

Introduction to Software Engineering

Social Learning Process

- Software is embodied knowledge that is initially dispersed, tacit and incomplete.
- In order to convert knowledge into software, dialogues are needed between users and designers, between designers and tools to bring knowledge into software.
- Software development is essentially an iterative social learning process, and the outcome is “software capital”.

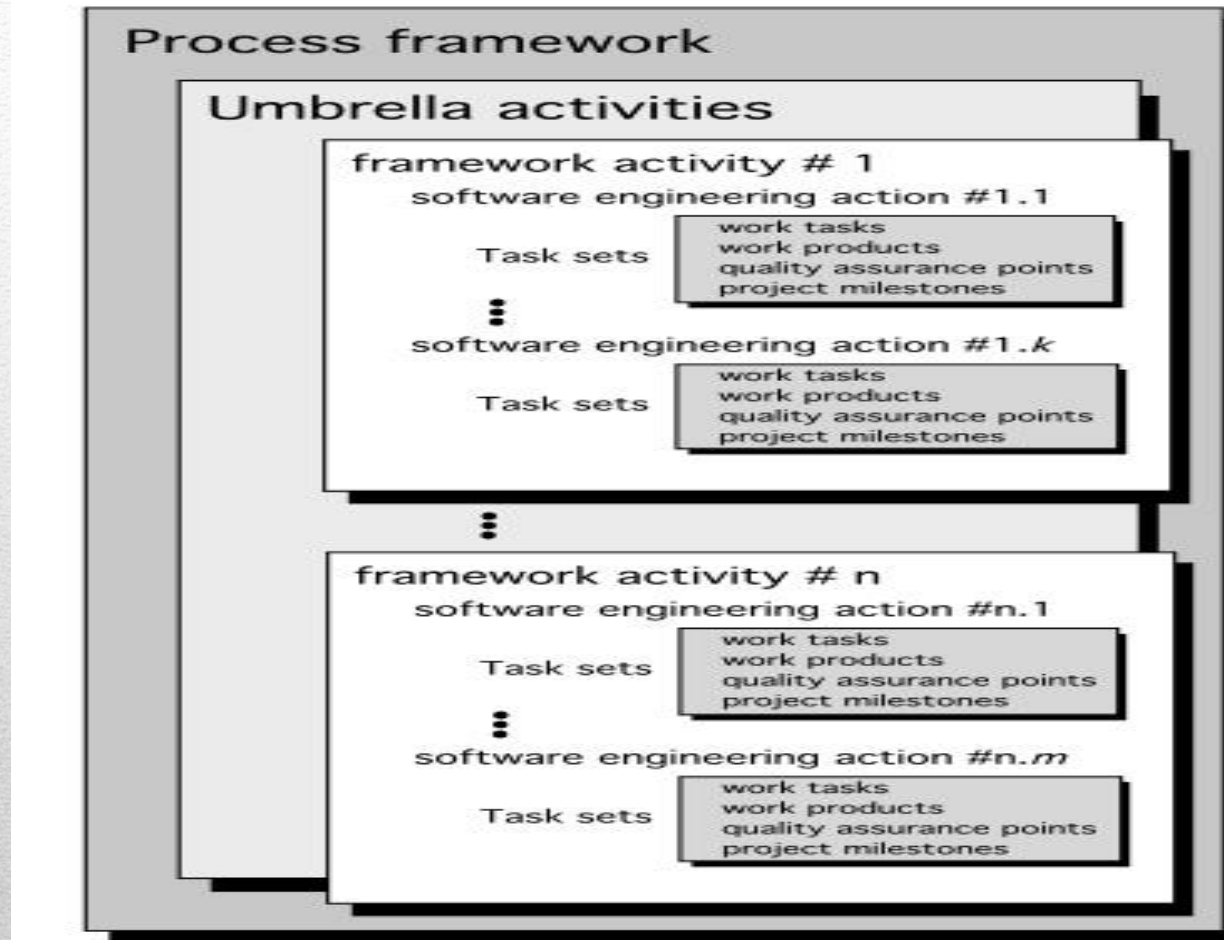
- **What:** Go through a series of predictable steps--- a **road map** that helps you create a timely, high-quality results.
- **Who:** Software engineers and their managers, clients also. People adapt the process to their needs and follow it.
- **Why:** Provides stability, control, and organization to an activity that can if left uncontrolled, become quite chaotic. However, modern software engineering approaches must be agile and demand **ONLY** those activities, controls and work products that are appropriate.
- **What Work products:** Programs, documents, and data
- **What are the steps:** The process you adopt depends on the software that you are building. One process might be good for aircraft avionic system, while an entirely different process would be used for website creation.
- **How to ensure right:** A number of software process assessment mechanisms that enable us to determine the maturity of the software process. However, the quality, timeliness and long-term viability of the software are the best indicators of the efficacy of the process you use.

What / who / why is Process Models?

Definition of Software Process

- A **framework** for the activities, actions, and tasks that are required to build high-quality software.
- SP defines the approach that is taken as software is engineered.
- Is not equal to software engineering, which also encompasses **technologies** that populate the process—technical methods and automated tools.

Software process



A Generic Process Model

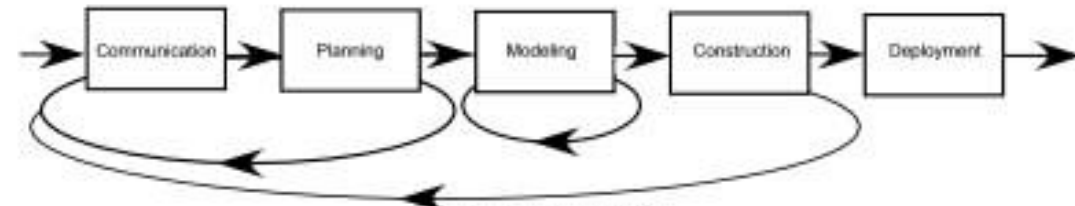
- As we discussed before, a generic process framework for software engineering defines five framework activities- communication, planning, modeling, construction, and deployment.
- In addition, a set of umbrella activities- project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others are applied throughout the process.
- Next question is: how the framework activities and the actions and tasks that occur within each activity are organized with respect to sequence and time? See the **process flow** for answer.

A Generic Process Model

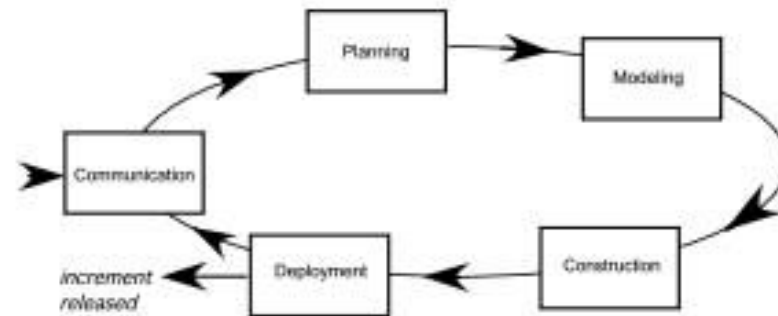
Process Flow



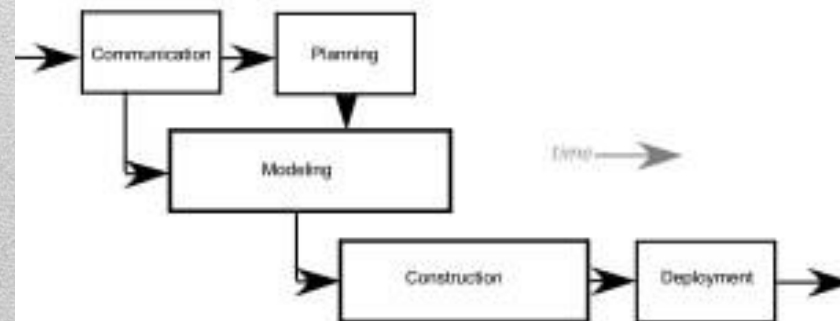
(a) linear process flow



(b) iterative process flow



(c) evolutionary process flow



(d) parallel process flow

- Linear process flow executes each of the five activities in sequence.
- 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 leads to a more complete version of the software.
- A parallel process flow executes one or more activities in parallel with other activities (modeling for one aspect of the software in parallel with construction of another aspect of the software.

Process Flow

- Before you can proceed with the process model, a key question: what **actions** are appropriate for a framework activity given the nature of the problem, the characteristics of the people and the stakeholders?
- A task set defines the actual work to be done to accomplish the objectives of a software engineering action.
 - A list of the task to be accomplished
 - A list of the work products to be produced
 - A list of the quality assurance filters to be applied

Identifying a Task Set

- For example, a small software project requested by one person with simple requirements, the communication activity might encompass little more than a phone call with the stakeholder. Therefore, the only necessary action is phone conversation, the work tasks of this action are:
 - 1. Make contact with stakeholder via telephone.
 - 2. Discuss requirements and take notes.
 - 3. Organize notes into a brief written statement of requirements.
 - 4. E-mail to stakeholder for review and approval.

Identifying a Task Set

- The task sets for Requirements gathering action for a **simple** project may include:
 1. Make a list of stakeholders for the project.
 2. Invite all stakeholders to an informal meeting.
 3. Ask each stakeholder to make a list of features and functions required.
 4. Discuss requirements and build a final list.
 5. Prioritize requirements.
 6. Note areas of uncertainty.

Example of a Task Set for Elicitation

■ The task sets for Requirements gathering action for a **big** project may include:

1. Make a list of stakeholders for the project.
2. Interview each stakeholders separately to determine overall wants and needs.
3. Build a preliminary list of functions and features based on stakeholder input.
4. Schedule a series of facilitated application specification meetings.
5. Conduct meetings.
6. Produce informal user scenarios as part of each meeting.
7. Refine user scenarios based on stakeholder feedback.
8. Build a revised list of stakeholder requirements.
9. Use quality function deployment techniques to prioritize requirements.
10. Package requirements so that they can be delivered incrementally.
11. Note constraints and restrictions that will be placed on the system.
12. Discuss methods for validating the system.

- Both do the same work with different depth and formality. Choose the task sets that achieve the goal and still maintain quality and agility.

Example of a Task Set for Elicitation

- 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
 - suggests one or more proven solutions to the problem.
- Stated in more general terms, a process pattern provides you with a *template* [Amb98]—a consistent method for describing problem solutions within the context of the software process. (defined at different levels of abstraction)
 1. Problems and solutions associated with a complete process model (e.g. prototyping).
 2. Problems and solutions associated with a framework activity (e.g. planning) or
 3. an action with a framework activity (e.g. project estimating).

Process Patterns

- *Stage patterns*—defines a problem associated with a framework activity for the process. It includes multiple task patterns as well. For example, EstablishingCommunication would incorporate the task pattern RequirementsGathering and others.
- *Task patterns*—defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice
- *Phase patterns*—define the sequence of framework activities that occur with the process, even when the overall flow of activities is iterative in nature. Example includes SprialModel or Prototyping.

Process Pattern Types

An Example of 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 that can be assessed iteratively by stakeholders in an effort to identify or solidify software requirements.
- **Type.** Phase pattern
- **Initial context.** Conditions must be met (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. stakeholders are unsure of what they want.
- **Solution.** A description of the prototyping process would be presented here.
- **Resulting context.** A software prototype that identifies basic requirements. (modes of interaction, computational features, processing functions) is approved by stakeholders. Following this, 1. This prototype may evolve through a series of increments to become the production software or 2. the prototype may be discarded.
- **Related patterns.** CustomerCommunication, IterativeDesign, IterativeDevelopment, CustomerAssessment, RequirementExtraction.

Process Assessment and Improvement

SP cannot guarantee that software will be delivered on time, meet the needs, or has the desired technical characteristics. However, the process can be assessed to ensure that it meets a set of basic process criteria that have been shown to be essential for a successful software engineering.

- **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 [Dun01]
- **SPICE — The SPICE (ISO/IEC15504)** standard 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. [ISO08]
- **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. [Ant06]

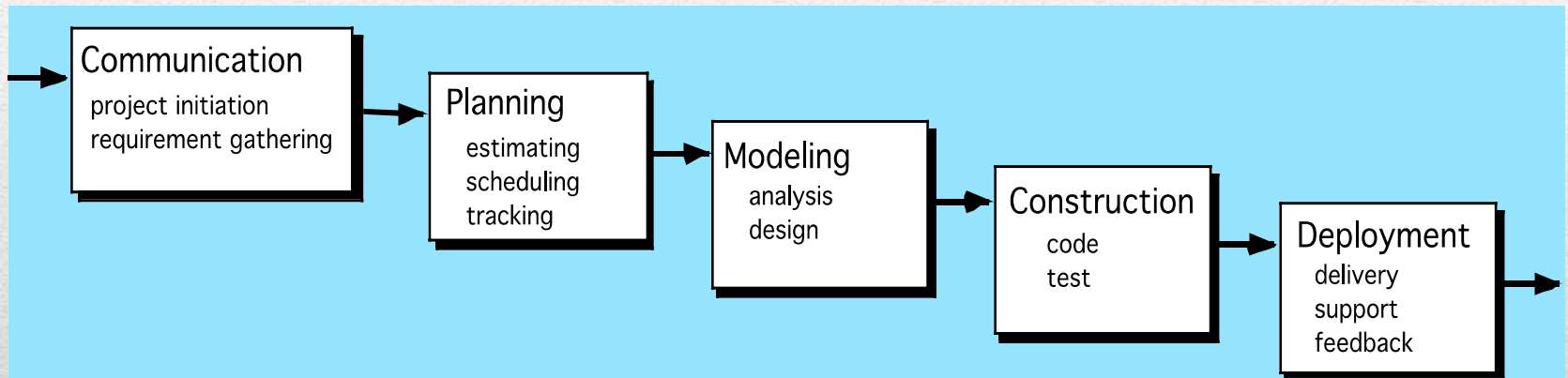
Prescriptive Models

- Originally proposed to bring order to chaos.
- Prescriptive process models advocate an orderly approach to software engineering. However, will some extent of chaos (less rigid) be beneficial to bring some creativity?

That leads to a few questions ...

- If prescriptive process models strive for structure and order (prescribe a set of process elements and process flow), *are they inappropriate for a software world that thrives on change?*
- Yet, if we reject traditional process models (and the order they imply) and replace them with something less structured, *do we make it impossible to achieve coordination and coherence in software work?*

The Waterfall Model

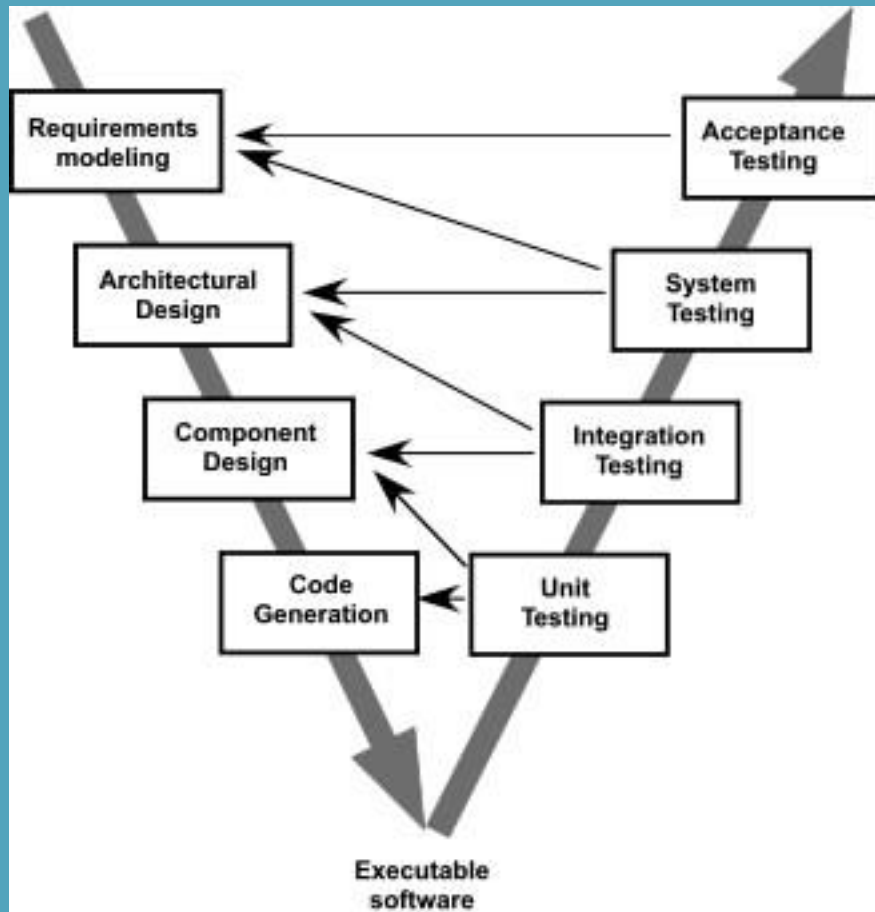


It is the oldest paradigm for SE. When requirements are well defined and reasonably stable, it leads to a linear fashion.

(problems: 1. rarely linear, iteration needed. 2. hard to state all requirements explicitly. Blocking state. 3. code will not be released until very late.)

The classic life cycle suggests a systematic, sequential approach to software development.

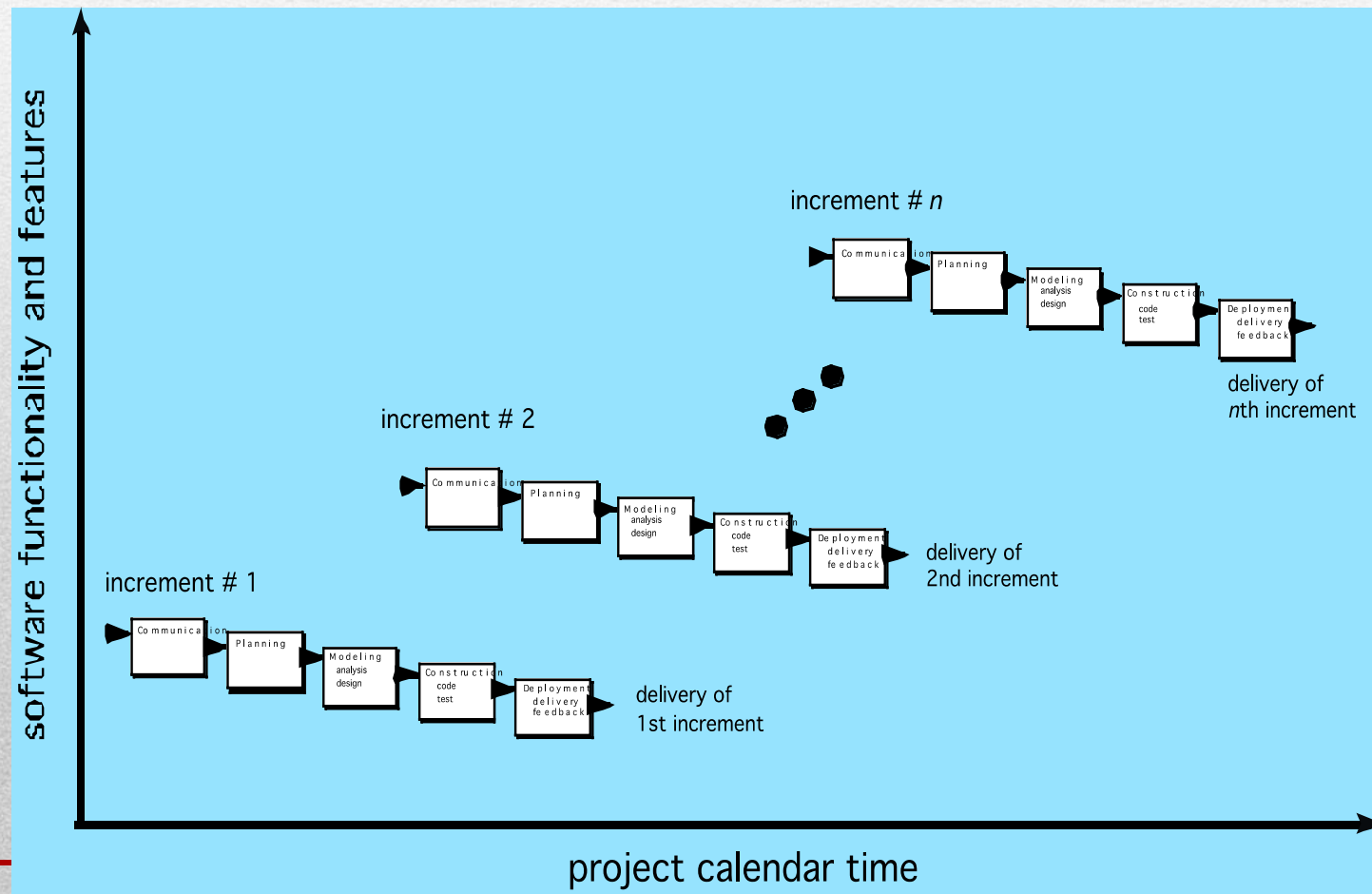
The V-Model



A variation of waterfall model depicts the relationship of quality assurance actions to the actions associated with communication, modeling and early code construction activates.

Team first moves down the left side of the V to refine the problem requirements. Once code is generated, the team moves up the right side of the V, performing a series of tests that validate each of the models created as the team moved down the left side.

The Incremental Model



The Incremental Model

- When initial requirements are reasonably well defined, but the overall scope of the development effort precludes a purely linear process. A compelling need to expand a limited set of new functions to a later system release.
- It combines elements of linear and parallel process flows. Each linear sequence produces deliverable increments of the software.
- The first increment is often a core product with many supplementary features. Users use it and evaluate it with more modifications to better meet the needs.

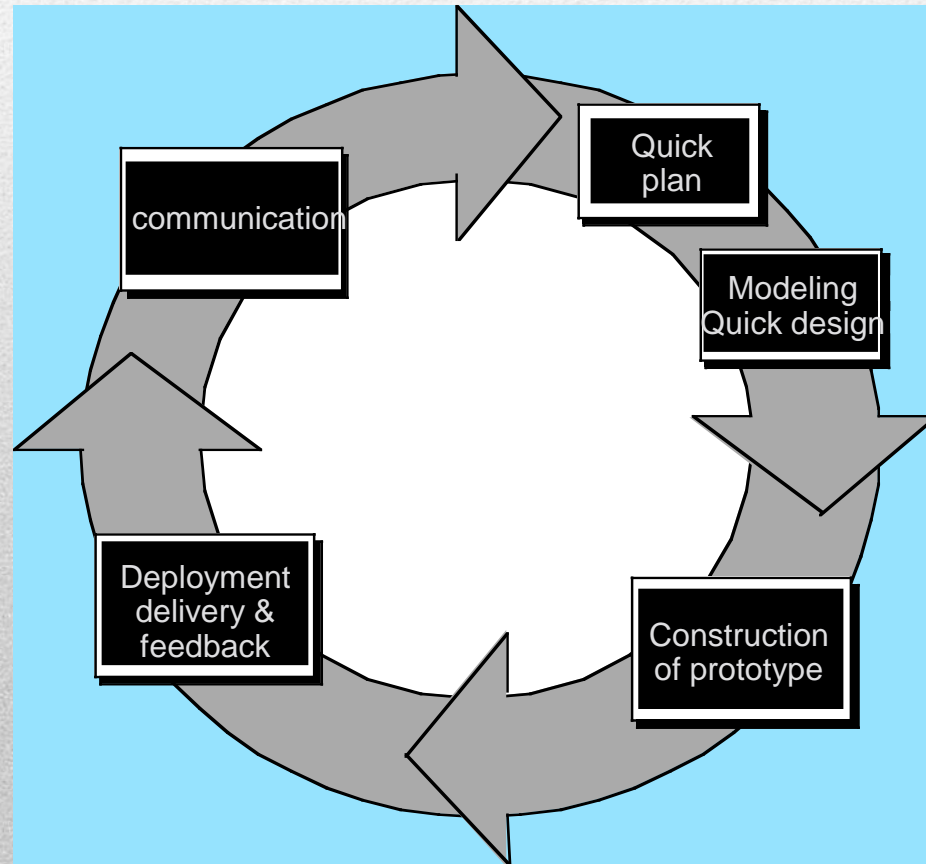
Evolutionary Models

- Software system evolves over time as requirements often change as development proceeds. Thus, a straight line to a complete end product is not possible. However, a limited version must be delivered to meet competitive pressure.
 - Usually a set of core product or system requirements is well understood, but the details and extension have yet to be defined.
 - You need a process model that has been explicitly designed to accommodate a product that evolved over time.
 - It is iterative that enables you to develop increasingly more complete version of the software.
 - Two types are introduced, namely **Prototyping and Spiral models.**
-

Evolutionary Models: Prototyping

- When to use: Customer defines a set of general objectives but does not identify detailed requirements for functions and features. Or Developer may be unsure of the efficiency of an algorithm, the form that human computer interaction should take.
- What step: Begins with communication by meeting with stakeholders to define the objective, identify whatever requirements are known, outline areas where further definition is mandatory. A quick plan for prototyping and modeling (quick design) occur. Quick design focuses on a representation of those aspects the software that will be visible to end users. (interface and output). Design leads to the construction of a prototype which will be deployed and evaluated. Stakeholder' s comments will be used to refine requirements.
- Both stakeholders and software engineers like the prototyping paradigm. Users get a feel for the actual system, and developers get to build something immediately. However, engineers may make compromises in order to get a prototype working quickly. The less-than-ideal choice may be adopted forever after you get used to it.

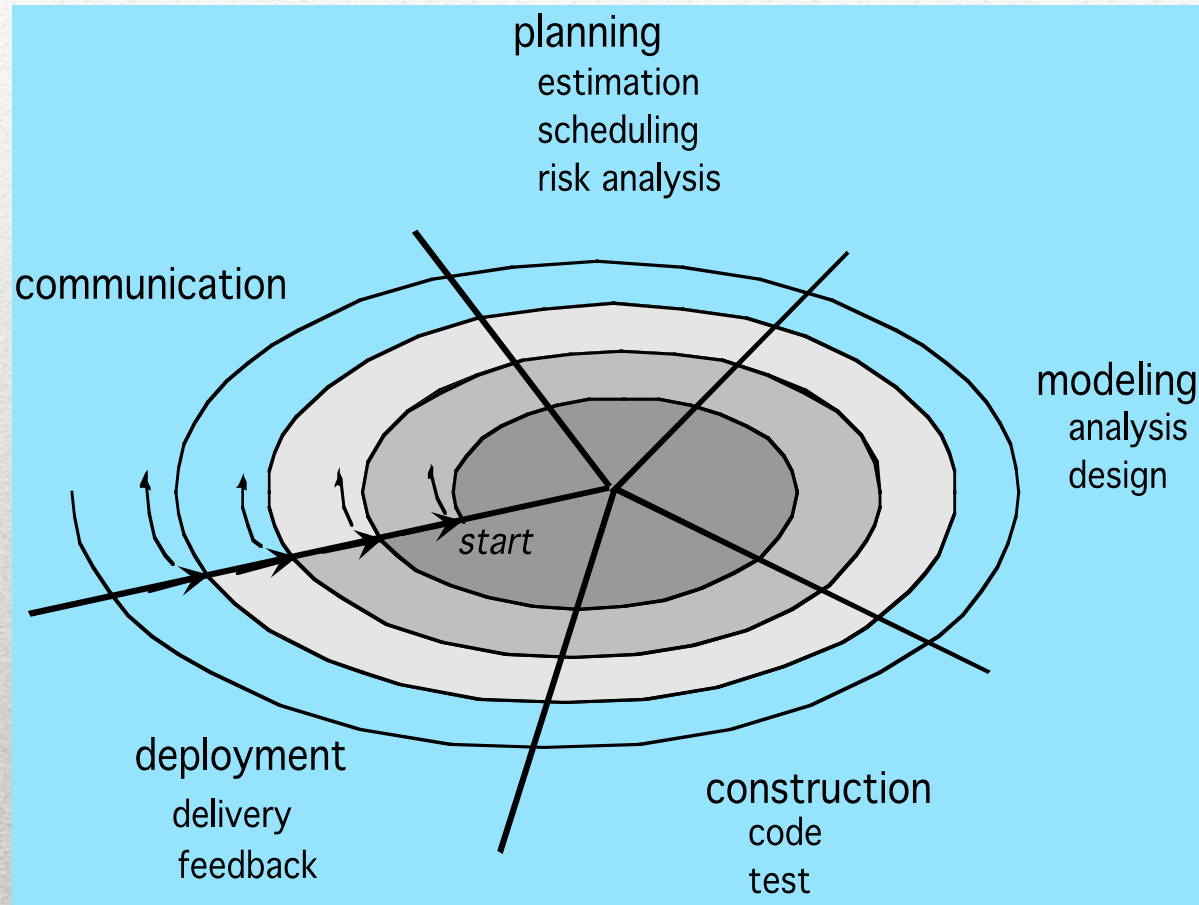
Evolutionary Models: Prototyping



Evolutionary Models: The Spiral

- It couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model and is a risk-driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems.
- Two main distinguishing features: one is **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.
- A series of evolutionary releases are delivered. During the early iterations, the release might be a model or prototype. During later iterations, increasingly more complete version of the engineered system are produced.
- The first circuit in the clockwise direction might result in the 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 results in adjustments to the project plan. Cost and schedule are adjusted based on feedback. Also, the number of iterations will be adjusted by project manager.
- Good to develop large-scale system as software evolves as the process progresses and risk should be understood and properly reacted to. Prototyping is used to reduce risk.
- However, it may be difficult to convince customers that it is controllable as it demands considerable risk assessment expertise.

Evolutionary Models: The Spiral



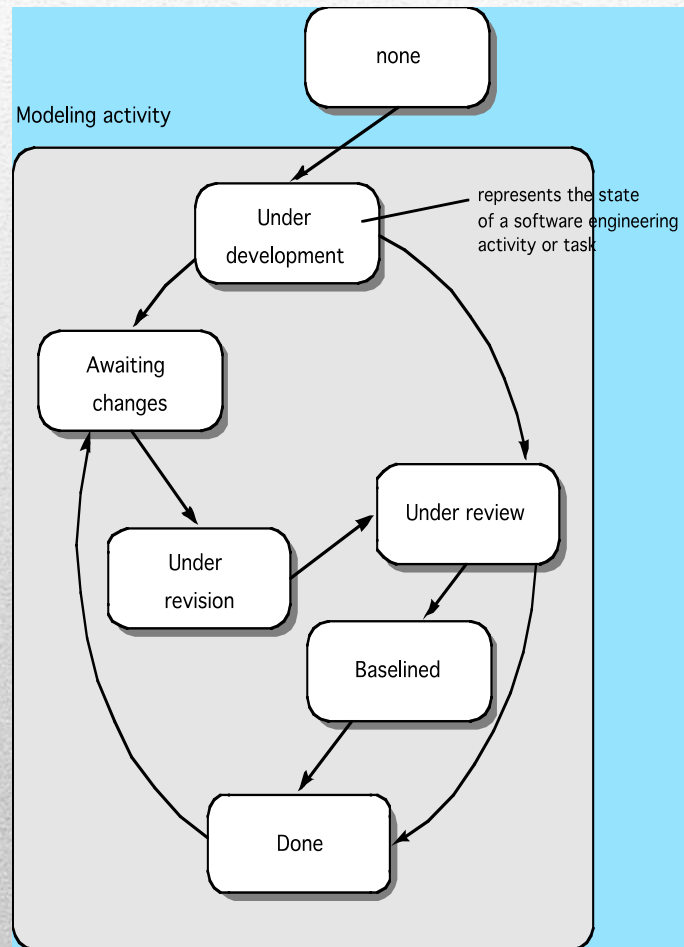
Three Concerns on Evolutionary Processes

- First concern is that prototyping poses a problem to project planning because of the uncertain number of cycles required to construct the product.
- Second, it does not establish the maximum speed of the evolution. If the evolution occur too fast, without a period of relaxation, it is certain that the process will fall into chaos. On the other hand if the speed is too slow then productivity could be affected.
- Third, software processes should be focused on flexibility and extensibility rather than on high quality. We should prioritize the speed of the development over zero defects. Extending the development in order to reach high quality could result in a late delivery of the product when the opportunity niche has disappeared.

Concurrent Model

- Allow 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 actions: prototyping, analysis and design.
 - The Figure shows modeling may be in any one of the states at any given time. For example, communication activity has completed its first iteration and in the awaiting changes state. The modeling activity was in inactive state, now makes a transition into the under development state. If customer indicates changes in requirements, the modeling activity moves from the under development state into the awaiting changes state.
 - 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 trigger transitions among the states.
-

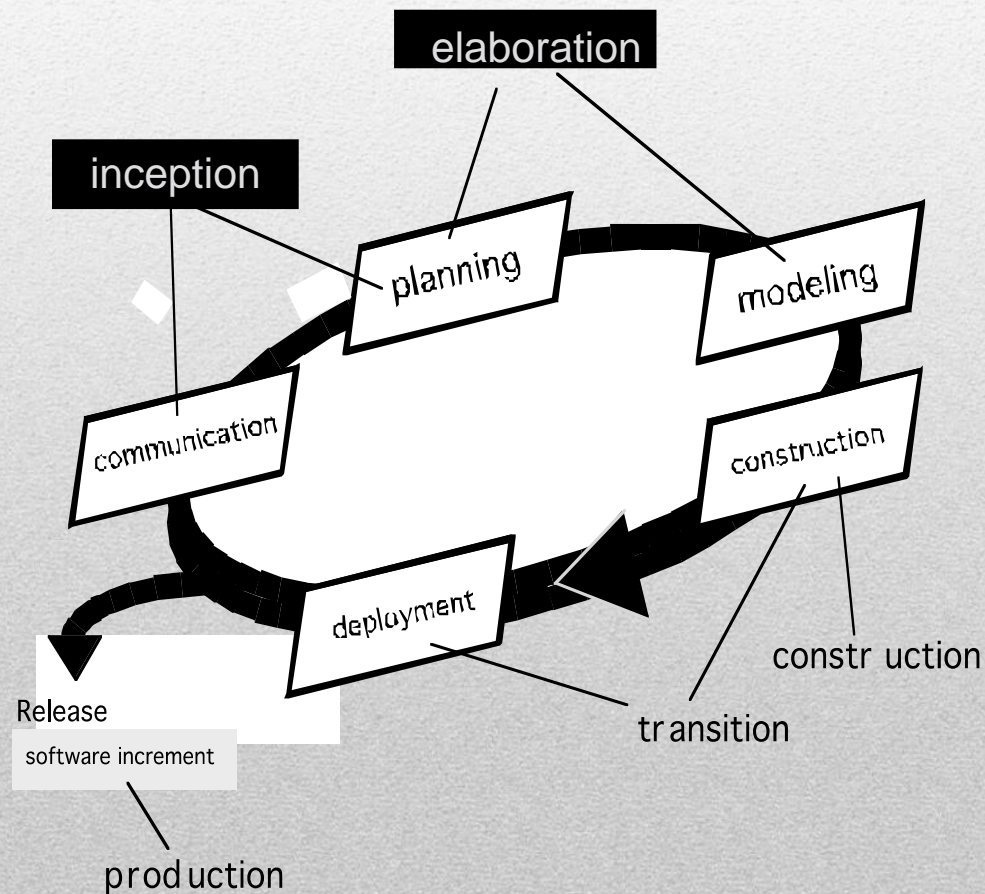
Concurrent Model



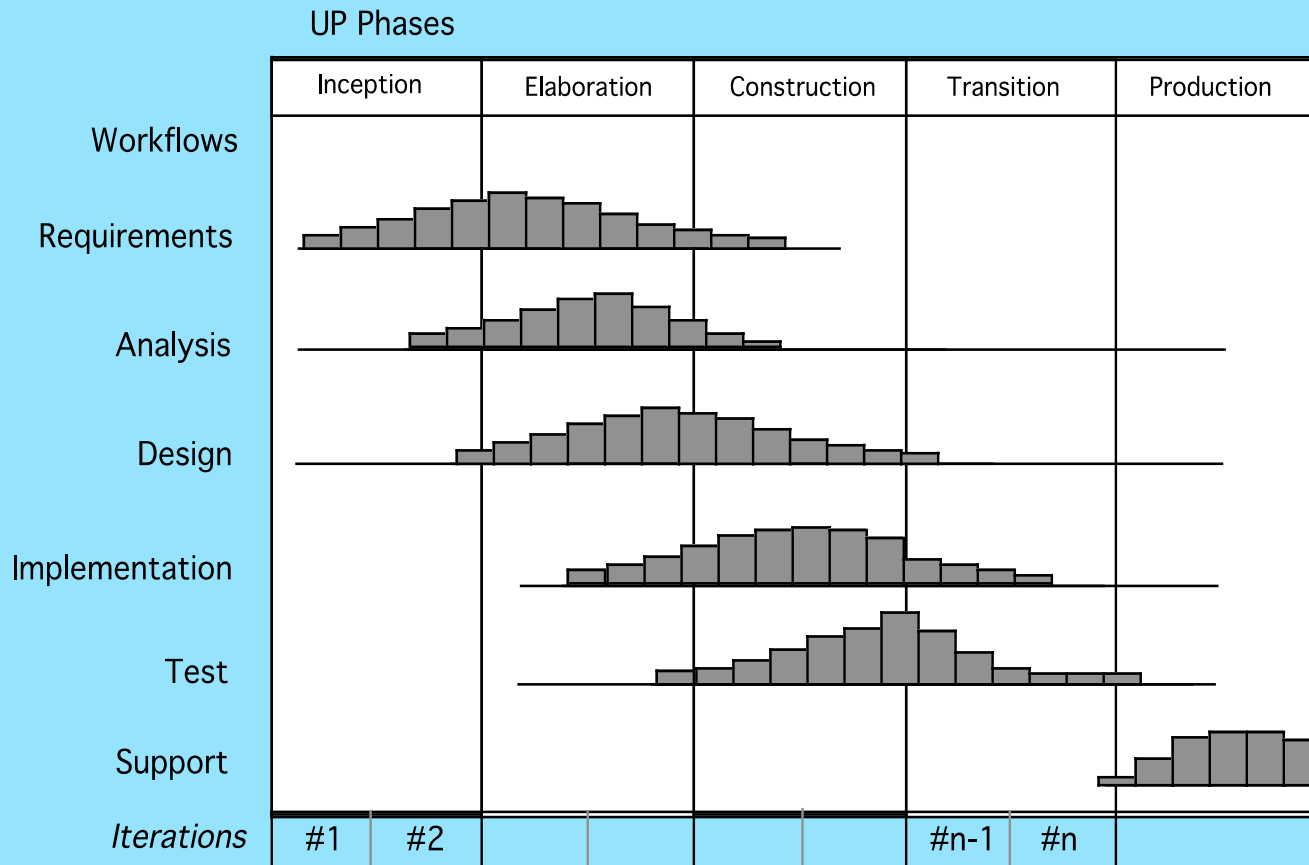
Still Other Process Models

- **Component based development**—the process to apply when reuse is a development objective (like spiral model)
- **Formal methods**—emphasizes the mathematical specification of requirements (easy to discover and eliminate ambiguity, incompleteness and inconsistency)
- **Aspect Oriented software development (AOSD)**—provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*
- **Unified Process**—a “use-case driven, architecture-centric, iterative and incremental” software process closely aligned with the Unified Modeling Language (UML) to model and develop object-oriented system iteratively and incrementally.

The Unified Process (UP)



UP Phases



UP Work Products

Inception phase

Vision document
Initial use-case model
Initial project glossary
Initial business case
Initial risk assessment.
Project plan,
phases and iterations.
Business model,
if necessary.
One or more prototypes

Elaboration phase

Use-case model
Supplementary requirements
including non-functional
Analysis model
Software architecture
Description.
Executable architectural
prototype.
Preliminary design model
Revised risk list
Project plan including
iteration plan
adapted workflows
milestones
technical work products
Preliminary user manual

Construction phase

Design model
Software components
Integrated software
increment
Test plan and procedure
Test cases
Support documentation
user manuals
installation manuals
description of current
increment

Transition phase

Delivered software increment
Beta test reports
General user feedback

Personal Software Process (PSP)

- **Planning.** This activity isolates requirements and develops both size and resource estimates. In addition, a defect estimate (the number of defects projected for the work) is made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.
- **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.
- **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 (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 (IPT) of three to about 20 engineers.
- Show managers how to coach and motivate their teams and how to help them sustain peak performance.
- Accelerate software process improvement by making CMM Level 5 behavior normal and expected.
 - The Capability Maturity Model (CMM), a measure of the effectiveness of a software process, is discussed in Chapter 30.
- Provide improvement guidance to high-maturity organizations.
- Facilitate university teaching of industrial-grade team skills.