Learning Resource

On

Software Engineering

Chapter-2
Software Development Life Cycle

Prepared By:
Kunal Anand, Asst. Professor
SoCE, KIIT, DU, Bhubaneswar-24

Chapter Outcomes:

After completing this chapter successfully, the students will be able to:

- Discuss the need of Software Development Life Cycle (SDLC) model
- Identify phase Entry and Exit Criteria for each phase in SDLC
- List and describe different SDLC models
- Compare and contrast several SDLC models
- Illustratre advanced SDLC models like RAD, V-Model, and Agile.

Organization of the Chapter:

- Introduction to Software Development Life Cycle (SDLC)
- Need of SDLC Model
- Phase Entry and Exit Criteria
- Software Development Life Cycle Models
- Comparison of different SDLC models
- Rapid Application Development model
- V-Model
- Agile Methodology

Software Development Life Cycle (SDLC)

- A software life cycle model (also called process model) is a descriptive and diagrammatic representation of the life cycle of a software product i.e., from the date of inception of idea till the last useful day of the product.

- It represents all the activities required to make a software product transit through its life cycle.

- It also captures the order in which these activities are to be undertaken.

- In other words, it maps the different activities performed on a software product from its inception to retirement.

Activities in SDLC

Fig. 2.1: Software Development Life Cycle

Need for a SDLC model

- Development team must identify a suitable life cycle model for the project and then adhere to it.

- Without using of a particular life cycle model, the development of a software product would not be in a systematic and disciplined manner.

- When a software product is being developed by a team there must be a clear understanding among team members about when and what to do.
  - Otherwise, it would lead to chaos, exceeding of budget and project failure.

Entry and Exit Criteria

- One of the major problems that arises in almost any software development project is "99% Complete" syndrome.
- The solution to this syndrome is to ensure a proper mechanism to track the progress of the development work.
  - This can be achieved by defining tangible and measurable entry and exit criteria for each phase.
- A phase can start only if its phase-entry criteria have been satisfied. Similarly, a phase can be considered as complete only if its phase-exit criteria have been satisfied.
- So, without SDLC model the entry and exit criteria for a phase cannot be recognized.
- Without SDLC models, it becomes difficult for software project managers to monitor the progress of the project.

SDLC Models

- Some of the popular life cycle models have been proposed so far.
  - Classical Waterfall Model
  - Iterative Waterfall Model
  - Prototyping Model
  - Evolutionary Model
  - Spiral Model

Classical Waterfall Model

Classical waterfall model divides the software life cycle into following phases:

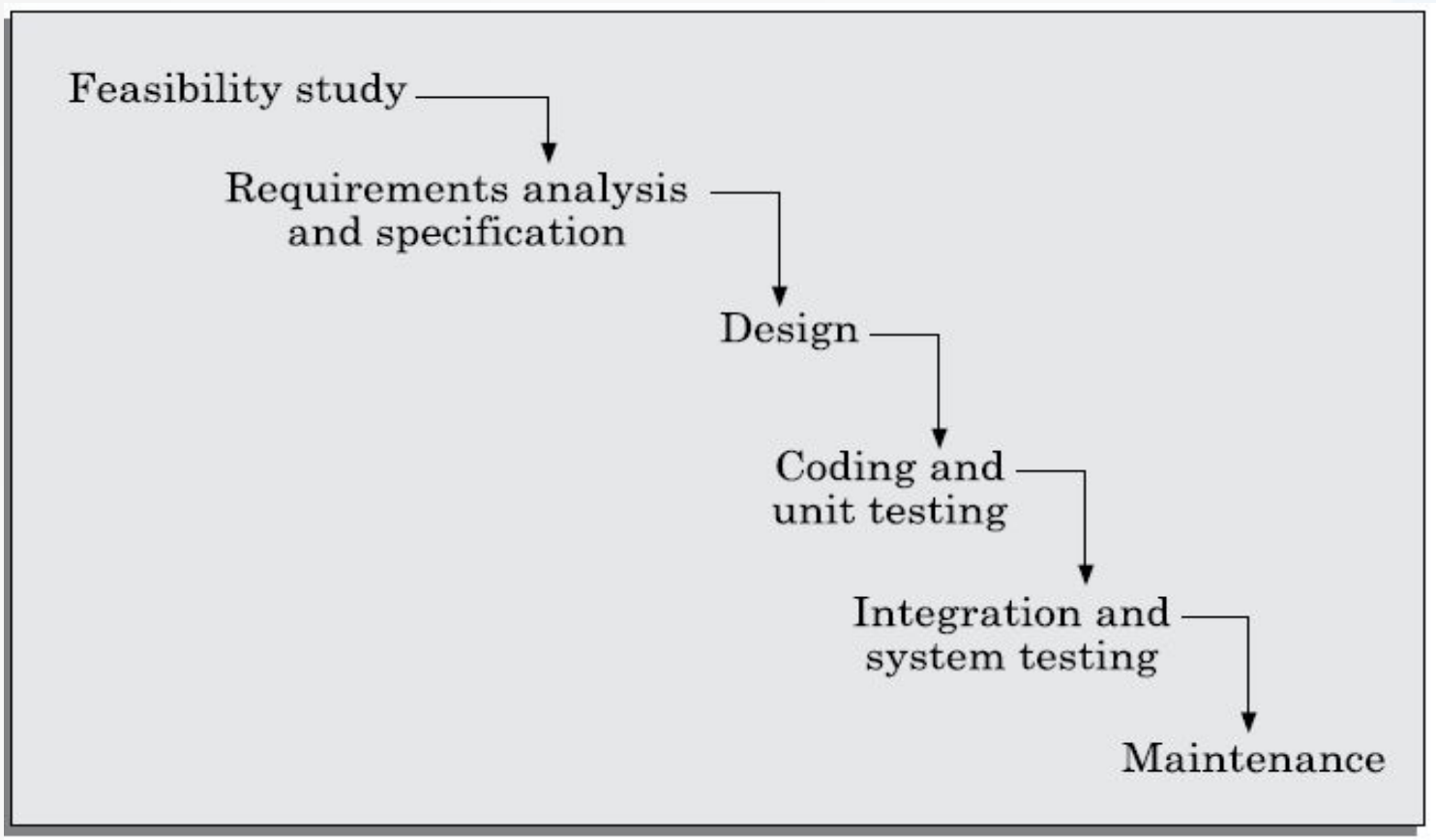


Fig. 2.2: Classical Waterfall model

Relative Effort for Phases

- Phases between feasibility study and testing known as development phases.
- Among all life cycle phases maintenance phase consumes maximum effort.
- Among development phases, testing phase consumes the maximum effort.

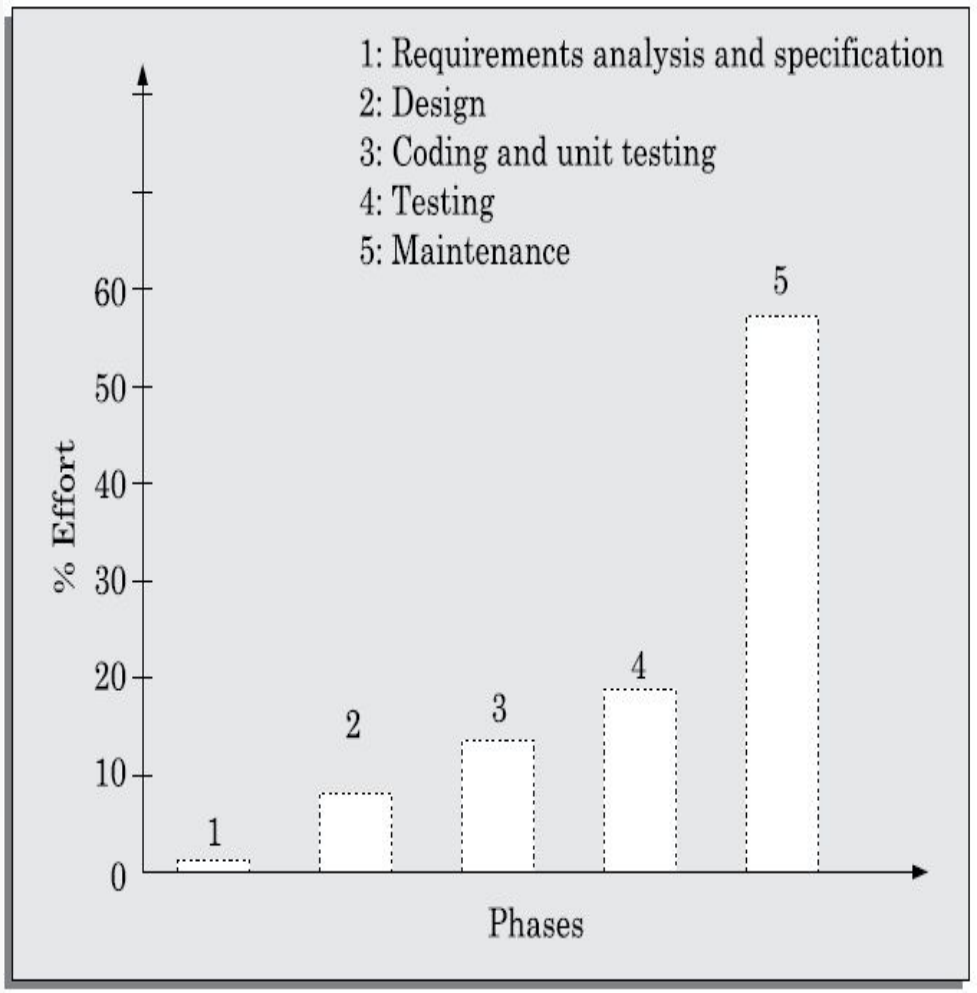


Fig. 2.3: Relative efforts for phases

Feasibility Study

- Main aim of feasibility study: determine whether developing the product will be:
  - Financially worthwhile
  - Technically feasible
  - Operational feasibility
  - Legal feasibility
  - Scheduling

Requirements Analysis and Specification

- Aim of this phase:
  - To understand the exact requirements of the customer,
  - To document them in proper and formal manner.

- Consists of two distinct activities:
  - Requirements Gathering and Analysis
  - Requirements Specification

Requirements Gathering

- **One-on-One Interviews:** These can be users who interact with the current or new system, management, project financers or anyone else that would be involved in the system. Contains open and close ended questions.
- **Group Interviews:** Important to note which issues are generally agreed upon, and on which issues it differs.
- **Questionnaires/Surveys:** This is especially helpful when stakeholders are spread out geographically.
- **User Observation:** For optimal results, the consultant should schedule three different periods of observation: low, normal, and peak times. This may prove helpful in because the user may interact with the system differently during different times.
- **Analysing Existing Documents:** Reviewing the current process and documentation can help the analyst understand the business, or system, and its current situation.

Requirements Analysis

- The data you initially collect from the users would usually contain several contradictions, incompleteness and ambiguities. These are known as requirement anomalies.

- Each user typically has only a partial and incomplete view of the system.

- The requirement anomalies must be identified and resolved by making detailed discussions with the customers.

- Engineers doing requirements analysis and specification are designated as an Analyst.

Software Requirement Specification

- Once the requirement analysis is over, the requirements are formally organized into Software Requirements Specification (SRS) document.

- The SRS document is sent to the client for their approval. The client goes through the SRS and approve it if they feel satisfied with the work. However, in case of any modifications or suggestions, the client sends back the SRS to the development team.

- The SRS document holds significance as it can be used as legal document in the court of law, in case of any dispute that may arise between the development team and client.

Software Design

- Design phase transforms the requirements specification into a form that is suitable for implementation using an appropriate programming language.

- In technical terms, Software Architecture is derived from the SRS document in design phase.

- Two design approaches:
  a) Traditional design approach,
  b) Object Oriented design approach.

Design: Traditional Approach

- Consists of two activities:
  - Structured Analysis
  - Structured Design

- **Structured Analysis**
  - Identify all the functions to be performed.
  - Identify data flow among the functions.
  - Decompose each function recursively into sub-functions and identify data flow among the subfunctions as well.
  - Carried out using Data flow diagrams (DFDs).

contd..

- **Structured Design:** After structured analysis, carry out structured design: architectural design (or high-level design), and detailed design (or low-level design).
  - **High-level design:**
    - decompose the system into *modules*,
    - represent invocation relationships among the modules.
  - **Detailed design:**
    - different modules designed in greater detail
      - data structures and algorithms for each module are designed.

Design: Object Oriented Approach

- First identify various objects (real world entities) occurring in the problem:
  - identify the relationships among the objects.
  - For example, the objects in a pay-roll software may be:
    - employees,
    - managers,
    - pay-roll register,
    - Departments, etc.
- Object structure is further refined to obtain the detailed design.
- OOD has several advantages:
  - lower development effort,
  - lower development time,
  - better maintainability.

Implementation

- Purpose of implementation phase (i.e., coding and unit testing phase) is to translate software design into source code.
- During the implementation phase:
  - each module of the design is coded.
  - each module is tested independently as a standalone unit and debugged.
  - each module is documented.
- The purpose of unit testing is to check if the individual modules work correctly.
- The end product of implementation phase is a set of program modules that have been tested individually.

Integration and System Testing

- Different modules are integrated in a planned manner:
  - modules are almost never integrated in one shot.
  - Normally integration is carried out through several steps.
- During each integration step, the partially integrated system is tested.
- After all the modules have been successfully integrated and tested, **system testing** is carried out.
- System testing ensures that the developed system functions according to its requirements as specified in the SRS document.
- Following types of System Testing exist: Alpha Testing, Beta Testing, Acceptance Testing

Maintenance

- **Maintenance of any software product**
  - requires much more effort than the effort to develop the product itself.
  - development effort to maintenance effort is typically 40:60.
- **Corrective maintenance**
  - Correct errors which were not discovered during the product development phases.
- **Perfective maintenance**
  - Improve implementation of the system
  - enhance functionalities of the system.
- **Adaptive maintenance**
  - Port software to a new environment, e.g., to a new computer or to a new operating system.

Limitations of Classical Waterfall Model

- **No Error Detection and Correction Mechanism**
  - The classical waterfall model assumes that everything will happen as per the expectations and no errors will happen during the development process. This assumption is flawed as in practice errors are bound to happen in almost every phases.

- **Poor Resource Utilization**
  - The classical waterfall model follows a rigid sequence of events due to which a phase can not be started if the previous phase is not complete.
  - It leads to poor utilization of resources as many team members will be sitting idle and wait for the other team to complete their work so that they can start.

Iterative Waterfall Model

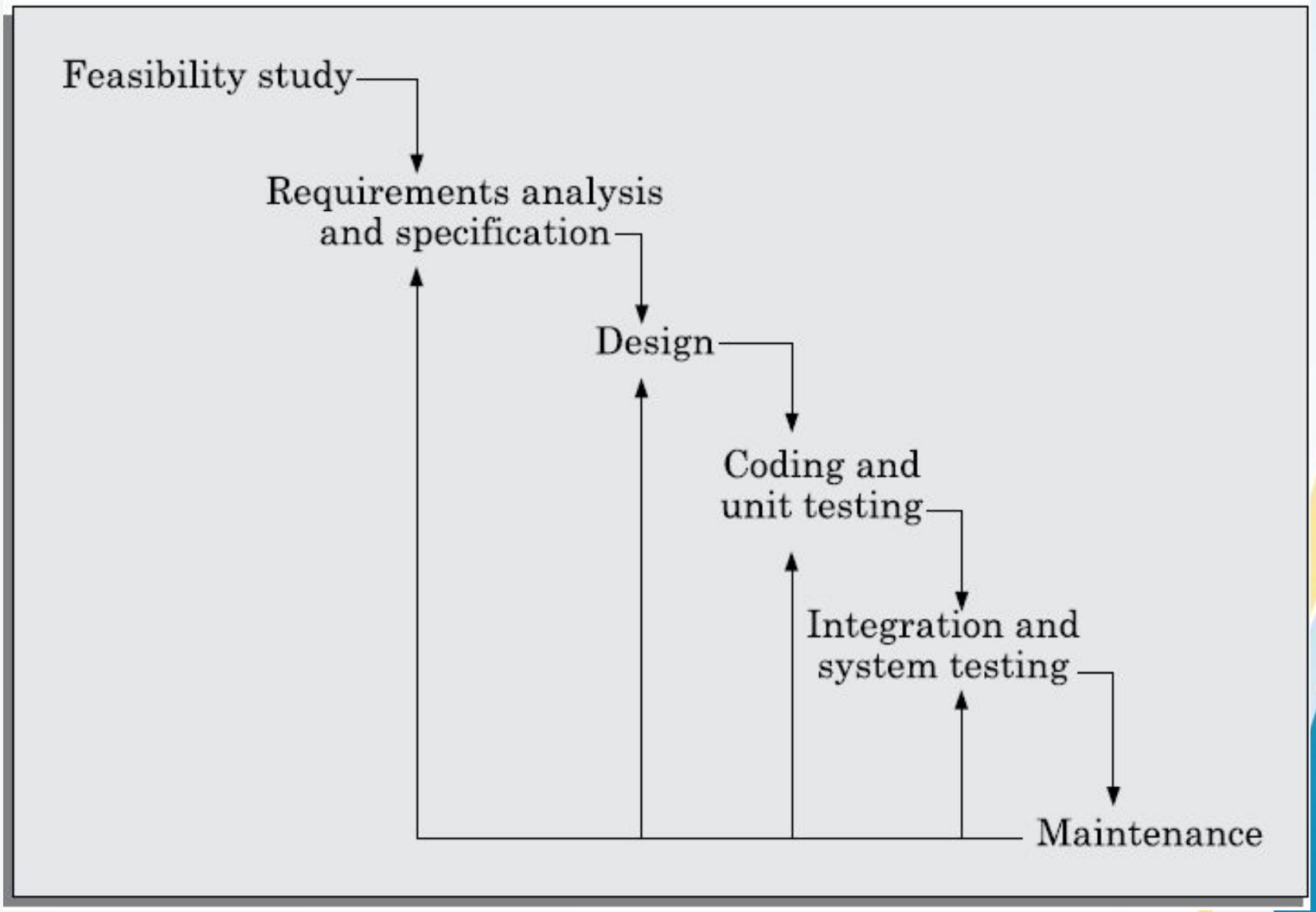


Fig. 2.4: Iterative waterfall model

Iterative Waterfall Model

- Errors should be detected in the same phase in which they are introduced.
  - **For example:** if a design problem is detected in the design phase itself, the problem can be taken care of much more easily than say if it is identified at the end of the integration and system testing phase.
  - **Reason:** rework must be carried out not only to the design but also to code and test phases.
- The principle of detecting errors as close to its point of introduction as possible is known as phase containment of errors.
- Iterative waterfall model is by far the most widely used model. Irrespective of the life cycle model followed, the documents should reflect a classical waterfall model of development.

Prototyping Model

- Before starting actual development, a working prototype of the system should first be built.
- A prototype is a fully functional but toy implementation of a system with low reliability, inefficient performance.

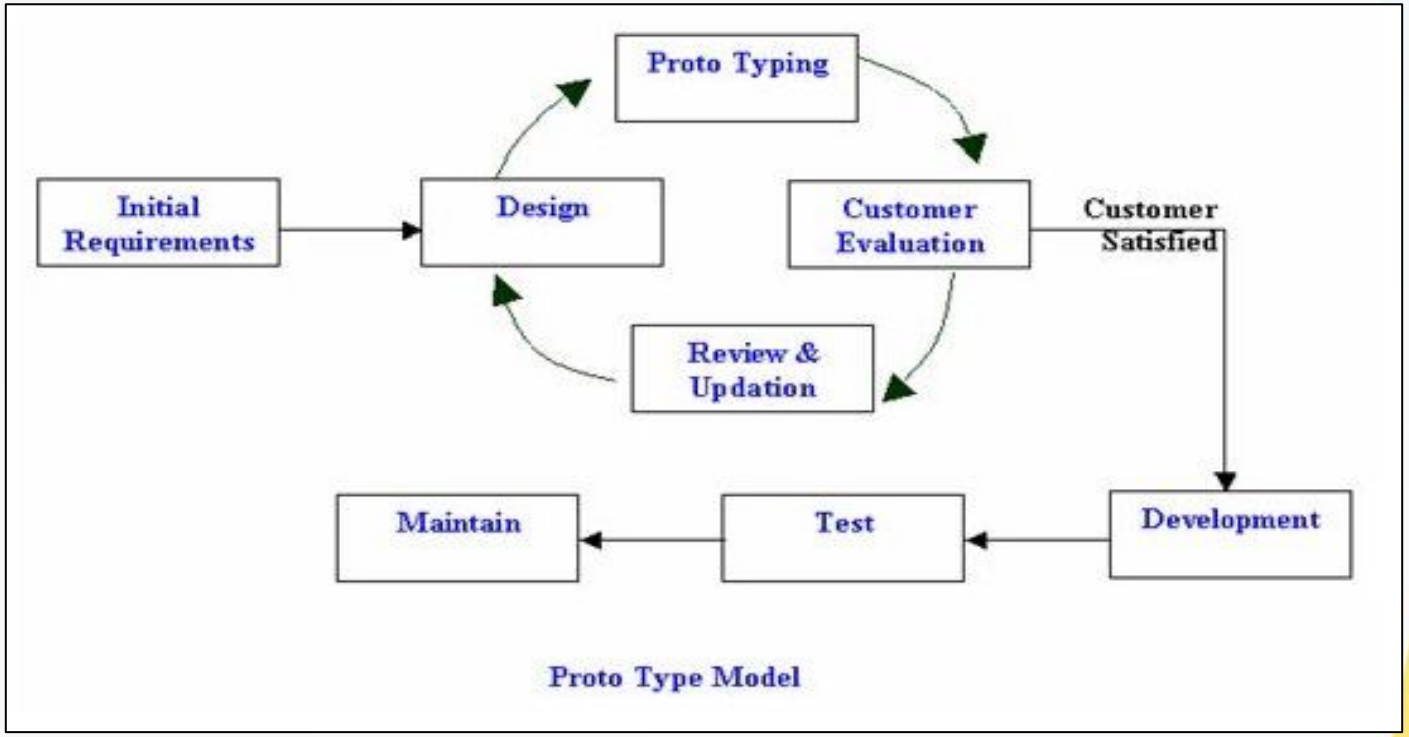


Fig. 2.5: Prototyping model

Reasons for developing a prototype

1) **Illustrate to the customer:**
   - input data formats, messages, reports, or interactive dialogs.

2) **Examine technical issues** associated with product development:
   - Often major design decisions depend on issues like:
     - response time of a hardware controller,
     - efficiency of a sorting algorithm, etc.

3)The third reason for developing a prototype is:
   - it is impossible to **"get it right"** the first time,
   - we must plan to throw away the first product
   - if we want to develop a good product.

Prototyping Model (contd..)

- Start with approximate requirements.
- Carry out a quick design.
- Prototype model is built using several short-cuts that involve using inefficient, inaccurate, or dummy functions
  - A function may use a table look-up rather than performing the actual computations.
- The developed prototype is submitted to the customer for evaluation. Based on the user feedback, requirements are refined.
- This cycle continues until the user approves the prototype.
- Once, prototype is approved, the actual system is developed using the iterative waterfall approach.

Prototyping Model (CONT.)

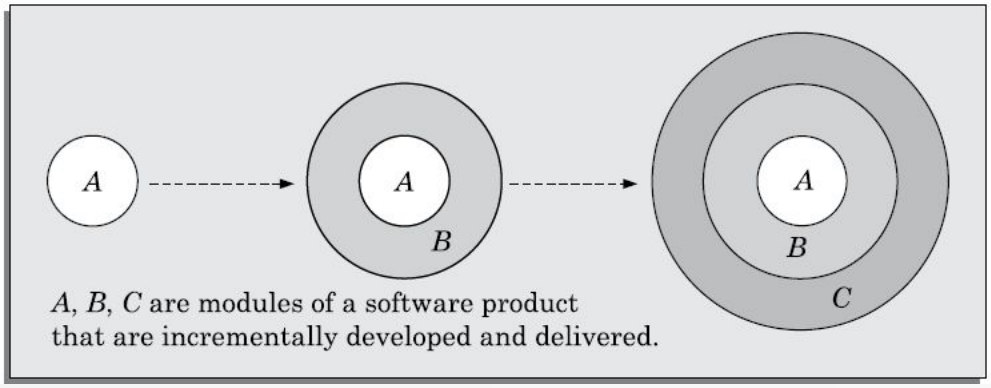
- **Advantages**
  - Even though construction of a working prototype model involves additional cost --- overall development cost might be lower for systems with unclear user requirements, and systems with unresolved technical issues.
  - Many user requirements, that may appear later as change requests, get properly defined and technical issues get resolved.

- **Disadvantages**
  - Requirements analysis and specification phase becomes redundant.
  - Design and code for the prototype is usually thrown away.

Incremental Model

- In Incremental model, the overall system is broken down into several modules which can be incrementally developed and delivered.
- First develop the core modules of the system.
- The initial product skeleton is refined into increasing levels of capability by adding new functionalities in successive versions.
- Successive version of the product:
  - functioning systems capable of performing some new work.
  - A new release may include new functionality:
    - also, existing functionality in the current release might have been enhanced.

Incremental Model (CONT.)



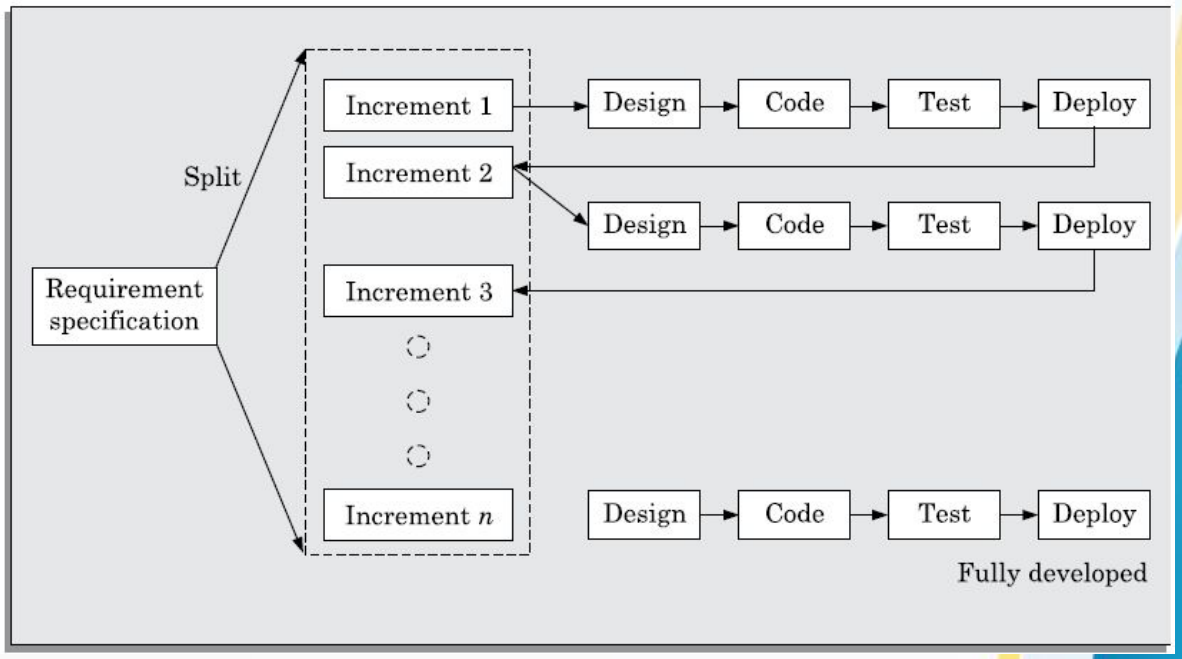


Fig. 2.6: Incremental model of software development

Advantages & Disadvantages of Incremental Model

- **Advantages:**
  - Users get a chance to experiment with a partially developed system much before the full working version is released.
  - Helps finding exact user requirements much before fully working system is developed.
  - Core modules get tested thoroughly that reduces chances of errors in final product.

- **Disadvantages:**
  - Often, difficult to subdivide problems into functional units which can be incrementally implemented and delivered.
  - evolutionary model is useful for very large problems, where it is easier to find modules for incremental implementation.

Evolutionary Model

- Many organizations use a combination of iterative and incremental development. This is known as Evolutionary Model.
- In the incremental development model, complete requirements are first the SRS document prepared and then the development begins in increments. In contrast, 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.
- Due to obvious reasons, the evolutionary software development process is sometimes referred to as *design a little, build a little, test a little, deploy a little model*.

contd..

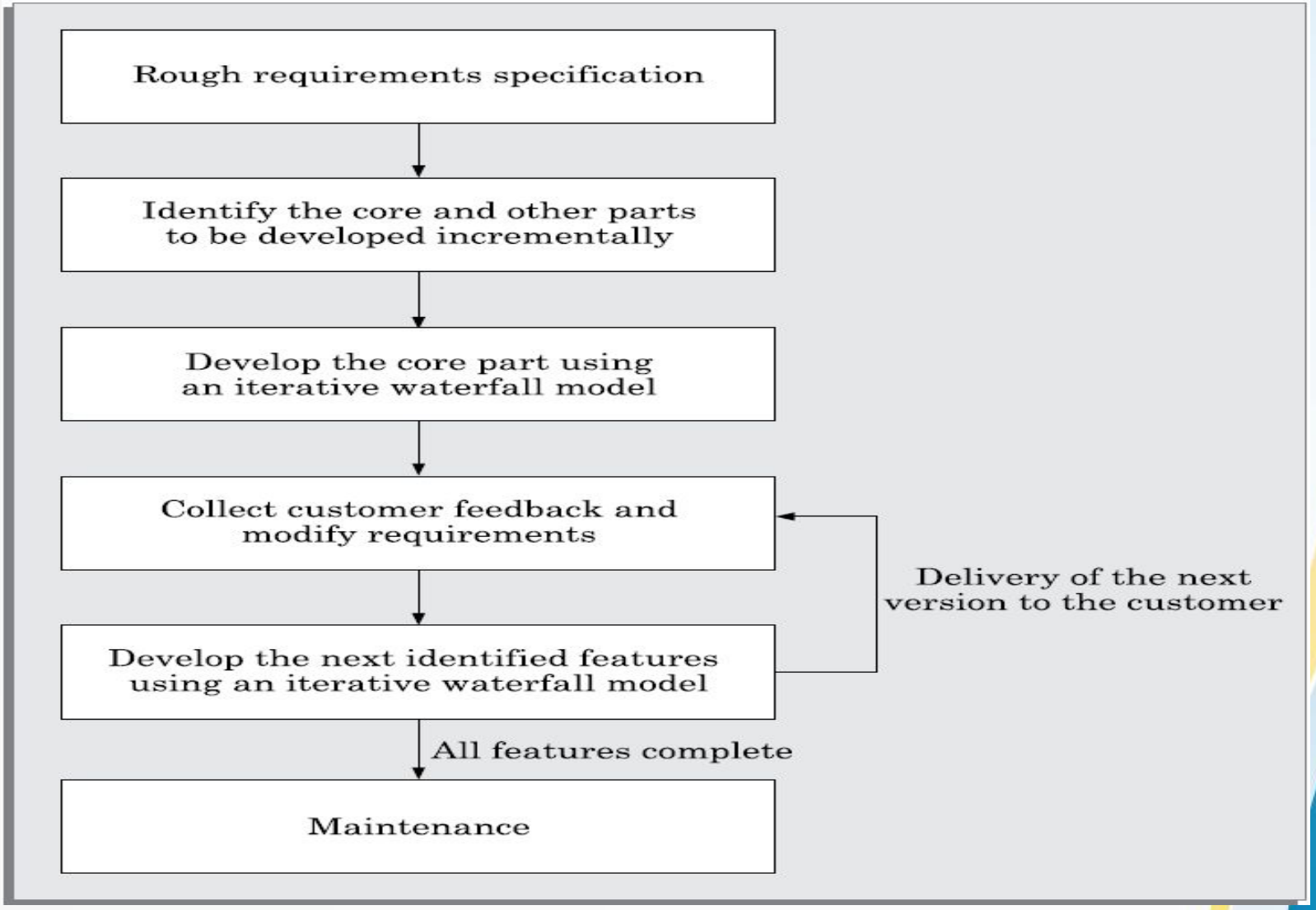


Fig. 2.7: Evolutionary model

Advantages & Disadvantages of Evolutionary Model

- **Advantages:**
  - Effective elicitation of actual customer requirements
  - Easy handling change requests

- **Disadvantages:**
  - Feature division into incremental parts can be non-trivial
  - Ad hoc design

Spiral Model

- It is one of the important life cycle model which gives a significant advantage of risk management.
- Spiral model consists of loops where each loop represents an iteration. The radius of the loop represents the cost involved in the development of that phase.
- The number of loops depends on the projects, and it varies as per the case.
- Each loop consists of 4 quadrants where each quadrant represents a particular activity in that iteration.

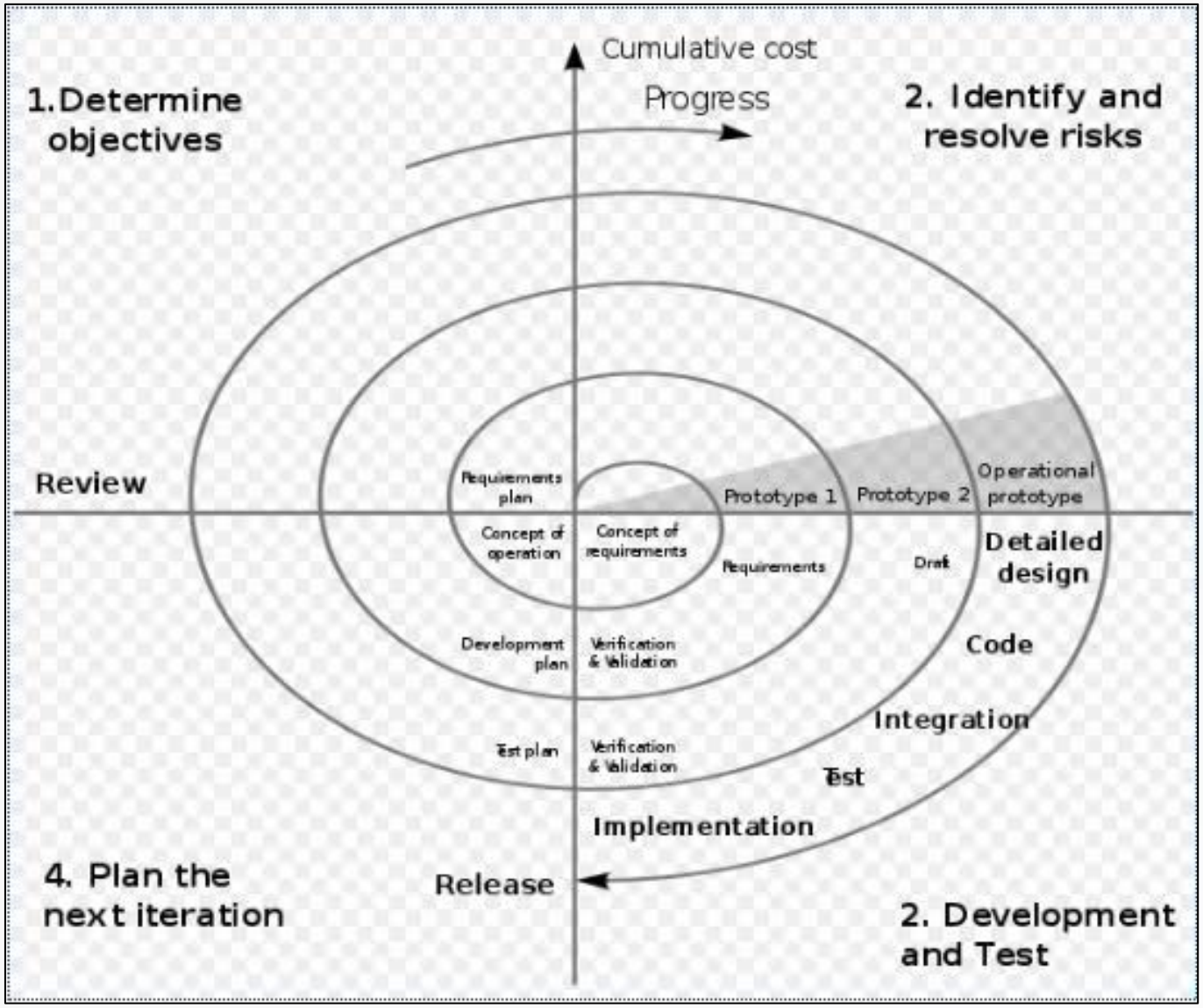


Fig. 2.8: Spiral model

Objective Setting (First Quadrant)

- Identify objectives of the phase,
- Examine the risks associated with these objectives and alternate possible solutions.
  - Risk is any adverse circumstance that might hamper successful completion of a software project.

Risk Assessment and Reduction (Second Quadrant)

- For each identified project risk, a detailed analysis is carried out.
- Steps are taken to reduce the risk.
  - For example, if there is a risk that the requirements are inappropriate: a prototype system may be developed.

Spiral Model (CONT.)

- **<u>Development and Validation</u> (Third quadrant):**
  - develop and validate the next level of the product.

- **<u>Review and Planning</u> (Fourth quadrant):**
  - review the results achieved so far with the customer and plan the next iteration around the spiral.

- With each iteration around the spiral:
  - progressively more complete version of the software gets built.

Spiral Model as a Meta model

- **Subsumes all discussed models:**
  - a single loop spiral represents waterfall model.
  - uses an evolutionary approach --
    - iterations through the spiral are evolutionary levels.
  - enables understanding and reacting to risks during each iteration along the spiral.
  - uses:
    - prototyping as a risk reduction mechanism
    - retains the step-wise approach of the waterfall model.

Advantages of Spiral Model

- High amount of risk analysis hence, avoidance of risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added later.

Disadvantages

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

Circumstances to use Spiral Model

- The spiral model is called a meta model since it encompasses all other life cycle models.
- Risk handling is inherently built into this model.
- The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks.
- However, this model is much more complex than the other models – this is probably a factor deterring its use in ordinary projects.

Comparison of Different Life Cycle Models

- **Iterative waterfall model**
  - most widely used model. But, suitable only for well-understood problems.
- **Prototype model**
  - suitable for projects not well understood user requirements or technical aspects
- **Evolutionary model**
  - suitable for large problems that can be decomposed into a set of modules that can be incrementally implemented,
  - incremental delivery of the system is acceptable to the customer.
- **Spiral model:**
  - suitable for development of technically challenging software products that are subject to several kinds of risks.

Rapid Application Development (RAD) Model

- It is a type of **incremental model**.
- In RAD model, the components or functions are developed in parallel as if they were multiple mini projects.
- The developments are time boxed, delivered and then assembled into a working prototype.
- This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements.

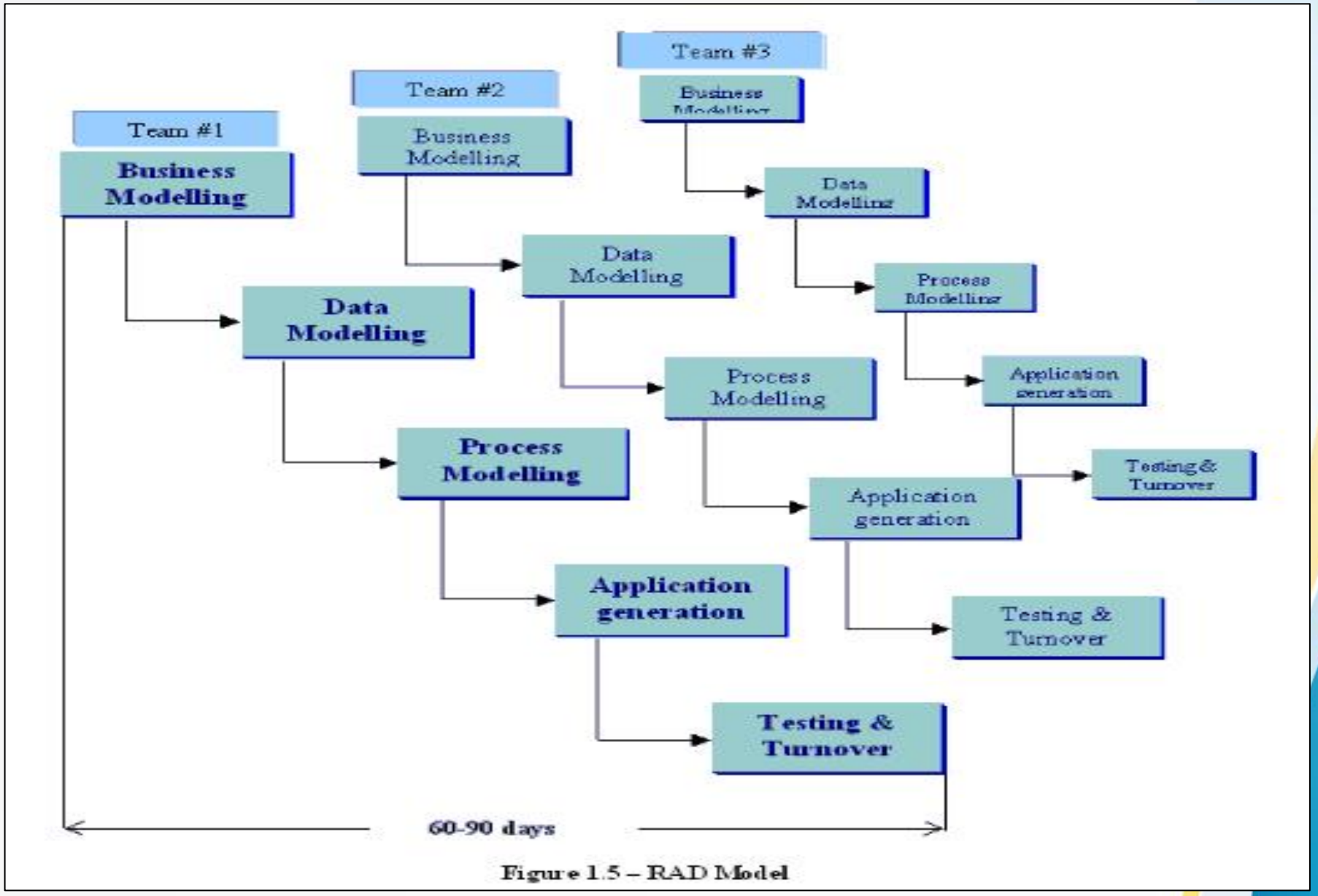


Figure 1.5 – RAD Model

Fig. 2.9: RAD model

- The phases in the rapid application development (RAD) model are:
  - **Business Modelling:** In this phase of development business model should be designed based on the information available from different business activities.
  - **Data Modelling:** Once the business modelling phase over and all the business analysis completed, all the required and necessary data based on business analysis are identified in data modelling phase.
  - **Process Modelling:** All the data identified in data modelling phase are planned to process or implement the identified data to achieve the business functionality flow. In this phase all the data modification process is defined.
  - **Application Modelling:** In this phase application is developed and coding completed. With help of automation tools all data implemented and processed to work as real time.
  - **Testing and turnover:** All the testing activities are performed to test the developed application.

- **Advantages of RAD Model:**
  - Fast application development and delivery.
  - Visualization of progress.
  - Less resources required.
  - Review by the client from the very beginning of development so very less chance to miss the requirements.
  - Very flexible if any changes required.
  - Cost effective.
  - Good for small projects.

- **Disadvantages of RAD Model:**
  - High skilled resources required.
  - On each development phase client's feedback required.
  - Automated code generation is very costly.
  - Difficult to manage.
  - Not a good process for long term and big projects.
  - Proper modularization of project required.

V-Shaped Model

- It is an extension for waterfall model, Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape.
- The major difference between v-shaped model and waterfall model is the early test planning in v-shaped model.

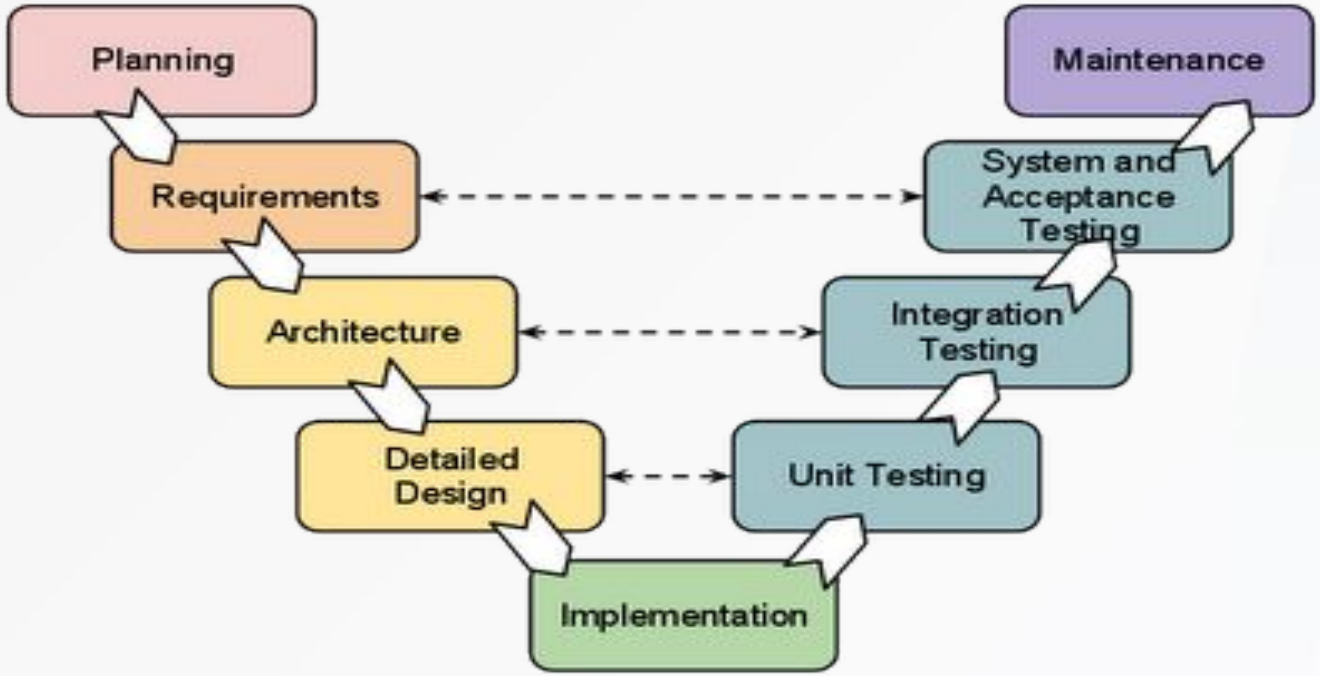


Fig. 2.10: V model

V-Model

- However, it allows to perform some testing related activities in parallel, with a corresponding phase of development.

- The usage
  - Software requirements clearly defined and known
  - Software development technologies and tools is well-known

- verification refers to the fact that **"Are we building things right?"** and validation refers to the fact that **"Are we building right thing?".**

contd..

- Here, in accordance with the requirement analysis and specification phase, a system test plan is created to meet the functionality specified in the requirements.

- The architecture phase refers to high level design that gives an overview of the proposed solution, platform, system and services. Here, an integration test plan is created to check the pieces of software system ability to work together.

- Detailed design refers to the low-level design where the actual components are designed. Component test plan is created in parallel during this phase.

- The coding takes place during implementation part following which unit testing takes place.

1. Verification is a static practice of verifying documents, design, code and program.

2. It does not involve executing the code.

3. It is human based checking of documents and files.

4. **Verification** uses methods like inspections, reviews, walk through, and Desk-checking etc.

5. **Verification** is to check whether the software conforms to specifications.

6. It can catch errors that validation cannot catch. It is low level exercise.

7. Target is requirements specification, application and software architecture, high level, complete design, and database design etc.

8. **Verification** is done by QA team to ensure that the software is as per the specifications in the SRS document.

1. **Validation** is a dynamic mechanism of validating and testing the actual product.

2. It always involves executing the code.

3. It is computer-based execution of program.

4. **Validation** uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.

5. **Validation** is to check whether software meets the customer expectations and requirements.

6. It can catch errors that verification cannot catch. It is High Level Exercise.

7. Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.

8. **Validation** is carried out with the involvement of testing team.

- **Advantages**
  - Simple and easy to use.
  - Each phase has specific deliverables.
  - Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle.
  - Works well for where requirements are easily understood.
  - Verification and validation of the product in early stages of product development
- **Disadvantages**
  - Very inflexible, like the waterfall model.
  - Little flexibility and adjusting scope is difficult and expensive.
  - Software is developed during the implementation phase, so no early prototypes of the software are produced.
  - Model doesn't provide a clear path for problems found during testing phases.
  - Costly and required more time, in addition to detailed plan

Agile Model

- Combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

- Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks.

- Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.

- At the end of the iteration a working product is displayed to the customer and important stakeholders.

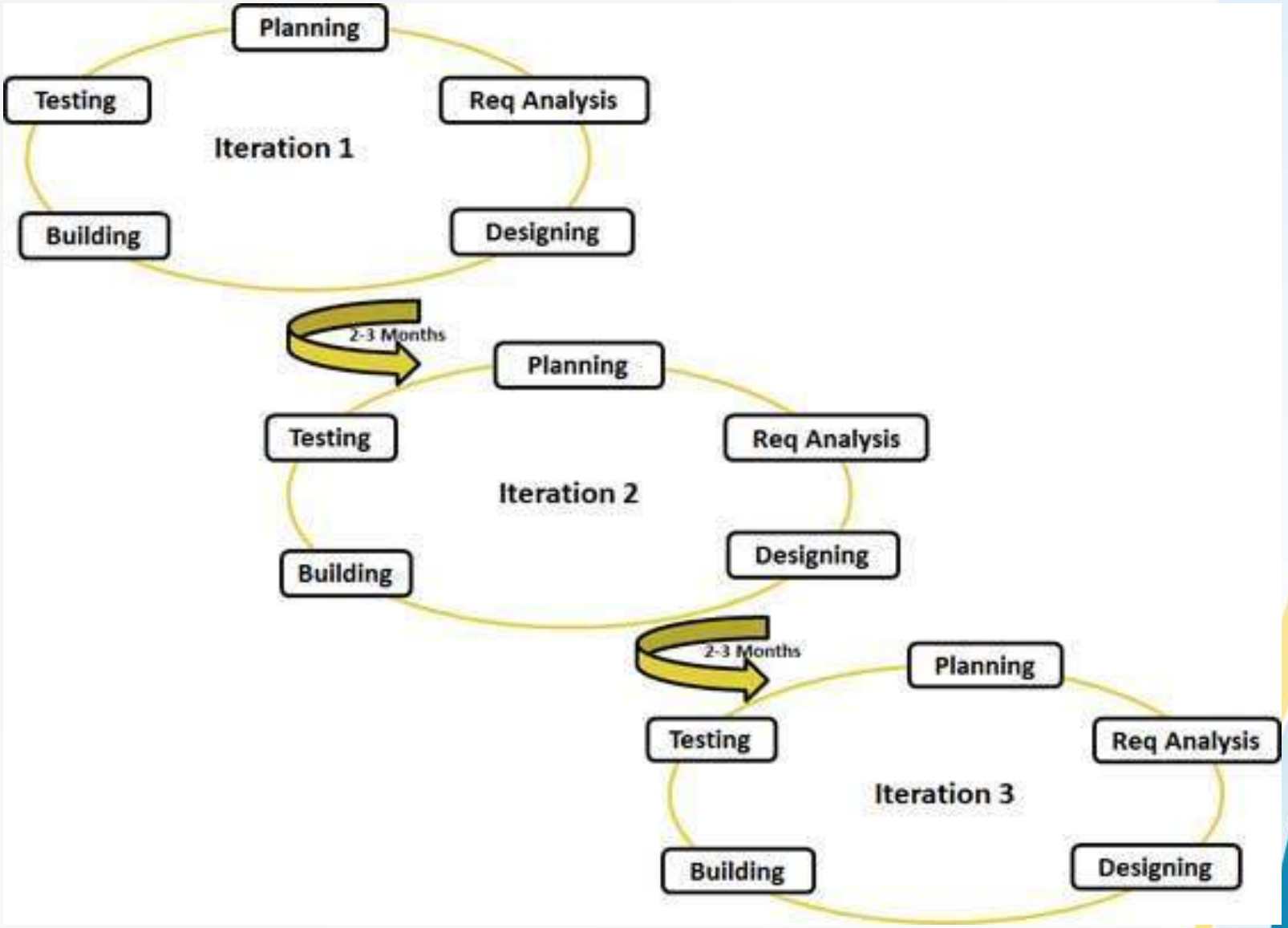


Fig. 2.11: Agile model

Advantages

- Is a very realistic approach to software development
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum.
- Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.

Disadvantages

- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- Transfer of technology to new team members may be quite challenging due to lack of very good documentation.

Agile vs Iterative vs Waterfall – {Process}

	Waterfall	Iterative (hybrid)	Agile
Quality	Quality focus changes from Analysis > Design > Code > Test	Quality focus shifts between Analysis/Design phase to Coding/Testing phase	Quality focus on all aspects of SDLC at any given time.
Quality Control	Detection & fixing during system and regression testing at the last phase of project.	Early detection & fixing in each iteration for new features. Followed by regression testing.	Early detection & fixing in each sprint followed by stabilization.
Continual Improvement (CA & PA)	Lessons learned from previous release implemented in next release	Lessons learned from previous Iteration implemented in next Iteration.	Lessons learned from previous sprint implemented in next sprint
Risk	No Risk Identification. Firefighting during testing phase.	Risk identification & mitigation in dev & test phase of each iteration.	Early identification & mitigation in every sprint.
Postmortem/ Retrospection	After every release	After every iteration/ milestone	After every sprint in retrospection meeting
Customer Feed back	At the end of the project.	At the end of every iteration	At the end of every sprint