

# Process Models

*For non-profit educational use only*

May be reproduced ONLY for student use at the university level when used in conjunction with *Software Engineering: A Practitioner's Approach*, 7/e. Any other reproduction or use is prohibited without the express written permission of the author.

These slides are designed to accompany *Software Engineering: A Practitioner's Approach*, 7/e (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

Some of the slides are taken from Sommerville, I., *Software Engineering*, Pearson Education, 9<sup>th</sup> Ed., 2010 and also other sources. Those are explicitly indicated

# Software Characteristics

## Review

- Software is developed or engineered....Not manufactured in the classical sense
- Software does NOT “wear out”
  - Aging of software is very unlike material objects (hardware)
- Industry is moving towards component-based construction, but most of the software effort is for the custom-building

# Software Engineering - Definitions

## Review

- (1969 – Fritz Bauer) Software engineering is the establishment and use of *sound engineering principles* in order to obtain *economically* software that is *reliable* and works *efficiently* on *real machines*
  - Some may want to have technical aspects, customer satisfaction, timeliness, measurements, process included in the definition
- (IEEE) The application of a *systematic, disciplined, quantifiable* approach to the *development, operation, and maintenance* of software; that is, the application of engineering to software
  - Some may consider *adaptability and agility* more important than *systematic, disciplined, quantifiable approach*

# Software engineering

(as per author Sommerville)

## Review

- Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.
  - Engineering discipline
    - Using appropriate theories and methods to solve problems bearing in mind organizational and financial constraints.
  - All aspects of software production
    - Not just technical process of development. Also project management and the development of tools, methods etc. to support software production.

# The software process (as per Sommerville)

## Review

- A structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - Specification – defining what the system should do;
  - Design and implementation – defining the organization of the system and implementing the system;
  - Validation – checking that it does what the customer wants;
  - Evolution – changing the system in response to changing customer needs.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

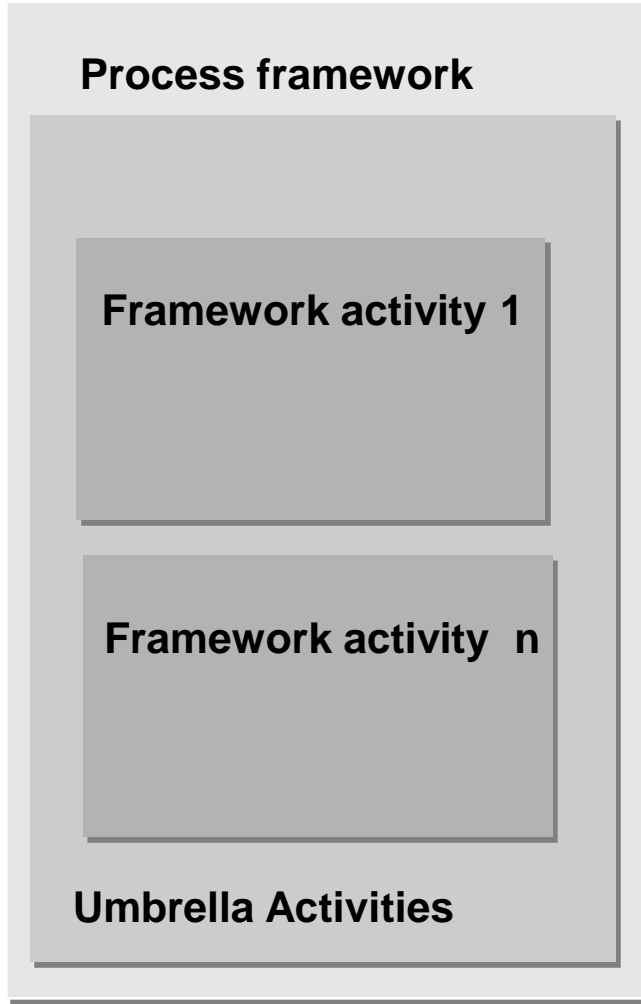
# Generic Process Framework (as per Pressman)

## Review

- **Communication**
  - Involves communication among the customer and other stake holders; encompasses requirements gathering
- **Planning**
  - Establishes a plan for software engineering work; addresses technical tasks, resources, work products, and work schedule
- **Modeling (Analyze, Design)**
  - Encompasses the creation of models to better understand the requirements and the design
- **Construction (Code, Test)**
  - Combines code generation and testing to uncover errors
- **Deployment**
  - Involves delivery of software to the customer for evaluation and feedback

# A Process Framework

## Software Process



## Review

### Process framework

**Framework activities**  
**work tasks**  
**work products**  
**milestones & deliverables**  
**QA checkpoints**

**Umbrella Activities**

# A Process Framework

Review

## Process framework

### Process framework

#### Framework activities

work tasks

work products

milestones &

deliverables

QA checkpoints

#### Umbrella Activities

#### Modeling activity

Software Engineering action: **Analysis**

work tasks: requirements gathering,  
elaboration, negotiation,  
specification, validation

work products: analysis model and/or  
requirements specification

milestones & deliverables

QA checkpoints

Software Engineering action: **Design**

work tasks: data design, architectural,  
interface design,  
component design

work products: design model  
and/or design specification

.....

#### Umbrella Activities



# Review **How Process Models Differ?**

While all Process Models take same framework and umbrella activities, they differ with regard to

- Overall flow of activities, actions, and tasks and the interdependencies among them
- Degree to which actions and tasks are defined within each framework activity
- Degree to which work products are identified and required
- Manner in which quality assurance activities are applied
- Manner in which project tracking and control activities are applied
- Overall degree of detail and rigor with which the process is described
- Degree to which customer and other stakeholders are involved in the project
- Level of autonomy given to the software team
- Degree to which team organization and roles are prescribed

# Process Patterns

## Review

- A proven solution to a problem
  - Describes process-related problem
  - Environment in which it was encountered
- A template to describe solution
  - Ambler proposed a template for describing a process pattern
- Can be defined at any level of abstraction
  - Complete process model (e.g. Prototyping)
  - Frame work activity (e.g. Planning)
  - An action within framework activity (e.g. Estimation)

# Process Patterns – Ambler's Template

## Review

- Pattern Name
- Forces
  - Describes environment of the problem
- Type
  - Phase (e.g. Prototyping), Stage (e.g. Planning), or Task (e.g. Estimation)
- Initial Context
- Problem
- Solution
- Resulting Context
- Related Patterns
- Known uses & examples

# Prescriptive and agile processes

- Prescriptive processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes may include elements of both plan-driven and agile approaches.
- *There are NO right or wrong software processes.*

# Prescriptive Process Model

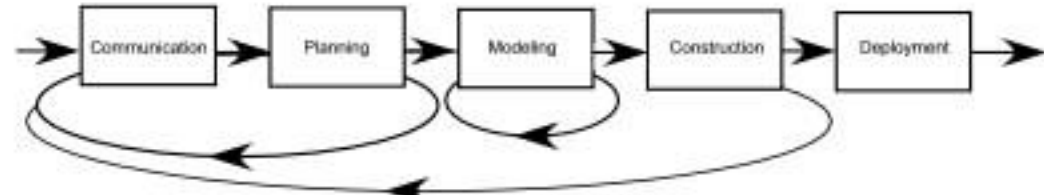
- Defines a distinct set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software
- The activities may be linear, incremental, or evolutionary

# Process Models

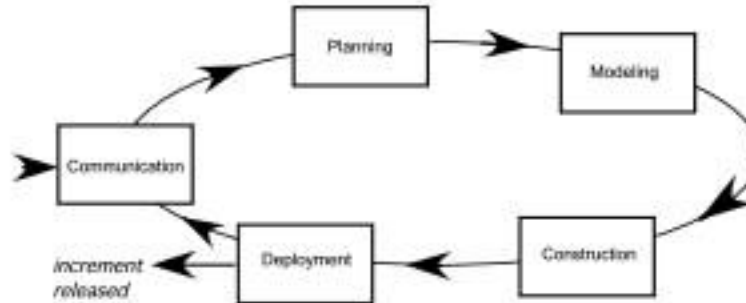
An important variation among process models comes from flow of activities



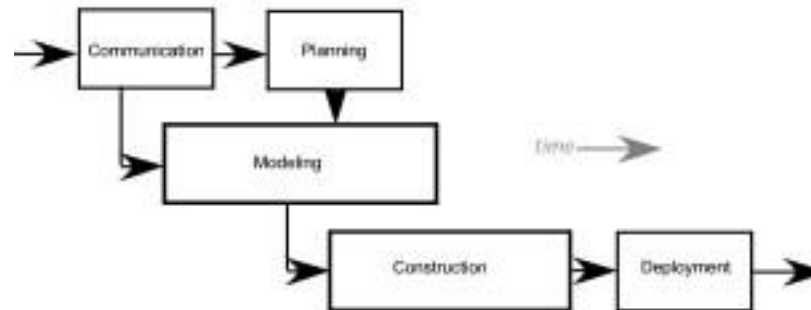
(a) linear process flow



(b) iterative process flow



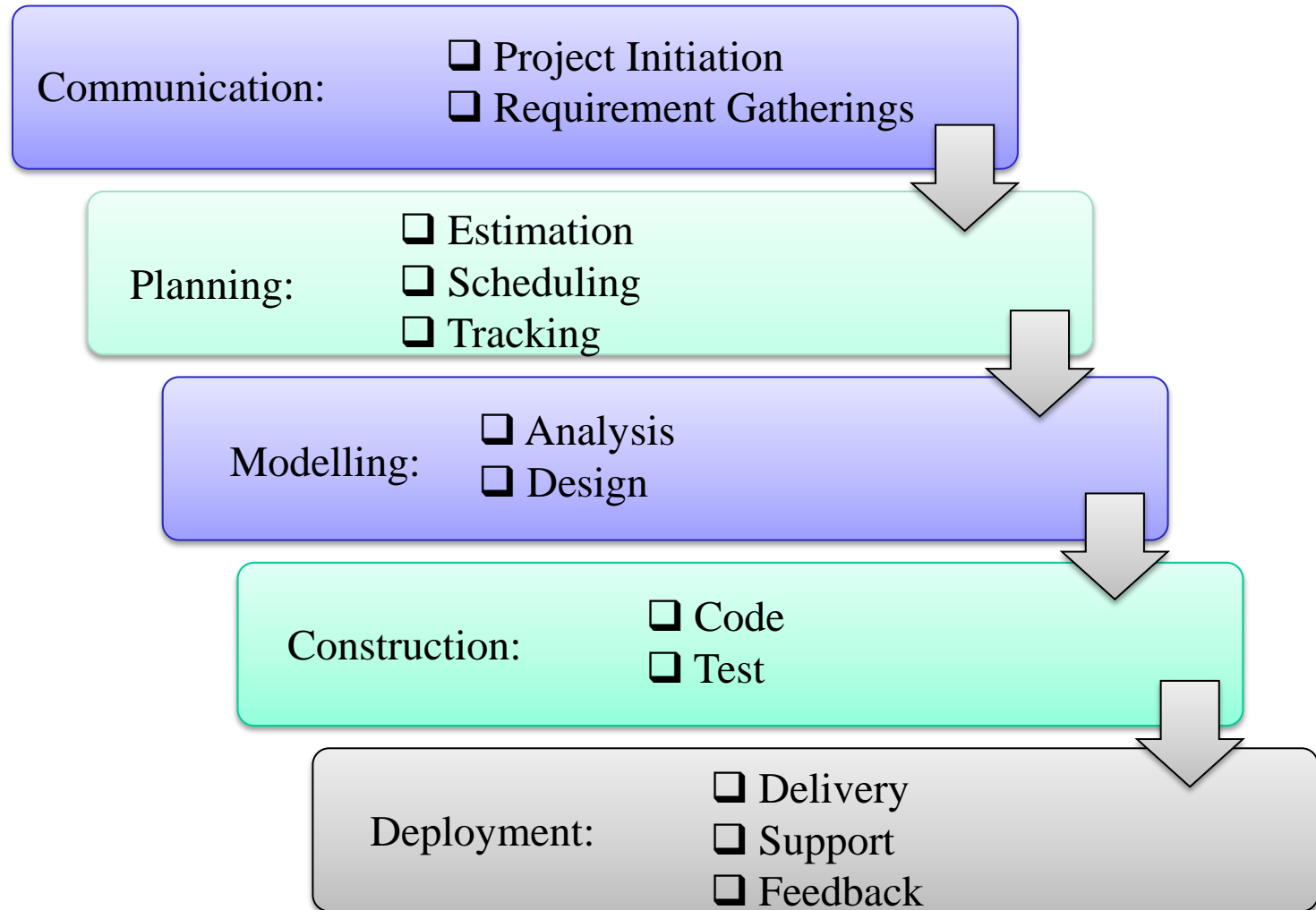
(c) evolutionary process flow



(d) parallel process flow

# Waterfall Model

(Diagram)



# Critique of the waterfall model

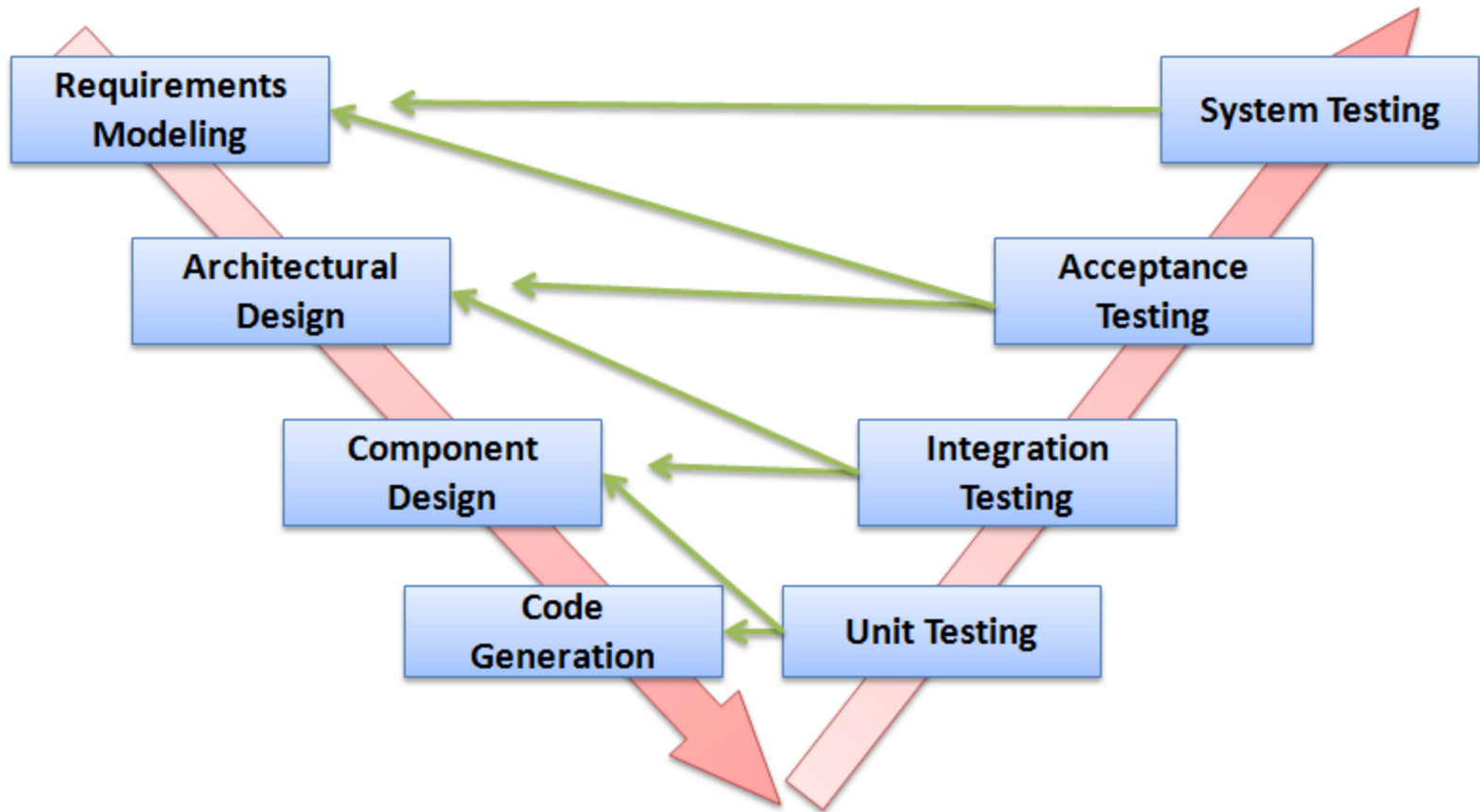
- The model implies that you should attempt to complete a given stage before moving on to the next stage
  - Does not account for the fact that requirements constantly change.
  - It also means that customers can not use anything until the entire system is complete.
- The model makes no allowances for prototyping.
- Assumes understanding of problem and full requirements early on
- It implies that you can get the requirements right by simply writing them down and reviewing them.
- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.



# Critique of Waterfall Model continued

- Follows systematic approach to development
- The model implies that once the product is finished, everything else is maintenance.
- Assumes patience from customer
- Surprises at the end are very expensive
- Some teams sit idle for other teams to finish
- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.

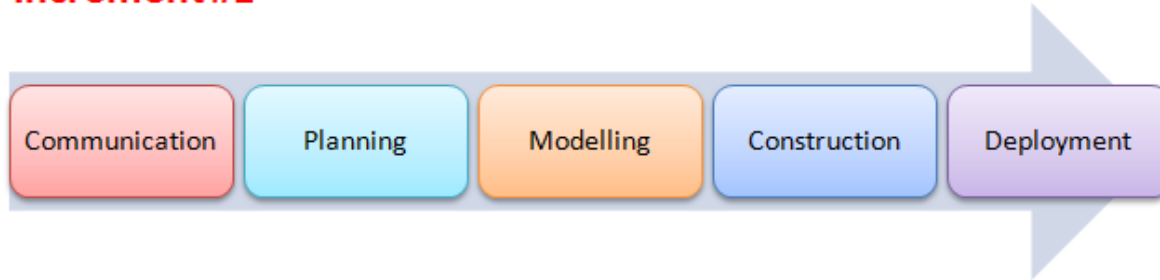
# V-Model



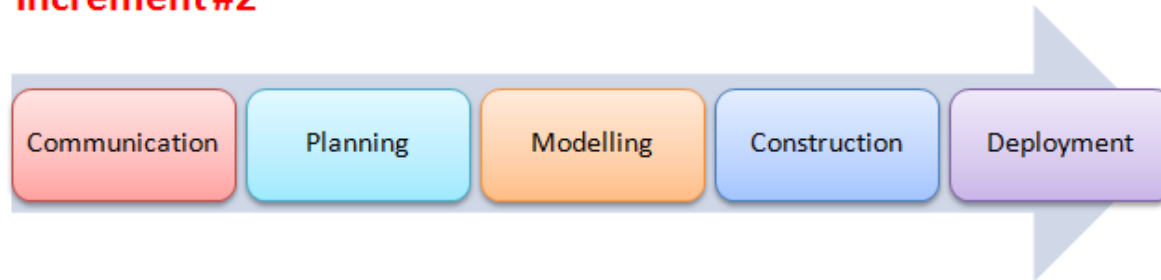
# Incremental Model

(Diagram)

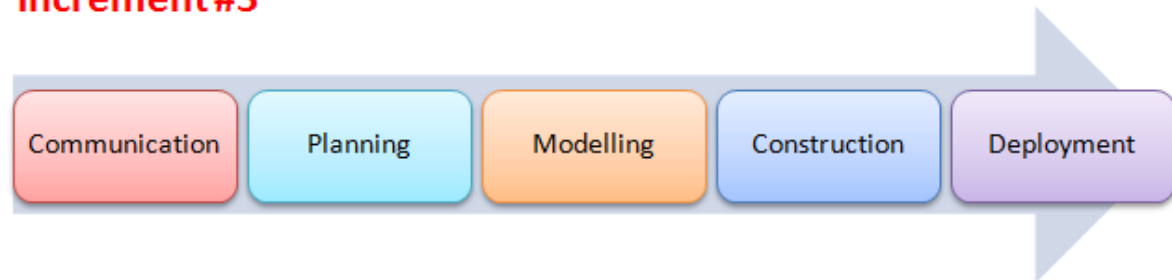
## Increment #1



## Increment #2



## Increment #3



# The Incremental Model

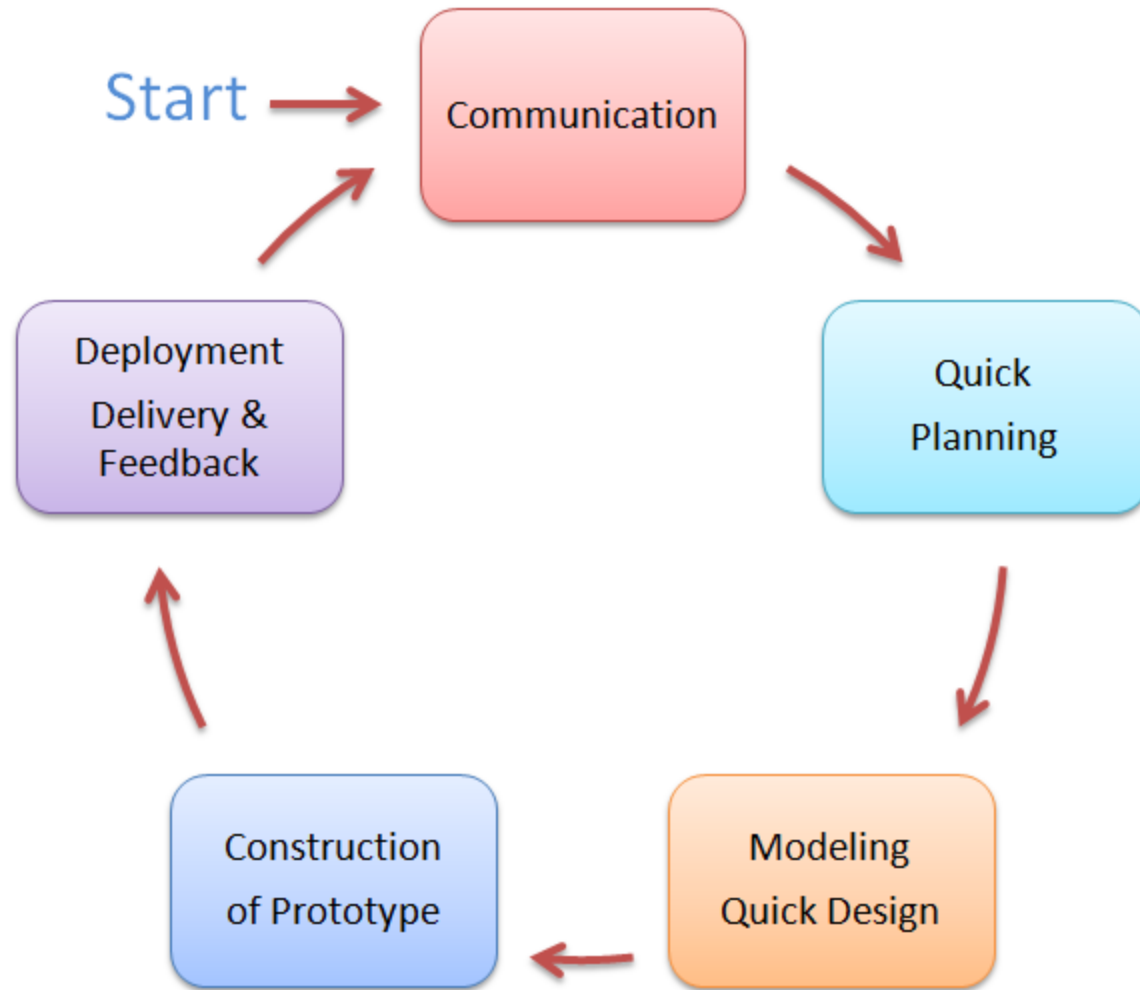
- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- First Increment is often core product
  - Includes basic requirement
  - Many supplementary features (known & unknown) remain undelivered
- First Increment is used or evaluated
- A plan of next increment is prepared
  - Modifications of the first increment
  - Additional features of the first increment
- It is particularly useful when enough staffing is not available for the whole project
- Increment can be planned to manage technical risks

# The Incremental Model

- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.
- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.

# Prototyping Model

(Diagram)



# Prototyping Model

(Description)

- Follows an evolutionary and iterative approach
- Used when requirements are not well understood
- Serves as a mechanism for identifying software requirements
- Focuses on those aspects of the software that are visible to the customer/user
- Feedback is used to refine the prototype

# Prototyping Model

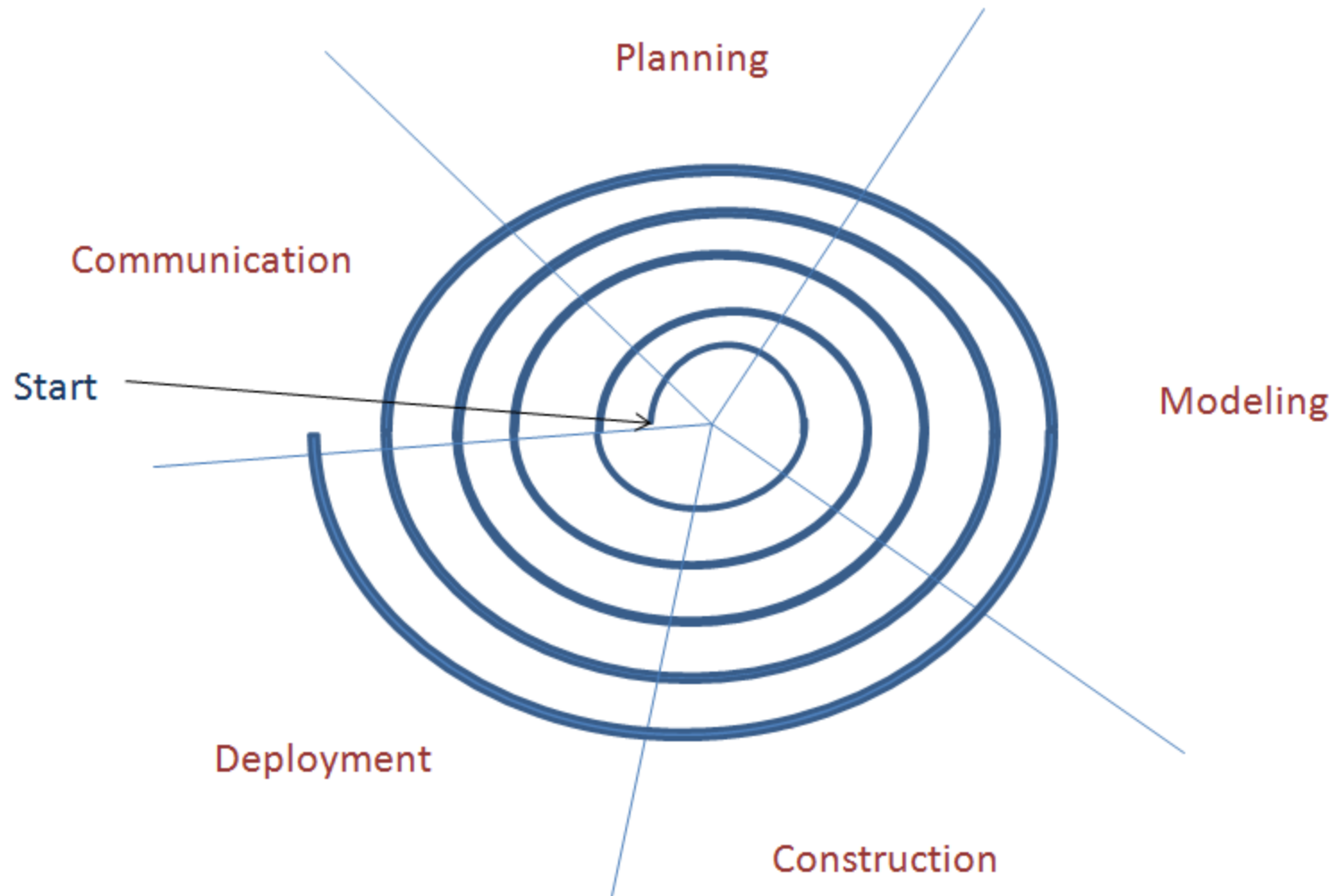
(Potential Problems)

- The customer sees a "working version" of the software, wants to stop all development and then buy the prototype after a "few fixes" are made
- Developers often make implementation compromises to get the software running quickly (e.g., language choice, user interface, operating system choice, inefficient algorithms)
- Important Considerations:
  - Define the rules up front on the final disposition of the prototype before it is built
  - In most circumstances, plan to discard the prototype and engineer the actual production software with a goal toward quality



# Spiral Model

(Diagram)



# Spiral Model

- Proposed by Dr. Barry Boehm in 1988 while working at TRW
- Follows an evolutionary approach
- Used when requirements are not well understood and risks are high
- Inner spirals focus on identifying software requirements and project risks; may also incorporate prototyping
- Outer spirals take on a classical waterfall approach after requirements have been defined, but permit iterative growth of the software
- Operates as a risk-driven model...a go/no-go decision occurs after each complete spiral in order to react to risk determinations
- Requires considerable expertise in risk assessment
- Serves as a realistic model for large-scale software development

# General Weaknesses of Evolutionary Process Models

As per Nogueira et al,

- 1) Evolutionary models pose a problem to project planning because of the uncertain number of iterations required to construct the product
- 2) Evolutionary software processes do not establish the maximum speed of the evolution
  - If too fast, the process will fall into chaos
  - If too slow, productivity could be affected
- 3) Software processes should focus first on flexibility and extensibility, and second on high quality
  - We should prioritize the speed of the development over zero defects
  - Extending the development in order to reach higher quality could result in late delivery

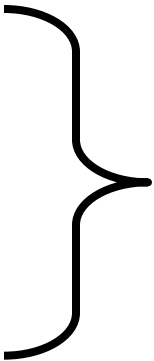
# Beyond the Text Book

Pressman included Spiral Model proposed by Barry Boehm in 1988. In recent times Barry Boehm is advocating Incremental Commitment Spiral Model (ICSM) for addressing future Hardware-software-human integration problems such as

- ***Complex, multi-owner systems of systems.*** Current collections of incompatible, separately-developed systems will cause numerous challenges.
- ***Emergent requirements.*** Demands for most appropriate user interfaces and collaboration modes for a complex human-intensive system.
- ***Rapid change.*** Specifying current-point-in-time snapshot requirements on a cost-competitive contract generally leads to a big design up front, and a point-solution architecture that is hard to adapt to new developments.
- ***Reused components.*** Reuse-based development has major bottom-up development implications, and is incompatible with pure top-down requirements-first approaches.
- ***High assurance of qualities.*** Future systems will need higher assurance levels of such qualities as safety, security, reliability/availability/maintainability, performance, adaptability, interoperability, usability, and scalability.

# What is the ICSM?

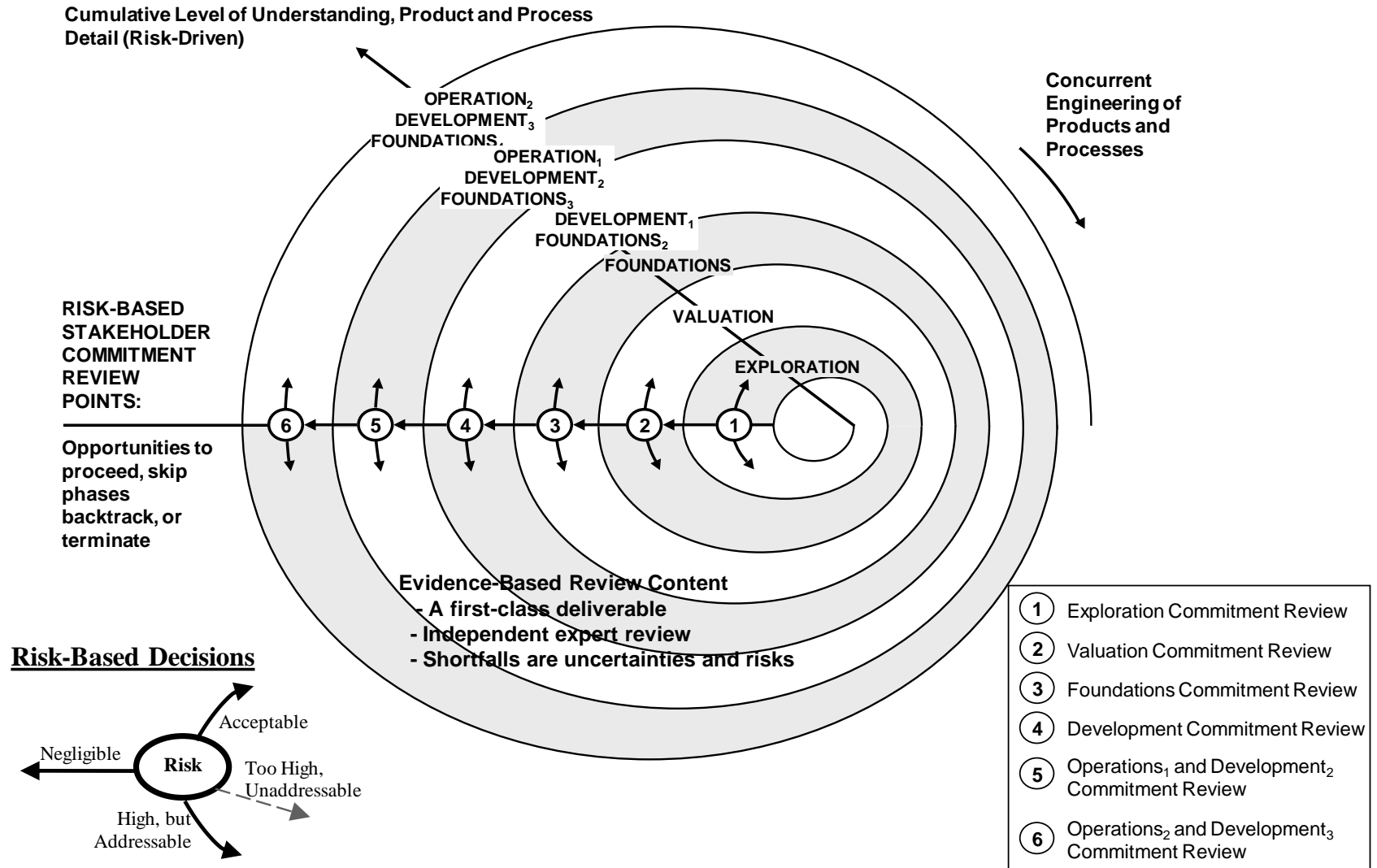
- Risk-driven framework for determining and evolving best-fit system life-cycle process
- Integrates the strengths of phased and risk-driven spiral process models
- Synthesizes together principles critical to successful system development
  - Stakeholder value-based guidance
  - Incremental commitment and accountability
  - Concurrent multidiscipline engineering
  - Evidence and risk-driven decisions



*Principles  
trump  
diagrams...*

Principles used by 60-80% of CrossTalk Top-5 projects, 2002-2005

# The Incremental Commitment Spiral Model



# Process Model Principles

Principles trump diagrams

1. Commitment and accountability
2. Success-critical stakeholder satisficing
3. Incremental growth of system definition and stakeholder commitment
- 4, 5. Concurrent, iterative system definition and development cycles

Cycles can be viewed as sequential concurrently-performed phases or spiral growth of system definition

6. Risk-based activity levels and anchor point commitment milestones

# Incremental Commitment in Gambling

- Total Commitment: Roulette
  - Put your chips on a number
    - E.g., a value of a key performance parameter
  - Wait and see if you win or lose
- Incremental Commitment: Poker, Blackjack
  - Put some chips in
  - See your cards, some of others' cards
  - Decide whether, how much to commit to proceed



# Specialized Process Models

# Component-based Development Model

- Consists of the following process steps
  - Available component-based products are researched and evaluated for the application domain in question
  - Component integration issues are considered
  - A software architecture is designed to accommodate the components
  - Components are integrated into the architecture
  - Comprehensive testing is conducted to ensure proper functionality
- Relies on a robust component library
- Capitalizes on software reuse, which leads to documented savings in project cost and time

# Formal Methods Model

- Encompasses a set of activities that leads to formal mathematical specification of computer software
- Enables a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation
- Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily through mathematical analysis
- Offers the promise of defect-free software
- Used often when building safety-critical systems

# Formal Methods Model

- Development of formal methods is currently quite time-consuming and expensive
- Because few software developers have the necessary background to apply formal methods, extensive training is required
- It is difficult to use the models as a communication mechanism for technically unsophisticated customers

# Aspect-Oriented Software Development

Provides a process and methodological approach for defining, specifying, designing, and constructing *aspects* such as

- user interfaces,
- security,
- memory management

that impact many parts of the system being developed

# Wisdom from Watt S Humphrey

- Some simple processes can improve quality when a software is being engineered by
  - An individual (Personal Software Process)
  - A small Team (Team Software Process)

# Personal Software Process (PSP)

- Recommends five framework activities:
  - Planning
  - High-level design
  - High-level design review
  - Development
  - Postmortem
- stresses the need for each software engineer to identify errors early and as important, to understand the types of errors

# Team Software Process (TSP)

- Each project is “launched” using a “script” that defines the tasks to be accomplished
- Teams are self-directed
- Measurement is encouraged
- Measures are analyzed with the intent of improving the team process



# Team Software Process (TSP)

- Recommends five framework activities for TSP:
  - Project Launch
  - High-level design
  - Implementation
  - Integration and Testing
  - Postmortem

# The Unified Process

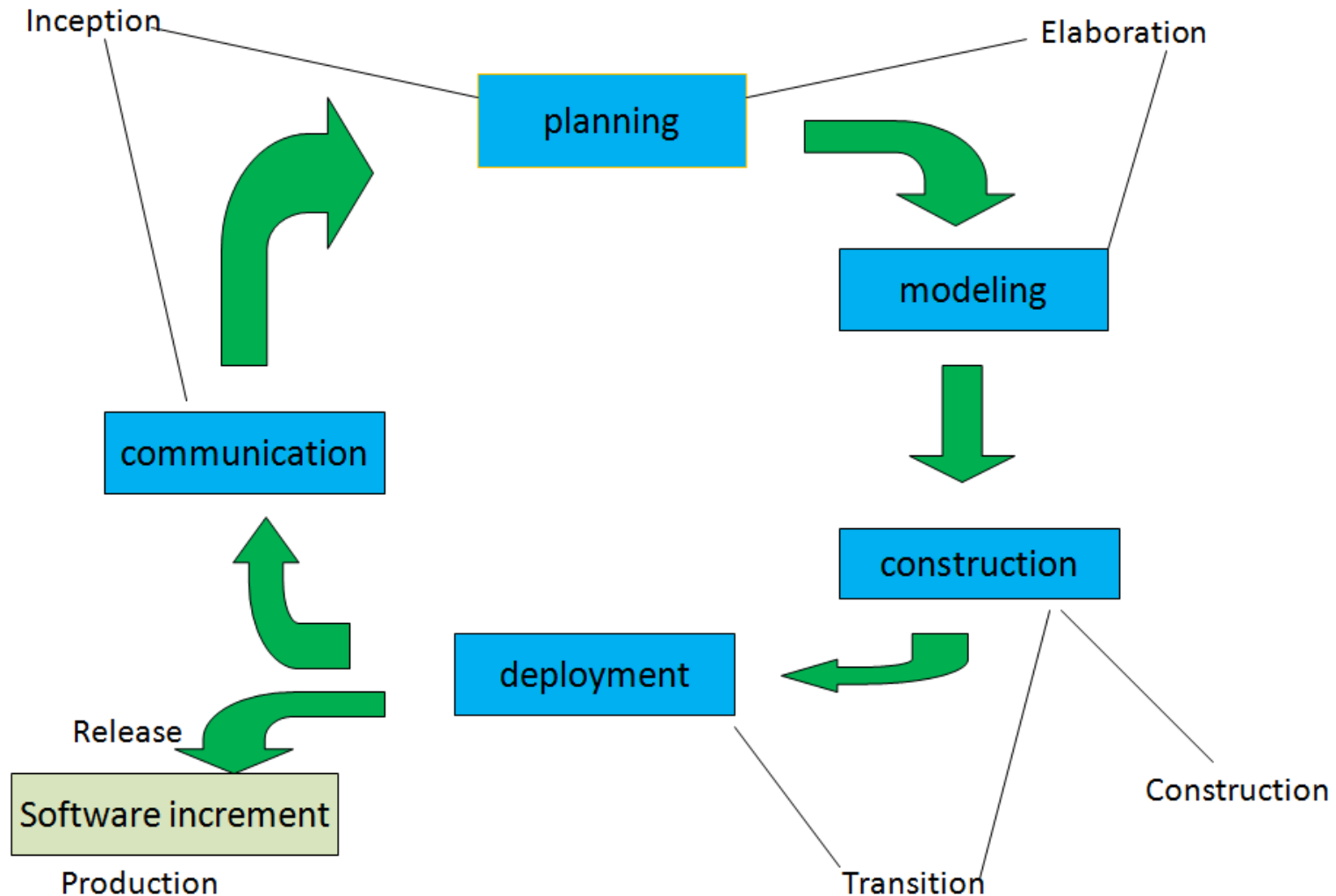
# Background

- Started during the late 1980's and early 1990s when object-oriented languages were gaining wide-spread use
- Many object-oriented analysis and design methods were proposed; three top authors were Grady Booch, Ivar Jacobson, and James Rumbaugh
- They eventually worked together on a unified method for modeling, called the Unified Modeling Language (UML)
  - UML is a robust notation for the modeling and development of object-oriented systems
  - UML became an industry standard in 1997
  - However, UML does not provide the process framework, only the necessary technology for object-oriented development

# Background (continued)

- Booch, Jacobson, and Rumbaugh later developed the unified process, which is a framework for object-oriented software engineering using UML
  - Draws on the best features and characteristics of conventional software process models
  - Emphasizes the important role of software architecture
  - Consists of a process flow that is iterative and incremental, thereby providing an evolutionary feel
- Consists of five phases: inception, elaboration, construction, transition, and production

# Phases of the Unified Process



# Inception Phase

- Encompasses both customer communication and planning activities of the generic process
- Business requirements for the software are identified
- A rough architecture for the system is proposed
- A plan is created for an incremental, iterative development
- Fundamental business requirements are described through preliminary use cases
  - A use case describes a sequence of actions that are performed by a user

# Elaboration Phase

- Encompasses both the planning and modeling activities of the generic process
- Refines and expands the preliminary use cases
- Expands the architectural representation to include five views
  - Use-case model
  - Analysis model
  - Design model
  - Implementation model
  - Deployment model
- Often results in an executable architectural baseline that represents a first cut executable system
- The baseline demonstrates the viability of the architecture but does not provide all features and functions required to use the system

# Construction Phase

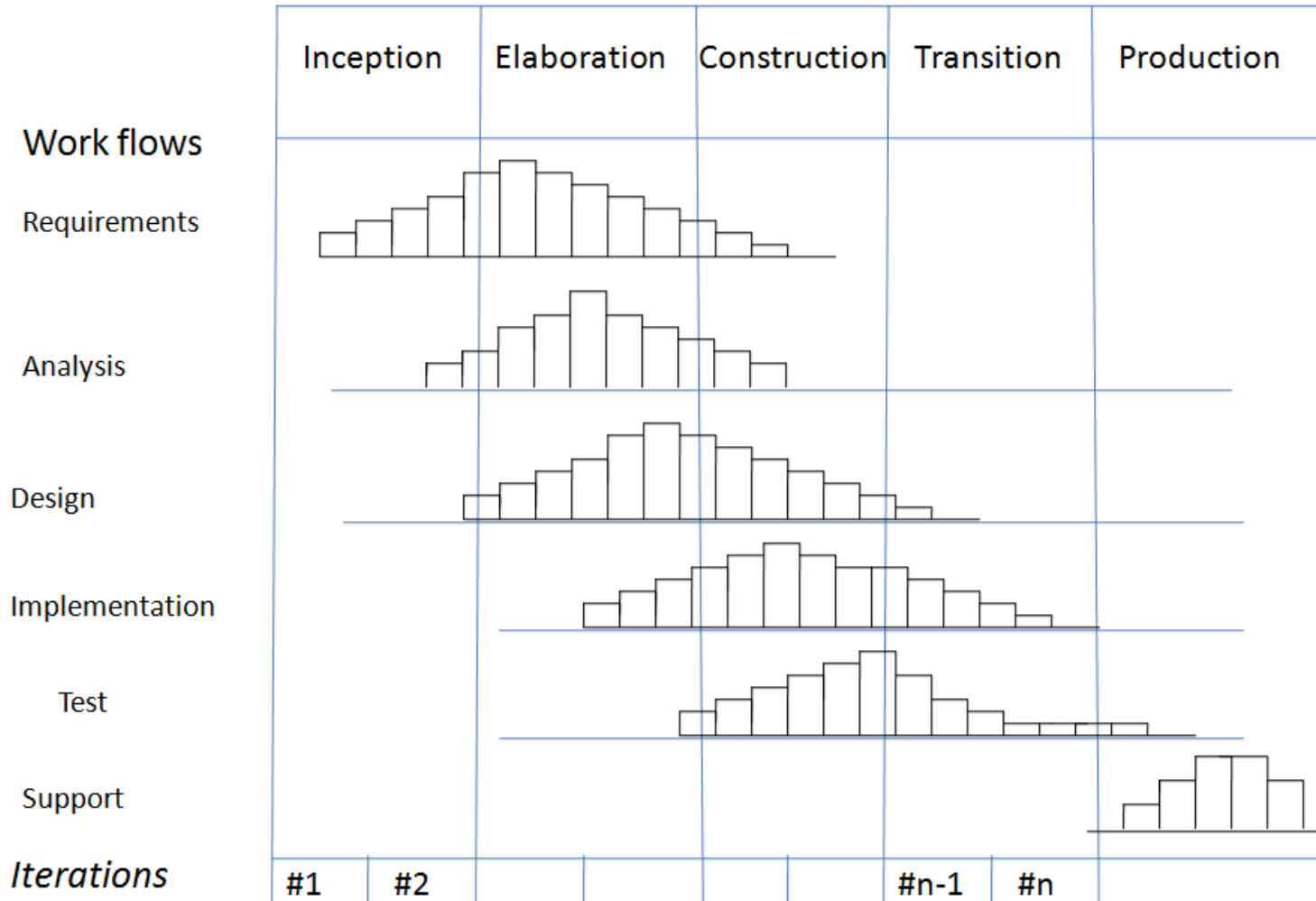
- Encompasses the construction activity of the generic process
- Uses the architectural model from the elaboration phase as input
- Develops or acquires the software components that make each use-case operational
- Analysis and design models from the previous phase are completed to reflect the final version of the increment
- Use cases are used to derive a set of acceptance tests that are executed prior to the next phase



# Transition Phase

- Encompasses the last part of the construction activity and the first part of the deployment activity of the generic process
- Software is given to end users for beta testing and user feedback reports on defects and necessary changes
- The software teams create necessary support documentation (user manuals, trouble-shooting guides, installation procedures)
- At the conclusion of this phase, the software increment becomes a usable software release

# UP Phases



# Production Phase

- Encompasses the last part of the deployment activity of the generic process
- On-going use of the software is monitored
- Support for the operating environment (infrastructure) is provided
- Defect reports and requests for changes are submitted and evaluated

# UP Work Products

## Inception phases

Vision document

Initial use case model  
Initial project glossary  
Initial business case  
Initial risk assessment  
Project plan, phase  
And iteration  
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 work flows  
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.

# Summary

- The roadmap to building high quality software products is software process.
- Software processes are adapted to meet the needs of software engineers and managers as they undertake the development of a software product.
- A software process provides a framework for managing activities that can very easily get out of control.
- Modern software processes must be agile, demanding only those activities, controls, and work products appropriate for team or product.
- Different types of projects require different software processes.
- The software engineer's work products (programs, documentation, data) are produced as consequences of the activities defined by the software process.
- The best indicators of how well a software process has worked are the quality, timeliness, and long-term viability of the resulting software product.