MENOUFIA UNIVERSITY
جامعة المنوفية
منارة المعرفة فى قلب الدلتا

Software Engineering

Dr. Mohammed Badawy

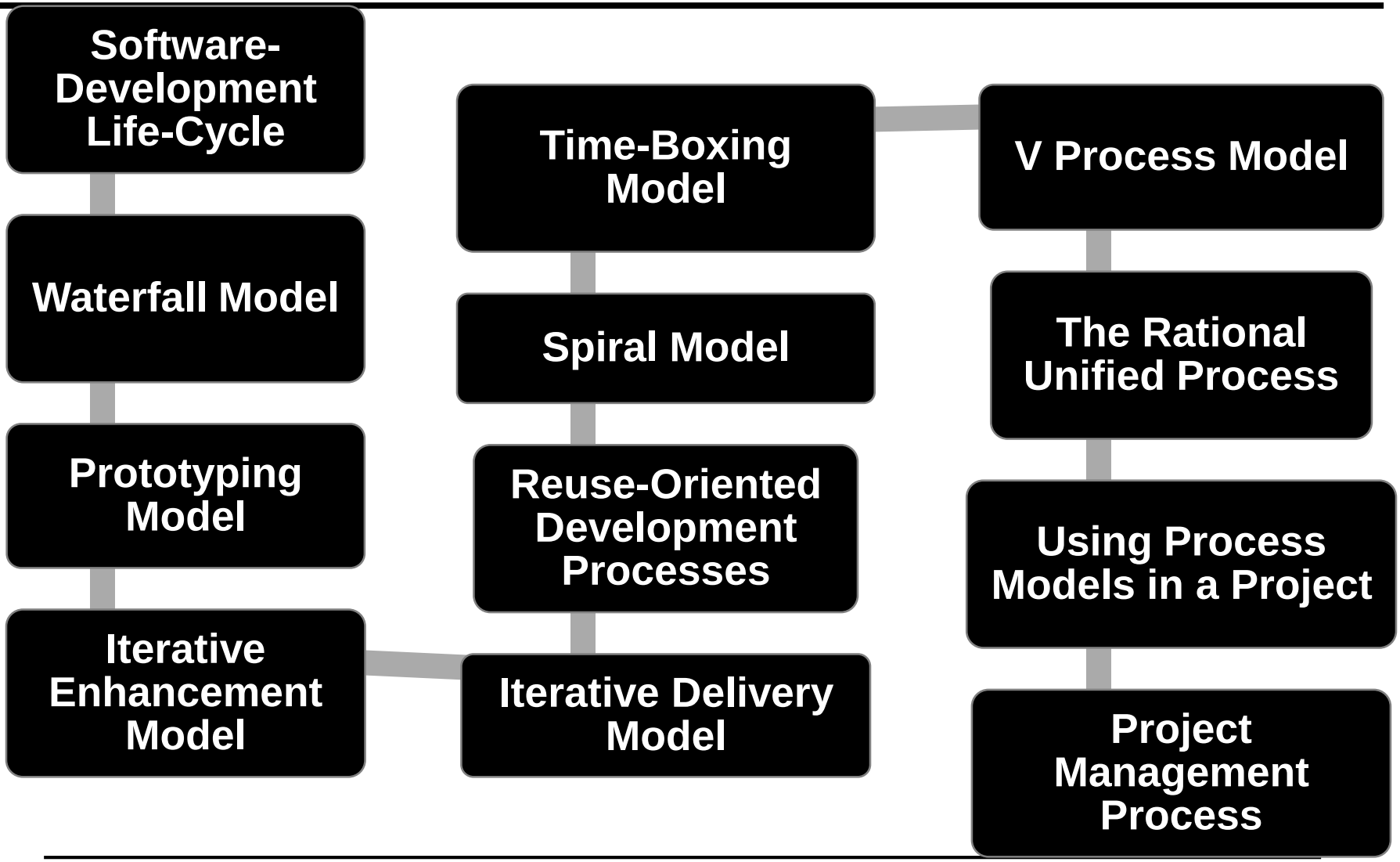
mbmbadawy@yahoo.com
mohamed.badawi@el-eng.menofia.edu.eg
Room: 417, Ext. 7332

13 May 2016

Chapter 2

Software-development Life-cycle Models

Contents


Software-Development Life-Cycle
Waterfall Model
Prototyping Model
Iterative Enhancement Model
Time-Boxing Model
Spiral Model
Reuse-Oriented Development Processes
Iterative Delivery Model
V Process Model
The Rational Unified Process
Using Process Models in a Project
Project Management Process

Software-development Life-cycle

- Besides delivering software, high quality, low cost, and low cycle time are also goals which software engineering must achieve.
- In other words, the systematic approach must help achieve a high quality and productivity (Q&P)
- The three main factors that influence Q&P are people, processes, and technology
- The final quality delivered and productivity achieved depends on the skills of the people involved in the software project, the processes people use to perform the different tasks in the project, and the tools they use

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
- The systematic approach must help achieve a high quality and productivity (Q&P)

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

Software-development Life-cycle

The SDLC can be divided into 5-9 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

Software-development Life-cycle

Recognition of Need:

- Recognition of need is nothing but the problem definition.
- The first stage of any project or system-development life-cycle.
- It is a brief investigation of the system under consideration.
- At this stage the need for changes in the existing system are identified and shortcomings of the existing system are detected.

Software-development Life-cycle

Feasibility Study:

- A feasibility study 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.

Software-development Life-cycle

Categories of Feasibility

- **Organizational Feasibility**: is how well a proposed information system supports the objectives of the organization
  - For example, projects that do not directly contribute to meeting an organization's strategic objectives are typically not funded
- **Economic Feasibility**: is concerned with whether expected cost savings, increased revenue, reductions in required investments, and other types of benefits will exceed the costs of developing and operating a proposed system

Software-development Life-cycle

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

- **Operational Feasibility:** is the willingness and ability of management, employees, customers, suppliers, and others to operate, use, and support a proposed system
  - If the software for a new system is too difficult to use, employees may make too many errors and avoid using it

Software-development Life-cycle

Project Analysis:

- Project analysis is a detailed study of the various operations performed by a system and their relationships within and outside the system.

System Design:

- System design is the most creative and challenging phase of the system-development life-cycle.
- It describes the final system and process by which it is developed.

Software-development Life-cycle

Coding:

- The goal of the coding phase is to translate the design of the system into code in a given programming language.
- This phase affects both testing and maintenance phases.
- Well-written code can reduce the testing and maintenance effort.

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.

Software-development Life-cycle

Implementation:

- 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.
- 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.

Software-development Life-cycle

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.
- 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.

Software-development Life-cycle

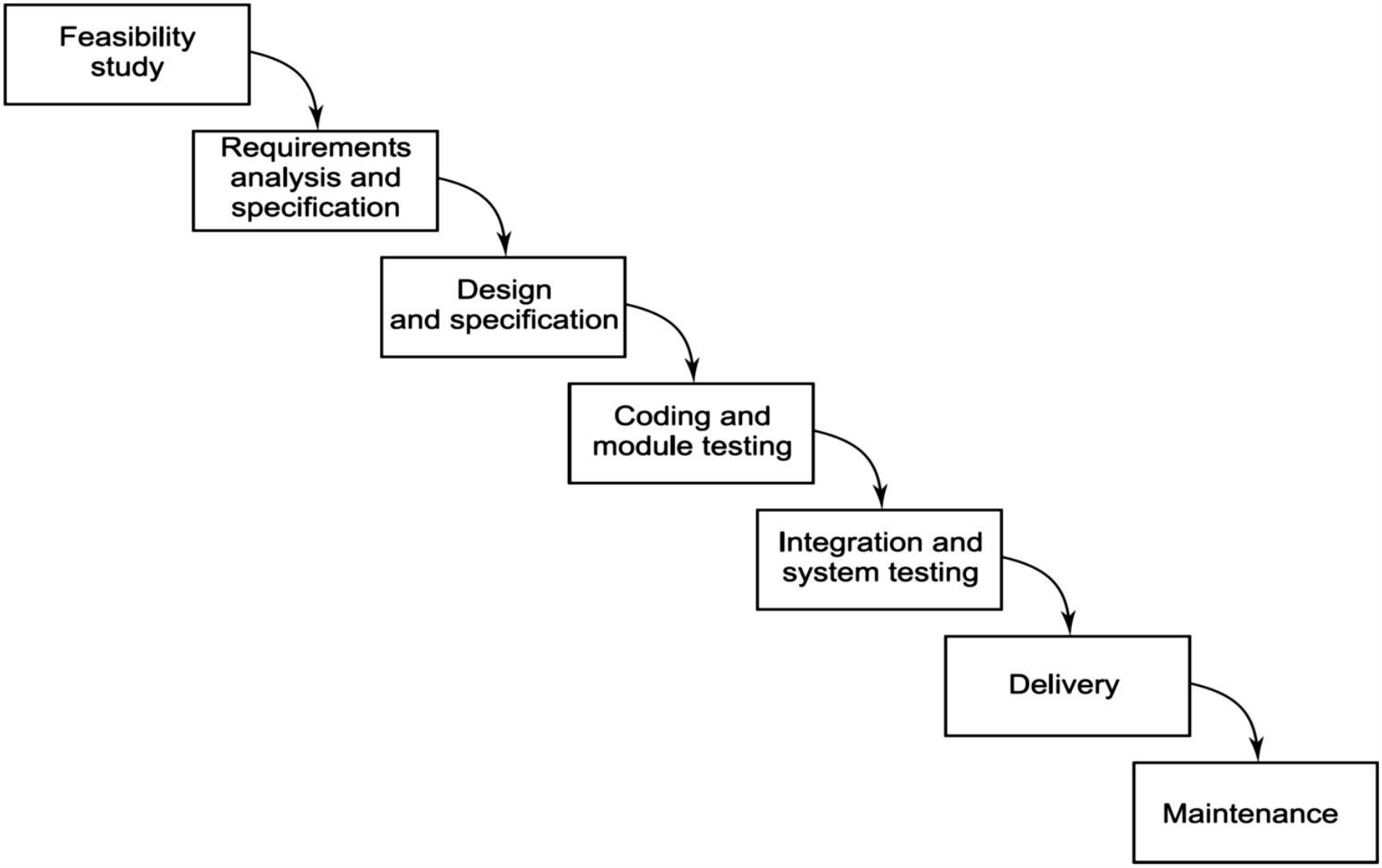
Types of Maintenance:

- ***Corrective Maintenance.*** Means repairing processing or performance failures or making changes because of previously uncorrected problems.
- ***Adaptive Maintenance.*** Means changing the program function. This is done to adapt to the external environment change.
- ***Perfective Maintenance.*** Means enhancing the performance or modifying the programs to respond to the user's additional or changing needs.
- ***Preventive Maintenance.*** Is the process by which we prevent our system from being obsolete. This maintenance prevents the system from dying out.

Waterfall Process Model

- The waterfall model is a very common software development process model.
- Because of the cascade from one phase to another, this model is known as the waterfall model.
- Each phase, in turn, is structured as a set of activities that might be executed by different people concurrently.
- The waterfall model is illustrated in the next Figure

Waterfall Process Model



Waterfall Process Model

1. Feasibility Study:

- The purpose of this phase is to produce a feasibility study document that evaluates the costs and benefits of the proposed application.
- The feasibility study is usually done within limited time bounds and under pressure.
- Often, its result is an offer to the potential customer.
- At the end of this phase, a report called a feasibility study is prepared by a group of software engineers.
- This report determines whether the project is feasible or not.

Waterfall Process Model

2. Requirement Analysis and Specification:

- This phase exactly tells the requirements and needs of the project.
- 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.”
- The resultant document is known as the software requirement specification (SRS) document.

Waterfall Process Model

3. Design and Specification:

- The goal is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.

4. Coding and Module Testing:

- Is the phase in which we actually write programs using a programming language.
- The output of this phase is an implemented and tested collection of modules.

Waterfall Process Model

5. Integration and System Testing:

- The modules are integrated in a planned manner.
- The different modules making up a software product.
- 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.

Waterfall Process Model

- 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.

Waterfall Process Model

- 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.

Waterfall Process Model

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.

Waterfall Process Model

Advantages of Waterfall Model

- The main advantage is its simplicity
- It is straightforward and divides the large task of building a software system into a series of cleanly divided phases, each phase dealing with a separate logical concern
- Waterfall model is also easy to administer in a contractual setup.
- As each phase is completed and its work product produced, some amount of money is given by the customer to the developing organization

Waterfall Process Model

Disadvantages of Waterfall Model

- **It assumes that the requirements of a system can be frozen** before the design begins (having unchanging requirements is unrealistic for some projects)
- Freezing the requirements usually requires choosing the hardware (due to the speed at which hardware technology is changing, it is likely that the final software will use a hardware technology on the verge of becoming obsolete).
- This is clearly not desirable for such expensive software systems.

Waterfall Process Model

- **It follows the "big bang" approach**. The entire software is delivered in one shot at the end.
- If the project runs out of money in the middle, then there will be no software. That is, it has the "all or nothing" value proposition.
- **It encourages "requirements bloating"**
- Since all requirements must be specified at the start, it encourages the users and other stakeholders to add even those features which they think might be needed.

Waterfall Process Model

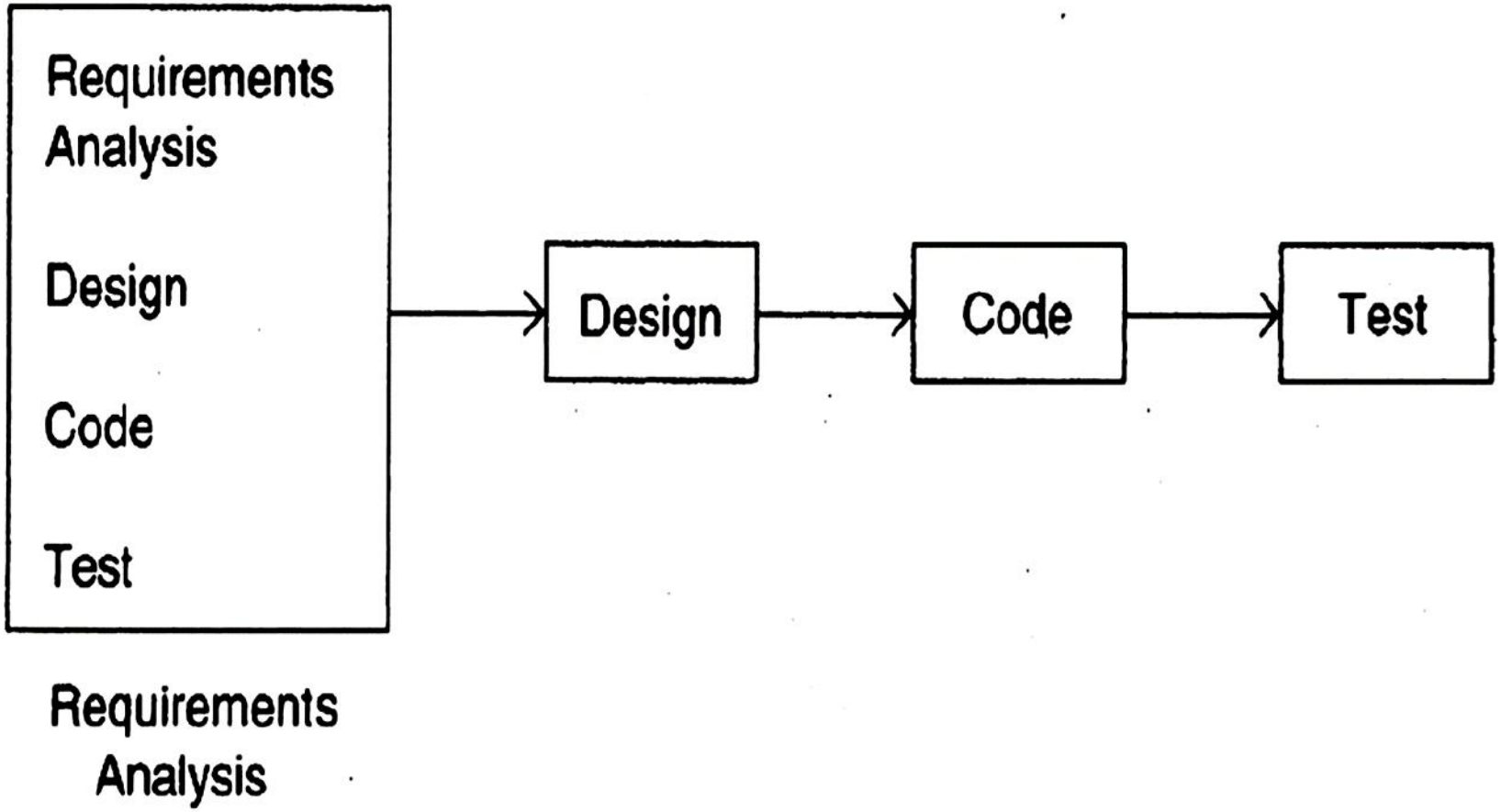
- Despite these limitations, the waterfall model has been the most widely used process model
- If the developing organization is quite familiar with the problem domain and the requirements for the software are quite clear, the waterfall model works well, and may be the most efficient process

Prototyping Process Model

- The goal of a prototyping-based development process is to counter the first limitation of the waterfall model
- The basic idea here is that instead of freezing the requirements before any design or coding can proceed, a throwaway prototype is built to help understand the requirements
- Prototyping is an attractive idea for complicated and large systems.
- This might be needed for novel systems, where it is not clear that constraints can be met or that algorithms can be developed to implement the requirements

Prototyping Process Model

- The Prototyping model is illustrated in the Figure below



Prototyping Process Model

- After the prototype has been developed, the users are given an opportunity to use and explore the prototype
- They provide feedback to the developers regarding the prototype: What is correct, what needs to be modified, what is missing, what is not needed, etc.
- Based on the feedback, the prototype is modified to incorporate some of the suggested changes that can be done easily, and then the users and the clients are again allowed to use the system
- This cycle repeats until we reach the requirements needed

Advantages of Prototyping Model

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

Iterative Model

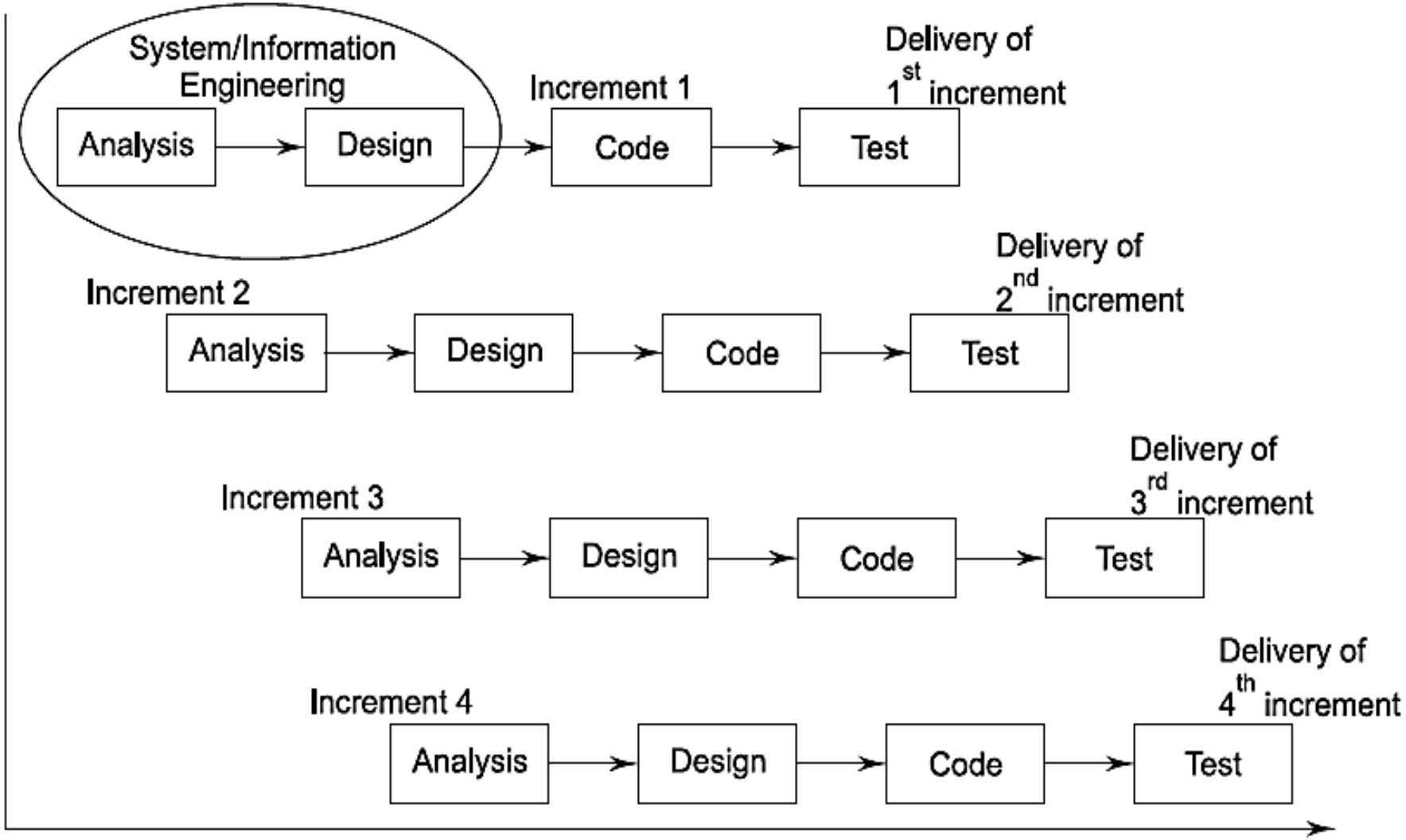
- The iterative development process model tries to combine the benefits of both prototyping and the waterfall model.

- Two common approaches for iterative development model:
  - ✓ ***Iterative-Enhancement*** model
  - ✓ ***Iterative Delivery*** model

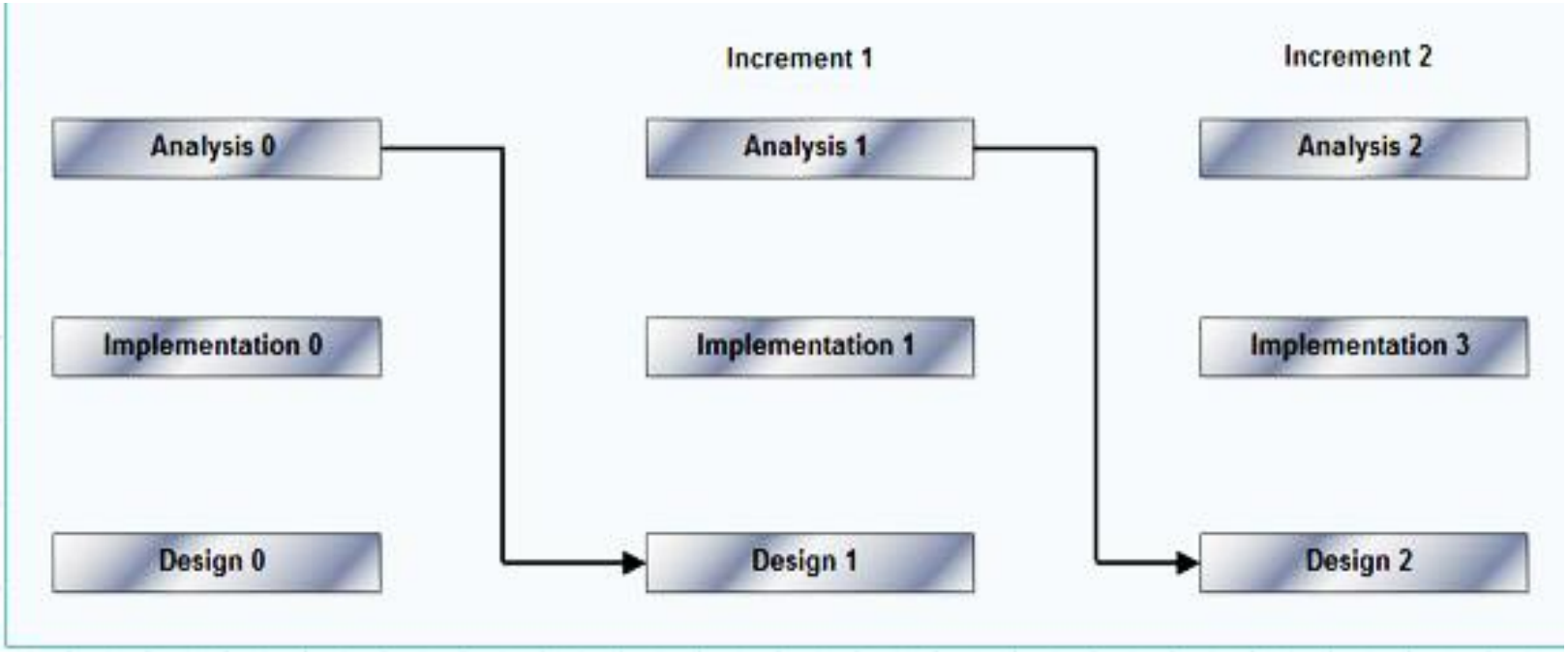
Iterative Enhancement Model

- The iterative-enhancement model combines elements of the linear sequential model (Waterfall) with the iterative philosophy of prototyping
- The software is broken down into several modules, which are incrementally developed and delivered
  1. The development team develops the core module of the system
  2. It is later refined into increasing levels of capability of adding new functionalities in successive versions

Iterative Enhancement Model



Iterative Enhancement Model



Iterative Enhancement Model

- 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; and spelling and grammar checking in the third increment

Advantages of Iterative Enhancement Model

- The feedback from early increments improve the later stages
- The possibility of changes in requirements is reduced
- Users get benefits earlier than with a conventional approach
- Early delivery of some useful components improves cash flow
- Smaller sub-projects are easier to control and manage
- The project can be temporarily abandoned if more urgent work crops up
- Job satisfaction is increased for developers

Disadvantages of Iterative Enhancement Model

- 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)

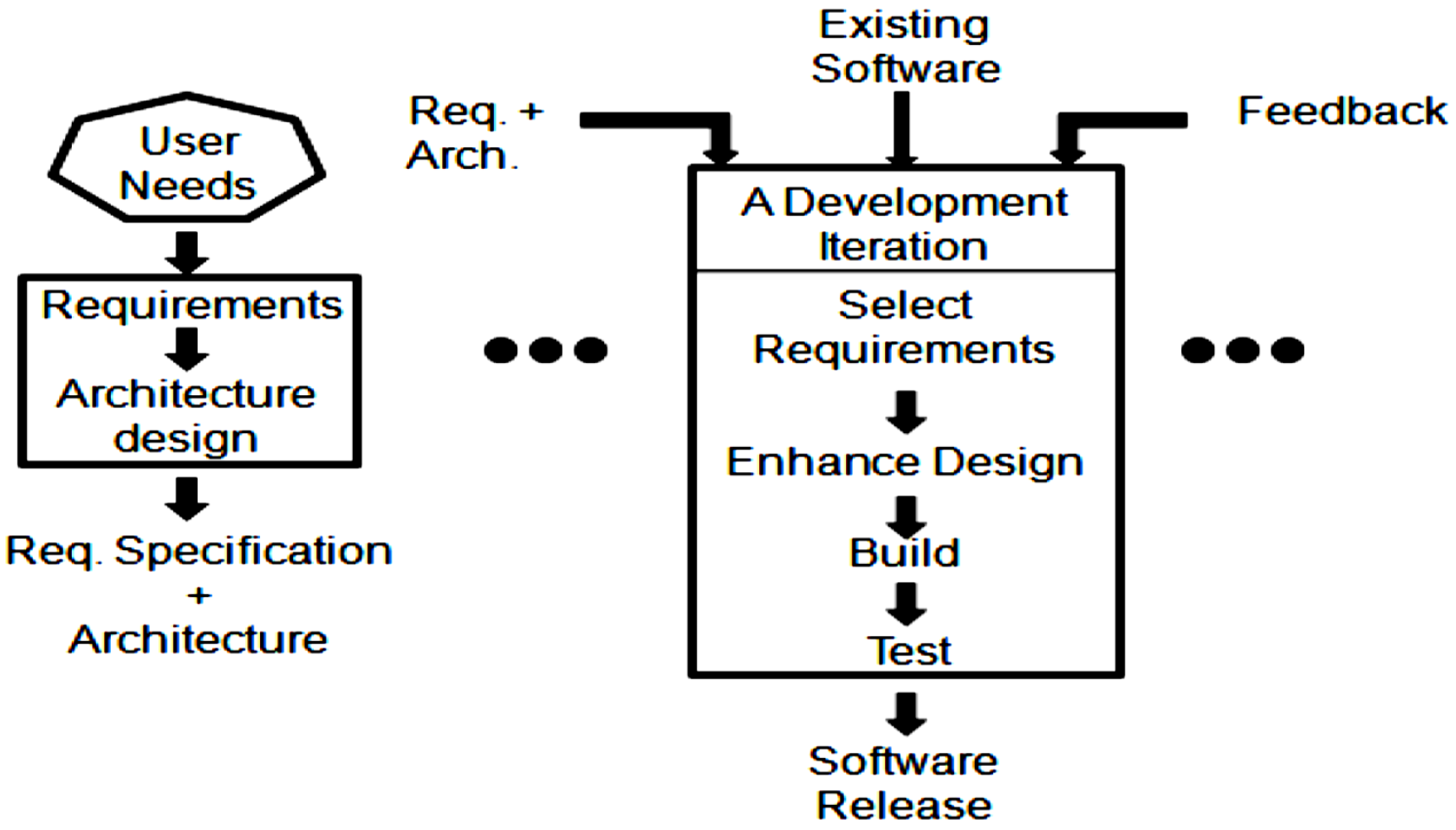
Iterative Delivery Model

- In this approach, the requirements and the architecture design is done in a standard waterfall or prototyping approach, but deliver the software iteratively
- That is, the building of the system, is done iteratively, though most of the requirements are specified upfront
- We can view this approach as having one iteration delivering the requirements and the architecture plan, and then further iterations delivering the software in increments.

Iterative Delivery Model

- At the start of each delivery iteration, requirements which will be implemented in this release are decided, and then the design is enhanced and code developed to implement the requirements
- The iteration ends with delivery of a working software system providing some value to the end user

Iterative Delivery Model

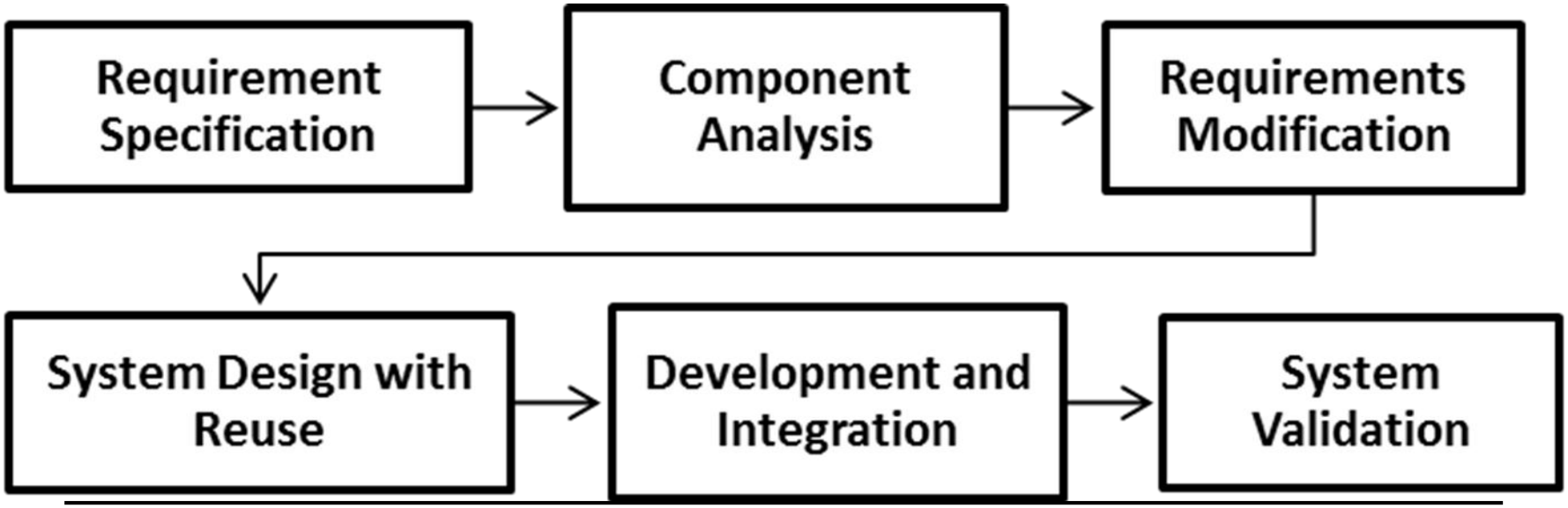


Component-Based (Reuse-Oriented) Development

- In the majority of software projects, there is some software reuse
- This usually happens when people working on the project know of designs or code which is similar to that required
- They look for these, modify them as required and incorporate them into their system
- This reuse-oriented approach relies on a large base of reusable software components.

Reuse-Oriented Development Processes

- Sometimes, these components are systems in their own right that may provide specific functionality such as text formatting or numeric calculation.
- The generic process model for component-based software engineering (CBSE) is shown in the next Figure



Reuse-Oriented Development Processes

- While the initial requirements specification stage and the validation stage are comparable with other processes, the intermediate stages in a reuse-oriented process are different.
- These stages are as follows:

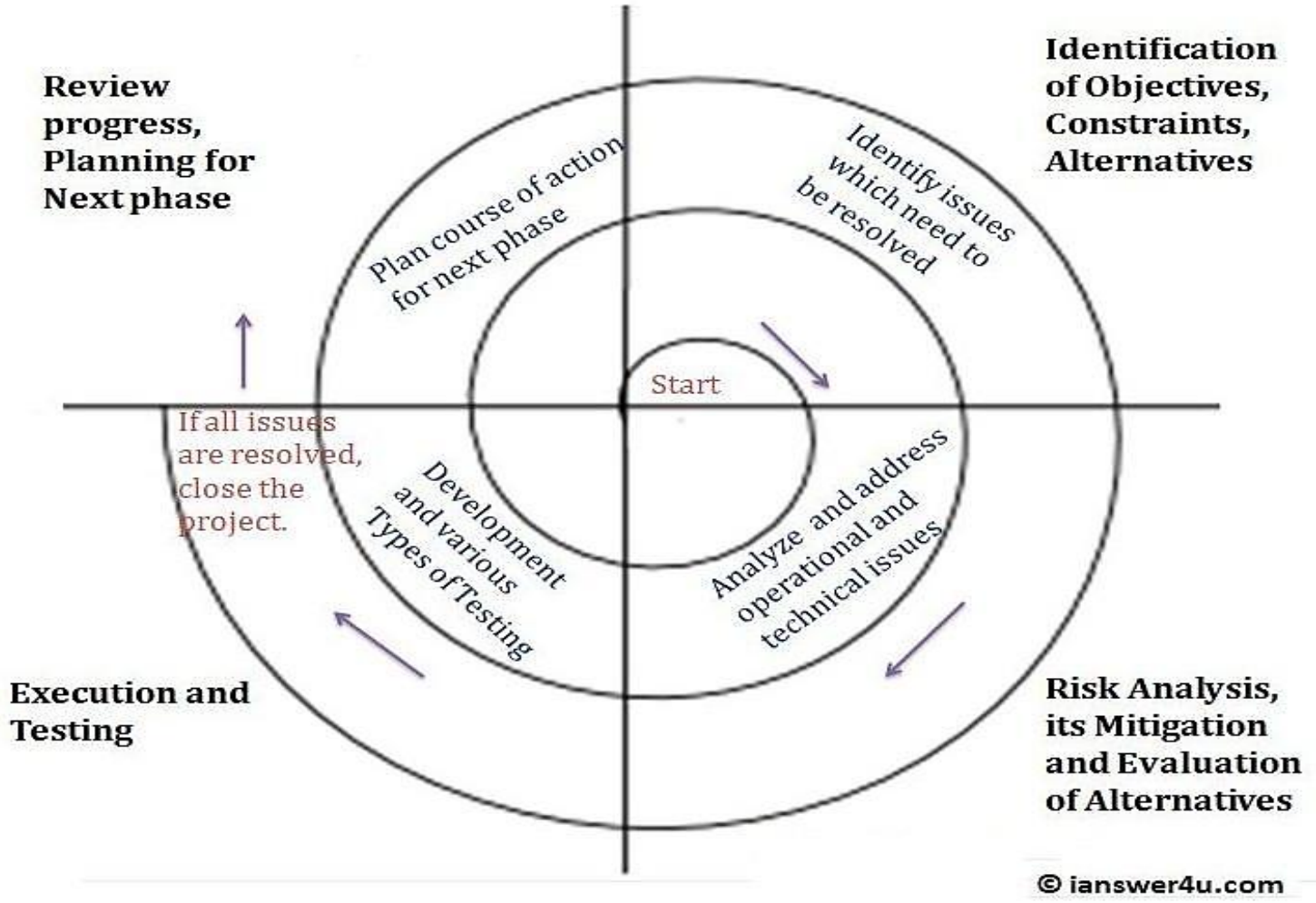
Reuse-Oriented Development Processes

1. **Component analysis**: Given the requirements specification, a search is made for components to implement that specification.
2. **Requirements modification**: During this stage, the requirements are analyzed using information about the components that have been discovered. They are then modified to reflect the available components.
3. **System design with reuse**: During this phase, the framework of the system is designed or an existing framework is reused.
4. **Development and integration**: Software that cannot be externally procured is developed, and the components and COTS systems are integrated to create the new system.

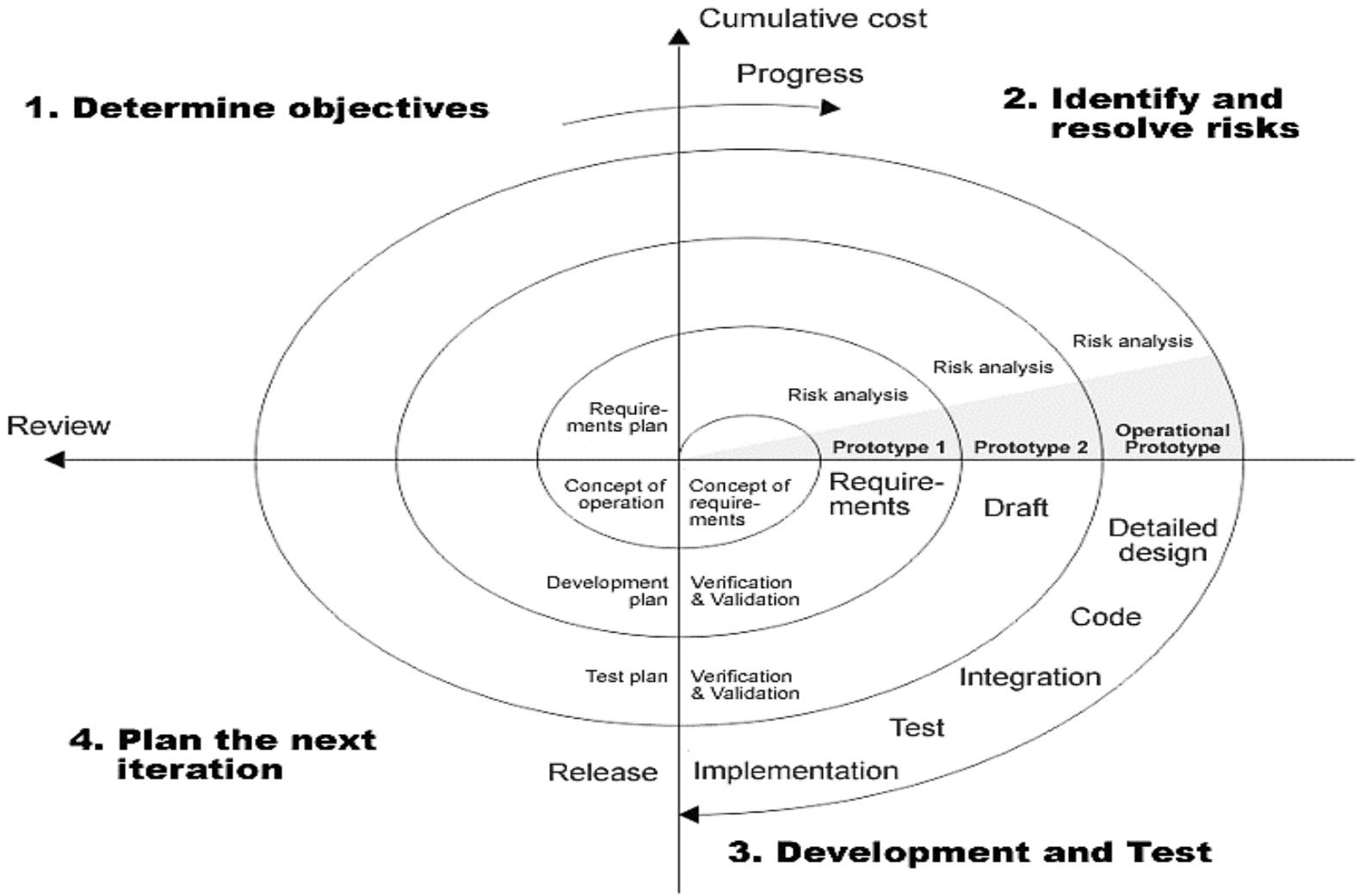
Spiral Model

- Rather than represent the software process as a sequence of activities with some backtracking from one activity to another, the process is represented as a spiral.
- Each loop in the spiral represents a phase of the software process.
- Thus, the innermost loop might be concerned with system feasibility, the next loop with requirements definition, the next loop with system design and so on.

Spiral Model



Spiral Model



Spiral Model

- Each loop in the spiral is split into four sectors:
  1. **Objective setting:** Specific objectives for that phase of the project are defined. Constraints on the process and the product are identified and a detailed management plan is drawn up. Project risks are identified. Alternative strategies, depending on these risks, may be planned.
  2. **Risk assessment and reduction**: For each of the identified project risks, 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

3. **Development and validation**: After risk evaluation, a development model for the system is chosen. For example, if user interface risks are dominant, an appropriate development model might be evolutionary prototyping. If safety risks are the main consideration, development based on formal transformations may be the most appropriate and so on.
4. **Planning:** The project is reviewed and a decision made whether to continue with a further loop of the spiral. If it is decided to continue, plans are drawn up for the next phase of the project.

Advantages of Spiral Model

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

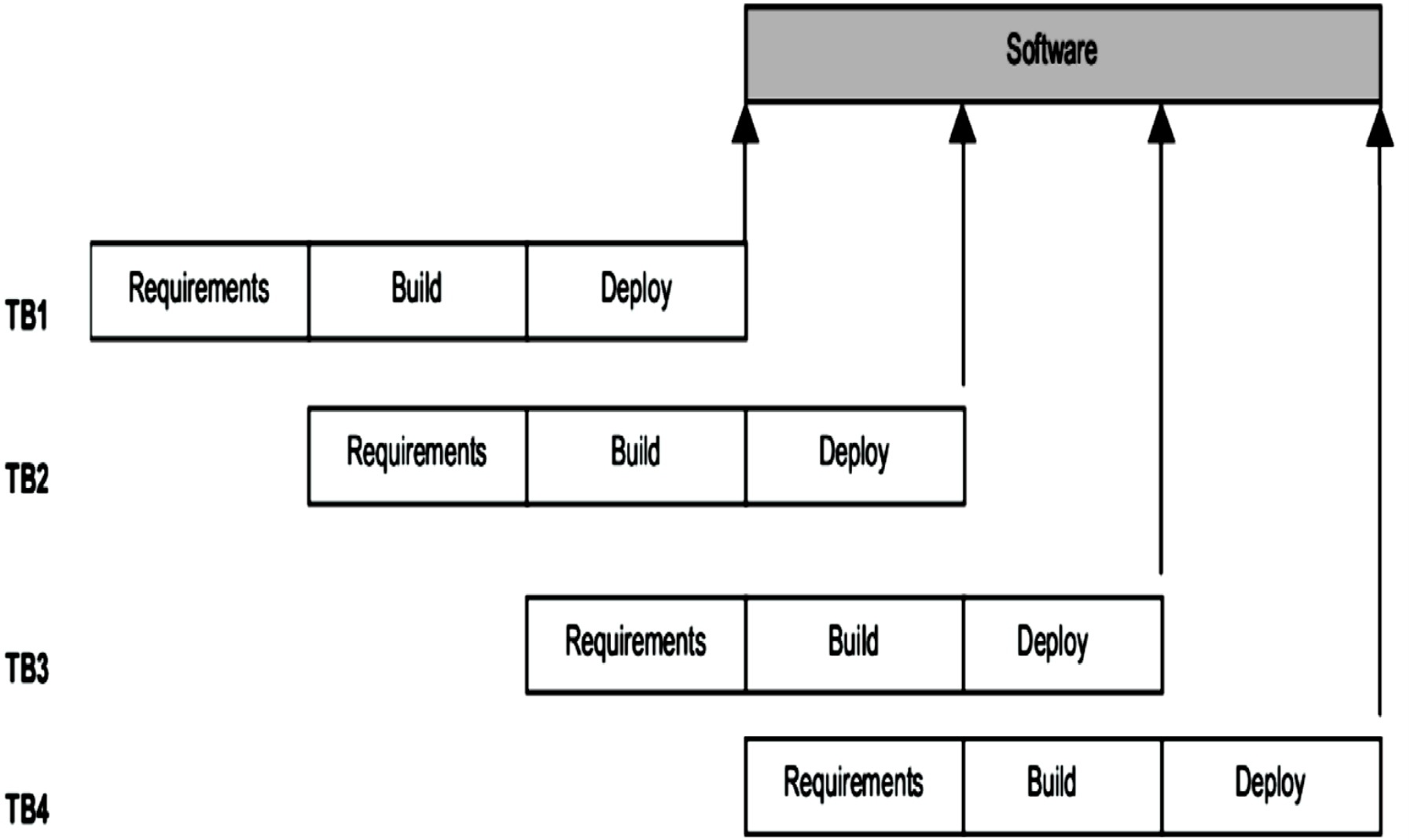
Disadvantages of Spiral Model

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

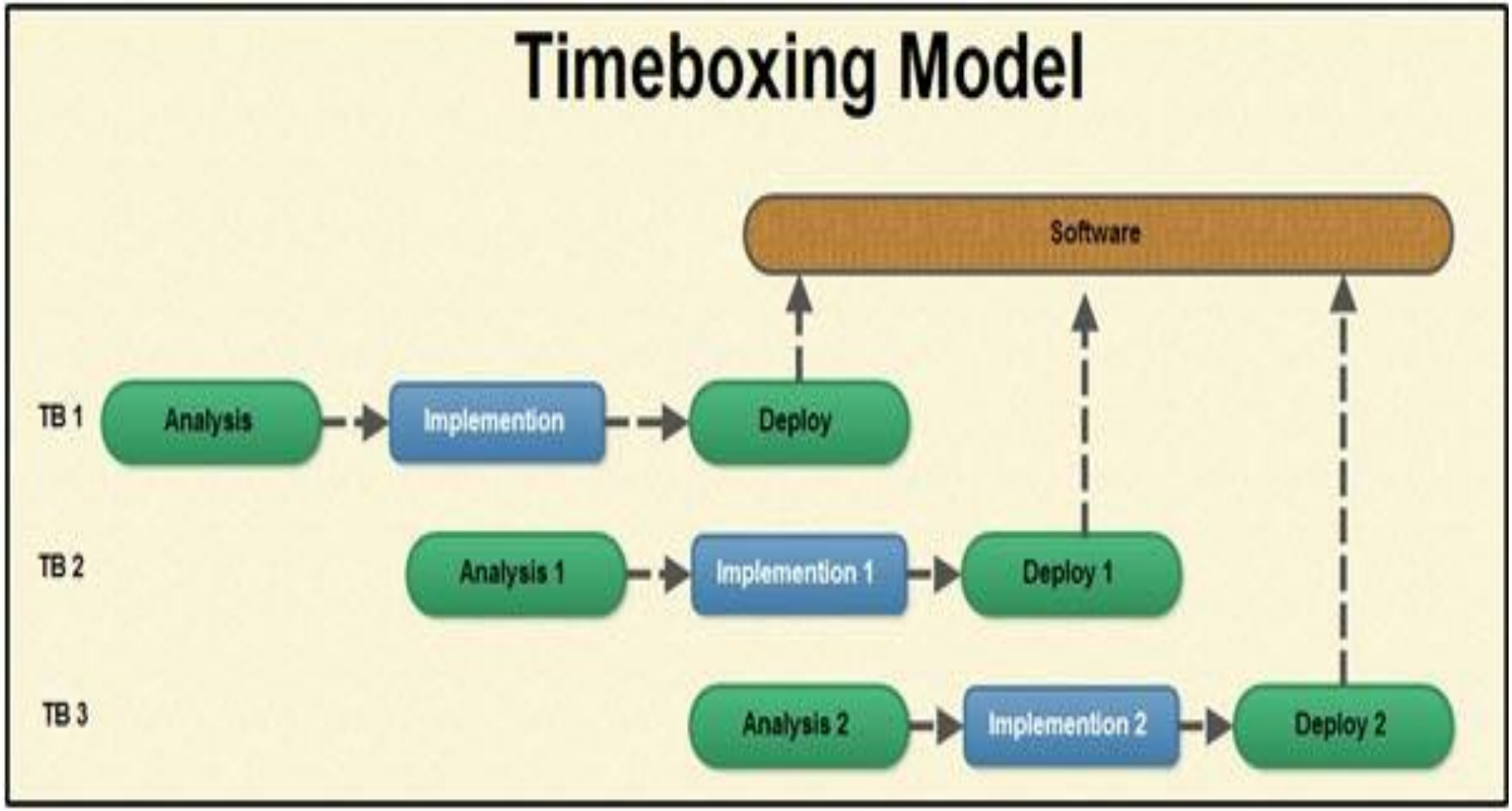
Timeboxing Process Model

- To speed up development, parallelism between the different iterations can be employed
- A new iteration commences before the system produced by the current iteration is released, and hence development of a new release happens in parallel with the development of the current release
- By starting an iteration before the previous iteration has completed, it is possible to reduce the average delivery time for iterations.

Timeboxing Process Model



Timeboxing Process Model



Timeboxing Process Model

- In the time boxing model, the basic unit of development is a time box, which is of fixed duration
- Since the duration is fixed, a key factor in selecting the requirements to be built in a time box is what can be fit into the time box
- This is in contrast to regular iterative approaches where the functionality is selected and then the time to deliver is determined
- The model requires that the duration of each stage is approximately the same

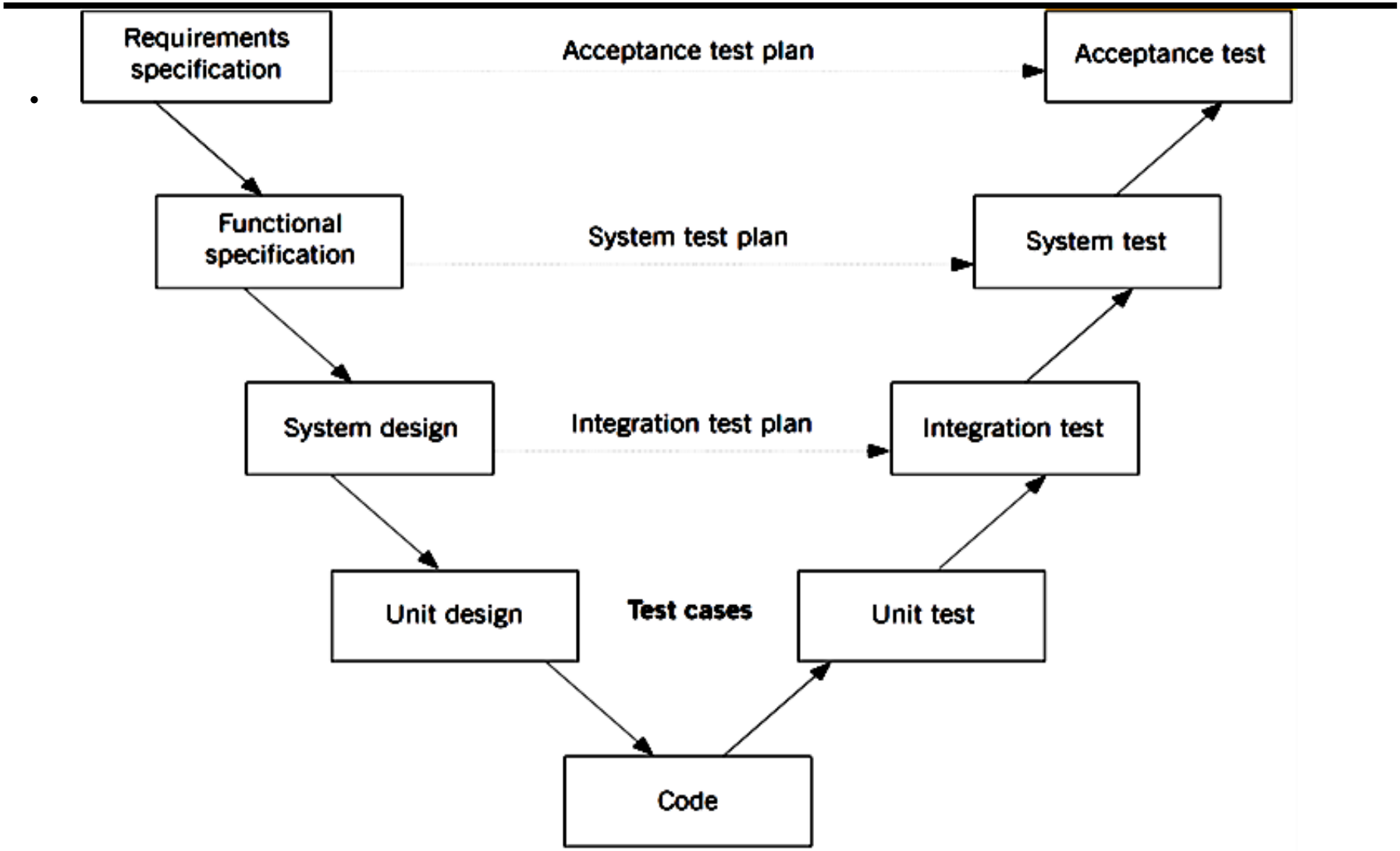
Timeboxing Process Model

- Having time-boxed iterations with stages of equal duration and having dedicated teams renders itself to pipelining of different iterations
- To illustrate the use of this model, consider a time box consisting of three stages: requirement specification, build, and deployment
- The requirement stage is executed by its team of analysts and ends with a prioritized list of requirements to be built in this iteration along with a high-level design

Timeboxing Process Model

- The build team develops the code for implementing the requirements, and performs the testing
- The tested code is then handed over to the deployment team, which performs pre-deployment tests, and then installs the system for use
- These three stages can be done in approximately equal time in an iteration
- Timeboxing is well suited for projects that require a large number of features to be developed in a short time around a stable architecture using stable technologies

V Process Model



V Process Model

- The V model is a variant of the waterfall model
- It represents a tacit recognition that there are testing activities occurring throughout the waterfall software life cycle model and not just during the software testing period
- For example, during requirements specification, the requirements are evaluated for testability and an STRS may be written
- This document would describe the strategy necessary for testing the requirements

V Process Model

- Testing is a full life-cycle activity and that it is important to constantly consider the testability of any software requirement and to design to allow for such testability