Fundamentals of Software Engineering

Meti Dejene
tiimeekoo@gmail.com
Haramaya University

Chapter 2 - Software Process Models

Process and Process Models

- In the engineering domain, developing a solution to a given problem, whether building a bridge or making an electronic component, involves a sequence of interconnected steps.
- This sequence of steps executed to achieve a given purpose is called a process.
- So formally defining,
  - a process is a collection of interrelated and structured activities, actions, and tasks that are performed when some work product is to be created.

Key characteristics of a process

- **Structured Activities**: Planned and structured activities designed to contribute to achieve process objectives.
- **Inputs and Outputs**: Processes take inputs and transform them into outputs (deliverables, products, or results).
- **Interconnected Steps**: Activities are interconnected, the output of one activity becomes the input for the next. This ensures a logical and coherent flow of work.
- **Defined Roles and Responsibilities for each participant**.
- **Measurable and Controllable**: Metrics and key performance indicators (KPIs) may be used to monitor, evaluate, and improve the effectiveness and efficiency of a process.

Cont.

- Likewise, Software engineering shares this common feature with other engineering disciplines.
- Thus, in the context of Software engineering,
  - A software process is a systematic and organized set of interrelated activities and steps that are performed for the production of a high-quality software product.
- Fundamentally, there are four broad activities that are common to all software processes.

Cont.

- These activities are:

1. Software specification, where customers and engineers define the functionality of the software that is to be produced and the constraints on its operation.
2. Software development, where the software is designed and programmed to meet the specification must be produced.
3. Software validation, where the software is checked to ensure that it is what the customer requires.
4. Software evolution, where the software will be modified, when required, to reflect changing customer and market requirements.

Software Development Lifecycle (SDLC).

These fundamental activities are further refined and organized specifically into phases known as software development lifecycle (SDLC). These phases of SDLC are:

1. Requirement analysis and specification
2. Design
3. Implementation (Coding)
4. Testing
5. Deployment
6. Maintenance and Evolution

1. **Requirements Analysis and Specification:** Understanding and documenting the needs and expectations of stakeholders to define the requirements.
2. **Design:** Creating a high-level design that outlines the overall architecture and structure of the software system based on the requirements.
3. **Implementation/Coding:** Writing and compiling the source code according to the design specifications, transforming the design into executable software.
4. **Testing:** Verifying and validating the software to ensure that it meets the specified requirements and behaves as expected.
5. **Deployment:** Releasing the software for use into the operational environment.
6. **Maintenance and Evolution:** Making modifications, enhancements, and updates to the software to address issues, add new features, or adapt to changing requirements.

Software Process Models

- At a detailed level, the process that you adopt depends on the software you are building.
  - One process might be appropriate for creating software for an aircraft avionics system, while an entirely different process would be used for the creation of a Web site.
  - The activities can occur in a rigid sequence, with each activity completed before the next one begins, or several of the activities can occur simultaneously.
  - In some cases, most of these activities are performed only once and in other cases, several of the activities may be iterated several times.
- For these reasons, we have process models in software engineering.

Cont.

- A software process model is a compilation of best software development practices into a "recipe" that serves as a specific roadmap for software engineers.
  - It is a framework that defines the workflow of software development activities and phases (how they are organized, sequenced, and iterated) the artifacts/deliverables/work products that should be produced, and roles and responsibilities of individuals, teams or stakeholders.
- The basic premise behind a software process model is that,
  - in the situations for which the model is applicable, using the process model as the project's process will lead to low cost, high quality, reduced production time.

Cont.

- In the initial days of the software development, a software used to be created using an Ad-hoc model – the 'Build and fix' model.
- In this model,
  - The software application is created without any design specification.
  - The development team builds the application and deliver it to the customer.
  - The customer evaluates the software and provides feedback to the development team.
  - Based on the feedback, the developers fix the issues and deliver again.
  - This process goes on until the software is considered fit to use in production.
- However nowadays we have several software development process models.

Cont.

- It's important to note that the choice of a software process model significantly influences the project's success, as each model has its strengths and weaknesses.
- The selection of a particular model depends on the characteristics of the specific software development project such as project requirements, constraints, project size, complexity, and the level of flexibility required.
  - Choose a specific model or use a combination of models (a hybrid approach).
- Some of these commonly known process models are discussed below.

1. Waterfall Model

- The waterfall model is a linear and sequential approach to software development where each phase must be completed before moving on to the next and each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks.
- It is one of the earliest approach used in software development.
- This model tends to be among the less flexible approaches.
- Activities are performed in a strict top-down fashion and progress flows in largely one direction i.e. "downwards" like a waterfall (hence the name Waterfall).
- The waterfall model dictates that one should move to a next phase only when its preceding phase is reviewed and verified.

Cont.

- It sets distinct endpoints, goals and milestones for each phase of development process.
  - These endpoints or goals cannot be revisited after their completion.
- It places emphasis on deliverables of each phase and stresses highly on writing complete documentation (such as requirements and design documents) at various stages of development. This is because:
  - In less thoroughly documented methodologies, knowledge is lost if team members leave before the project is completed, and it may be difficult for a project to recover from the loss.
  - If a fully working design document is present new team members or even entirely new teams should be able to familiarize themselves by reading the documents.

Cont.

- One of the key features of the waterfall model is that all customer requirements are collected and approved before the beginning of the project, because no mid-project changes are allowed.
- The waterfall model doesn't include a project's end user or client as much as other development methodologies.
  - Users are consulted during the initial stages of gathering and defining requirements, incorporating client feedback after that.
- This methodology is good for teams and projects that want to develop a project according to fixed or unchanging requirements set forth at the beginning of the project.

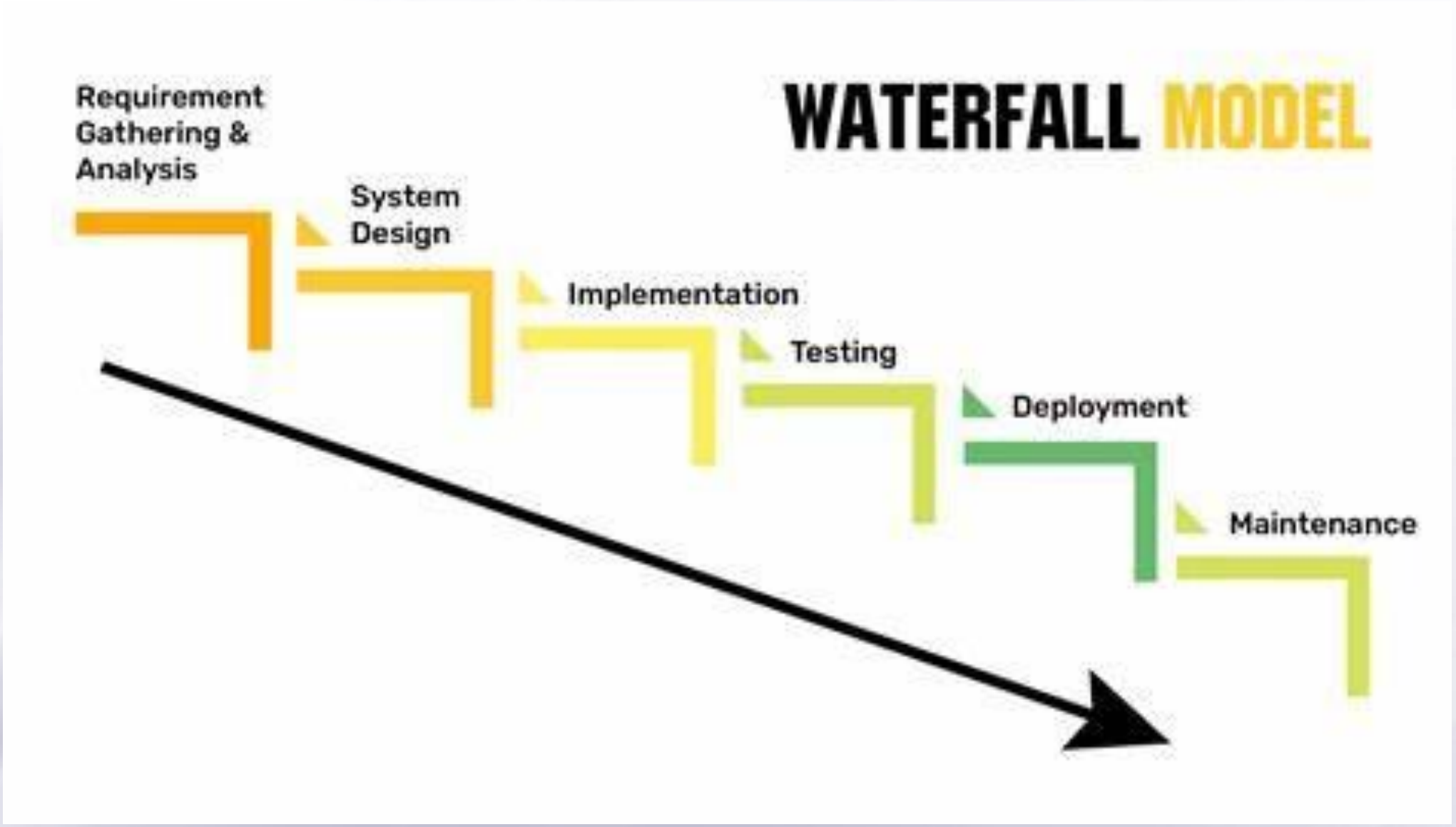

WATERFALL MODEL
Requirement Gathering & Analysis
System Design
Implementation
Testing
Deployment
Maintenance

Figure 2.1

Advantages of waterfall model

1. It is easy and straightforward to understand, use and manage.
2. It is easy to arrange tasks.
3. Has clearly defined stages and milestones.
4. It works well for smaller and low budget projects where requirements are very well understood and where changes are unlikely or expected to be minimal..
5. Predictable: All the requirements, processes, timelines, deadlines and end-product are known beforehand.
6. Process and results are well documented: This facilitates understanding of the tasks and the end product and can be useful for future reference and maintenance.

Disadvantages of waterfall model

1. No working software is produced until late during the development life cycle.
2. Inflexibility to changes: It is not flexible to accommodate changing requirements after the development process has started.
3. It is one directional: It does not allow teams to get back and make changes to the previous project development phases, once they are completed.
4. Delayed testing: In waterfall model, testing is delayed until the end of the project development, meaning you discover mistakes and flaws too late and have to invest a lot of time in fixing them instead of managing them early on.
5. Waterfall model does not let processes overlap for simultaneous work on different phases, reducing overall efficiency.

Disadvantages of waterfall model

6. Stakeholder or client involvement is often limited until the later stages of the project, which may potentially lead to misunderstandings or dissatisfaction.
7. Success in the Waterfall Model is highly dependent on accurately gathering and documenting requirements at the beginning of the project.
8. Risk of Project Failure: If requirements are not well-defined initially or change significantly during the project, there is a higher risk of project failure, as the model may struggle to adapt to these changes.
9. Pressure and stress caused by inflexible deadlines.

When should we use the waterfall model?

- Employ waterfall model only if :
  - All the requirements are known, clear/well-defined, and fixed.
  - There are no ambiguous requirements.
  - The project is short and simple.
  - The development environment is stable.
  - Personnel are trained and resources are adequately available.
  - The necessary technologies, tools and techniques used are well understood and stable.
  - Associated project risk is low.

2. Iterative Development Model

- The basic idea behind this model is to gradually develop a software in smaller portions at a time through repeated cycles, revisiting certain phases multiple times.
- The development does not begin with a full specification of requirements as it does not require a complete list of requirements before the project starts.
  - Rather it begins by specifying and implementing just part of the software.
- First focus on an initial simplified set of user features then the product can be constantly improved with the addition of new features during each iteration.
- Each iteration builds on the previous one.

Cont.

- However, each iteration may not produce a complete, shippable product.
- This iterative process is repeated and the product progressively gains more complexity and a broader set of features until the targeted system is complete.

Advantage

- the product can be built step by step and can be constantly improved with the addition of new functionalities/features during each iteration.
- it facilitates continuous improvement based on user feedback and changing requirements.
- Applicable for projects where requirements are unclear or expected to evolve.

3. Incremental Development Model

- This model involves breaking the development process into smaller increments, and dividing the requirements into multiple builds to gradually develop a software in "Lego-style" (smaller portions at a time).
- In a purely incremental model, all requirements must be specified upfront with each increment building upon the previous one.
  - So, requirement gathering and specification is completed before any other development activities start.

Cont.

- Once the requirement specification is completed, these requirements are first broken down into several modules or features (a set of increments) that can be constructed and delivered incrementally.
  - Each increment/build is a fully functional, deployable product focusing on a specific selected set of requirements or features.
- Selecting of requirements for an increment is done primarily based on the value the requirement provides to the end users and how critical they are for supporting other requirements.
- Then, the next task is implementation of these subsets of requirements and incrementally enhance the evolving versions until the full system is implemented.

Cont.

- The first increment is a simple working system implementing only a few basic features and providing a sub-set of the system functionality that is most needed by the customer.
- Then after, at each increment, design modifications are made and new functional capabilities are added until the full system is implemented.
- Also, since the delivery is done incrementally,
  - feedback from an increment can be incorporated in the next increment.
  - software developers can take advantage of what was learned during development of earlier increments of the system.
  - even new requirements that may get uncovered can also be incorporated.

Cont.

Advantages

- **Flexibility to Changes:** Adaptability to changing requirements and the ability to incorporate feedback, as modifications can be incorporated in subsequent increments without disrupting the entire project..
- **Early and Continuous Delivery:** Stakeholders get early visibility of the system, and this enables stakeholders to identify issues or changes early in the development process, reducing the risk of overall project failure.
- **Easier to Manage and Control:** Since the project is divided into smaller, manageable increments, it is easier to control and manage each increment individually.

Cont.

Disadvantage

- Managing multiple increments and their interdependencies can introduce complexity in terms of coordination, integration, and overall project control.

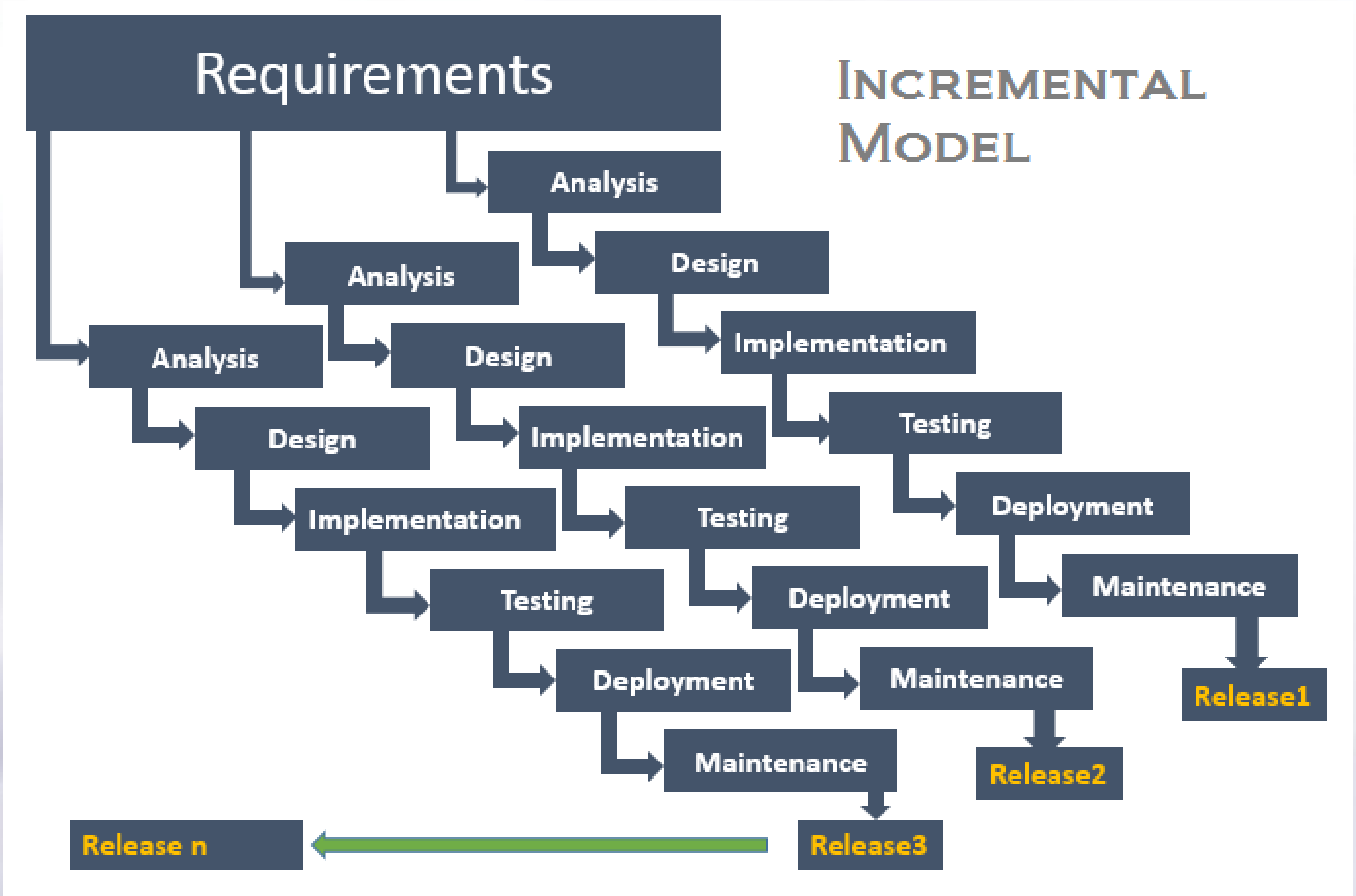

Requirements
Incremental Model
Analysis
Design
Implementation
Testing
Deployment
Maintenance
Release1
Release2
Release3
Release n

Figure 2.2

4. Spiral model

- The spiral model is a risk-driven software development process model with intensive customer involvement.
- This model gets its name from the appearance of its diagrammatic representation that looks like a spiral with many loops (see Figure 2.3).
- The development process in Spiral model, starts with a small set of requirement and goes through each development phase for those set of requirements.
  - The software engineering team adds functionality for the additional requirement in every-increasing spirals until the application is ready for final release.
- The exact number of loops of the spiral is not fixed and can vary from project to project.

Cont.

- The Spiral Model is often used for complex and large-scale software development projects, as it allows for a more flexible approach.
- It is also well-suited to projects with significant uncertainty or high levels of risk.
  - A risk is any adverse situation that might affect the successful completion of a project.
  - Some areas of risks in the software project are project overruns (time cost resources), change in requirements, loss of key project personnel, delay of necessary hardware, and technological breakthroughs which can make the project obsolete.
- A prominent feature of the spiral model is handling unforeseen risks that can show up much after the project has started.

Cont.

- So, this Spiral model is a combination of iterative development process model with a very high emphasis on risk identification, analysis and mitigation.
- It allows the potential for rapid development of iterative releases/refinement of the product through each iteration around the spiral.
- The first version is called the 'baseline spiral' and each subsequent spiral builds on the baseline spiral, each producing a new version with increased functionality.
- Each loop in the spiral is split into four sectors (or quadrants) as shown in Figure 2.3.

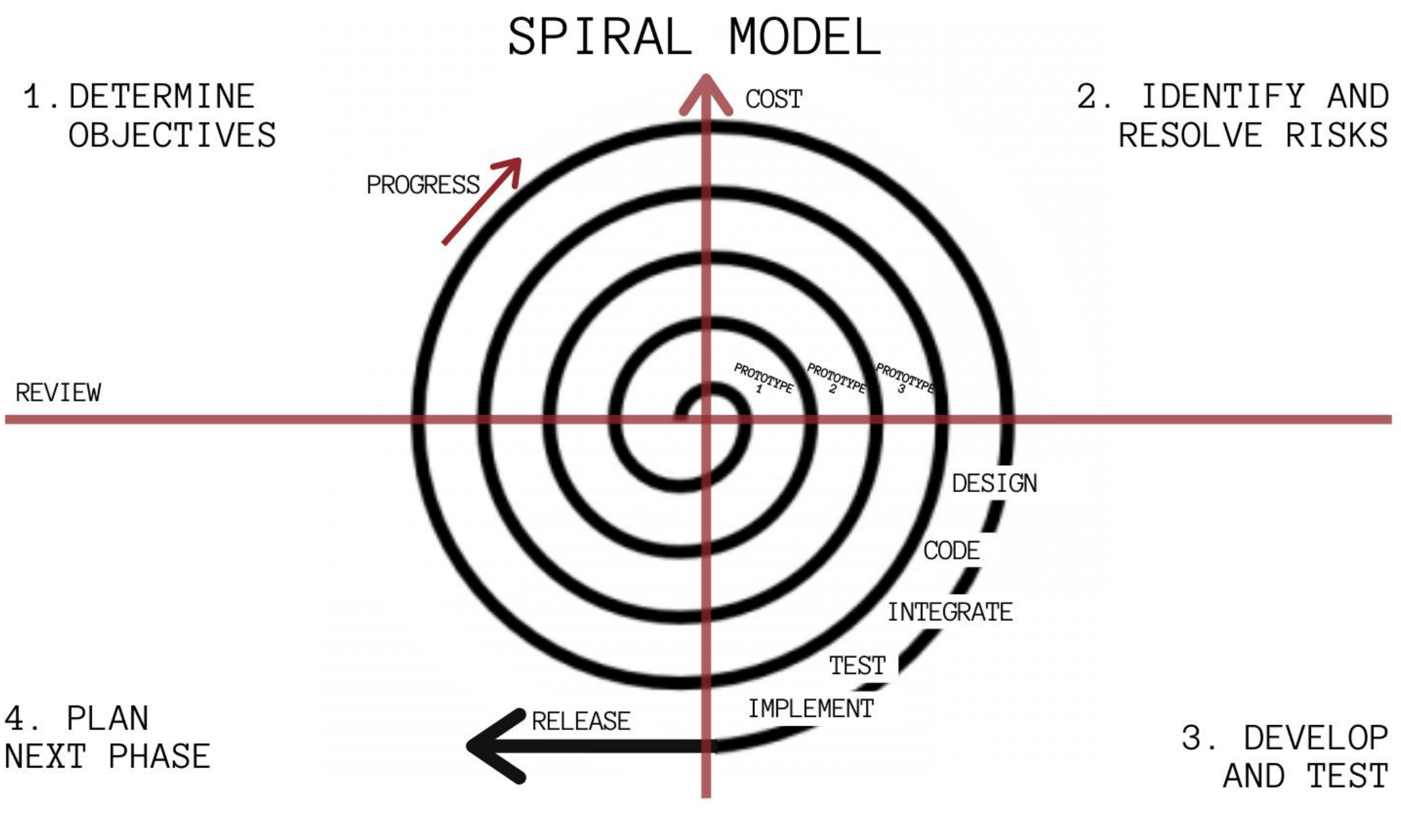

SPIRAL MODEL
1. DETERMINE OBJECTIVES
2. IDENTIFY AND RESOLVE RISKS
3. DEVELOP AND TEST
4. PLAN NEXT PHASE
COST
PROGRESS
REVIEW
PROTOTYPE 1
PROTOTYPE 2
PROTOTYPE 3
DESIGN
CODE
INTEGRATE
TEST
IMPLEMENT
RELEASE

Figure 2.3

Cont.

Quadrant 1: Define objectives and discover alternative solutions

- Each cycle in the spiral starts with the identification of purpose for that cycle.
- So, in this quadrant, specific objectives for that cycle of the project are defined, requirements and constraints are identified, a detailed management plan is drawn up and alternative solutions that are possible for achieving the targets are also proposed.

Quadrant 2: Risk analysis and resolution

- In this quadrant, all the proposed solutions are analyzed to select the best possible solution and potential risks are identified.
- For each of the identified risks, a detailed analysis is carried out and risk mitigation measures are taken to resolve and reduce the risk.

Cont.

Quadrant 3: Develop and test

- In this quadrant, identified features of the software are developed and then verified by a thorough testing.

Quadrant 4: Review and planning of the next phase

- This quadrant concerned with reviewing the progress made against defined objectives and 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.

Advantages of Spiral model

1. It is perfect for projects that are large and complex in nature as continuous development and evaluation help in mitigating any risk.
2. Because of its risk handling ability, the model is best suited for projects which are very safety critical as the strong emphasis on risk management helps to minimize the impact of uncertainty.
3. This model supports the client feedback and implementation of change requests (CRs) which is not possible in conventional models like a waterfall.
4. Since customers are involved, there are higher chances of customer satisfaction.

Disadvantages of Spiral model

1. Because of the prototype development and risk analysis in each cycle, it is very expensive and time taking.
2. It is not suitable for a simpler and smaller project because of multiple cycles.
3. It requires more documentation as compared to other models.
4. Difficulty in time management: As the number of cycles is unknown at the start of the project, time estimation is very difficult.
5. Time-consuming: It requires multiple evaluations and reviews.
6. Resource intensive: It requires a significant investment in planning, risk analysis, and evaluations.

5. Prototyping Model

- This model suggests building a working prototype of a system, before development of the actual software.
  - A prototype is a crude (partial and preliminary) implementation and working model of a system with limited functional capabilities, low reliability, or inefficient performance as compared to the actual software.

Cont.

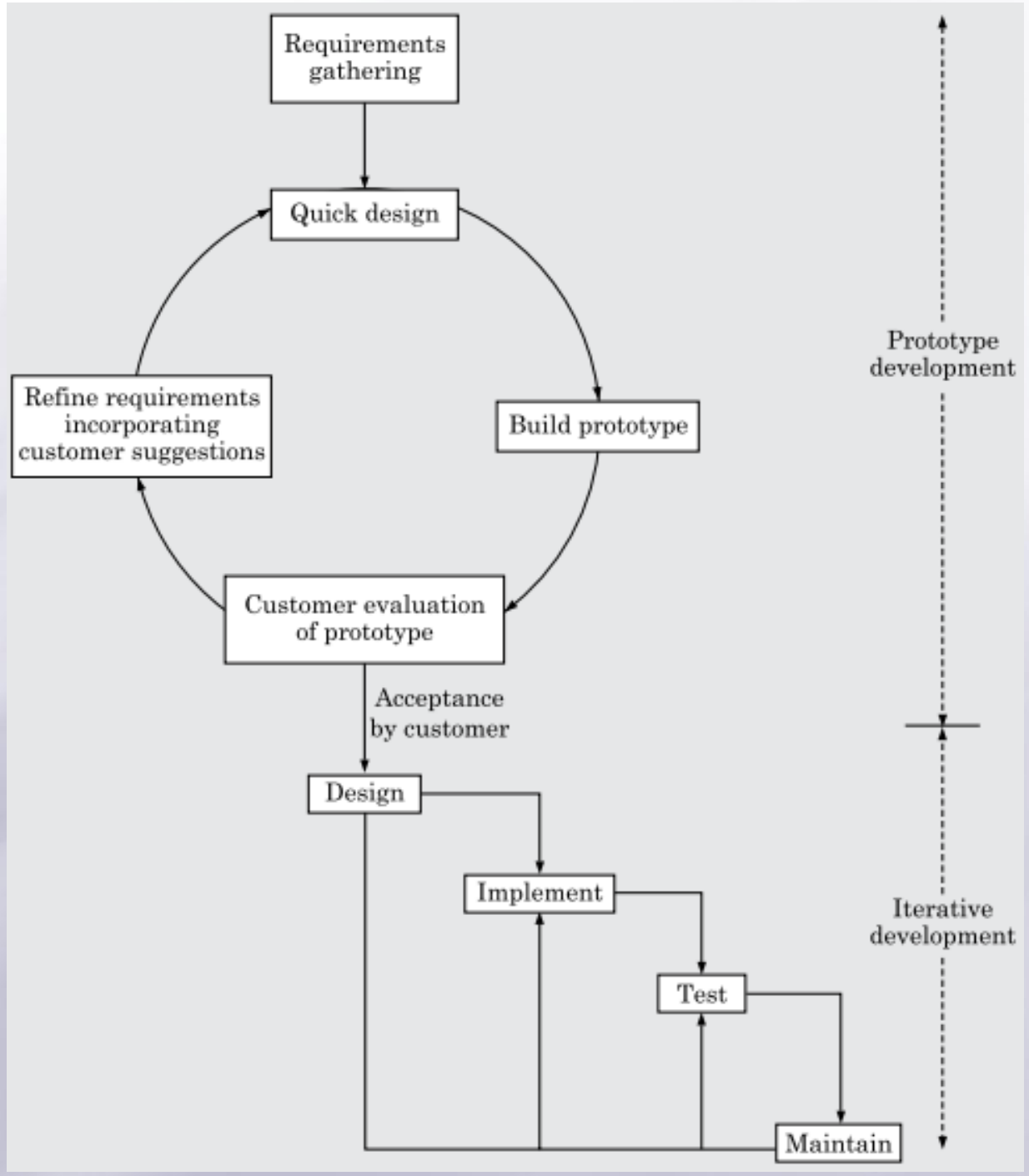
- The Prototyping model is particularly useful for complicated and large systems for which there is no manual process or existing system to help determine the requirements.
- In such cases, clients may not know exactly what their requirements are before they see working model and so change their requirements, leading to redesign, redevelopment, and retesting, and increased costs.
- So, a throwaway prototype is built to help understand the requirements based on the currently known requirements.

Cont.

- By using this prototype, the client can get an actual feel of the system, which can enable the client to better understand the requirements of the desired system.
  - This results in more stable requirements that change less frequently.
- After the prototype has been developed, the end users and clients are given an opportunity to use and explore the prototype.
- Based on their experience, they provide feedback to the developers about the prototype: what is correct, what needs to be modified, what is missing, what is not needed, etc.

Cont.

- Based on the feedback, the prototype is modified to incorporate some of the suggested changes, and then the users and the clients are again allowed to explore the system.
- This cycle repeats until the benefit from further changing the system and obtaining feedback is outweighed by the cost and time involved in making the changes and obtaining the feedback.
- Overall, prototyping is well suited for projects where requirements are hard to determine and the confidence in the stated requirements is low.


Requirements gathering
Quick design
Build prototype
Customer evaluation of prototype
Refine requirements incorporating customer suggestions
Acceptance by customer
Design
Implement
Test
Maintain
Prototype development
Iterative development

Cont.

Advantage

- Heavy user involvement allows to identify stable requirements.
- This mitigate risks associated with not well-known requirements.
- Prototypes provide a tangible representation of the system, facilitating improved communication between developers and stakeholders.

Disadvantage

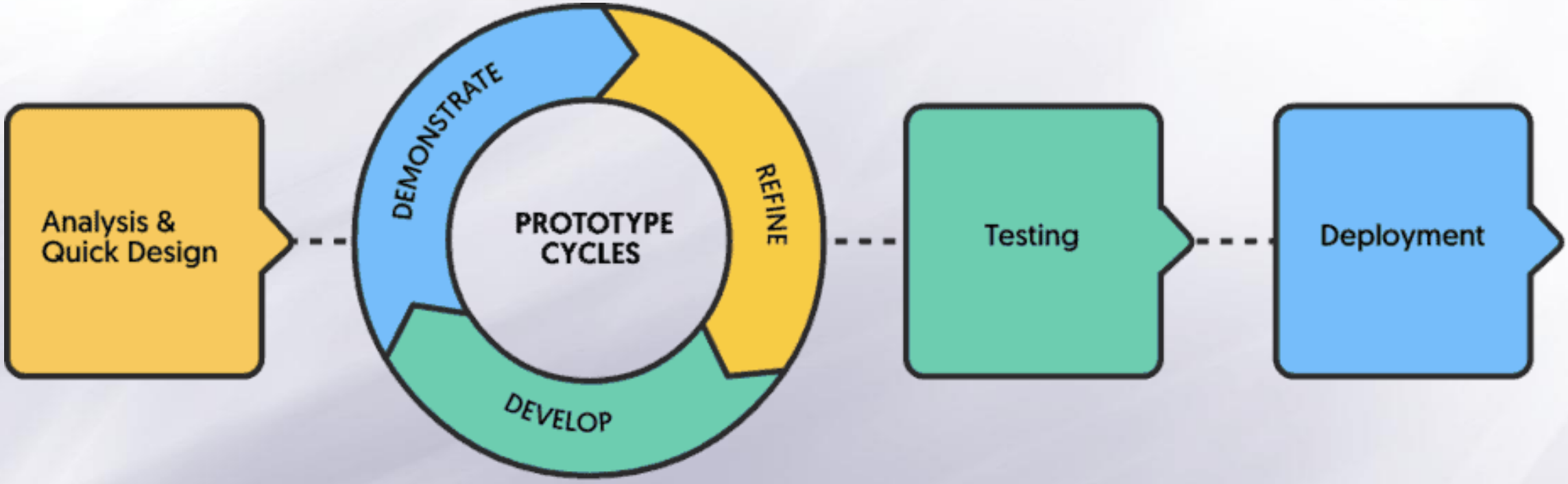
- Cost and Time Overruns: If not managed properly, the iterative nature of prototyping can lead to increased development time and costs.

6. RAD Model

- The RAD (Rapid Application Development) model has the features of both prototyping and evolutionary models.
- It prioritizes rapid prototyping and quick feedback with minimal planning and testing.
- In this model, quick prototypes are constructed and incrementally the features are refined and delivered to the customer over extremely short development cycles.
- But unlike in the prototyping model, here the prototypes are not thrown away but they are enhanced further through multiple iterations.
  - The customer evaluates the prototype and gives feedback on the specific improvements that may be necessary to incorporate.

Cont.

- In RAD model, development takes place in a series of short cycles or iterations.
  - It distributes the analysis, design, build and test phases into a series of short, iterative development cycles.
- It divides the system into components that are developed in parallel by teams, then integrated in a short time frame.
- The RAD model emphasizes on reuse of the existing components and code modules to accelerate development and for completing a project faster.
- So, minimal time spent in planning and heavy reuse facilitates faster development in RAD.
- Use RAD model when time to market is critical, and the system can be modularized for rapid parallel development.



A typical RAD cycle