Raise Your Hands if You Feel You Have Learned something last week (Lecture, Assignment)

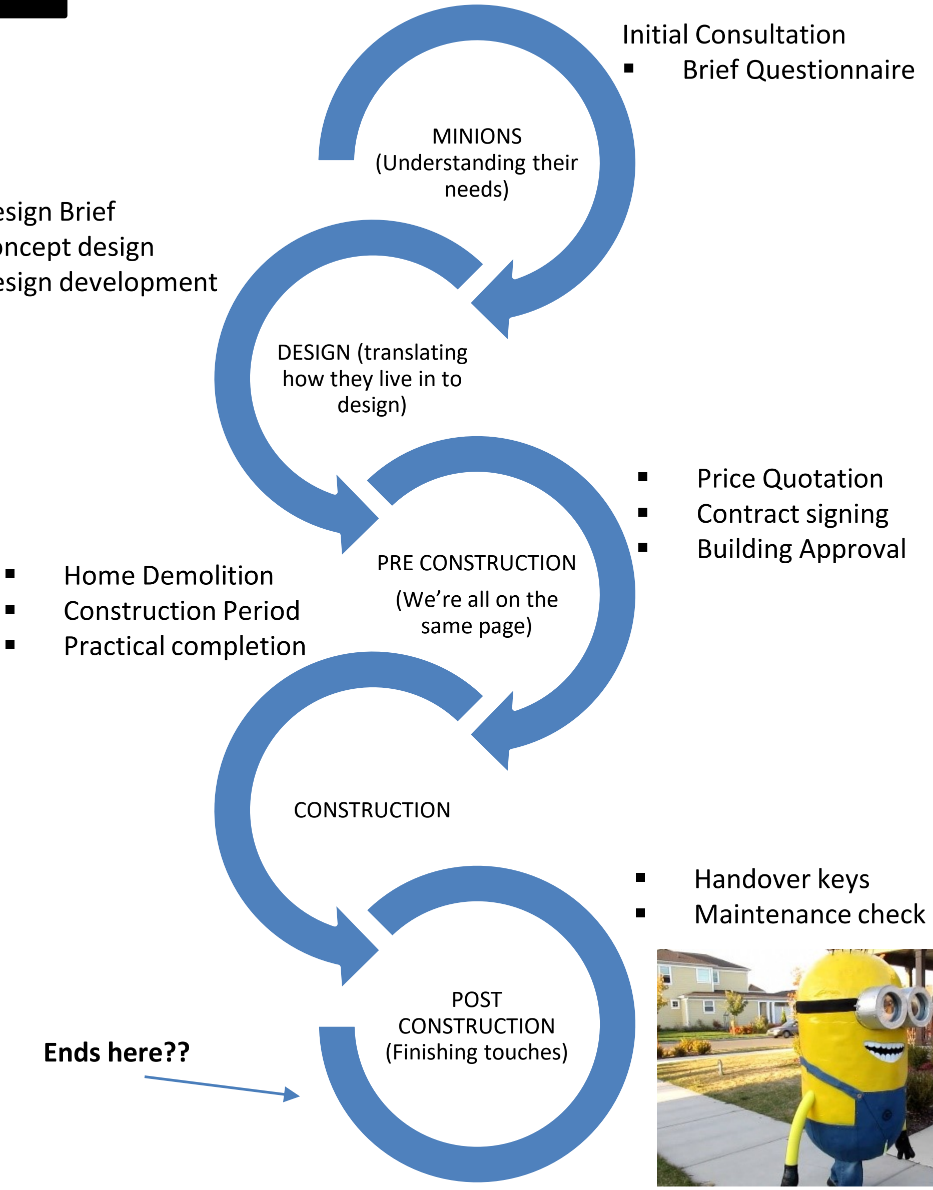
Software Process Structure

- Jayden Khakurel
- Jayden.khakurel@csulb.edu

What is it?

Example: Minion's home

- Minions (Client)
- Who does it?
  - Property Developer (Many stakeholders involved)
- Construction Process
- Key parameters:
  - Communication (Initial consultation, requirement, gathering)
  - Planning
    - Budget
    - Time, project, tools constraints
  - Modeling (Analysis and Design)
  - Construction (wall, floor, test)
  - Post construction (delivery, support, receive feedback)



$$\text{Construction} \xrightarrow{\qquad\qquad\qquad} \text{software}$$

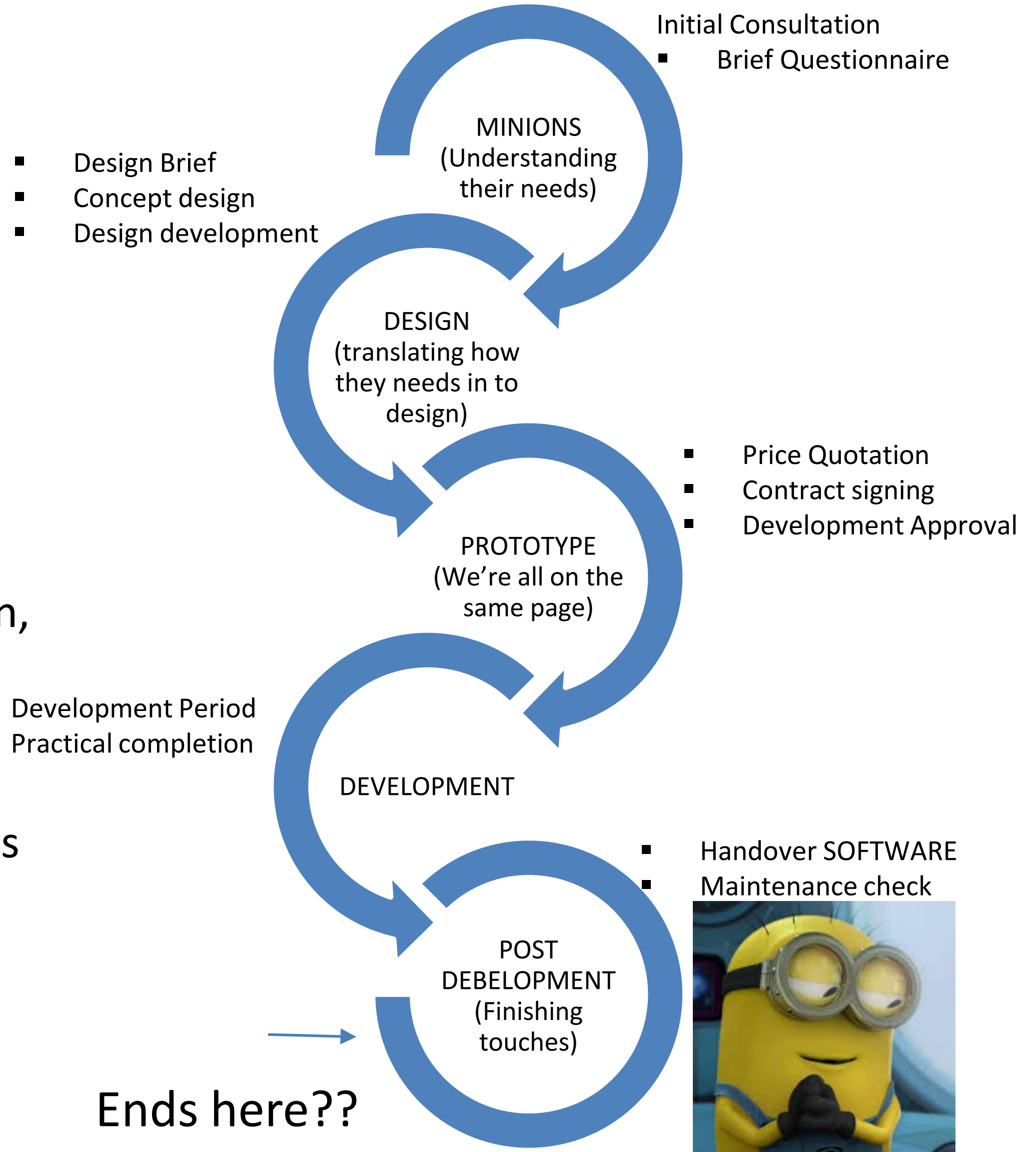
Example: Facebookie

- Minions (Client)

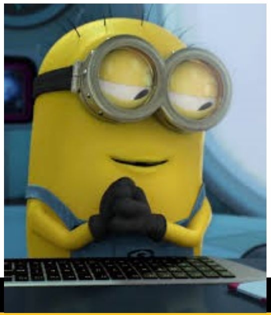
- Who does it?
  - Many stakeholders involved- Software Engineers, designers, Project manager, etc

- Software Process

- Key parameters:
  - Communication (Initial consultation, requirement, gathering)
  - Planning
    - Budget
    - Time, project, tools constraints
  - Modeling (Analysis and Design)
  - Construction (code, test)
  - Deployment(delivery, support, feedback)



Ends here??



Brief

Software process

- A software process is a series of activities, actions and tasks within a framework that helps you to create a timely, high quality software.

Important

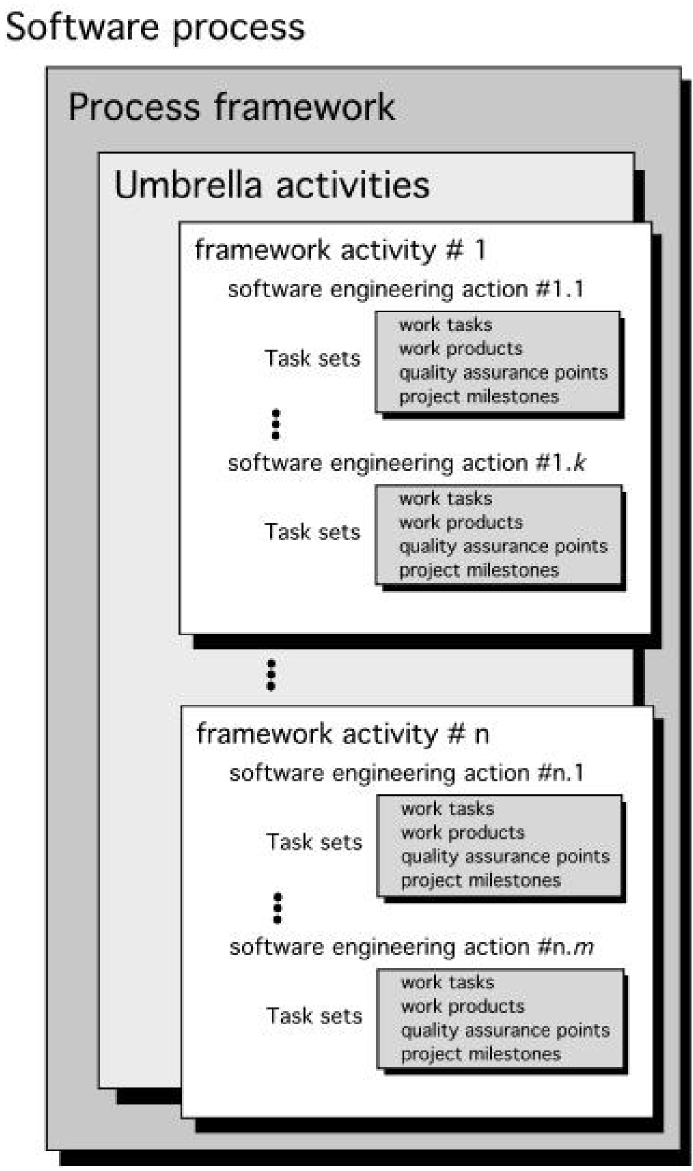
- Provides stability control, and organization to an activity that can, if left uncontrolled becomes quite chaotic.

Steps

- Depends on software you are developing

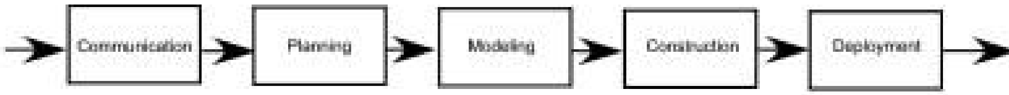
Work product

- Programs, documents and data that are produced as the consequences of the activities and tasks defined by the process.

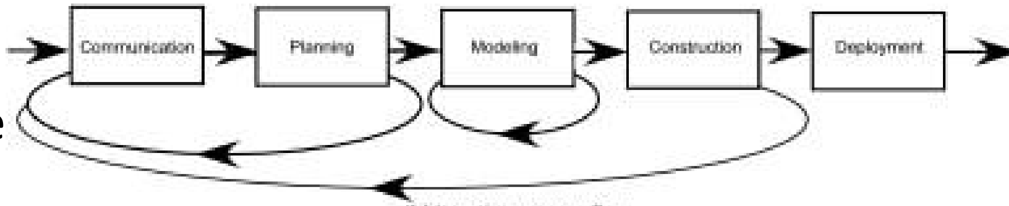


Process flow

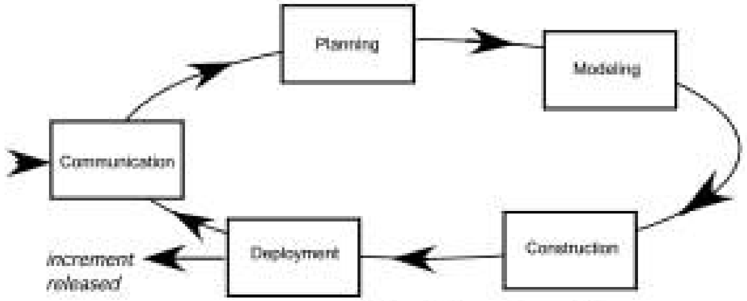
- **Linear :** Executes each of the five framework activities in the sequences,
- **Iterative :** Repeats one or more of the activities before proceeding to the next
- **Evolutionary :** executes the activities in a circular manner
- **Parallel :** executes one or more activities in parallel with other activities



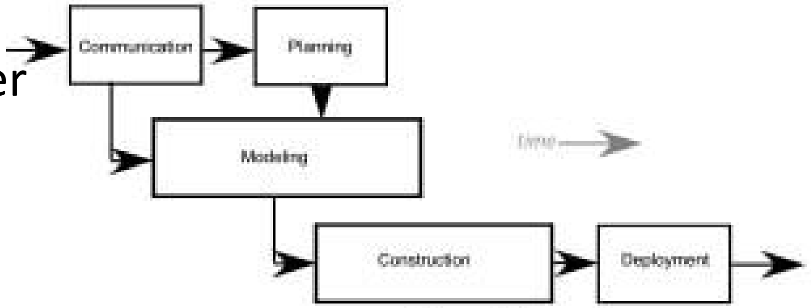
(a) linear process flow



(b) iterative process flow



(c) evolutionary process flow



(d) parallel process flow

Defining a Framework Activity

- A software team would need significantly more information before it could properly execute any one of framework activities
- What actions are appropriate for a framework activity given the following situation?
  - the nature of the problem to be solved,
  - the characteristics of the people doing the work,
  - the stakeholders who are sponsoring the project

Example: Communication Activity

- For a small software project requested by one person
  - Actions $\longrightarrow$ phone conversation
  - Work tasks (task set)
    - •Make contact with stakeholder via telephone
    - •Discuss requirements and develop notes
    - •Organize notes into a brief written statement of requirements
    - •Email to stakeholder for review and approval

Example: Communication Activity

- For a complicated project with many stakeholders
  - Actions
    - Inception
    - Elicitation
    - Elaboration
    - Negotiation
    - Specification
    - Validation

Identifying a task set

- Each software engineering action can be represented by a number of different **task sets**

- 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**

Defining Framework Activity

- Different projects demand different task sets
- The software team chooses the task set based on problem and project characteristics

Process Patterns

- 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.**

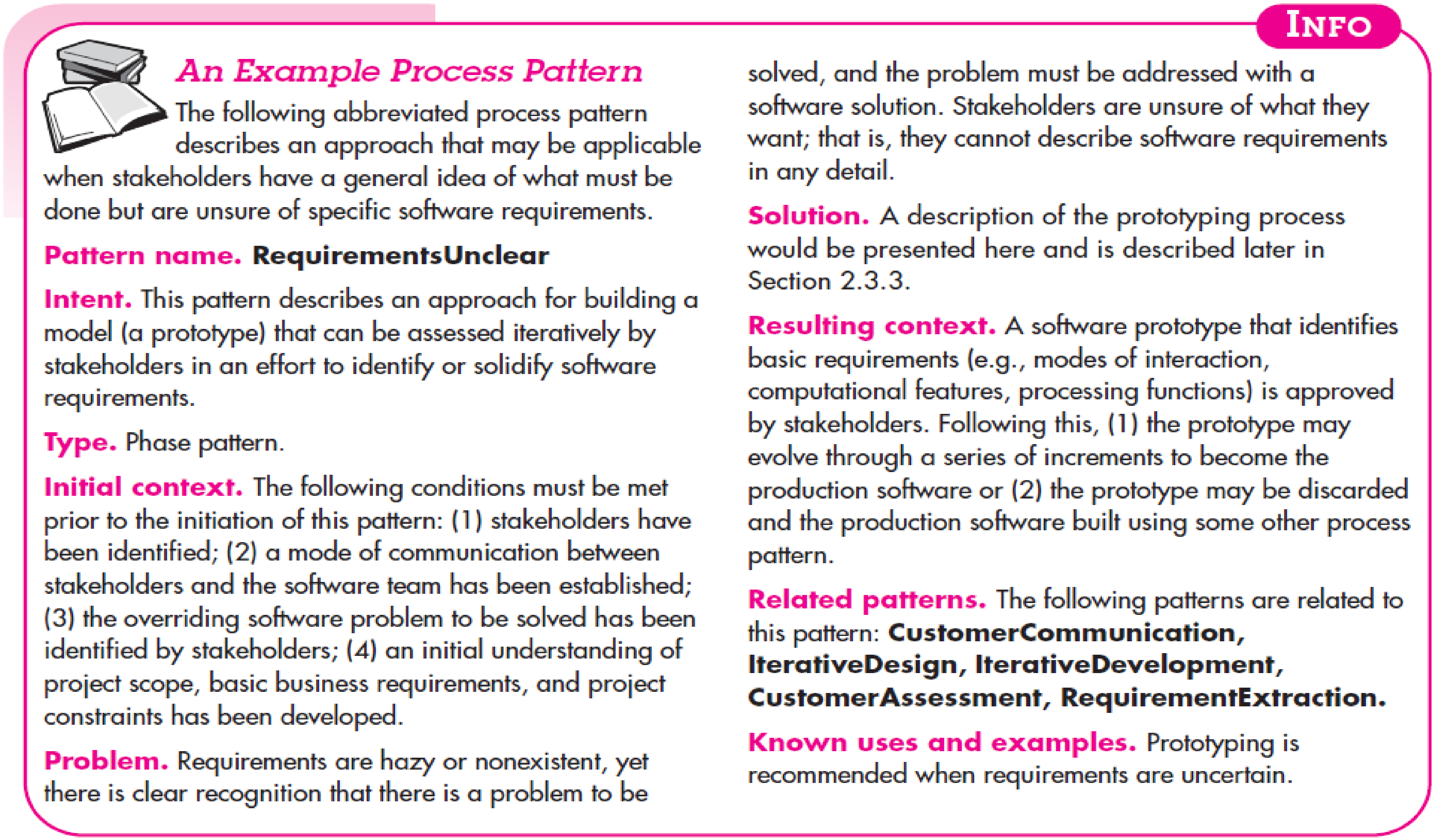
Process Patterns
[Ambler '98]

- Pattern Name
- Forces
  - The environment in which the pattern is encountered and the issues that make the problem visible
- Type: Stage pattern, task pattern, phrase pattern
- Initial context
- Problem
- Solution
- Resulting context
- Related patterns
- Known Uses and Examples

Process Patterns
[Ambler '98]

- Stage patterns—defines a problem associated with a framework activity for the process.
  - E.g.) Establishing Communication
- Task patterns—defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice
  - E.g.) Requirement Gathering
- Phase patterns—define the sequence of framework activities that occur with the process, even when the overall flow of activities is iterative in nature.
  - E.g.) Spiral Model or Prototyping

Process Patterns



INFO

An Example Process Pattern

The following abbreviated 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 (a prototype) that can be assessed iteratively by stakeholders in an effort to identify or solidify software requirements.

Type. Phase pattern.

Initial context. The following conditions must be met prior to the initiation of this pattern: (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, yet there is clear recognition that there is a problem to be solved, and the problem must be addressed with a software solution. Stakeholders are unsure of what they want; that is, they cannot describe software requirements in any detail.

Solution. A description of the prototyping process would be presented here and is described later in Section 2.3.3.

Resulting context. A software prototype that identifies basic requirements (e.g., modes of interaction, computational features, processing functions) is approved by stakeholders. Following this, (1) the prototype may evolve through a series of increments to become the production software or (2) the prototype may be discarded and the production software built using some other process pattern.

Related patterns. The following patterns are related to this pattern: **CustomerCommunication, IterativeDesign, IterativeDevelopment, CustomerAssessment, RequirementExtraction.**

Known uses and examples. Prototyping is recommended when requirements are uncertain.

Process Assessment and Improvement

- The existence of a software process is no guarantee that ..
  - The software will be delivered on time
  - It will meet the customer’s needs
  - It will exhibit the technical characteristics that will lead to long-term quality characteristics
- Process can be assessed to ensure that it meets a set of basic process criteria
  - Process patterns must be coupled with solid software engineering

Process Assessment and Improvement

- **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]

Summary

- A general process model for software engineering encompasses a set of framework and umbrella activities, actions, and work tasks
- Each of a variety of process models can be described by a different process flow
  - Process flow: A description of how the framework activities, actions, tasks are organized sequentially and chronologically
- Process patterns can be used to solve common problems

Process Model

Process Model

- The purpose: to try to reduce the chaos present in developing new software products
- Prescriptive process models
  - Focuses on structure, order and project consistency in software

Prescriptive Process Models

- A prescriptive process models strives for **structure and an order** in software development

- Few questions …
  - Are they appropriate for a software world that thrives on change?
  - When we replace traditional process models (and the order they imply) with something less structured, do we make it impossible to achieve coordination and coherence in software work?
  - No easy answer. Alternatives available to software engineers.

Prescriptive Process Models

Define a prescribed set of process elements and a predictable process work flow

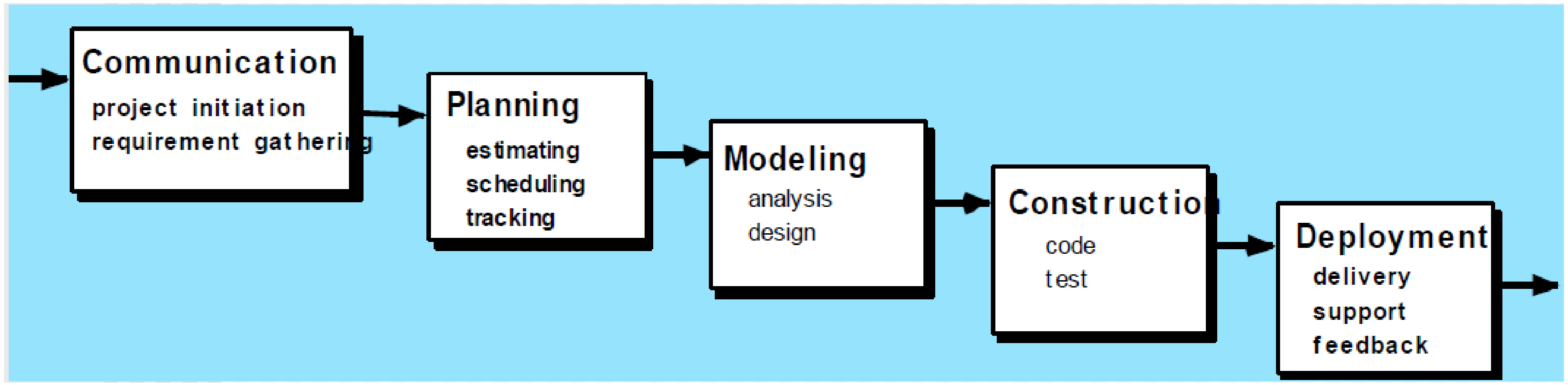
- Framework activities
- Software engineering actions, tasks
- Work products
- Quality assurance
- Change control mechanisms

- A process flow (work flow)
  - The manner in which the process elements are interrelated to one another

Prescriptive Process Models

- The water Fall (life-cycle) model
- The V-model
- Incremental process model
- Evolutionary process model
    - Prototyping
    - Spiral
- Concurrent model

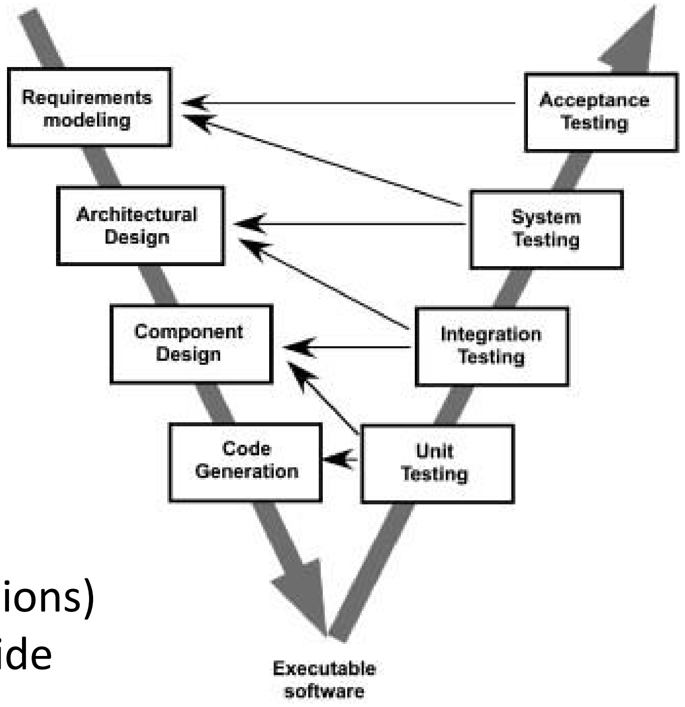
The water fall model



- classic life cycle which suggests a systemic, sequential approach to software development

The V-model

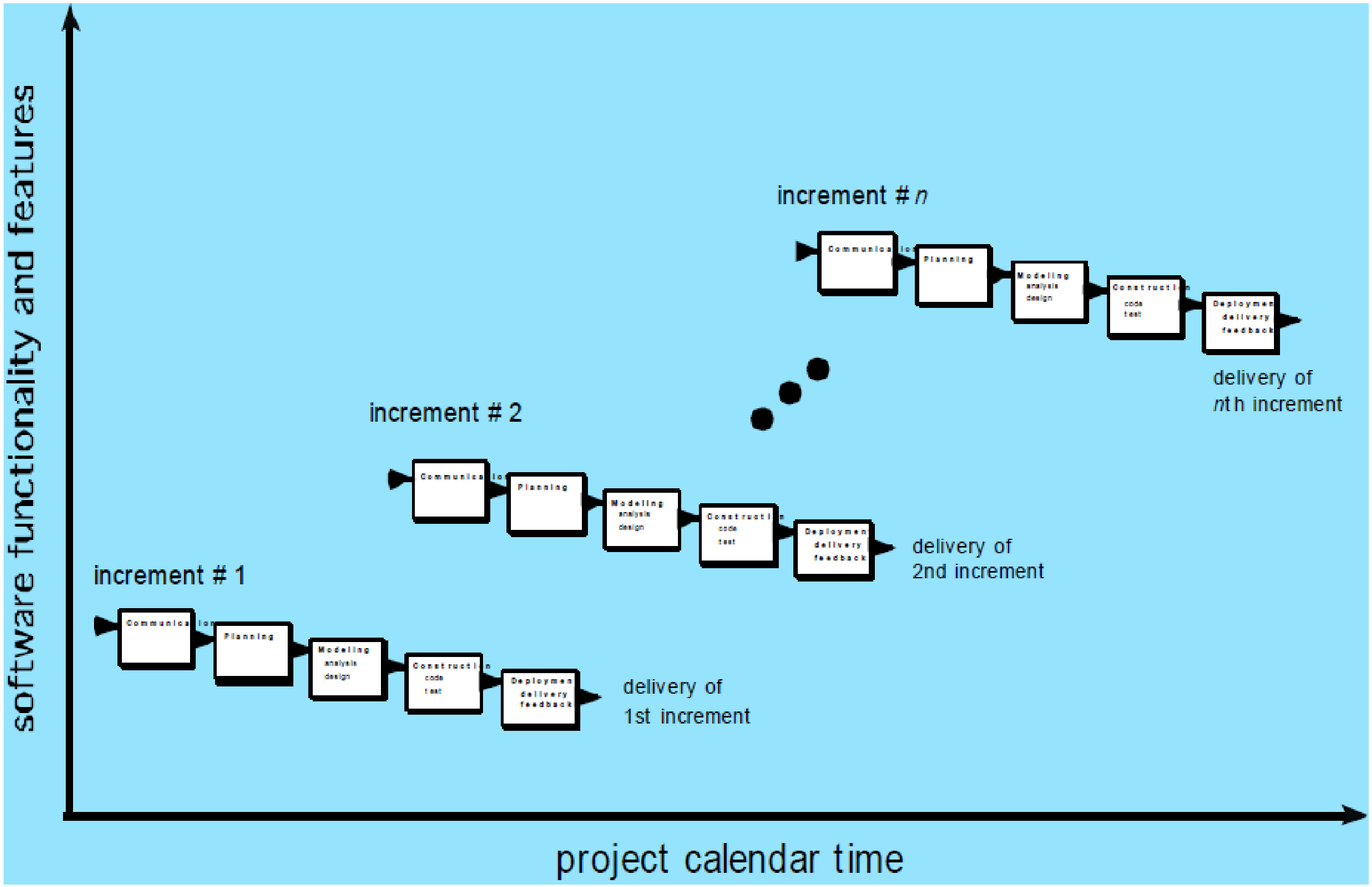
- A variation in the representation of the waterfall model
- Depicts **the relationship of quality assurance actions to the actions associated with some framework activities**, such as communication, modeling, early construction activities
- Illustrates how verification and validation actions are associated with earlier engineering actions
- Moving down the left side of the V
- Moving up the right side of the V
  - Once code has been generated
  - Performing a series of tests (Quality assurance actions)
  - Validates each of the models created on the left side



Criticism

- Real projects rarely follow the sequential flow
  - Changes can cause confusion
- It is often difficult for the customer to state all requirements explicitly
- The customer must have patience
  - A working version of the program 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 Incremental Model


software functionality and features
project calendar time
increment # 1
Communication
Planning
Modeling analysis design
Construction code test
Deployment delivery feedback
delivery of 1st increment
increment # 2
delivery of 2nd increment
increment # n
delivery of nth increment

Criticism

- Motivation
  - Need to provide a limited set of software functionality to users quickly and then refine and expand on that functionality in later software releases

- The incremental model applies **linear sequence in a staggered fashion** as calendar time progresses
  - Each linear sequence **produces deliverable "increments"** of the software

- **Delivers a series of releases, called increments**, that provide progressively more functionality for the customer as each increment is delivered

Example

Example: Word-processing software

- #1) Deliver basic file management, editing, and document production functions
- #2) More sophisticated editing and document production capabilities
- #3) Spelling and grammar checking
- #4) Advanced page layout capability

- The process flow for any increment can incorporate **the prototyping paradigm**

The Incremental Model

- The first increment is often a core product
  - Only basic requirements are addressed
  - It is used by the customer, and he gives feedback
- The plan for the next increment addresses the modification of the core product to better meet the needs of the customer

Evolutionary Models: Motivation

- Software 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
- A set of core product or system requirements is well understood, but the details of product or system extensions have yet to be defined
- We need a process model that has been explicitly designed to accommodate a product that grows and changes

Evolutionary Models

- Produce an increasingly more complete version of the software with each iteration

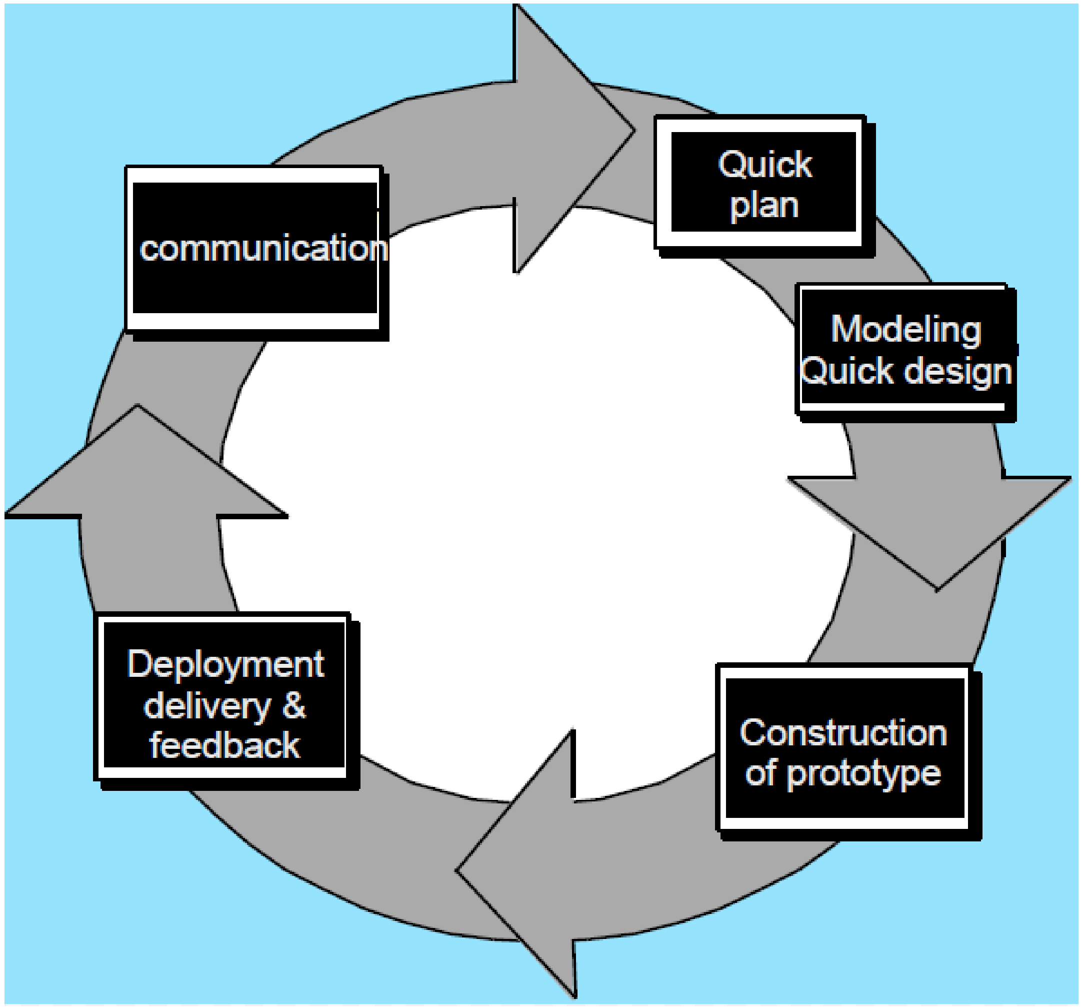
- What is the difference b/w incremental models and evolutionary models?
  - Incremental models $\longrightarrow$ each pass produces simply increments
  - Evolutionary models $\longrightarrow$ each pass produces more complete version of software

Evolutionary Models: Prototyping

- Unclear requirement (by Customer)
  - Often, a customer defines a set of general objectives for software, but **does not identify detailed requirements** for functions and features

- Not-convincing implementation issues (by Developer)
  - **The developer may be unsure of implementation issues**, such as the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take

Evolutionary Models: Prototyping



Prototyping : Flow

- Communication
  - You meet with other stakeholders to define the overall objectives, identify known requirements, and outline areas where further definition are mandatory
- **Quick plan**
  - A prototyping iteration is planed quickly
- Modeling
- **Quick design**
  - Focuses on a representation of those aspects of the software that will be visible to end users
- Construction of prototype
- Deployment, delivery & **feedback**
  - The prototype is deployed and evaluated

Prototyping

- Iteration occurs as the prototype is tuned to satisfy the needs of various stakeholders, while at the same time enabling you to better understand what needs to be done
- The prototype serves as a mechanism for **identifying software requirements**
- What do you do with the prototype?
  - The prototype serves as the first system
  - Ideal case: **Prototypes are built as throwaways**
    - They are the ones that Brooks recommends you throw away
  - **But, some others are evolutionary** in the sense that the prototype slowly evolves into the actual system

Prototyping: Advantages

- Both stakeholders and software engineer can like the prototyping paradigm
  - User gets a feel for the actual system
  - Developers get to build something immediately

Prototyping: Limitations

- Stakeholders see what appears to be a working version of the software, but unaware of the inside process
    - they unaware that the prototype is held together haphazardly, unware that in the rush to get it working
    - Overall software quality or long-term maintainability is not considered
  - When informed that the product must be rebuilt for high quality, **stakeholders will only demand "a few fixes" over "the prototype" due to the cost problem**

- Software engineers make implementation compromises in order to get a prototype working quickly
    - An inappropriate operating system or programming language may be used simply; an inefficient algorithm many be implemented simply

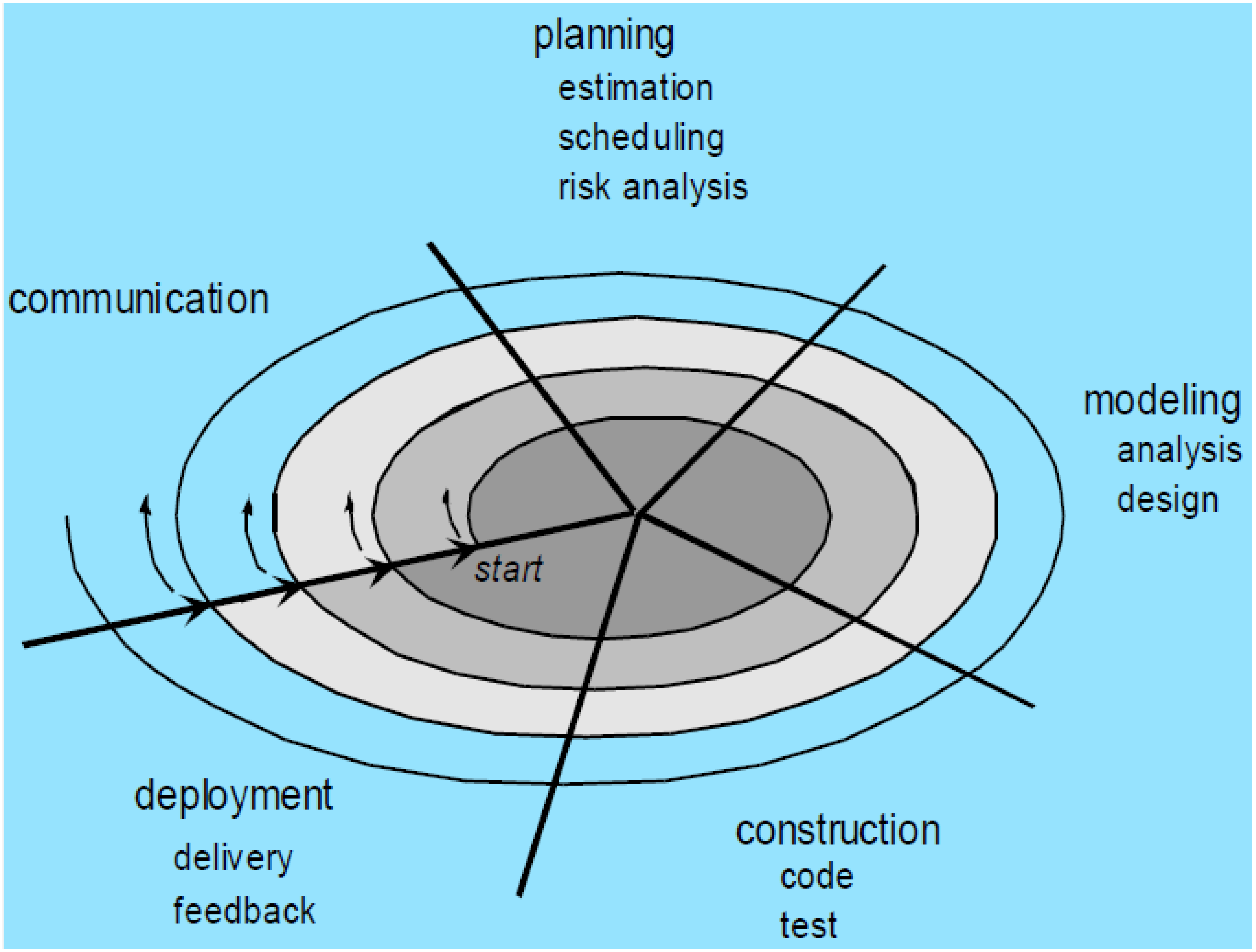
- **They become comfortable with less-than-ideal choice**, which becomes an integral part of the final system

- The final system suffers from its low quality

Prototyping: Principle

- All stakeholder should agree the following:
- 1. The prototype is build to serve as a mechanism for defining requirements
- 2. The prototype is then discarded (at least in part)
- 3. The actual software is engineered with an eye toward quality

Evolutionary Models: The Spiral



The Spiral model

- Characteristics
  - Couples the **iterative nature of prototyping** with the controlled and systematic aspects of the waterfall model
  - Provides **the potential for rapid development** of increasingly more complete versions of the software

- The spiral model can be adopted to **apply throughout the entire life cycle of an application**, from concept development to maintenance

The Spiral model : Adaptation

- Adaptively represents whole development process throughout the life of the software

- Stage 1) **Concept development project** $\rightarrow$
- Stage 2) **New product development project** $\rightarrow$
- Stage 3) **Product enhancement project** $\rightarrow$ …

- The spiral remains operative until the software is retired

The Spiral model : Adaptation

- The spiral model is **a risk-driven process model generator** that is used to guide multi-stakeholder concurrent engineering of software intensive systems.

- 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 solution

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

- **Each of the framework activities represent one segment of the spiral path**
  - Risk is considered as each revolution is made
  - **Anchor point milestones** are noted for each evolutionary process
    - A combination of work products and conditions that are attained along the path of the spiral

The Spiral model: Evolutionary process

- The first pass: Result in the development of a **product specification**
- Subsequent passes: Develop a **prototype** and then **progressively more sophisticated versions of the software**
- Each pass through the planning region: results in **adjustments to the project plan**
- **Cost and schedule are adjusted based on feedback** derived from the customer after delivery
- Project manager **adjusts the planed number of iterations** required to complete the software

The Spiral model: Advantages

- A realistic approach to the development of **large-scale systems and software**
- **The developer and customer better understand and react to risks** at each evolutionary level
  - As a risk reduction mechanism, it applies the prototyping approach at any stage in the evolution
- **Incorporates the classic life cycle into an iterative framework** $\longrightarrow$ more realistically reflects the real world
- Demands a direct consideration of technical risks at all stages of the project $\longrightarrow$ **reduce risks** before they become problematic

The Spiral model: Limitations

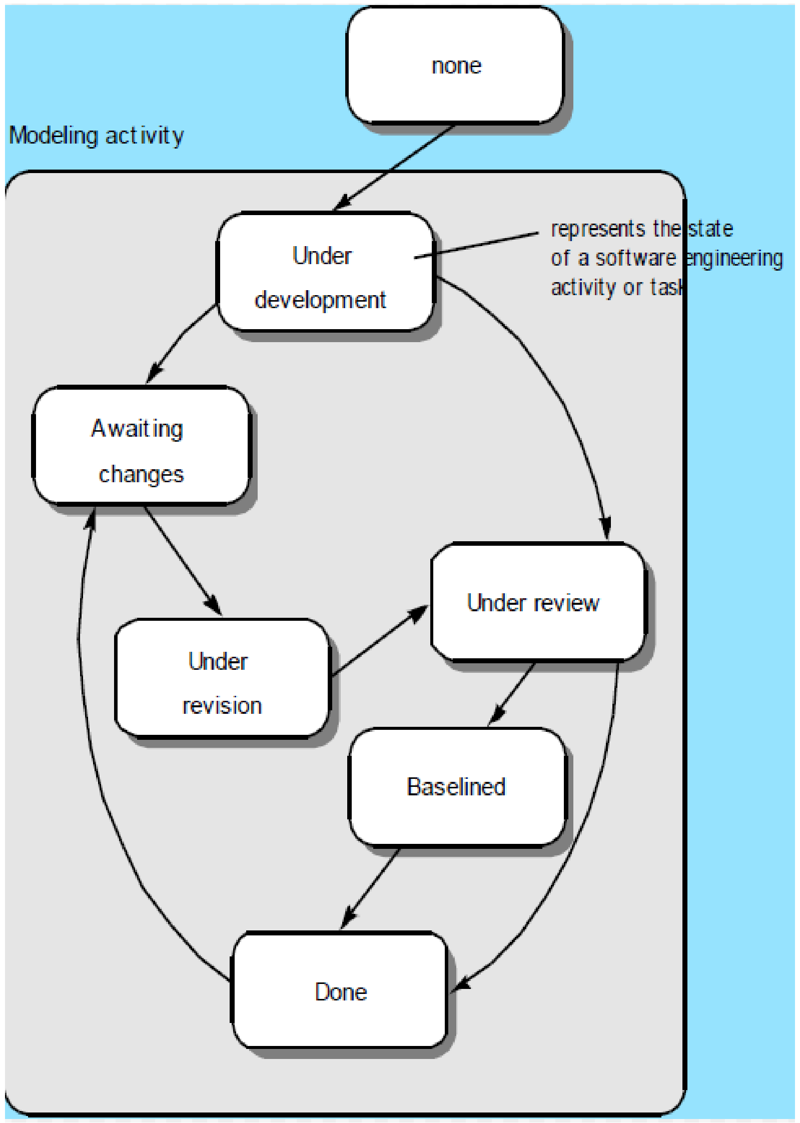
- It may be difficult to convince customers that the evolutionary approach is controllable
- It **demands considerable risk assessment expertise** and relies on this expertise for success
- If a major risk is not uncovered and managed, problems will undoubtedly occur

Evolutionary Process: The concurrent Models

- Allows a software team to **represent iterative and concurrent elements** of any of the process models

- E.g.: The **modeling activity** defined for the spiral model is accomplished by invoking one or more of the software engineering actions –**prototyping, analysis, and design**

The concurrent Models



The concurrent Models: Example

- The communication activity has completed its first iteration $\longrightarrow$
- The modeling activity makes a transition from **inactive(none)** state into the **under development** $\longrightarrow$
- The customer indicates that changes in requirements must be made $\longrightarrow$
- the modeling activity moves from the **under development** state into **the awaiting changes** state

The concurrent Models: Example

- **Applicable to all types of software development** and provides an accurate picture of the current state of a project
- **Defines a process network**, rather than confining software engineering activities, actions, and tasks to a sequence of events
- **Each activity, action or task on the network exists simultaneously** with other activities, actions or tasks
- **Events generated in the process network trigger transitions** among the states of each activity

Evolutionary Process Models: Weaknesses [Nogueira '00]

- Prototyping poses **a problem of project planning** because of the **uncertain number of cycles** required to construct the product
- Evolutionary software processes do not establish the maximum speed of the evolution
- Evolutionary software processes should be focused on flexibility and extensibility rather than high quality