Chapter 2

SOFTWARE-DEVELOPMENT LIFE-CYCLE MODELS

Achmad Benny Mutiara
amutiara@staff.gunadarma.ac.id

Content

2.1 SOFTWARE-DEVELOPMENT LIFE-CYCLE

- The software-development life-cycle is used to facilitate the development of a large software product in a systematic, well-defined, and cost-effective way.
- An information system goes through a series of phases from conception to implementation. This process is called the Software-Development Life-Cycle. Various reasons for using a life-cycle model include:
  - Helps to understand the entire process
  - Enforces a structured approach to development
  - Enables planning of resources in advance
  - Enables subsequent controls of them
  - Aids management to track progress of the system

- The software development life-cycle consists of several phases and these phases need to be identified along with defining the entry and exit criteria for every phase.
- A phase can begin only when the corresponding phase-entry criteria are satisfied.
- Similarly, a phase can be considered to be complete only when the corresponding exit criteria are satisfied. If there is no clear indication of the entry and exit for every phase, it becomes very difficult to track the progress of the project.

- The software development life-cycle can be divided into 5-9 phases, i.e., it must have a minimum of five phases and a maximum of nine phases. On average it has seven or eight phases. These are:
  - Project initiation and planning/Recognition of need/Preliminary investigation
  - Project identification and selection/Feasibility study
  - Project analysis
  - System design
  - Coding
  - Testing
  - Implementation
  - Maintenance

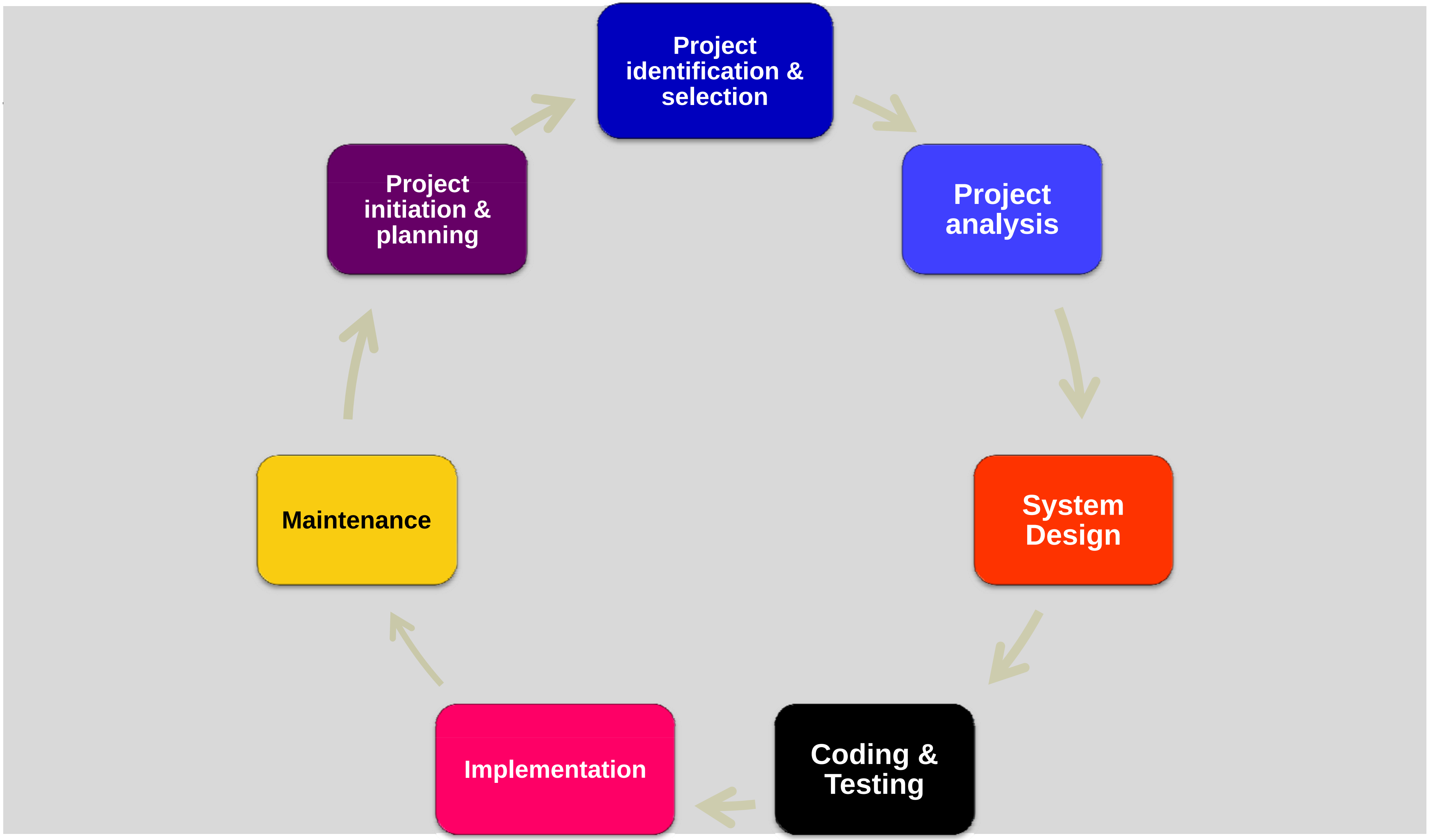

FIGURE 2.1 Software-Development Life-Cycle

1. Recognition of Need

- Recognition of need is nothing but **the problem definition**.
  - It is the decision about problems in the existing system and the impetus for system change.
- The first stage of any project or system-development life-cycle is called the **preliminary investigation**.
  - It is a brief investigation of the system under consideration.
  - This investigation provides the organization's computer steering committee and any project team a set of terms or references for more detailed work.
  - This is carried out by a senior manager and will result in **a study proposal.**
- At this stage the need for changes in the existing system are identified and shortcomings of the existing system are detected. These are stated clearly providing **the basis for the initial or feasibility study.**

2. Feasibility Study

- A feasibility study is **a preliminary study** which **investigates** the information needs of prospective users and **determines** the resource requirements, costs, benefits, and feasibility of a proposed project.
- **The goal of feasibility studies** is to evaluate alternative systems and to propose the most feasible and desirable systems for development.
- The feasibility of a proposed system **can be evaluated** in terms of four major categories, as illustrated in Table 2.1.

TABLE 2.1 Key Features of Categories of Feasibility

Organizational Feasibility	**Economic Feasibility**
How well the proposed system supports the strategic objectives of the organization	Cost savings Increased revenue Decreased investment Increased profits
Technical Feasibility	**Operational Feasibility**
Hardware, software, and network capability, reliability and availability	End-user acceptance Management support Customer, supplier, and government requirements

- ***Organizational Feasibility***. Organizational feasibility is how well a proposed information system supports the objectives of the organization and is a strategic plan for an information system. For example, projects that do not directly contribute to meeting an organization's strategic objectives are typically not funded.

- ***Economic Feasibility***. Economic feasibility is concerned with whether expected cost savings, increased revenue, increased profits, reductions in required investments, and other types of benefits will exceed the costs of developing and operating a proposed system. For example, if a project can't cover its development costs, it won't be approved, unless mandated by government regulations or other considerations.

- ***Technical Feasibility***. Technical feasibility can be demonstrated if reliable hardware and software capable of meeting the needs of a proposed system can be acquired or developed by the business in the required time.

- ***Operational Feasibility***. Operational feasibility is the willingness and ability of management, employees, customers, suppliers, and others to operate, use, and support a proposed system. For example, if the software for a new system is too difficult to use, employees may make too many errors and avoid using it. Thus, it would fail to show operational feasibility.

3. Project Analysis

- Project analysis is a detailed study of the various operations performed by a system and their relationships within and outside the system. Detailed investigation should be conducted with personnel closely involved with the area under investigation, according to the precise terms of reference arising out of the initial study reports.

- The tasks to be carried out should be clearly defined such as:
  - Examine and document the relevant aspects of the existing system, its shortcomings and problems.
  - Analyze the findings and record the results.
  - Define and document in an outline the proposed system.
  - Test the proposed design against the known facts.
  - Produce a detailed report to support the proposals.
  - Estimate the resources required to design and implement the system.

- **The objectives** at this stage are
  - to provide solutions to stated problems, usually in the form of specifications to meet the users requirements and
  - to make recommendations for a new computer-based system.
- Analysis is **an iterative and progressive process**, examining information flows and evaluating various alternative design solutions until a preferred solution is available.
- This is **documented as the system proposal**.

4. System Design

- System design is the most creative and challenging phase of the system-development life-cycle.
- The term design **describes** the final system and process by which it is developed.
- Different stages of the design phase are shown in Figure 2.2.
- This phase is **a very important phase** of the life-cycle.
- This is **a creative as well as a technical activity** including the following tasks:

- Appraising the terms of reference
- Appraising the analysis of the existing system, particularly problem areas
- Defining precisely the required system output
- Determining data required to produce the output
- Deciding the medium and opening the files
- Devising processing methods and using software to handle files and to produce output
- Determining methods of data capture and data input
- Designing the output forms
- Defining detailed critical procedures
- Calculating timings of processing and data movements
- Documenting all aspects of design

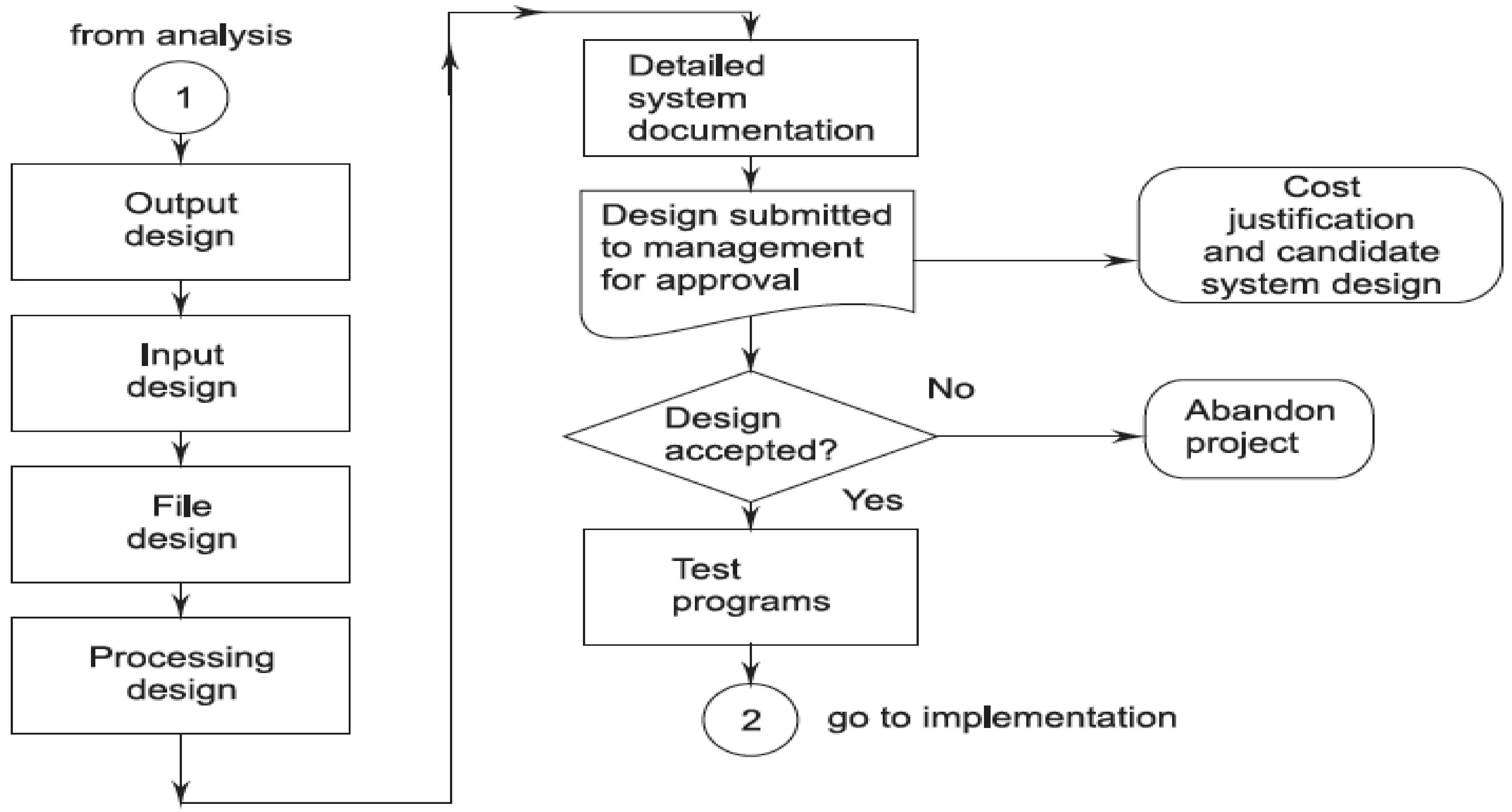

FIGURE 2.2 Cycle of Design Phase

5. Coding

- The goal of the coding phase is **to translate the design of the system into code** in a given programming language.
- In this phase **the aim is to implement the design** in the best possible manner.
- This phase affects both testing and maintenance phases. **Well-written code can reduce** the testing and maintenance effort.
- Hence, during coding the focus is on developing programs that are easy to read and understand and not simply on developing programs that are simple to write.

- Coding can be subject to company-wide standards that may define the entire layout of programs, such as headers for comments in every unit, naming conventions for variables, classes and functions, the maximum number of lines in each component, and other aspects of standardization.

- Structured programming helps the understandability of programs.
  - The goal of structured programming is to linearize the control flow in the program.
  - Single entry-single exit constructs should be used.
  - The constructs include selection (if-then-else) and iteration (while, repeat-unit).

6. Testing

- Testing is the major quality-control measure used during software development.
  - **Its basic function is to detect errors in the software**.
  - Thus, **the goal of testing is** to uncover requirement, design, and coding errors in the program.
- Testing is an extremely critical and time-consuming activity.
  - **It requires proper planning of the overall testing process**.
  - During the testing of the unit, the specified test cases are executed and the actual results are compared with the expected output.
- **The final output of the testing phase** is the **test report and the error report, or a set of such reports** (one for each unit tested).
  - Each test report contains the set of test cases and the result of executing the code with these test cases.
  - The error report describes the errors encountered and the action taken to remove the errors.

- Testing **cannot show** the absence of defects; it **can show only** **software errors present**.

- During the testing phase emphasis should be on the following:
  - Tests should be planned long before testing begins.
  - All tests should be traceable to customer requirements.
  - Tracing should begin "in the small" and progress toward testing "in the large."
  - For most effective testing, independent, third parties should conduct testing.

7. Implementation

- The implementation phase is less creative than system design.
  - It is mainly concerned with user training, site selection, and preparation and file conversion.

- Once the system has been designed, it is ready for implementation.
  - Implementation is concerned with those tasks leading immediately to a fully operational system.
  - It involves programmers, users, and operations management, but its planning and timing is a prime function of a systems analyst.
  - It includes the final testing of the complete system to user satisfaction, and supervision of initial operation of the system.
  - Implementation of the system also includes providing security to the system.

Types of Implementation

- There are **three types of implementation**:
  - Implementation of a computer system to replace a manual system.
  - Implementation of a new computer system to replace an existing one.
  - Implementation of a modified application (software) to replace an existing one using the same computer.

8. Maintenance

- Maintenance is an important part of the SDLC.
  - If there is any error to correct or change then it is done in the maintenance phase.
- Maintenance of software is also a very necessary aspect related to software development.
- Many times maintenance may consume more time than the time consumed in the development.
- Also, the cost of maintenance varies from 50% to 80% of the total development cost.

- Maintenance is not as rewarding or exciting as developing the systems. It may have problems such as:
  - Availability of only a few maintenance tools.
  - User may **not accept** the cost of maintenance.
  - Standards and guidelines of project may be poorly defined.
  - A good test plan is lacking.
  - Maintenance is viewed as a necessary evil often delegated to junior programmers.
  - Most programmers view maintenance as low-level drudgery.

Types of Maintenance

Maintenance may be classified as:

- ***Corrective Maintenance*.**
  - Corrective maintenance means repairing processing or performance failures or making changes because of previously uncorrected problems.
- ***Adaptive Maintenance*.**
  - Adaptive maintenance means changing the program function.
  - This is done to adapt to the external environment change.
    - **For example**, the current system was designed so that it calculates taxes on profits after deducting the dividend on equity shares. The government has issued orders now to include the dividend in the company profit for tax calculation. This function needs to be changed to adapt to the new system.

- ***Perfective Maintenance*.**
  - Perfective maintenance means enhancing the performance or modifying the programs to respond to the user's additional or changing needs.
    - **For example**, earlier data was sent from stores to headquarters on magnetic media but after the stores were electronically linked via leased lines, the software was enhanced to send data via leased lines.
  - As maintenance is very costly and very essential, efforts have been done to reduce its costs. One way to reduce the costs is through maintenance management and software modification audits. Software modification consists of program rewriting and system-level-upgrading.

- ***Preventive Maintenance*.**
  - Preventive maintenance is the process by which we prevent our system from being obsolete.
  - Preventive maintenance involves the concept of re-engineering and reverse engineering in which an old system with an old technology is re-engineered using new technology.
  - This maintenance prevents the system from dying out.

2.2 WATERFALL MODEL

- The waterfall model is a very common software development process model.
  - The waterfall model was popularized in the 1970s and permeates most current software-engineering textbooks and standard industrial practices.
  - Its first appearance in the literature dates back to the late 1950s as the result of experience gained in the development of a large air-defense software system called SAGE (Semi-Automated Ground Environment).
- The waterfall model is illustrated in Figure 2.3. **Because of the cascade from one phase to another, this model is known as the waterfall model or software life-cycle**.

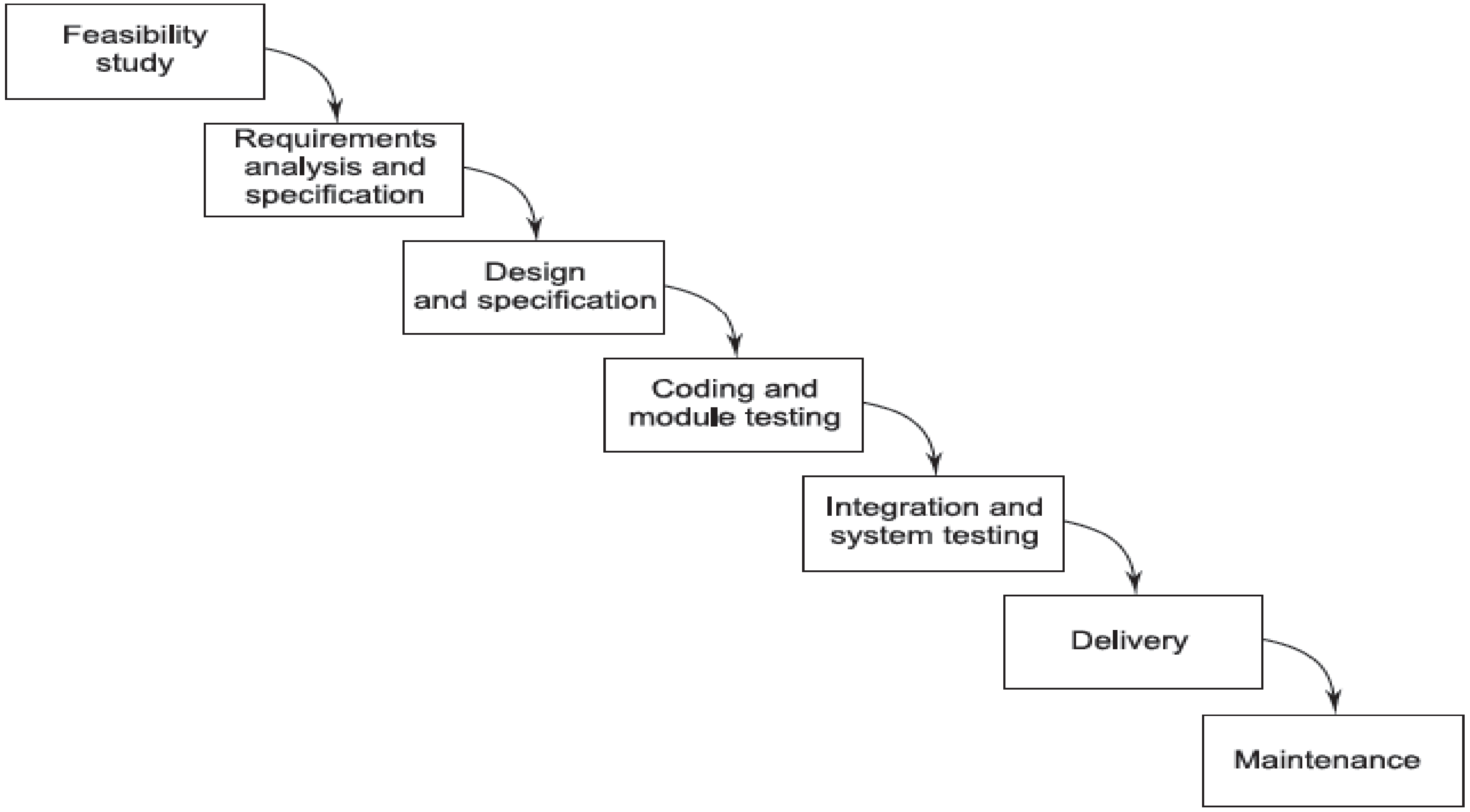
Feasibility study
Requirements analysis and specification
Design and specification
Coding and module testing
Integration and system testing
Delivery
Maintenance

FIGURE 2.3 Waterfall Model

1. Feasibility Study

- The first phase is the feasibility study.
  - The purpose of this phase is to produce a feasibility study document that evaluates the costs and benefits of the proposed application.
  - To do so, it is first necessary to analyze the problem, at least at a global level. Obviously, the more we understand the problem, the better we can identify alternative solutions, their costs, and their potential benefits to the user.
  - Therefore, ideally, one should perform as much analysis of the problem as is needed to do a well-founded feasibility study.
- The feasibility study is usually done within limited time bounds and under pressure.
  - Often, its result is an offer to the potential customer. Since we cannot be sure that the offer will be accepted, economic reasons prevent us from investing too many resources into analyzing the problem.

- As the figure shows, the process is structured as a cascade of phases, where the output of one phase constitutes the input to the next one.
  - Each phase, in turn, is structured as a set of activities that might be executed by different people concurrently.
- The phases shown in the figure are the following:
  - Feasibility study
  - Requirements analysis and specification
  - Design and specification
  - Coding and module testing
  - Integration and system testing
  - Delivery
  - Maintenance

- In sum, the feasibility study tries to anticipate future scenarios of software development. Its result is a document that should contain at least the following items:
  - A definition of the problem.
  - Determination of technical and economic viability.
  - Alternative solutions and their expected benefits.
  - Required resources, costs, and delivery dates in each proposed
  - alternative solution.

- At the end of this phase, *a report called a feasibility study is prepared by a group of software engineers*.
- The client or the customer is also consulted through a questionnaire.
- This report determines whether the project is feasible or not.
- After being successful in the feasibility study, the *requirement analysis* is carried out.

2. Requirement Analysis and Specification

- This phase exactly tells the requirements and needs of the project. This is a very important and critical phase in the waterfall model.
- **The purpose of a requirements analysis** is to identify the qualities required of the application, in terms of functionality, performance, ease of use, portability, and so on.
- The requirements describe the “what” of a system, not the “how.”
- This phase produces a large document and contains a description of what the system will do without describing how it will be done.
- The resultant document is known as ***the software requirement specification (SRS) document.***

- An SRS document must contain the following:
  - Detailed statement of problem.
  - Possible alternative solution to problem.
  - Functional requirements of the software system.
  - Constraints on the software system.
- The SRS document must be precise, consistent, and complete. There is no scope for any ambiguity or contradiction in the SRS document.
- An SRS document may be organized as a problem statement, introduction to the problem, functional requirements of the system, non-functional requirements of the system, behavioral descriptions, and validation criteria.

3. Design and Specification

- 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 available: the ***traditional design approach*** and the ***object-oriented design approach***.

(i) Traditional Design Approach

- The traditional design approach is currently being used by many software-development houses.

- Traditional design consists of two different activities:
  - first *a structured analysis* of the requirements specification is carried out where the detailed structure of the problem is examined.
  - This is followed by *a structured design activity*. During structured design, the results of structured analysis are transformed into the software design.

- 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.
  - During ***detailed design***, internals of the individual modules are designed in greater detail (e.g., the data structures and algorithms of the modules are designed and documented).

(ii) Object-Oriented Design Approach

- This is a new paradigm.
- Various objects in the system are identified.
- After the identification of objects, the relationships among them are also explored.
- The OOD approach has several benefits, such as lower development time and effort and better maintainability.

4. Coding and Module Testing

- Coding and module testing is the phase in which we actually write programs using a programming language.
  - It was the only recognized development phase in early development processes, but it is just one of several phases in a waterfall process.
  - The output of this phase is an implemented and tested collection of modules.
- Coding can be subject to company-wide standards, which may define the entire layout of programs, such as
  - the headers for comments in every unit, naming conventions for variables and sub-programs, the maximum number of lines in each component, and other aspects that the company deems worthy of standardization.

- Module testing is also often subject to company standards, including ***a precise definition of a test plan, the definition of testing criteria to be followed (e.g., black-box versus white-box, or a mixture of the two), the definition of completion criteria (when to stop testing), and the management of test cases. Debugging*** is a related activity performed in this phase.

- Module testing is the ***main quality-control activity*** that is carried out in this phase. Other such activities may include code inspections to check adherence to coding standards and, more generally, to check for a disciplined programming style, as well as checking of software qualities other than functional correctness (e.g., performance), although this is often better done at a later stage of coding.

5. Integration and System Testing

- During the integration and system testing phase, the modules are integrated in a planned manner. The different modules making up a software product are almost never integrated in one shot (can you guess the reason for this?).
- Integration is normally carried out incrementally over a number of steps.
  - During each integration step, the partially integrated system is tested and a set of previously planned modules are added to it.
- Finally, when all the modules have been successfully integrated and tested, system testing is carried out.

- The objective of system testing is to determine whether the software system performs per the requirements mentioned in the SRS document.
- This testing is known as system testing. A fully developed software product is system tested.
- The system testing is done in three phases: ***Alpha, Beta, and Acceptance Testing.***
  - ***Alpha Testing*** is conducted by the software-development team at the developer's site.
  - ***Beta Testing*** is performed by a group of friendly customers in the presence of the software-development team.
  - ***Acceptance Testing*** is performed by the customers themselves. If the software is successful in acceptance testing, the product is installed at the customer's site.

6. Delivery and Maintenance

- The delivery of software is often done in two stages.
  - In the first stage, the application is distributed among a selected group of customers prior to its official release. The purpose of this procedure is to perform a kind of controlled experiment to determine, on the basis of feedback from users, whether any changes are necessary prior to the official release.
  - In the second stage, the product is distributed to the customers.
- We define maintenance as the set of activities that are performed after the system is delivered to the customer.
- Basically, maintenance consists of correcting any remaining errors in the system (corrective maintenance), adapting the application to changes in the environment (adaptive maintenance), and improving, changing, or adding features and qualities to the application (perfective maintenance).

- Recall that the cost of maintenance is often more than **60%** of the total cost of software and
  - that about **20%** of maintenance costs may be attributed to **corrective and adaptive maintenance**,
  - while over **50%** is attributable to **perfective maintenance**.
- Based on this breakdown, we observed that evolution is probably a better term than maintenance, although the latter is used more widely.

2.2.1 Advantages of Waterfall Model

The various advantages of the waterfall model include:

- It is a linear model.
- It is a segmental model.
- It is systematic and sequential.
- It is a simple one.
- It has proper documentation

2.2.2 Disadvantages of Waterfall Model

The various disadvantages of the waterfall model include:

- It is difficult to define all requirements at the beginning of a project.
- This model is not suitable for accommodating any change.
- A working version of the system is not seen until late in the project’s life.
- It does not scale up well to large projects.
- It involves heavy documentation.
- We cannot go backward in the SDLC.
- There is no sample model for clearly realizing the customer’s needs.
- There is no risk analysis.
- If there is any mistake or error in any phase then we cannot make good software.
- It is a document-driven process that requires formal documents at the end ofeach phase.

2.3 PROTOTYPING MODEL

- It always happens that a customer defines a set of general objectives for software but does not identify detailed input, processing, or output requirements.
- In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human/machine interaction should take. In these and many other situations, a prototyping paradigm may offer the best approach.
- Prototyping begins with communication. The developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.
- A quick design then occurs. The quick design focuses on the representation of those aspects of the software that will be visible to the customer/user (e.g., input approaches and output formats).
- The quick design leads to the construction of a prototype.
- The prototype is evaluated by the customer/user to refine requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.

- Ideally, the prototype serves as a mechanism for identifying software requirements. If a working prototype is built, the developer attempts to use existing program fragments or applies tools that enable working programs to be generated quickly.

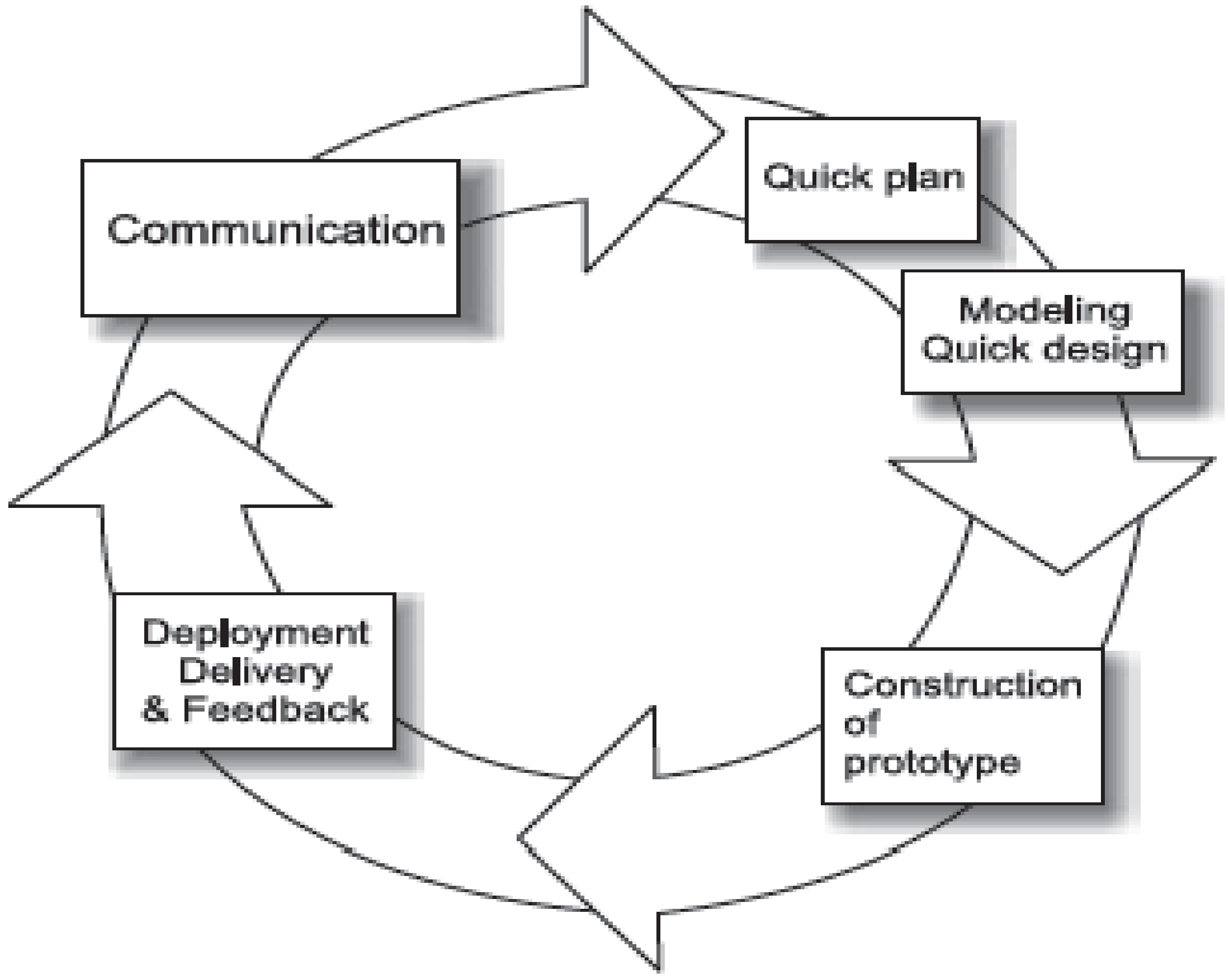

FIGURE 2.4 Prototyping Model

2.3.1 Reasons for Using Prototyping Model

There are several uses of a prototype.

1. An important purpose is **to illustrate** the input data formats, messages, reports, and the interactive dialogues to the customer.
   - This is valuable for gaining a better understanding of the customer’s needs. The prototype model is very useful in developing the Graphical User Interface (GUI) part of a system.

2. The prototyping model ***can be used when*** the technical solutions are ***unclear to*** the development team.
   - Often, major design decisions depend on issues, such as the response time of a hardware controller or the efficiency of a sorting algorithm, etc.
   - In such circumstances, a prototype may be the best or the only way to resolve the technical issues.

3. The third reason for developing a prototype is that ***it is impossible to “get it right” the first time and one must plan to throw away the first product in order to develop a good product later, as advocated.***

2.3.2 Controlling Changes During Prototyping

A major problem with prototyping is controlling changes to the prototype following suggestions by the users. One approach has been to categorize changes as belonging to one of three types:

- **Cosmetic (about 35% of changes)**

  These are simply changes to the layout of the screen. They are:

  a) *Implemented.*

  b) *Recorded.*

- **Local (about 60% of changes)**

  These involve changes to the way the screen is processed but do not affect other parts of the system. They are:

  a) *Implemented.*

  b) *Recorded.*

  c) *Backed-up so that they can be removed at a later stage is necessary.*

  d) *Inspected retrospectively.*

- **Global (about 5% of changes)**

  These are changes that affect more than one part of the processing. All changes here have to be the subject of a design review before they can be implemented.

2.3.3 Advantages of Prototyping Models

- Suitable for large systems for which there is no manual process to define the requirements.
- User training to use the system.
- User services determination.
- System training.
- Quality of software is good.
- Requirements are not freezed.

2.3.4 Limitations of Prototyping Model

1. It is difficult to find all the requirements of the software initially.
2. It is very difficult to predict how the system will work after development.

These two limitations ***are removed*** in the prototyping model.

- The first limitation ***is removed by unfreezing*** the requirements before any design or coding can proceed.
- The second limitation ***is removed by making a throw-away prototype*** to understand the requirements.

2.4 SPIRAL MODEL

- The spiral model, originally proposed by Boehm, is an evolutionary software model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear segmental model.
- The goal of the spiral model of the software production process is to provide a framework for designing such processes, guided by the risk levels in the projects at hand. As opposed to the previously presented models, the spiral model may be viewed as a meta-model, because it can accommodate any process-development model. By using it as a reference, one may choose the most appropriate development model (e.g., evolutionary versus waterfall). The guiding principle behind such a choice is the level of risk; accordingly, the spiral model provides a view of the production process that supports risk management.

- Let us present a few definitions. Risks are potentially adverse circumstances that may impair the development process and the quality of products.

- Boehm [1989] defines risk management as
  - ***a discipline whose objectives are to identify, address, and eliminate software risk items before they become either threats to successful software operation or a major source of expensive software rework.***

- ***The spiral model focuses*** *on* identifying and eliminating high-risk problems by careful process design, rather than treating both trivial and severe problems uniformly.

- The spiral model is recommended where the requirements and solutions call for developing full-fledged, large, complex systems with many features and facilities from scratch.
- It is used when experimenting on technology, trying out new skills, and when the user is not able to offer requirements in clear terms.
- It is also useful when the requirements are not clear and when the solution intended has multi-users, multi-functions, multi-features, multi-location applications to be used on multiple platforms, where seamless integration, interfacing, data migration, and replication are the issues.
- The radial dimension of a cycle represents the cumulative costs, and the angular dimension represents the progress made in completing each cycle of the spiral.

Each phase in this model is split into four sectors (or quadrants) as shown in Figure 2.5.

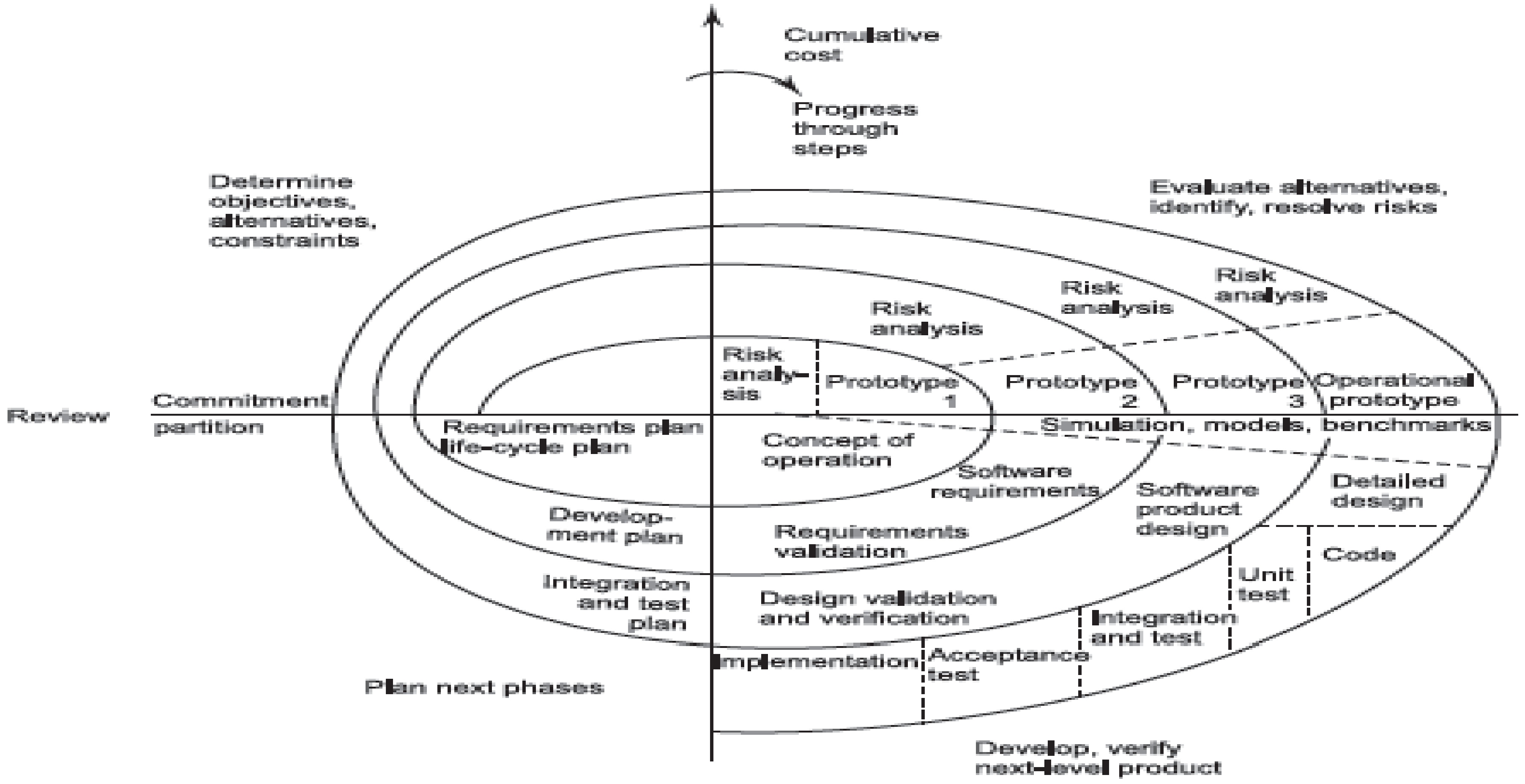

FIGURE 2.5 The Spiral Model (From Boehm [1988])

- The first quadrant identifies the objectives of the phase and the alternative solutions possible for the phase under consideration.
- In the second quadrant we evaluate different alternatives based on the objectives and constraints. The evaluation is based on the risk perceptions for the project.
- The third quadrant is the next step that emphasizes development of strategies that resolve the uncertainties and risks. This may involve activities, such as benchmarking, simulation, and prototyping.
- In the last or fourth quadrant we determine the objective that should be fulfilled in the next cycle of our software development in order to build the complete system.

2.4.1 Characteristics of Spiral Model

- The main characteristic of the Spiral Model is that it is cyclic and not linear like the waterfall model (see Figure 2.5).
- Each cycle of the spiral consists of four stages, and each stage is represented by one quadrant of the Cartesian diagram.
- The radius of the spiral represents the cost accumulated so far in the process; the angular dimension represents the progress in the process.

2.4.2 Limitations of Spiral Model

There are some limitations of the spiral model which include:

- No strict standards for software development.
- No particular beginning or end of a particular phase.

2.4.3 Advantages of Spiral Model

There are some advantages of the spiral model which include:

- It is risk-driven model.
- It is very flexible.
- Less documentation is needed.
- It uses prototyping.

2.4.4 Disadvantages of Spiral Model

- The spiral model has some disadvantages that need to be resolved before it can be a universally applied life-cycle model.
- The disadvantages include lack of explicit process guidance in determining objectives, constraints, alternatives, relying on risk assessment expertise, and providing more flexibility than required for many applications.

2.5 EVOLUTIONARY DEVELOPMENT MODEL

- In the Evolutionary Model, development engineering effort is made first to establish correct, precise requirement definitions and system scope, as agreed by all the users across the organization.
- This is achieved through application of iterative processes to evolve a system most suited to the given circumstances. The process is *iterative as the software engineer goes through a repetitive process of requirement* called Analysis-Design-Testing through Prototype-Implementation-Assessment-Evaluation until all users and stakeholders are satisfied.
- This model differs from the iterative enhancement model in the sense that this does not require a useable product at the end of each cycle. In evolutionary development, requirements are implemented by category rather than by priority.

2.5.1 Need of an Evolutionary Model

The various reasons why there exists a need for an evolutionary model include:

- Business and product requirements often change as development proceeds.
- Tight market deadlines make completion of a comprehensive software product impossible but a limited version must be introduced to meet competitive and business pressures.
- A set of core product or system requirements is well understood, but the details of product or system extensions have yet to be defined.

- This model is useful for projects using new technology that is ***not well*** understood.
- This is also used for complex projects where all functionality must be delivered at one time, but the requirements are ***unstable or not well understood at the beginning.***

2.6 ITERATIVE-ENHANCEMENT MODEL

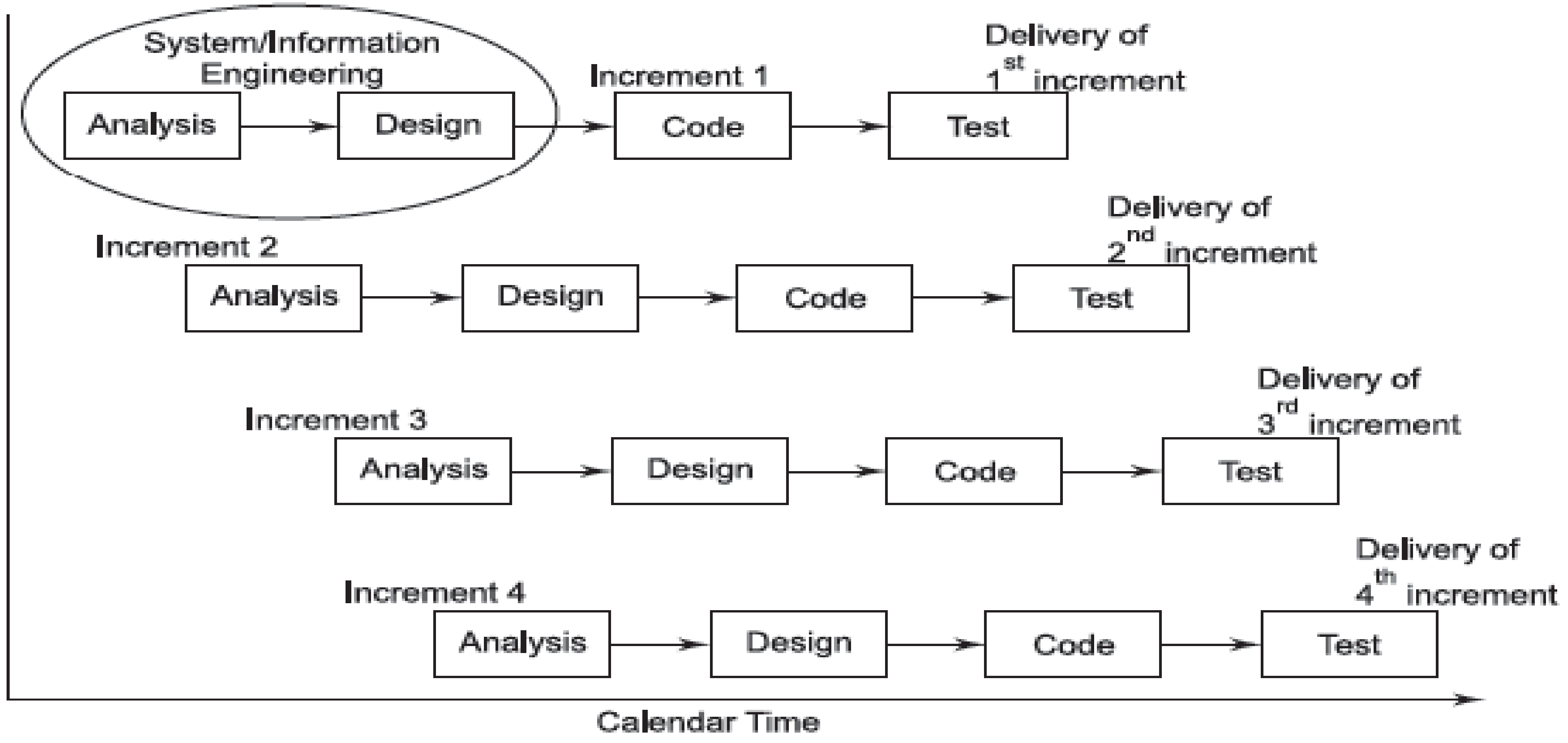

FIGURE 2.6 Iterative-Enhancement Model

- The iterative-enhancement model combines elements of the linear sequential model (applied repetitively) with the iterative philosophy of prototyping.
- In this model, the software is broken down into several modules, which are incrementally developed and delivered.
- First, the development team develops the core module of the system and then it is later refined into increasing levels of capability of adding new functionalities in successive versions.
- Each linear sequence produces a deliverable *increment of the software.*
  - *For* example, word-processing software developed using the iterative paradigm might deliver basis file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment. It should be noted that the process flow for any increment could incorporate the prototyping paradigm.

- When an iterative-enhancement model is used, the first increment is often a core product.
  - That is, basic requirements are addressed, but many supplementary features (some known, other unknown) remain undelivered.
- The core product is used by the customer (or undergoes detailed review).
  - As a result of use and/or evaluation, a plan is developed for the next increment.
  - The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality.
- This process is repeated following the delivery of each increment, until the complete product is produced.

2.6.1 Advantages of Iterative-Enhancement Model

The various advantages of following the approach of the iterative-enhancement model are as follows:

- The feedback from early increments improve the later stages.
- The possibility of changes in requirements is reduced because of the shorter time span between the design of a component and its delivery.
- Users get benefits earlier than with a conventional approach.
- Early delivery of some useful components improves cash flow, because you get some return on investment early on.
- Smaller sub-projects are easier to control and manage.
- ‘Gold-plating,’ that is the requesting of features that are unnecessary and not in fact used, is less as users will know that if a feature is not in the current increment then it can be included in the next.
- The project can be temporarily abandoned if more urgent work crops up.
- Job satisfaction is increased for developers who see their labors bearing fruit at regular, short intervals.

2.6.2 Disadvantages of Iterative-Enhancement Model

- The various disadvantages of the iterative enhance-ment model include:
- Software breakage, that is, later increments may require modifications to earlier increments.
- Programmers may be more productive working on one large system than on a series of smaller ones.
- Some problems are difficult to divide into functional units (modules), which can be incrementally developed and delivered.

2.7 RAD MODEL

- ***The RAD (Rapid Application Development Model) model*** is proposed when requirements and solutions can be modularized as independent system or software components, each of which can be developed by different teams.
  - After these smaller system components are developed, they are integrated to produce the large software system solution.
  - The modularization could be on a functional, technology, or architectural basis, such as front-end-back-end, client side-server side, and so on.
- RAD becomes faster if the software engineer uses the component's technology such that the components are really available for reuse. Since the development is distributed into component-development teams, the teams work in tandem and total development is completed in a short period (i.e., 60 to 90 days). Figure 2.7 shows the RAD model.

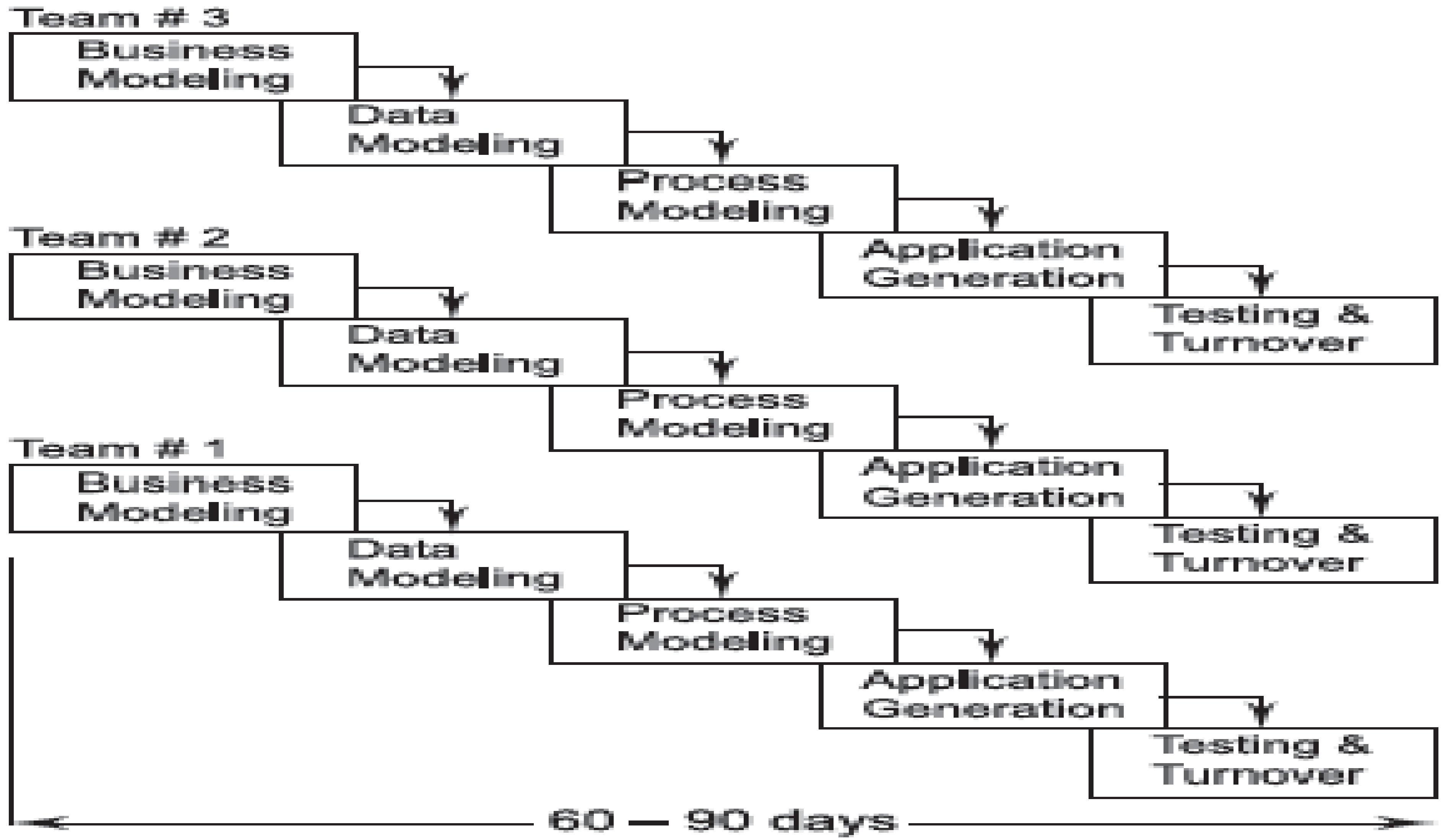

FIGURE 2.7 The RAD Model

1. **Business Modeling.** It covers the following functions:
   - Where does information come from and go? Who processes it?
   - What information drives the business process?
   - What information is generated?

2. **Data Modeling.** The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business. The characteristics of each object are identified and the relationships between these objects are defined.

3. **Process Modeling.** In this model, information flows from object to object to implement a business function. To add, modify, delete, or retain a data object, there is a need for description which is done in this phase.

4. **Application Generation.** RAD assumes the use of fourth-generation techniques. The RAD process works to reuse existing program components or create reusable components. To facilitate the construction of the software using the above cases, automated tools are used.

5. **Testing and Turnover.** In this phase we have to test the programs, but we use some already existing programs which are already tested, so the time involved in testing is less. Only the new programs or components must be tested.

2.7.1 Disadvantages of RAD Model

- The various disadvantages of the RAD model include:
- For large, but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
- If developers and customers are not committed to the rapid-fire activities necessary to complete the system in a much abbreviated timeframe, RAD projects will fail.
- If a system cannot be properly modularized, building the components necessary for RAD will be problematic.
- If high performance is an issue, and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work.
- RAD may not be appropriate when technical risks are high (for example, when a new application makes heavy use of new technology).

2.8 COMPARISON OF VARIOUS PROCESS MODELS

- The comparison between various process models is given in Table 2.2.

TABLE 2.2 Comparison of Process Models

Strengths	Weaknesses	Types of Projects
WATERFALL Simple Easy to execute Intuitive and logical	All or nothing approach Requirements frozen early Disallows changes Cycle time too long May choose outdated hardware technology User feedback not allowed Encourages requirement bloating	For well-understood problems Short duration projects Automation of existing manual systems
PROTOTYPING Helps in requirements elicitation Reduces risk Leads to a better system	Front heavy process Possibly higher costs Disallows later changes	Systems with novice users When there are uncertainties in requirements
ITERATIVE ENHANCEMENT Regular/quick deliveries Reduces risk Accommodates changes Allows user feedback Allows reasonable exit points Avoids requirement bloating Prioritizes requirements	Each iteration can have planning overhead Costs may increase as work done in one iteration may have to be undone later System architecture and structure may suffer as frequent changes are made	For businesses where time is of essence Where risk of a long project cannot be taken Where requirements are not known
SPIRAL Controls project risks Very flexible Less documentation needed	No strict standards for software development No particular beginning or end of particular phase	Projects built on untested assumptions

Tanya Jawab

The End