Software Engineering (23ISEN34)

Prepared By

Dr. Veena R S
Associate Professor
Department of ISE
DSATM

Software Engineering (23ISEN34)

Syllabus

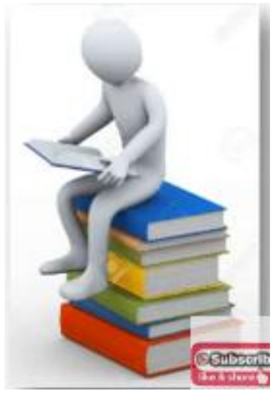
Textbooks and Reference Books

Textbooks:

1. Software Engineering-A Practitioners Approach, Roger S. Pressman, Tata McGraw Hill,7th Edition,2010.
2. Software Engineering, Ian Somerville, Pearson Education,9th Edition ,2011.

Reference Books:

1. Software Project Management, Bob Hughes, Mike Cotterell, Rajib Mall, McGraw Hill Education, 6th Edition,2018.
2. Software Engineering theory and Practice, Shari Lawrence Pfleeger, Joanne m Alec, Pearson Education ,3rd Edition,2006.
3. Fundamentals of Software Engineering, Rajib Mall, Prentice-hall Of India Pvt Ltd.,3rd Edition, 2012.



Module – 1

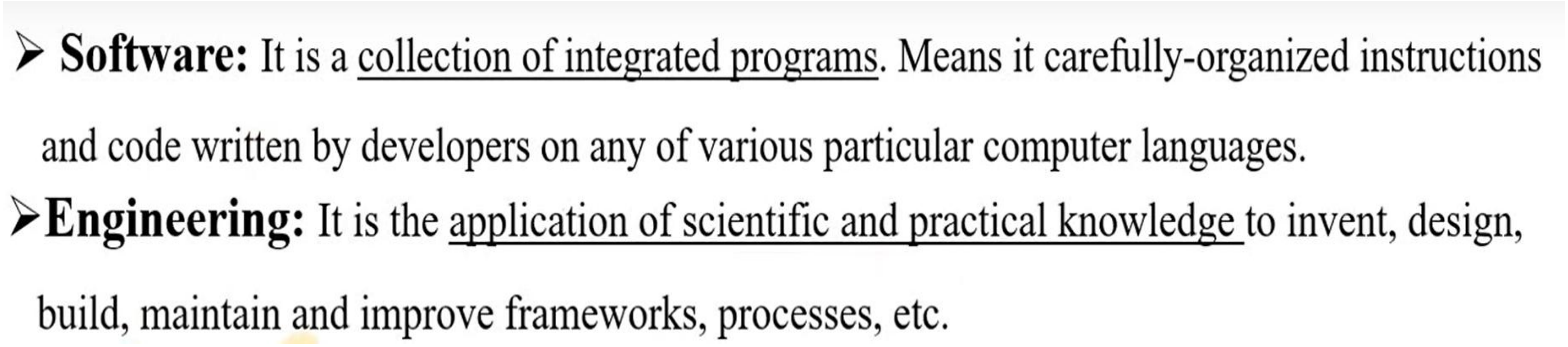
Introduction to Software Engineering

Process Models

Contents

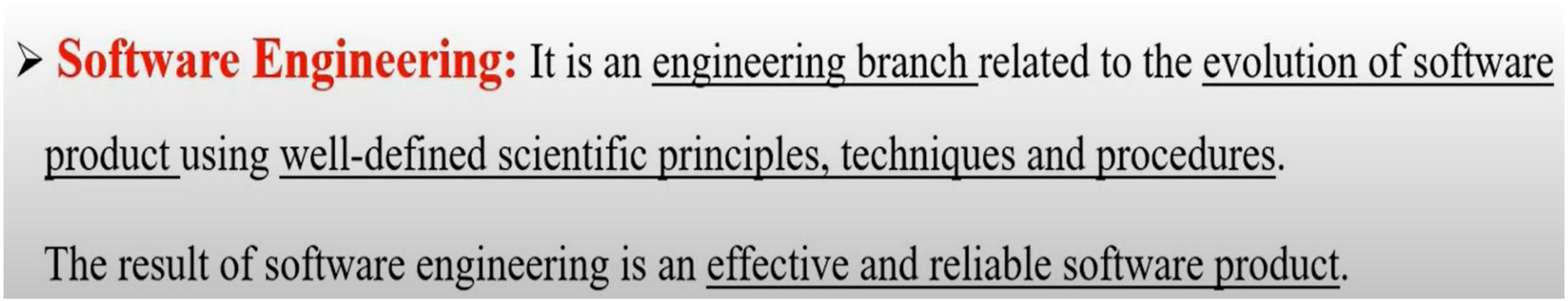
IMPLEMENTATION
RESEARCH
MANAGEMENT
OPERATION
SOFTWARE
DESIGN
SYSTEMATIC
TESTING
ENGINEERING
MAINTENANCE
DEVELOPMENT
DOCUMENTATION
APPLICATION
REQUIREMENTS
Models
PROGRAMMING
Tool
PROCESS
METHOD

What is Software Engineering



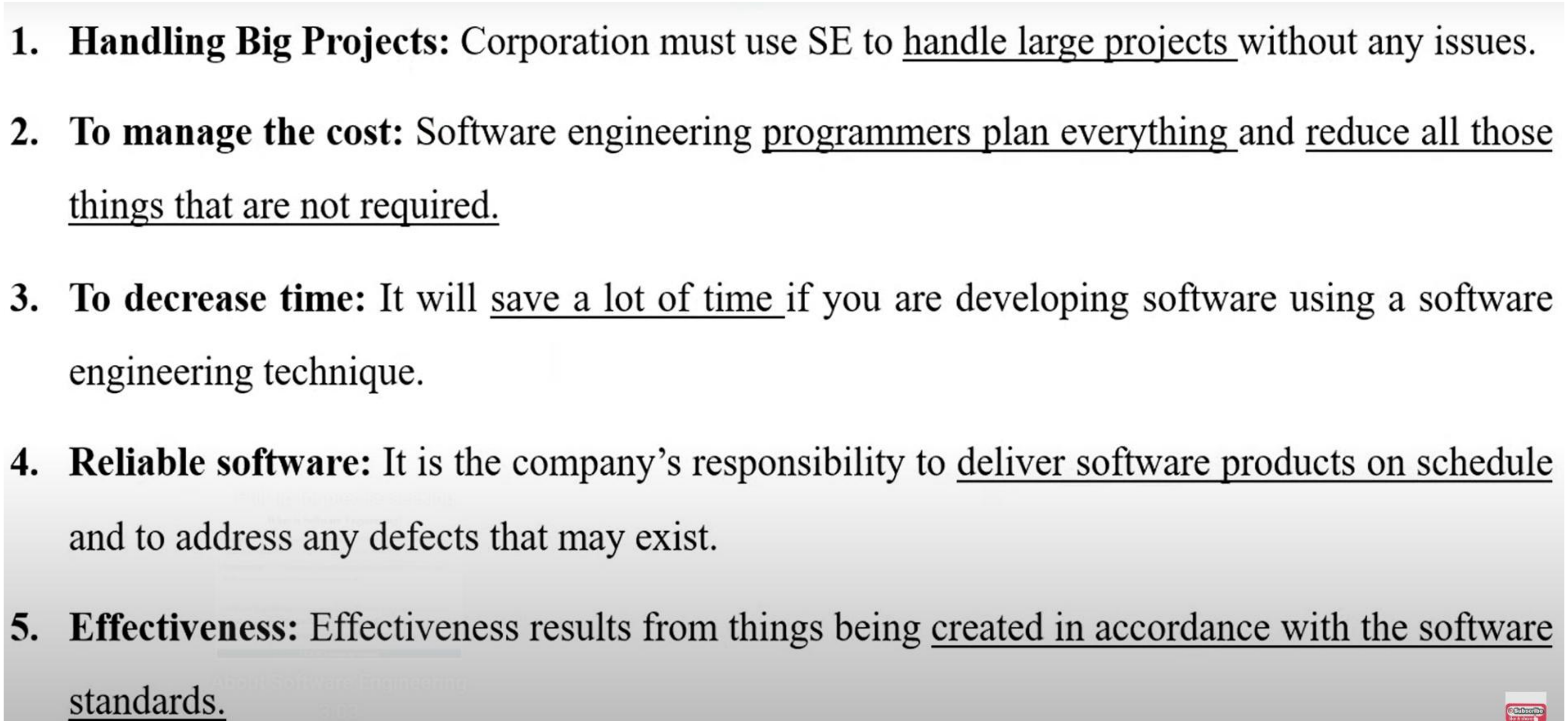
➢ **Software:** It is a collection of integrated programs. Means it carefully-organized instructions and code written by developers on any of various particular computer languages.

➢ **Engineering:** It is the application of scientific and practical knowledge to invent, design, build, maintain and improve frameworks, processes, etc.



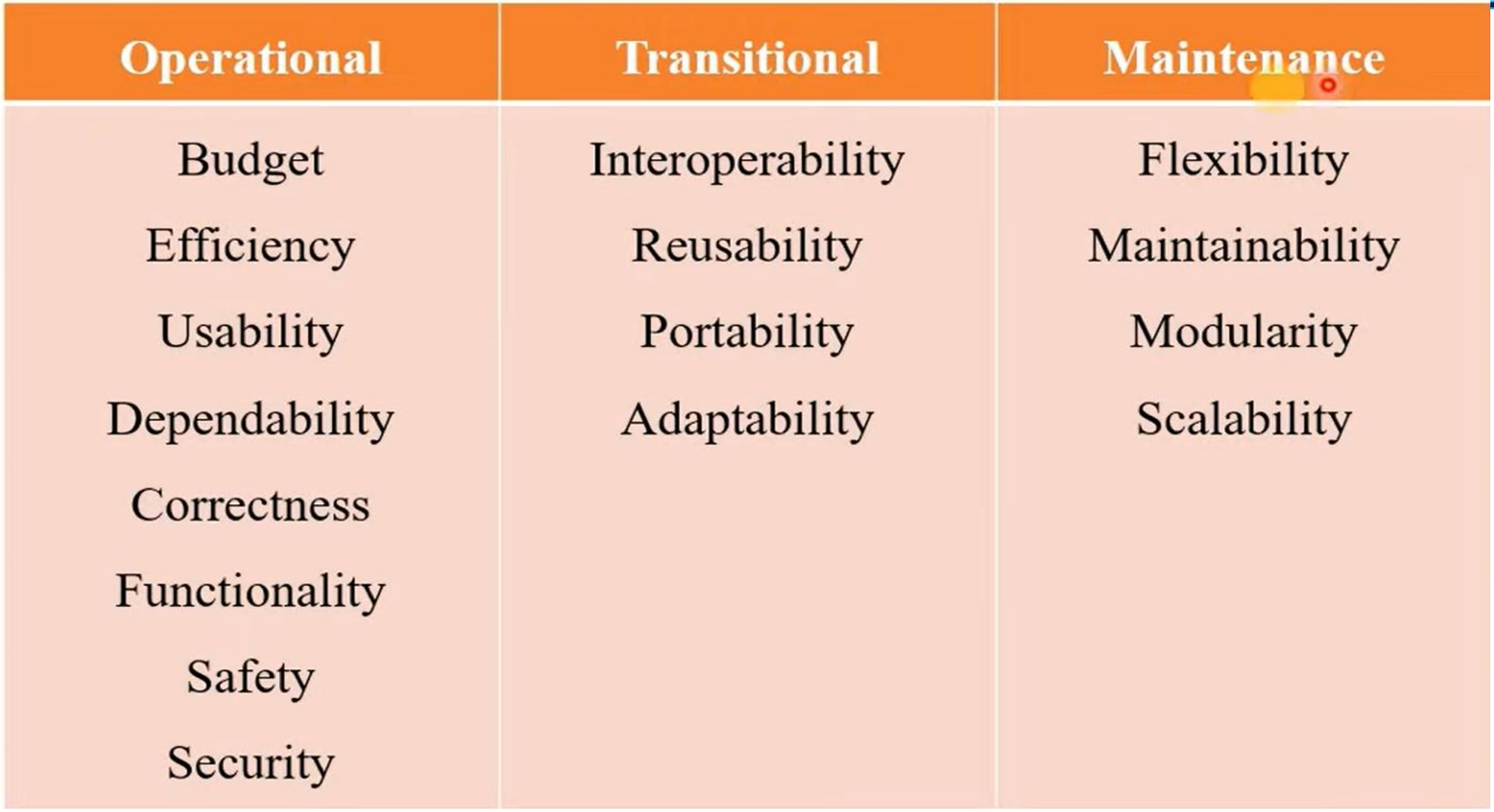
➢ **Software Engineering:** It is an engineering branch related to the evolution of software product using well-defined scientific principles, techniques and procedures. The result of software engineering is an effective and reliable software product.

Need of Software Engineering



1. **Handling Big Projects:** Corporation must use SE to handle large projects without any issues.
2. **To manage the cost:** Software engineering programmers plan everything and reduce all those things that are not required.
3. **To decrease time:** It will save a lot of time if you are developing software using a software engineering technique.
4. **Reliable software:** It is the company’s responsibility to deliver software products on schedule and to address any defects that may exist.
5. **Effectiveness:** Effectiveness results from things being created in accordance with the software standards.

Characteristics of Good Software



Operational	Transitional	Maintenance
Budget	Interoperability	Flexibility
Efficiency	Reusability	Maintainability
Usability	Portability	Modularity
Dependability	Adaptability	Scalability
Correctness		
Functionality		
Safety		
Security		

NATURE OF SOFTWARE

- Software is a product, and at the same time, the vehicle for delivering a product.

- As a product, it delivers the computing potential embodied by computer hardware or more broadly, by a network of computers that are accessible by local hardware.

NATURE OF SOFTWARE

DEFINING SOFTWARE

- Instructions (computer programs) that when executed provide desired features, function, and performance;
- Data structures that enable the programs to adequately manipulate information, and
- Descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.

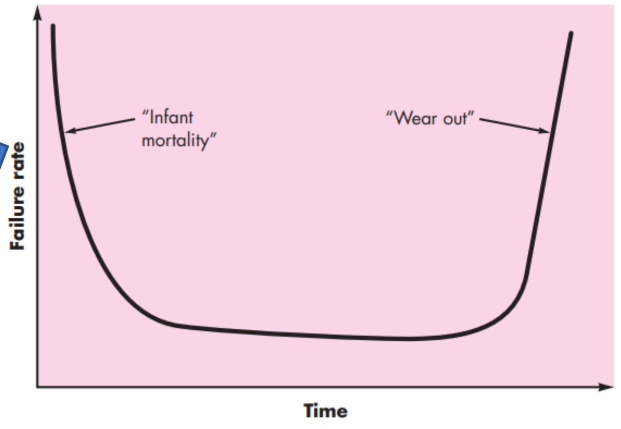
NATURE OF SOFTWARE

- It is important to examine the characteristics of software that make it different from other things that human beings build.
- Software is a logical rather than a physical system element.

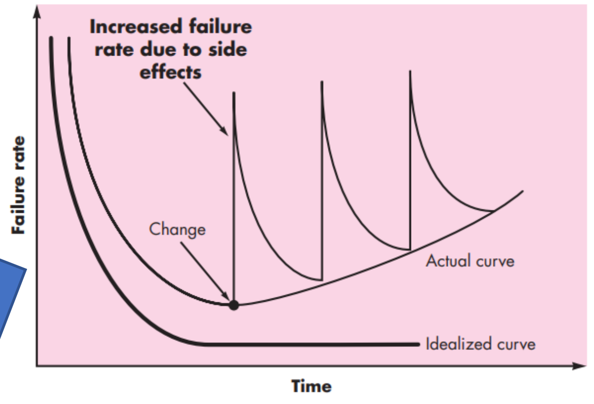
Software has characteristics that are considerably different than those of hardware

1. Software is developed or engineered; it is not manufactured in the classical sense
2. Software doesn't "wear out"
3. Although the industry is moving toward component-based construction, most software continues to be custom built

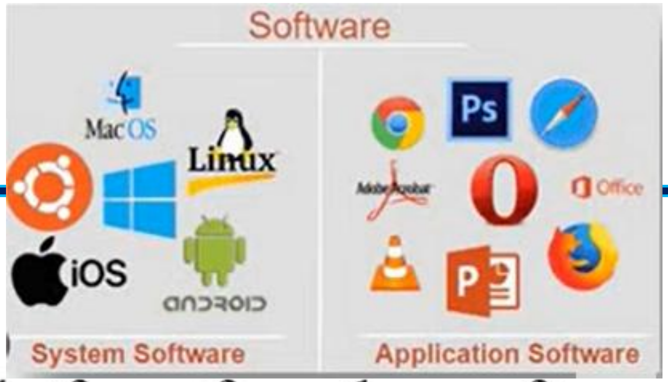
Failure Curve for Hardware



Failure Curve for Software



NATURE OF SOFTWARE



1. System Software:

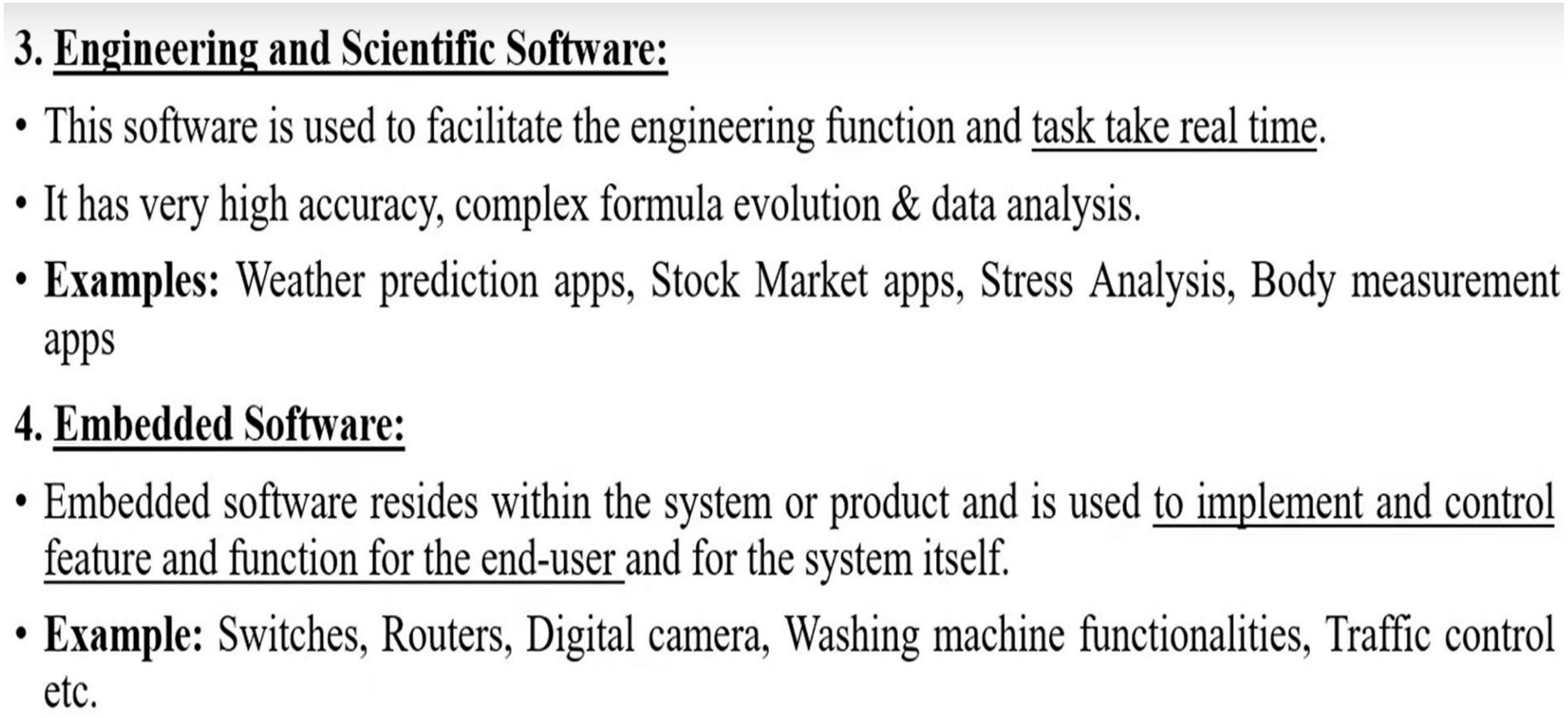
- System software is software designed to provide a platform for other software's.
- It is a interaction between hardware & application software.
- **Examples :** Operating Systems like macOS, Linux, Android and Microsoft Windows.

2. Application Software:

- Application Software is a computer program designed to carry out a specific task as per user & business need.

Examples: Social Media Apps, Gaming Apps, Word Processing Apps, Multimedia Apps, Banking Apps, Shopping Apps, Booking Apps etc.

NATURE OF SOFTWARE



3. Engineering and Scientific Software:

- This software is used to facilitate the engineering function and task take real time.
- It has very high accuracy, complex formula evolution & data analysis.
- **Examples:** Weather prediction apps, Stock Market apps, Stress Analysis, Body measurement apps

4. Embedded Software:

- Embedded software resides within the system or product and is used to implement and control feature and function for the end-user and for the system itself.
- **Example:** Switches, Routers, Digital camera, Washing machine functionalities, Traffic control etc.

NATURE OF SOFTWARE

5. Web Applications:

- It is a client-server computer program which the client runs on the web browser.
- Web apps can be little more than a set of linked hypertext files that present information using text and limited graphics.
- **Examples:** Online forms, Shopping carts, Gmail, Yahoo, Photo editing, File conversion etc.

6. Artificial Intelligence Software:

- It makes use of a nonnumerical algorithm to solve a complex problem.
- Application within this area includes robotics, expert system, pattern recognition, artificial neural network, theorem proving and game playing.
- **Examples:** Google Cloud, Azure studio, Tensor Flow, Salesforce etc.

NATURE OF SOFTWARE

7. Product-line Software—designed to provide a specific capability for use by many different customers. Product-line software can focus on a limited and esoteric marketplace or address mass consumer markets.

Examples: Inventory control products, word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, and personal and business financial applications

THESE ARE THE SEVEN BROAD CATEGORIES OF COMPUTER SOFTWARE AND SEVEN CHALLENGES FOR SOFTWARE ENGINEERS

YET NEW CHALLENGES FOR SOFTWARE ENGINEERS HAVE APPEARED ON THE HORIZON:

- **Open World Computing:** The rapid growth of wireless networking may soon lead to true pervasive, distributed computing. The challenge for software engineers will be to develop systems and application software that will allow mobile devices, personal computers, and enterprise systems to communicate across vast networks
- **Netsourcing:** The World Wide Web is rapidly becoming a computing engine as well as a content provider. The challenge for software engineers is to architect simple and sophisticated applications that provide a benefit to targeted end-user markets worldwide.
- **Open Source:** A growing trend that results in distribution of source code for systems applications (e.g., operating systems, database, and development environments) so that many people can contribute to its development. The challenge for software engineers is to build source code that is self-descriptive.

LEGACY SOFTWARE

The older programs are often referred as legacy software which have been the focus of continuous attention and concern since the 1960s

Legacy software systems . . . were developed decades ago and have been continually modified to meet changes in business requirements and computing platforms. The proliferation of such systems is causing headaches for large organizations who find them costly to maintain and risky to evolve.

LEGACY SOFTWARE

Legacy systems often evolve for one or more of the following reasons:

- The software mu ds of new computing environments or technology.
- The software mu ew business requirements.
- The software mu operable with other more modern systems or datal
- The software must it viable within a network environment.

When these modes of evolution occur, a legacy system must be reengineered

UNIQUE NATURE OF WEBAPPS

In the early days of the World Wide Web (circa 1990 to 1995), *websites* consisted of little more than a set of linked hypertext files that presented information using text and limited graphics.

As time passed, the augmentation of HTML by development tools (e.g., XML, Java) enabled Web engineers to provide computing capability along with informational content. *Web-based systems and applications* (refer to these collectively as *WebApps*) were born

ATTRIBUTES OF WEBPPS

1. **Network intensiveness:** The network may enable worldwide access and communication (i.e., the Internet) or more limited access and communication (e.g., a corporate Intranet).

2. Concurrency: A large number of users may access the WebApp at one time. In many cases, the patterns of usage among end users will vary greatly.

3. **Unpredictable load:** The number of users of the WebApp may vary by orders of magnitude from day to day.

One hundred users may show up on Monday; 10,000 may use the system on Thursday

4. **Performance:** If a WebApp user must wait too long (for access, for serverside processing, for client-side formatting and display), he or she may decide to go elsewhere.

5. **Availability:** Although expectation of 100 percent availability is unreasonable, users of popular WebApps often demand access on a 24/7/365 basis.

Users in Australia or Asia might demand access during times when traditional domestic software applications in North America might be taken off-line for maintenance.

6. **Data driven:** The primary function of many WebApps is to use hypermedia to present text, graphics, audio, and video content to the end user. In addition, WebApps are commonly used to access information that exists on databases that are not an integral part of the Web-based environment (e.g., e-commerce or financial applications).

ATTRIBUTES OF WEBPPS

7. **Content sensitive:** The quality and aesthetic nature of content remains an important determinant of the quality of a WebApp.
8. **Continuous evolution:** Unlike conventional application software that evolves over a series of planned, chronologically spaced releases, Web applications evolve continuously.
9. **Immediacy:** Although *immediacy*—the compelling need to get software to market quickly—is a characteristic of many application domains.
10. **Security:** Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end users who may access the application. In order to protect sensitive content and provide secure modes of data transmission, strong security measures must be implemented throughout the infrastructure that supports a WebApp and within the application itself.
11. **Aesthetics:** An undeniable part of the appeal of a WebApp is its look and feel.

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 (e.g., 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* (e.g., architectural design) encompasses a set of tasks that produce a major work product (e.g., an architectural design

A *task* focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome

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 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

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

SOFTWARE ENGINEERING PRACTICE

Generic framework activities—**communication, planning, modeling, construction,** and **deployment**—and umbrella activities establish a skeleton architecture for software engineering work.

The essence of software engineering practice:

1. *Understand the problem* (communication and analysis).
2. *Plan a solution* (modeling and software design).
3. *Carry out the plan* (code generation).
4. *Examine the result for accuracy* (testing and quality assurance).

GENERAL PRINCIPLES

The word principle is defined as "an important underlying law or assumption required in a system of thought."

David Hooker [Hoo96] has proposed seven principles that focus on software engineering practice as a whole:

The First Principle: The reason it all exists

The Second Principle: Keep it Simple

The Third Principle: Maintain the vision

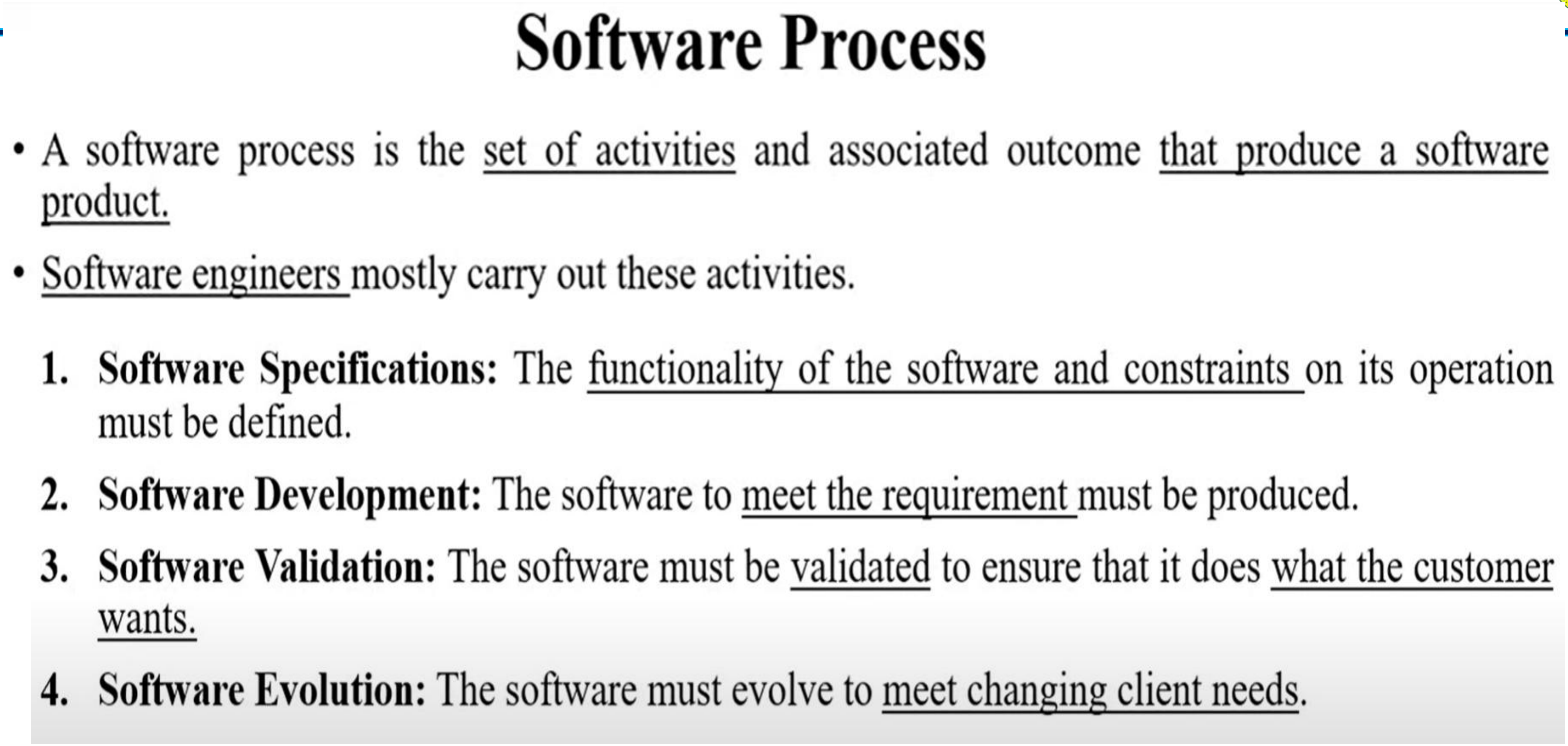
The Fourth Principle: What your Produce, Others consume

The Fifth Principle: Be Open to the Future

The Sixth Principle: Plan ahead for Resue

The Seventh Principle: Think

Process Models

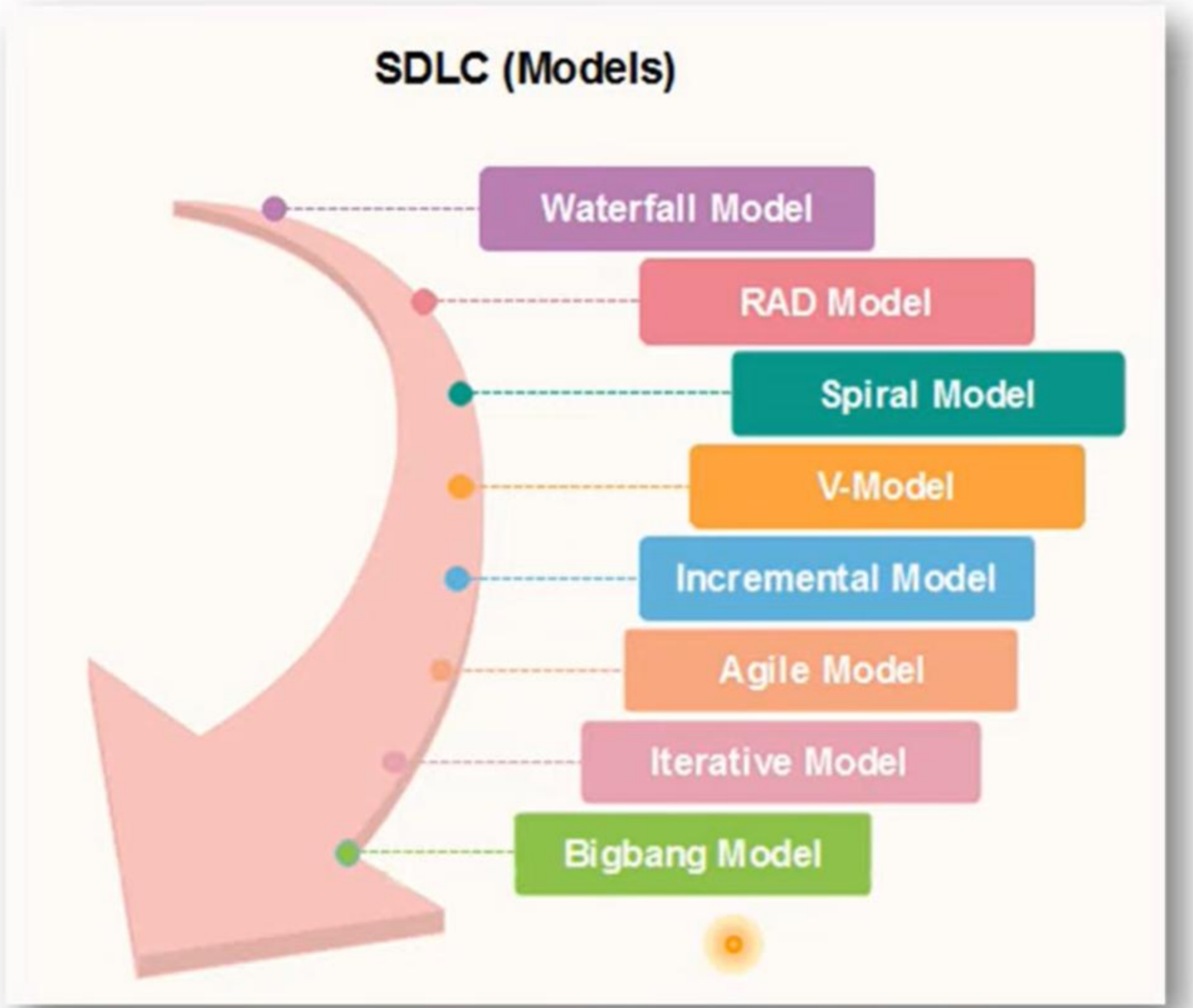


Software Process

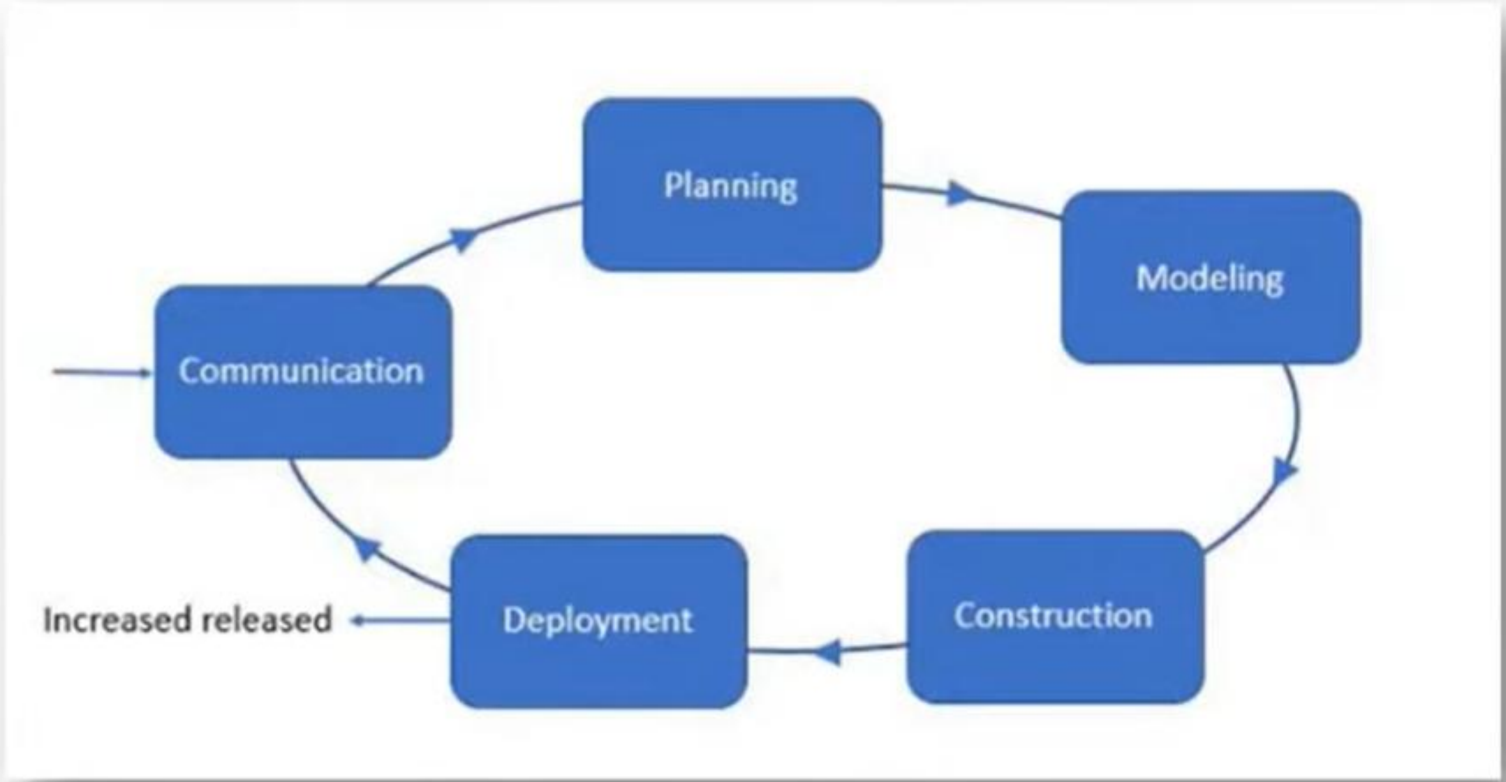
- A software process is the set of activities and associated outcome that produce a software product.
- Software engineers mostly carry out these activities.

1. **Software Specifications:** The functionality of the software and constraints on its operation must be defined.
2. **Software Development:** The software to meet the requirement must be produced.
3. **Software Validation:** The software must be validated to ensure that it does what the customer wants.
4. **Software Evolution:** The software must evolve to meet changing client needs.

Software Engineering – SDLC Models



GENERIC PROCESS MODEL

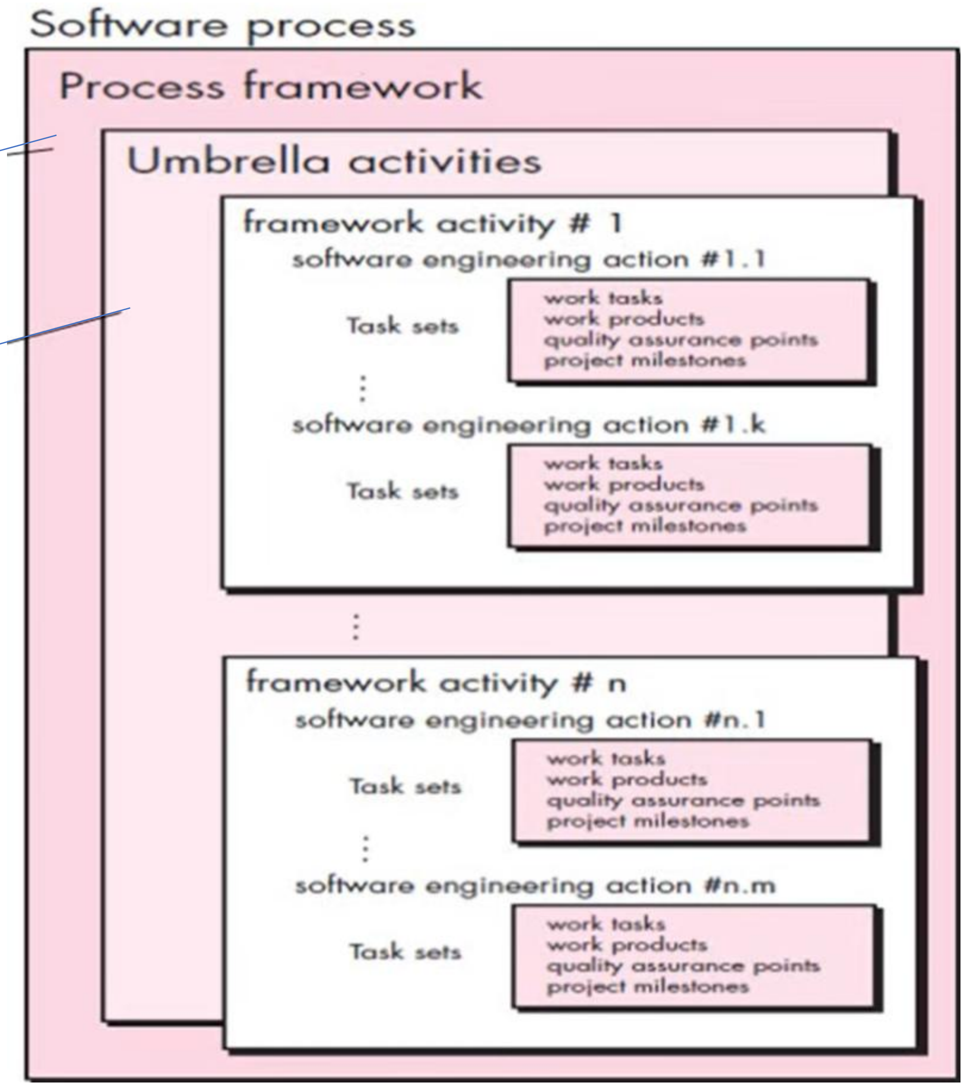


GENERIC PROCESS MODEL

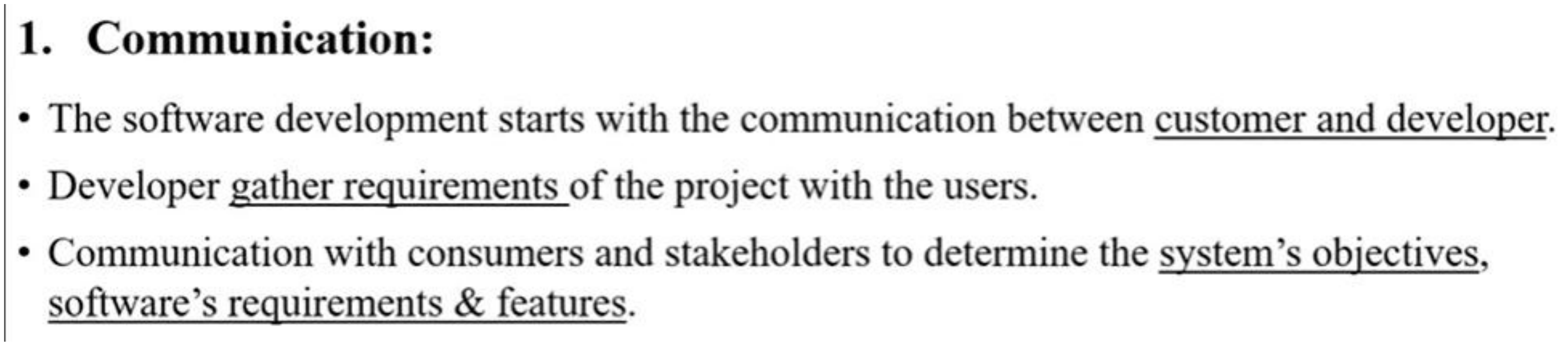
Communication, Planning, Modelling, Construction & Deployment

S/w project tracking, Risk management, SQA, Technical Reviews, SCM, Reusability

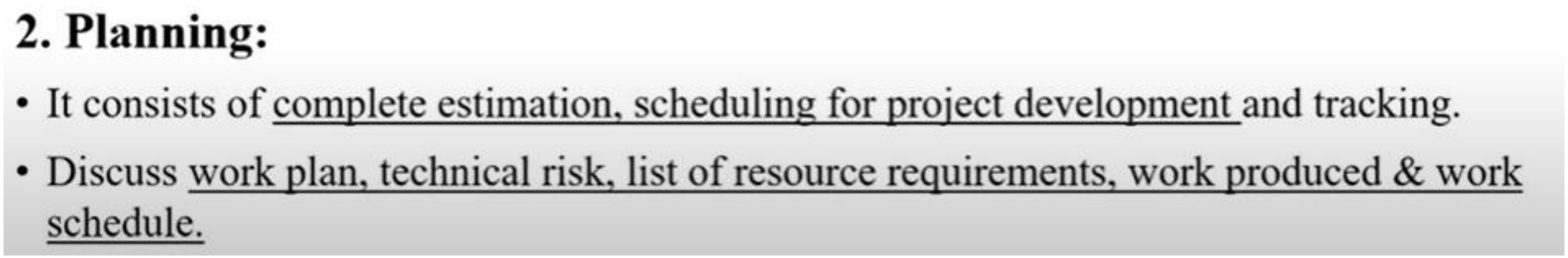
- **Software process** means to develop quality products by using technical & management rules.
- **The generic process model** is an description of the software development process.
- It is used in most software models since it provides a base for them.



PROCESS FRAMEWORK ACTIVITIES

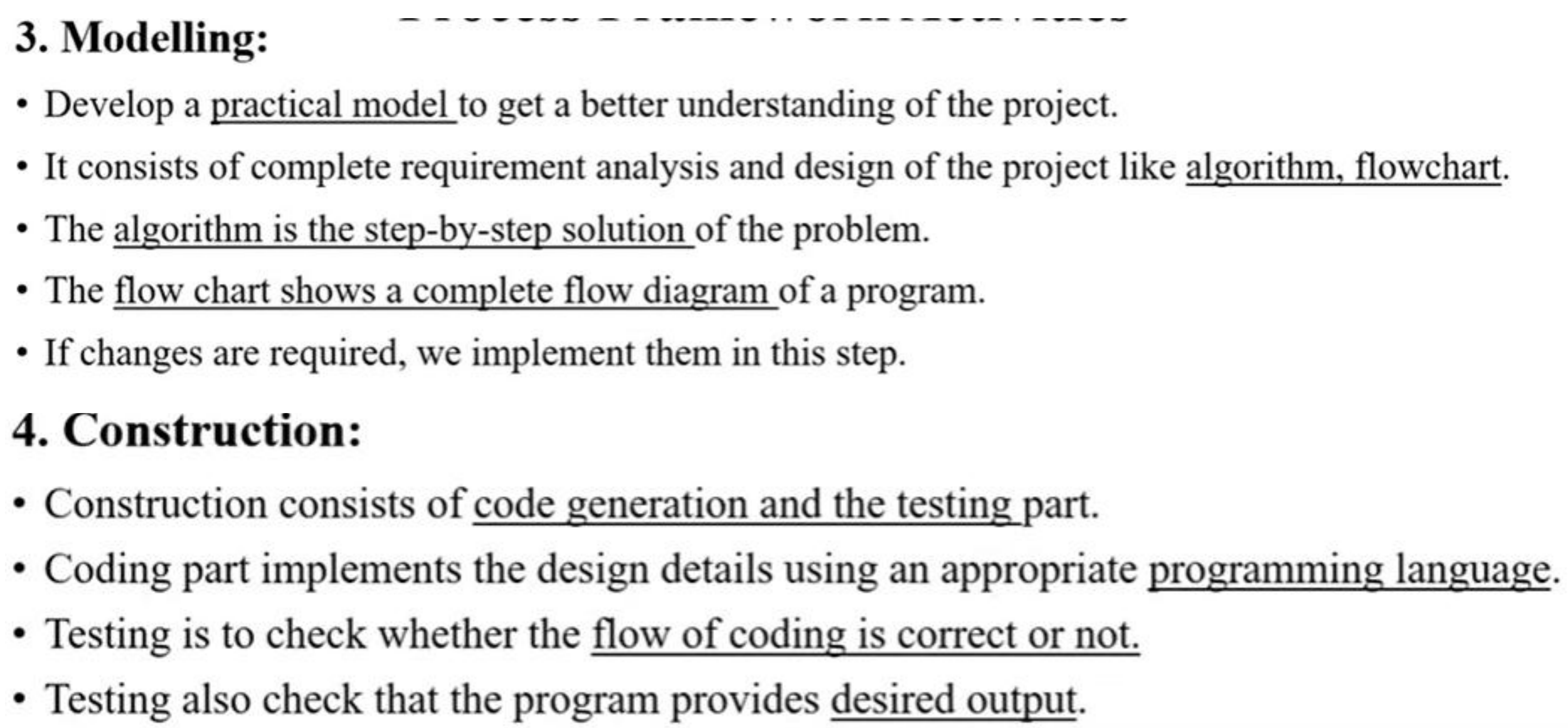

1. Communication:

- The software development starts with the communication between customer and developer.
- Developer gather requirements of the project with the users.
- Communication with consumers and stakeholders to determine the system's objectives, software's requirements & features.


2. Planning:

- It consists of complete estimation, scheduling for project development and tracking.
- Discuss work plan, technical risk, list of resource requirements, work produced & work schedule.

PROCESS FRAMEWORK ACTIVITIES



3. Modelling:

- Develop a practical model to get a better understanding of the project.
- It consists of complete requirement analysis and design of the project like algorithm, flowchart.
- The algorithm is the step-by-step solution of the problem.
- The flow chart shows a complete flow diagram of a program.
- If changes are required, we implement them in this step.

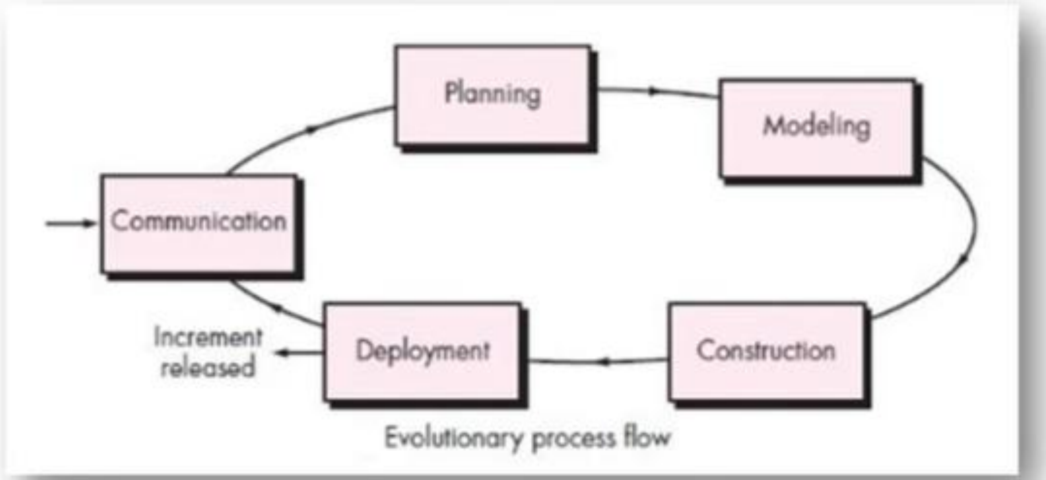
4. Construction:

- Construction consists of code generation and the testing part.
- Coding part implements the design details using an appropriate programming language.
- Testing is to check whether the flow of coding is correct or not.
- Testing also check that the program provides desired output.

PROCESS FRAMEWORK ACTIVITIES

5. Deployment:

- Deployment step consists of delivering the product to the customer and take feedback from them.
- If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.



Evolutionary process flow

UMBRELLA ACTIVITIES

- Activities that occur throughout a software process for better management and tracking of the project.

1. Software Project Tracking and Control:

- Compare the progress of the project with the plan and take steps to maintain a planned schedule.

2. Risk Management:

- Analyze any potential risks that may have an impact on the software product's quality and outcome.

3. Software Quality Assurance (SQA):

- These are activities required to maintain software quality.
- Perform actions to ensure the product's quality.

UMBRELLA ACTIVITIES

4. Technical Reviews :

- Assessment of errors and correction done at each stage of activity.

5. Software Configuration Management (SCM):

- Managing of configuration process when any change in the software occurs.

6. Reusability Management:

- Reusable work items should be backed up, and reusable software components should be achieved.

7. Work Product Preparation and Production:

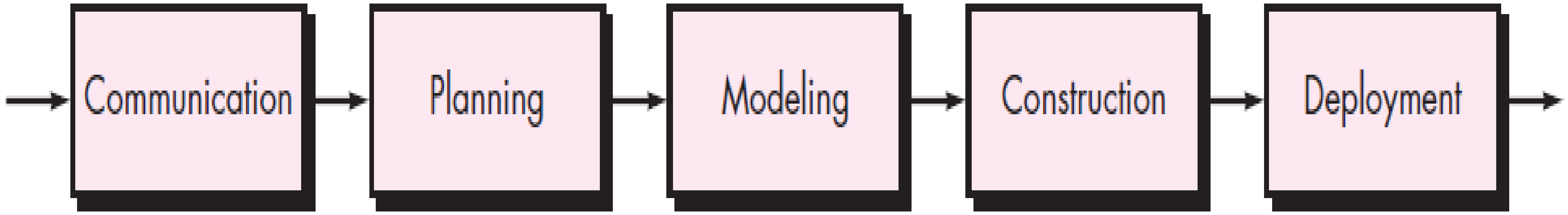
- The activities to create models, documents, logs, forms and lists are carried out.

PROCESS FLOW

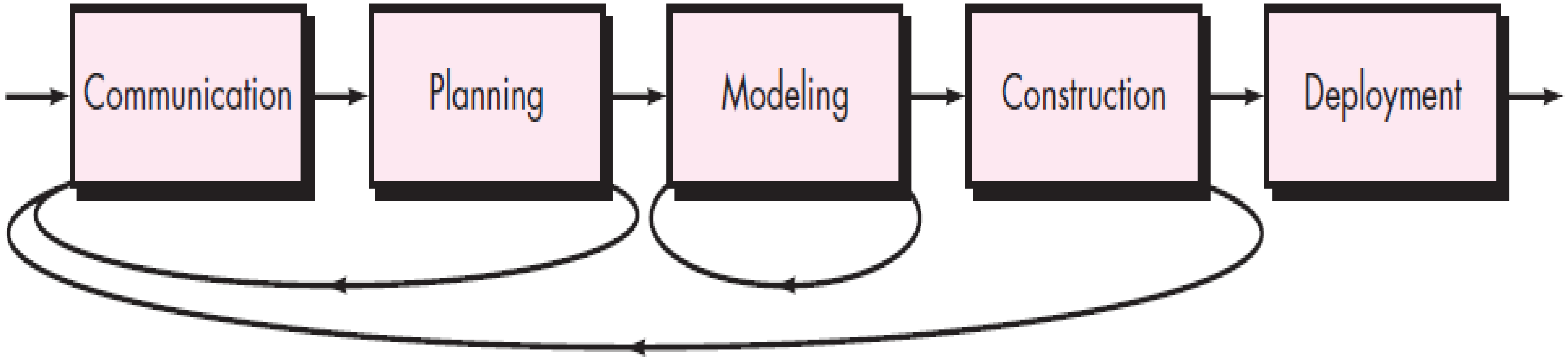
Process flow—describes how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to sequence and time

1. *Linear process flow*
2. *Iterative process flow*
3. *Evolutionary process flow*
4. *Parallel process flow*

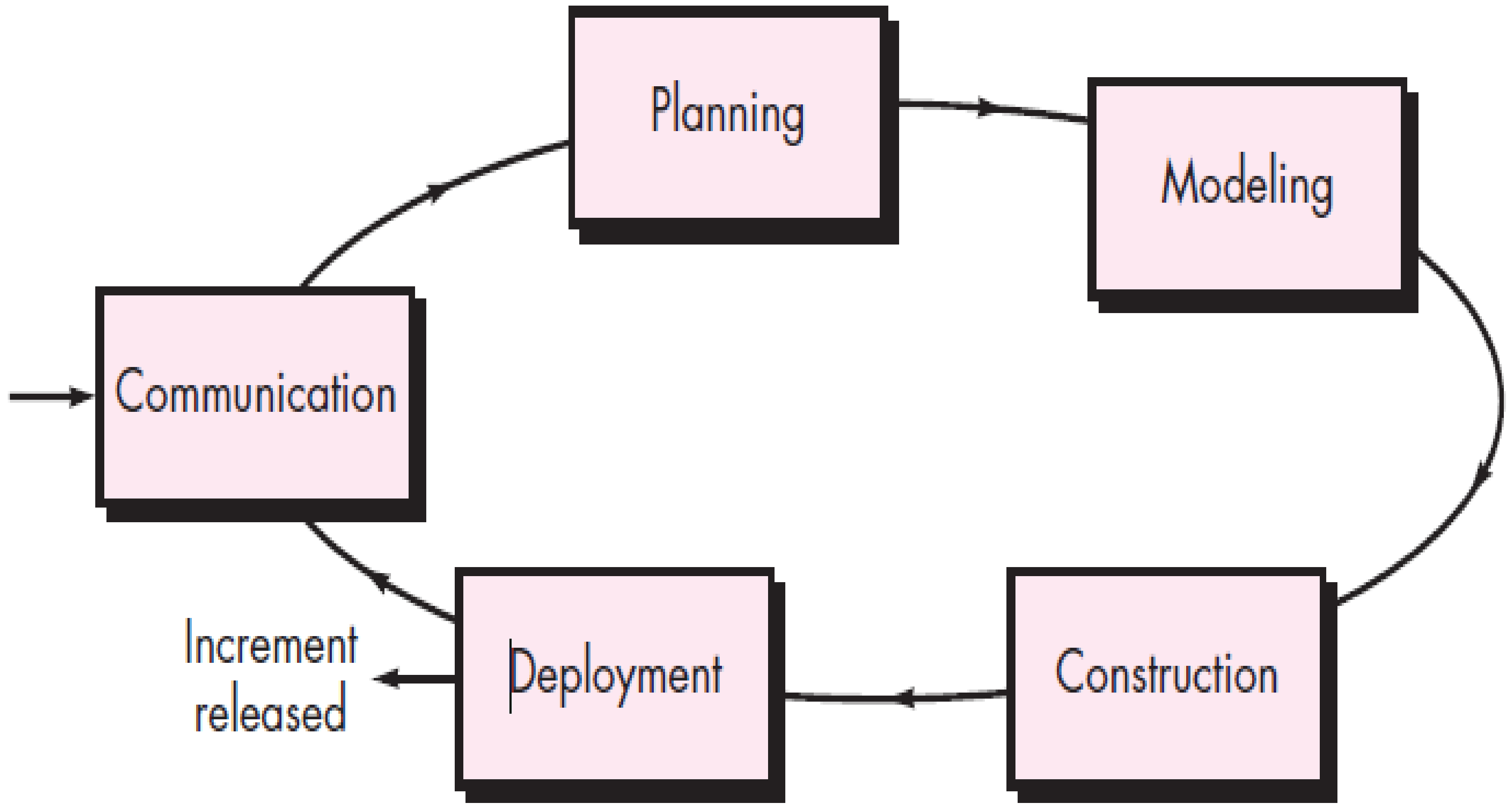
LINEAR PROCESS FLOW



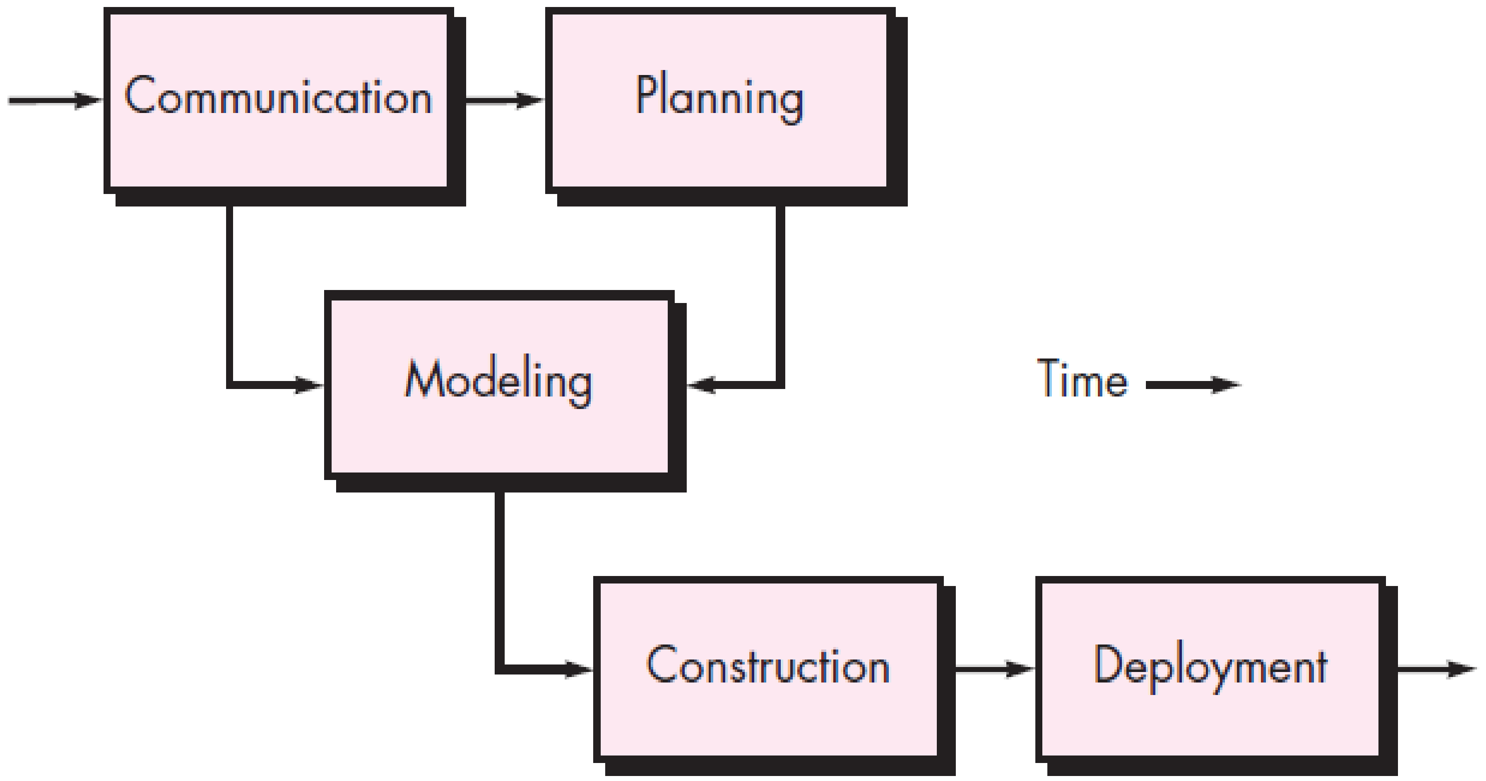
ITERATIVE PROCESS FLOW



EVOLUTIONARY PROCESS FLOW



PARALLEL PROCESS FLOW



DEFINING FRAMEWORK ACTIVITY

A software team would need significantly more information before it could properly execute any one of these activities as part of the software process.

Therefore, A key question is faced:
What actions are appropriate for a framework activity, given the nature of the problem to be solved, the characteristics of the people doing the work, and the stakeholders who are sponsoring the project?

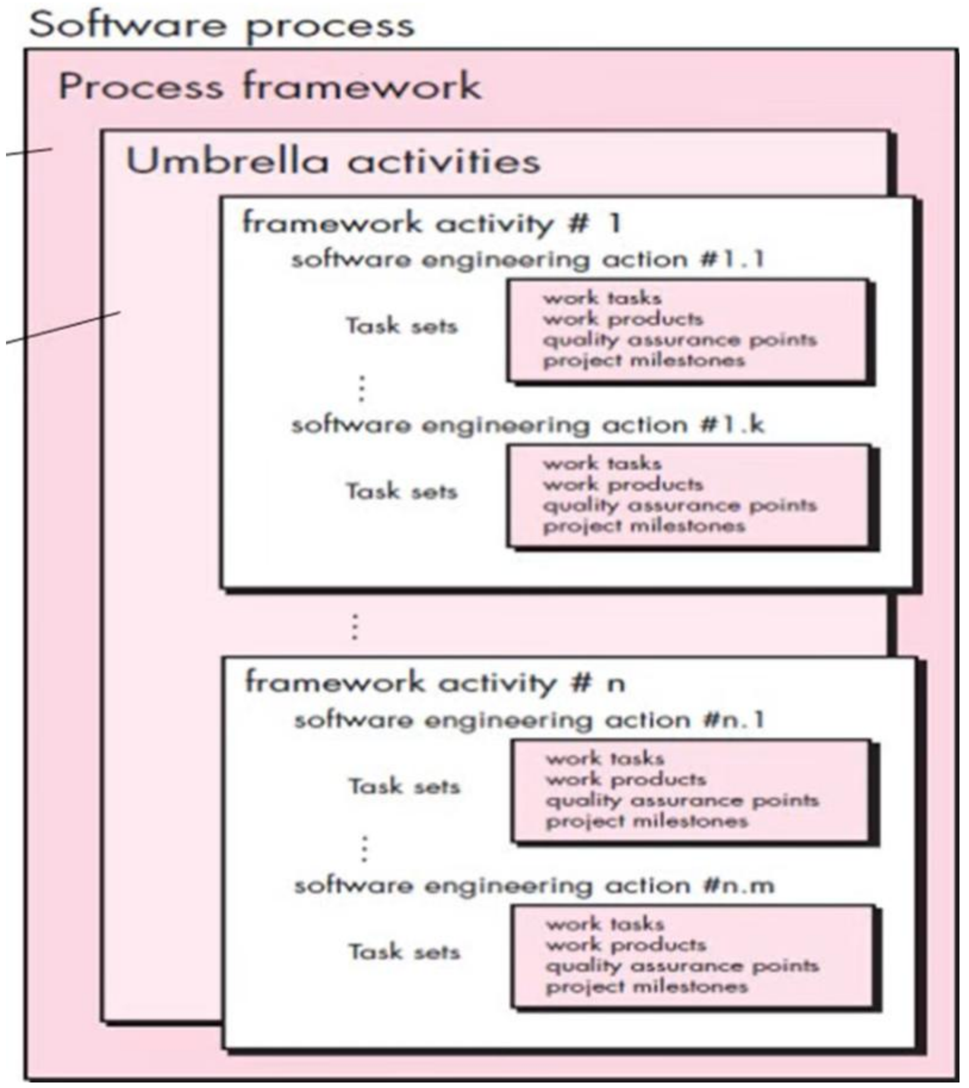
FRAMEWORK ACTIVITIES

➢ **Consider Communication Activity for small project:**

- Making a phone call with stakeholder.
- Discuss requirements and note them down.
- Organize requirements.
- Mail stakeholder for review and approval.

➢ **Consider Communication Activity for large project:**

- Arrange live meeting
- Complete feasibility study
- Requirement analysis
- Specification documents

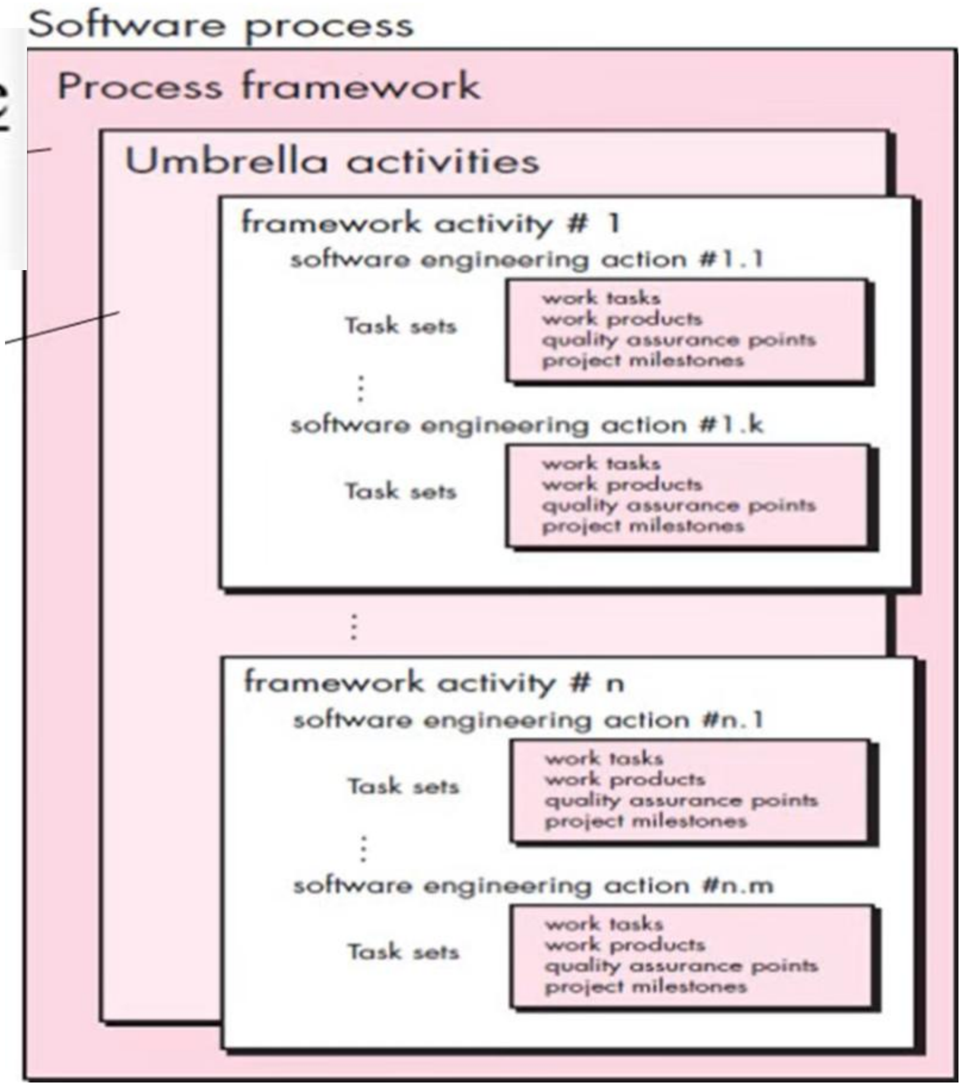


IDENTIFYING TASK SETS

➢**Task set** is the actual work to be done to achieve an objective of each engineering module.

➢**Consider Communication Activity Task Set:**

- Prepare a list of stakeholders of the project.
- Organize a meeting for stakeholders.
- Discuss requirements.
- Finalize requirements list.
- Make a list of issues raised.



PROCESS PATTERNS

Every software team encounters problems as it moves through the software process.

A *process pattern*[1] describes a process-related problem that is encountered during software engineering work, identifies the environment in which the problem has been encountered, and suggests one or more proven solutions to the problem.

A process pattern provides with a template

Patterns can be defined at any level of abstraction.[2] In some cases, a pattern might be used to describe a problem (and solution) associated with a complete process model (e.g., prototyping).
In other situations, patterns can be used to describe a problem (and solution) associated with a framework activity (e.g., **planning**) or an action within a framework activity (e.g., project estimating).

TEMPLATE FOR DESCRIBING A PROCESS PATTERN

Pattern Name. The pattern is given a meaningful name describing it within the context of the software process (e.g., **TechnicalReviews**).

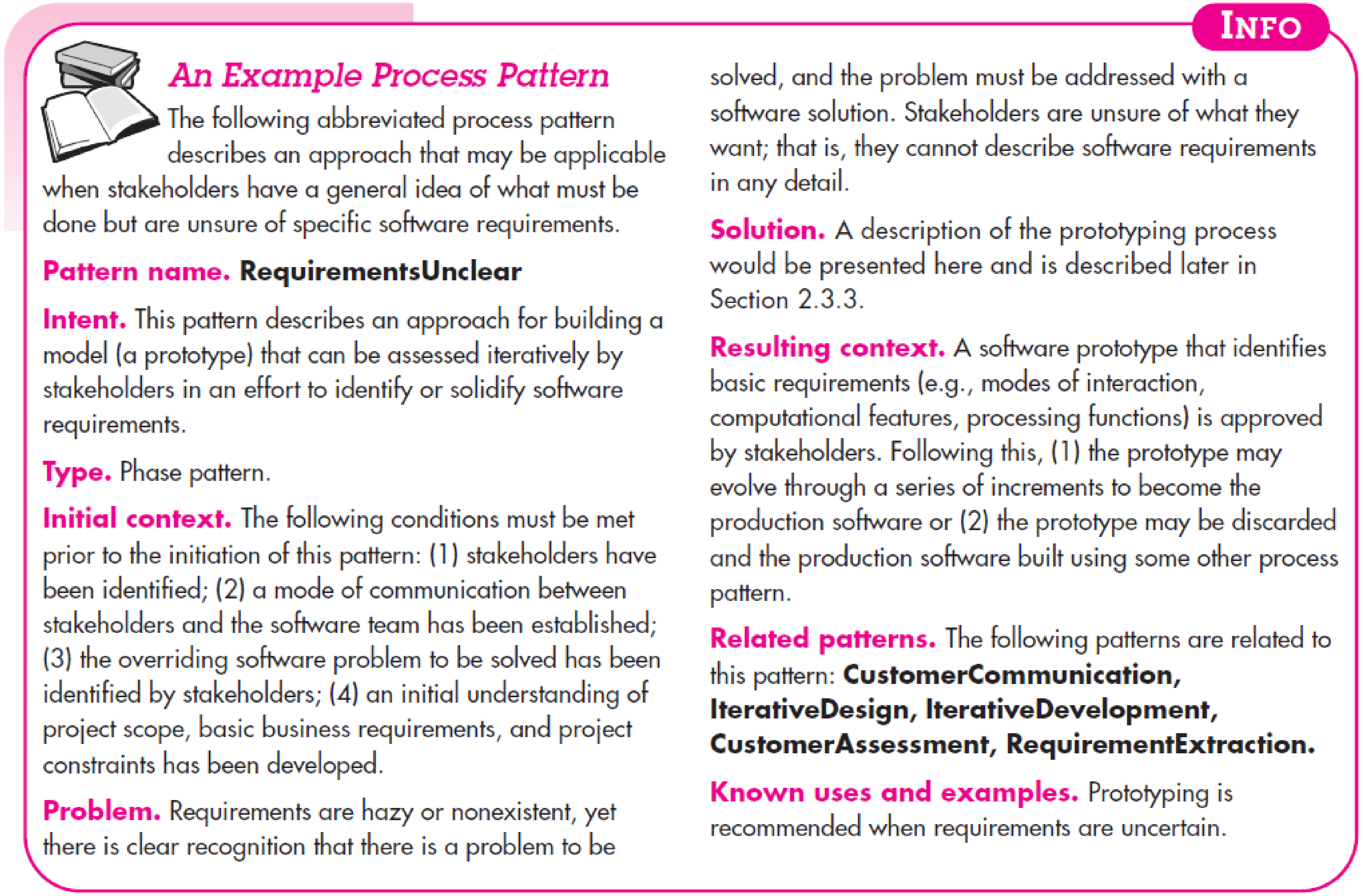
Forces. The environment in which the pattern is encountered and the issues that make the problem visible and may affect its solution.

Type. The pattern type is specified. There are three types:

1. Stage Pattern: defines a problem associated with a framework activity for the process.
Example: **EstablishingCommunication with** the task pattern **RequirementsGathering**

2. Task Pattern: defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice
Example: **RequirementsGathering**

3. Phase Pattern: define the sequence of framework activities that occurs within the process, even when the overall flow of activities is iterative in nature
Example: **SpiralModel** or **Prototyping**



INFO

An Example Process Pattern

The following abbreviated process pattern describes an approach that may be applicable when stakeholders have a general idea of what must be done but are unsure of specific software requirements.

Pattern name. RequirementsUnclear

Intent. This pattern describes an approach for building a model (a prototype) that can be assessed iteratively by stakeholders in an effort to identify or solidify software requirements.

Type. Phase pattern.

Initial context. The following conditions must be met prior to the initiation of this pattern: (1) stakeholders have been identified; (2) a mode of communication between stakeholders and the software team has been established; (3) the overriding software problem to be solved has been identified by stakeholders; (4) an initial understanding of project scope, basic business requirements, and project constraints has been developed.

Problem. Requirements are hazy or nonexistent, yet there is clear recognition that there is a problem to be solved, and the problem must be addressed with a software solution. Stakeholders are unsure of what they want; that is, they cannot describe software requirements in any detail.

Solution. A description of the prototyping process would be presented here and is described later in Section 2.3.3.

Resulting context. A software prototype that identifies basic requirements (e.g., modes of interaction, computational features, processing functions) is approved by stakeholders. Following this, (1) the prototype may evolve through a series of increments to become the production software or (2) the prototype may be discarded and the production software built using some other process pattern.

Related patterns. The following patterns are related to this pattern: **CustomerCommunication, IterativeDesign, IterativeDevelopment, CustomerAssessment, RequirementExtraction.**

Known uses and examples. Prototyping is recommended when requirements are uncertain.

PROCESS ASSESSMENT AND IMPROVEMENT

The existence of a software process is no guarantee that software will be delivered on time, that it will meet the customer's needs, or that it will exhibit the technical characteristics that will lead to long-term quality characteristics

A number of different approaches to software process assessment and improvement have been proposed:

Standard CMMI Assessment Method for Process Improvement (SCAMPI): provides a five-step process assessment model that incorporates five phases: initiating, diagnosing, establishing, acting, and learning.
CMM-Based Appraisal for Internal Process Improvement (CBA IPI): provides a diagnostic technique for assessing the relative maturity of a software organization; uses the SEI CMM as the basis for the assessment
SPICE (ISO/IEC15504): a standard that defines a set of requirements for software process assessment. The intent of the standard is to assist organizations in developing an objective evaluation of the efficacy of any defined software process
ISO 9001:2000 for Software: a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides. the standard is directly applicable to software organizations and companies

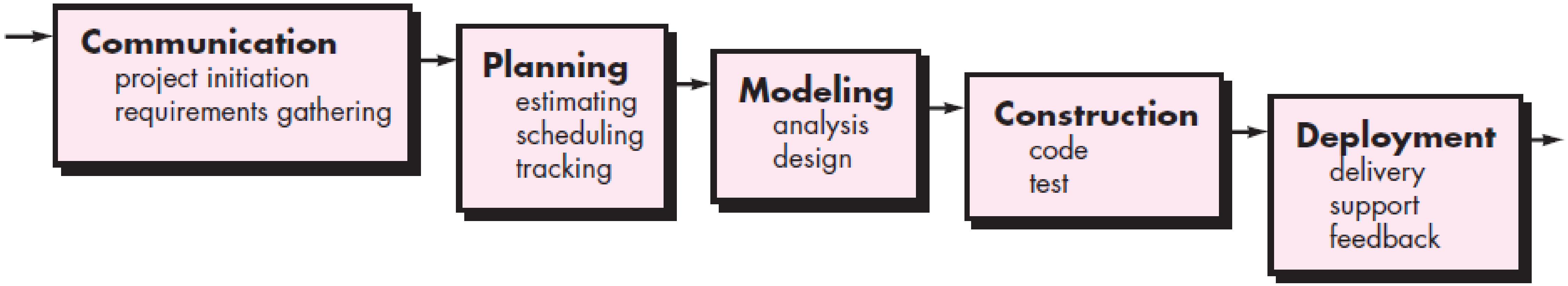
PRESCRIPTIVE PROCESS MODELS

Prescriptive process models were originally proposed to bring order to the chaos of software development.

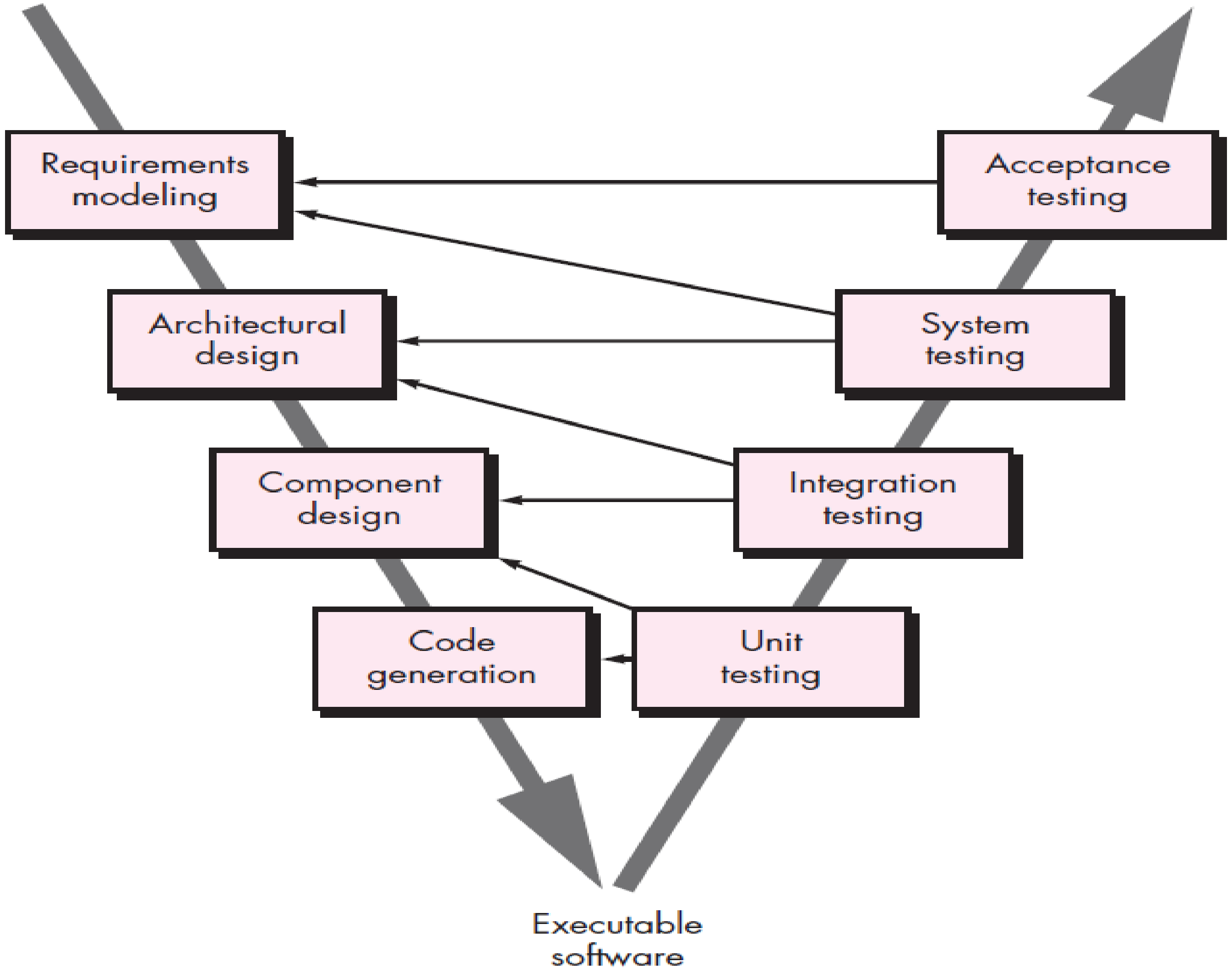
All software process models can accommodate the generic framework activities but each applies a different emphasis to these activities and defines a process flow that invokes each framework activity (as well as software engineering actions and tasks) in a different manner.

1. Waterfall Model
2. Incremental Process Model
3. Evolutionary Process Model
4. Concurrent Model

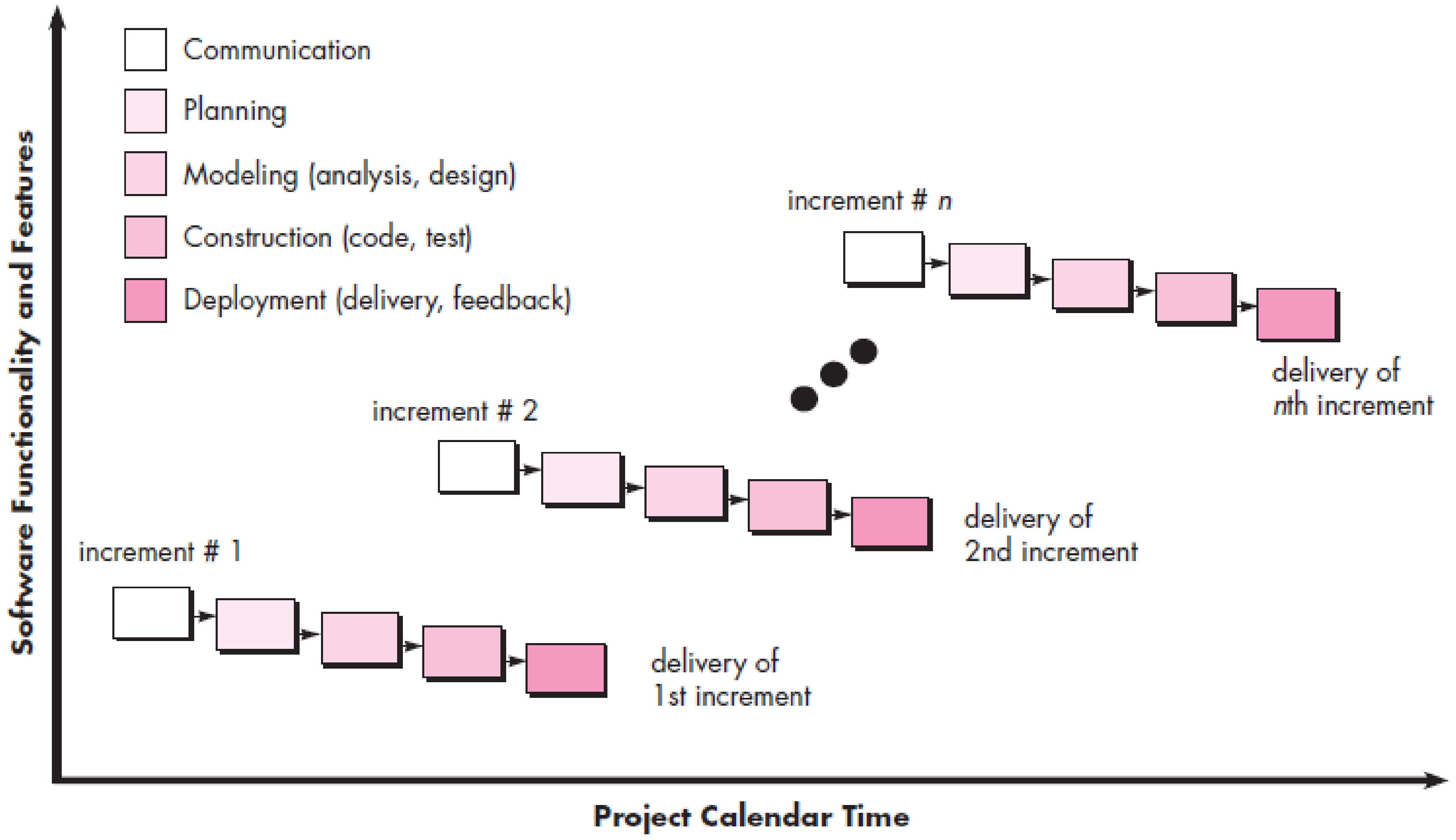
WATERFALL MODEL



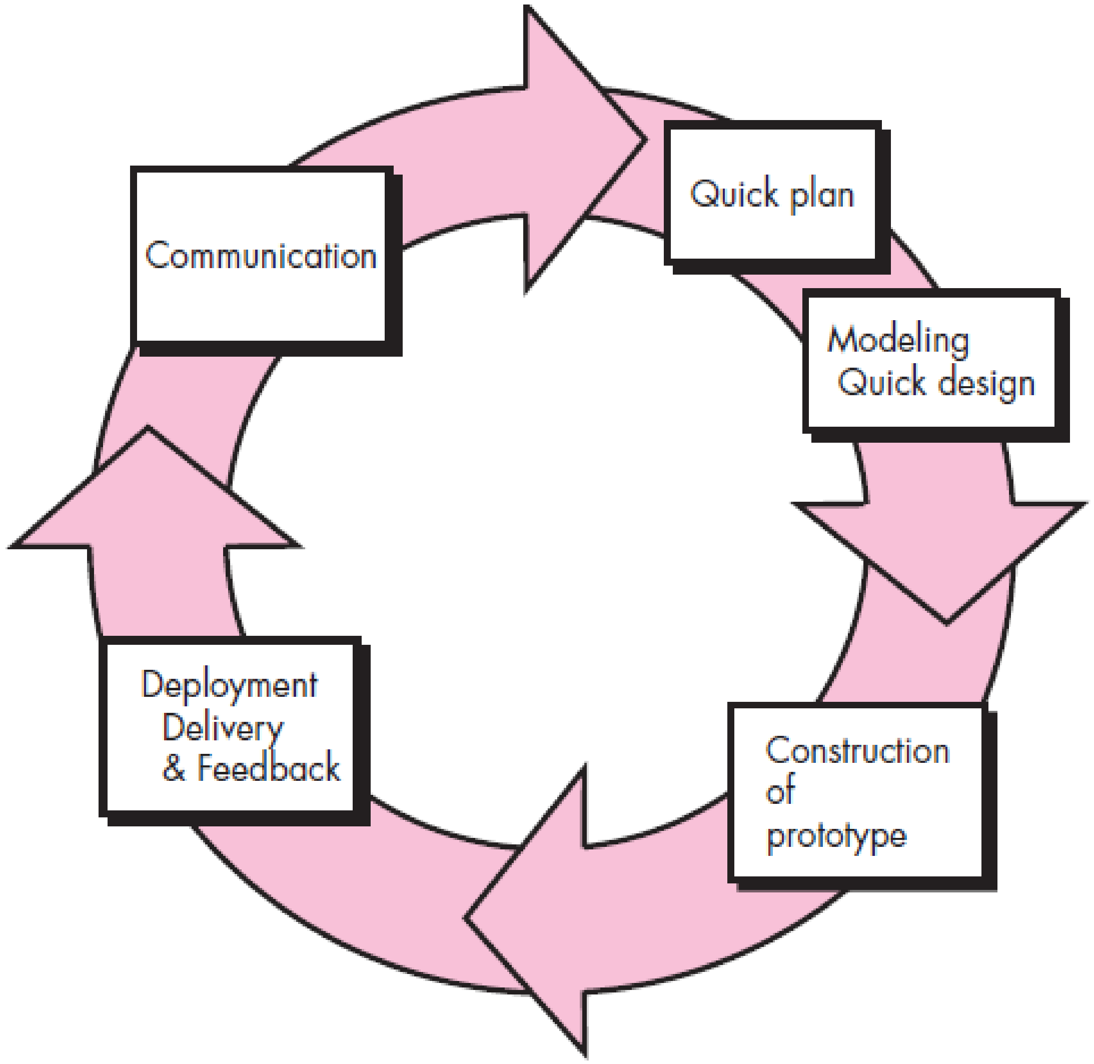
VARIATION IN WATERFALL MODEL- V MODEL



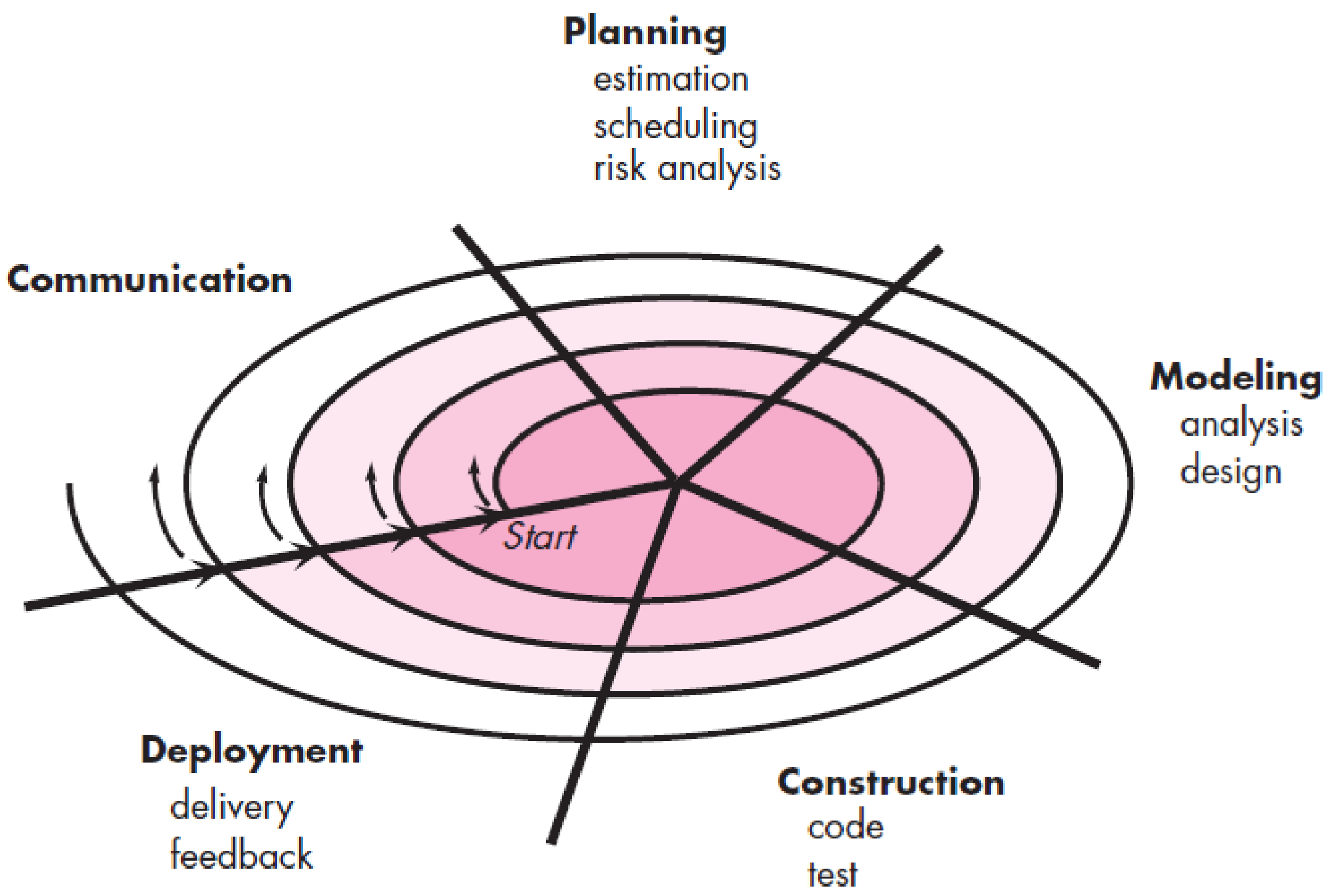
INCREMENTAL PROCESS MODEL


Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # 1
delivery of 1st increment
increment # 2
delivery of 2nd increment
increment # n
delivery of nth increment
Software Functionality and Features
Project Calendar Time

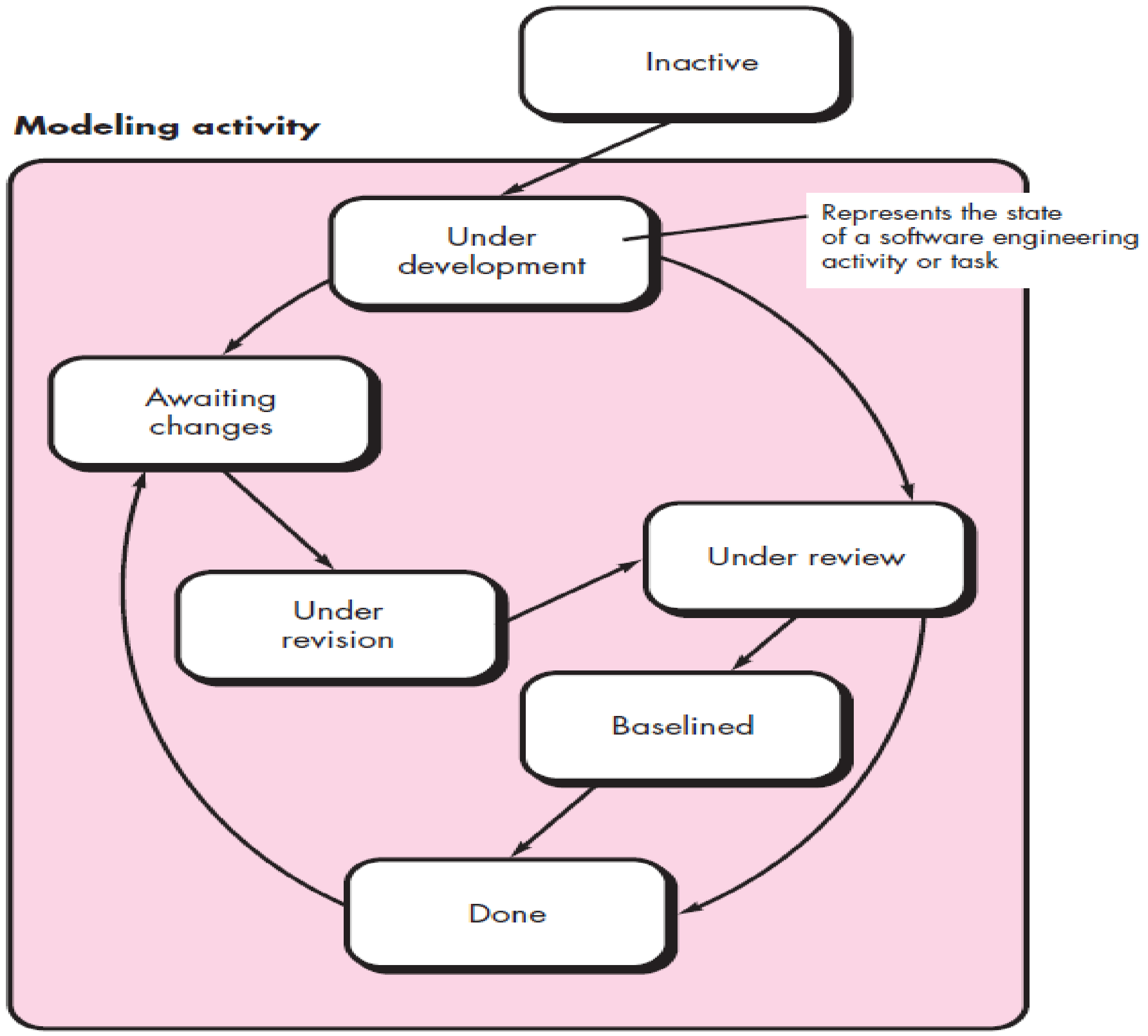
EVOLUTIONARY PROCESS MODEL



SPIRAL MODEL



CONCURRENT MODELS



SPECIALIZED PROCESS MODELS

These models tend to be applied when a specialized or narrowly defined software engineering approach is chosen

1. **Component-Based Development**
2. **Formal Methods Model**
3. **Aspect-Oriented Software Development**

COMPONENT BASED DEVELOPMENT

Commercial off-the-shelf (COTS) software components, developed by vendors who offer them as products, provide targeted functionality with well-defined interfaces that enable the component to be integrated into the software that is to be built. The *component-based development model* incorporates many of the characteristics of the spiral model erative approach to the creation of software.

The component-based develo ed software components. Modeling and construction a onents. These components can be design ted classes or packages of classes.
The component-based develo mented using an evolutionary approach)
1. Available component-based p plication domain in question.
2. Component integration issues a
3. A software architecture is designed to accommodate the components.
4. Components are integrated into the architecture.
5. Comprehensive testing is conducted to ensure proper functionality.

The component-based development model leads to software reuse, and reusability provides software engineers with a number of measurable benefits

FORMAL METHODS MODEL

The *formal methods model* encompasses a set of activities that leads to formal mathematical specification of computer software. Formal methods enable you to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation. A variation on this approach, called *cleanroom software engineering* is currently applied by some software development organizations.

Concerns about its applicability in a business environment:

- The development of formal models is currently quite time consuming and expensive.
- Because few software developers have the necessary background to apply formal methods, extensive training is required.
- It is difficult to use the models as a communication mechanism for technically unsophisticated customers.

ASPECT-ORIENTED SOFTWARE DEVELOPMENT

Regardless of the software process that is chosen, the builders of complex software invariably implement a set of localized features, functions, and information content.

These localized software characteristics are modeled as components (e.g., object oriented classes) and then constructed within the context of a system architecture.

As modern computer-based systems become more sophisticated (and complex), certain *concerns*—customer required properties or areas of technical interest—span the entire architecture.

When concerns cut across multiple system functions, features, and information, they are often referred to as *crosscutting concerns. Aspectual requirements* define those crosscutting concerns that have an impact across the software architecture.

Aspect-oriented software development (AOSD), often referred to as *aspect-oriented programming* (AOP), is a relatively new software engineering paradigm that provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*—"mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concern"

Thank you