Software Engineering and Information System

Lecture 08: Software Process

Md. Al-Hasan

Department of Computer Science & Engineering (CSE)

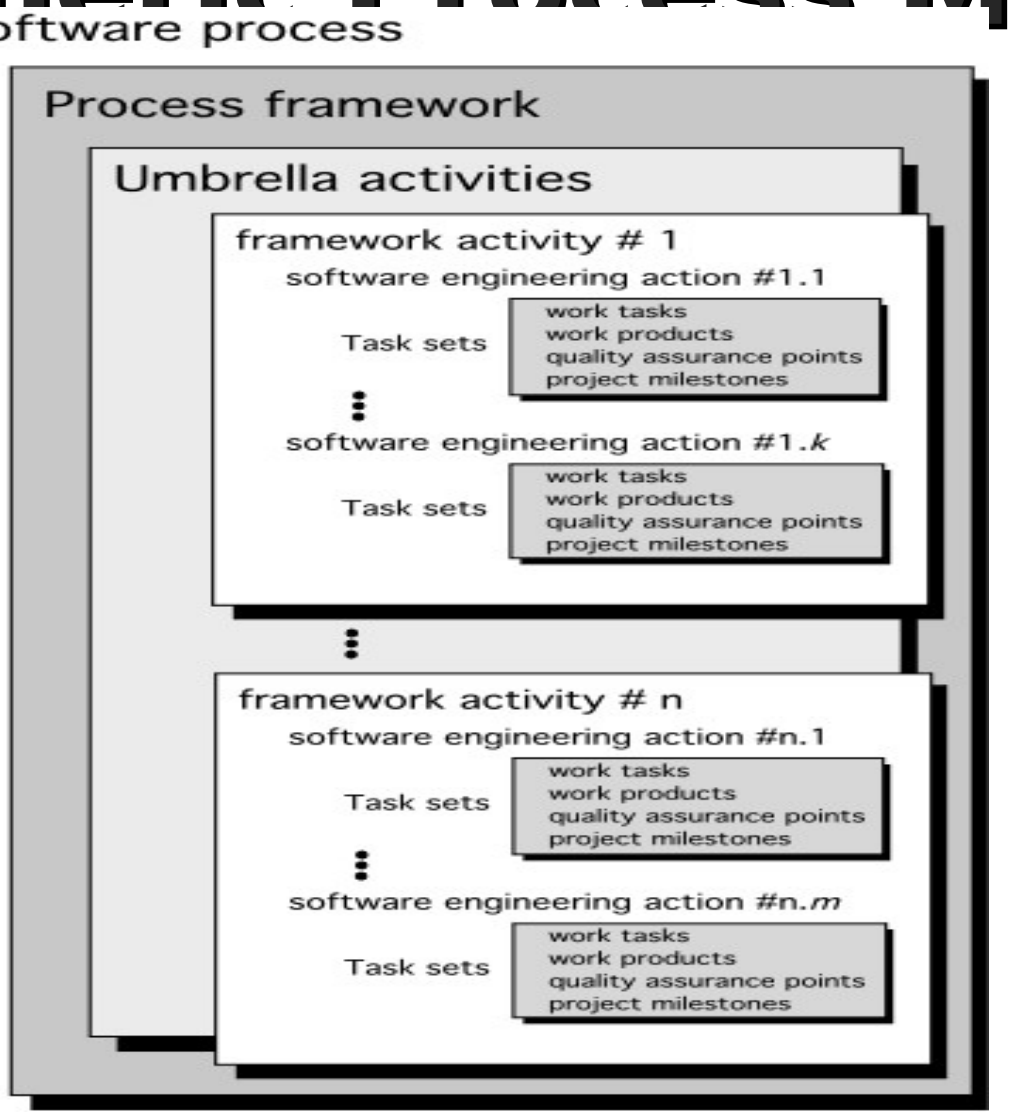
Bangladesh Army University of Science & Technology (BAUST)

Software Engineering- A PRACTITIONER ' S APPROACH *By* Roger S. Pressman

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

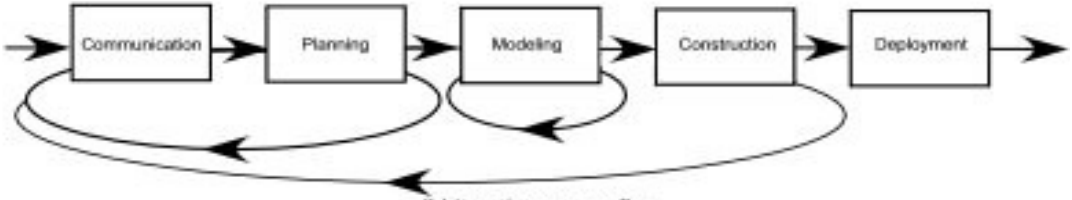


A Generic Process Model

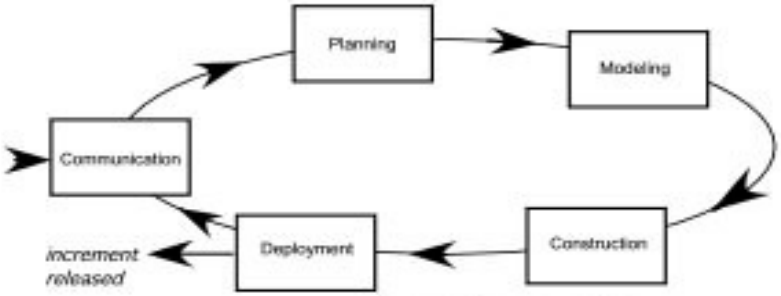
- A generic process framework for software engineering defines five framework activities-communication, planning, modeling, construction, and deployment.
- In addition, a set of umbrella activities-project tracking and control, risk management, quality assurance, configuration management, technical reviews,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

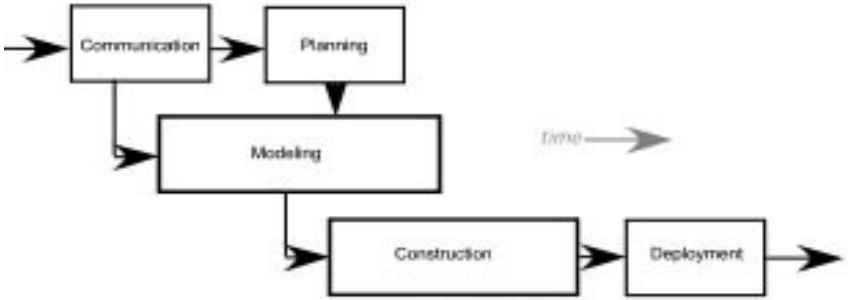
(a) linear process flow



(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).

Identifying a Task Set

- 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.

Example of a Task Set for Elicitation

- The task set 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.

Prescriptive Models

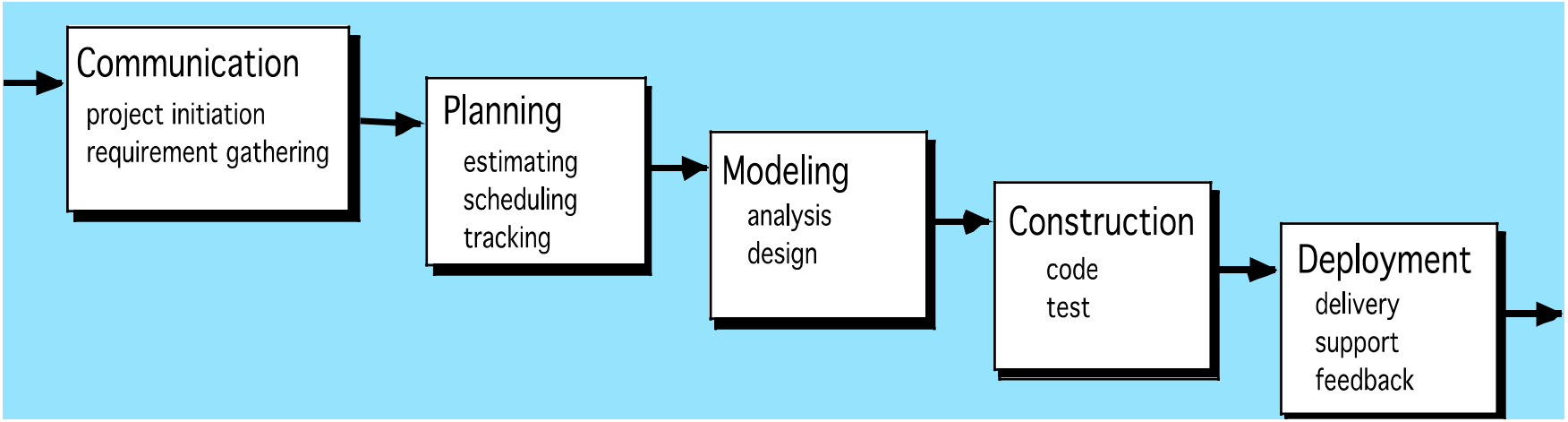
- Originally proposed to bring order to chaos.
- Prescriptive process models advocate an orderly approach to software engineering. However, will some extent of chaos (less rigid) be beneficial to bring some creativity?

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 structure, do we make it impossible to achieve coordination and coherence in software work?

https://www.tutorialride.com/software-engineering/prescriptive-process-models.htm

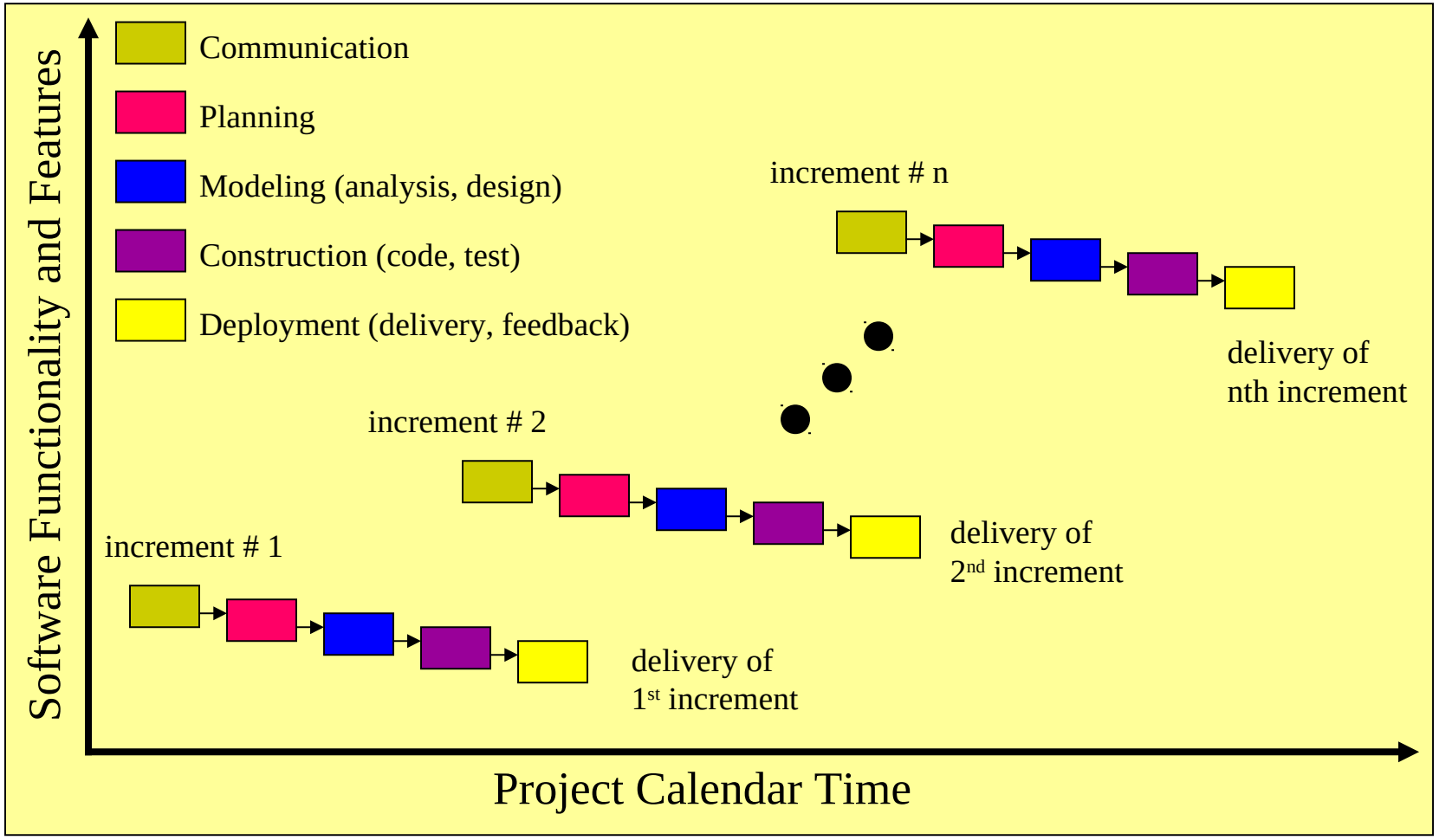
The Waterfall Model



- **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.**

http://tryqa.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/

The Incremental Model



The Incremental Model

- When initial requirements are reasonably well define, 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.

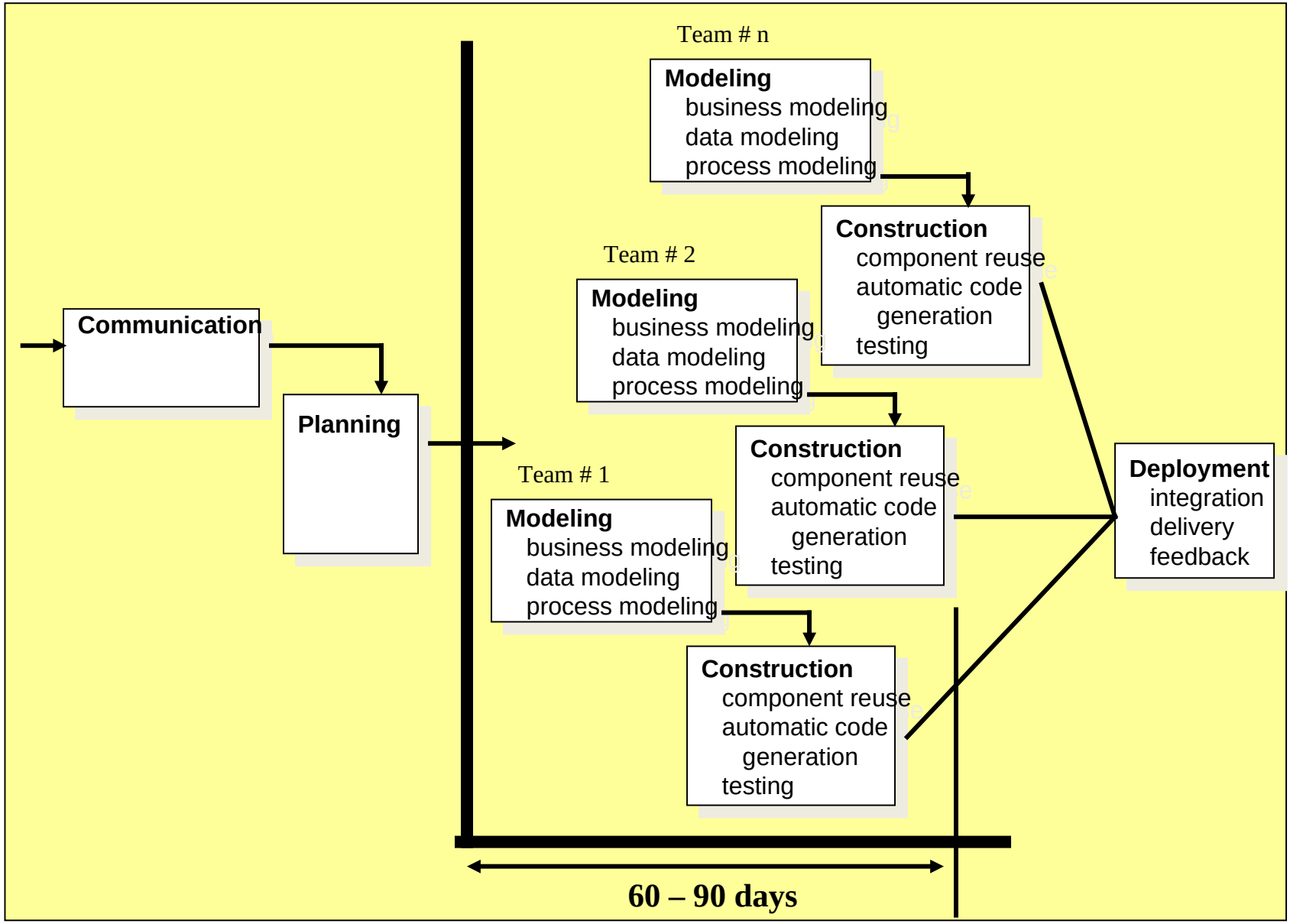
The Incremental Model

- The incremental process model, like prototyping and other evolutionary approaches, is iterative in nature
- But unlike prototyping, the incremental model focuses on the delivery of an operational product with each increment
- Particularly useful when
  - Staffing is unavailable
- Increments can be planned to manage technical risks

The RAD Model

- Rapid Application Development
- Emphasizes a short development cycle
- A “high speed” adaptation of the waterfall model
- Uses a component-based construction approach
- May deliver software within a very short time period (e.g. , 60 to 90 days) if requirements are well understood and project scope is constrained

The RAD Model



The RAD Model

- The time constraints imposed on a RAD project demand “scalable scope”
- The application should be modularized and addressed by separate RAD teams
- Integration is required

The RAD Model - Drawbacks

- For large, but scalable projects, RAD requires sufficient human resources
- RAD projects will fail if developers and customers are not committed to the rapid-fire activities
- If a system cannot be properly modularized, building the components necessary for RAD will be problematic
- If high performance is an issue, and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work
- RAD may not be appropriate when technical risks are high

Evolutionary Models

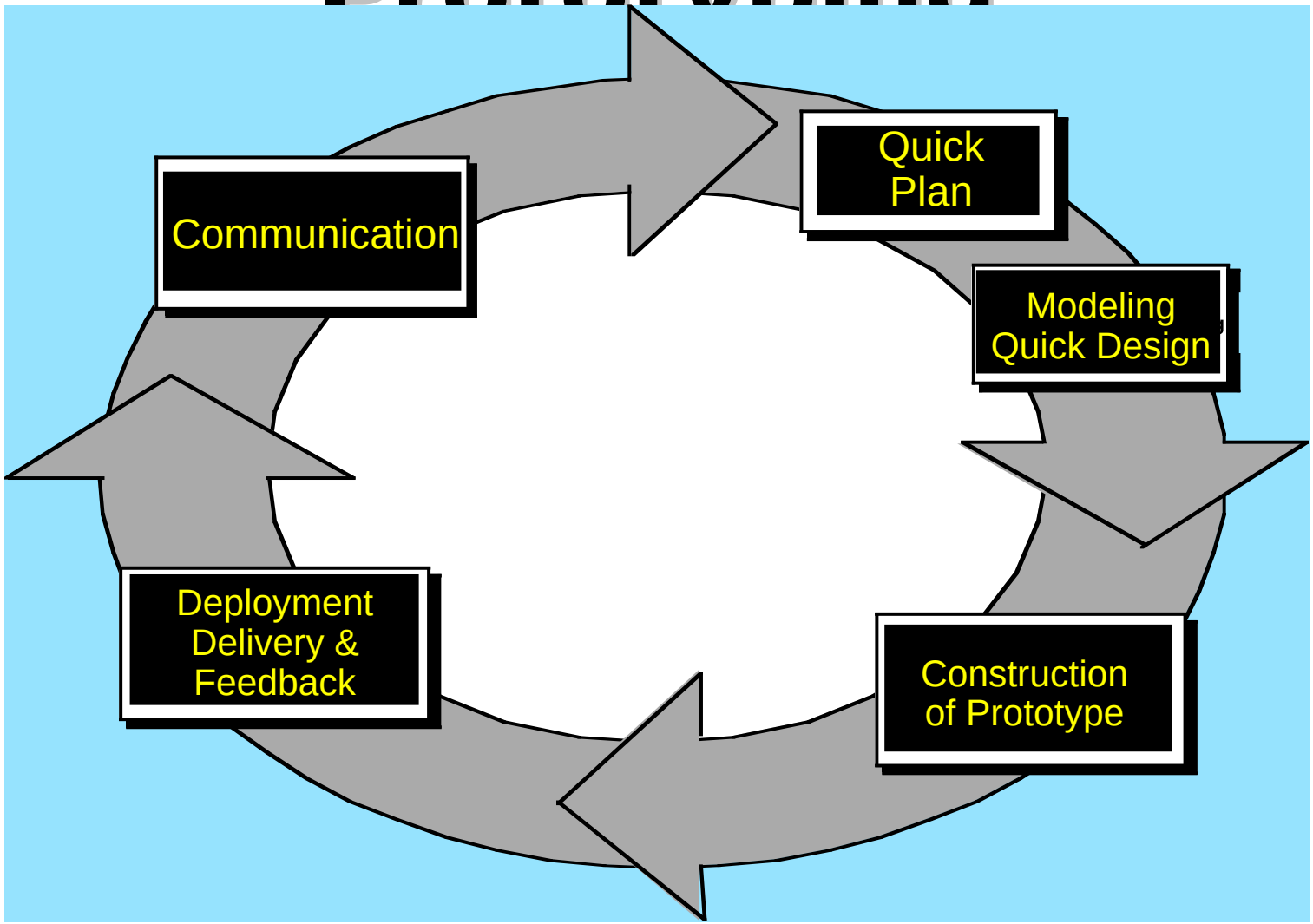
- Software system evolves over time as requirements often change as development proceeds. Thus, a straight line to a complete end product is not possible. However, a limited version must be delivered to meet competitive pressure.
- Usually a set of core product or system requirements is well understood, 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



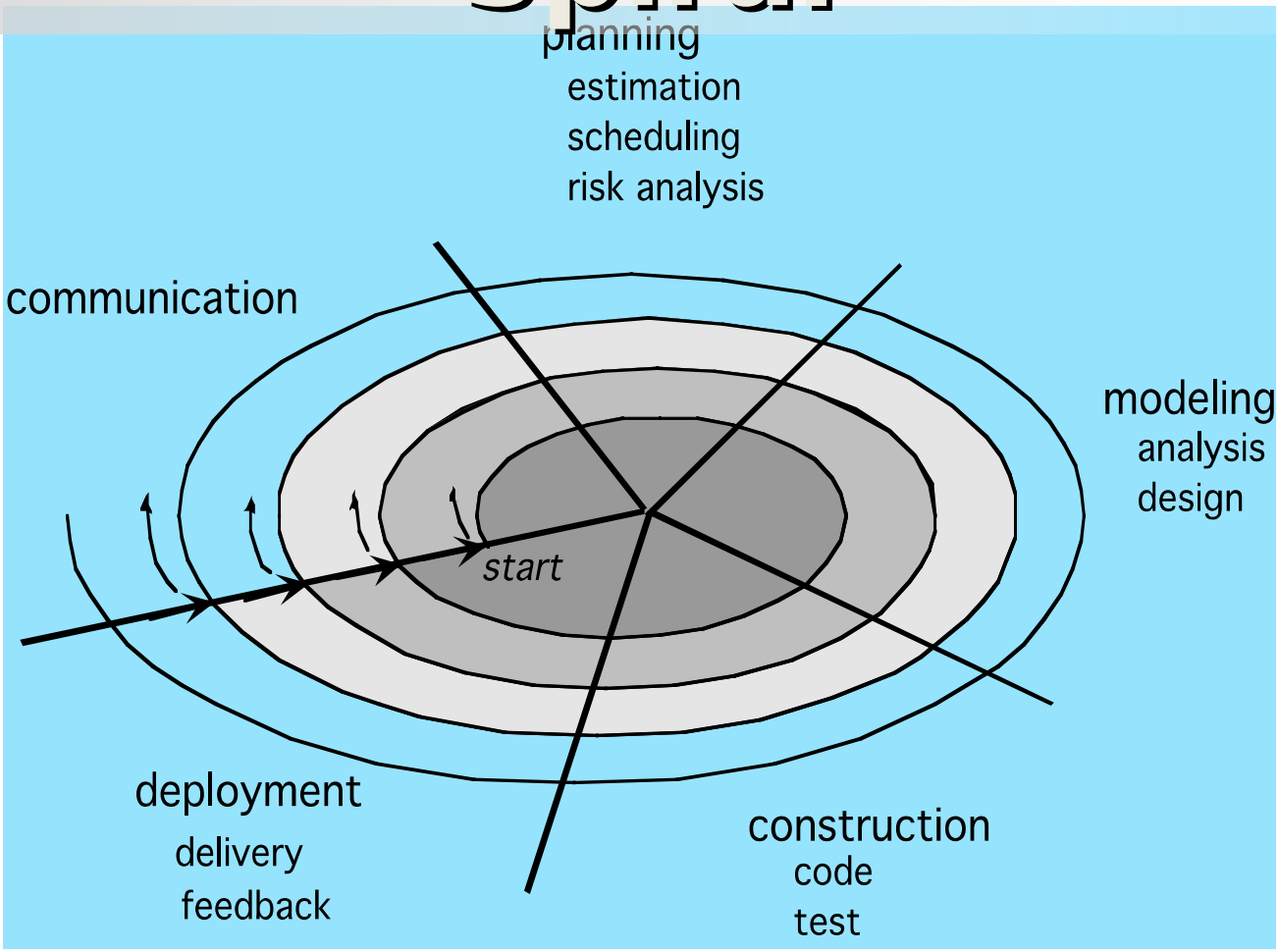
Evolutionary Models: The Spiral

- 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.

Evolutionary Models: The Spiral

- The first circuit in the spiral 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



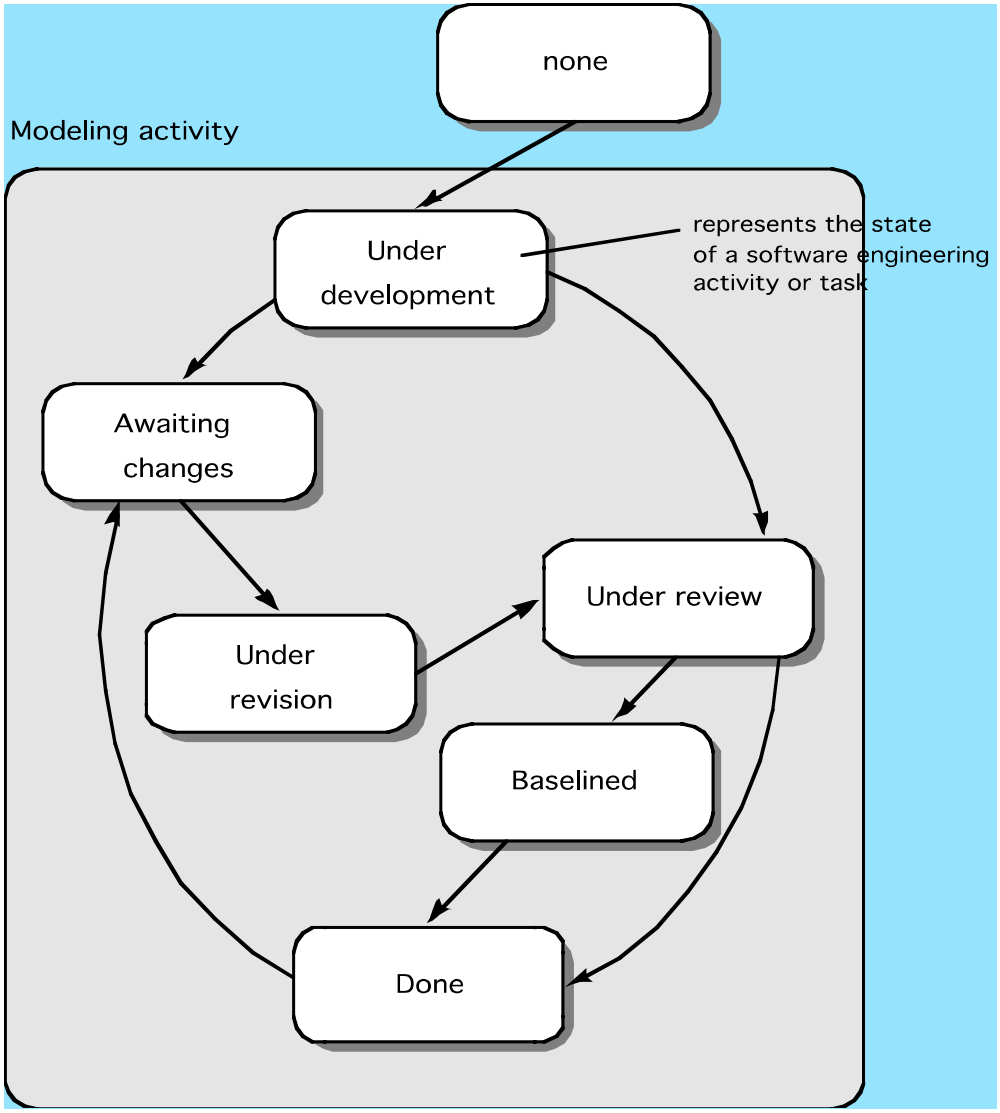
Three Concerns on Evolutionary Processes

- 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

The 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.

The Concurrent Model



Thank You