Chapter 5
Software Engineering Practice

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

Software Engineering Practice

- Software engineering practices
- Communication practices
- Planning practices
- Analysis modeling practices
- Design modeling practices
- Construction practices
- Deployment practices

Software Engineering Practice

- Consists of a **collection of concepts, principles, methods, and tools** that a software engineer calls upon on a daily basis
- Equips managers to **manage** software projects and software engineers to **build** computer programs
- Provides necessary **technical and management** how-to's in getting the job done
- **Transforms** a haphazard (**random**) unfocused approach into something that is **more organized**, **more effective**, and more **likely to achieve success**

The Essence of Practice

1) **Understand the problem** (communication and analysis)
   - Who has a stake (prize) in the solution to the problem?
   - What are the unknowns (data, function, behavior)?
   - Can the problem be compartmentalized?
   - Can the problem be represented graphically?
2) **Plan a solution** (planning, modeling and software design)
   - Have you seen similar problems like this before?
   - Has a similar problem been solved and is the solution reusable?
   - Can sub problems be defined and are solutions available for the sub problems?

(more on next slide)

The Essence of Practice (continued)

3) **Carry out the plan** (construction; code generation)

- Does the solution conform to the plan? Is the source code traceable back to the design?
- Is each component of the solution correct? Has the design and code been reviewed?

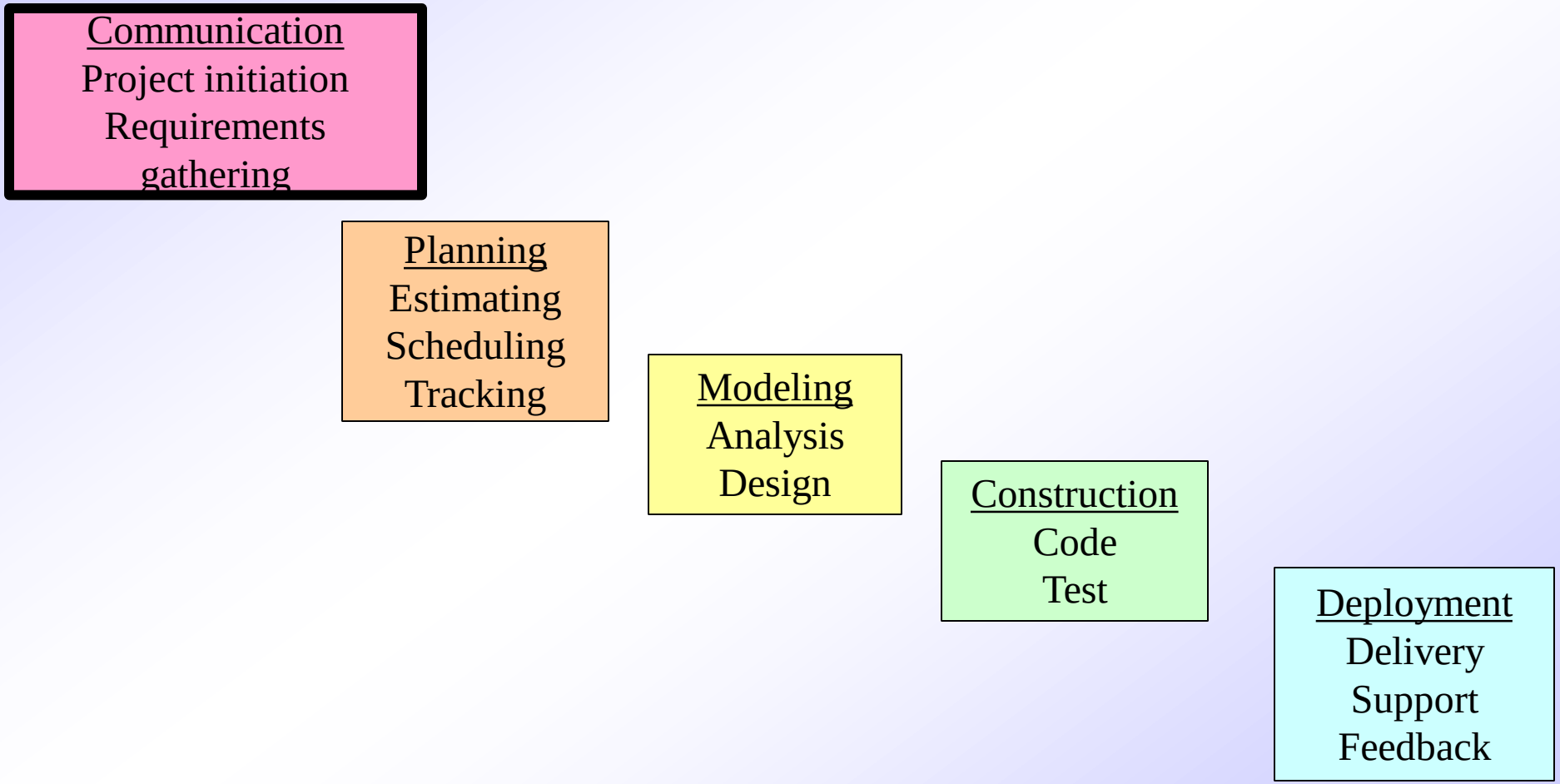
4) **Examine the results for accuracy** (testing and quality assurance)

- Is it possible to test each component of the solution?
- Does the solution produce results that conform to the data, function, and behavior that are required?

Seven Core Principles for Software Engineering

1) Remember **the reason** that the software exists
   - The software should provide **value to its users and satisfy the requirements**
2) **Keep it simple, stupid**
   - All design and implementation should be as simple as possible
3) Maintain the **vision of the project**
   - A clear vision is essential to the project's success
4) **Others will consume** what you produce
   - Always specify, design, and implement knowing that someone else will later have to understand and modify what you did
5) **Be open to the future**
   - Never design yourself into a corner; **build software that can be easily changed and adapted**
6) **Plan ahead for software reuse**
   - Reuse of software **reduces the long-term cost** and **increases the value of the program and the reusable components**
7) **Think, then act**
   - Placing clear, complete thought before action will almost always produce better results

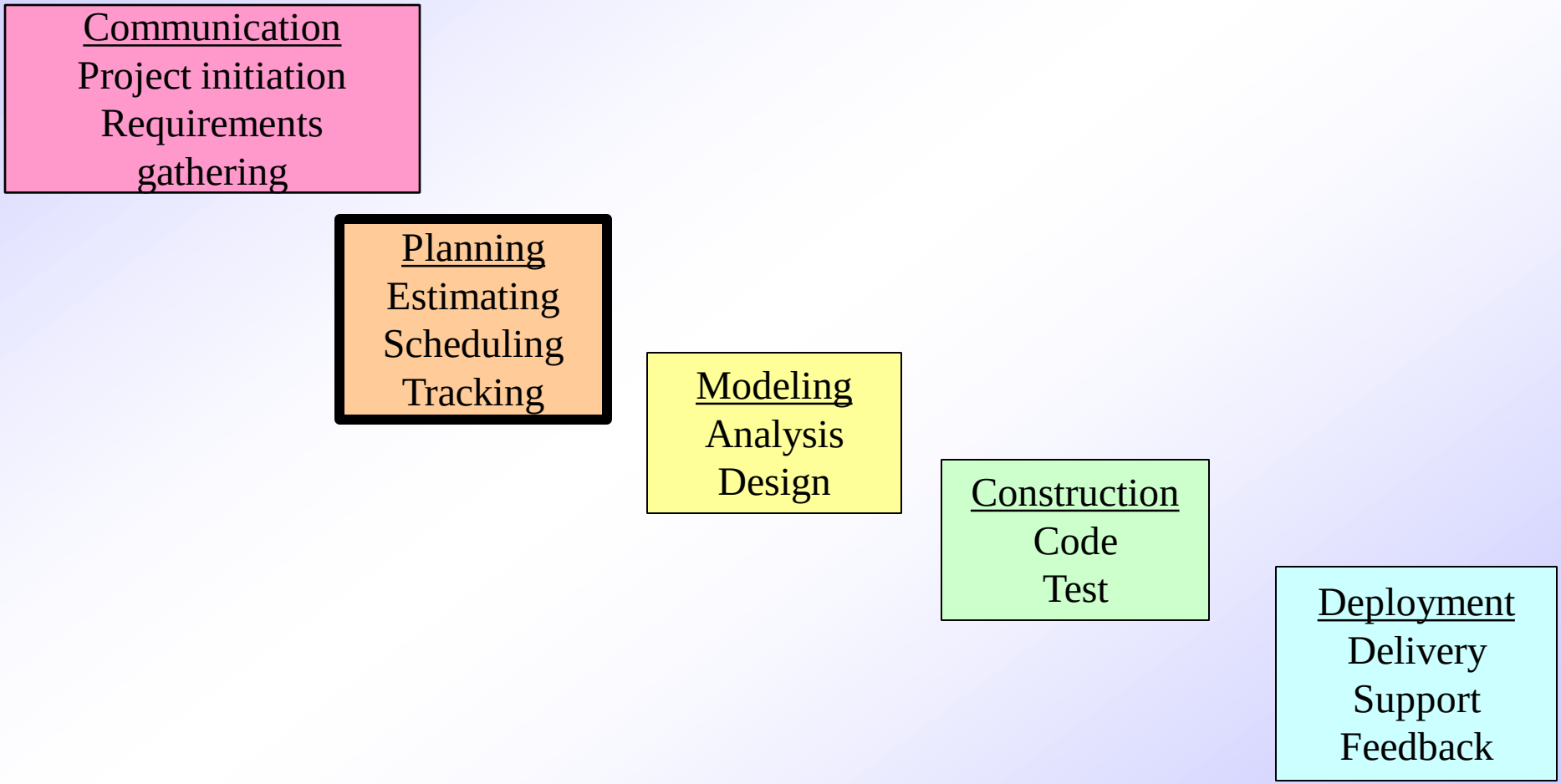
Communication Practices
(Requirements Elicitation)



Communication Principles

1) Listen to the speaker and concentrate on what is being said
2) Prepare yourself before you meet by researching and understanding the problem
3) Someone should facilitate the meeting and have an agenda
4) Face-to-face communication is best, but also have a document or presentation to focus the discussion
5) Take notes and document decisions
6) Strive for collaboration and agreement
7) Stay focused on a topic; modularize your discussion
8) If something is unclear, draw a picture
9) Move on to the next topic
    a) after you agree to something,
    b) if you cannot agree to something, or
    c) if a feature or function is unclear and cannot be clarified at the moment
1) Negotiation is not a contest or a game; it works best when both parties win

Planning Practices
(Defining a Road Map)



Planning Principles

1) Understand the scope of the project
2) Involve the customer in the planning activity
3) Recognize that planning is iterative; things will change
4) Estimate based only on what you know
5) Consider risk as you define the plan
6) Be realistic on how much can be done each day by each person and how well
7) Adjust granularity as you define the plan
8) Define how you intend to ensure quality
9) Describe how you intend to accommodate change
10) Track the plan frequently and make adjustments as required

Barry Boehm's W^5HH Principle

- Why is the system being developed?
- What will be done?
- When will it be accomplished?
- Who is responsible for each function?
- Where are they organizationally located?
- How will the job be done technically and managerially?
- How much of each resource is needed?

The answers to these questions lead to a definition of key project characteristics and the resultant project plan

W^{5}HH Principle

Why is the System being Developed?

The answer to this question enables all parties to asses the validity of business reasons for the

Software work. (To justify the expenditure of People, Time, and Money for the Business purpose)

What will be done?

The answer establishes the task set that will be required for the project

When will it be done?

The answer helps the Project Team to establish a Project Schedule by identifying when Project Tasks

are to be conducted and when Milestones are to be reached.

Who is responsible for a function?

The answer helps to define the roles and responsibilities of each member within the Project team.

Where are they organizationally located?

Not all roles and responsibilities reside within the Project team itself. Users, Customers and other stakeholders may have responsibilities as well.

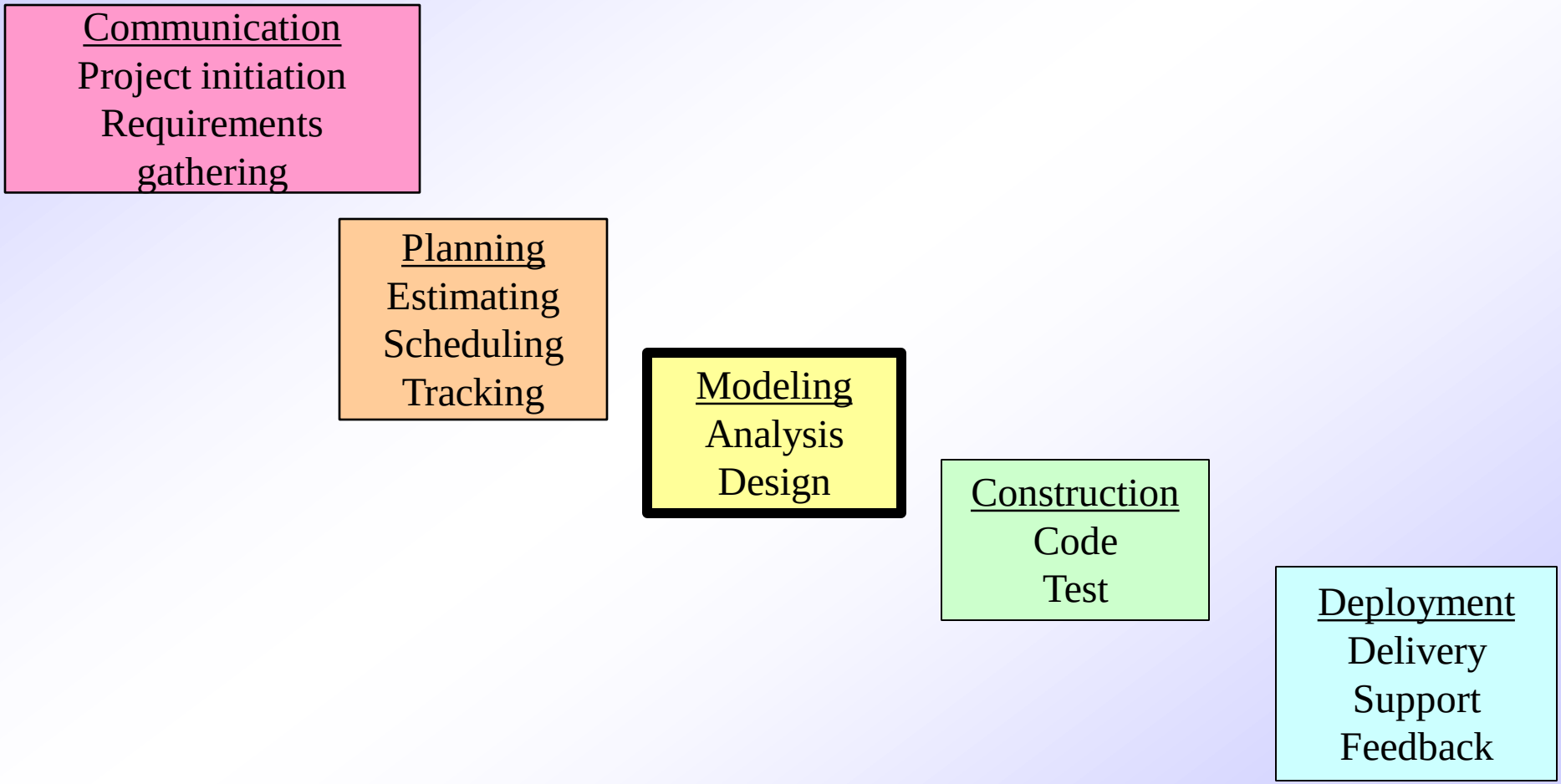
How will the job be done technically and Managerially?

Once Product Scope is established a Management and Technical strategy for Project must be defined.

How much of each resource is needed?

The answer is derived by developing estimates based on answers of the above questions

Modeling Practices
(Analysis and Design)



Analysis Modeling Principles

1) The **information domain** of a problem (the data that flows in and out of a system) must be represented and understood
2) The **functions** that the software performs must be defined
3) The **behavior** of the software (as a consequence of external events) must be represented
4) The models that depict information, function, and behavior **must be partitioned in a manner that uncovers detail in a layered** (or hierarchical) fashion
5) The analysis task should move from **essential information toward implementation detail**

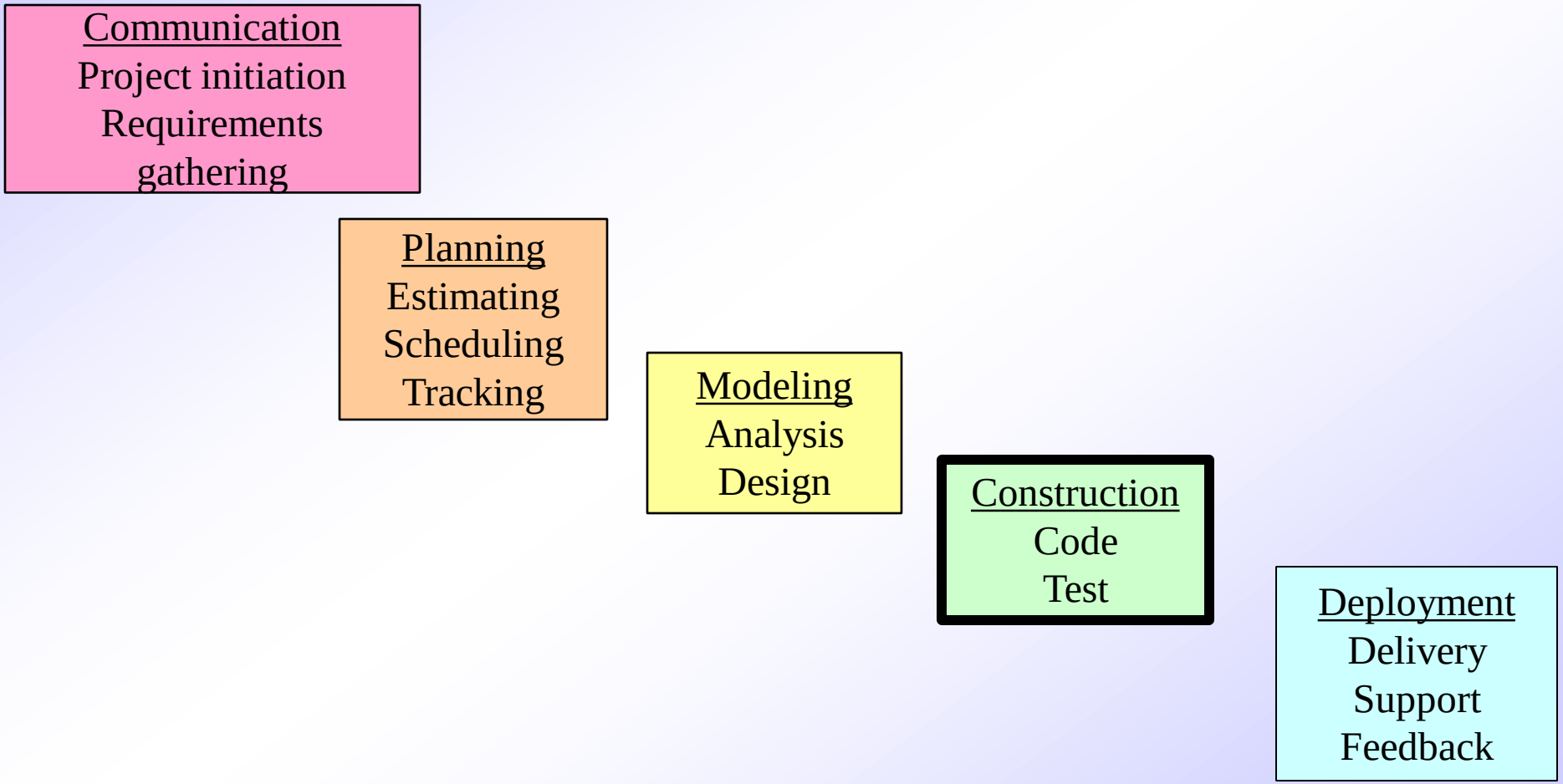
Design Modeling Principles

1) The design should be **traceable to the analysis model**
2) Always consider the **software architecture of the system to be built**
3) **Design of data** is as important as design of processing functions
4) **Interfaces** (both internal and external) must be designed with care
5) **User interface design** should be tuned to the needs of the end-user and should stress ease of use
6) **Component-level design** should be functionally independent (high cohesion)
7) Components should be **loosely coupled** to one another and to the external environment
8) Design representations (**models**) should be easily understandable
9) The design should be developed iteratively; with each iteration, the designer should strive for greater simplicity

External quality factors: those properties that can be readily observed

Internal quality factors: those properties that lead to a high-quality design from a technical perspective

Construction Practices



Coding Principles
(Preparation before coding)

1) Understand the problem you are trying to solve
2) Understand basic design principles and concepts
3) Pick a programming language that meets the needs of the software to be built and the environment in which it will operate
4) Select a programming environment that provides tools that will make your work easier
5) Create a set of unit tests that will be applied once the component you code is completed

Coding Principles
(As you begin coding)

1) Constrain your algorithms by following structured programming practices
2) Select data structures that will meet the needs of the design
3) Understand the software architecture and create interfaces that are consistent with it
4) Keep conditional logic as simple as possible
5) Create nested loops in a way that makes them easily testable
6) Select meaningful variable names and follow other local coding standards
7) Write code that is self-documenting
8) Create a visual layout (e.g., indentation and blank lines) that aids code understanding

Coding Principles
(After completing the first round of code)

1) Conduct a code walkthrough
2) Perform unit tests and correct errors you have uncovered
3) Refactor the code

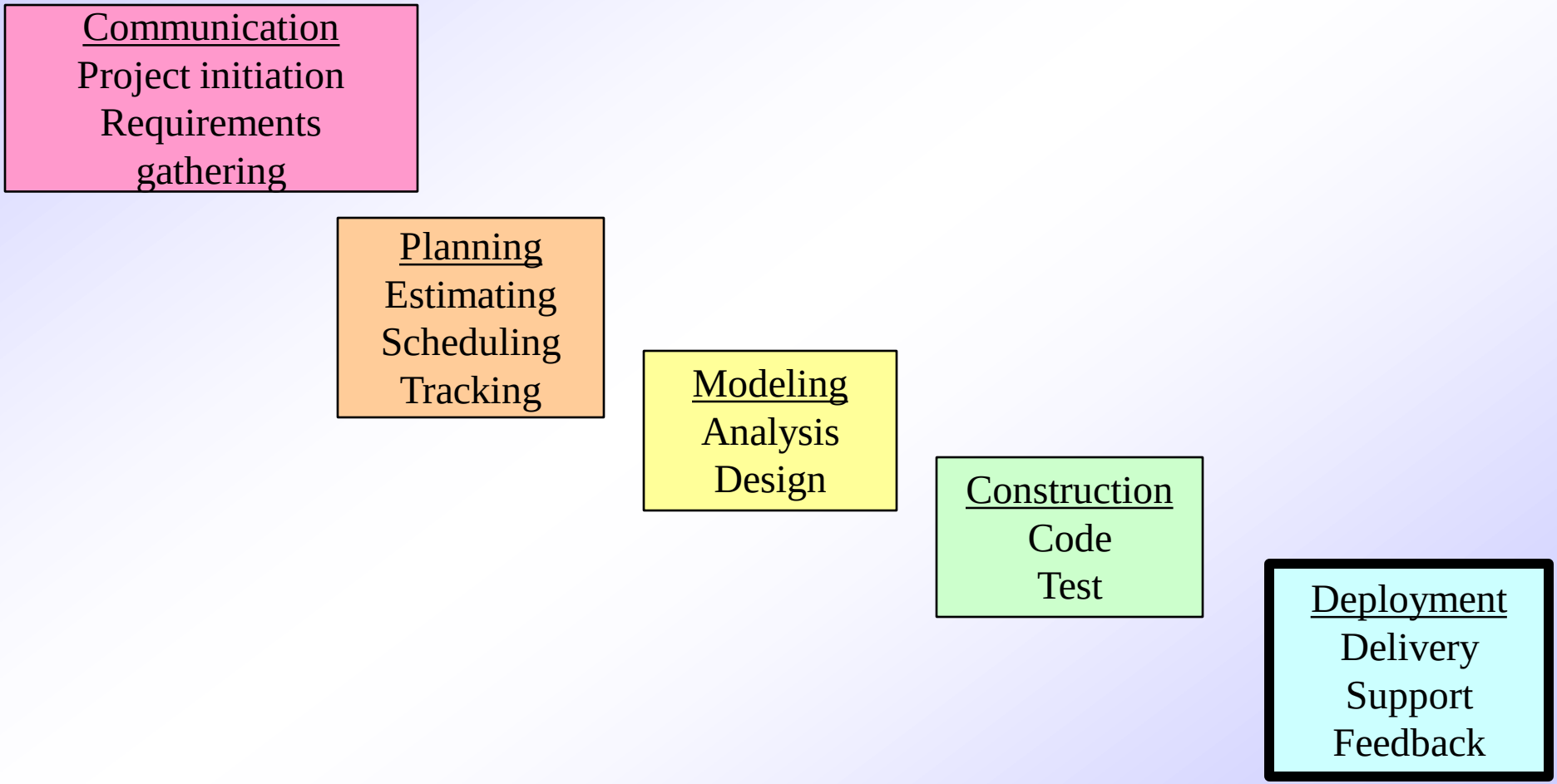
Testing Principles

1) All tests should be traceable to the software requirements
2) Tests should be planned long before testing begins
3) The Pareto principle applies to software testing
   - 80% of the uncovered errors are in 20% of the code
4) Testing should begin "in the small" and progress toward testing "in the large"
   - Unit testing --> integration testing --> validation testing --> system testing
5) Exhaustive testing is not possible

Test Objectives

1) Testing is a process of executing a program with the intent of finding an error
2) A good test case is one that has a high probability of finding an as-yet undiscovered error
3) A successful test is one that uncovers an as-yet undiscovered error

Deployment Practices



Deployment Principles

1) Customer expectations for the software must be managed
   - Be careful not to promise too much or to mislead the user
2) A complete delivery package should be assembled and tested
3) A support regime must be established before the software is delivered
4) Appropriate instructional materials must be provided to end users
5) Buggy software should be fixed first, delivered later