2019 级本科

软件工程

Software Engineering

张昕

zhangxin@cust.edu.cn

计算机科学技术学院软件工程系

Process Model

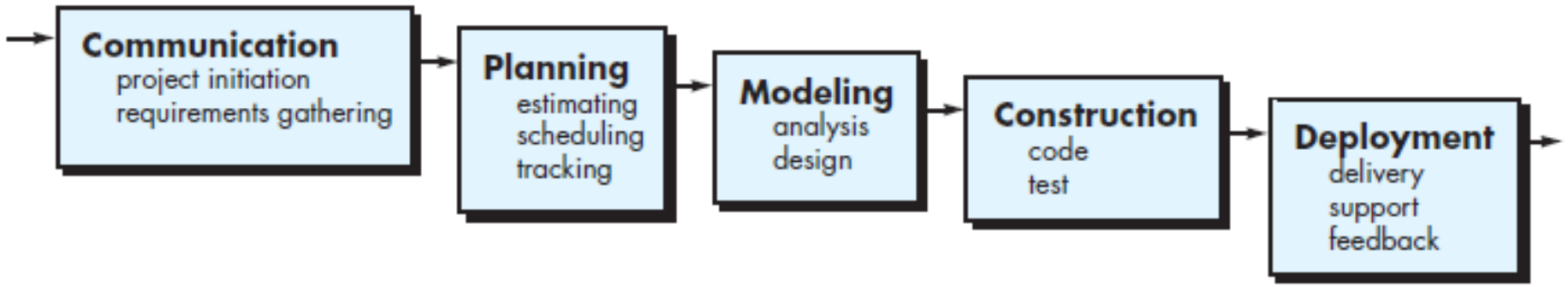
Process model

- Process models were originally proposed to bring order to the chaos of software development.
- A process model provides a specific roadmap for software engineering work.
  - It defines the flow of all activities, actions and tasks, the degree of iteration, the work products, and the organization of the work that must be done.
- Software engineers and their managers adapt a process model to their needs and then follow it.
  - In addition, the people who have requested the software have a role to play in the process of defining, building, and testing it.
- Why is process model important
  - Because process provides stability, control, and organization to an activity that can, if left uncontrolled, become quite chaotic.
  - However, a modern software engineering approach must be "agile."
  - It must demand only those activities, controls, and work products that are appropriate for the project team and the product that is to be produced.

Prescriptive process model

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.
- The waterfall model is the oldest paradigm for software engineering.



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

Prescriptive process model

Waterfall model

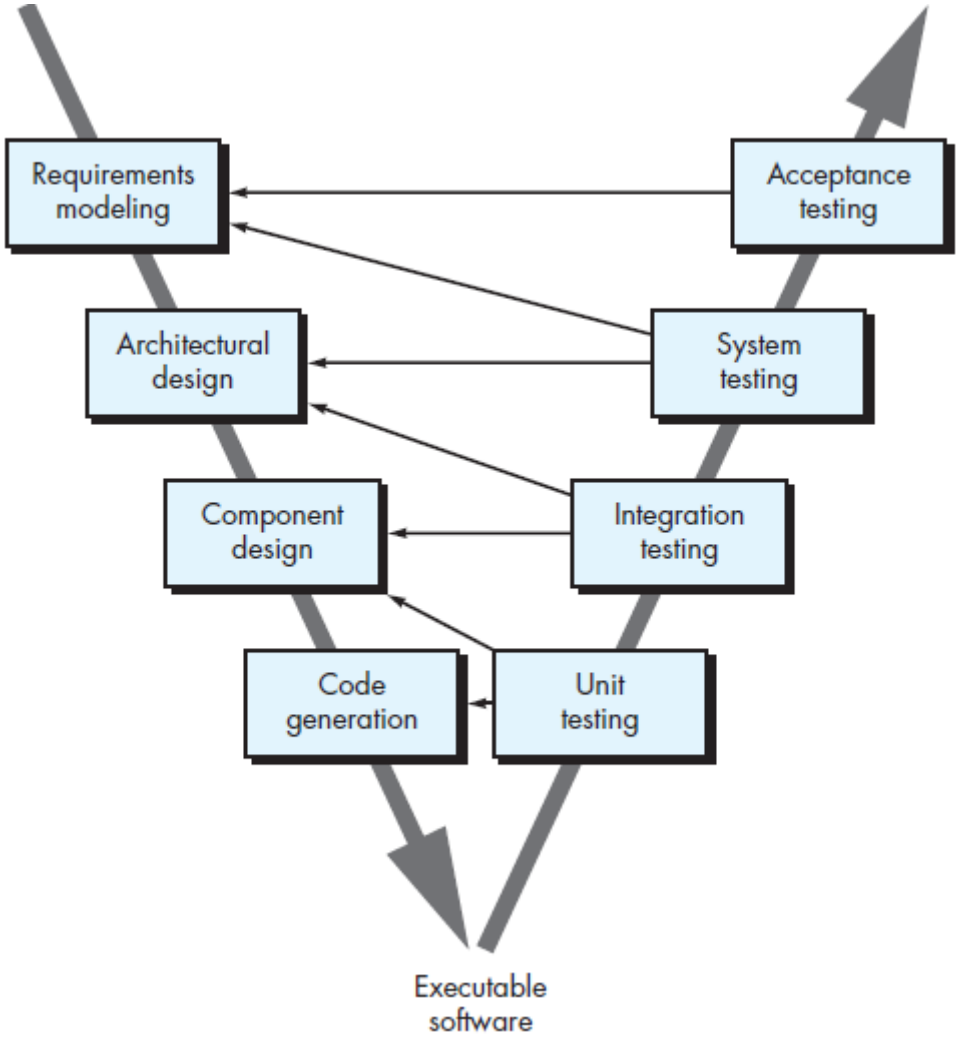
- Criticism
  1. Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.
  2. It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.
  3. The customer must have patience. A working version of the program(s) will not be available until late in the project time span. A major blunder, if undetected until the working program is reviewed, can be disastrous.

- The linear nature of the classic life cycle leads to "blocking states" in which some project team members must wait for other members of the team to complete dependent tasks.
- In fact, the time spent waiting can exceed the time spent on productive work!
- The blocking state tends to be more prevalent at the beginning and end of a linear sequential process.

Prescriptive process model

V model

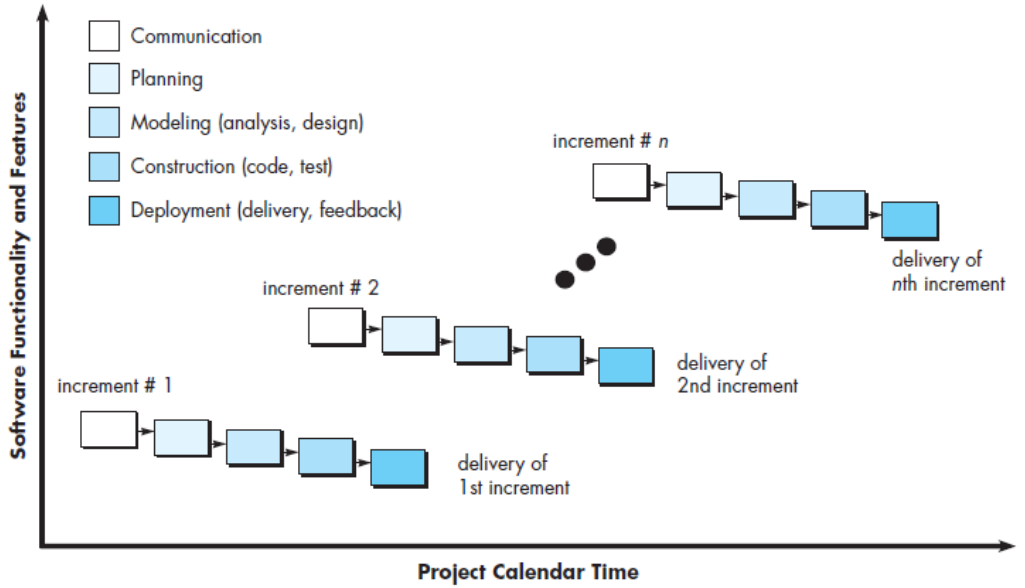
- V-model depicts the relationship of quality assurance actions to the actions associated with communication, modeling, and early construction activities.
- As a software team moves down the left side of the V, basic problem requirements are refined into progressively more detailed and technical representations 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 moves down the left side.
- There is no fundamental difference between the classic life cycle and the V-model. The V-model provides a way of visualizing how verifi cation and validation actions are applied to earlier engineering work.



Prescriptive process model

Incremental Process Models

- There are many situations in which initial software requirements are reasonably well defined, but the overall scope of the development effort precludes a purely linear process.
- In addition, there may be a compelling need to provide a limited set of software functionality to users quickly and then refine and expand on that functionality in later software releases.
- When an incremental model is used, the first increment is often a core product, i.e., 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. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality.



- This process is repeated following the delivery of each increment, until the complete product is produced.

Prescriptive process model

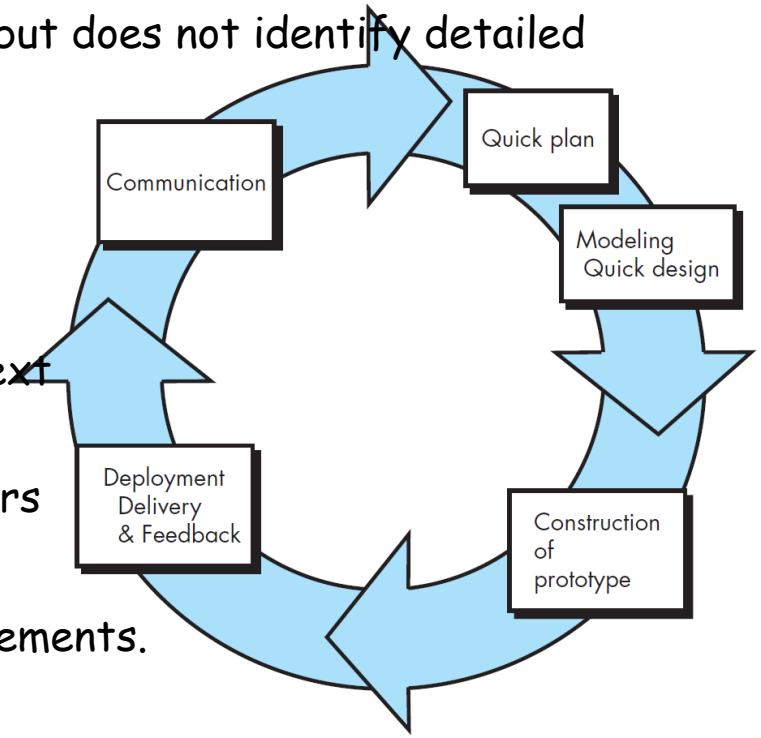
Evolutionary Process Models

- Evolutionary models are iterative. They are characterized in a manner that enables to develop increasingly more complete versions of the software.
- Common evolutionary process models:
  - Prototyping
  - The Spiral Model

Prescriptive process model

Evolutionary Process Models - Prototyping

- Usual delimma: a customer defines a set of general objectives for software, but does not identify detailed requirements for functions and features.
  - the efficiency of an algorithm
  - the adaptability of an operating system
  - the form that human-machine interaction
- More commonly used as a technique that can be implemented within the context of any one of the introduced process models.
- The prototyping paradigm assists designers/developers and other stakeholders to better understand what is to be built when requirements are fuzzy.
- Ideally, the prototype serves as a mechanism for identifying software requirements.
- The prototype can serve as "the first system."
  - *In most projects, the first system built is barely usable. It may be too slow, too big, awkward in use or all three. There is no alternative but to start again, smarting but smarter, and build a redesigned version in which these problems are solved.*
  - *Although some prototypes are built as "throwaways," others are evolutionary in the sense that the prototype slowly evolves into the actual system.*



Prescriptive process model

Evolutionary Process Models - Prototyping

- However, prototyping is problematic
  - Stakeholders see what appears to be a working version of the software, unaware that the prototype is held together haphazardly, unaware that in the rush to get it working you haven't considered overall software quality or long-term maintainability.
    - When informed that the product must be rebuilt so that high levels of quality can be maintained, stakeholders may demand that "a few fixes" be applied to make the prototype a working product.
  - Software engineers often make implementation compromises in order to get a prototype working quickly. An inappropriate operating system or programming language may be used simply because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability.
    - After a time, developers may become comfortable with these choices and forget all the reasons why they were inappropriate. The less-than-ideal choice has become an integral part of the system.

Prototyping can be an effective paradigm for software engineering.

The key is to define the rules that all stakeholders should agree that the prototype is built to serve as a mechanism for defining requirements.

Selecting a Process Model, Part 1

The scene: Meeting room for the software engineering group at CPI Corporation, a (fictional) company that makes consumer products for home and commercial use.

The players: Lee Warren, engineering manager; Doug Miller, software engineering manager; Jamie Lazar, software team member; Vinod Raman, software team member; and Ed Robbins, software team member.

The conversation:

Lee: So let's recapitulate. I've spent some time discussing the *SafeHome* product line as we see it at the moment. No doubt, we've got a lot of work to do to simply define the thing, but I'd like you guys to begin thinking about how you're going to approach the software part of this project.

Doug: Seems like we've been pretty disorganized in our approach to software in the past.

Ed: I don't know, Doug, we always got product out the door.

Doug: True, but not without a lot of grief, and this project looks like it's bigger and more complex than anything we've done in the past.

Jamie: Doesn't look that hard, but I agree . . . our ad hoc approach to past projects won't work here, particularly if we have a very tight time line.

Doug (smiling): I want to be a bit more professional in our approach. I went to a short course last week and learned a lot about software engineering . . . good stuff. We need a process here.

Jamie (with a frown): My job is to build computer programs, not push paper around.

Doug: Give it a chance before you go negative on me. Here's what I mean. (Doug proceeds to describe the process framework described in Chapter 3 and the prescriptive process models presented to this point.)

Doug: So anyway, it seems to me that a linear model is not for us . . . assumes we have all requirements up front and, knowing this place, that's not likely.

Vinod: Yeah, and it sounds way too IT-oriented . . . probably good for building an inventory control system or something, but it's just not right for *SafeHome*.

Doug: I agree.

Ed: That prototyping approach seems okay. A lot like what we do here anyway.

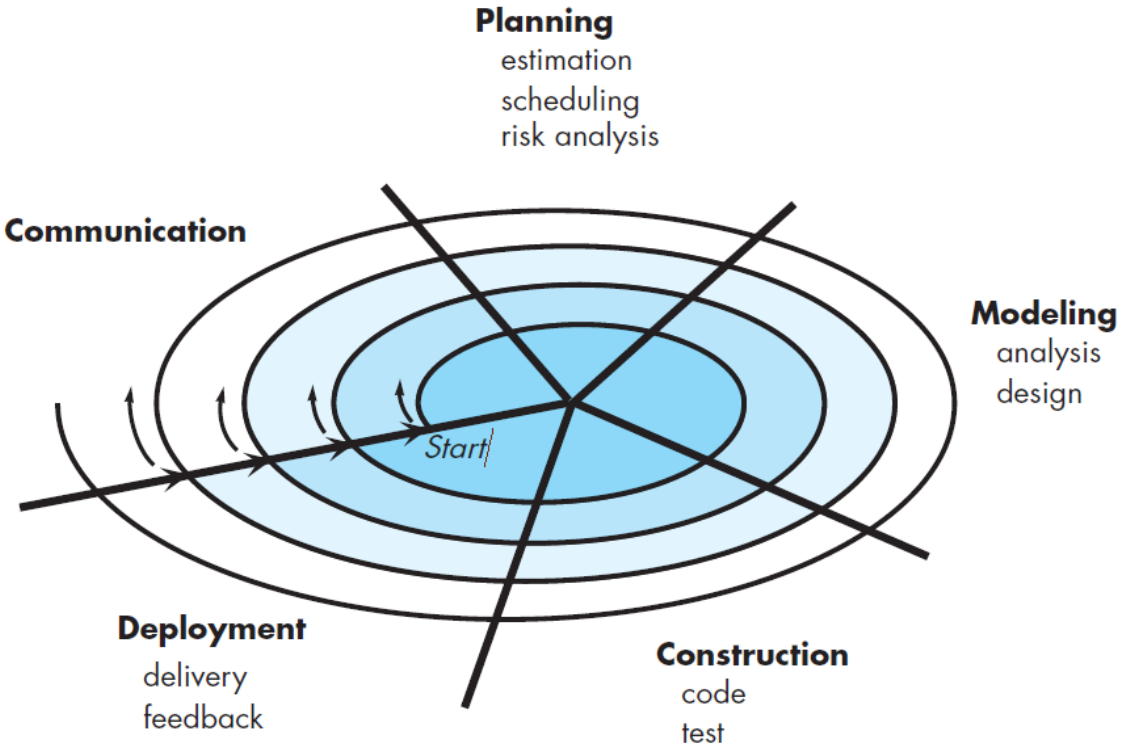
Vinod: That's a problem. I'm worried that it doesn't provide us with enough structure.

Doug: Not to worry. We've got plenty of other options, and I want you guys to pick what's best for the team and best for the project.

Prescriptive process model

Evolutionary Process Models - The Spiral Model

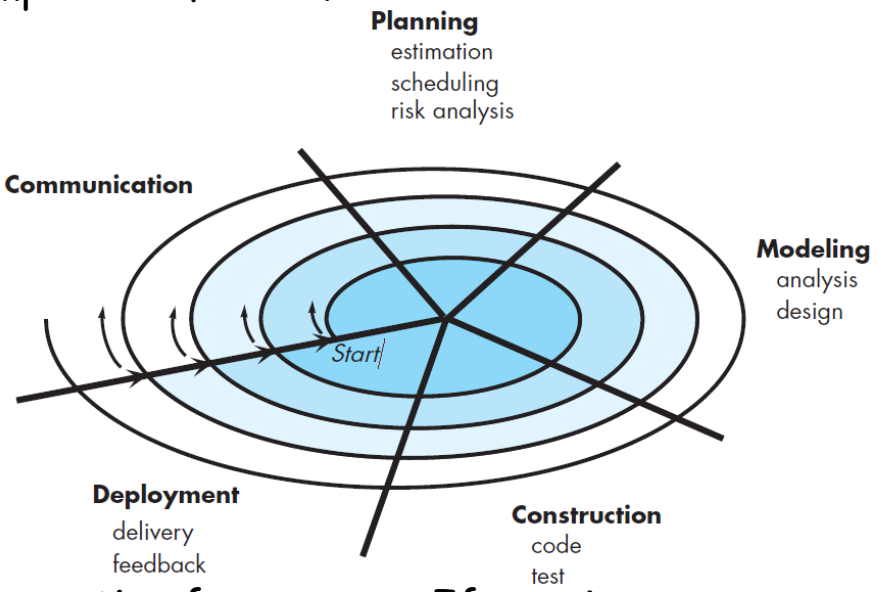
- It is the model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complete versions of the software.
  - The spiral development model is a risk-driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems.
  - It has two main distinguishing features:
    - A cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk.
    - A set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.



Prescriptive process model

Evolutionary Process Models - The Spiral Model

- Using the spiral model, software is developed in a series of evolutionary releases.
  - During early iterations, the release might be a model or prototype.
  - During later iterations, increasingly more complete versions of the engineered system are produced.
- The spiral model can be adapted to apply throughout the life of the computer software.
- The spiral model is a realistic approach to the development of large-scale systems and software.
  - Because software evolves as the process progresses, the developer and customer better understand and react to risks at each evolutionary level.
  - The spiral model uses prototyping as a risk reduction mechanism but, more important, enables designers/ developers to apply the prototyping approach at any stage in the evolution of the product.

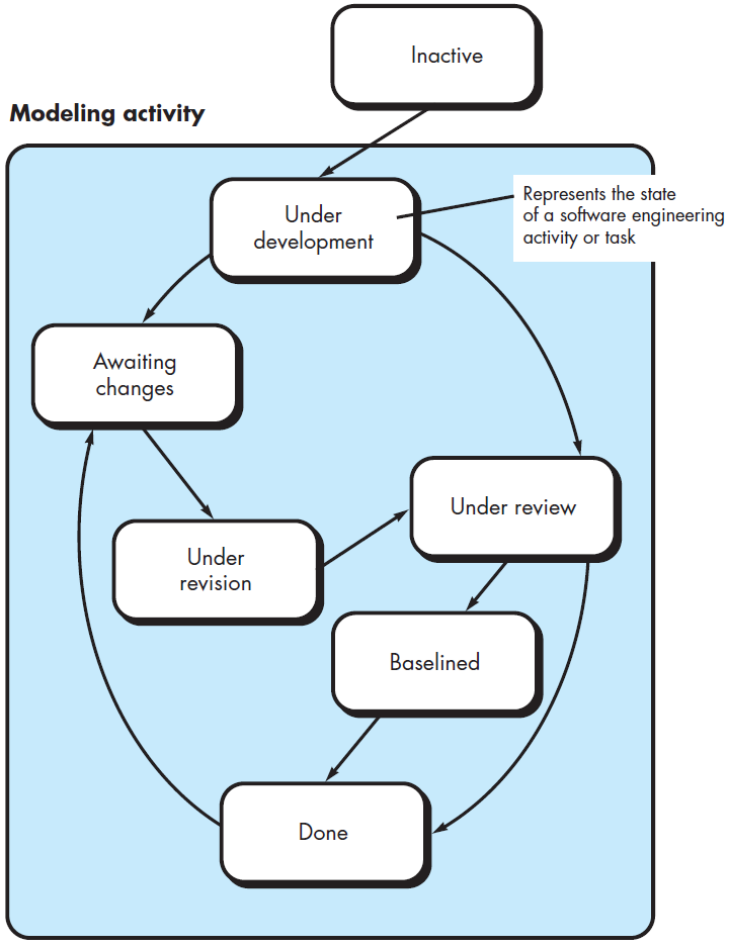


- It demands considerable risk assessment expertise and relies on this expertise for success. If a major risk is not uncovered and managed, problems will undoubtedly occur.

Prescriptive process model

Concurrent Models

- Sometimes called concurrent engineering. It allows a software team to represent iterative and concurrent elements of any of the other process models.
- 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.
- Concurrent modeling is applicable to all types of software development and provides an accurate picture of the current state of a project.
  - Rather than confining software engineering activities, actions, and tasks to a sequence of events, it defines a process network.
  - Each activity, action, or task on the network exists simultaneously with other activities, actions, or tasks.
  - Events generated at one point in the process network trigger transitions among the states associated with each activity.



Short Summary

- The challenge for software teams and their managers is to establish a proper balance between these critical project and product parameters and customer satisfaction (the ultimate arbiter of software quality).

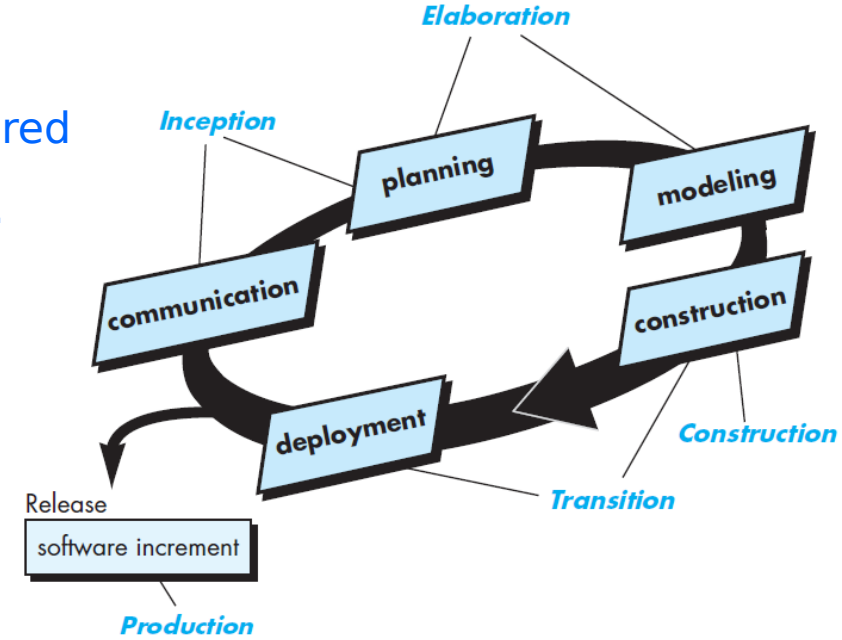
The Unified Process

- In some ways 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 (the use case).
  - 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 fl ow that is iterative and incremental, providing the evolutionary feel that is essential in modern software development.

Phases of the Unified Process

- The "phases" of the UP are related to the generic activities.
  - The Unified Process recognizes the importance of customer communication and streamlined methods for describing the customer's view of a system (the use case).
  - 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.

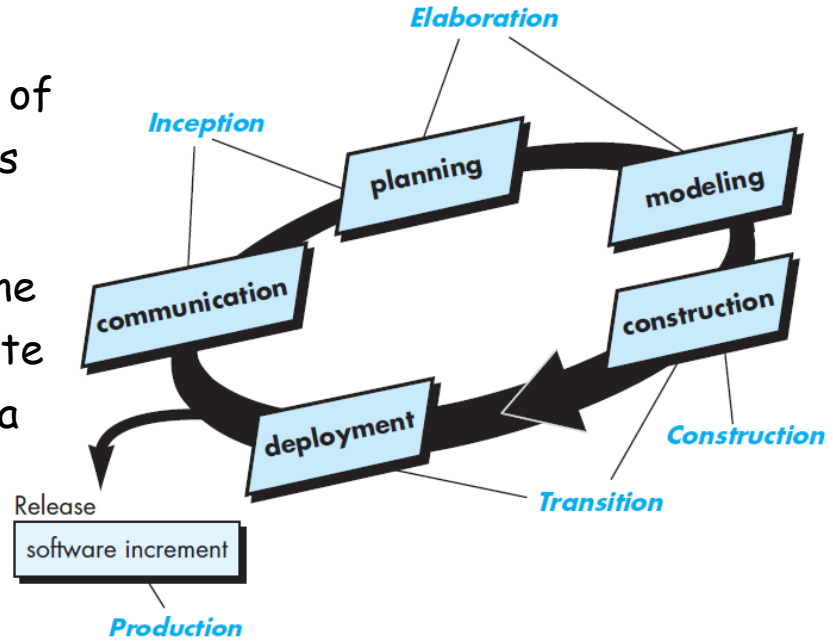
- The five UP phases do not occur in a sequence, but rather with staggered concurrency.
  - A software engineering workflow is distributed across all UP phases.
  - A workflow is analogous to a task set.
  - A workflow identifies the tasks required to accomplish an important software engineering action and the work products that are produced as a consequence of successfully completing the tasks.
  - Not every task identified for a UP workflow is conducted for every software project. The team adapts the process (actions, tasks, subtasks, and work products) to meet its needs.



Phases of the Unified Process

Inception phase

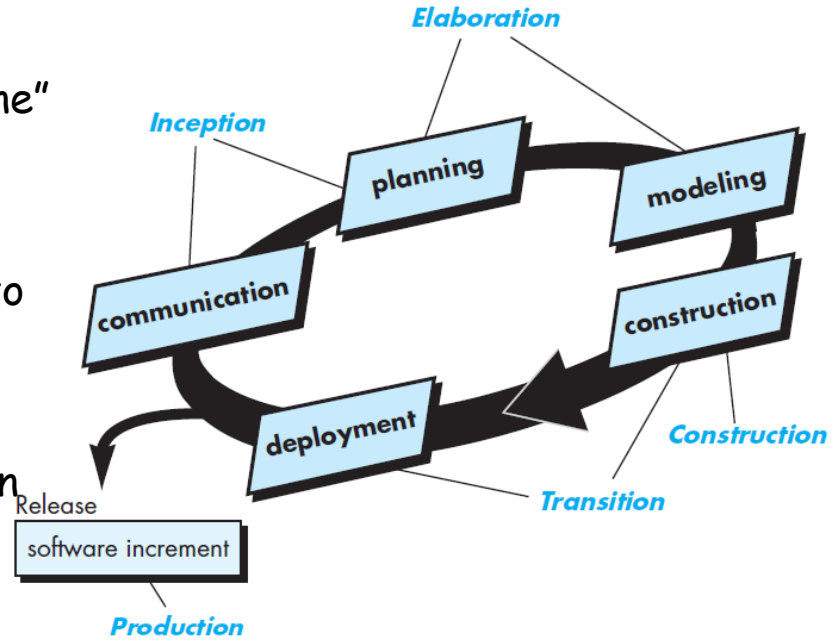
- encompasses both customer communication and planning activities.
- By collaborating with stakeholders, business requirements for the software are identified; a rough architecture for the system is proposed; and a plan for the iterative, incremental nature of the ensuing project is developed.
  - Fundamental business requirements are described through a set of preliminary use cases that describe which features and functions each major class of users desires.
  - Architecture at this point is nothing more than a tentative outline of major subsystems and the functions and features that populate them. Later, the architecture will be refined and expanded into a set of models that will represent different views of the system.
  - Planning identifies resources, assesses major risks, defines a schedule, and establishes a basis for the phases that are to be applied as the software increment is developed.



Phases of the Unified Process

Elaboration phase

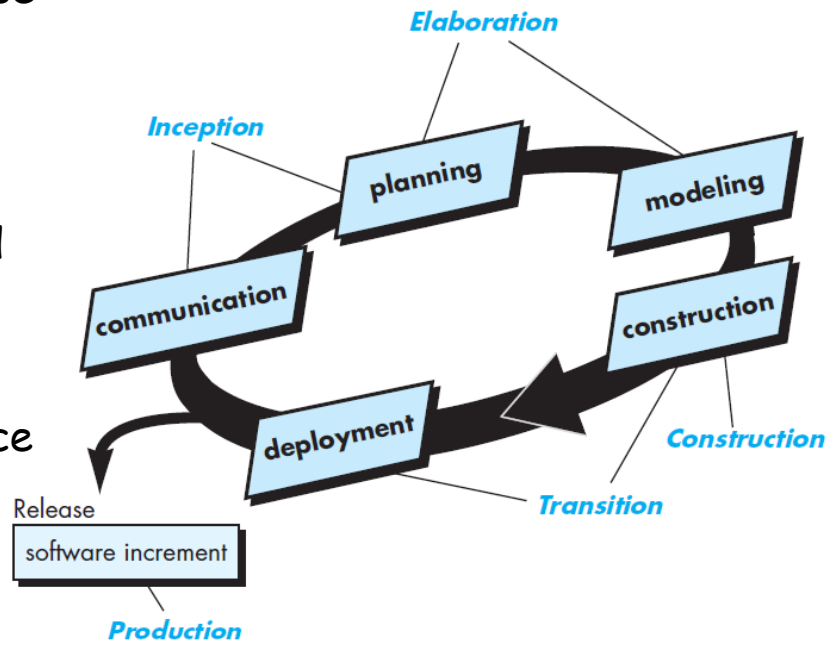
- encompasses the communication and modeling activities of the generic process model.
- Elaboration refines and expands the preliminary use cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software.
  - The use case model, the analysis model, the design model, the implementation model, and the deployment model.
  - In some cases, elaboration creates an "executable architectural baseline" that represents a "first cut" executable system.
  - The architectural baseline demonstrates the viability of the architecture but does not provide all features and functions required to use the system.
  - In addition, the plan is carefully reviewed at the culmination of the elaboration phase to ensure that scope, risks, and delivery dates remain reasonable. Modifications to the plan are often made at this time.



Phases of the Unified Process

Construction phase

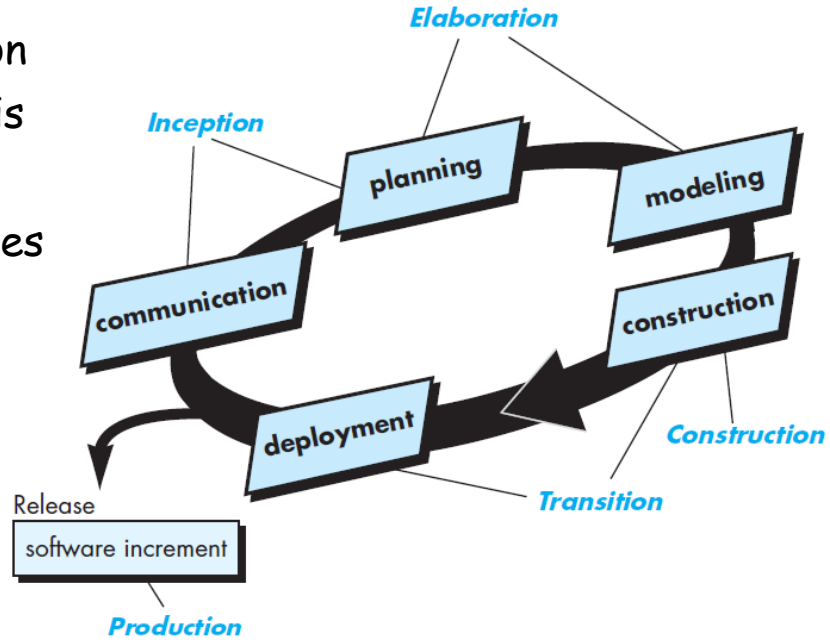
- is identical to the construction activity defined for the generic software process.
- Using the architectural model as input, the construction phase develops or acquires the software components that will make each use case operational for end users.
- Analysis and design models that were started during 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. As components are being implemented, unit tests are designed and executed for each.
- In addition, integration activities (component assembly and integration testing) are conducted. Use cases are used to derive a suite of acceptance tests that are executed prior to the initiation of the next UP phase.



Phases of the Unified Process

Transition phase

- encompasses the latter stages of the generic construction activity and the fi rst 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.
- In addition, the software team creates the necessary support information (e.g., user manuals, troubleshooting guides, installation procedures) that is required for the release.
- At the conclusion of the transition phase, the software increment becomes a usable software release.



Phases of the Unified Process

Production phase

- coincides with the deployment activity of the generic process.
- Software is given to end users for beta testing, and user feedback reports both defects and necessary changes.
- 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.