Chapter 3
Prescriptive Process Models

- Generic process framework (revisited)
- Traditional process models
- Specialized process models
- The unified process

(Source: Pressman, R. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 2005)

Generic Process Framework

- 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 under 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

Modeling: Software Requirements Analysis

- Helps software engineers to better understand the problem they will work to solve
- Encompasses the set of tasks that lead to an understanding of what the business impact of the software will be, what the customer wants, and how end-users will interact with the software
- Uses a combination of text and diagrams to depict requirements for data, function, and behavior
  - Provides a relatively easy way to understand and review requirements for correctness, completeness and consistency

Modeling: Software Design

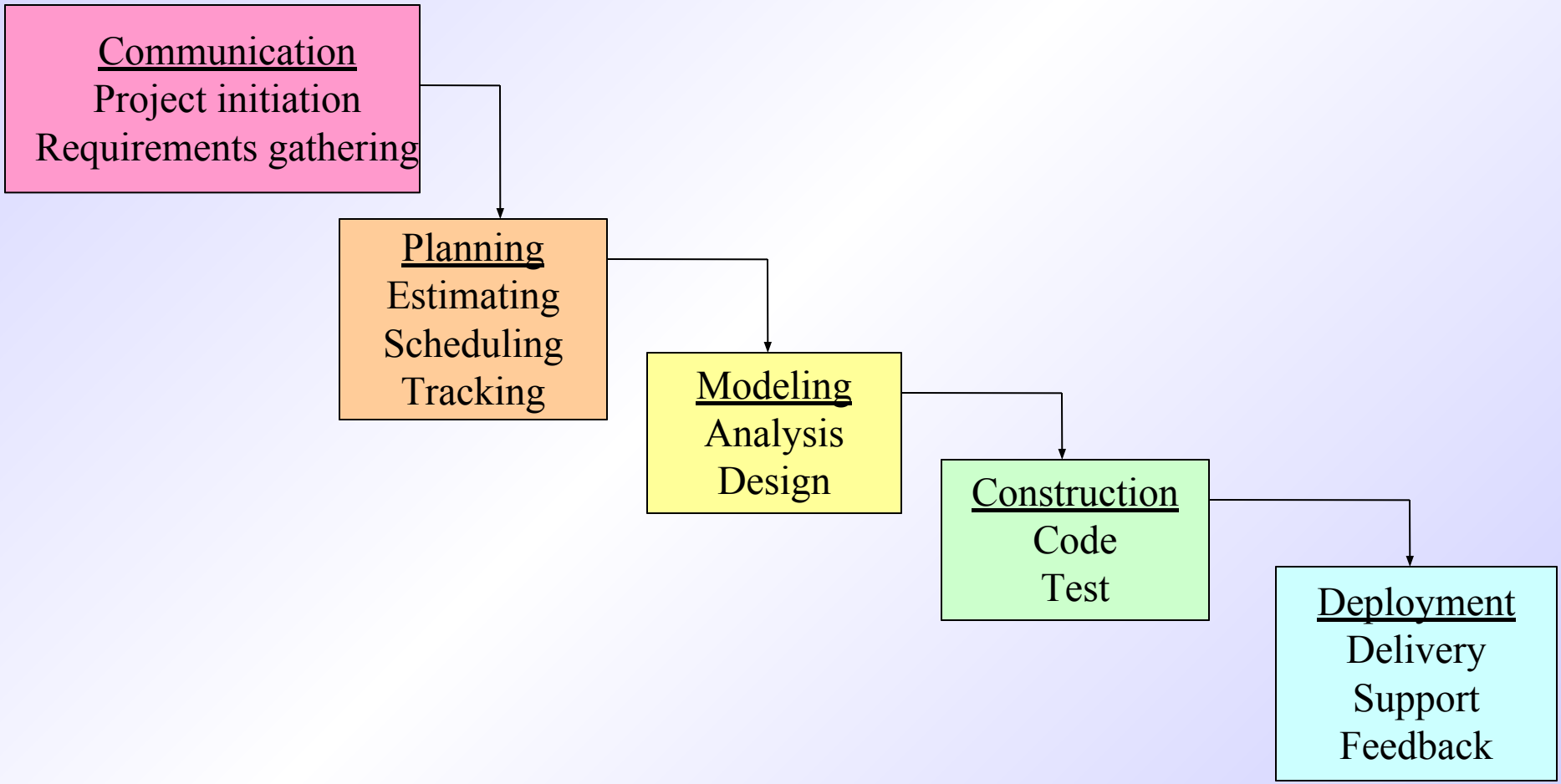
- Brings together customer requirements, business needs, and technical considerations to form the “blueprint” for a product
- Creates a model that that provides detail about software data structures, software architecture, interfaces, and components that are necessary to implement the system
- Architectural design
  - Represents the structure of data and program components that are required to build the software
  - Considers the architectural style, the structure and properties of components that constitute the system, and interrelationships that occur among all architectural components
- User Interface Design
  - Creates an effective communication medium between a human and a computer
  - Identifies interface objects and actions and then creates a screen layout that forms the basis for a user interface prototype
- Component-level Design
  - Defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each software component

Traditional Process Models

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
- Prescriptive process models
- 1 Water fall
- 2 incremental process model
- 1 Incremental model
- 2 RAD model
- 3 Evolutionary process models
- 1Prototyping model
- 2 spiral model
- 3 Test Driven Development

Waterfall Model (Diagram)



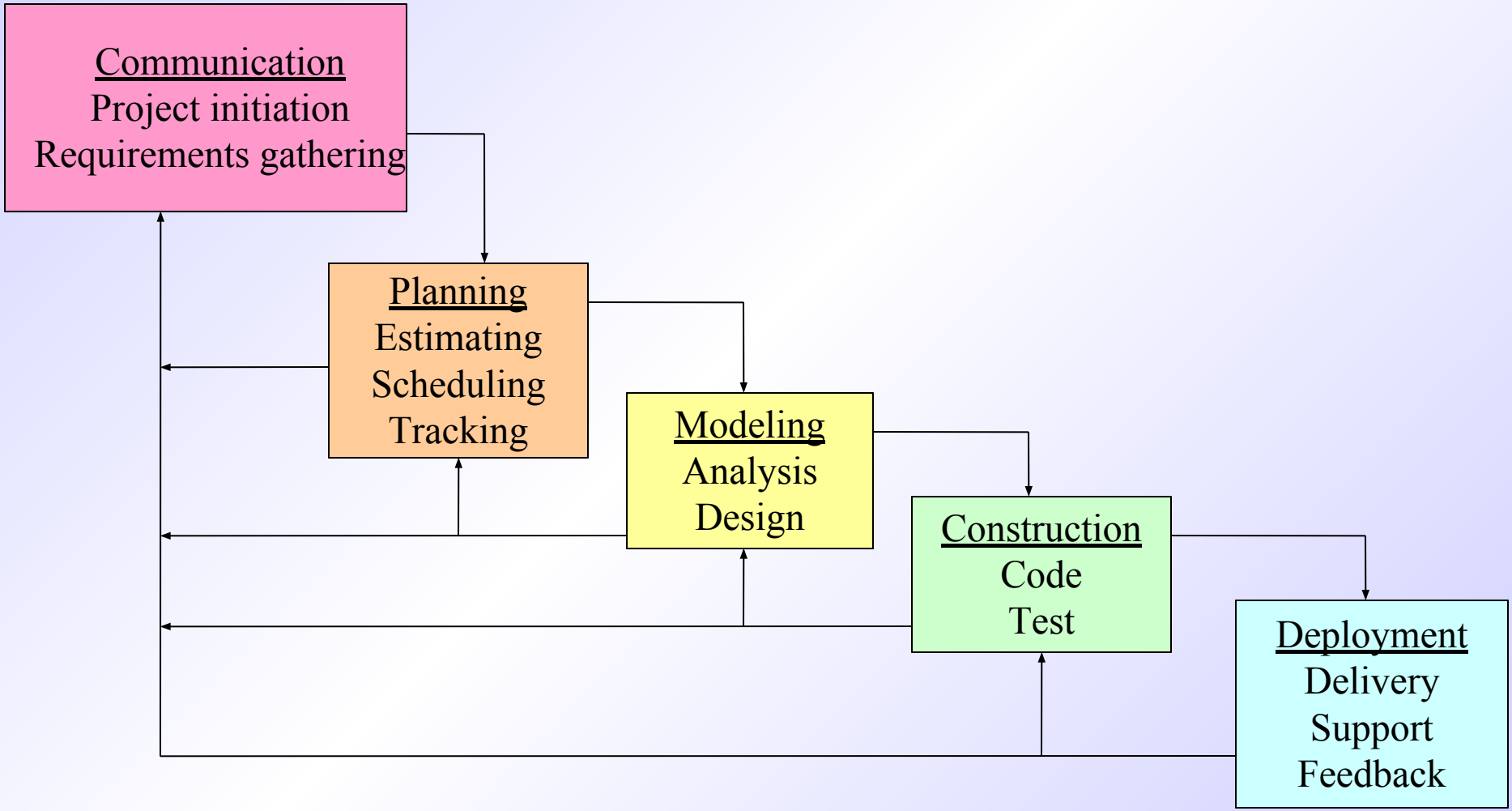
Waterfall Model (Description)

- Oldest software lifecycle model and best understood by upper management
- Used when requirements are well understood and risk is low
- Work flow is in a linear (i.e., sequential) fashion
- Used often with well-defined adaptations or enhancements to current software
- Simple to implement
- For small systems

Waterfall Model (Problems)

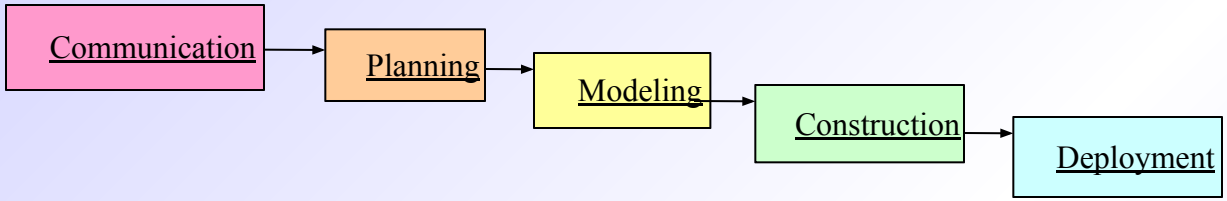
- Doesn't support iteration, so changes can cause confusion
- Difficult for customers to state all requirements explicitly and up front
- Requires customer patience because a working version of the program doesn't occur until the final phase
- Problems can be somewhat alleviated in the model through the addition of feedback loops (see the next slide)

Waterfall Model with Feedback (Diagram)

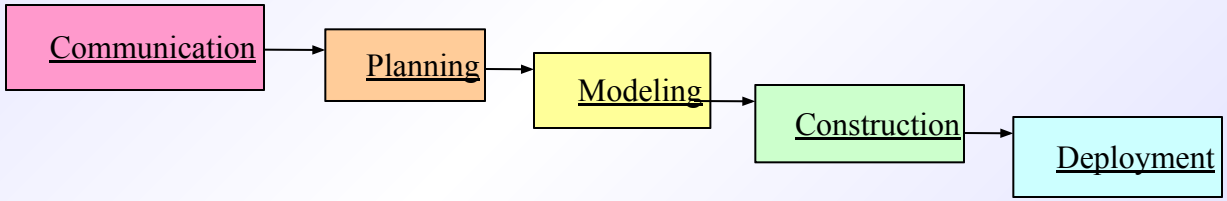


Incremental Model (Diagram)

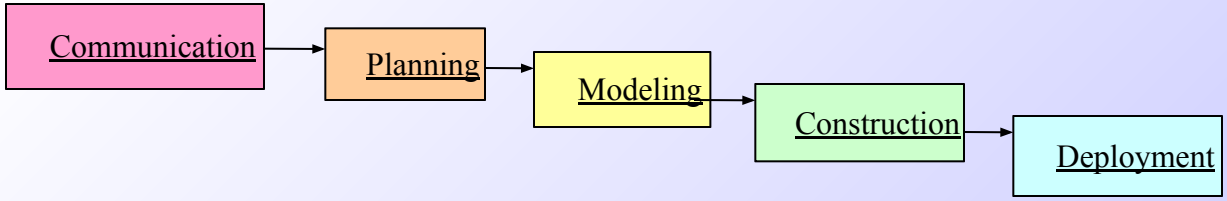
Increment #1

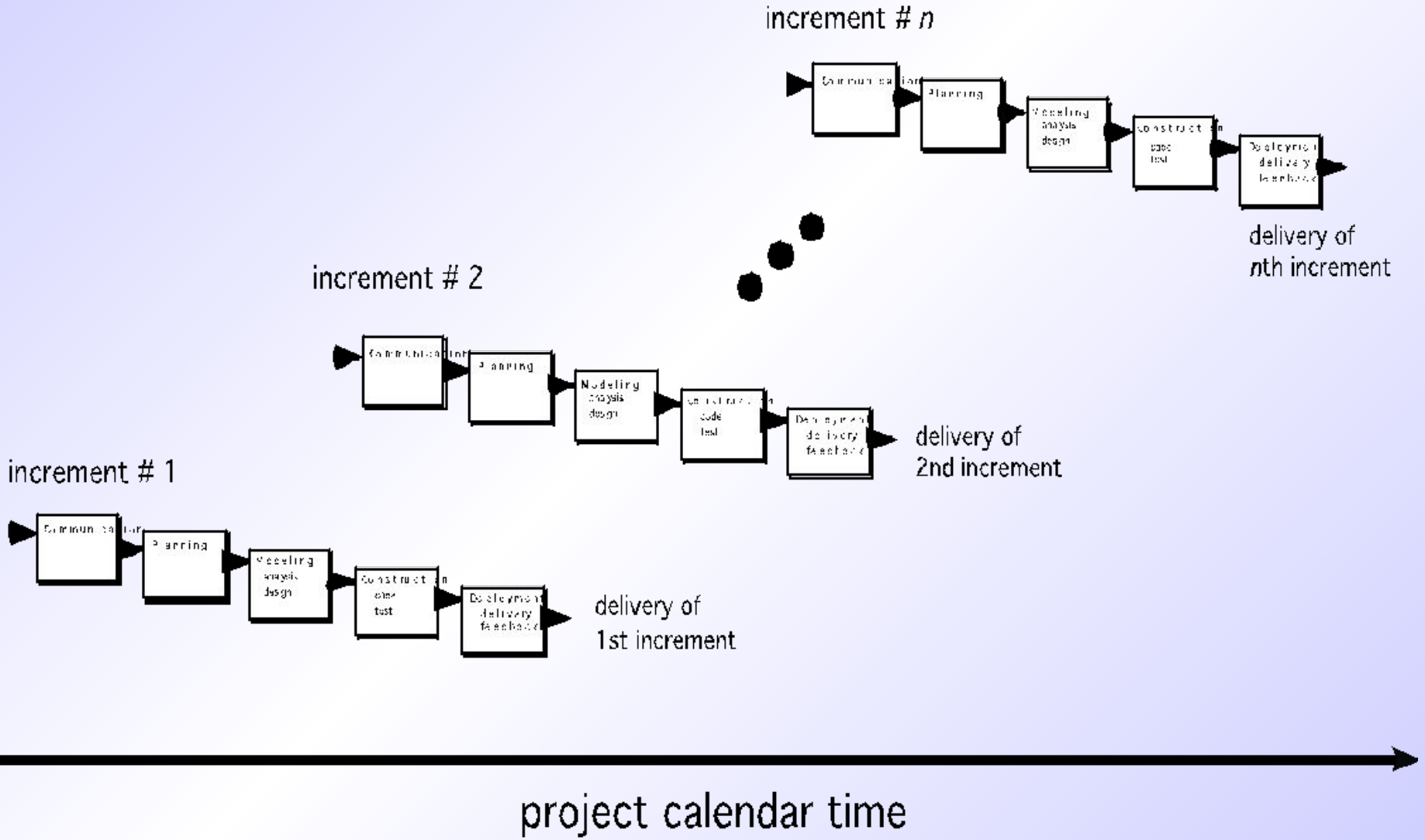


Increment #2



Increment #3

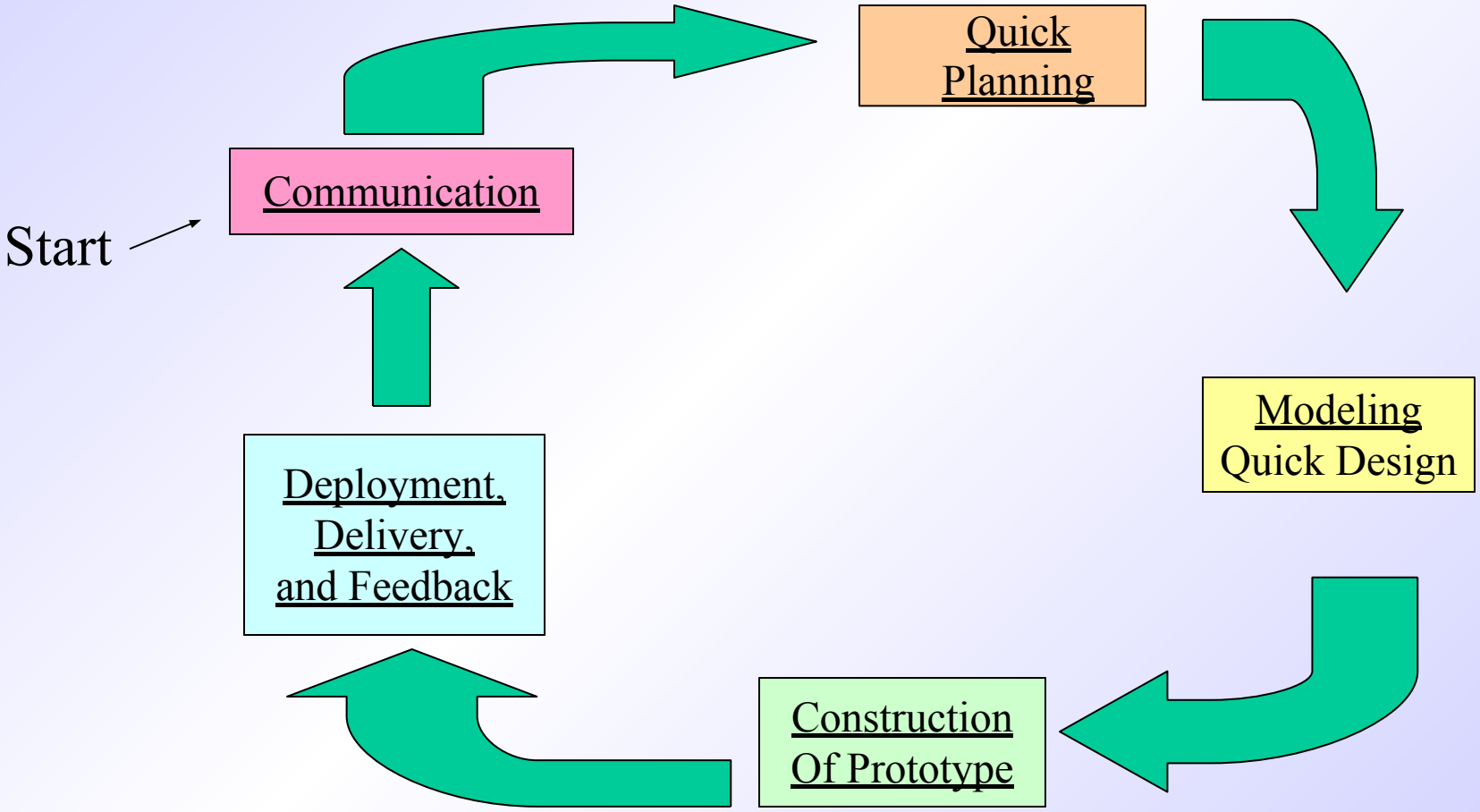



increment # n
Planning
Modeling
analysis
design
Construction
code
test
Deployment
delivery
feedback
delivery of
nth increment
increment # 2
Planning
Modeling
analysis
design
Construction
code
test
Deployment
delivery
feedback
delivery of
2nd increment
increment # 1
Planning
Modeling
analysis
design
Construction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

Incremental Model (Description)

- Used when requirements are well understood
- Multiple independent deliveries are identified
- Work flow is in a linear (i.e., sequential) fashion within an increment and is staggered between increments
- Iterative in nature; focuses on an operational product with each increment
- Provides a needed set of functionality sooner while delivering optional components later
- Useful also when staffing is too short for a full-scale development

Prototyping Model (Diagram)



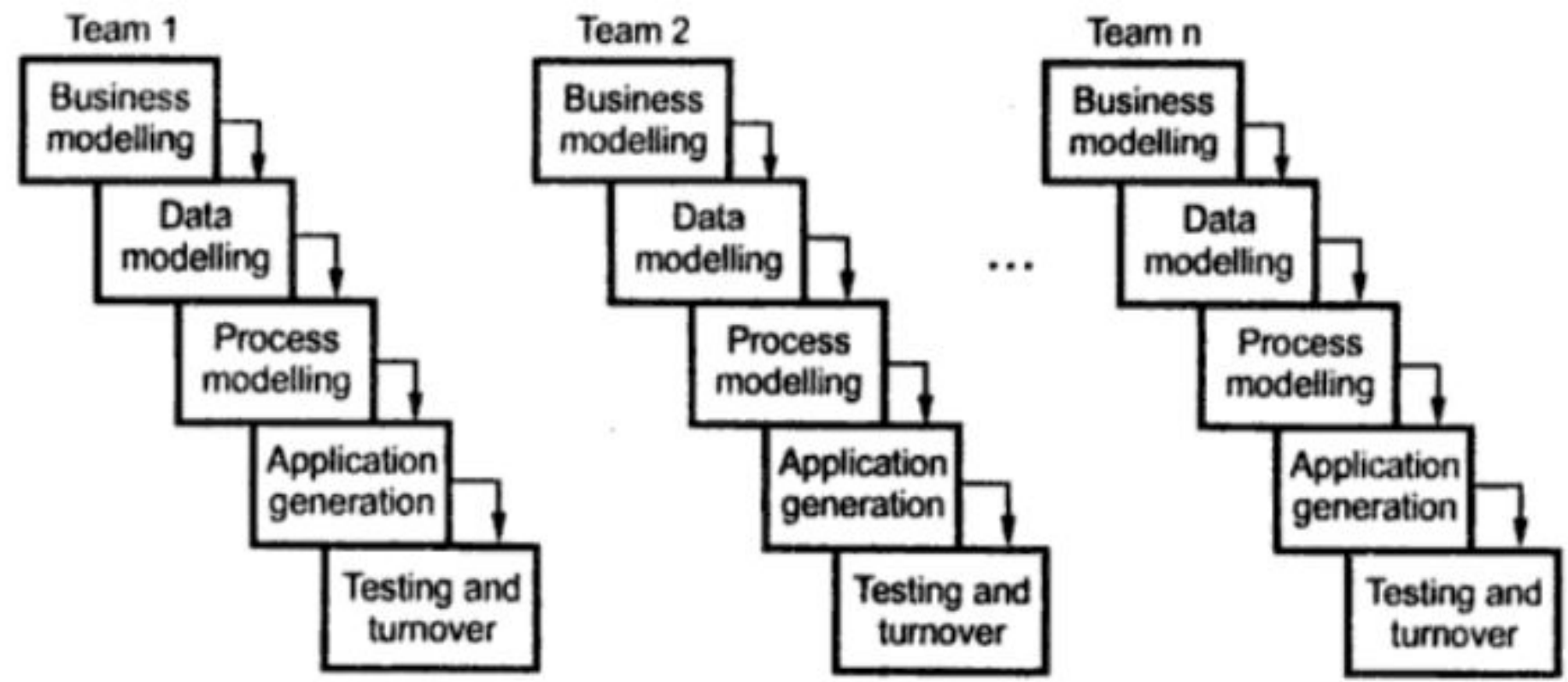
Prototyping Model (Description)

- 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)
- Lesson learned
  - 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

RAD Model(60-90 days)



Rapid Application Development (RAD) Model

The rapid application development model is type of incremental software process model in which there is extremely short development cycle.

This model is similar to waterfall model which achieves the high speed development using **component based construction**.

To develop the fully functional system within short time period using this model it is necessary to understand the requirements fully and to have a restricted project scope.

1) **Business modeling** - In business modeling, the information flow is modeled into various business functions. These business functions collect following information.

- Information that drives the business process.
- The type of information being generated.
- The generator of information.
- The information flow.
- The processor of information.

2) **Data modeling** - In this phase the information obtained in business model is classified into data objects. The characteristics of data objects (attributes) are identified. The relationship among various data objects is defined.

3) **Process modeling** - In this phase the data objects are transformed into processes. These processes are to extract the information from data objects and are responsible for implementing business functions.

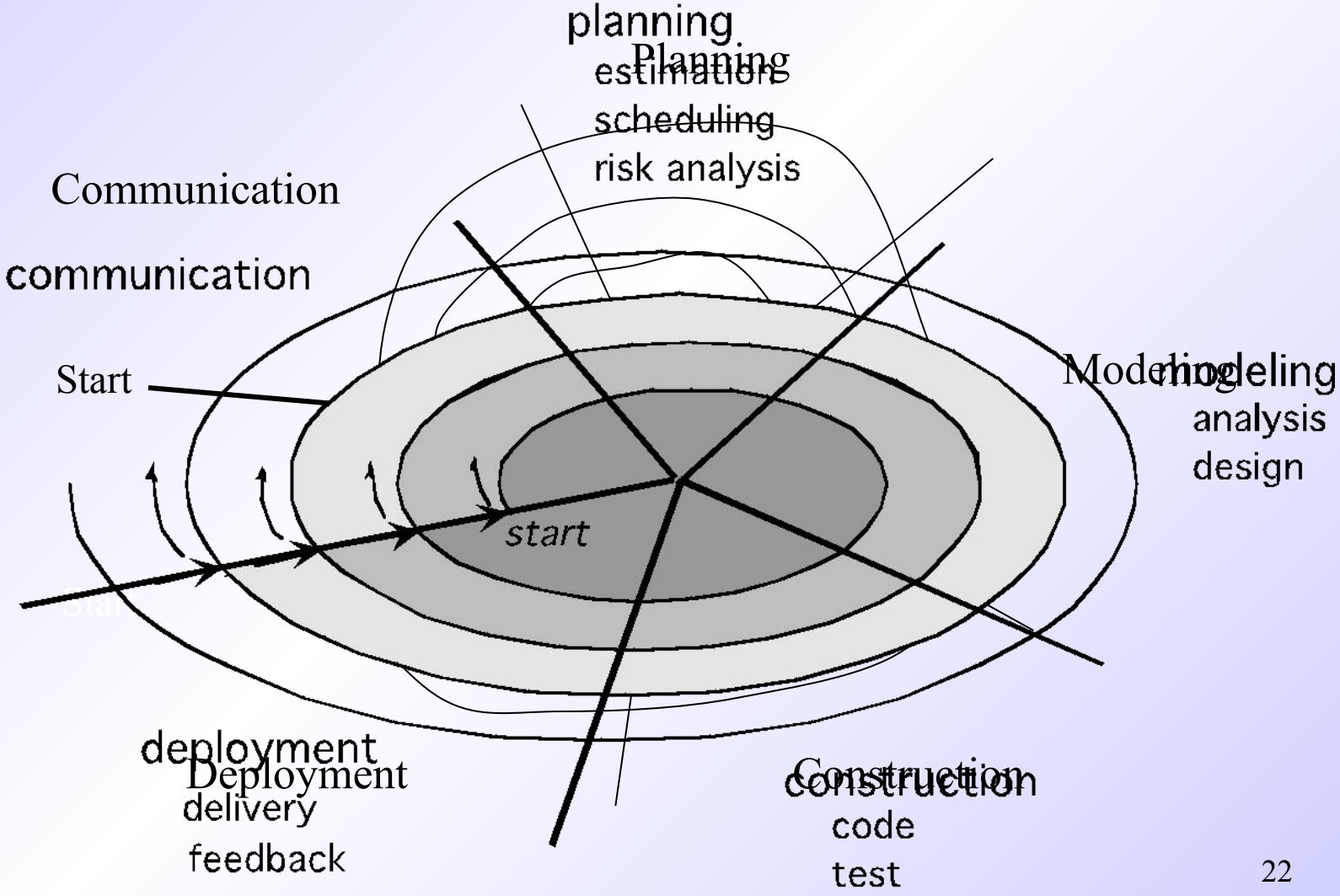
4) **Application generation** - For creating software various automation tools can be used. RAD also makes use of reusable components or creates reusable components to have rapid development of software.

5) **Testing and turnover** - As RAD uses reusable components the testing efforts are reduced. But if new components are added in software development process then such components need to be tested. It is equally important to test all the interfaces.

disadvantages

- Multiple teams
- Heavily committed devlepor
- Heavy resources
- Appropriate modularization else failing of project
- Difficult to adopt new technology

Spiral Model (Diagram)



Spiral Model (Description)

- Invented 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

i) **Customer communication** - In this region it is suggested to establish customer communication.

ii) **Planning** - All planning activities are carried out in order to define resources time line and other project related activities.

iii) **Risk analysis** - The tasks required to calculate technical and management risks are carried out.

iv) **Engineering** - In this task region, tasks required to build one or more representations of applications are carried out.

v) **Construct and release** - All the necessary tasks required to construct, test, install the application are conducted. Some tasks that are required to provide user support are also carried out in this task region.

vi) **Customer evaluation** - Customer's feedback is obtained and based on customer evaluation required tasks are performed and implemented at installation stage.

- In each region, number of **work tasks** are carried out depending upon the characterisics of project. For a small project relatively small number of work tasks are adopted but for a complex project large number of work tasks can be carried out.
- In spiral model, the software engineering team **moves around the spiral** in a clockwise direction beginning at the core.

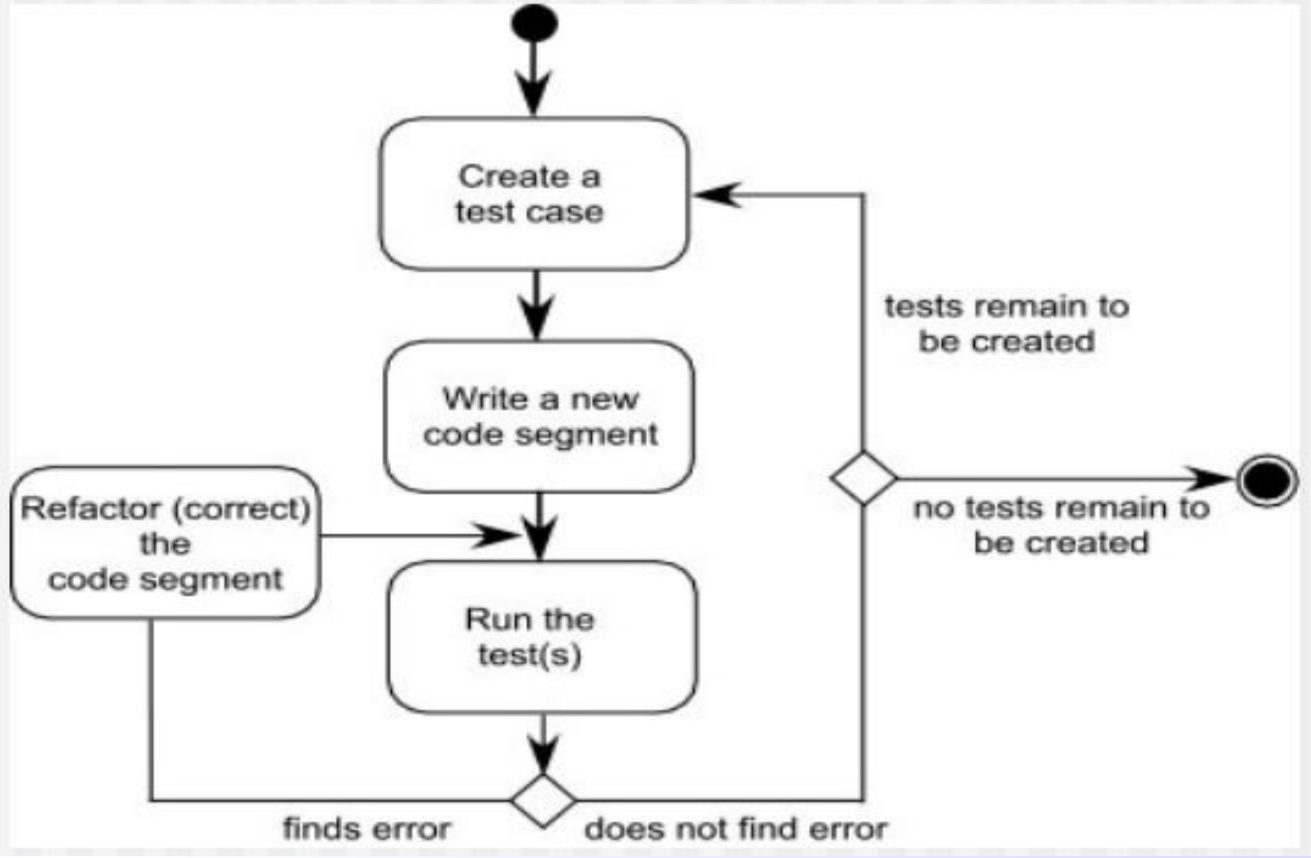
Advantages of spiral model

1. Requirement changes can be made at every stage.
2. Risks can be identified and rectified before they get problematic.

Drawbacks of spiral model

- It is based on customer communication. If the communication is not proper then the software product that gets developed will not be up to the mark.
- It demands considerable risk assessment. If the risk assessment is done properly then only the successful product can be obtained.

Test driven development


Create a test case
Write a new code segment
Refactor (correct) the code segment
Run the test(s)
tests remain to be created
no tests remain to be created
finds error
does not find error

General Weaknesses of Evolutionary Process Models

1) Prototyping poses 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

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 (Description)

- 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 (Challenges)

- 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

The Unified Process

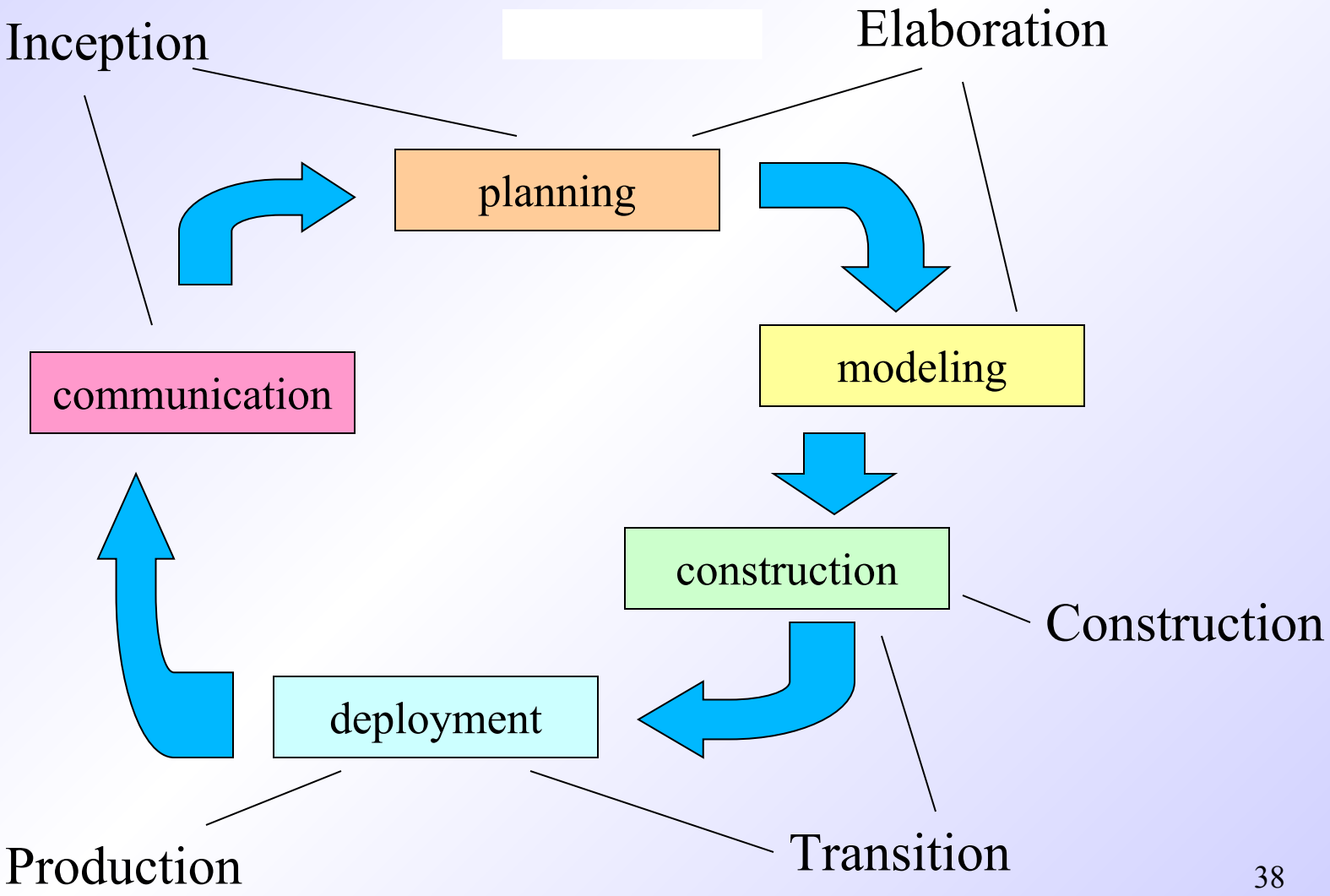
Background

- Birthed 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, 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

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

Unified Process Work Products

- Work products are produced in each of the first four phases of the unified process
- In this course, we will concentrate on the analysis model and the design model work products
- Analysis model includes
  - Scenario-based model, class-based model, and behavioral model
- Design model includes
  - Component-level design, interface design, architectural design, and data/class design