Software Engineering-BSCE-301L

Module 1:

Overview of Software Engineering

Dr . Saurabh Agrawal

Faculty Id: 20165

School of Computer Science and Engineering

VIT, Vellore-632014

Tamil Nadu, India

Outline

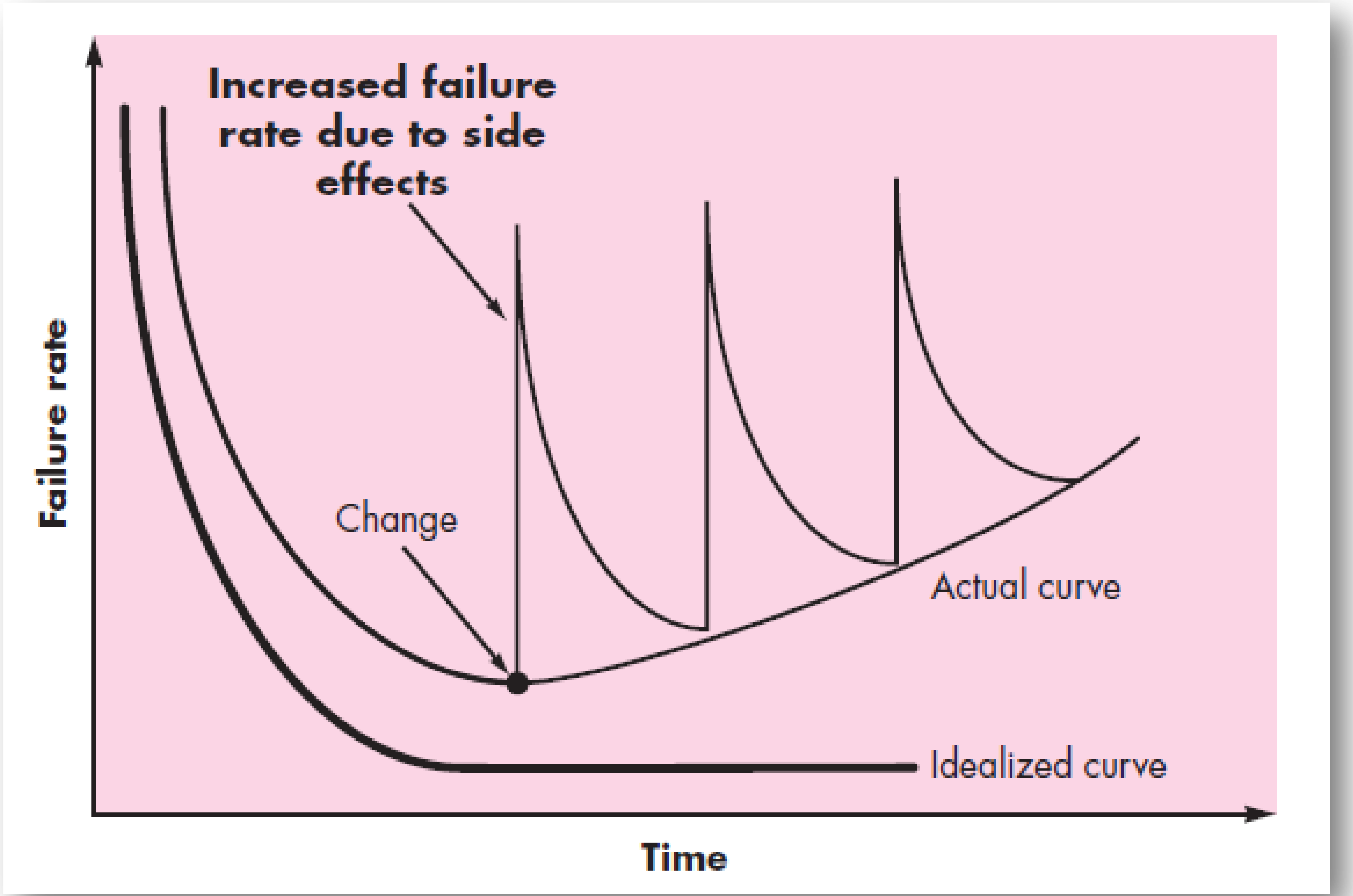
- ❑Nature of Software (L2)
- ❑Introduction to Software Engineering (L2)
- ❑Software Process, Project, Product (L2)
- ❑Process Models (Waterfall, Incremental, RAD) (L3)
- ❑Classical Evolutionary Models (Prototype, Spiral, and Concurrent) (L4)
- ❑Introduction to Agility and Agile Process (L5)
- ❑Principles of Agile Software Development framework (L6)
- ❑Extreme programming (XP) Process(L6)
- ❑Overview of System Engineering (L7)

Nature of Software

- ❑The software is instruction or computer program that when executed provide desired features, function, and performance.
- ❑A data structure that enables the program to adequately manipulate information and document that describe the operation and use of the program.
- ❑There are some **characteristics** of software as aforementioned.

1. **Functionality**
2. **Reliability**
3. **Usability**
4. **Efficiency**
5. **Maintainability**
6. **Portability**

Nature of Software



Nature of Software

Changing Nature of Software:

❑Following are the **Seven Broad Categories** of software are challenges for software engineers:

1. System Software: is a collection of programs written to service other programs. System software: such as compilers, editors, file management utilities.
2. Application Software: stand-alone programs for specific needs. This software are used to controls business needs. Ex: Transaction processing.
3. Embedded Software: This software resides within a system or product and it is used to implement and control features and functions from end user and system itself. This software performs limited function like keypad control for microwave oven.
4. Artificial Intelligence Software: makes use of nonnumeric algorithms to solve complex problems. Application within this area include robotics, pattern recognition, game playing.

Nature of Software

Changing Nature of Software:

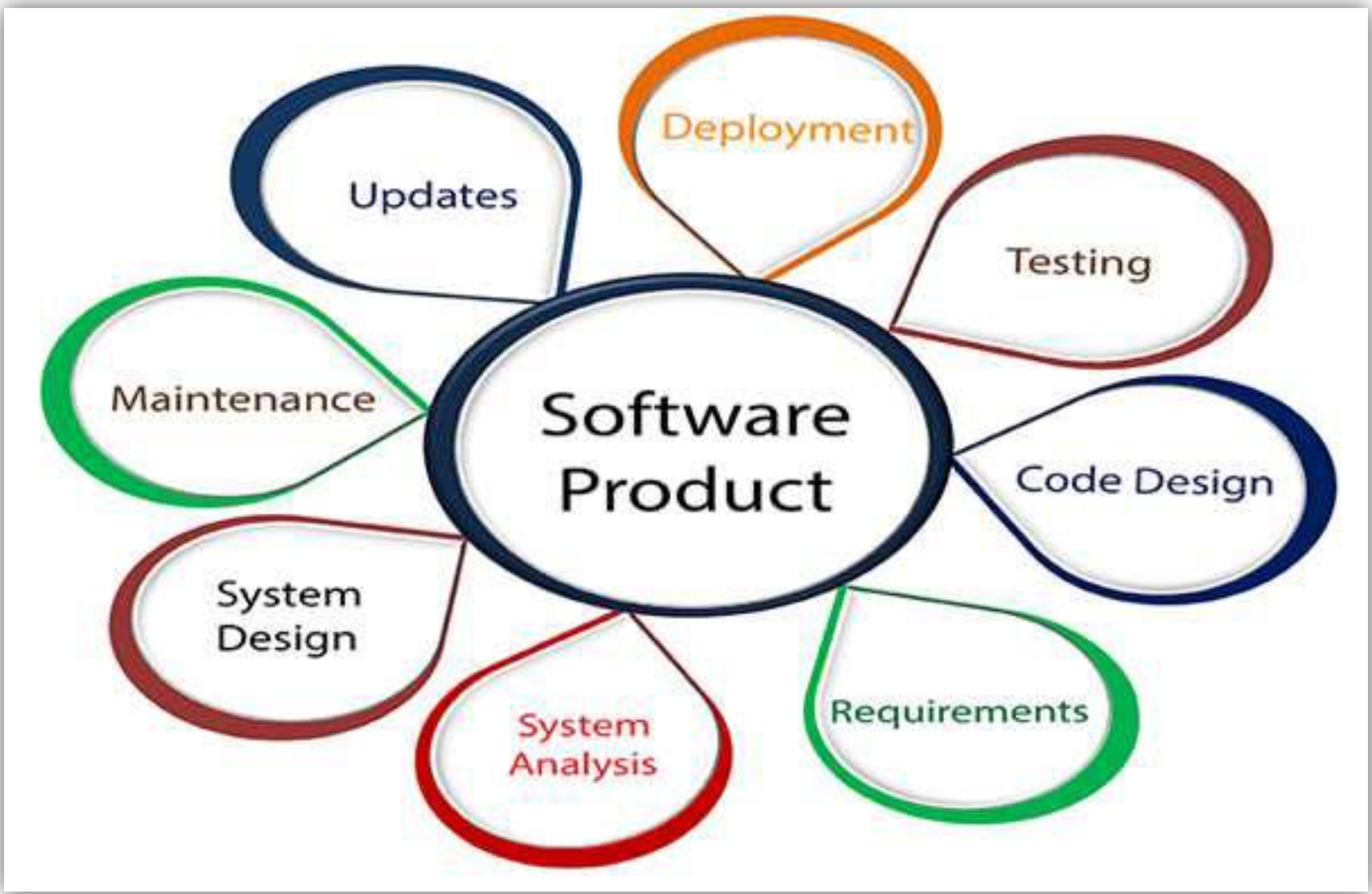
5. Engineering and Scientific Software: Engineering and scientific software have been characterized by "number crunching" algorithm.
6. Product-line Software: focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management) .
7. WebApps (Web applications) Network Centric Software: As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.

Introduction to Software Engineering

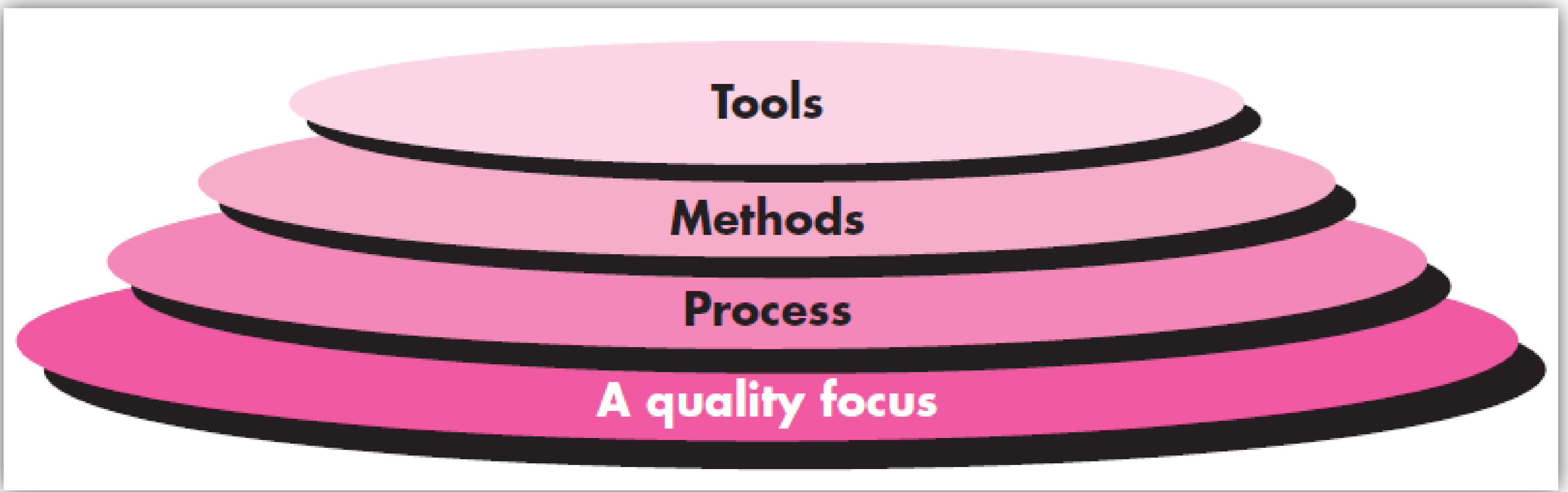
What is Software Engineering?

- The term software engineering is the product of two words, software, and engineering.
- The software is a collection of integrated programs.
- Software subsists of carefully-organized instructions and code written by developers on any of various particular computer languages.
- Computer programs and related documentation such as requirements, design models and user manuals.
- Engineering is the application of scientific and practical knowledge to invent, design, build, maintain, and improve frameworks, processes, etc.

Introduction to Software Engineering



Introduction to Software Engineering



Introduction to Software Engineering

Why is Software Engineering required?

- Software Engineering is required due to the following reasons:
  - **To manage Large software**
  - **For more Scalability**
  - **Cost Management**
  - **To manage the dynamic nature of software**
  - **For better quality Management**

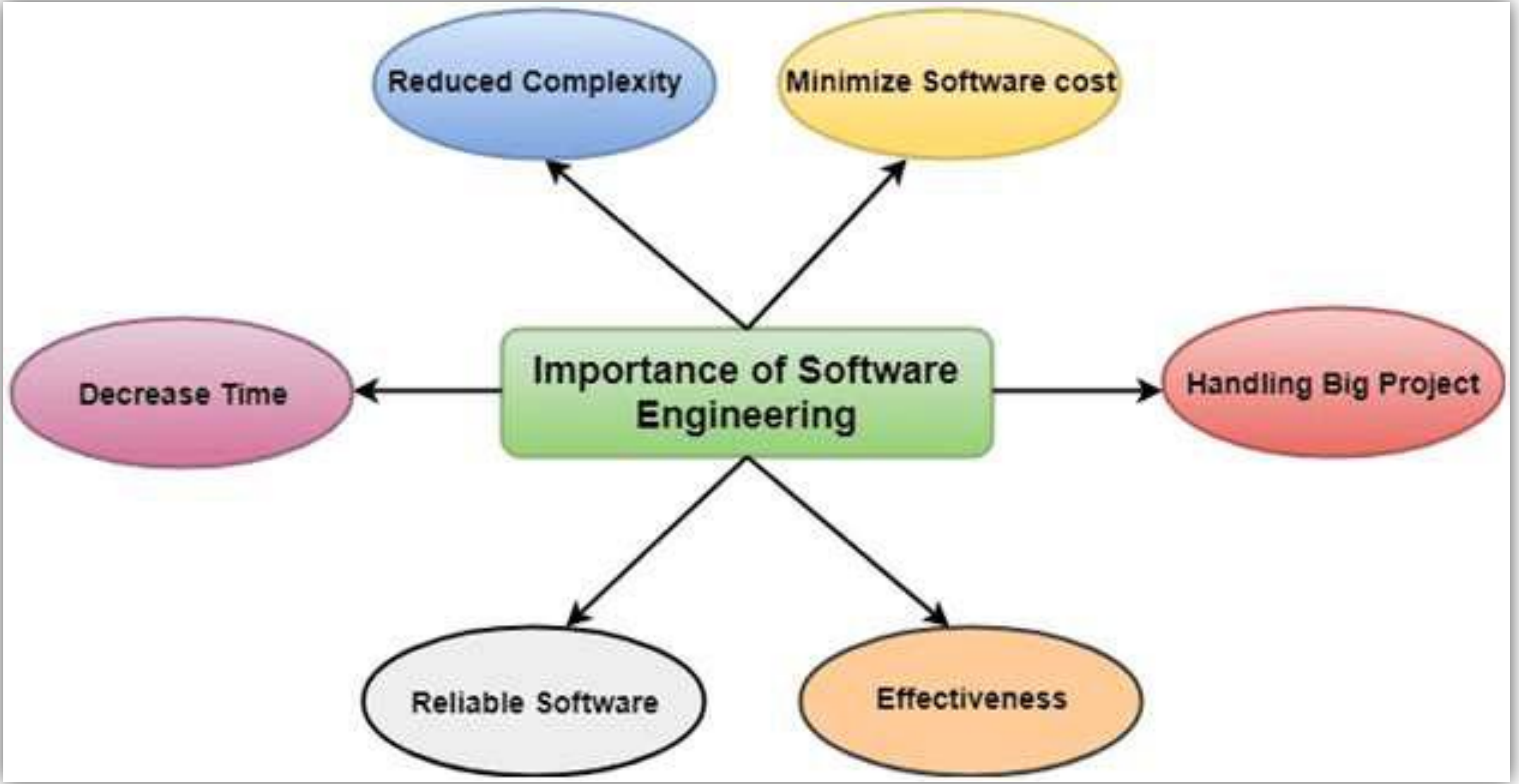
Introduction to Software Engineering

Need of Software Engineering

- The necessity of software engineering appears because of a higher rate of progress in user requirements and the environment on which the program is working.
  - **Huge Programming:** It is simpler to manufacture a wall than to a house or building, similarly, as the measure of programming become extensive engineering has to step to give it a scientific process.
  - **Adaptability:** If the software procedure were not based on scientific and engineering ideas, it would be simpler to re-create new software than to scale an existing one.
  - **Cost:** As the hardware industry has demonstrated its skills and huge manufacturing has let down the cost of computer and electronic hardware, but cost of programming remains high if the proper process is not adapted.
  - **Dynamic Nature:** The continually growing and adapting nature of programming hugely depends upon the environment in which the client works. If the quality of the software is continually changing, new upgrades need to be done in the existing one.
  - **Quality Management:** Better procedure of software development provides a better and quality software product.

Introduction to Software Engineering

Importance of Software Engineering



Introduction to Software Engineering

Importance of Software Engineering

1.Reduces Complexity: Big software is always complicated and challenging to progress. Software engineering has a great solution to reduce the complication. Software engineering divides big problems into various small issues. And then start solving each small issue one by one. All these small problems are solved independently to each other.

2.To Minimize Software Cost: A lot of manpower is required to develop software with a large number of codes. But in software engineering, programmers project everything and decrease all those things that are not needed. In turn, the cost for software productions becomes less as compared to any software that does not use software engineering method.

3.To Decrease Time: Anything that is not made according to the project always wastes time. And if you are making great software, then you may need to run many codes to get the definitive running code. This is a very time-consuming procedure, and if it is not well handled, then this can take a lot of time. So if you are making your software according to the software engineering method, then it will decrease a lot of time.

Introduction to Software Engineering

Importance of Software Engineering

4.Handling Big Projects: Big projects are not done in a couple of days, and they need lots of patience, planning, and management. And to invest six and seven months of any company, it requires heaps of planning, direction, testing, and maintenance. No one can say that he has given four months of a company to the task, and the project is still in its first stage. Because the company has provided many resources to the plan and it should be completed. So to handle a big project without any problem, the company has to go for a software engineering method.

5.Reliable Software: Software should be secure, means if you have delivered the software, then it should work for at least its given time or subscription. And if any bugs come in the software, the company is responsible for solving all these bugs. Because in software engineering, testing and maintenance are given, so there is no worry of its reliability.

6.Effectiveness: Effectiveness comes if anything has made according to the standards. Software standards are the big target of companies to make it more effective.

Introduction to Software Engineering (Experiential Learning)

Question: **What is the difference between “Program” and “Software”?**

Software Process

- ❑A **process is a collection of** activities, actions, and tasks that are performed when some work product is to be created.
- ❑An activity strives to achieve a broad objective (communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.
- ❑An action (architectural design) encompasses a set of tasks that produce a major work product (an architectural design model).
- ❑A task focuses on a small, but well-defined objective that produces a tangible outcome.
- ❑In the context of software engineering, a process is *not a rigid prescription for how* to build computer software.
- ❑Rather, it is an adaptable approach that enables the people doing the work (the software team) to pick and choose the appropriate set of work actions and tasks.
- ❑The intent is always to deliver software in a timely manner and with sufficient quality to satisfy those who have sponsored its creation and those who will use it.

Software Process

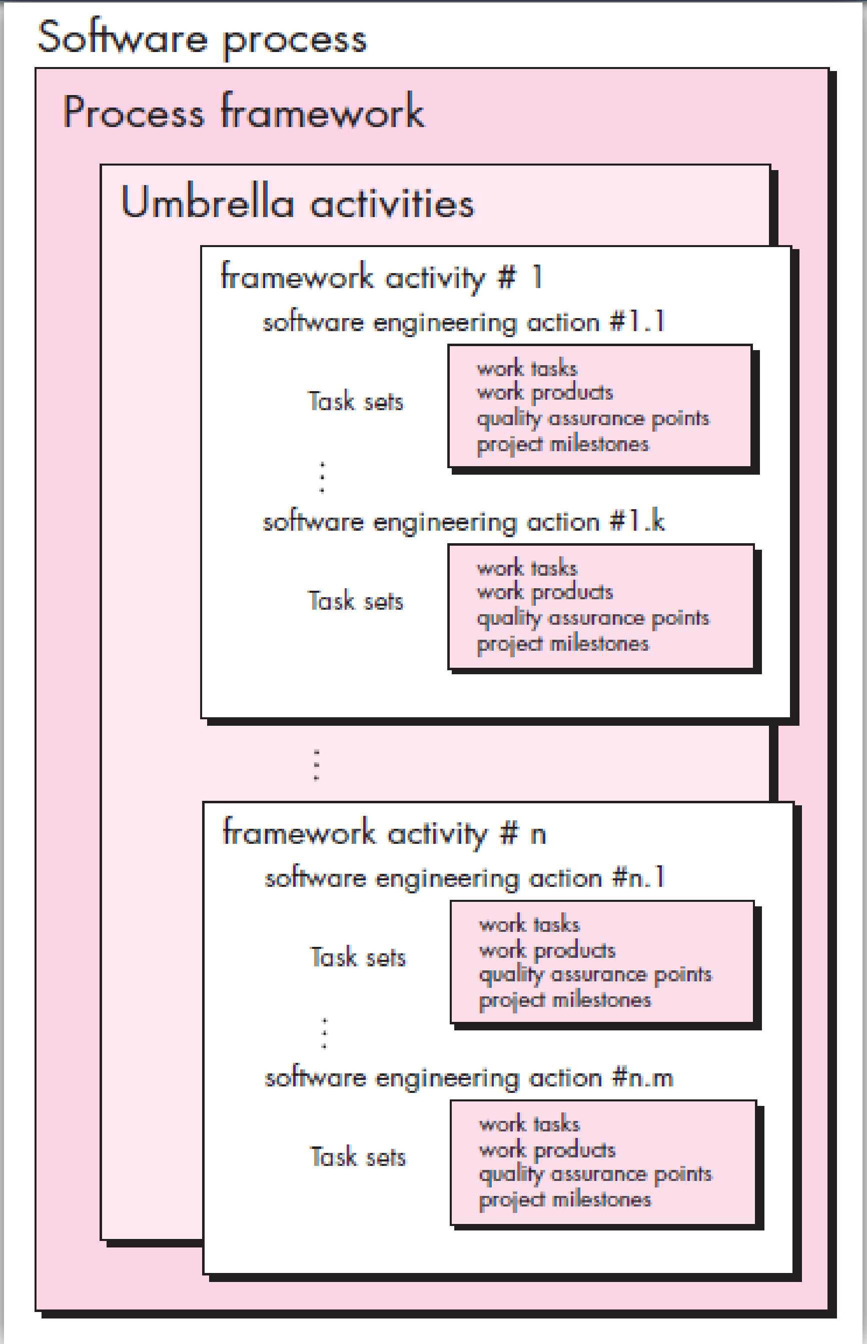
❑A **process framework** establishes the foundation for a complete software engineering process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.

❑ In addition, the **process framework** encompasses **a set of umbrella activities** that are applicable across the entire software process.

❑A generic process framework for software engineering encompasses five activities:

1. **Communication**
2. **Planning**
3. **Modeling**
4. **Construction**
5. **Deployment**

Software Process



Software Process

1. **Communication:** Before any technical work can commence, it is critically important to communicate and collaborate with the customer and other stakeholders. The intent is to understand stakeholders' objectives for the project and to gather requirements that help define software features and functions.
2. **Planning:** A software project is a complicated journey, and the planning activity creates a "map" that helps guide the team as it makes the journey. The map—called a software project plan—defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.
3. **Modeling:** You create a "sketch" of the thing so that you'll understand the big picture—what it will look like architecturally, how the constituent parts fit together, and many other characteristics. If required, you refine the sketch into greater and greater detail in an effort to better understand the problem and how you're going to solve it. A software engineer does the same thing by creating models to better understand software requirements and the design that will achieve those requirements.

Software Process

4. **Construction:** This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.
5. **Deployment:** The software (as a complete entity or as a partially completed increment) is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

Software Process

- ❑ Software engineering process framework activities are complemented by a number of umbrella activities.
- ❑ In general, umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk.
- ❑ Typical umbrella activities include:

1. **Software Project Tracking and Control:** Allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.
2. **Risk Management:** Assesses risks that may affect the outcome of the project or the quality of the product.
3. **Software Quality Assurance:** Defines and conducts the activities required to ensure software quality.
4. **Technical reviews:** Assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.

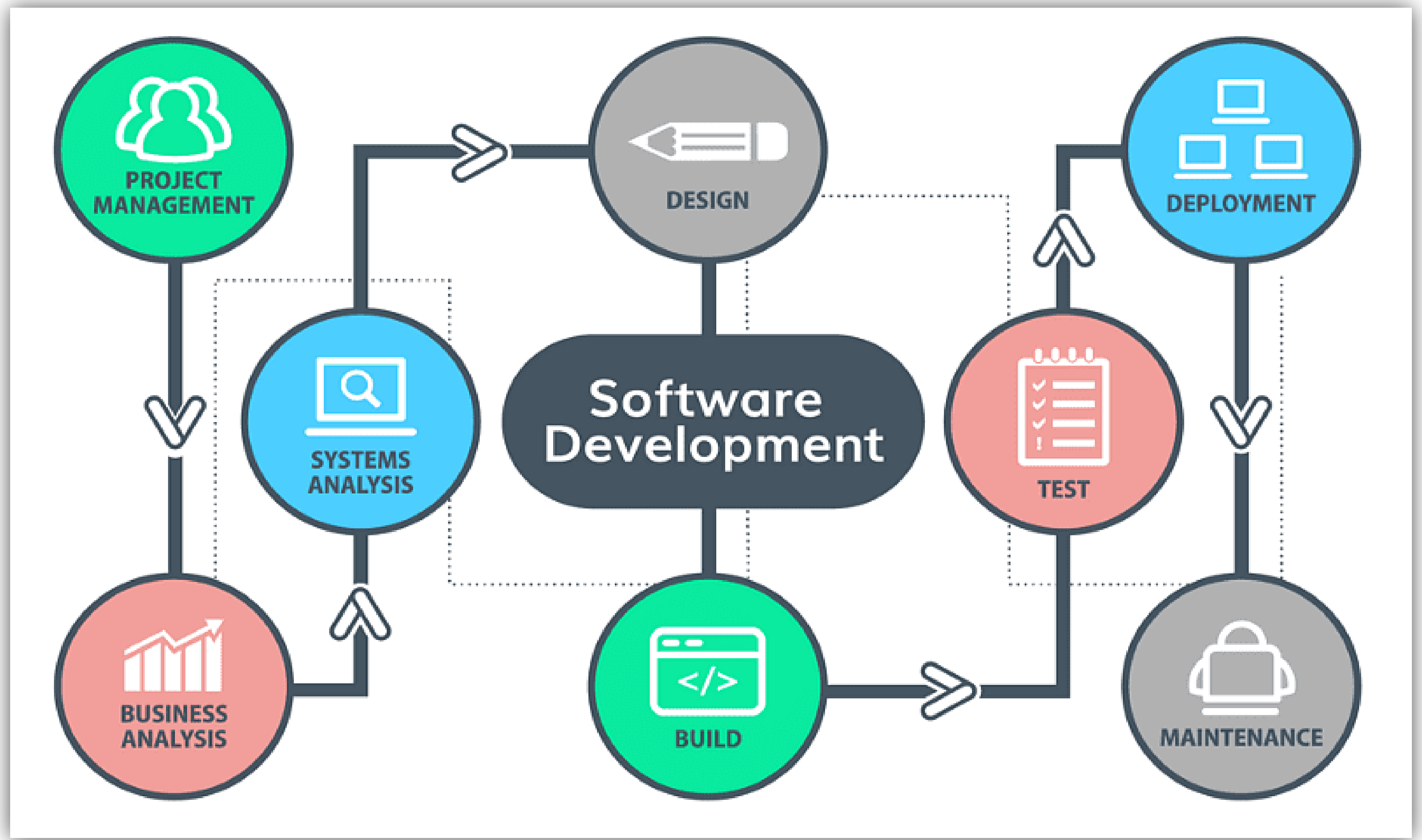
Software Process

5. **Measurement:** Defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs; can be used in conjunction with all other framework and umbrella activities.
6. **Software Configuration Management:** Manages the effects of change throughout the software process.
7. **Reusability Management:** Defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.
8. **Work Product Preparation and Production:** Encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

Software Project

- A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.
- Also known as a software project comprises the steps involved in making a product before it is actually available to the market.
- The project can be handled by people which are as less as one person to the involvement of a lot of people.
- These are usually assigned by an enterprise and are undertaken to form a new product that has not already been made.

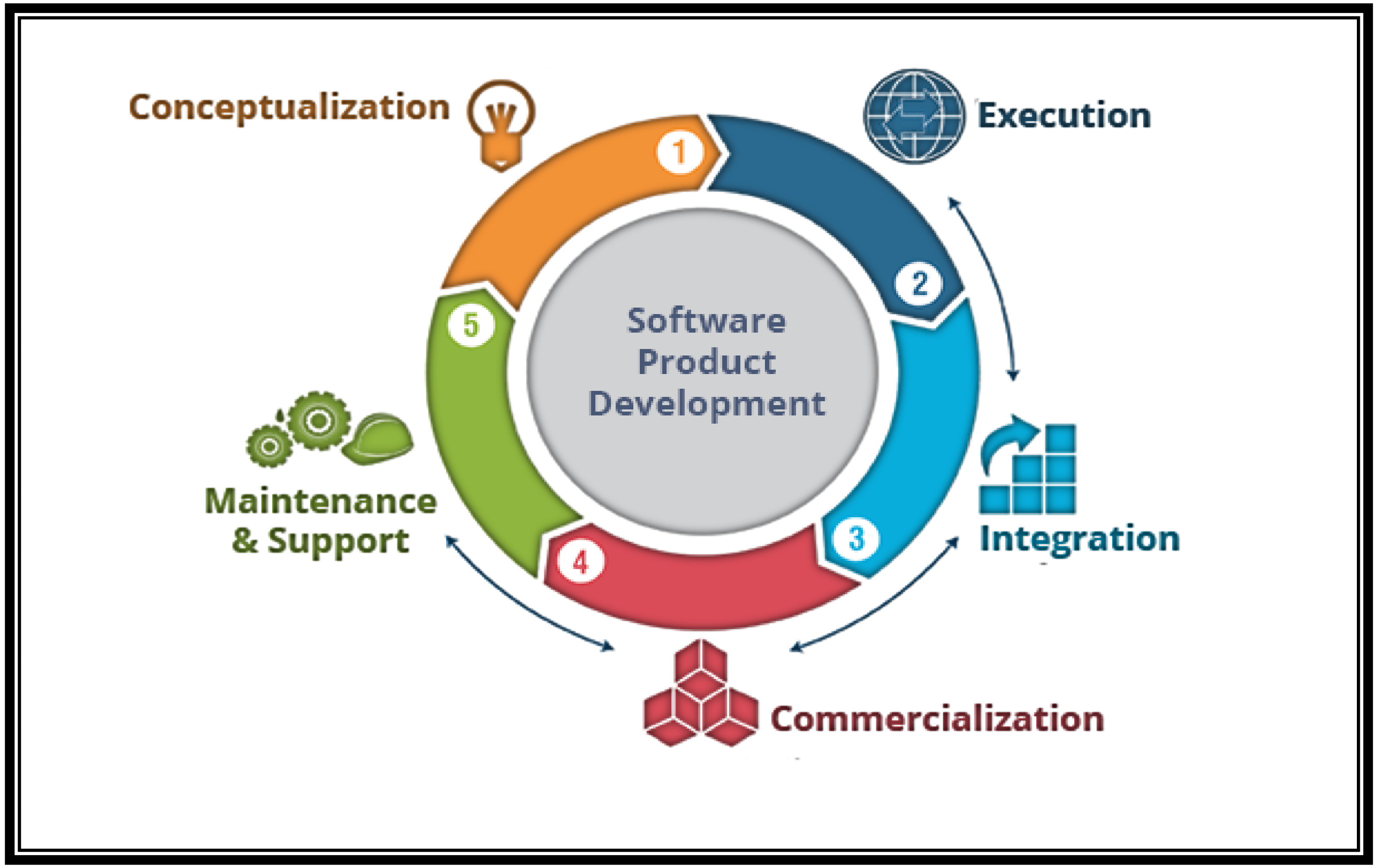
Software Project



Software Product

- The product is the final outcome of the software development process.
- It is the software application or system that is being built or maintained.
- A product is built on the customer's requirements/requests.
- In the context of software development, a product consists of several components such as requirement specifications, design and test documents, user manuals, etc.
- To develop a desired software product, the software development team and the customer should define the purpose and scope of the product.
- The "purpose" gives the information about the objective of the product, and the "scope" gives the details about fundamental data, functions, and behavior of the product.

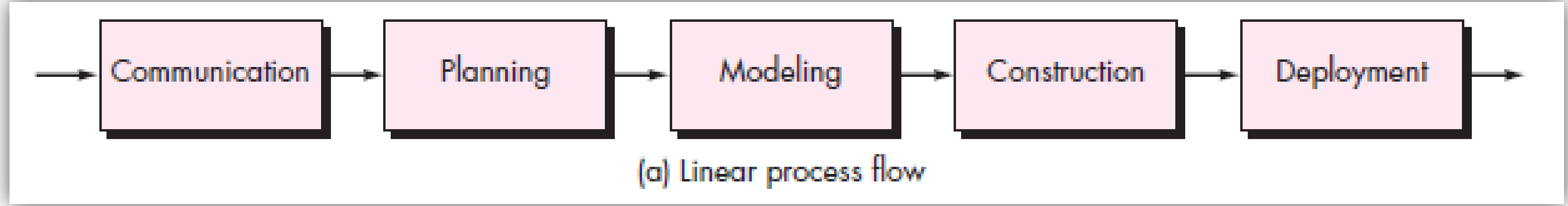
Software Product



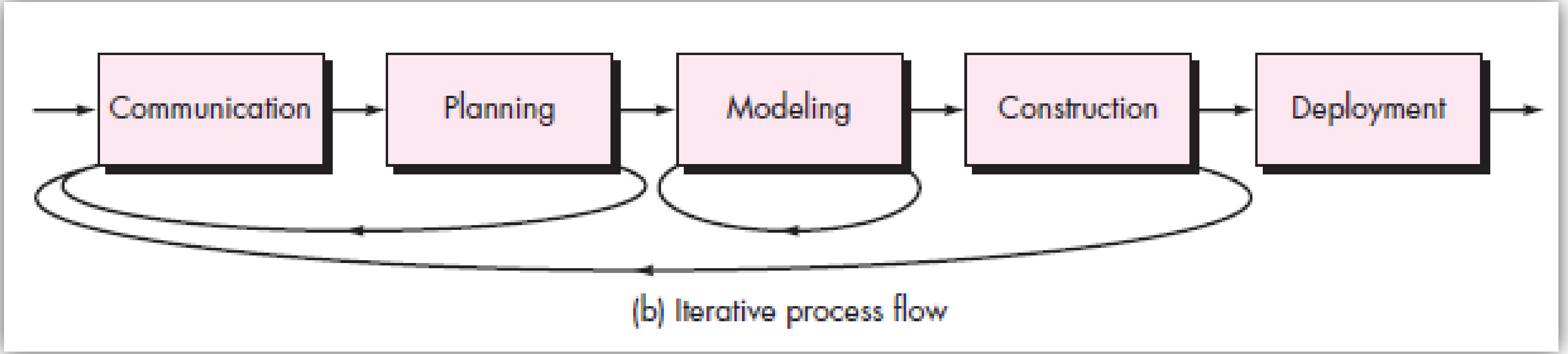
Process Models

- A generic process framework for software engineering defines five framework activities—**communication, planning, modeling, construction, and deployment.**
- Important aspect of the software process has been process flow.
- Process flow describes how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to sequence and time.
- A **linear process** flow executes each of the five framework activities in sequence, beginning with communication and culminating with deployment.
- An **iterative process** flow repeats one or more of the activities before proceeding to the next.
- An **evolutionary process** flow executes the activities in a "circular" manner. Each circuit through the five activities leads to a more complete version of the software.
- A **parallel process** flow executes one or more activities in parallel with other activities (e.g., modeling for one aspect of the software might be executed in parallel with construction of another aspect of the software).

Process Models

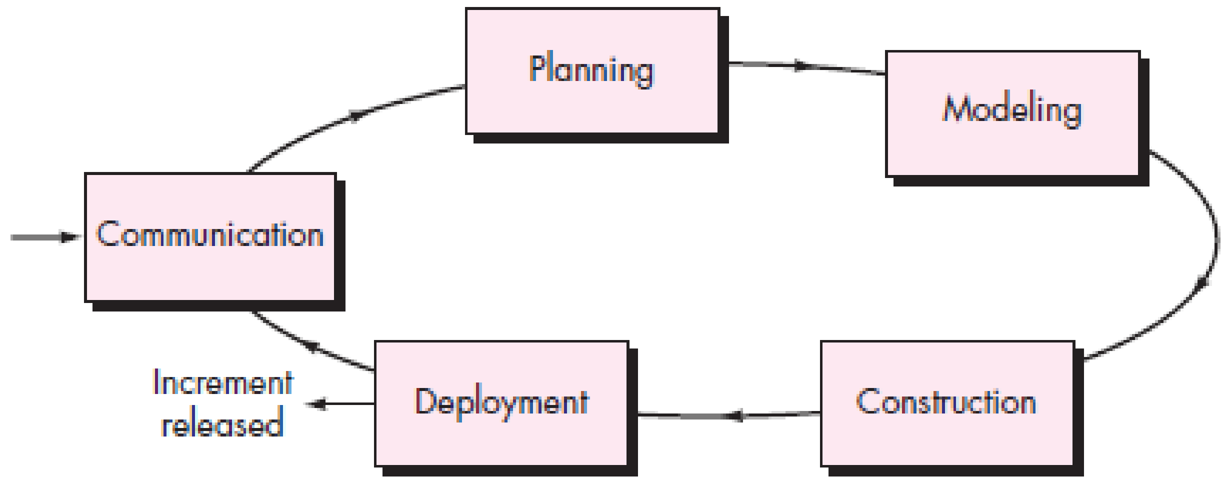


(a) Linear process flow

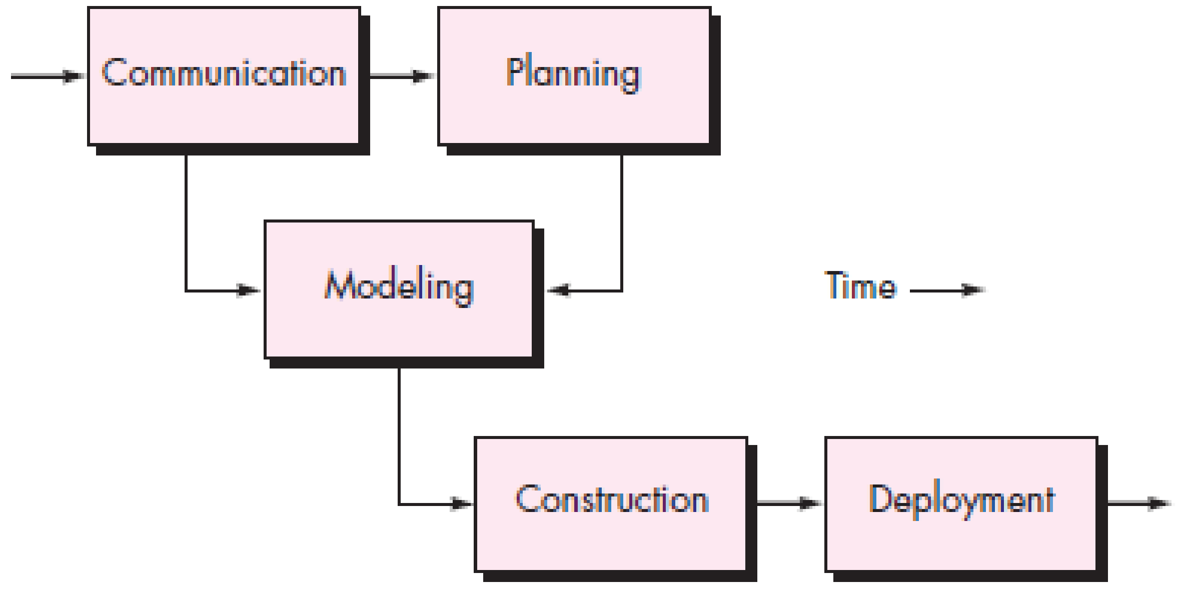


(b) Iterative process flow

Process Models



(c) Evolutionary process flow



(d) Parallel process flow

Perspective Process Models (Waterfall, Incremental, RAD)

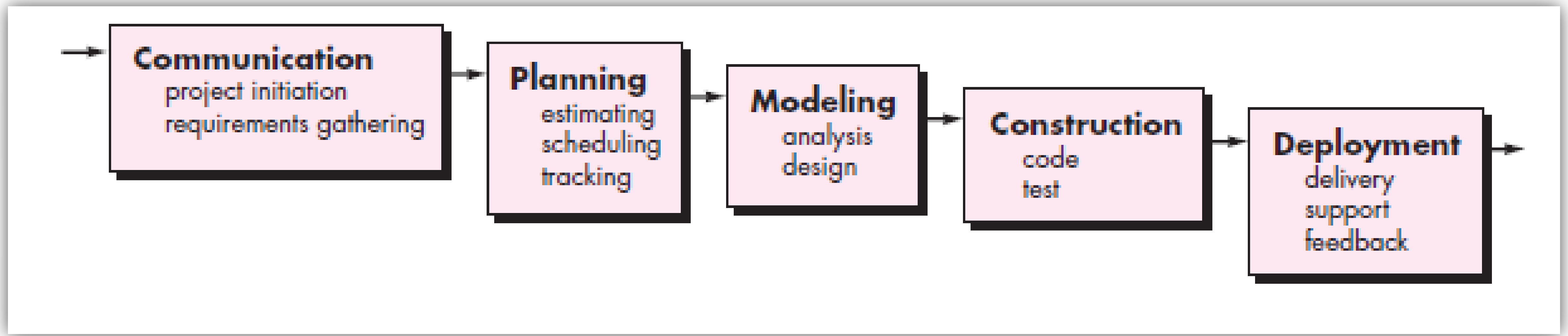
- ❑ Prescriptive process models were originally proposed to bring order to the chaos of software development.
- ❑ History has indicated that these traditional models have brought a certain amount of useful structure to software engineering work and have provided a reasonably effective road map for software teams.
- ❑ However, software engineering work and the product that it produces remain on “the edge of chaos.”
- ❑ All software process models can accommodate the generic framework activities.
- ❑ But each applies a different emphasis to these activities and defines a process flow that invokes each framework activity (as well as software engineering actions and tasks) in a different manner.
- ❑ Following are some popular perspective process models:

1. **Waterfall**
2. **Incremental**
3. **Rapid Application Development (RAD)**

Perspective Process Models (Waterfall-V1)

- There are times when the requirements for a problem are well understood when work flows from **communication through deployment in a reasonably linear fashion.**
- This situation is sometimes encountered when well-defined adaptations or enhancements to an existing system must be made.
- It may also occur in a limited number of new development efforts, but only when requirements are well defined and reasonably stable.
- The waterfall model, sometimes called the classic life cycle, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software.

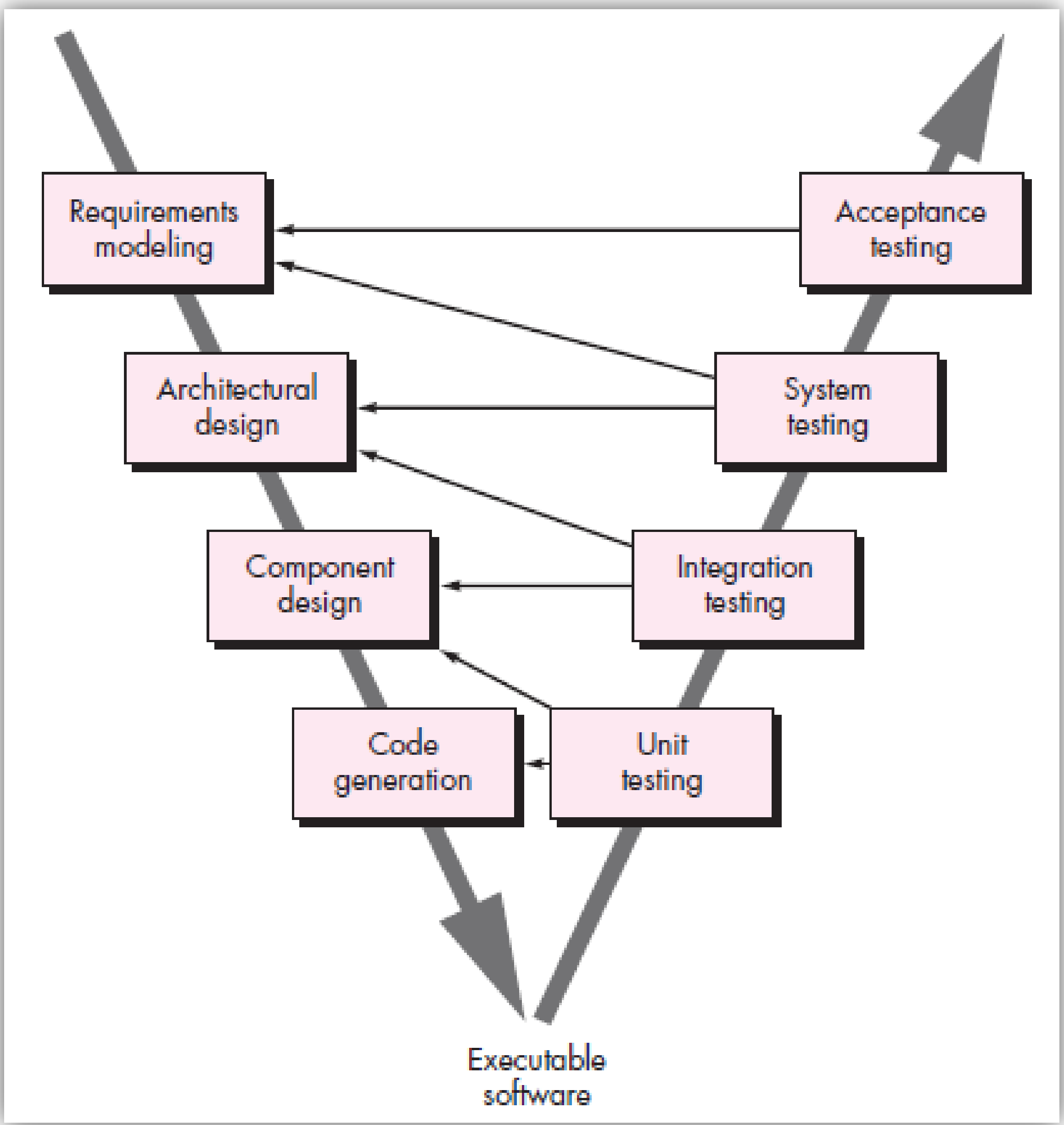
Perspective Process Models (Waterfall-V1)



Perspective Process Models (Waterfall-V1)

- A variation in the representation of the waterfall model is called the *V-model.*
- V-model depicts the relationship of quality assurance actions to the actions associated with communication, modeling, and early construction activities.
- As a software team moves down the left side of the V, basic problem requirements are refined into progressively more detailed and technical representations of the problem and its solution.
- Once code has been generated, the team moves up the right side of the V, essentially performing a series of tests (quality assurance actions) that validate each of the models created as the team moved down the left side.
- In reality, there is no fundamental difference between the classic life cycle and the V-model.
- The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work.

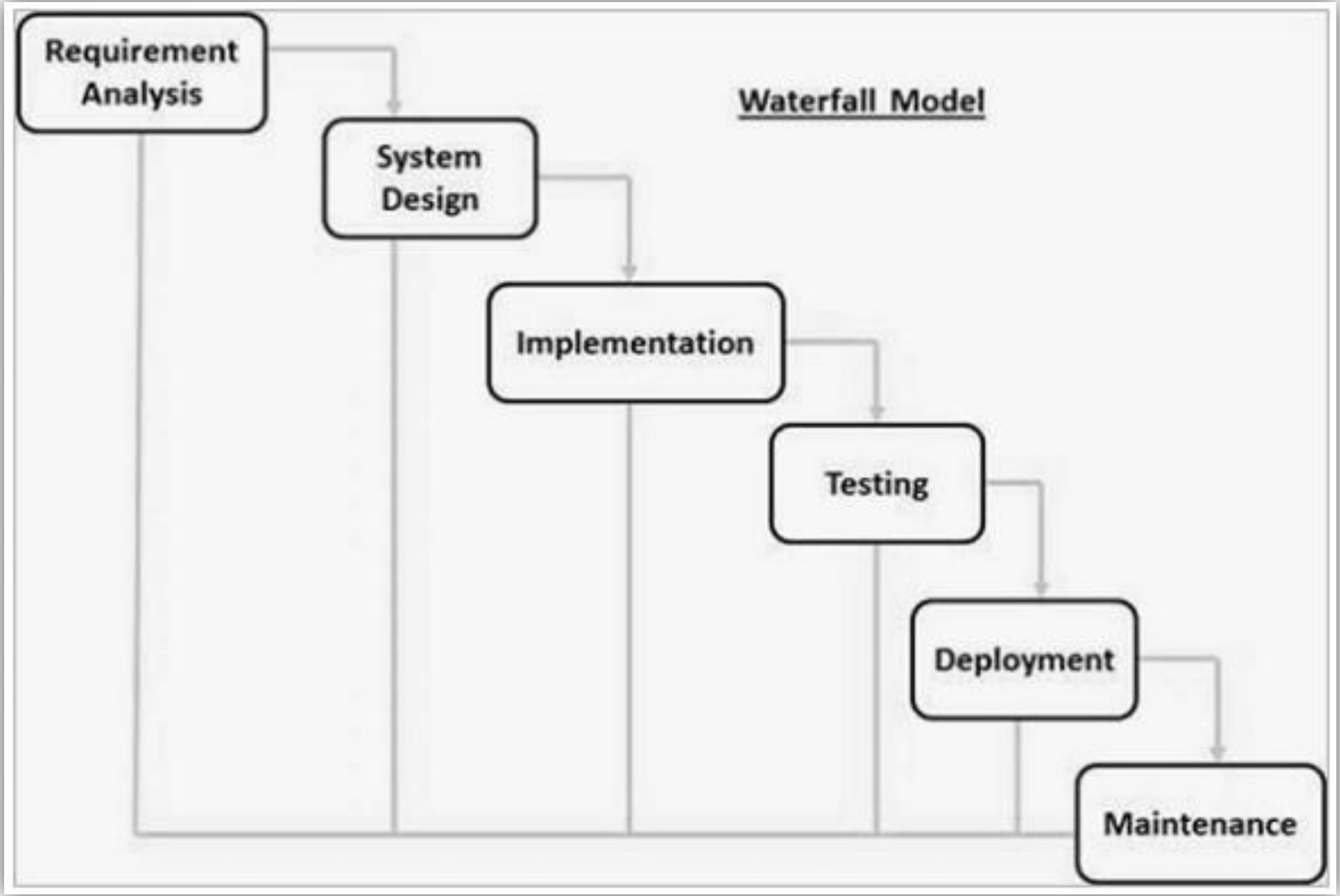
Perspective Process Models (Waterfall-V1)



Perspective Process Models (Waterfall-V2)

- The Waterfall Model was the first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**.
- It is very simple to understand and use.
- In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.
- The Waterfall model is the earliest SDLC approach that was used for software development.
- The waterfall Model illustrates the software development process in a linear sequential flow.
- This means that any phase in the development process begins only if the previous phase is complete.
- In this waterfall model, the phases do not overlap.

Perspective Process Models (Waterfall-V2)



Perspective Process Models (Waterfall-V2)

1. **Requirement Gathering and Analysis:** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
2. **System Design:** The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
3. **Implementation:** With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
4. **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

Perspective Process Models (Waterfall-V2)

5. **Deployment of system:** Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
6. **Maintenance:** There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Perspective Process Models (Waterfall-V2)

Waterfall Model: Application

- ❑ Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors.
- ❑ Some situations where the use of Waterfall model is most appropriate are:
- ▪ Requirements are very well documented, clear and fixed.
- ▪ Product definition is stable.
- ▪ Technology is understood and is not dynamic.
- ▪ There are no ambiguous requirements.
- ▪ Ample resources with required expertise are available to support the product.
- ▪ The project is short.

Perspective Process Models (Waterfall-V2)

❑ Advantages

- Easy to manage due to the rigidity of the model.
- Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.
- Simple and easy to understand and use

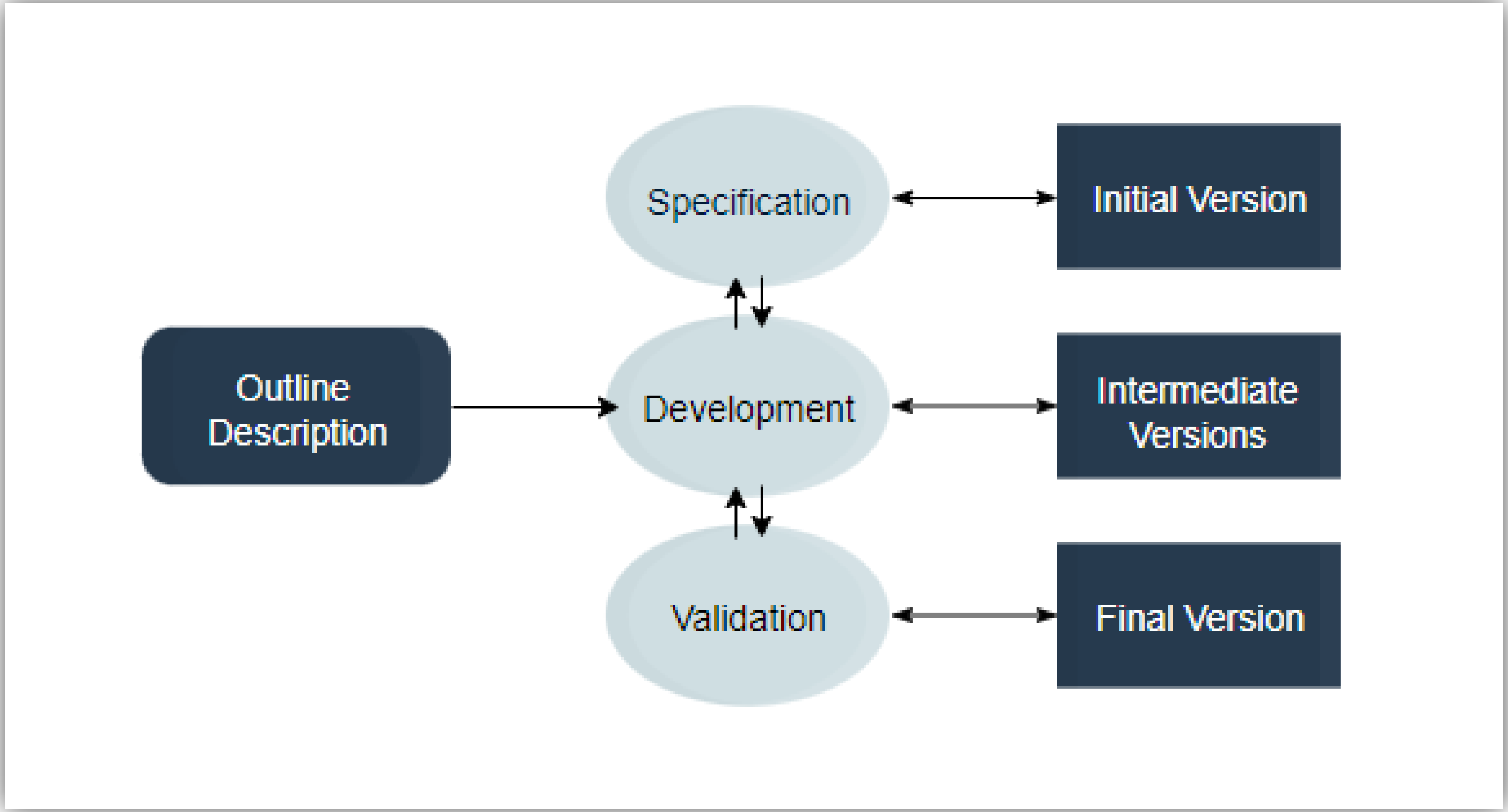
❑ Disadvantages

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.

Perspective Process Models (Incremental)

- The incremental model divides the system’s functionality into **small increments** that are delivered one after the other in quick succession.
- The most important functionality is implemented in the initial increments.
- The subsequent increments expand on the previous ones until everything has been updated and implemented.
- Incremental development is based on developing an initial implementation, exposing it to user feedback, and evolving it through new versions.
- The process’ activities are interwoven by feedback.
- Each iteration passes through the requirements, design, coding, and testing stages.
- The incremental model lets stakeholders and developers see results with the first increment. If the stakeholders don’t like anything, everyone finds out a lot sooner.
- It is efficient as the developers only focus on what is important and bugs are fixed as they arise, but you need a **clear and complete definition** of the whole system before you start.
- The incremental model is great for projects that have loosely-coupled parts and projects with complete and clear requirements.

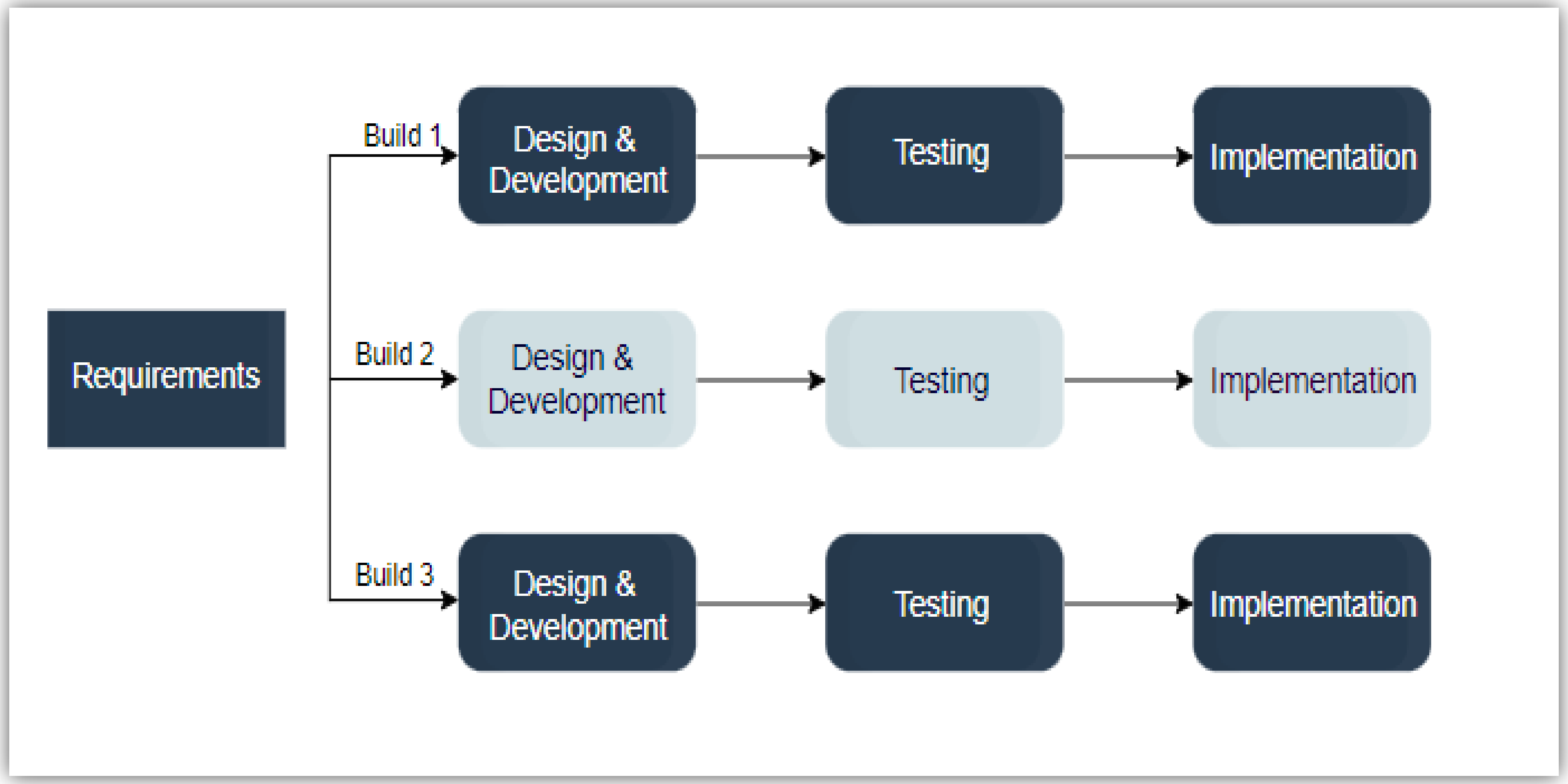
Perspective Process Models (Incremental)



Perspective Process Models (Incremental-Iterative Model-Version Control)

- The iterative development model develops a system through **building small portions** of all the features.
- This helps to meet initial scope quickly and release it for feedback.
- In the iterative model, you start off by implementing a small set of the software requirements.
- These are then **enhanced iteratively** in the evolving versions until the system is completed.
- This process model starts with part of the software, which is then implemented and reviewed to identify further requirements.
- Like the incremental model, the iterative model allows you to see the results at the early stages of development.
- This makes it easy to identify and **fix any functional or design flaws**. It also makes it easier to manage risk and change requirements.
- The deadline and budget may change throughout the development process, especially for large complex projects.
- The iterative model is a good choice for large software that can be **easily broken down into modules**.

Perspective Process Models (Incremental-Iterative Model-Version Control)



Perspective Process Models (Incremental-Iterative Model-Version Control)

- **Various phases of incremental model are as follows:**

1. Requirement analysis: In this phase, product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team.

2. Design & Development: In this phase, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

3. Testing: In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

4. Implementation: Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

Perspective Process Models (Incremental-Iterative Model-Version Control)

- **When to use Incremental Model:**

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Perspective Process Models (Incremental-Iterative Model-Version Control)

❑**Advantages:**

- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

❑**Disadvantages:**

- Need for good planning
- Total Cost is high.
- Well defined module interfaces are needed.

Perspective Process Models (RAD → Rapid Application Development)

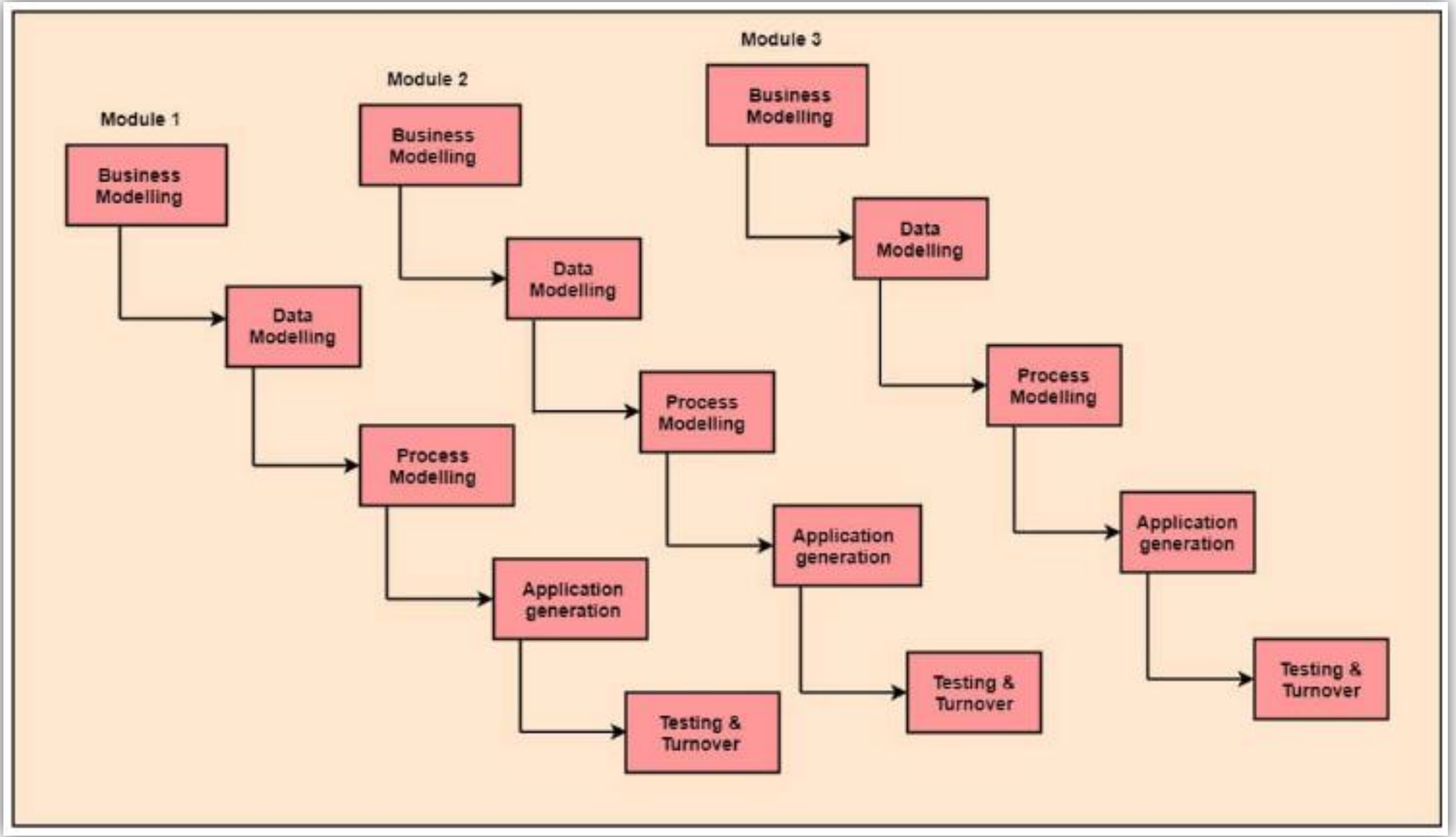
❑RAD is a linear sequential software development process model that emphasizes a concise development cycle using an element based construction approach.

❑If the requirements are well understood and described, and the project scope is a constraint, the RAD process enables a development team to create a fully functional system within a concise time period.

❑RAD (Rapid Application Development) is a concept that products can be developed faster and of higher quality through:

- Gathering requirements using workshops or focus groups
- Prototyping and early, reiterative user testing of designs
- The re-use of software components
- A rigidly paced schedule that refers design improvements to the next product version
- Less formality in reviews and other team communication

Perspective Process Models (RAD → Rapid Application Development)



Perspective Process Models (RAD → Rapid Application Development)

❑**Various phases of RAD are as follows:**

1.Business Modelling: Information flow among business functions is defined by answering questions like what data drives the business process, what data is generated, who generates it, where does the information go, who process.

2. Data Modelling: Data collected from business modeling is refined into a set of data objects that are needed to support the business. The attributes are identified, and the relation between these data objects is defined.

3. Process Modelling: The information object defined in the data modeling phase are transformed to achieve the data flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

4. Application Generation: Automated tools are used to facilitate construction of the software; even they use the 4th GL techniques.

5.Testing & Turnover: Many of the programming components have already been tested since RAD emphasis reuse. This reduces the overall testing time. But the new part must be tested, and all interfaces must be fully exercised.

Perspective Process Models (RAD → Rapid Application Development)

❑When to use RAD Model:

- When the system should need to create the project that modularizes in a short span time (2-3 months).
- When the requirements are well-known.
- When the technical risk is limited.
- When there's a necessity to make a system, which modularized in 2-3 months of period.
- It should be used only if the budget allows the use of automatic code generating tools.

Perspective Process Models (RAD → Rapid Application Development)

❑Advantages:

- This model is flexible for change.
- In this model, changes are adoptable.
- Each phase in RAD brings highest priority functionality to the customer.
- It reduced development time.
- It increases the reusability of features.

❑Disadvantages:

- It required highly skilled designers.
- All application is not compatible with RAD.
- For smaller projects, we cannot use the RAD model.
- On the high technical risk, it's not suitable.
- Required user involvement.

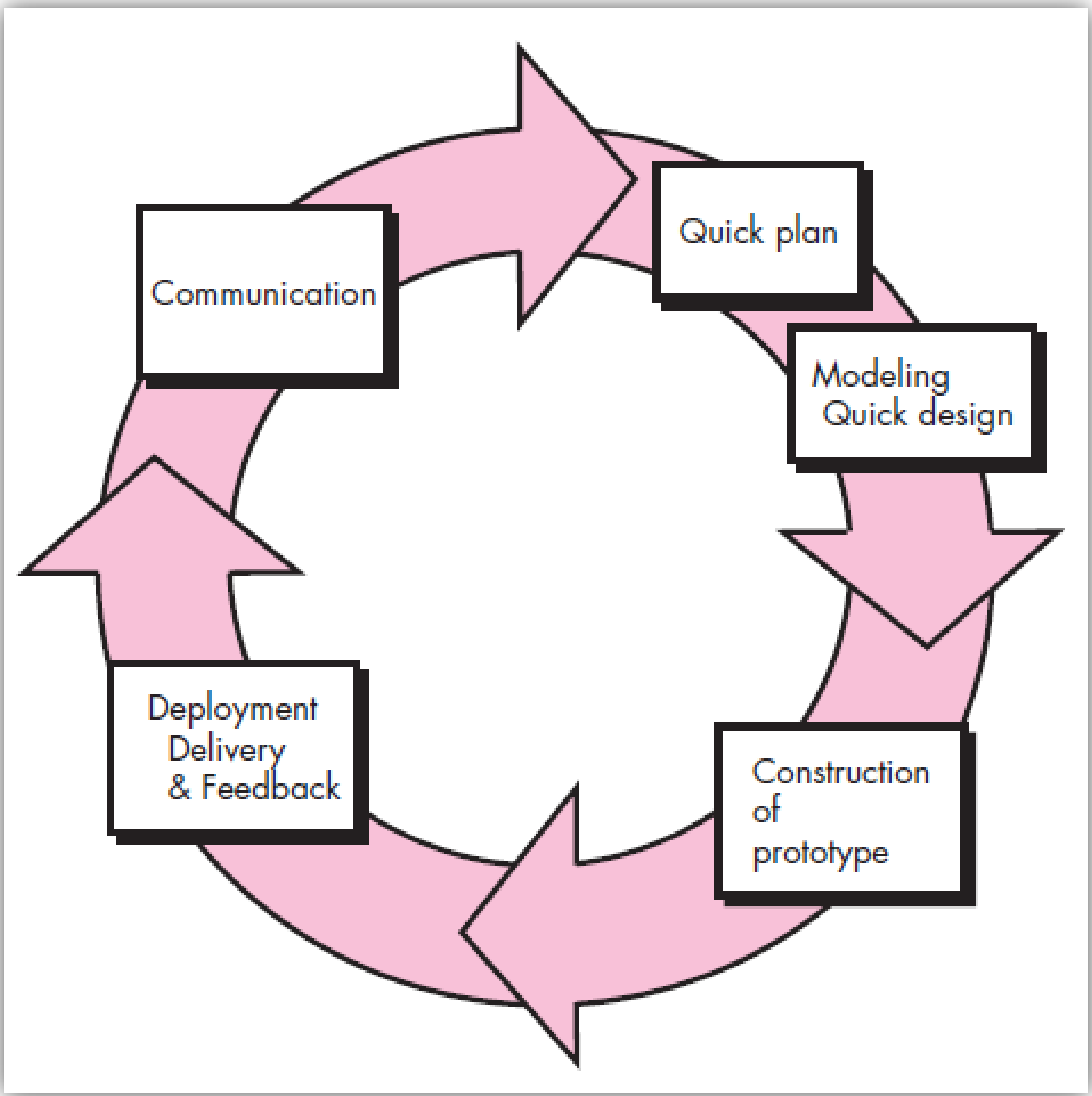
Classical Evolutionary Models (Prototype, Spiral)

- ❑ Software, like all complex systems, evolves over a period of time.
- ❑ Business and product requirements often change as development proceeds, making a straight line path to an end product unrealistic.
- ❑ Tight market deadlines make completion of a comprehensive software product impossible, but a limited version must be introduced to meet competitive or business pressure.
- ❑A set of core product or system requirements is well understood, but the details of product or system extensions have yet to be defined.
- ❑In these and similar situations, you need a process model that has been explicitly designed to accommodate a product that evolves over time.
- ❑Evolutionary models are iterative and they are characterized in a manner that enables you to develop increasingly more complete versions of the software.
- ❑Two common evolutionary process models are :
- ❑**Prototype** and **Spiral**

Classical Evolutionary Models (Prototype)

- ❑The prototyping paradigm begins with communication.
- ❑You meet with other stakeholders to define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.
- ❑A prototyping iteration is planned quickly, and modeling (in the form of a “quick design”) occurs.
- ❑A quick design focuses on a representation of those aspects of the software that will be visible to end users (e.g., human interface layout or output display formats).
- ❑The quick design leads to the construction of a prototype.
- ❑The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements.
- ❑Iteration occurs as the prototype is tuned to satisfy the needs of various stakeholders, while at the same time enabling you to better understand what needs to be done.

Classical Evolutionary Models (Prototype)



Classical Evolutionary Models (Prototype)

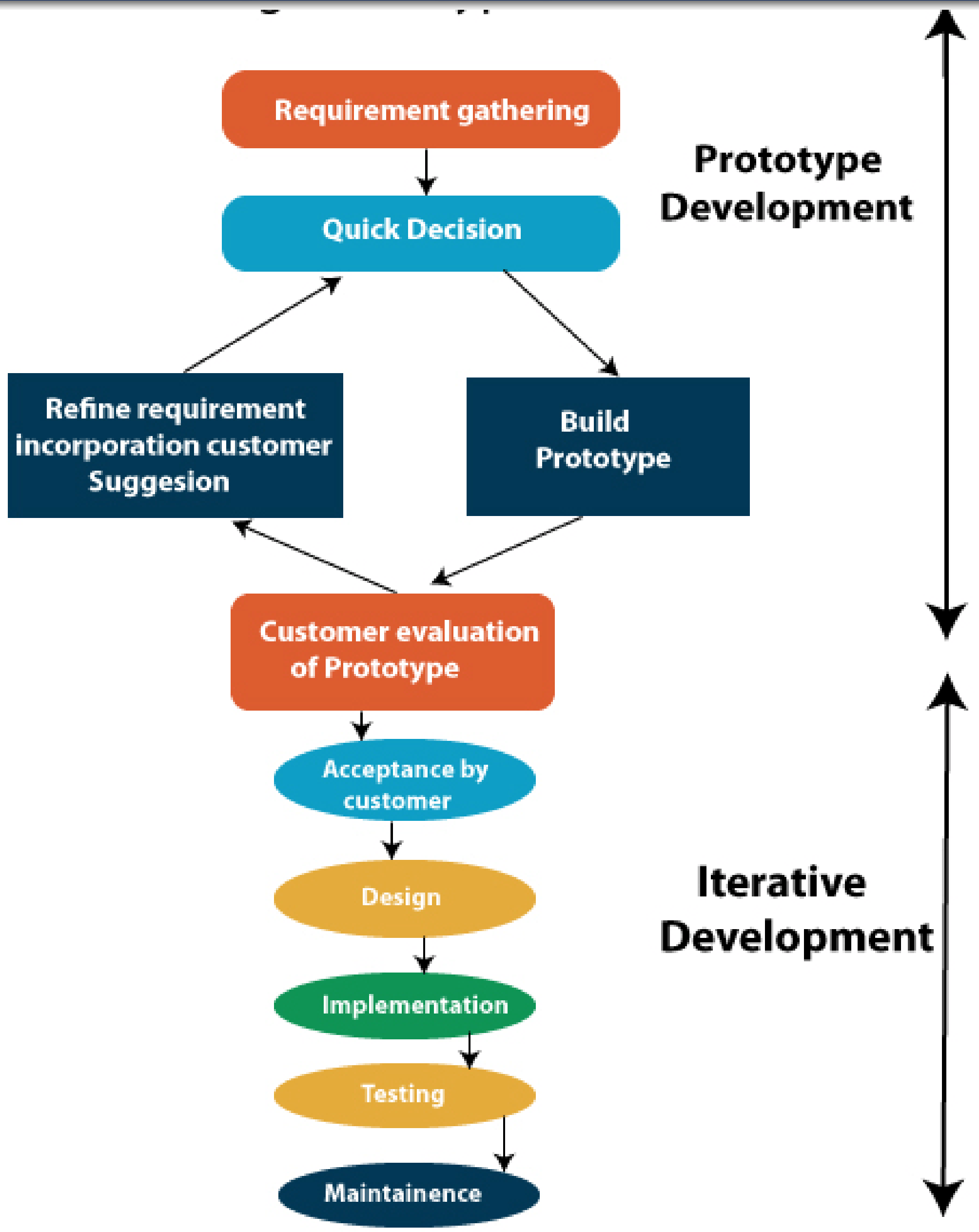
❑Ideally, the prototype serves as a mechanism for identifying software requirements.

❑If a working prototype is to be built, you can make use of existing program fragments or apply tools (e.g., report generators and window managers) that enable working programs to be generated quickly.

❑**Steps for the Prototype Model:**

1. Requirement Gathering and Analyst
2. Quick Decision
3. Build a Prototype
4. Assessment or User Evaluation
5. Prototype Refinement
6. Engineer Product

Classical Evolutionary Models (Prototype)



Classical Evolutionary Models (Prototype)

❑Advantages

1. Reduce the risk of incorrect user requirement
2. Good where requirement are changing/uncommitted
3. Regular visible process aids management
4. Support early product marketing
5. Reduce Maintenance cost.
6. Errors can be detected much earlier as the system is made side by side.

Classical Evolutionary Models (Prototype)

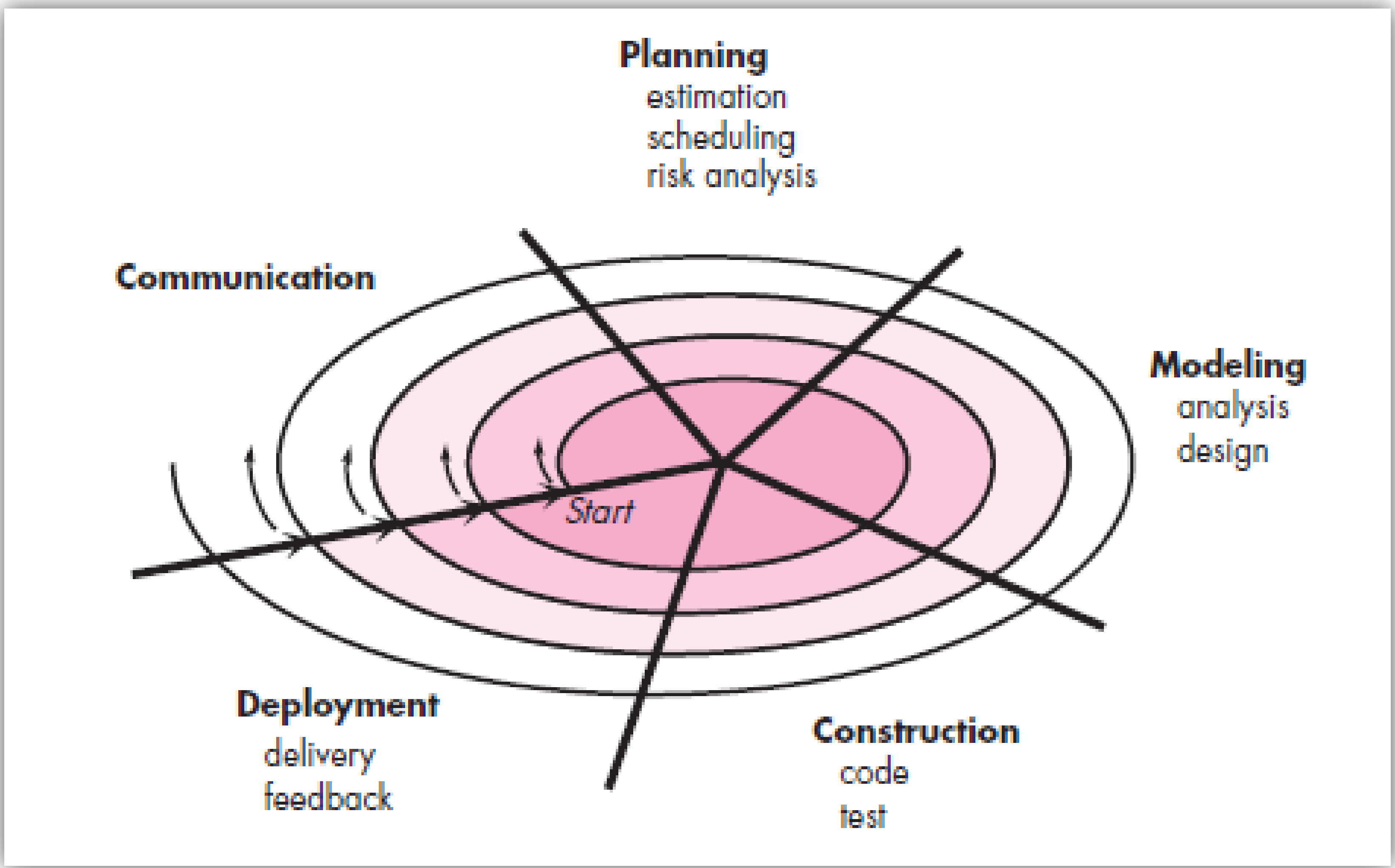
❑Disadvantages

1. An unstable/badly implemented prototype often becomes the final product.
2. Require extensive customer collaboration
   - Costs customer money
   - Needs committed customer
   - Difficult to finish if customer withdraw
   - May be too customer specific, no broad market
3. Difficult to know how long the project will last.
4. Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
5. Prototyping tools are expensive.
6. Special tools & techniques are required to build a prototype.
7. It is a time-consuming process.

Classical Evolutionary Models (Spiral)

- ❑Originally proposed by Barry Boehm , the spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.
- ❑It provides the potential for rapid development of increasingly more complete versions of the software.
- ❑Boehm describes the model in the following manner:
  - ▪The spiral development model is a risk-driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems.
  - ▪It has two main distinguishing features.
  - ▪One is a cyclic approach for incrementally growing a system’s degree of definition and implementation while decreasing its degree of risk.
  - ▪The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.

Classical Evolutionary Models (Spiral)



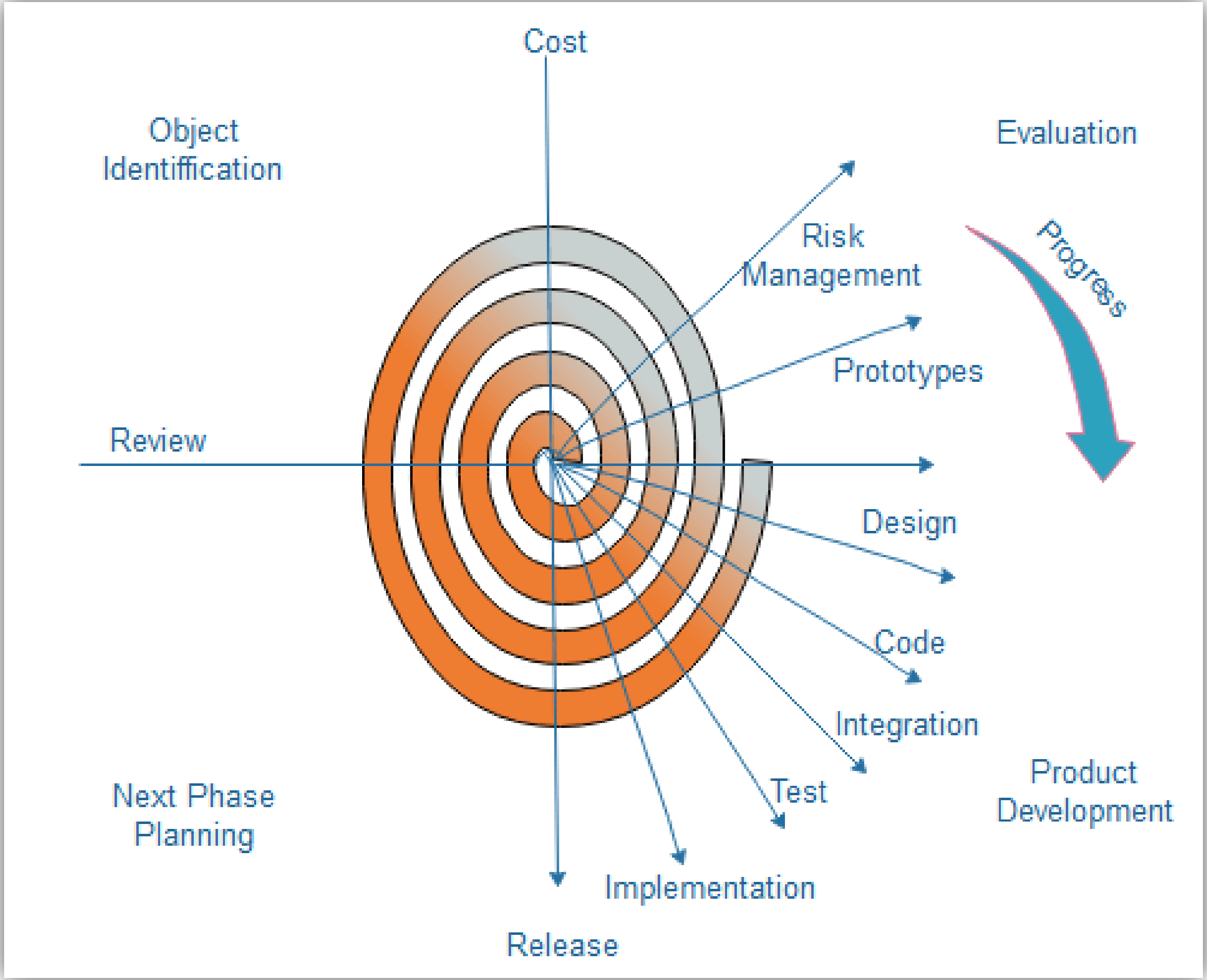
Classical Evolutionary Models (Spiral)

- A spiral model is divided into a set of framework activities defined by the software engineering team.
- Each of the framework activities represent one segment of the spiral path. As this evolutionary process begins, the software team performs activities that are implied by a circuit around the spiral in a clockwise direction, beginning at the center.
- Risk is considered as each revolution is made.
- Anchor point milestones: a combination of work products and conditions that are attained along the path of the spiral, are noted for each evolutionary pass.
- The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.
- Each pass through the planning region results in adjustments to the project plan.
- Cost and schedule are adjusted based on feedback derived from the customer after delivery.
- In addition, project manager adjusts the planned number of iterations required to complete the software.

Classical Evolutionary Models (Spiral)

- ❑The spiral model, is an evolutionary software process model that couples the iterative feature of prototyping with the controlled and systematic aspects of the linear sequential model.
- ❑It implements the potential for rapid development of new versions of the software.
- ❑Using the spiral model, the software is developed in a series of incremental releases.
- ❑During the early iterations, the additional release may be a paper model or prototype.
- ❑During later iterations, more and more complete versions of the engineered system are produced.

Classical Evolutionary Models (Spiral)



Classical Evolutionary Models (Spiral)

- **Each cycle in the spiral is divided into four parts:**
- **Objective setting:** Each cycle in the spiral starts with the identification of purpose for that cycle, the various alternatives that are possible for achieving the targets, and the constraints that exists.
- **Risk Assessment and reduction:** The next phase in the cycle is to calculate these various alternatives based on the goals and constraints. The focus of evaluation in this stage is located on the risk perception for the project.
- **Development and validation:** The next phase is to develop strategies that resolve uncertainties and risks. This process may include activities such as benchmarking, simulation, and prototyping.
- **Planning:** Finally, the next step is planned. The project is reviewed, and a choice made whether to continue with a further period of the spiral. If it is determined to keep, plans are drawn up for the next step of the project.

Classical Evolutionary Models (Spiral)

❑When to use Spiral Model:

- When deliverance is required to be frequent.
- When the project is large
- When requirements are unclear and complex
- When changes may require at any time
- Large and high budget projects

Classical Evolutionary Models (Spiral)

❑Advantages:

- High amount of risk analysis
- Useful for large and mission-critical projects.

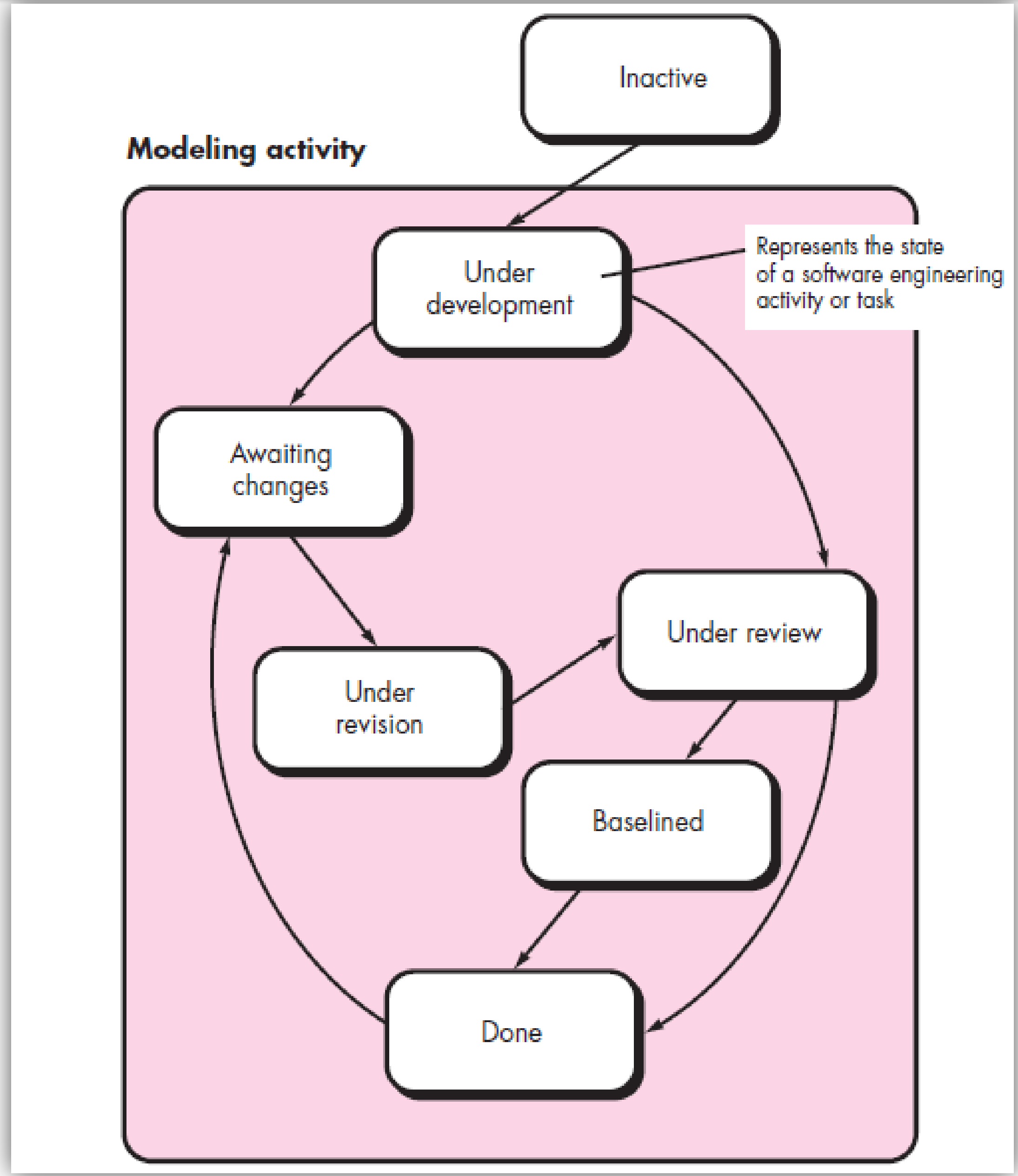
❑Disadvantages

- Can be a costly model to use.
- Risk analysis needed highly particular expertise
- Doesn't work well for smaller projects.

Concurrent Model

- The concurrent development model, sometimes called concurrent engineering, allows a software team to represent iterative and concurrent elements of any of the process models.
- For example, the modeling activity defined for the spiral model is accomplished by invoking one or more of the following software engineering actions: prototyping, analysis, and design.
- Figure provides a schematic representation of one software engineering activity within the modeling activity using a concurrent modeling approach.
- The activity—modeling—may be in any one of the states noted at any given time.
- Similarly, other activities, actions, or tasks (e.g., communication or construction) can be represented in an analogous manner.
- All software engineering activities exist concurrently but reside in different states.

Concurrent Model



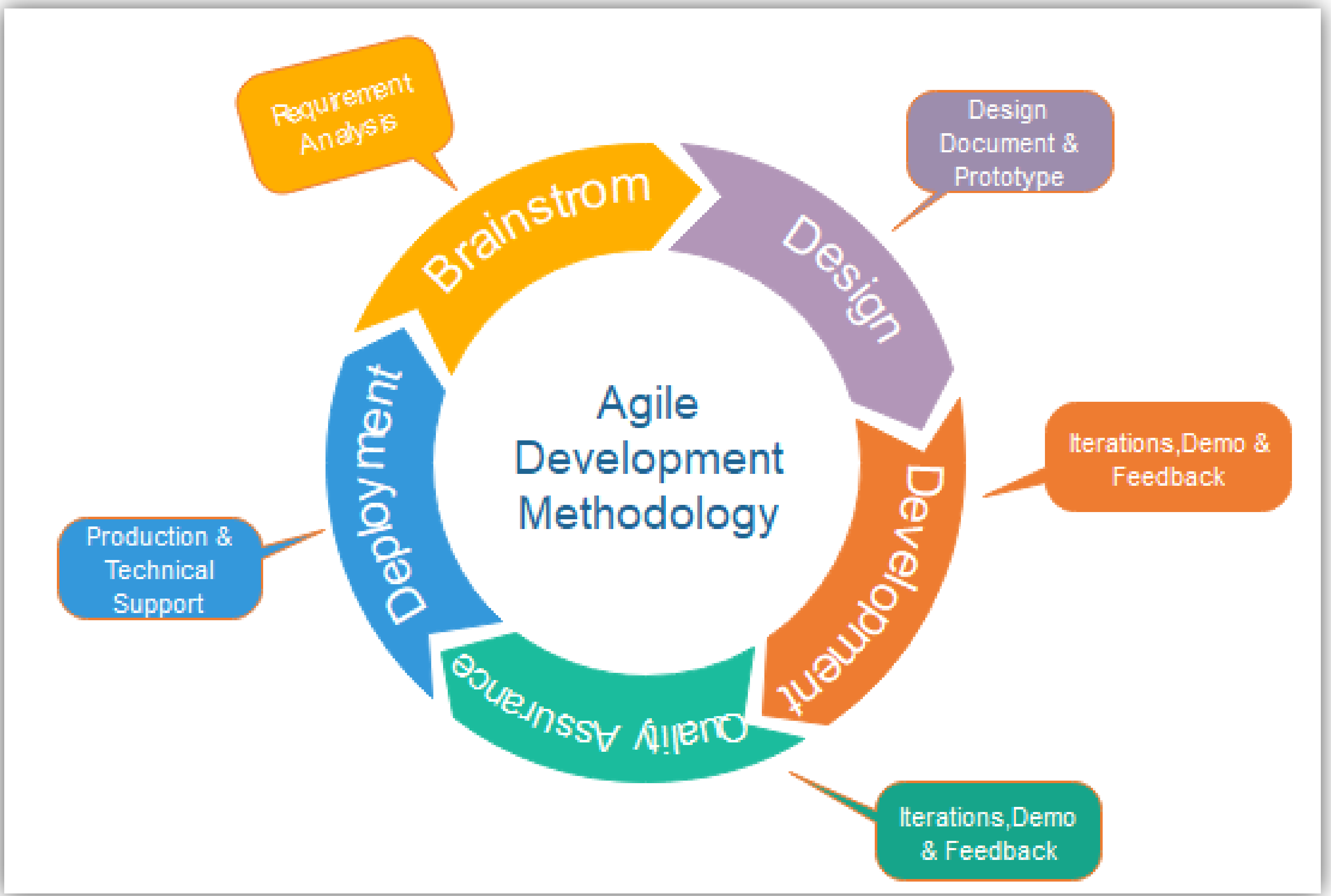
Concurrent Model

- ❑For example, early in a project the communication activity (not shown in the figure) has completed its first iteration and exists in the awaiting changes state.
- ❑The modeling activity (which existed in the inactive state while initial communication was completed, now makes a transition into the under development state.
- ❑If, however, the customer indicates that changes in requirements must be made, the modeling activity moves from the under development state into the awaiting changes state.
- ❑Concurrent modeling defines a series of events that will trigger transitions from state to state for each of the software engineering activities, actions, or tasks.
- ❑For example, during early stages of design (a major software engineering action that occurs during the modeling activity), an inconsistency in the requirements model is uncovered.
- ❑This generates the event analysis model correction, which will trigger the requirements analysis action from the done state into the awaiting changes state.

Introduction to Agility and Agile Process

- The meaning of Agile is swift or versatile."**Agile process model**" refers to a software development approach based on iterative development.
- Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning.
- The project scope and requirements are laid down at the beginning of the development process.
- Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.
- Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks.
- The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements.
- Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product is demonstrated to the client.

Introduction to Agility and Agile Process


Requirement Analysis
Brainstrom
Design
Design Document & Prototype
Agile Development Methodology
Development
Iterations,Demo & Feedback
Quality Assurance
Iterations,Demo & Feedback
Deployment
Production & Technical Support

Introduction to Agility and Agile Process

❑Following are the **Phases in the Agile model** are as follows:

1. **Requirements gathering:** In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.
2. **Design the requirements:** When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.
3. **Construction/ iteration:** When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.
4. **Testing:** In this phase, the Quality Assurance team examines the product's performance and looks for the bug.
5. **Deployment:** In this phase, the team issues a product for the user's work environment.
6. **Feedback:** After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Introduction to Agility and Agile Process

❑**Agile Testing Methods:**

1. **Scrum**
2. **Crystal**
3. **Dynamic Software Development Method(DSDM)**
4. **Feature Driven Development(FDD)**
5. **Lean Software Development**
6. **eXtreme Programming(XP)**

Introduction to Agility and Agile Process

❑**SCRUM:** is an agile development process focused primarily on ways to manage tasks in team-based development conditions. There are three roles in it, and their responsibilities are:

i. **Scrum Master:** The scrum can set up the master team, arrange the meeting and remove obstacles for the process

ii. **Product owner:** The product owner makes the product backlog, prioritizes the delay and is responsible for the distribution of functionality on each repetition.

iii. **Scrum Team:** The team manages its work and organizes the work to complete the sprint or cycle.

❑**Crystal:** There are three concepts of this method:

i. **Chartering:** Multi activities are involved in this phase such as making a development team, performing feasibility analysis, developing plans, etc.

ii. **Cyclic delivery:** under this, two more cycles consist, these are:

   a. Team updates the release plan.

   b. Integrated product delivers to the users.

iii. **Wrap up:** According to the user environment, this phase performs deployment, post-deployment.

Introduction to Agility and Agile Process

❑**DSDM:** is a rapid application development strategy for software development and gives an agile project distribution structure. The essential features of DSDM are that users must be actively connected, and teams have been given the right to make decisions. The techniques used in DSDM are:

a. Time Boxing
b. MoSCoW Rules
c. Prototyping

❑The DSDM project contains seven stages:

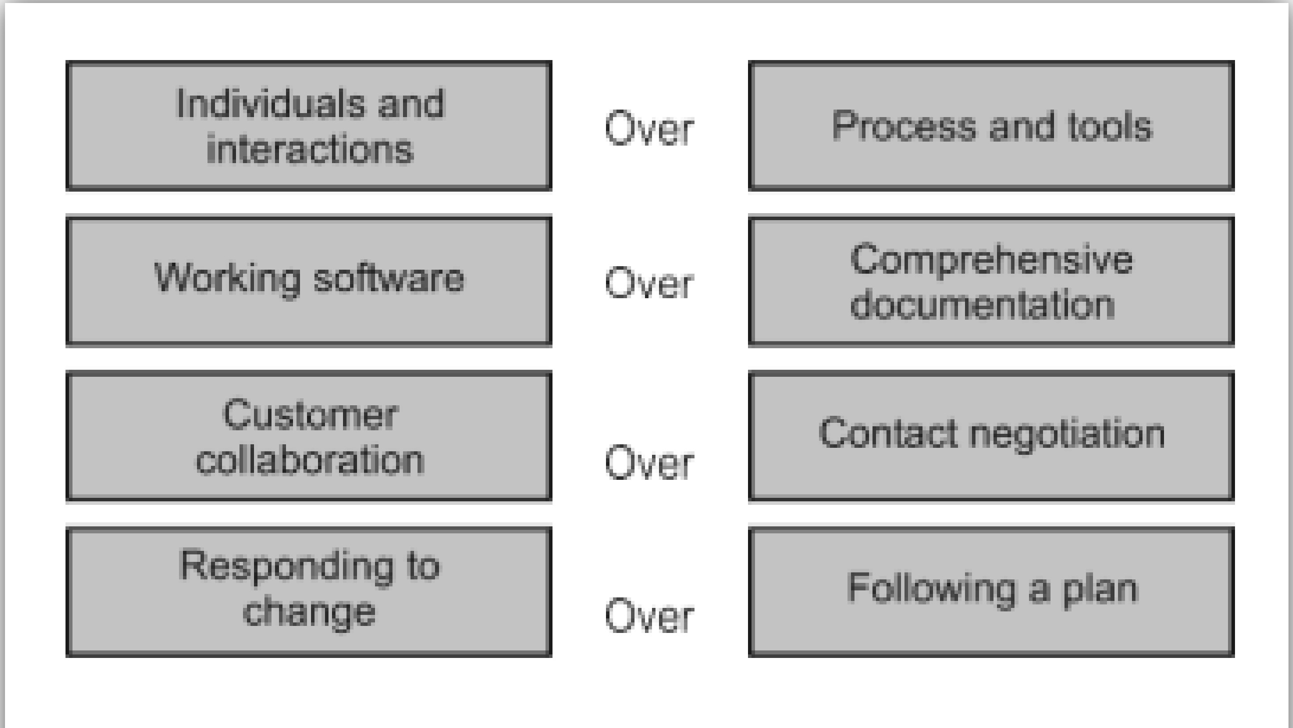
i. Pre-project
ii. Feasibility Study
iii. Business Study
iv. Functional Model Iteration
v. Design and build Iteration
vi. Implementation
vii. Post-project

Introduction to Agility and Agile Process

- ❑**FDD:** This method focuses on "Designing and Building" features. In contrast to other smart methods, FDD describes the small steps of the work that should be obtained separately per function.
- ❑**Lean Software Development:** Lean software development methodology follows the principle "just in time production." The lean method indicates the increasing speed of software development and reducing costs. Lean development can be summarized in seven phases.
- ❑**eXtreme Programming(XP) :**This type of methodology is used when customers are constantly changing demands or requirements, or when they are not sure about the system's performance.

Principles of Agile Software Development Framework

- The Agile Manifesto, also called the Manifesto for Agile Software Development, is a formal declaration of four key values and 12 principles to guide an iterative and people-centric approach to software development.
- The agile methods were developed to overcome the weakness of conventional software engineering.



Agile manifesto

Principles of Agile Software Development Framework

❑Agile Process:

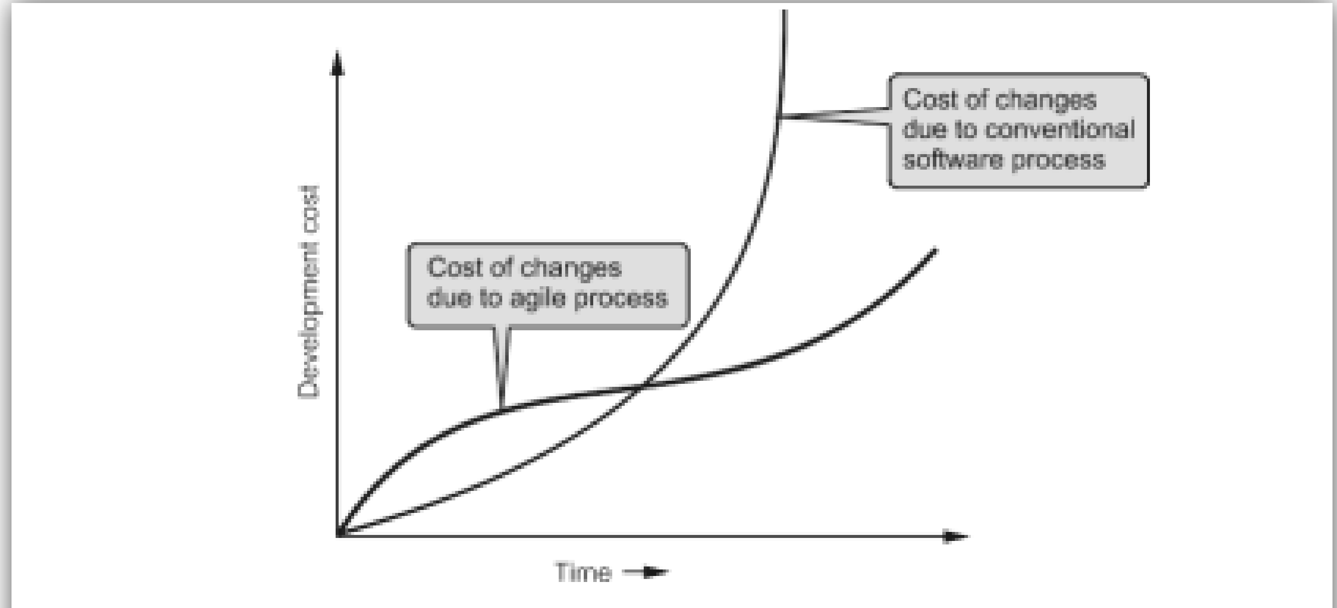
- In 1980's the heavy weight, plan based software development approach was used to develop any software product.
- In this approach too many things are done which were not directly related to software product being produced.
- This approach was rigid. That means if requirements get changed, then rework was essential. Hence new methods were proposed in 1990's which are known as agile processes.
- The agile processes are the light-weight methods are people-based rather than planbased methods.
- The agile process forces the development team to focus on software itself rather than design and documentation.
- The agile process believes in iterative method.
- The aim of agile process is to deliver the working model of software quickly to the customer.
- For example : Extreme programming is the best known of agile process.

Principles of Agile Software Development Framework

❑Agile Methodology

❑ The agile method proponents claim that if the software development is carried out using the agile approach then it will allow the software team to accommodate changes late in a software project without dramatic cost and time impact. If the incremental delivery is combined with agile practices such as continuous unit testing and pair programming then the cost of changes can be controlled.

❑ The following graph represents the how the software development approach has a strong influence on the development cost due to changes suggested.



Influence of software development approach on agile process

Principles of Agile Software Development Framework

❑**Agile Principles:** There are famous 12 principles used as agility principles

1. Satisfy the customer by early and continuous delivery of valuable software.
2. The changes in the requirements must be accommodated. Even though the changes occur late in the software development process, the agile process should help to accommodate them.
3. Deliver working software quite often. Within the shorter time span deliver the working unit.
4. Business people and developers must work together throughput the project.
5. Motivate the people who are building the projects. Provide the environment and support to the development team and trust them for the job to be done.
6. The working software is the primary measure of the progress of the software development.
7. The agile software development approach promote the constant project development. The constant speed for the development of the product must be maintained.
8. To enhance the agility there should be continuous technical excellence.

Principles of Agile Software Development Framework

❑**Agile Principles:** There are famous 12 principles used as agility principles

9. The proper attention to be given to technical excellence and good design.
10. The simplicity must be maintained while developing the project using this approach.
11. The teams must be the self-organizing team for getting best architecture, requirements and design.
12. At regular intervals the team thinks over the issue of becoming effective. After the careful review the team members adjusts their behavior accordingly.

Agile Software

❑When to use the Agile Model?

▪When frequent changes are required.

▪When a highly qualified and experienced team is available.

▪When a customer is ready to have a meeting with a software team all the time.

▪When project size is small.

❑Advantage(Pros) of Agile Method:

▪Frequent Delivery

▪Face-to-Face Communication with clients.

▪Efficient design and fulfils the business requirement.

▪Anytime changes are acceptable.

▪It reduces total development time.

❑Disadvantages(Cons) of Agile Model:

▪Due to the shortage of formal documents, it creates confusion and crucial decisions taken throughout various phases can be misinterpreted at any time by different team members.

▪Due to the lack of proper documentation, once the project completes and the developers allotted to another project, maintenance of the finished project can become a difficulty.

extreme Programming (XP) Process

- **Extreme programming (XP) Process :A way to handle the common shortcomings**
- Software Engineering involves:

I. Creativity

II. Learning and improving through trials and errors

III. Iterations

- Extreme Programming builds on these activities and coding.
- It is the detailed (not the only) design activity with multiple tight feedback loops through effective implementation, testing and refactoring continuously.

extreme Programming (XP) Process

- ❑Extreme Programming is based on the following values:

I. **Communication**

II. **Simplicity**

III. **Feedback**

IV. **Courage**

V. **Respect**

eXtreme Programming (XP) Process

- ❑XP is a lightweight, efficient, low-risk, flexible, predictable, and scientific way to develop a software.
- ❑e**X**treme **P**rogramming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements.
- ❑Extreme Programming is one of the Agile software development methodologies.
- ❑It provides values and principles to guide the team behavior.
- ❑The team is expected to self-organize.
- ❑Extreme Programming provides specific core practices where:
    - ▪Each practice is simple and self-complete.
    - ▪Combination of practices produces more complex and emergent behavior.

eXtreme Programming (XP) Process

- A key assumption of Extreme Programming is that the cost of changing a program can be held mostly constant over time.
- This can be achieved with:
  - Emphasis on continuous feedback from the customer
  - Short iterations
  - Design and redesign
  - Coding and testing frequently
  - Eliminating defects early, thus reducing costs
  - Keeping the customer involved throughout the development
  - Delivering working product to the customer

eXtreme Programming (XP) Process

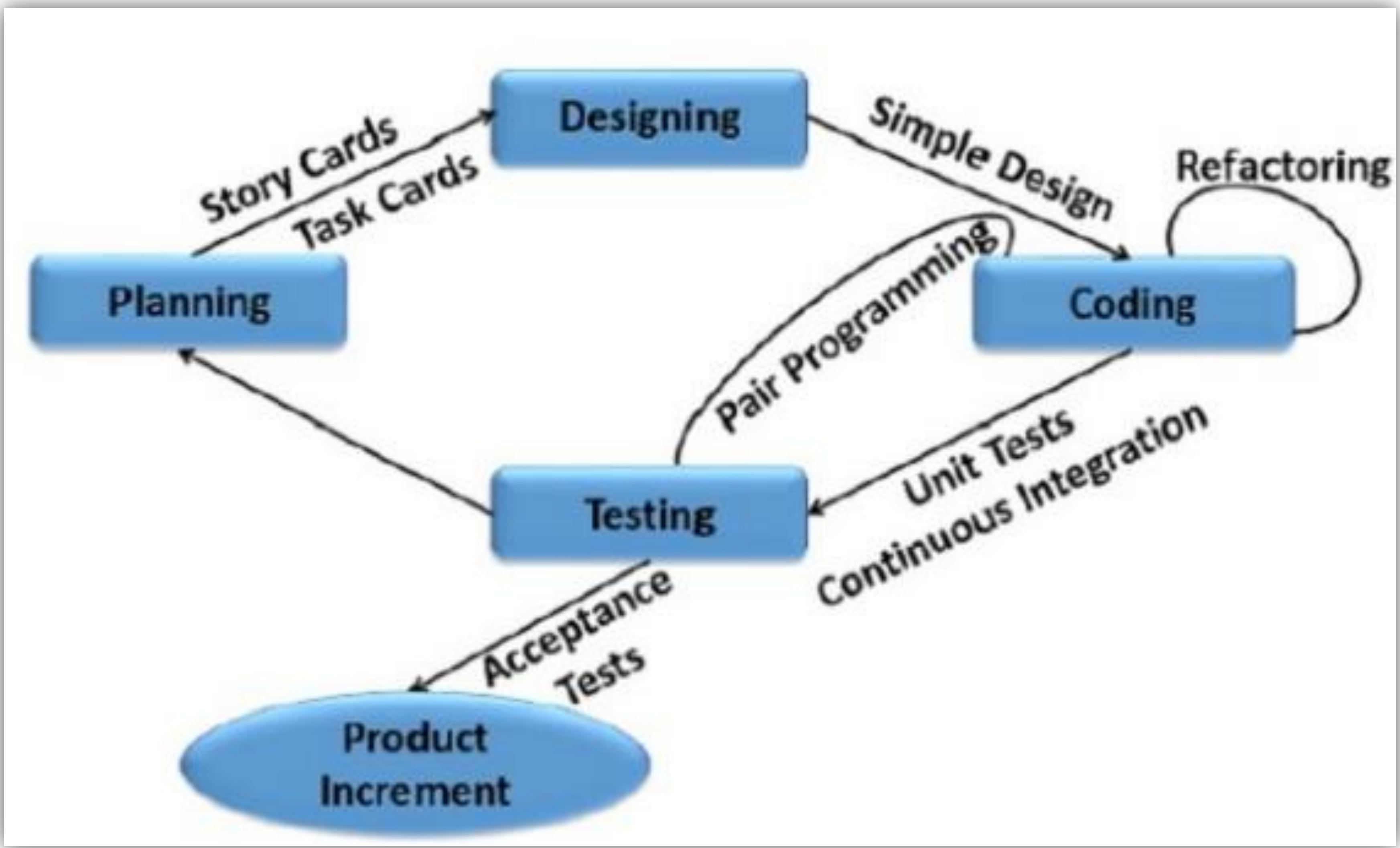
- A key assumption of Extreme Programming is that the cost of changing a program can be held mostly constant over time.
- This can be achieved with:
  - Emphasis on continuous feedback from the customer
  - Short iterations
  - Design and redesign
  - Coding and testing frequently
  - Eliminating defects early, thus reducing costs
  - Keeping the customer involved throughout the development
  - Delivering working product to the customer

eXtreme Programming (XP) Process

❑**Extreme Programming Involves:**

1. Writing unit tests before programming and keeping all of the tests running at all times. The unit tests are automated and eliminates defects early, thus reducing the costs.
2. Starting with a simple design just enough to code the features at hand and redesigning when required.
3. Programming in pairs (called pair programming), with two programmers at one screen, taking turns to use the keyboard. While one of them is at the keyboard, the other constantly reviews and provides inputs.
4. Integrating and testing the whole system several times a day.
5. Putting a minimal working system into the production quickly and upgrading it whenever required.
6. Keeping the customer involved all the time and obtaining constant feedback.

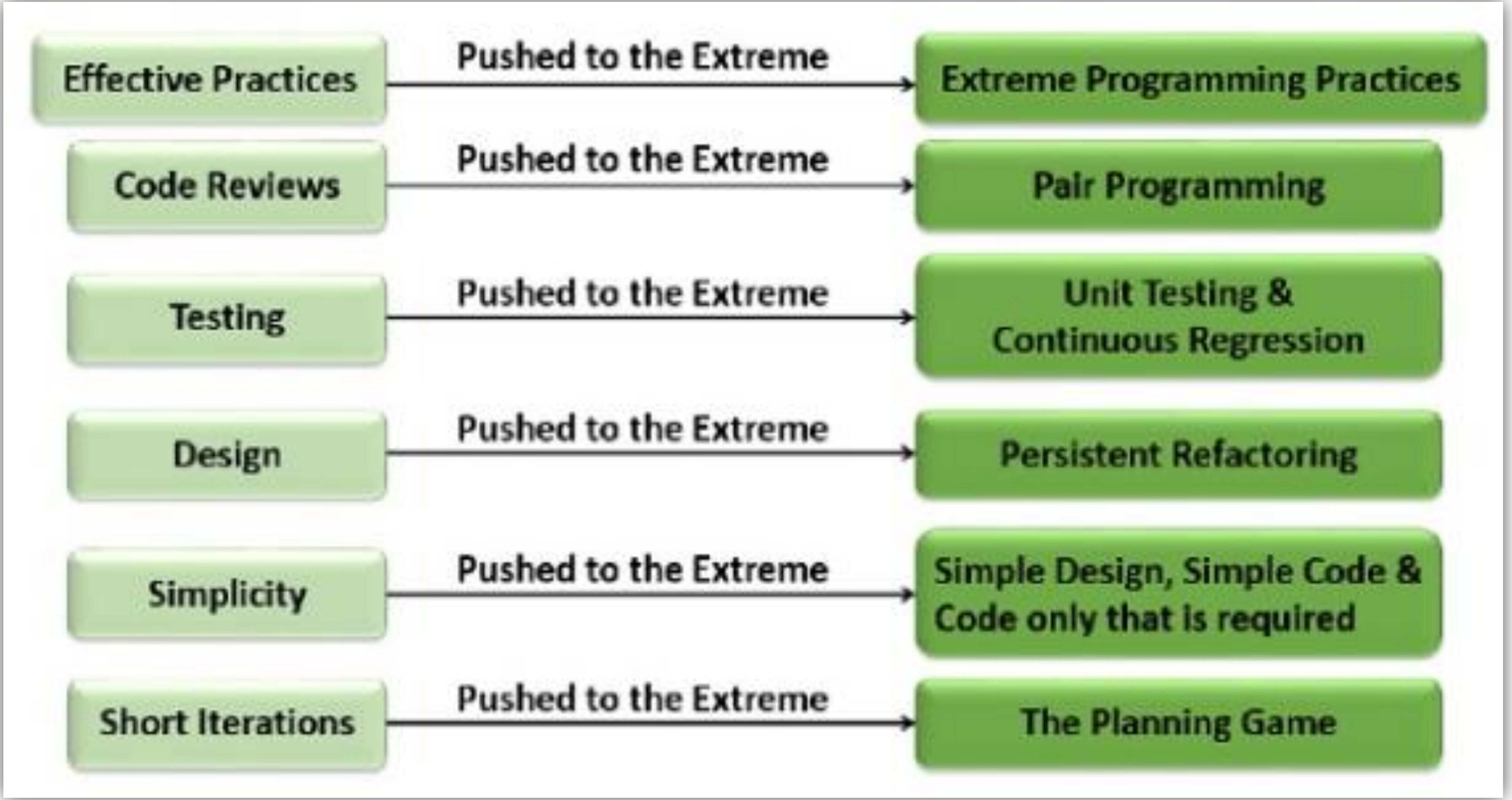
eXtreme Programming (XP) Process



eXtreme Programming (XP) Process

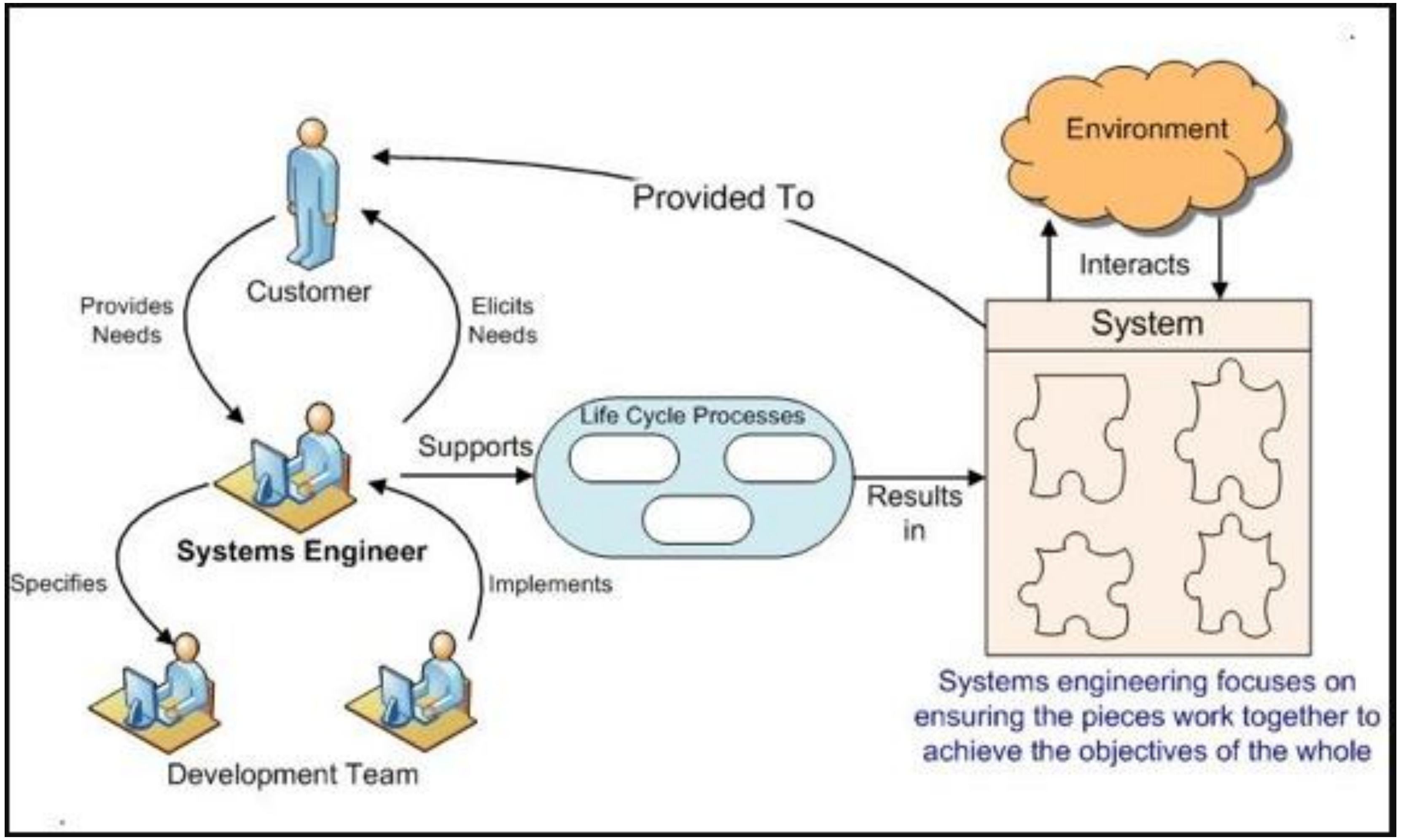
- **Why is it called "Extreme?"**
- Extreme Programming takes the effective principles and practices to extreme levels.
    - Code reviews are effective as the code is reviewed all the time.
    - Testing is effective as there is continuous regression and testing.
    - Design is effective as everybody needs to do refactoring daily.
    - Integration testing is important as integrate and test several times a day.
    - Short iterations are effective as the planning game for release planning and iteration planning.

eXtreme Programming (XP) Process



Overview of System Engineering (SE)

- ❑SE is a interdisciplinary approach and means to enable the realization of successful systems.
- ❑Successful systems must satisfy the needs of their customers, users and other stakeholders.
- ❑Some key elements of systems engineering are highlighted in Figure and include:



Overview of System Engineering (SE)

- The principles and concepts that characterize a system, where a **system** is an interacting combination of **system elements** that accomplish a defined objective(s).
- The system interacts with its environment, which may include other systems, users, and the natural environment.
- The system elements that compose the system may include hardware, software, firmware, people, information, techniques, facilities, services, and other support elements.
- A **systems engineer** is a person or role who supports this interdisciplinary approach. In particular, the **systems engineer** often serves to elicit and translate customer needs into specifications that can be realized by the system development team.
- In order to help realize successful systems, the systems engineer supports a set of **life cycle processes** beginning early in conceptual design and continuing throughout the **life cycle** of the system through its manufacture, deployment, use and disposal.

Overview of System Engineering (SE)

- ❑The systems engineer must analyze, specify, design, and verify the system to ensure that its functional, interface, performance, physical, and other quality characteristics, and cost are balanced to meet the needs of the system stakeholders.
- ❑A systems engineer helps ensure the elements of the system fit together to accomplish the objectives of the whole, and ultimately satisfy the needs of the **customers** and other **stakeholders** who will acquire and use the system.

Note for Students

- This power point presentation is for lecture, therefore it is suggested that also utilize the text books and lecture notes.