CS435: Introduction to Software Engineering

Dr. M. Zhu

Process Models

Software Engineering: A Practitioner's Approach, 7/e
by Roger S. Pressman

Software Engineering 9/e
By Ian Sommerville

Chapter 2

These slides are designed and adapted from slides provided by *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009) by Roger Pressman and *Software Engineering 9e* Addison Wesley 2011 by Ian Sommerville

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.

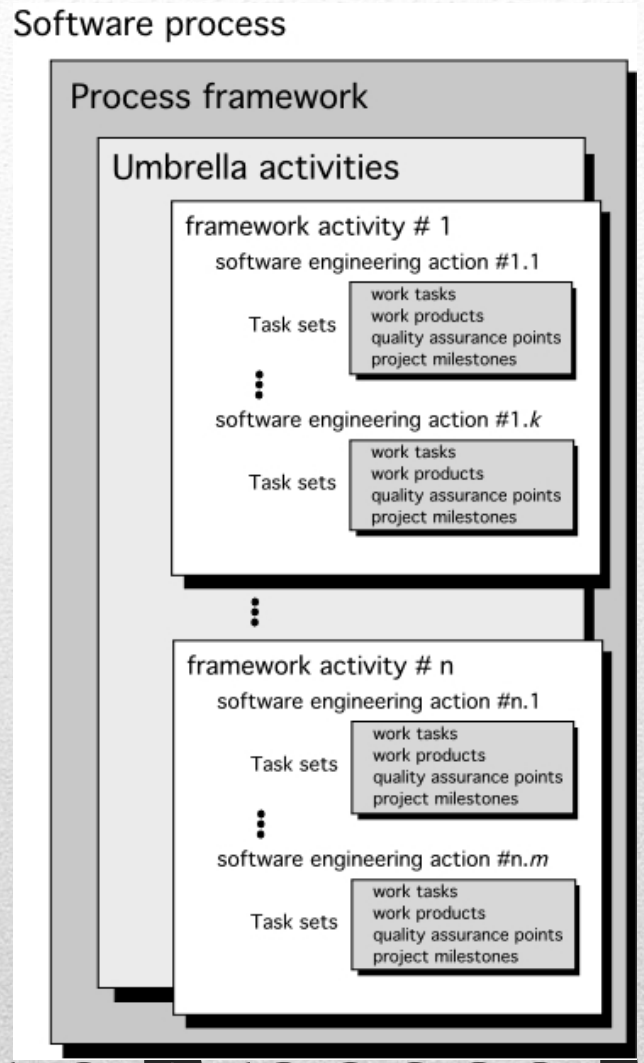
- Software development is essentially an iterative social learning process, and the outcome is “software capital”.

What / who / why is Process Models?

- **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.

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.



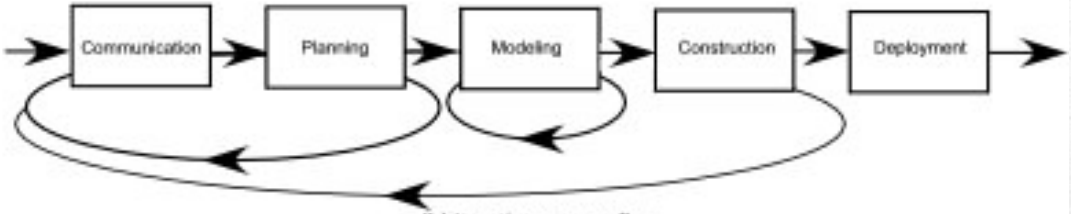
A Generic Process Model

A Generic Process Model

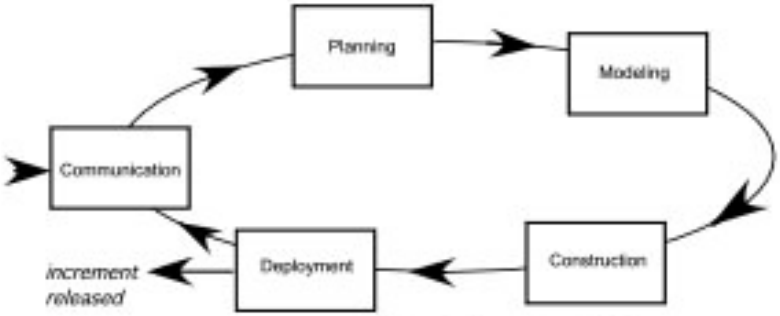
- As we discussed before, a generic process framework for software engineering defines five framework activities-communication, planning, modeling, construction, and deployment.
- In addition, a set of umbrella activities- project tracking and control, risk management, quality assurance, configuration 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.

Process Flow

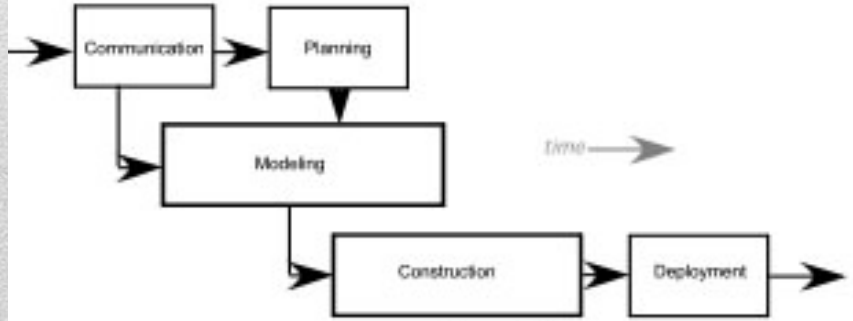
(a) linear process flow



(b) iterative process flow



(c) evolutionary process flow



(d) parallel process flow

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.

- 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 all 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

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.

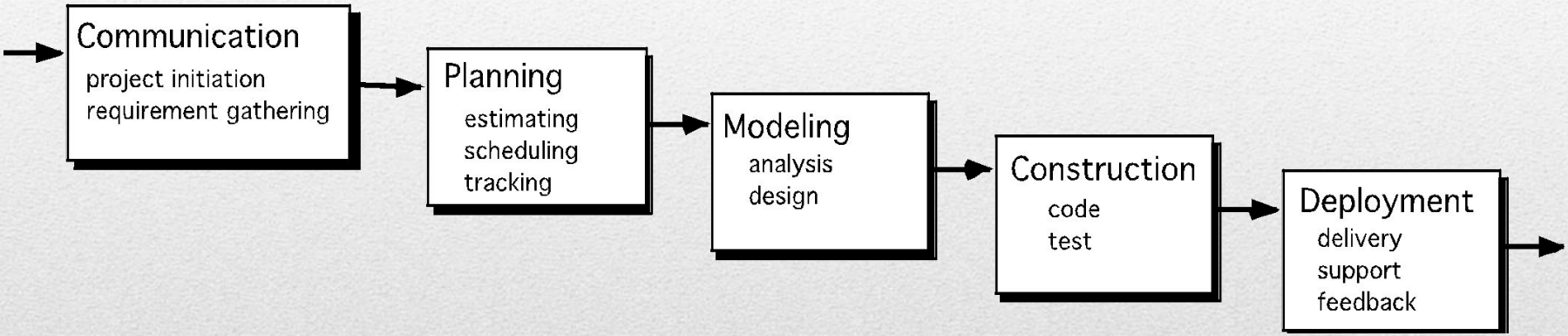
Prescriptive Models

- 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.)

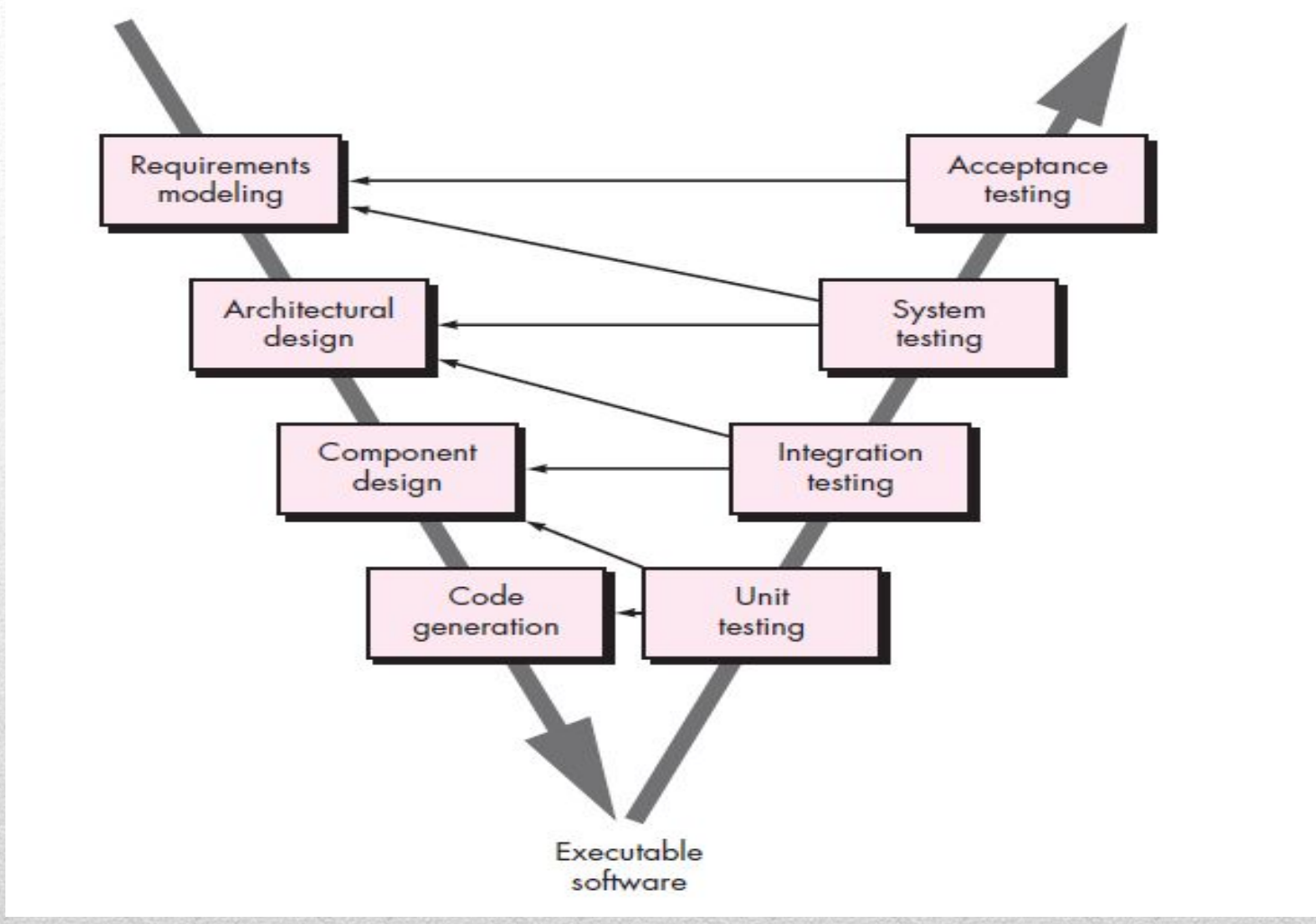
The classic life cycle suggests a systematic, sequential approach to software development.

Advantages of Waterfall Model

- This model is very simple and is easy to understand.
- Phases in this model are processed one at a time.
- Each stage in the model is clearly defined.
- This model has very clear and well understood milestones.
- Process, actions and results are very well documented.
- Reinforces good habits: define-before-design, design-before-code.
- This model works well for smaller projects and projects where requirements are well
  understood.

Disadvantages of Waterfall Model

- **No feedback path:** In classical waterfall model evolution of a software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phases. Therefore, it does not incorporate any mechanism for error correction.
- **Difficult to accommodate change requests:** This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but actually customers' requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.
- **No overlapping of phases:** This model recommends that new phase can start only after the completion of the previous phase. But in real projects, this can't be maintained. To increase the efficiency and reduce the cost, phases may overlap.

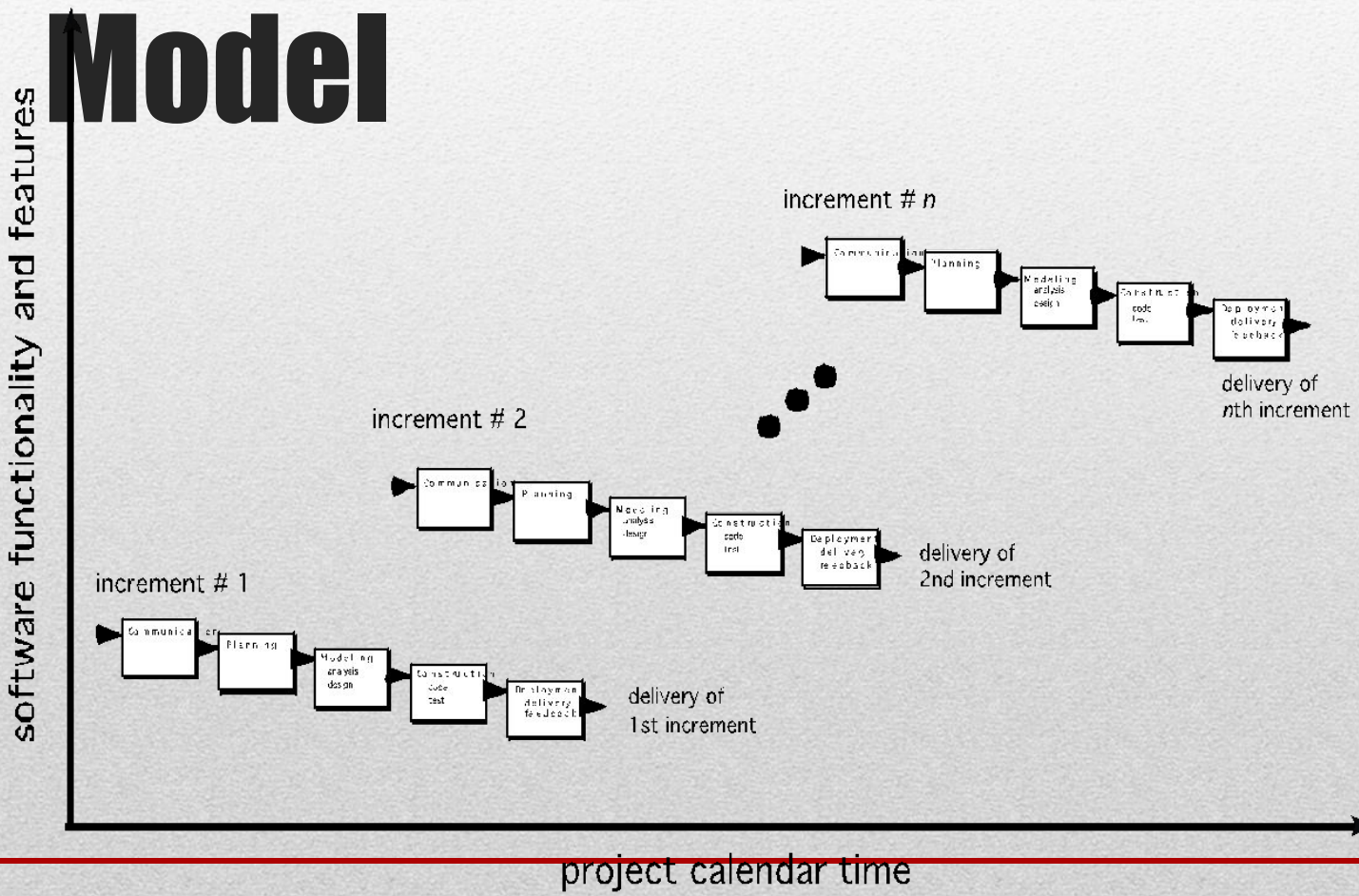

Requirements modeling
Architectural design
Component design
Code generation
Unit testing
Integration testing
System testing
Acceptance testing
Executable software

The V-Model

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

The V-Model

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.

- The software will be generated quickly during the software life cycle.
- It is flexible and less expensive to change requirements and scope.
- Throughout the development stages changes can be done.
- This model is less costly compared to others.
- A customer can respond to each building.

Advantages

- It requires a good planning designing.
- Problems might cause due to system architecture as such not all requirements collected up front for the entire software lifecycle.
- Each iteration phase is rigid and does not overlap each other.
- Rectifying a problem in one unit requires correction in all the units and consumes a lot of time.

Disadvantages

- Requirements of the system are clearly understood
- When demand for an early release of a product arises
- When software engineering team are not very well skilled or trained
- When high-risk features and goals are involved
- Such methodology is more in use for web application and product based companies

When to use Incremental models?

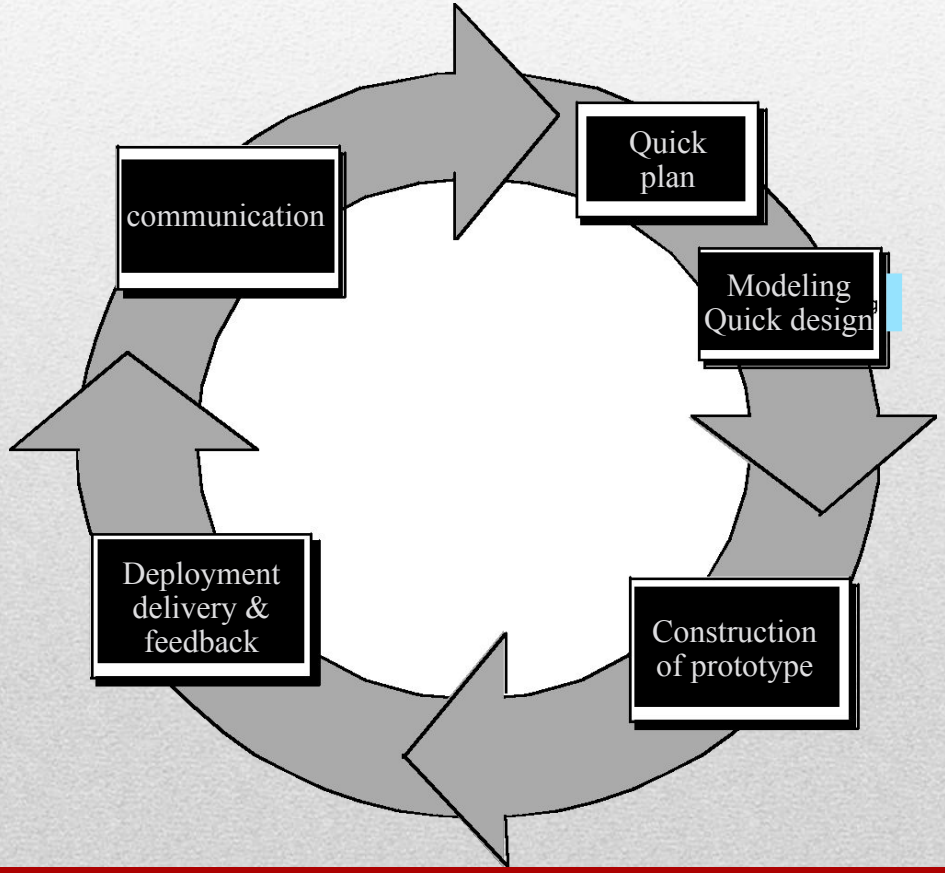
Evolutionary Models

- 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



- The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.
- New requirements can be easily accommodated as there is scope for refinement.
- Missing functionalities can be easily figured out.
- Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.
- The developed prototype can be reused by the developer for more complicated projects in the future.
- Flexibility in design.

Advantages

- Costly w.r.t time as well as money.
- There may be too much variation in requirements each time the prototype is evaluated by the customer.
- Poor Documentation due to continuously changing customer requirements.
- It is very difficult for developers to accommodate all the changes demanded by the customer.
- There is uncertainty in determining the number of iterations that would be required before the prototype is finally accepted by the customer.
- After seeing an early prototype, the customers sometimes demand the actual product to be delivered soon.

Disadvantages

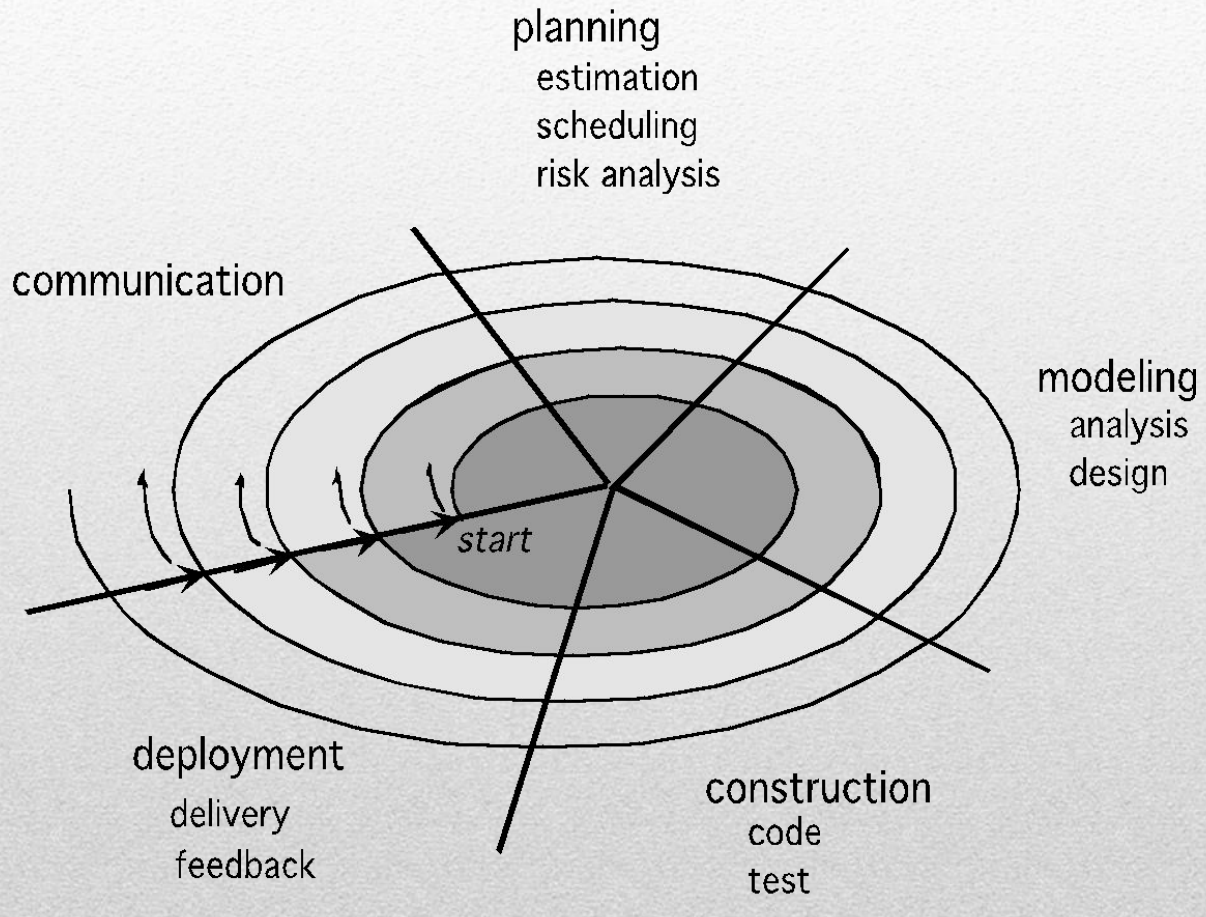
When to use Prototype models?

- The Prototyping Model should be used when the requirements of the product are not clearly understood or are unstable.
- It can also be used if requirements are changing quickly. This model can be successfully used for developing user interfaces, high technology software-intensive systems, and systems with complex algorithms and interfaces.
- It is also a very good choice to demonstrate the technical feasibility of the product.

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.

Evolutionary Models: The Spiral



- **Risk Handling:** Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
- **Good for large projects:** It is recommended to use the Spiral Model in large and complex projects.
- **Flexibility in Requirements:** Change requests in the Requirements at later phase can be incorporated accurately by using this model.
- **Customer Satisfaction:** Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

Advantages

- **Complex:** The Spiral Model is much more complex than other SDLC models.
- **Expensive:** Spiral Model is not suitable for small projects as it is expensive.
- **Too much dependable on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced expertise, it is going to be a failure to develop a project using this model.
- **Difficulty in time management:** As the number of phases is unknown at the start of the project, so time estimation is very difficult.

Disadvantages

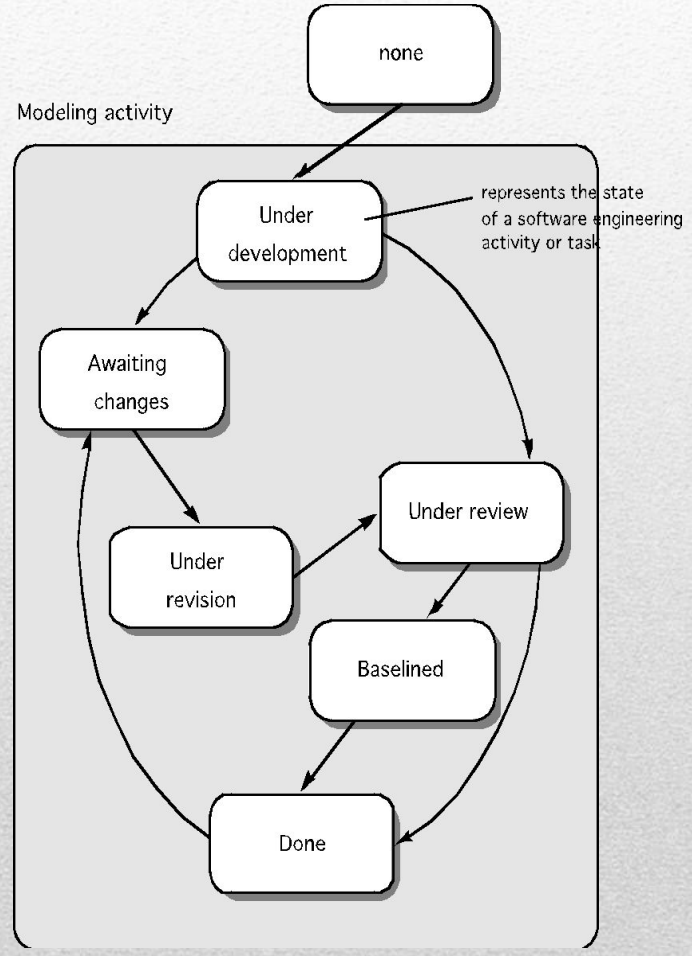
Three Concerns on Evolutionary Processes

- 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.

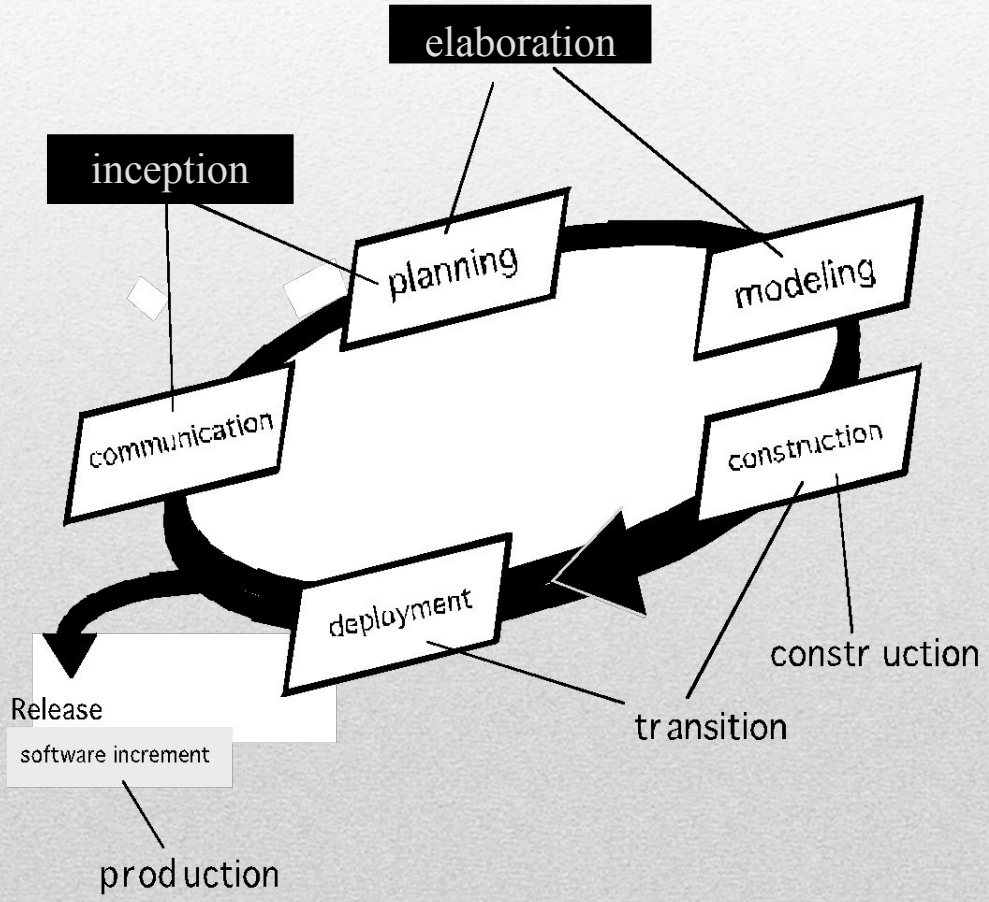
Concurrent Model

- 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.

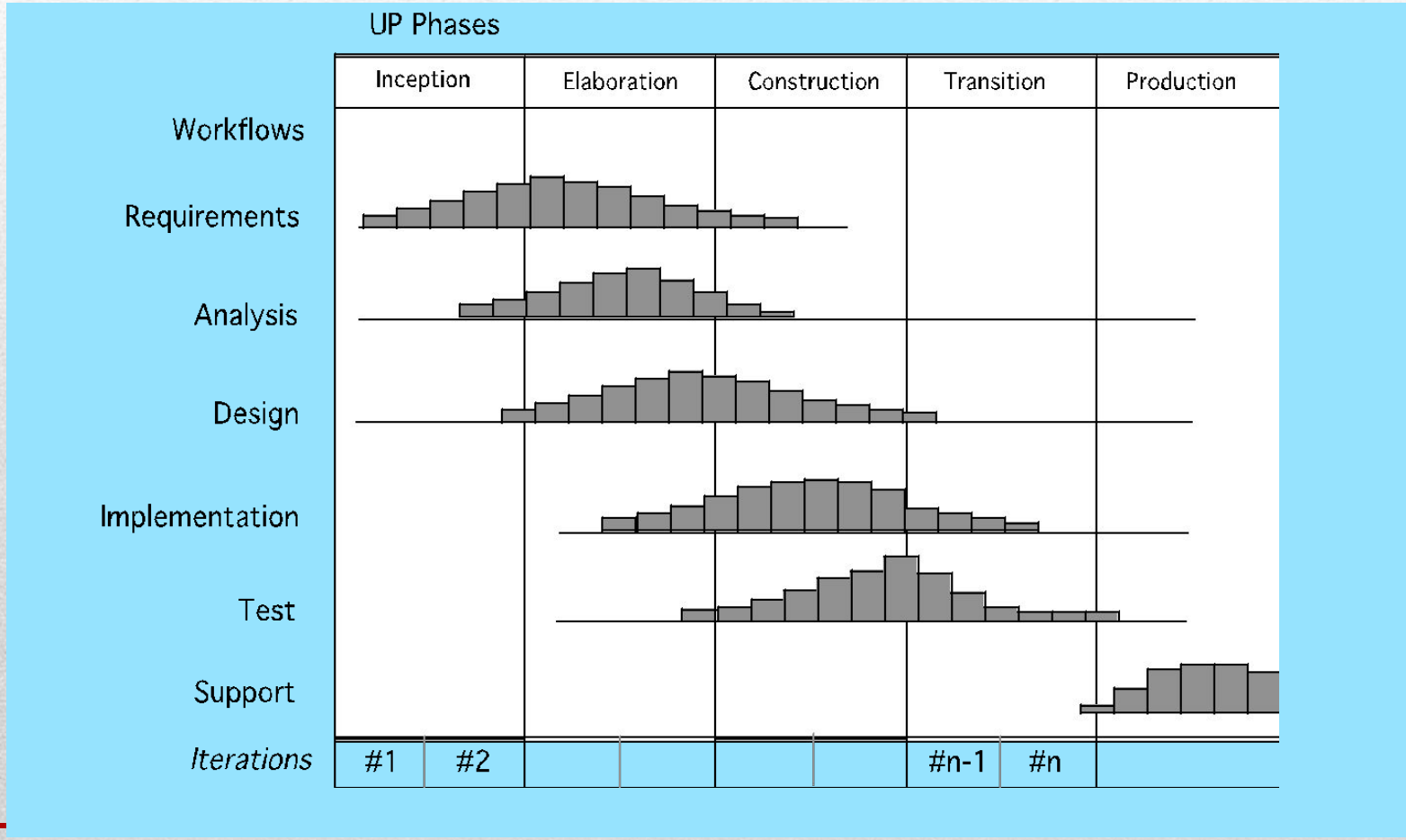
Concurrent Model



The Unified Process (UP)



UP Phases


UP Phases
Inception
Elaboration
Construction
Transition
Production
Workflows
Requirements
Analysis
Design
Implementation
Test
Support
Iterations
#1
#2
#n-1
#n

UP Work Products