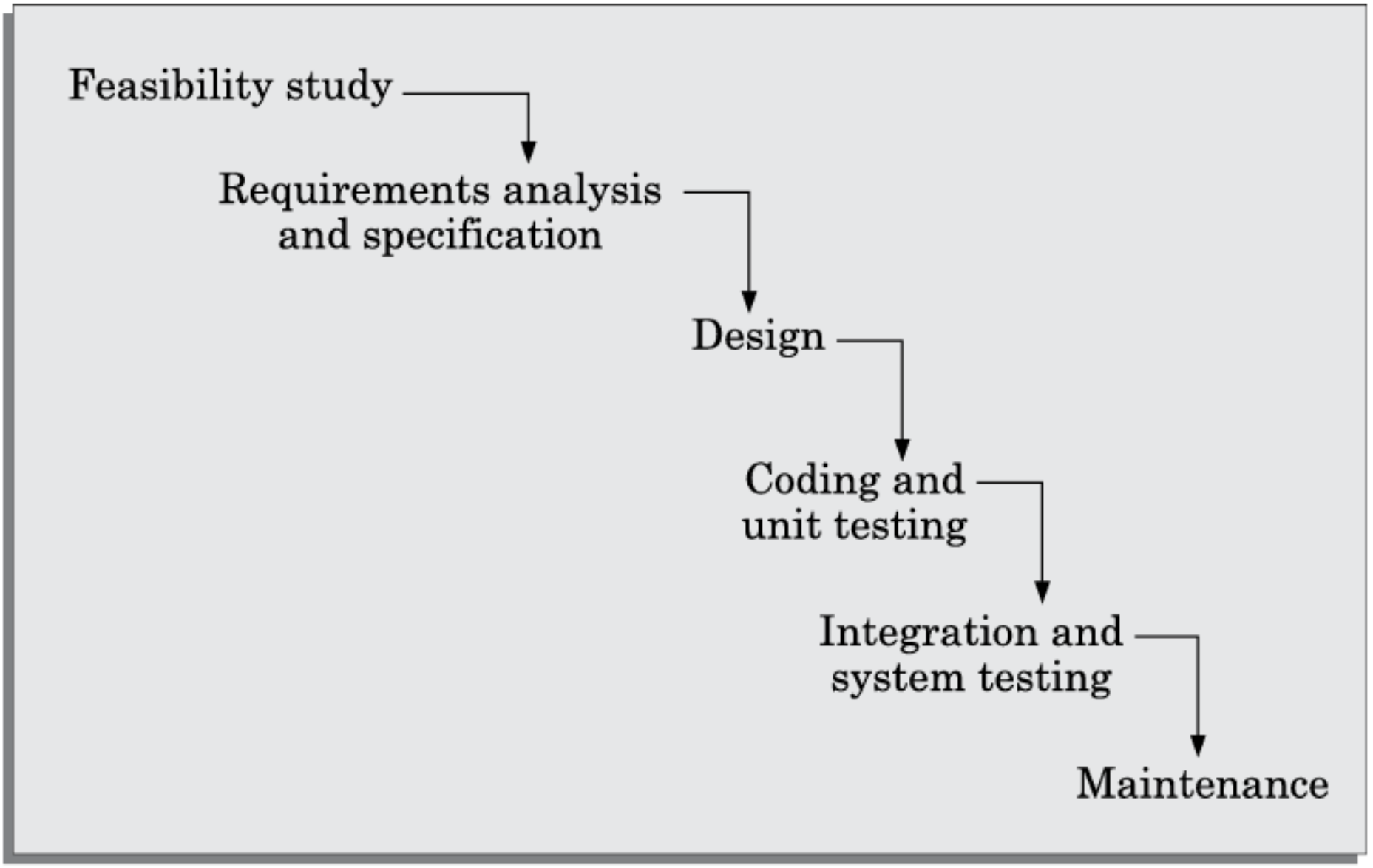


FIGURE 2.1 Classical waterfall model.

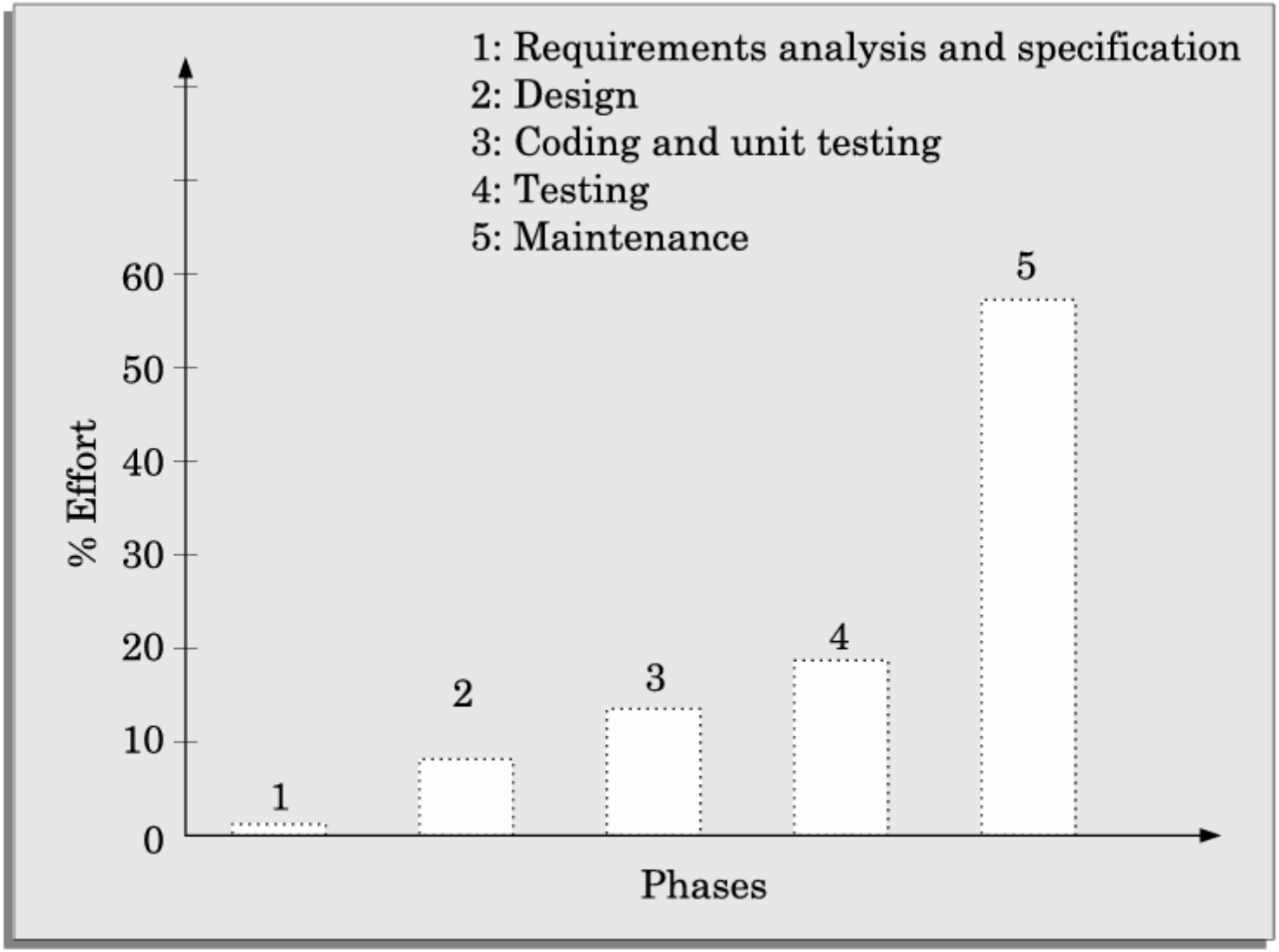


FIGURE 2.2 Relative effort distribution among different phases of a typical product.

Feasibility study

- The main focus of the feasibility study stage is to determine whether it would be financially and technically feasible to develop the software.
- The feasibility study involves carrying out several activities such as collection of basic information relating to the software such as the different data items that would be input to the system, the processing required to be carried out on these data, the output data required to be produced by the system, as well as various constraints on the development.

Development of an overall understanding of the problem:

It is necessary to first develop an overall understanding of what the customer requires to be developed.

Formulation of various possible strategies for solving the problem:

In this activity, various possible high-level solution schemes to the problem are determined.

Evaluation of the different solution strategies:

The different identified solution schemes are analysed to evaluate their benefits and shortcomings.

Requirements analysis and specification

This phase consists of two distinct activities, namely requirements gathering and analysis, and requirements specification.

■ Requirements gathering and analysis:

- The goal of the requirements gathering activity is to collect all relevant information regarding the software to be developed from the customer with a view to clearly understand the requirements.

■ Requirements specification:

- After the requirement gathering and analysis activities are complete, the identified requirements are documented.
- This document is called a software requirements specification (SRS) document.
- The SRS document is written using end-user terminology.
- This makes the SRS document understandable to the customer.
- The SRS document not only forms the basis for carrying out all the development activities, but several documents such as users' manuals, system test plan, etc. are prepared directly based on it.

Design

- ✓ The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.
- ✓ In technical terms, during the design phase the software architecture is derived from the SRS document.
- ✓ Two distinctly different design approaches are popularly being used at present – the procedural and object-oriented design approaches.

Procedural design approach:

- The traditional procedural design approach is in use in many software development projects at the present time.
- This traditional design technique is based on data flow modelling.
- It consists of two important activities; first structured analysis of the requirements specification is carried out where the data flow structure of the problem is examined and modelled.
- This is followed by a structured design step where the results of structured analysis are transformed into the software design.
- During structured analysis, the functional requirements specified in the SRS document are decomposed into subfunctions and the data-flow among these subfunctions is analysed and represented diagrammatically in the form of DFDs.

- ✓ Structured design is undertaken once the structured analysis activity is complete. Structured design consists of two main activities — architectural design (also called high-level design) and detailed design (also called Low-level design).
- ✓ High-level design involves decomposing the system into modules, and representing the interfaces and the invocation relationships among the modules.
- ✓ A high-level software design is sometimes referred to as the software architecture

Object-oriented design approach:

- ➢ In this technique, various objects that occur in the problem domain and the solution domain are first identified and the different relationships that exist among these objects are identified.
- ➢ The object structure is further refined to obtain the detailed design.
- ➢ The OOD approach is credited to have several benefits such as lower development time and effort, and better maintainability of the software.

Coding and unit testing:

- ✓ The purpose of the coding and unit testing phase is to translate a software design into source code and to ensure that individually each function is working correctly.
- ✓ The coding phase is also sometimes called the implementation phase, since the design is implemented into a workable solution in this phase.
- ✓ Each component of the design is implemented as a program module.
- ✓ The end-product of this phase is a set of program modules that have been individually unit tested.
- ✓ The main objective of unit testing is to determine the correct working of the individual modules.
- ✓ The specific activities carried out during unit testing include designing test cases, testing, debugging to fix problems, and management of test cases.

Integration and system testing

Integration testing is carried out to verify that the interfaces among different units are working satisfactorily. On the other hand, the goal of system testing is to ensure that the developed system conforms to the requirements that have been laid out in the SRS document

System testing usually consists of three different kinds of testing activities:

- **α-testing:** α testing is the system testing performed by the development team.
- **β-testing:** This is the system testing performed by a friendly set of customers.
- **Acceptance testing:** After the software has been delivered, the customer performs system testing to determine whether to accept the delivered software or to reject it.

Maintenance

- ✓ The total effort spent on maintenance of a typical software during its operation phase is usually far greater than that required for developing the software itself.
- ✓ Many studies carried out in the past confirm this and indicate that the ratio of relative effort of developing a typical software product and the total effort spent on its maintenance is roughly 40:60.

Maintenance is required in the following three types of situations:

- **Corrective maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
- **Perfective maintenance:** This type of maintenance is carried out to improve the performance of the system, or to enhance the functionalities of the system based on customer's requests.
- **Adaptive maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system.

Applications of Waterfall Model

- **Large-Scale Software Projects**: Works well for projects with clearly defined requirements and large scope.
- **Safety-Critical Systems**: Ideal for systems where failure could result in serious consequences (e.g., medical devices, aerospace systems).
- **Government/Defense Projects**: Suitable for structured, rigorous projects requiring clear documentation.
- **Well-Defined Projects**: Works best when the project requirements are stable and unlikely to change.

Advantages of Waterfall Model

1. **Simplicity**: Easy to understand and implement.
2. **Clear Milestones**: Clearly defined phases and checkpoints for progress assessment.
3. **Proper Documentation**: Comprehensive documentation ensures clarity and serves as a future reference.
4. **Best for Smaller Projects**: Effective for projects with well-understood requirements and smaller scopes.

Disadvantages of Waterfall Model

1. **No Feedback Path**: Once a phase is completed, there's no going back to modify earlier stages, which makes correcting errors difficult.
2. **Inflexibility**: It is hard to accommodate changes after the requirements specification phase is completed.
3. **Late Defect Detection**: Issues may only surface later in the process, leading to costly fixes.
4. **Limited Stakeholder Involvement**: Stakeholders are usually only involved during the early phases, limiting their input throughout the project.
5. **Long Development Cycle**: The rigid, step-by-step process can make the development cycle lengthy.

Extensions of the Waterfall Model

1. V-Model (Verification and Validation Model)
2. Incremental Model
3. Spiral Model
4. Iterative Model
5. Rapid Application Development (RAD)

Iterative Waterfall Model

The main change brought about by the iterative waterfall model to the classical waterfall model is in the form of providing feedback paths from every phase to its preceding phases.

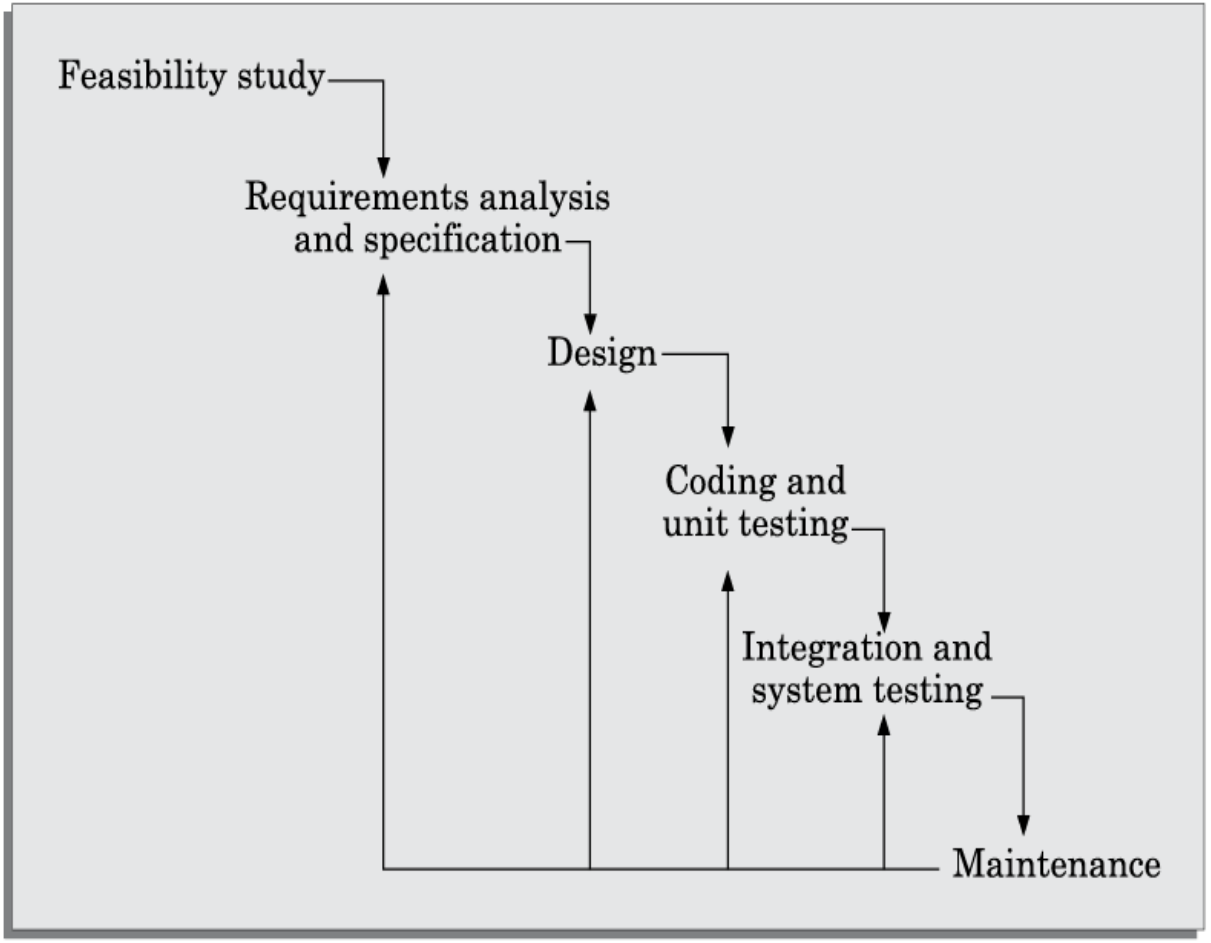


FIGURE 2.3 Iterative waterfall model.

- ✓ **Requirements Gathering:** This is the first stage where the business owners and developers meet to discuss the goals and requirements of the website.
- ✓ **Design:** In this stage, the developers create a preliminary design of the website based on the requirements gathered in stage 1.
- ✓ **Implementation:** In this stage, the developers begin to build the website based on the design created in stage 2.
- ✓ **Testing:** Once the website has been built, it is tested to ensure that it meets the requirements and functions properly.
- ✓ **Deployment:** The website is then deployed and made live to the public.
- ✓ **Review and Improvement:** After the website has been live for a while, the business owners and developers review its performance and make any necessary improvements.

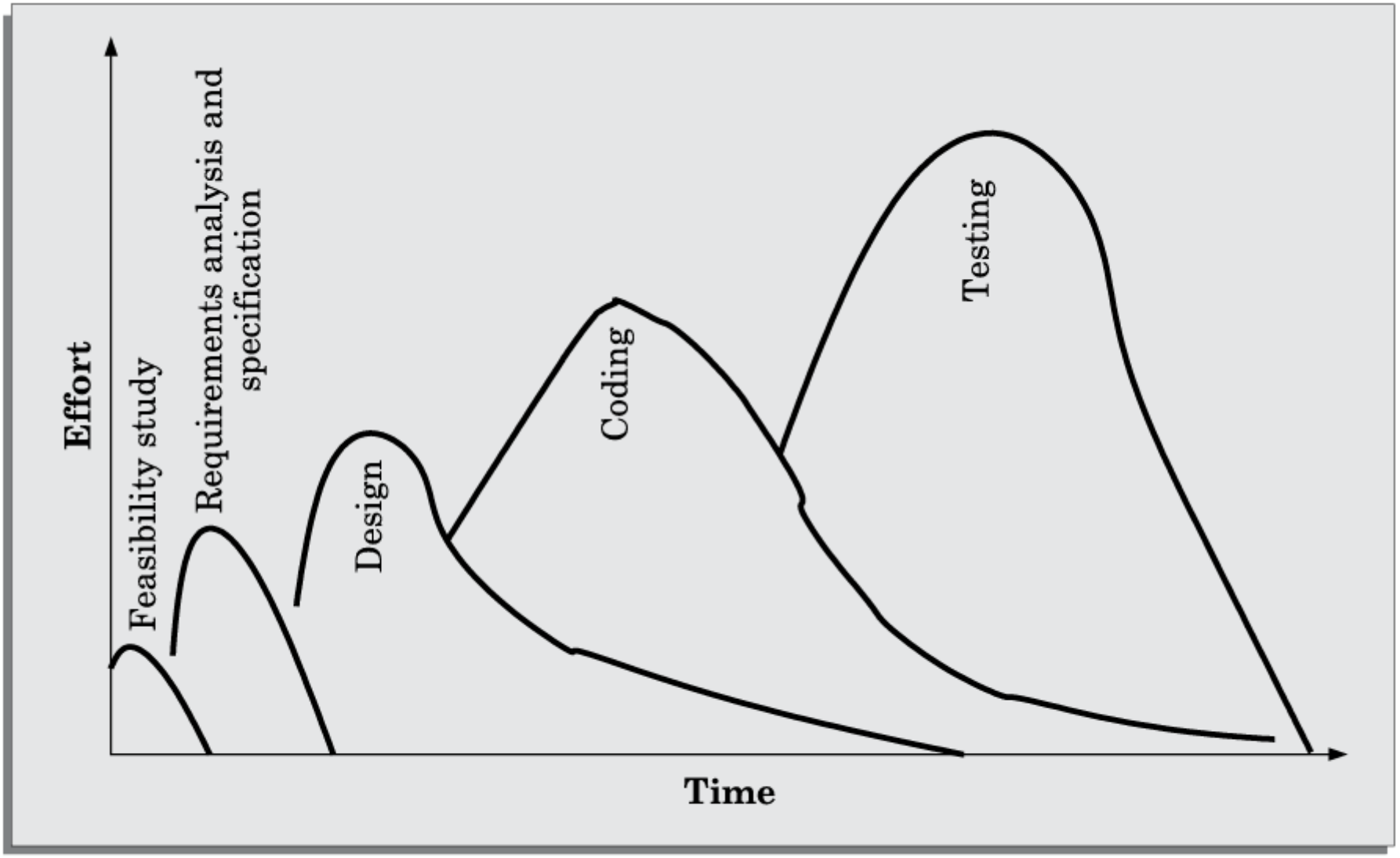


FIGURE 2.4 Distribution of effort for various phases in the iterative waterfall model.

Application of Iterative Waterfall Model

1.The essential needs are established, but as time passes, the finer points may become relevant.

2.Programmers have a learning curve to climb when they utilize new technology.

3.The resources needed to complete a large project are constrained, hence on a smaller scale, the automation is more temporary.

4.Very high risk as the project's objective may occasionally alter.

Advantages of Iterative Waterfall Model

1.Phase Containment of Errors: Errors are detected and fixed as close to their source as possible, reducing costly rework and delays.

2.Collaboration: Continuous collaboration between business owners and developers ensures the product meets business needs and improves with feedback at each iteration.

3.Flexibility: The model allows for easy incorporation of new requirements or features in subsequent iterations, ensuring the product evolves with the business.

4.Testing and Feedback: Regular testing and feedback cycles help identify and fix issues early, improving the product's quality and relevance.

5.Faster Time to Market: Incremental development allows parts of the product to be delivered sooner, enabling user feedback while further improvements are made.

6.Risk Reduction: Continuous feedback and testing help identify risks early, reducing the likelihood of costly errors and delays.

Shortcomings of the iterative waterfall model

1.Difficult to incorporate change requests: The major drawback of the iterative waterfall model is that all the requirements must be clearly stated before starting the development phase. Customers may change requirements after some time but the iterative waterfall model does not leave any scope to incorporate change requests that are made after the development phase starts.

2.Incremental delivery not supported: In the iterative waterfall model, the full software is completely developed and tested before delivery to the customer. There is no scope for any intermediate delivery. So, customers have to wait a long for getting the software.

3.Overlapping of phases not supported: Iterative waterfall model assumes that one phase can start after completion of the previous phase, But in real projects, phases may overlap to reduce the effort and time needed to complete the project.

4.Risk handling not supported: Projects may suffer from various types of risks. But, the Iterative waterfall model has no mechanism for risk handling.

5.Limited customer interactions: Customer interaction occurs at the start of the project at the time of requirement gathering and at project completion at the time of software delivery. These fewer interactions with the customers may lead to many problems as the finally developed software may differ from the customers' actual requirements.

Incremental Development Model

- In the incremental life cycle model, the software is developed in increment.
- In this life cycle model, first the requirements are split into a set of increments.
- The first increment is a simple working system implementing only a few basic features.
- Over successive iterations, successive increments are implemented and delivered to the customer until the desired system is realised.

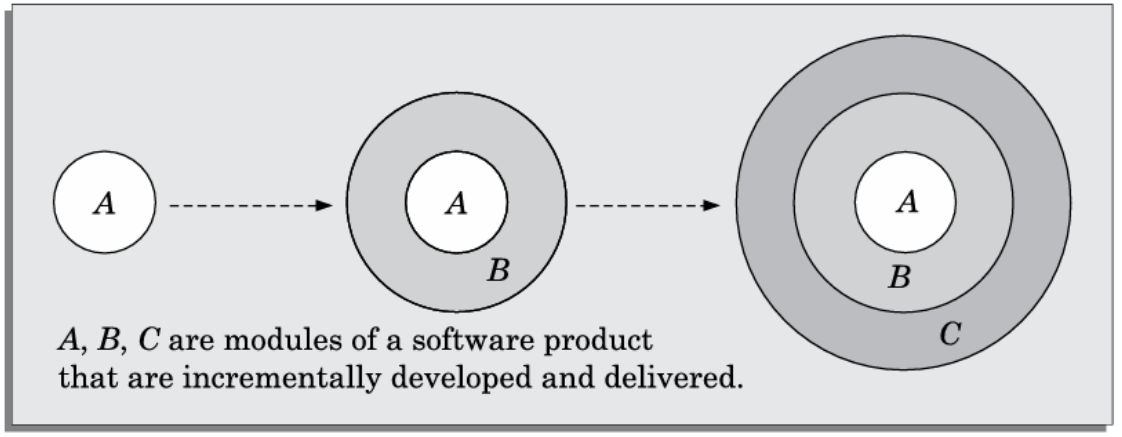


FIGURE 2.7 Incremental software development.

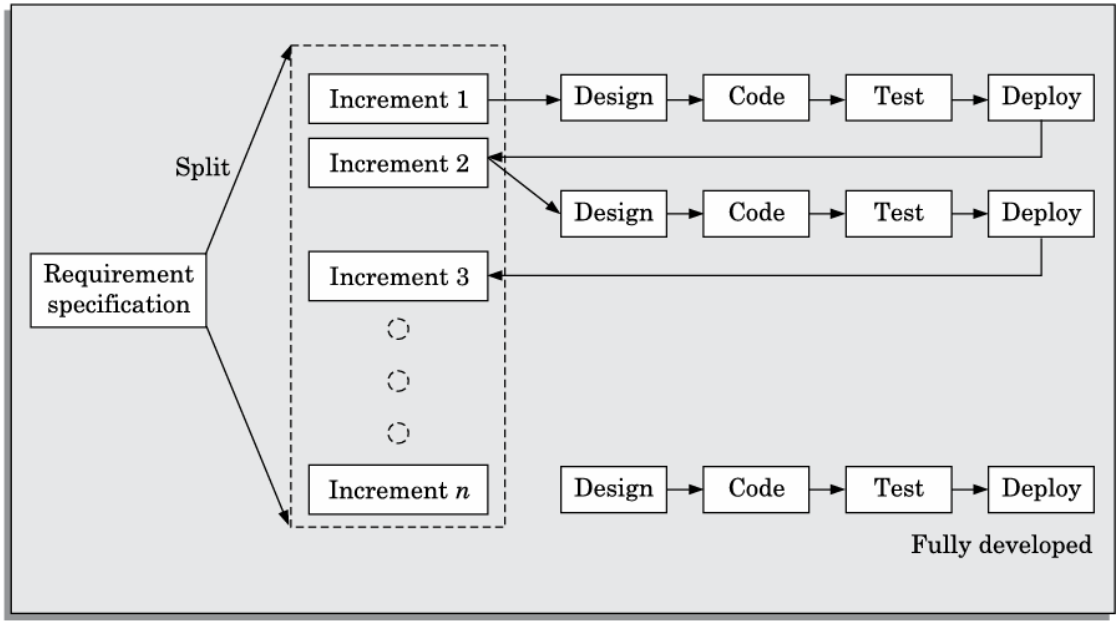


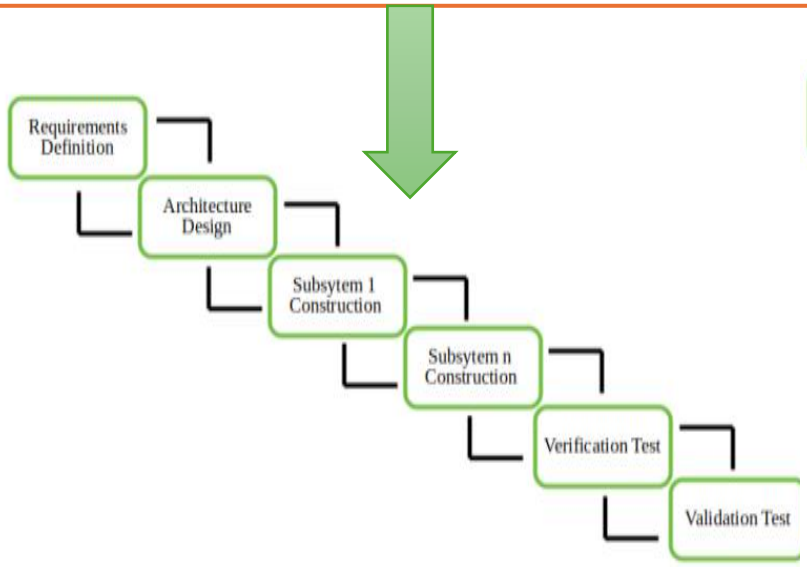
FIGURE 2.8 Incremental model of software development.

Types of Incremental Model

Staged Delivery Model:

This type of incremental model involves only one part of the project being built at a time.

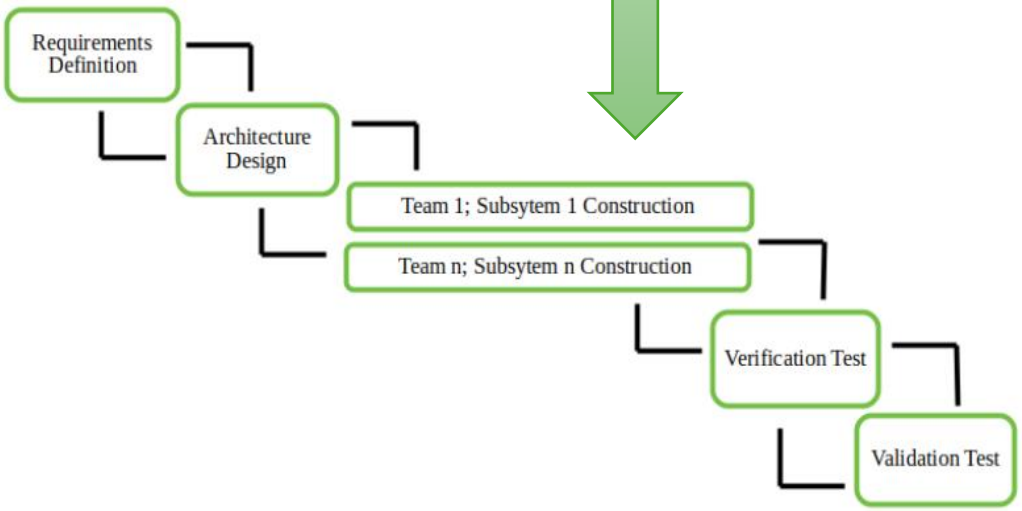
In the staged-delivery model, you do not deliver the software all at once, but rather in successive stages throughout the project as shown below.



Parallel Development Model:

This incremental model involves the simultaneous development of different sub-systems as shown below.

As long as sufficient resources are available, it can decrease the amount of time needed for the development process, i.e., TTM (Time to Market).



Advantages –

- Error Reduction (core modules are used by the customer from the beginning of the phase and then these are tested thoroughly)
- Uses divide and conquer for breakdown of tasks.
- Lowers initial delivery cost.
- Incremental Resource Deployment.

Disadvantages –

- Requires good planning and design.
- Total cost is not lower.
- Well defined module interfaces are required.

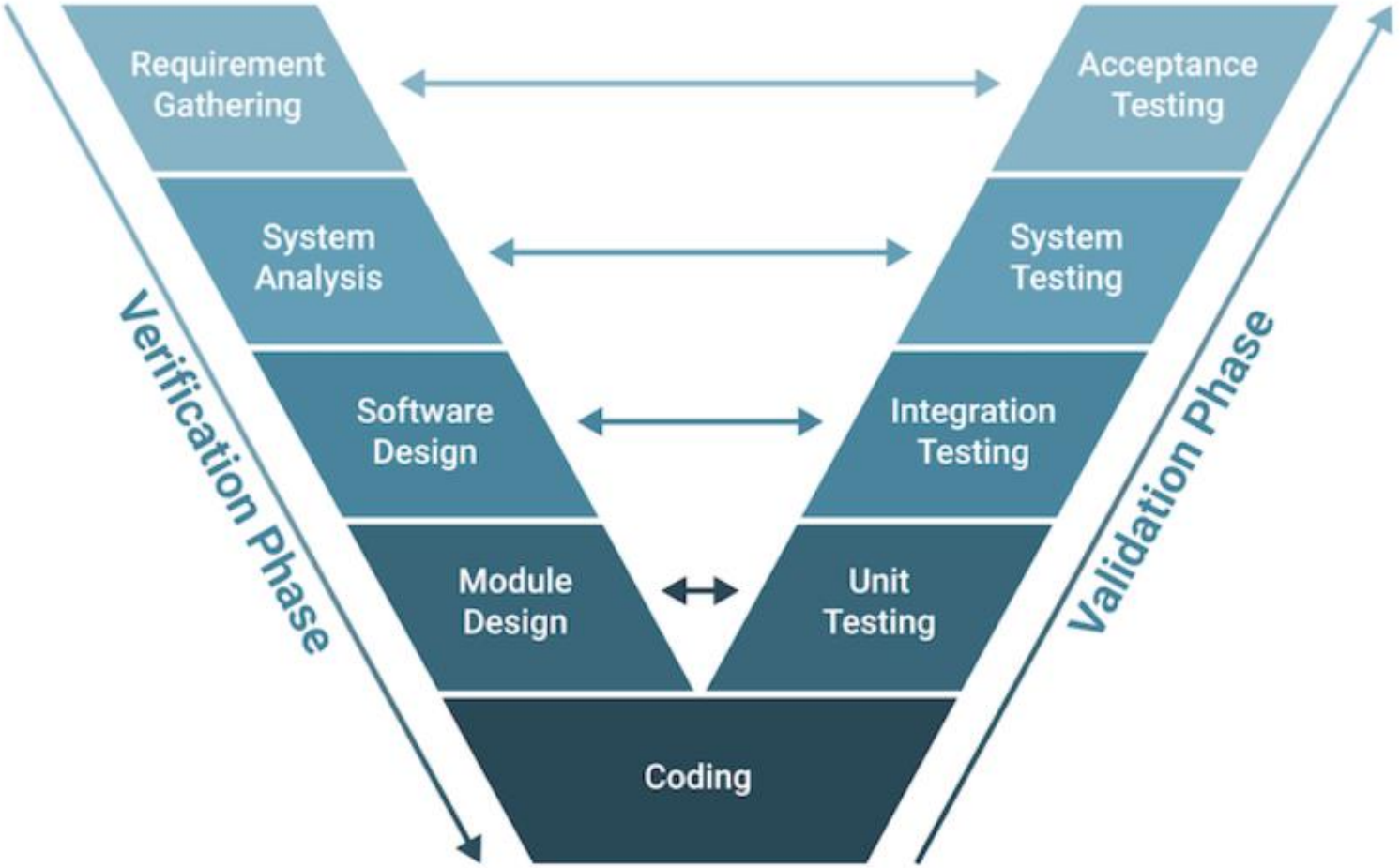
Applications:

1. Funding Schedule, Risk, Program Complexity, or need for early realization of benefits.
2. When Requirements are known up-front.
3. When Projects having lengthy developments schedules.
4. Projects with new Technology

V-Model

There are two main phases—development and validation phases.

The left half of the model comprises the development phases and the right half comprises the validation phases.



When to Use a V-Model

- V-models are used in situations wherein the requirements and understanding of the software's functionality are well-defined from the beginning.
- The V-model assumes stable requirements and discourages iteration, so in cases where the project scope is clear and the development team has a solid understanding of the requirements, the V-model can be an effective tool for delivering high-quality software.

V-Model Verification Phases

The verification phase refers to the practice of evaluating the product development process to ensure the team meets the specified requirements.

1. Development Phases (Verification)

These are the stages where the product is designed, developed, and coded:

- **Business Requirement Analysis:** Gather and analyze customer requirements to define the scope and expectations of the project.
- **System Design:** Develop a high-level design of the system based on business requirements. This includes defining architecture and structure.
- **Architectural Design:** Break down the system into modules, defining the communication and data exchange between them.
- **Module Design (Low-Level Design):** Detailed design of individual system modules and their functionalities.
- **Coding:** Implement the design by writing code, adhering to coding standards and ensuring it meets the system and architectural requirements.

2. Testing Phases (Validation)

These phases focus on testing to ensure the product meets user expectations:

- **Unit Testing:** Test individual components (or units) to check for bugs at the code level. This is planned during the module design phase.
- **Integration Testing:** After unit testing, integrate modules and verify communication and data transfer between them.
- **System Testing:** Test the entire system as a whole to verify that all components work together correctly.
- **User Acceptance Testing (UAT):** Conduct tests in a user environment to ensure the software meets real-world requirements.

Advantages of Using the V-Model

- **Improves Quality**: From the beginning, the V-model ensures that quality is built into the development process, which results in fewer bugs in code and higher-quality software.
- **Reduces Risks**: The V-model provides a clear roadmap for the entire development process, which allows for better risk management and mitigation.
- **Increases Efficiency**: The V-model encourages collaboration between different teams and stakeholders, which results in more efficient development and testing.
- **Improves Communication**: The V-model emphasizes communication between stakeholders, to ensure everyone has a clear understanding of the requirements and objectives.
- **Enhances Testing**: The V-model places a strong emphasis on thorough and effective testing throughout the entire development process.
- **Improves Documentation**: The V-model requires comprehensive documentation at every stage of the development process, which leads to better record-keeping and easier code maintenance.

Disadvantages of Using the V-Model

- **Rigid**: The V-model can be inflexible and provide very little room for changes or deviations from the plan. This rigidity can make it difficult to adapt to changing project requirements or new information.
- **Time-Consuming**: The V-model can be time-consuming due to its focus on thorough planning and documentation at every stage. These factors can slow down the development process and lead to longer project timelines.
- **Resource Intensive**: The V-model requires a significant amount of resources including time, budget and personnel, thereby making it a difficult model for small teams or organizations with limited resources to implement.
- **Limited Agility**: The V-model may not be well suited for Agile development approaches, which rely on flexibility, iterative development and continuous feedback.
- **Overemphasis on Testing**: While thorough testing is a critical component of software development, the V-Model may place too much emphasis on testing, which can lead to production delays and increased costs.

Evolutionary Model

- ✓ This model has many of the features of the incremental model.
- ✓ The principal idea behind the evolutionary life cycle model is conveyed by its name.
- ✓ In the evolutionary model, the requirements, plan, estimates, and solution evolve over the iterations, rather than fully defined and frozen in a major up-front specification effort before the development iterations begin.
- ✓ Such evolution is consistent with the pattern of unpredictable feature discovery and feature changes that take place in new product development.
- ✓ the evolutionary software development process is sometimes referred to as design a little, build a little, test a little, deploy a little model.

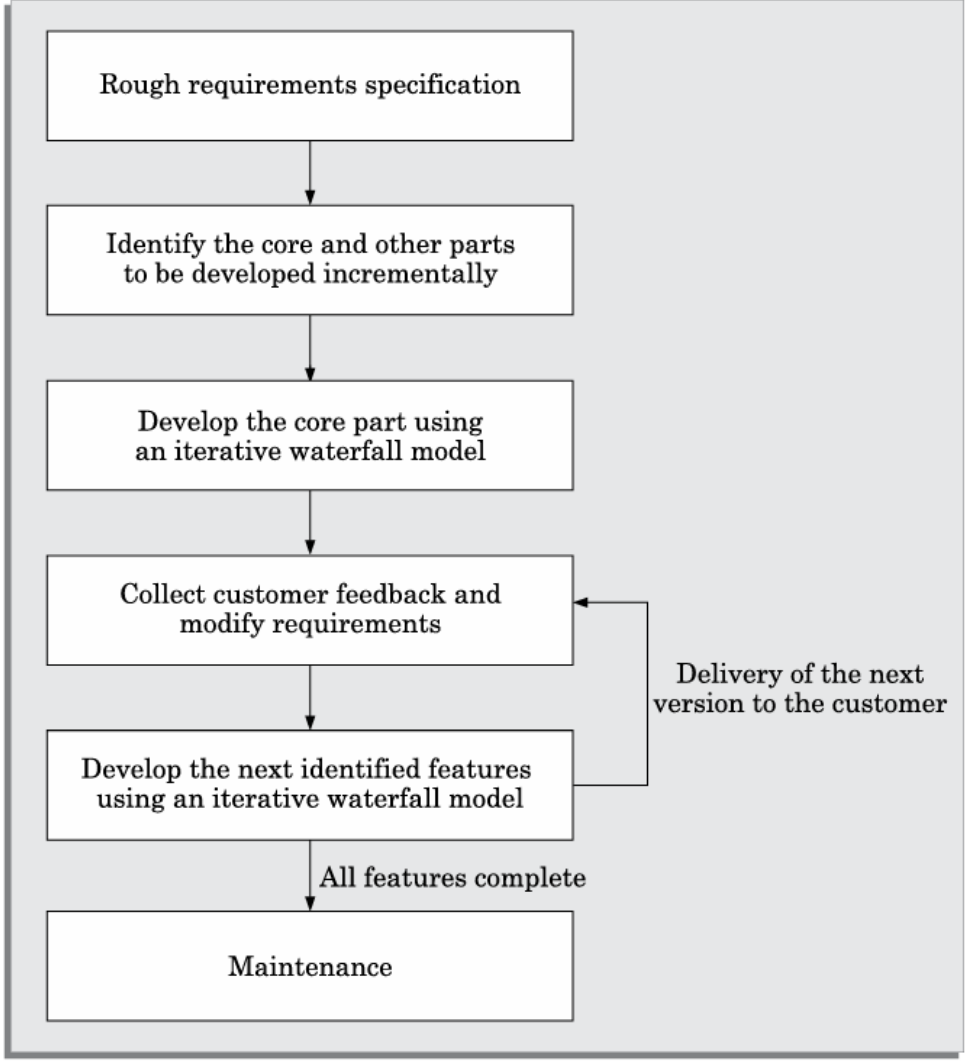


FIGURE 2.9 Evolutionary model of software development.

- ✓ This means that after the requirements have been specified, the design, build, test, and deployment activities are iterated.

Advantages

The evolutionary model of development has several advantages. Two important advantages of using this model are the following:

- **Effective elicitation of actual customer requirements:** In this model, the user gets a chance to experiment with a partially developed software much before the complete requirements are developed. Therefore, the evolutionary model helps to accurately elicit user requirements with the help of feedback obtained on the delivery of different versions of the software. As a result, the change requests after delivery of the complete software gets substantially reduced.
- **Easy handling change requests:** In this model, handling change requests is easier as no long-term plans are made. Consequently, reworks required due to change requests are normally much smaller compared to the sequential models.

Disadvantages

The main disadvantages of the successive versions model are as follows:

- **Feature division into incremental parts can be non-trivial:** For many development projects, especially for small-sized projects, it is difficult to divide the required features into several parts that can be incrementally implemented and delivered. Further, even for larger problems, often the features are so intertwined and dependent on each other that even an expert would need considerable effort to plan the incremental deliveries.
- ***Ad hoc* design:** Since at a time design for only the current increment is done, the design can become ad hoc without specific attention being paid to maintainability and optimality. Obviously, for moderate sized problems and for those for which the customer requirements are clear, the iterative waterfall model can yield a better solution.

Prototyping Model

The prototyping model is considered to be useful for the development of not only the GUI parts of a software, but also for a software project for which certain technical issues are not clear to the development team.

The GUI part of a software system is almost always developed using the prototyping model.

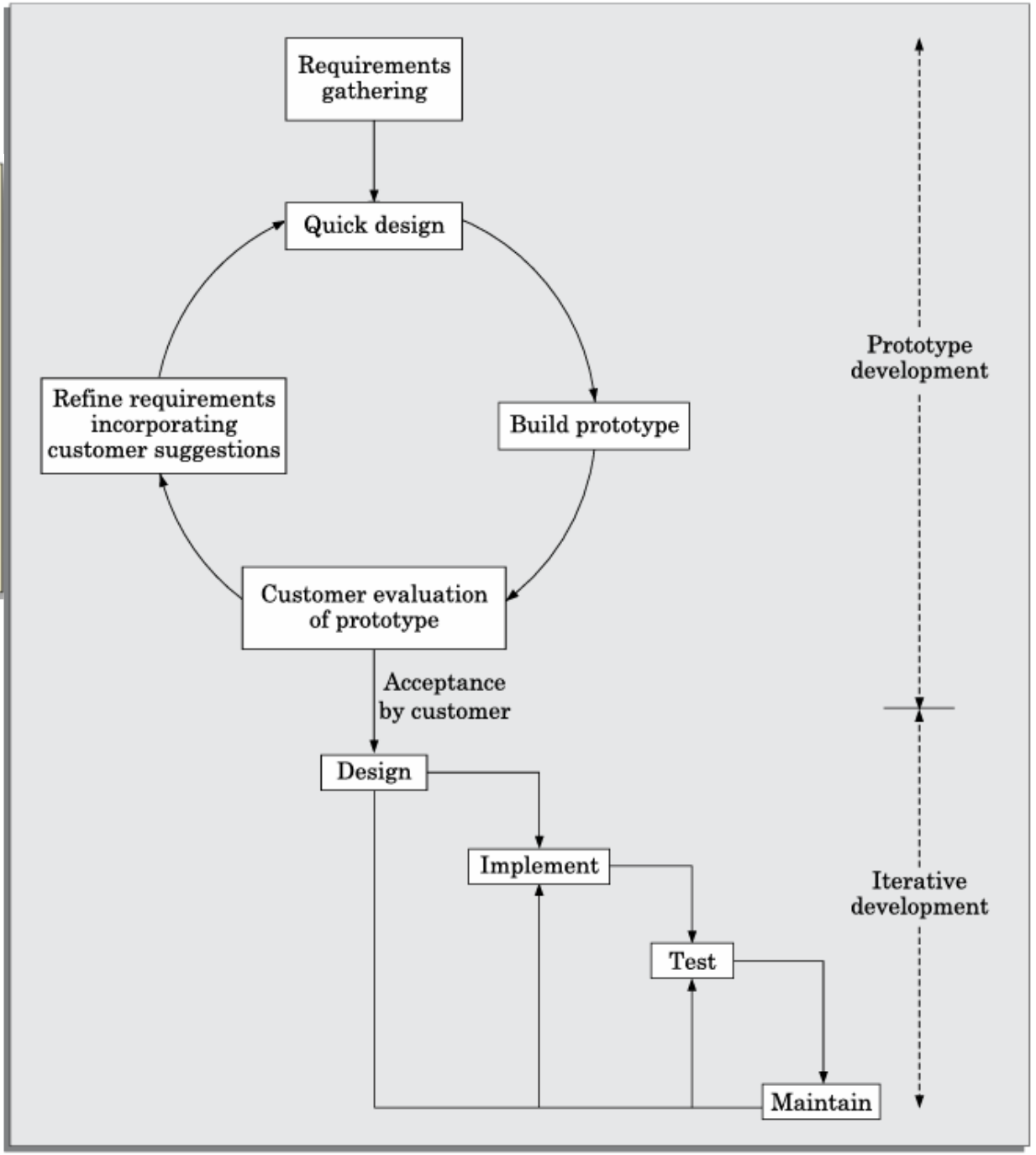


FIGURE 2.6 Prototyping model of software development.

Prototype development:

- Prototype development starts with an initial requirements gathering phase.
- A quick design is carried out and a prototype is built.
- The developed prototype is submitted to the customer for evaluation.
- Based on the customer feedback, the requirements are refined and the prototype is suitably modified.
- This cycle of obtaining customer feedback and modifying the prototype continues till the customer approves the prototype.

Iterative development:

- Once the customer approves the prototype, the actual software is developed using the iterative waterfall approach.
- In spite of the availability of a working prototype, the SRS document is usually needed to be developed since the SRS document is invaluable for carrying out traceability analysis, verification, and test case design during later phases.
- However, for GUI parts, the requirements analysis and specification phase becomes redundant since the working prototype that has been approved by the customer serves as an animated requirements specification.

Advantages of Prototype Model

•The customer gets to see partial products early in the lifecycle, hence ensuring customer satisfaction.

•The developed prototype can be **reused** for bigger projects in the future.

•There is scope to accommodate new requirements.

•**Errors and missing functionalities** can be identified much early in the lifecycle because the users are actively involved.

•User feedback is accommodated quickly.

•The model is very straightforward and does not require skilled experts to implement.

Disadvantages of Prototype Model

•Prototyping is a **slow and time taking process**.

•Documentation is poor as the requirements **change frequently**.

•When the customer evaluates the prototype, there may be much too many variances in software needs.

•There is a risk of inadequate requirement analysis owing to too much dependency on the prototype.

RAPID APPLICATION DEVELOPMENT (RAD)

- The rapid application development (RAD) model was proposed in the early nineties to overcome the rigidity of the waterfall model (and its derivatives) that makes it difficult to accommodate any change requests from the customer.
- It proposed a few radical extensions to the waterfall model.
- This model has the features of both prototyping and evolutionary models.
- It deploys an evolutionary delivery model to obtain and incorporate the customer feedbacks on incrementally delivered versions.
- In this model prototypes are constructed, and incrementally the features are developed and delivered to the customer.
- But unlike the prototyping model, the prototypes are not thrown away but are enhanced and used in the software construction.

The major goals of the RAD model are as follows:

- To decrease the time taken and the cost incurred to develop software systems.
- To limit the costs of accommodating change requests.
- To reduce the communication gap between the customer and the developers.

- ✓ This is intended to make the system tuned to the exact customer requirements and also to bridge the communication gap between the customer and the development team.
- ✓ The development team usually consists of about five to six members, including a customer representative.

Working of RAD

- ✓ In the RAD model, development takes place in a series of short cycles or iterations.
- ✓ At any time, the development team focuses on the present iteration only, and therefore plans are made for one increment at a time.
- ✓ The time planned for each iteration is called a *time box.*
- ✓ Each iteration is planned to enhance the implemented functionality of the application by only a small amount.
- ✓ During each time box, a quick-and-dirty prototype-style software for some functionality is developed.
- ✓ The customer evaluates the prototype and gives feedback on the specific improvements that may be necessary.
- ✓ The prototype is refined based on the customer feedback. Please note that the prototype is not meant to be released to the customer for regular use though.
- ✓ The development team almost always includes a customer representative to clarify the requirements.

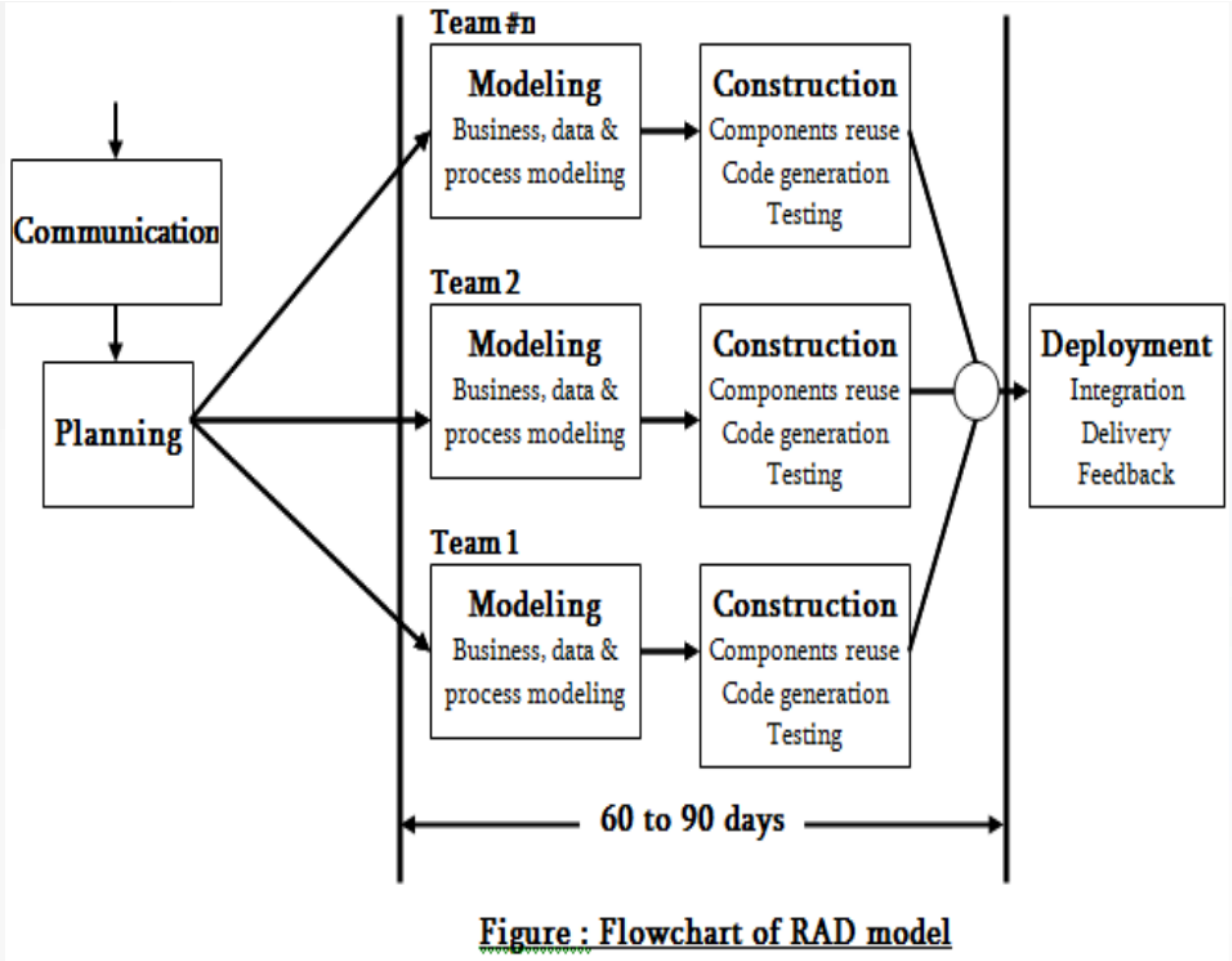


Figure : Flowchart of RAD model

Advantages:

- Fast products.
- Efficient Documentation.
- Interaction with user.

Disadvantages:

- User may not like fast activities.
- Not suitable for technical risks.

SPIRAL MODEL

- The exact number of loops of the spiral is not fixed and can vary from project to project.
- Each loop of the spiral is called a phase of the software process.
- The exact number of phases through which the product is developed can be varied by the project manager depending upon the project risks.
- A prominent feature of the spiral model is handling unforeseen risks that can show up much after the project has started.

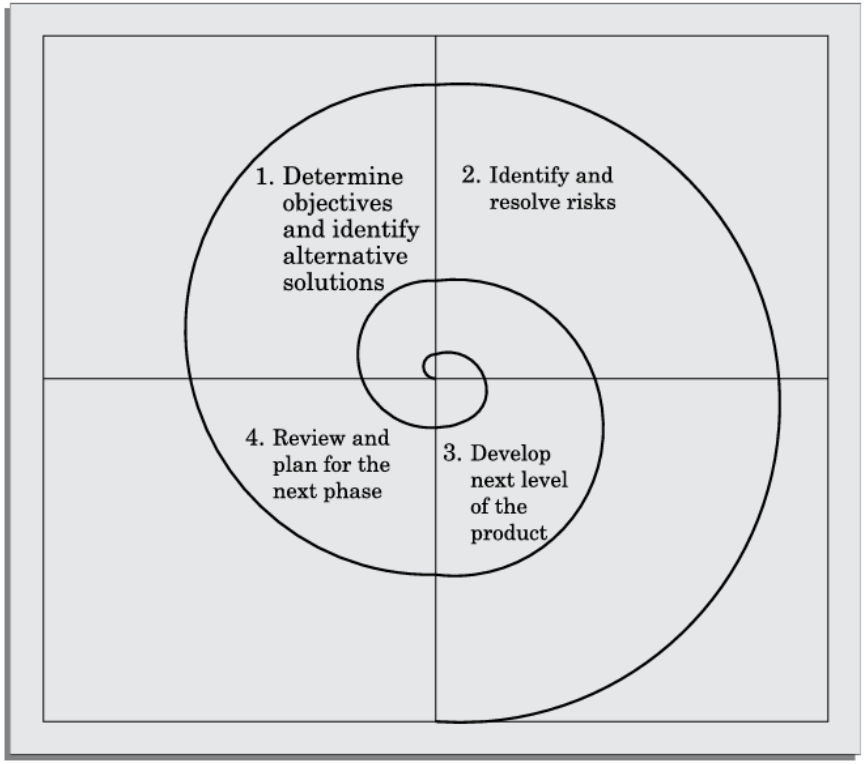


FIGURE 2.12 Spiral model of software development.

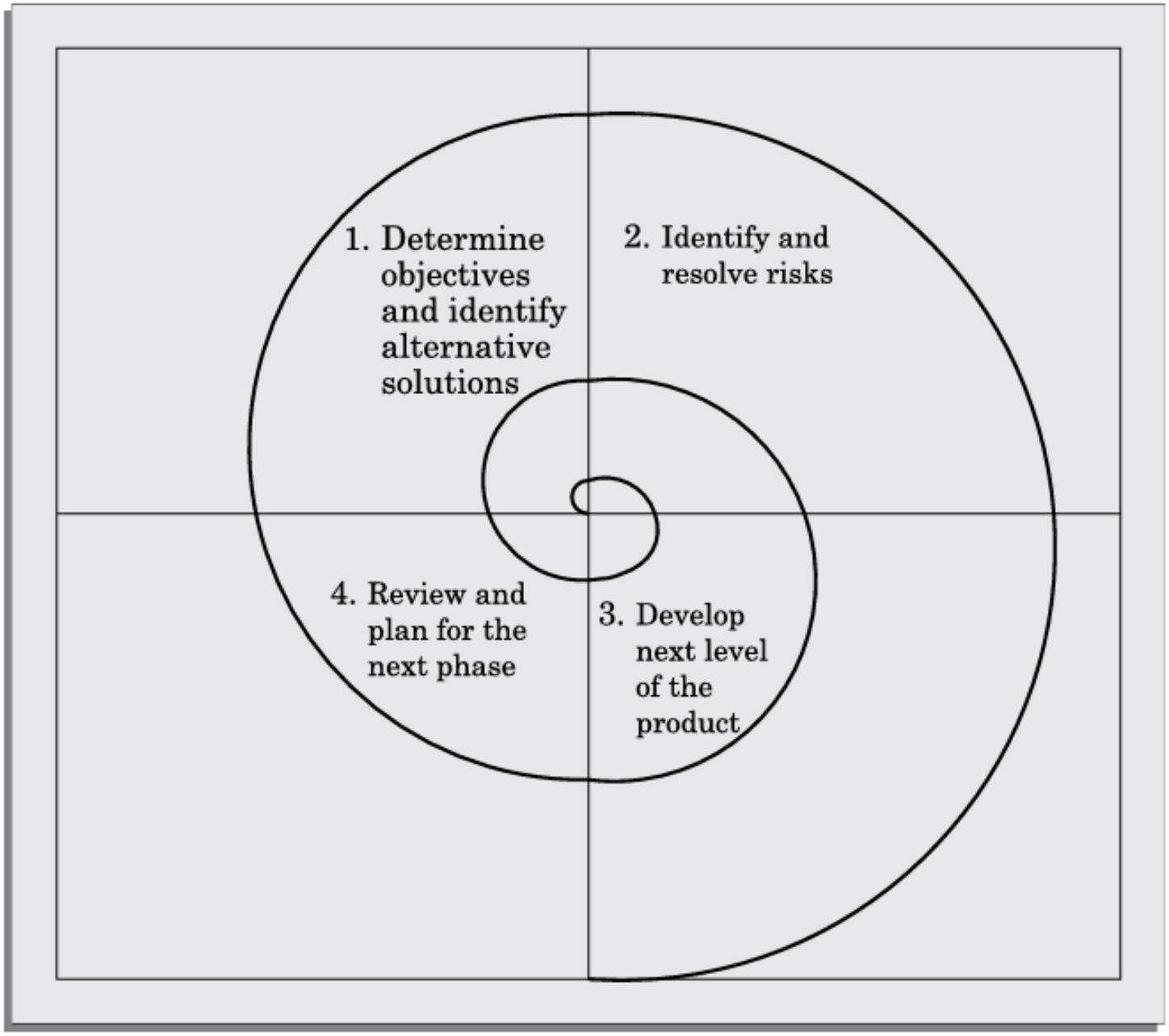


FIGURE 2.12 Spiral model of software development.

Phases of the Spiral Model

Quadrant 1: The objectives are investigated, elaborated, and analysed. Based on this, the risks involved in meeting the phase objectives are identified. In this quadrant, alternative solutions possible for the phase under consideration are proposed.

Quadrant 2: During the second quadrant, the alternative solutions are evaluated to select the best possible solution. To be able to do this, the solutions are evaluated by developing an appropriate prototype.

Quadrant 3: Activities during the third quadrant consist of developing and verifying the next level of the software. At the end of the third quadrant, the identified features have been implemented and the next version of the software is available.

Quadrant 4: Activities during the fourth quadrant concern reviewing the results of the stages traversed so far (i.e. the developed version of the software) with the customer and planning the next iteration of the spiral.

The radius of the spiral at any point represents the cost incurred in the project so far, and the angular dimension represents the progress made so far in the current phase.

Risk handling in spiral model

A risk is any adverse situation that might affect the successful completion of a software project. The most important feature of the spiral model is handling these unknown risks after the project has started. Such risk resolutions are easier done by developing a prototype.

1.The spiral model supports coping with risks by providing the scope to build a prototype at every phase of software development.

2.The **Prototyping Model** also supports risk handling, but the risks must be identified completely before the start of the development work of the project.

3.But in real life, project risk may occur after the development work starts, in that case, we cannot use the Prototyping Model.

4.In each phase of the Spiral Model, the features of the product dated and analyzed, and the risks at that point in time are identified and are resolved through prototyping.

5.Thus, this model is much more flexible compared to other SDLC models.

Spiral model as a meta model

The Spiral model is called a **Meta-Model** because it subsumes all the other SDLC models. For example, a single loop spiral actually represents the **Iterative Waterfall Model**.

1.The spiral model incorporates the stepwise approach of the **Classical Waterfall Model**.

2.The spiral model uses the approach of the **Prototyping Model** by building a prototype at the start of each phase as a risk-handling technique.

3.Also, the spiral model can be considered as supporting the **Evolutionary model** - the iterations along the spiral can be considered as evolutionary levels through which the complete system is built.

Advantages/pros and disadvantages/cons of the spiral model

1.Risk Handling: The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.

2.Good for large projects: It is recommended to use the Spiral Model in large and complex projects.

3.Flexibility in Requirements: Change requests in the Requirements at a later phase can be incorporated accurately by using this model.

4.Customer Satisfaction: Customers can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

5.Iterative and Incremental Approach: The Spiral Model provides an iterative and incremental approach to software development, allowing for flexibility and adaptability in response to changing requirements or unexpected events.

6.Emphasis on Risk Management: The Spiral Model places a strong emphasis on risk management, which helps to minimize the impact of uncertainty and risk on the software development process.

Advantages/pros and disadvantages/cons of the spiral model

7.Improved Communication: The Spiral Model provides for regular evaluations and reviews, which can improve communication between the customer and the development team.

8.Improved Quality: The Spiral Model allows for multiple iterations of the software development process, which can result in improved software quality and reliability.

disadvantages/cons of the spiral model

1.Complex: The Spiral Model is much more complex than other SDLC models.

2.Expensive: Spiral Model is not suitable for small projects as it is expensive.

3.Too much dependability on Risk Analysis: The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.

4.Difficulty in time management: As the number of phases is unknown at the start of the project, time estimation is very difficult.

5.Time-Consuming: The Spiral Model can be time-consuming, as it requires multiple evaluations and reviews.

6. **Resource Intensive:** The Spiral Model can be resource-intensive, as it requires a significant investment in planning, risk analysis, and evaluations.

AGILE DEVELOPMENT MODELS

- ✓ The agile software development model was proposed in the mid-1990s to overcome the shortcomings of the waterfall model of development identified above.
- ✓ The agile model could help a project to adapt to change requests quickly.
- ✓ Thus, a major aim of the agile models is to facilitate quick project completion.
- ✓ Agile model is being used as an umbrella term to refer to a group of development processes.
- ✓ While these processes share certain common characteristics, yet they do have certain subtle differences among themselves.

A few popular agile SDLC models are the following:

- Crystal
- Atern (formerly DSDM)
- Feature-driven development
- Scrum
- Extreme programming (XP)
- Lean development
- Unified process

Fig. Agile Model

- In an agile model, the requirements are decomposed into many small parts that can be incrementally developed.
- The agile models adopt an incremental and iterative approach.
- Each incremental part is developed over an iteration.
- Each iteration is intended to be small and easily manageable and lasts for a couple of weeks only.
- At a time, only one increment is planned, developed, and then deployed at the customer site.
- No long-term plans are made.
- The time to complete an iteration is called a time box.
- The implication of the term time box is that the end date for an iteration does not change.

Agile Methodologies

- **Scrum**: Uses short, time-boxed sprints, with roles like **Product Owner**, **Scrum Master**, and **Development Team**. Scrum focuses on regular meetings (Daily Standups, Sprint Planning, etc.) to manage progress.
- **Kanban**: A visual management method using boards and limits on work-in-progress to optimize flow.
- **Extreme Programming (XP)**: Emphasizes best engineering practices, including pair programming and Test-Driven Development (TDD).
- **Lean Software Development**: Focuses on reducing waste and improving efficiency, borrowing principles from Lean manufacturing.
- **Feature-Driven Development (FDD)**: Focuses on delivering small, working features through iterative development.

Scrum

- Scrum is one of the agile development models.
- In the scrum model, the entire project work is divided into small work parts that can incrementally be developed and delivered over time boxes.
- These time boxes are called sprints.
- At the end of each sprint, the stakeholders and the team members meet to assess the developed software increment.
- The stakeholders may suggest any changes and improvements to the developed software that they might feel necessary.
- In scrum, the software gets developed over a series sprints.
- In each sprint, manageable increments to the software (or chunks) are deployed at the client site and the client feedback is obtained after each sprint.
- In the scrum model, the team members assume three basic roles: product owner, scrum master, and team member.

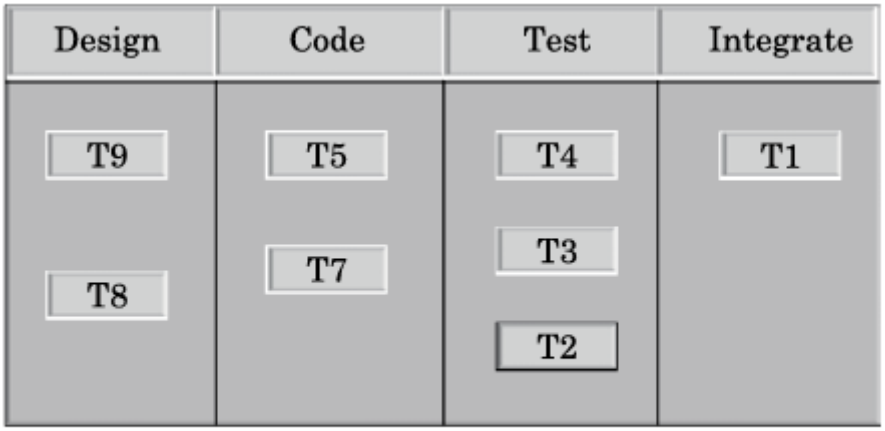


FIGURE 2.11 An example Kanban board.

Advantages & Disadvantages

Agile Methodology

Advantages

- Agile reduces risks and increases transparency
- Agile should deliver a product to market faster
- Agile promotes collaboration and creativity
- Agile can accommodate the unexpected
- QA and testing are carried out with each increment
- High efficiency

Disadvantages

- Agile has steep adaptation curve
- Unpredictability in early stages