QUEZON CITY UNIVERSITY
COLLEGE OF COMPUTER STUDIES

WEEK 3-4
THE SOFTWARE PROCESS

SE 101 – SOFTWARE ENGINEERING

LEARNING OUTCOMES:

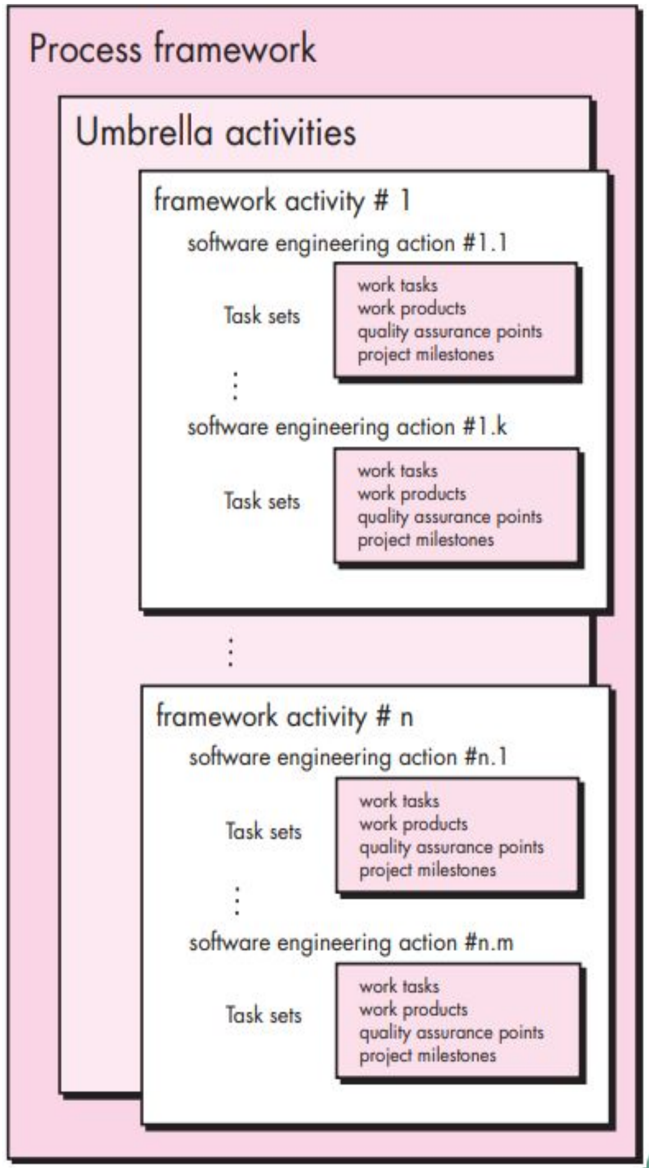
- Distinguish various software process models.
- Identify process technology tools that can be used to allocate, monitor, and even control all software engineering activities, actions, and tasks.

SOFTWARE PROCESS

A process was defined as a collection of work activities, actions, and tasks that are performed when some work product is to be created. Each of these activities, actions, and tasks reside within a framework or model that defines their relationship with the process and with one another.

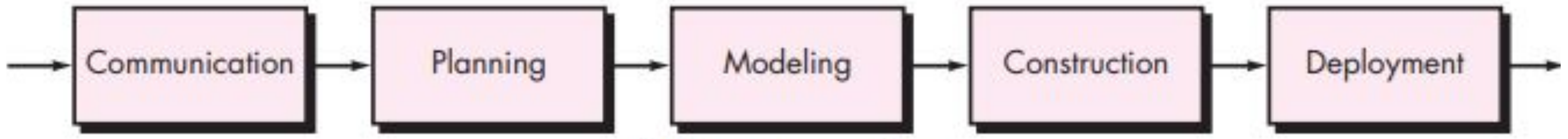
SOFTWARE PROCESS

Therefore; Referring to the figure, each framework activity is populated by a set of software engineering actions. Each software engineering action is defined by a task set that identifies the work tasks that are to be completed, the work products that will be produced, the quality assurance points that will be required, and the milestones that will be used to indicate progress

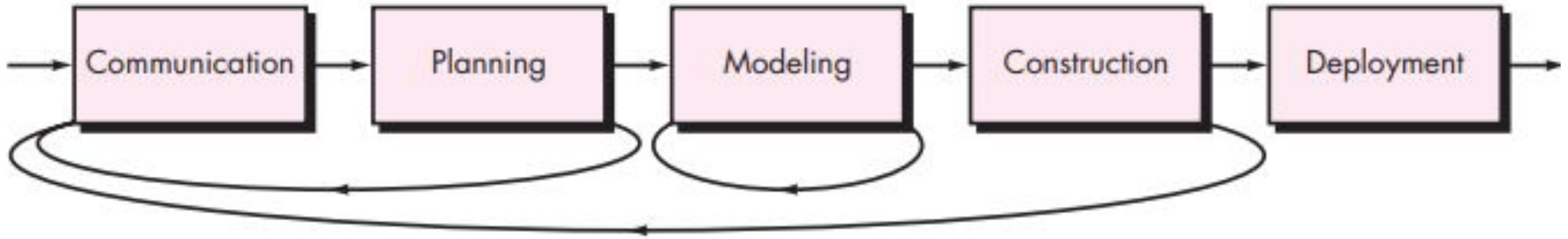


PROCESS FLOW

This aspect was called process flow that describes how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to sequence and time.

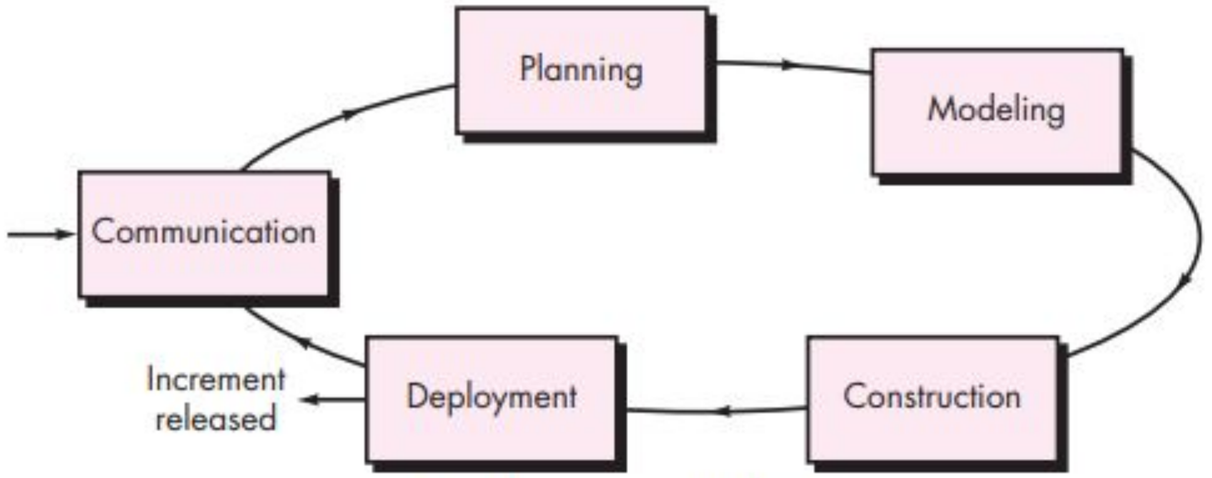


(a) Linear process flow

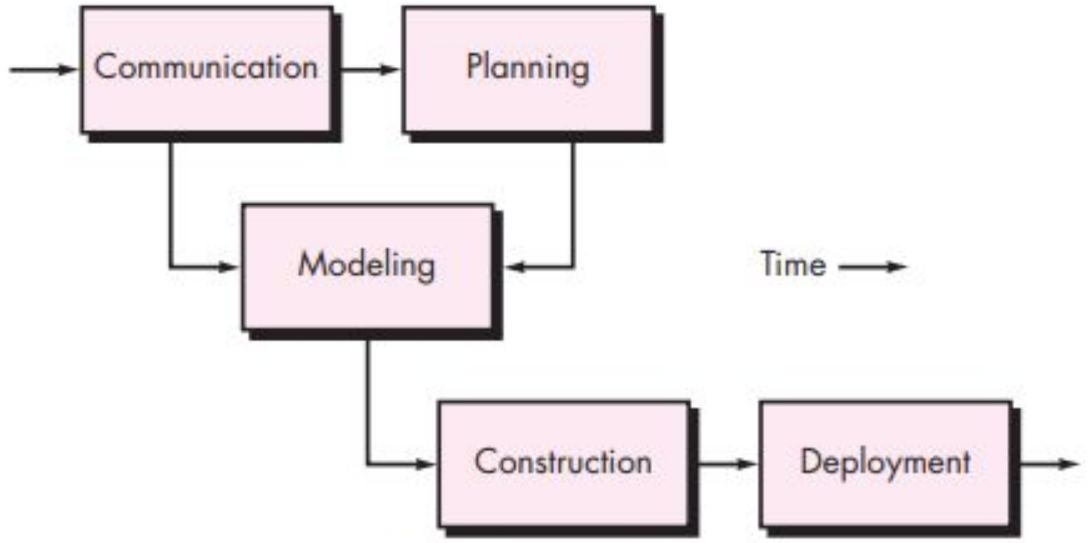


(b) Iterative process flow

WEEK 3-4 – THE SOFTWARE PROCESS



(c) Evolutionary process flow



(d) Parallel process flow

PROCESS PATTERNS

Process patterns provide an effective mechanism for addressing problems associated with any software process. The patterns enable you to develop a hierarchical process description that begins at a high level of abstraction (a phase pattern).

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

PROCESS PATTERN

Ambler has proposed a 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., Technical Reviews).

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.

PROCESS PATTERN

Ambler suggests three types:
1. **Stage pattern**—defines a problem associated with a framework activity for the process. Since a framework activity encompasses multiple actions and work tasks, a stage pattern incorporates multiple task patterns (see the following) that are relevant to the stage (framework activity). An example of a stage pattern might be Establishing Communication. This pattern would incorporate the task pattern Requirements Gathering and others.

PROCESS PATTERN

2. **Task pattern**—defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice (e.g., Requirements Gathering is a task pattern).

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. An example of a phase pattern might be Spiral Model or Prototyping.

PROCESS PATTERN

Initial context. Describes the conditions under which the pattern applies. Prior to the initiation of the pattern:
(1) What organizational or team-related activities have already occurred?
(2) What is the entry state for the process?
(3) What software engineering information or project information already exists?

Problem. The specific problem to be solved by the pattern.
Solution. Describes how to implement the pattern successfully.

PROCESS PATTERN

Resulting Context. Describes the conditions that will result once the pattern has been successfully implemented. Upon completion of the pattern:

(1) What organizational or team-related activities must have occurred?
(2) What is the exit state for the process?
(3) What software engineering information or project information has been developed?

PROCESS PATTERN

Related Patterns. Provide a list of all process patterns that are directly related to this one. This may be represented as a hierarchy or in some other diagrammatic form.

For example, the stage pattern Communication encompasses the task patterns: ProjectTeam, Collaborative Guidelines, Scope Isolation, Requirements Gathering, ConstraintDescription, and Scenario Creation

PROCESS PATTERN

Known Uses and Examples.
Indicate the specific instances in which the pattern is applicable. For example, Communication is mandatory at the beginning of every software project, is recommended throughout the software project, and is mandatory once the deployment activity is under way.

PROCESS ASSESSMENT AND IMPROVEMENT

A number of different approaches to software process assessment and improvement have been proposed over the past few decades:

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. The SCAMPI method uses the SEI CMMI as the basis for assessment.

PROCESS ASSESSMENT AND IMPROVEMENT

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 [Dun01].

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.

PROCESS ASSESSMENT AND IMPROVEMENT

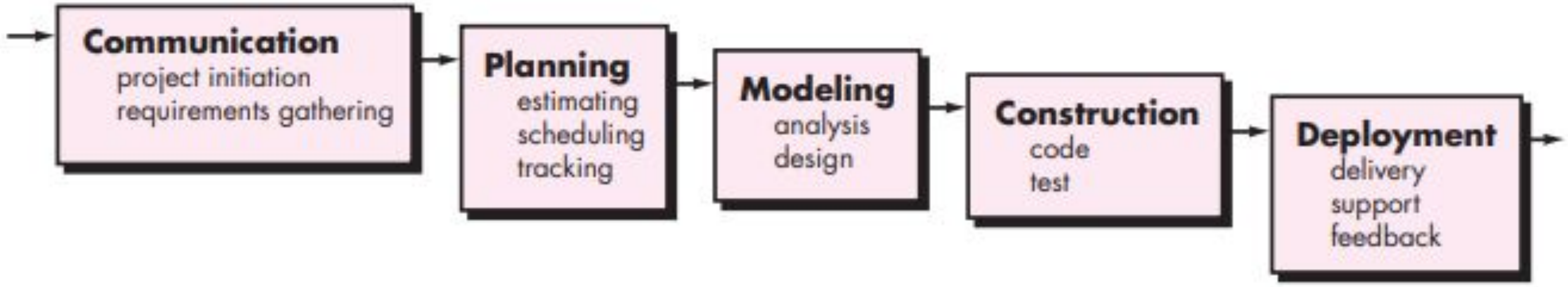
ISO 9001:2000 for Software—a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies.

PRESCRIPTIVE PROCESS MODELS

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

WATERFALL MODEL

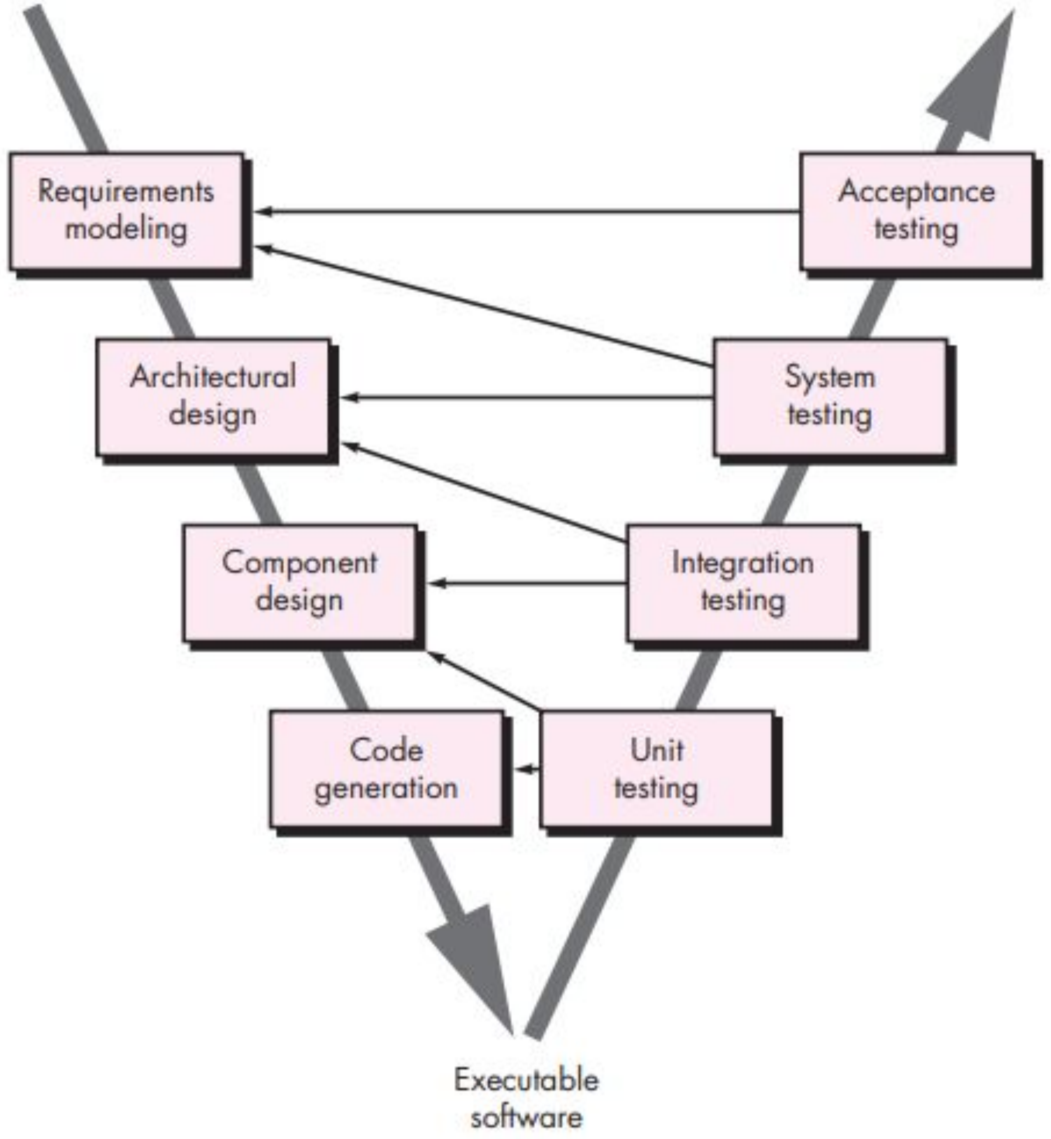
The waterfall model, sometimes called the classic life cycle, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software



V-MODEL

A variation in the representation of the waterfall model is called the V-model. The 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

WEEK 3-4 – THE SOFTWARE PROCESS



INCREMENTAL PROCESS MODEL

The incremental model combines elements of linear and parallel process flows.

the incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable "increments" of the software in a manner that is similar to the increments produced by an evolutionary process flow.

INCREMENTAL PROCESS MODEL

When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed but many supplementary features (some known, others unknown) remain undelivered. The core product is used by the customer (or undergoes detailed evaluation). As a result of use and/or evaluation, a plan is developed for the next increment.

EVOLUTIONARY PROCESS MODEL

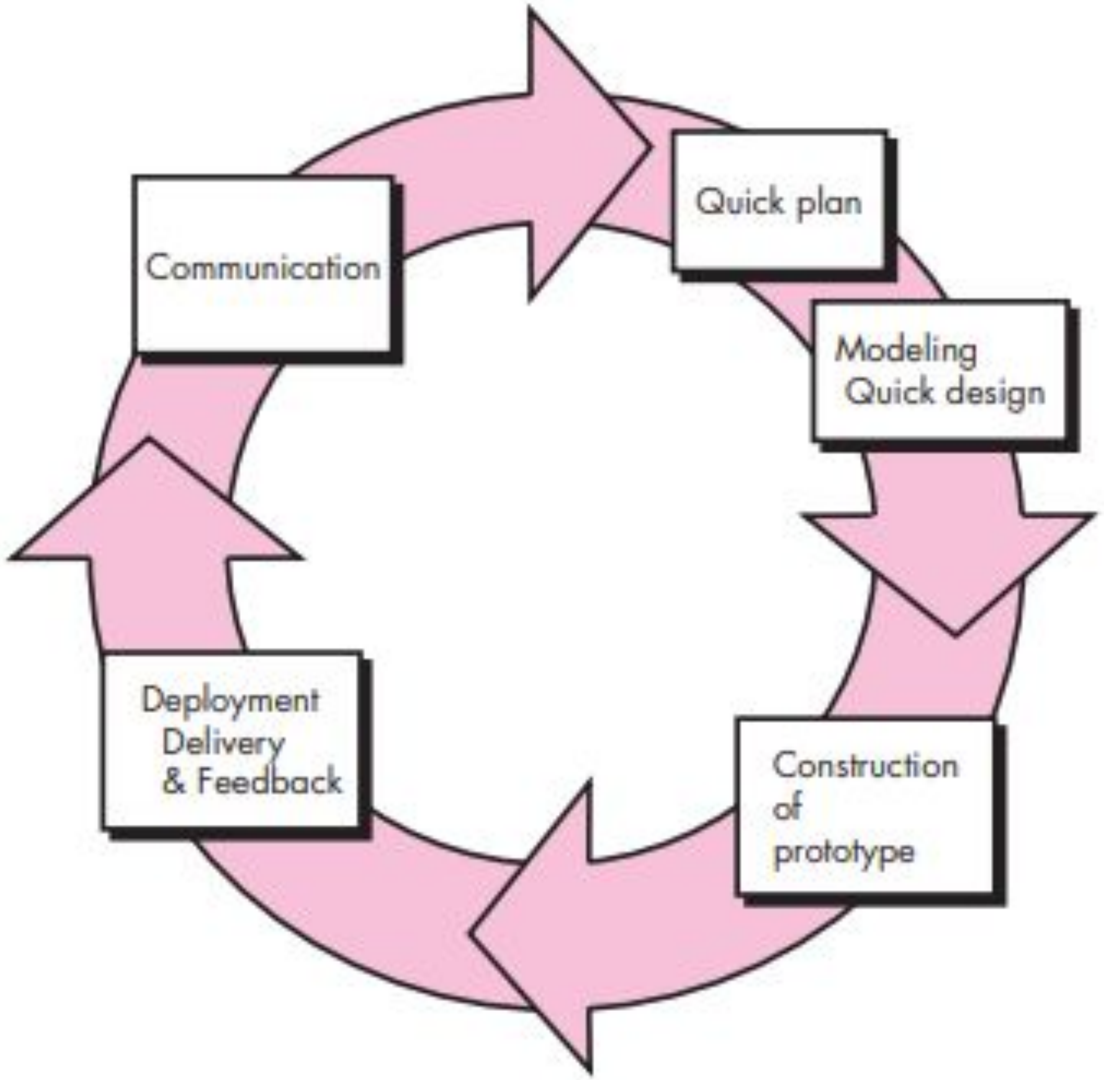
Software, like all complex systems, evolves over a period of time. Business and product requirements often change as development proceeds, making a straight line path to an end product unrealistic; tight market deadlines make completion of a comprehensive software product impossible, but a limited version 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 PROCESS MODEL

Evolutionary models are iterative. They are characterized in a manner that enables you to develop increasingly more complete versions of the software.

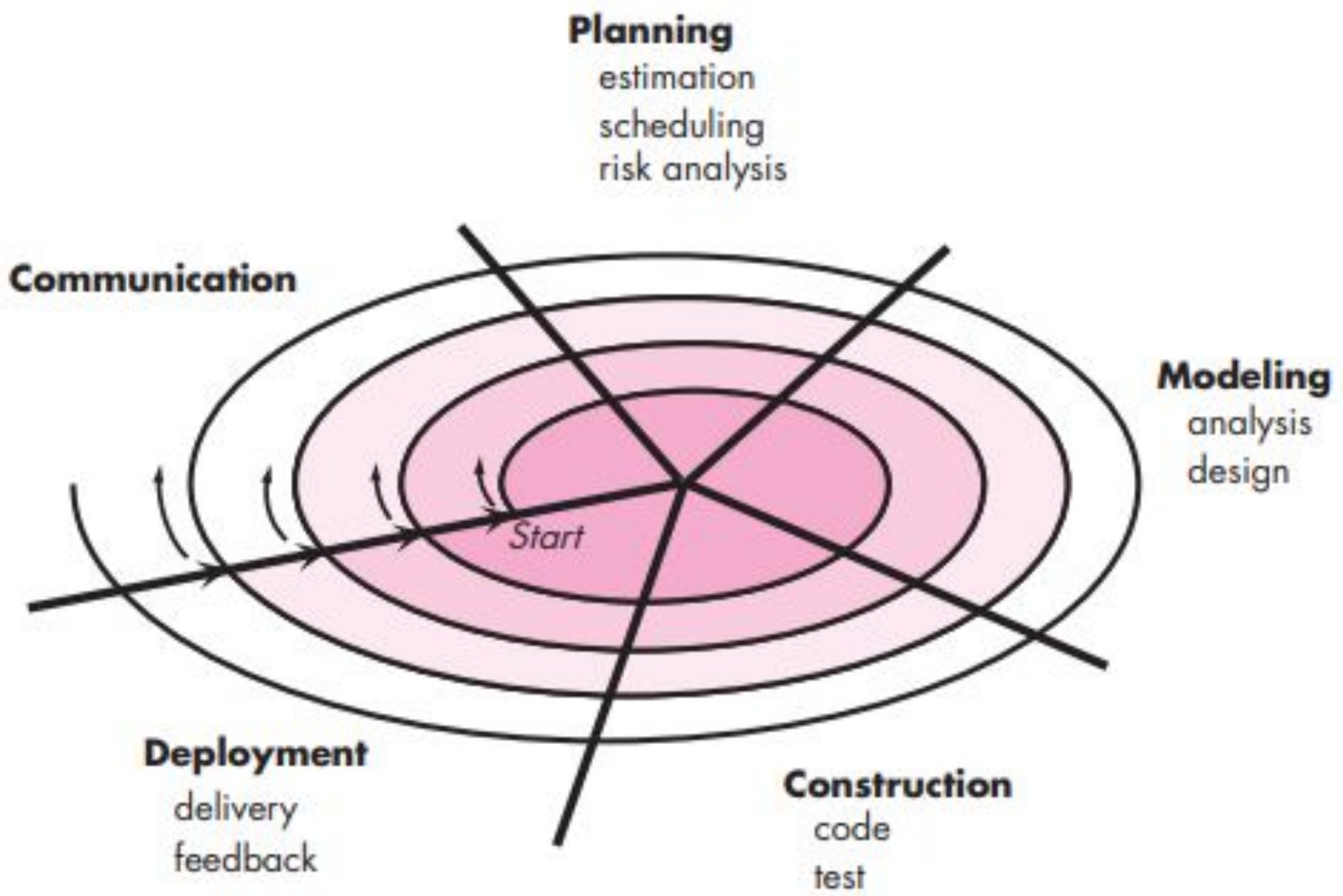
Prototyping. Often, a customer defines a set of general objectives for software, but does not identify detailed requirements for functions and features. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take. In these, and many other situations, a prototyping paradigm may offer the best approach

WEEK 3-4 – THE SOFTWARE PROCESS



SPIRAL MODEL

Originally proposed by Barry Boehm [Boe88], 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 [Boe01a] describes the model in the following manner


Planning
estimation
scheduling
risk analysis
Communication
Modeling
analysis
design
Start
Deployment
delivery
feedback
Construction
code
test

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 described in this chapter. 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.

SPECIALIZED PROCESS MODELS

Specialized process models take on many of the characteristics of one or more of the traditional models presented in the preceding sections. However, these models tend to be applied when a specialized or narrowly defined software engineering approach is chosen.

SPECIALIZED PROCESS MODELS

Specialized process models take on many of the characteristics of one or more of the traditional models presented in the preceding sections. However, these models tend to be applied when a specialized or narrowly defined software engineering approach is chosen.

The **component-based development model** leads to software reuse, and reusability provides software engineers with a number of measurable benefits. Your software engineering team can achieve a reduction in development cycle time as well as a reduction in project cost if component reuse becomes part of your culture

SPECIALIZED PROCESS MODELS

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.

When formal methods are used during development, they provide a mechanism for eliminating many of the problems that are difficult to overcome using other software engineering paradigms. Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily—not through ad hoc review, but through the application of mathematical analysis.

SPECIALIZED PROCESS MODELS

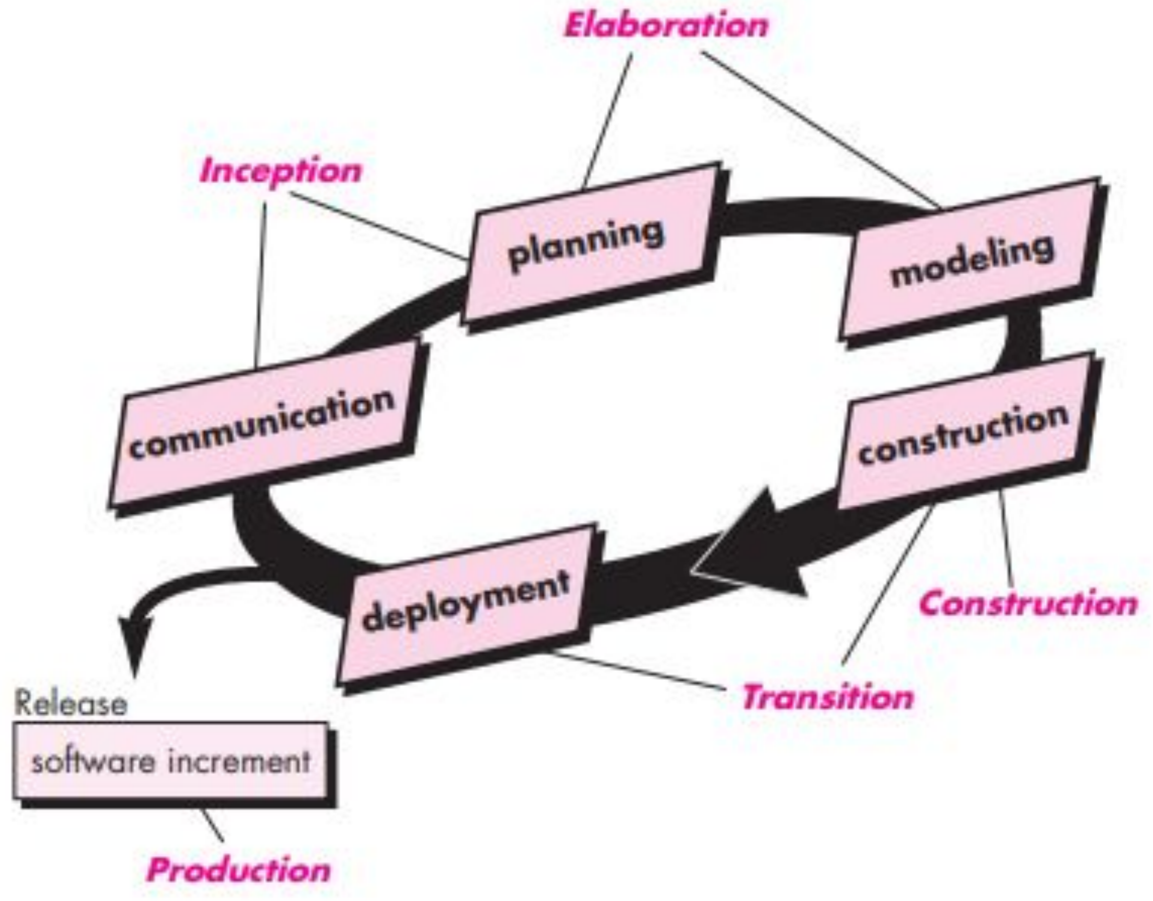
When formal methods are used during design, they serve as a basis for program verification and therefore enable you to discover and correct errors that might otherwise go undetected.

THE UNIFIED PROCESS

The **Unified Process** is an attempt to draw on the best features and characteristics of traditional software process models, but characterize them in a way that implements many of the best principles of agile software development.

The Unified Process recognizes the importance of customer communication and streamlined methods for describing the customer’s view of a system. It emphasizes the important role of software architecture and “helps the architect focus on the right goals, such as understandability, reliance to future changes, and reuse”. It suggests a process flow that is iterative and incremental, providing the evolutionary feel that is essential in modern software development.

PHASES OF THE UNIFIED PROCESS



PHASES OF THE UNIFIED PROCESS

The **inception phase** of the UP encompasses both customer communication and planning activities. By collaborating with stakeholders, business requirements for the software are identified; a rough architecture for the system is proposed; and a plan for the iterative, incremental nature of the ensuing project is developed.

PHASES OF THE UNIFIED PROCESS

The **elaboration phase** encompasses the communication and modeling activities of the generic process model.

Elaboration refines and expands the preliminary use cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software—the use case model, the requirements model, the design model, the implementation model, and the deployment model

PHASES OF THE UNIFIED PROCESS

The **construction phase** of the UP is identical to the construction activity defined for the generic software process. Using the architectural model as input, the construction phase develops or acquires the software components that will make each use case operational for end users. To accomplish this, requirements and design models that were started during the elaboration phase are completed to reflect the final version of the software increment. All necessary and required features and functions for the software increment (i.e., the release) are then implemented in source code

PHASES OF THE UNIFIED PROCESS

The **transition phase** of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment (delivery and feedback) activity. Software is given to end users for beta testing and user feedback reports both defects and necessary changes

PHASES OF THE UNIFIED PROCESS

The **production phase** of the UP coincides with the deployment activity of the generic process. During this phase, the ongoing use of the software is monitored, support for the operating environment (infrastructure) is provided, and defect reports and requests for changes are submitted and evaluated.

PERSONAL AND TEAM PROCESS MODELS

The best software process is one that is close to the people who will be doing the work. If a software process model has been developed at a corporate or organizational level, it can be effective only if it is amenable to significant adaptation to meet the needs of the project team that is actually doing software engineering work. In an ideal setting, you would create a process that best fits your needs, and at the same time, meets the broader needs of the team and the organization. Alternatively, the team itself can create its own process, and at the same time meet the narrower needs of individuals and the broader needs of the organization. Watts Humphrey and argues that it is possible to create a "personal software process" and/or a "team software process." Both require hard work, training, and coordination, but both are achievable.

PERSONAL SOFTWARE PROCESS (PSP)

It emphasizes personal measurement of both the work product that is produced and the resultant quality of the work product.

The PSP model defines five framework activities:

Planning. This activity isolates requirements and develops both size and resource estimates. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.

PERSONAL SOFTWARE PROCESS (PSP)

High-level design. External specifications for each component to be constructed are developed and a component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked.

High-level design review. Formal verification methods (Chapter 21) are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results

PERSONAL SOFTWARE PROCESS (PSP)

Development. The component-level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.

Postmortem. Using the measures and metrics collected (this is a substantial amount of data that should be analyzed statistically), the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness

TEAM SOFTWARE PROCESS (PSP)

The goal of TSP is to build a “self directed” project team that organizes itself to produce high-quality software.

Humphrey defines the following objectives for TSP:
• Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPTs) of 3 to about 20 engineers.

• Show managers how to coach and motivate their teams and how to help them sustain peak performance.

TEAM SOFTWARE PROCESS (PSP)

• Accelerate software process improvement by making CMM23 Level 5 behavior normal and expected.

• Provide improvement guidance to high-maturity organizations. • Facilitate university teaching of industrial-grade team skills.

TSP defines the following framework activities: project launch, high-level design, implementation, integration and test, and postmortem. Like their counterparts in PSP, these activities enable the team to plan, design, and construct software in a disciplined manner while at the same time quantitatively measuring the process and the product. The postmortem sets the stage for process improvements.

PROCESS TECHNOLOGY

One or more of the process models discussed in the preceding sections must be adapted for use by a software team. To accomplish this, process technology tools have been developed to help software organizations analyze their current process, organize work tasks, control and monitor progress, and manage technical quality. Process technology tools allow a software organization to build an automated model of the process framework, task sets, and umbrella activities. The model, normally represented as a network, can then be analyzed to determine typical workflow and examine alternative process structures that might lead to reduced development time or cost.

PRODUCT AND PROCESS

If the process is weak, the end product will undoubtedly suffer. But an obsessive over reliance on process is also dangerous.

People derive as much (or more) satisfaction from the creative process as they do from the end product. An artist enjoys the brush strokes as much as the framed result.

END OF PRESENTATION.
THANK YOU!