Review
On Software Engineering

Contents

- Software and Software Engineering
- Process and Process model： a road map guiding development
- Requirement Engineering: Understand Problem
- Requirement modeling： Conventional and Object Oriented
- Software Design: principle , Archtecture,Component-level
- User Interface design
- Software testing: strategy and technology

What is Software？

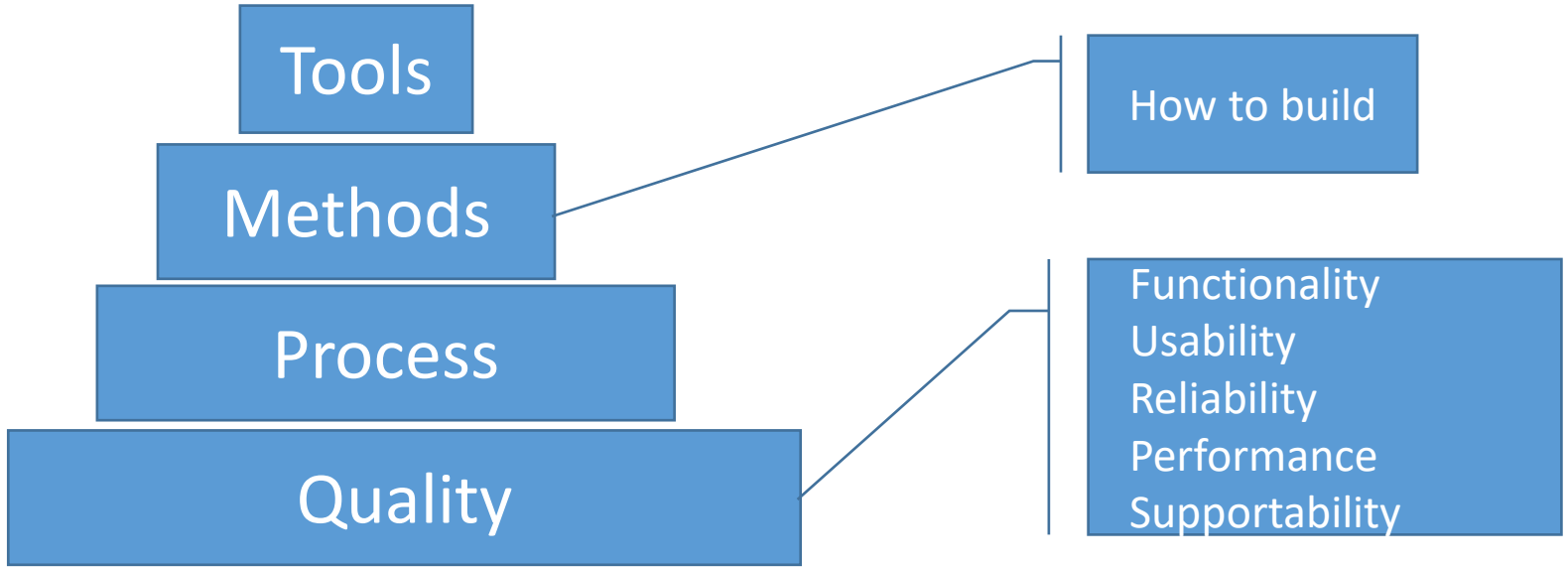
- Instructions +Data structure + Descriptive document
- What the difference between software and hardware?
- Why does software need Change or Evolved?

What is Software engineering?

- Informal Definition:
- IEEE Formal Definition:

> *(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.*
> *(2) The study of approaches as in (1)*

SE is a layered Technology



Software Process： A collection of *activities*, *actions*, and *tasks* that a performed when some work product is to be created

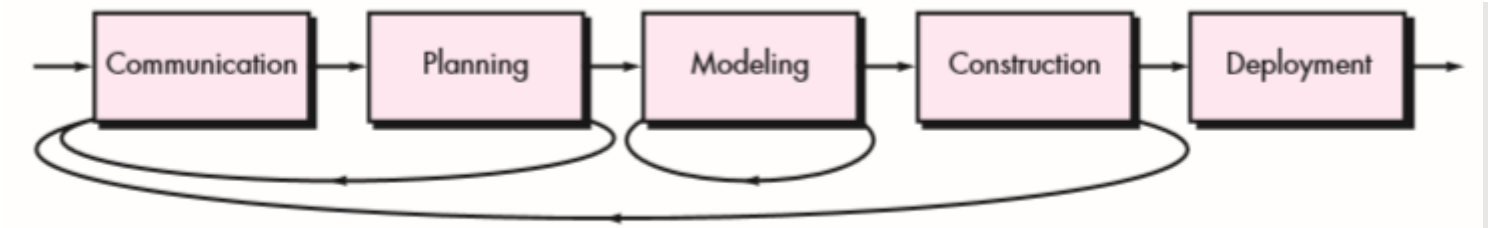
Generic Framework Activity

- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
- Construction
  - Code generation
  - Testing
- Deployment

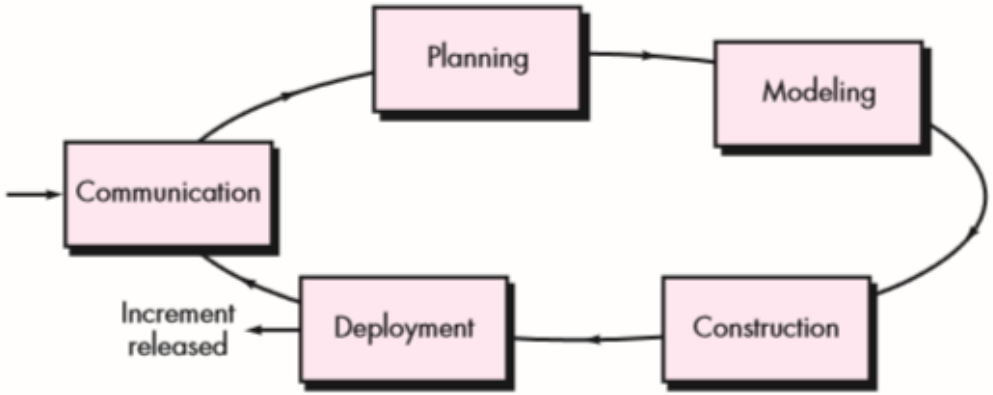
Umbrella Activities

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

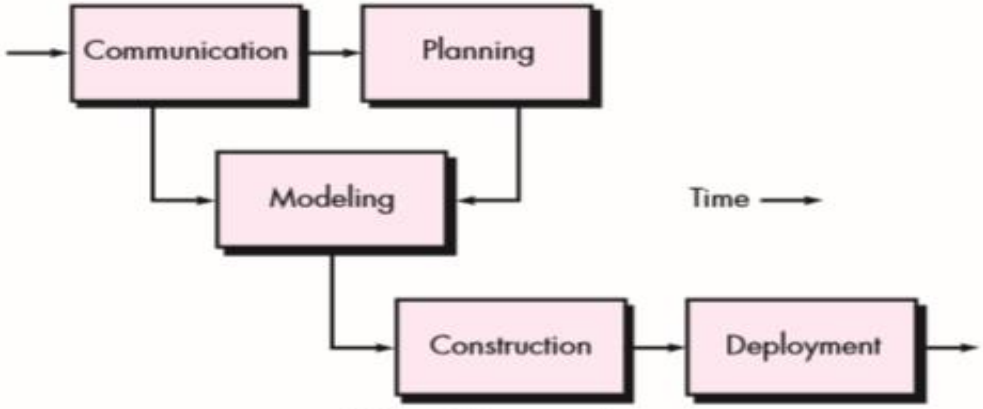
Process Flow



(b) Iterative process flow



(c) Evolutionary process flow

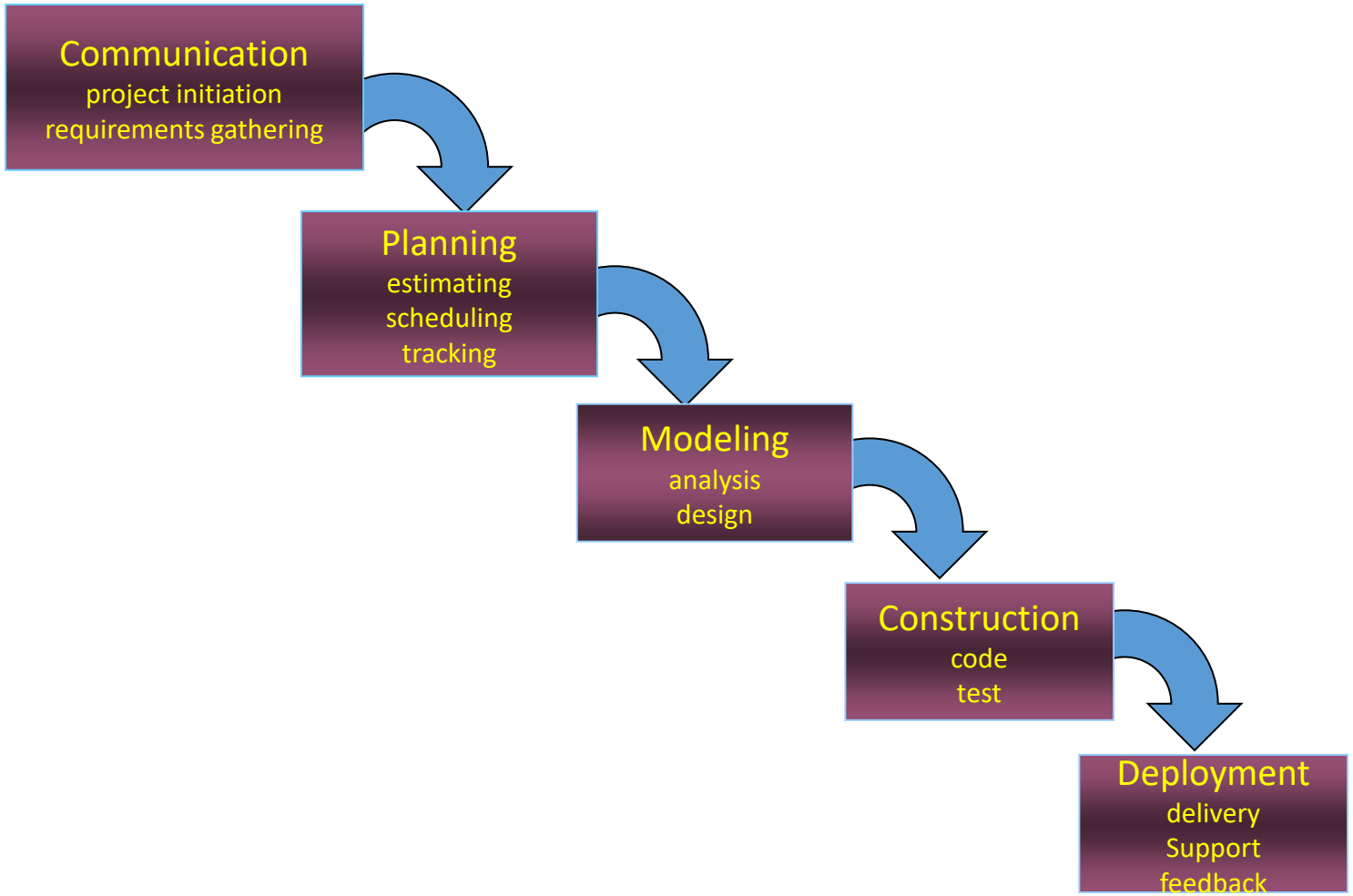


Process Model

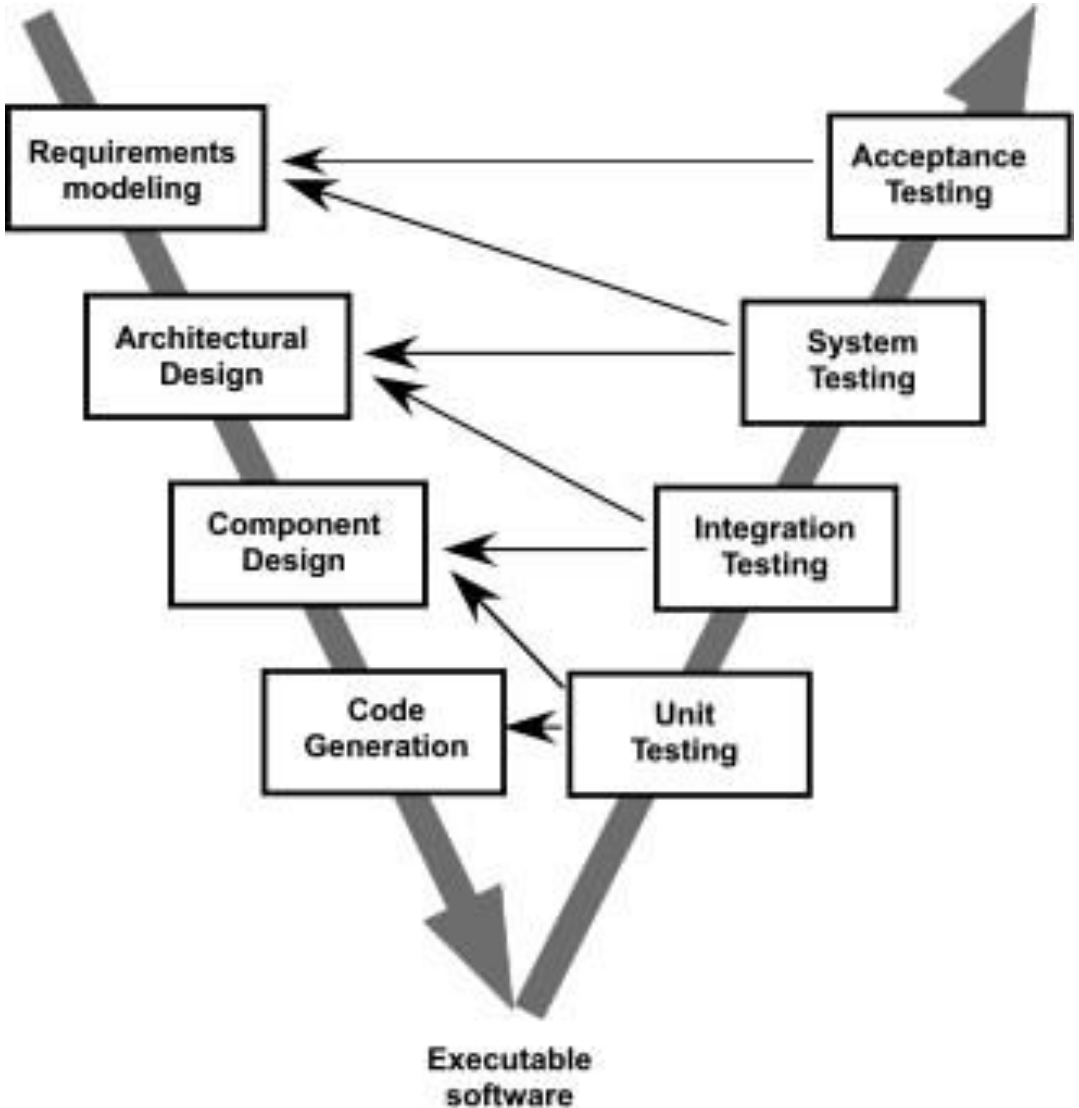
- Waterfall
- V-model
- Incremental Model
- Evolutionary Model： prototyping and Spiral Model
- Concurrent
- Unified Process

Remember: **Every model has merit and demerit!**

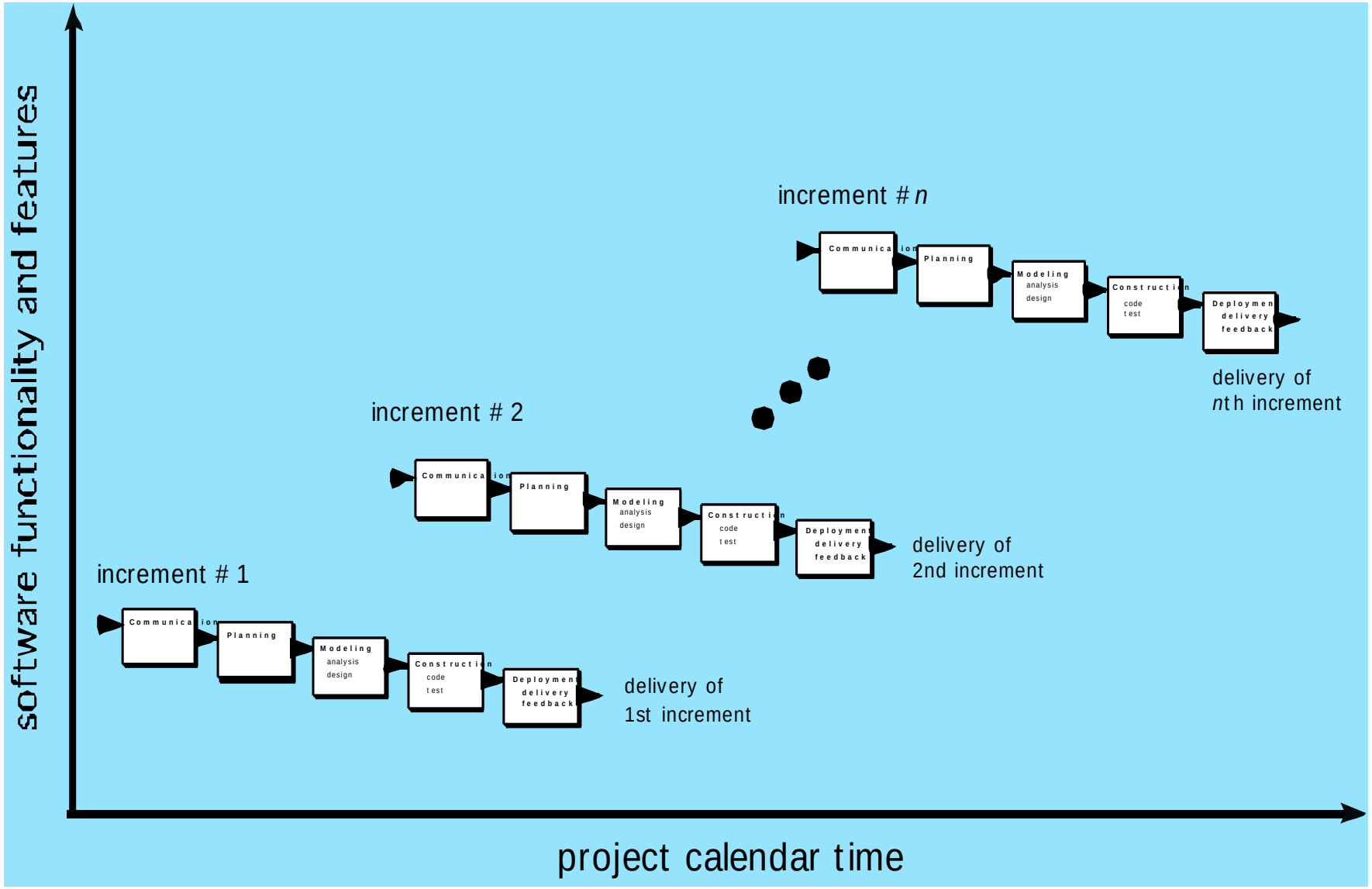
Waterfall model



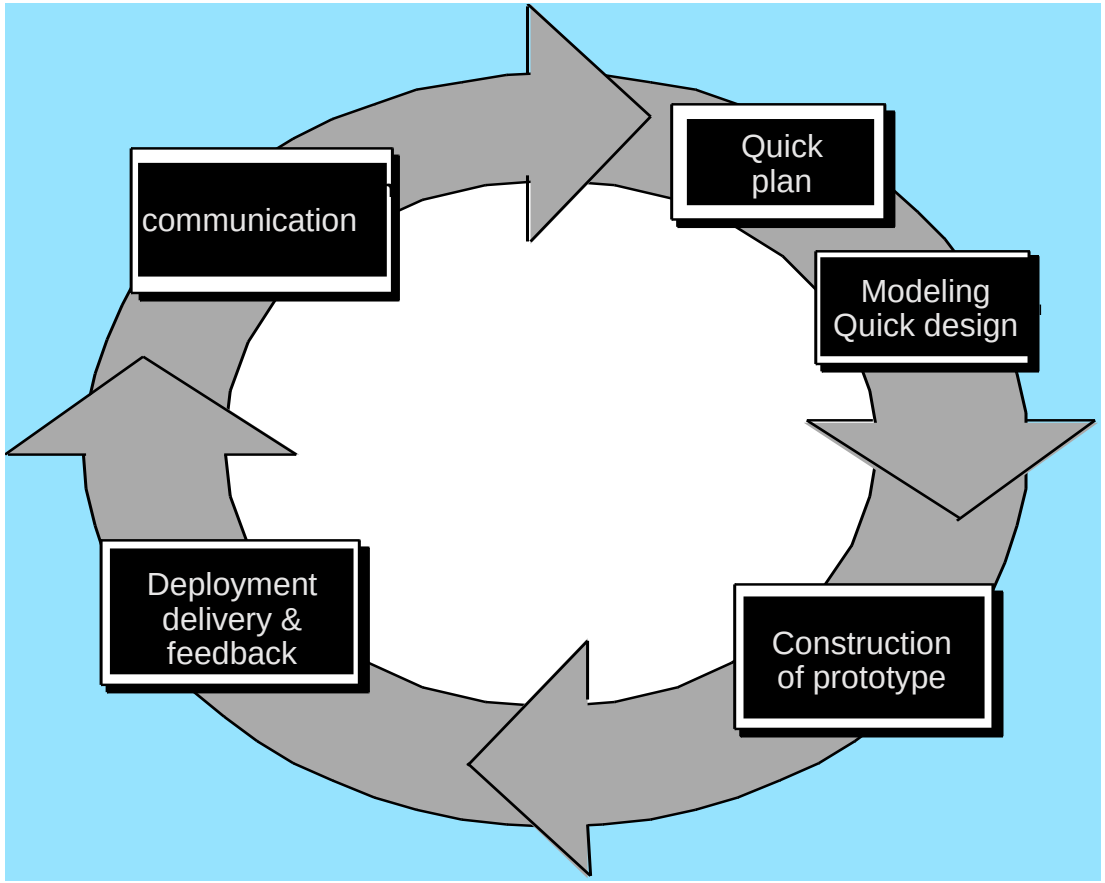
V-model


Requirements modeling
Acceptance Testing
Architectural Design
System Testing
Component Design
Integration Testing
Code Generation
Unit Testing
Executable software

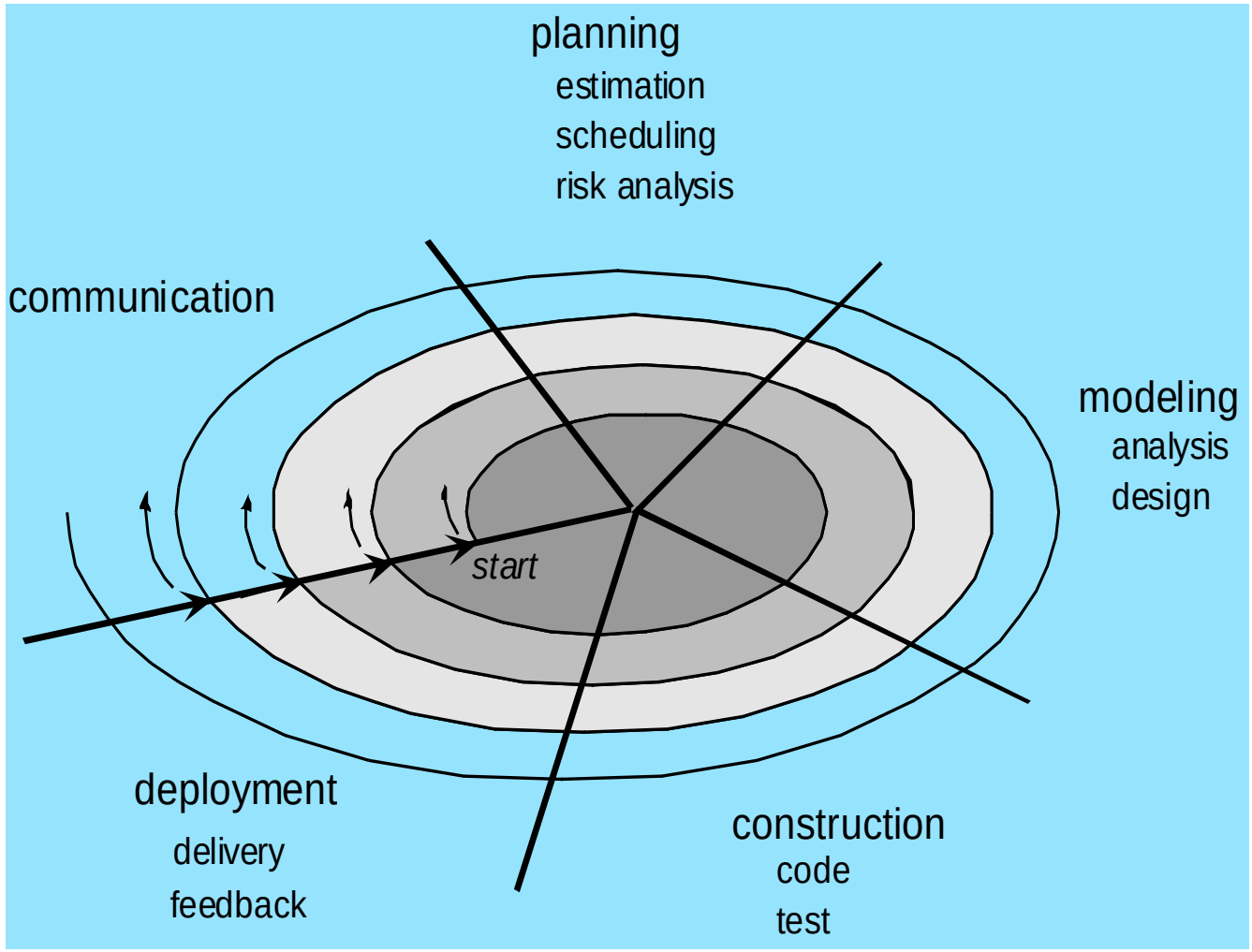
Incremental Model



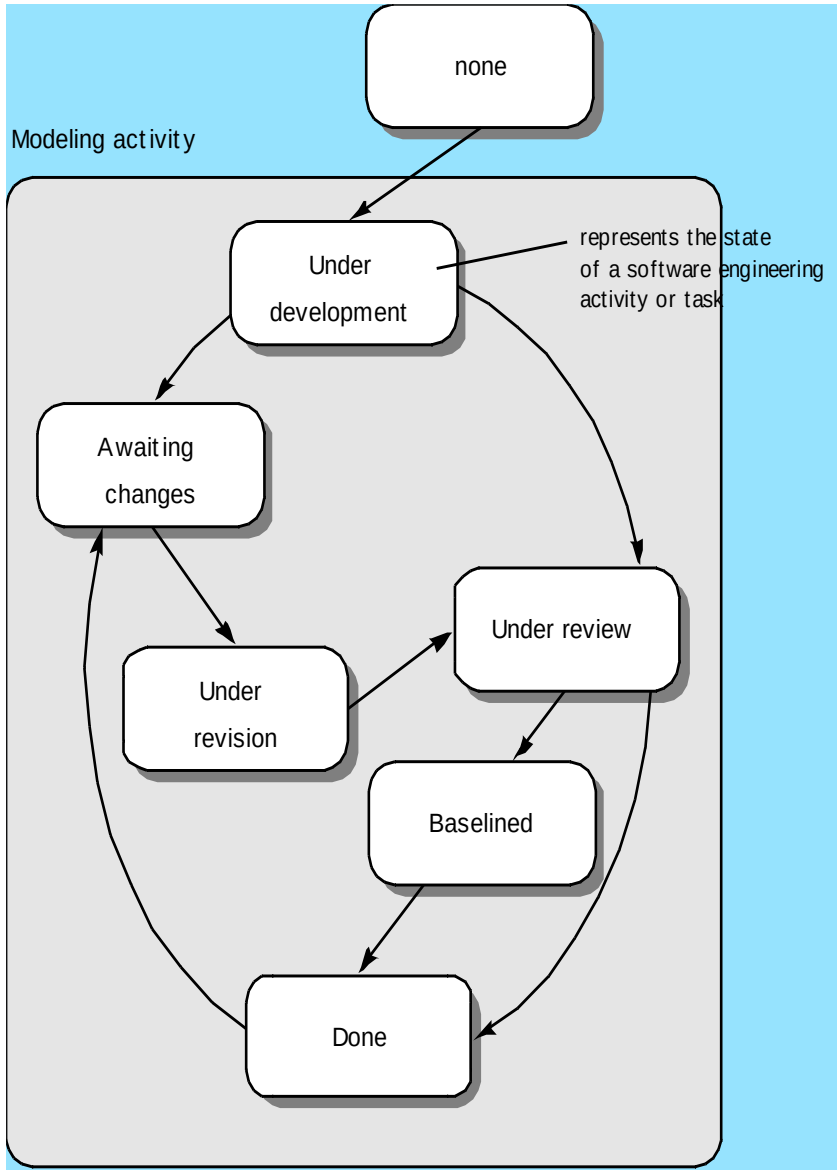
Prototyping



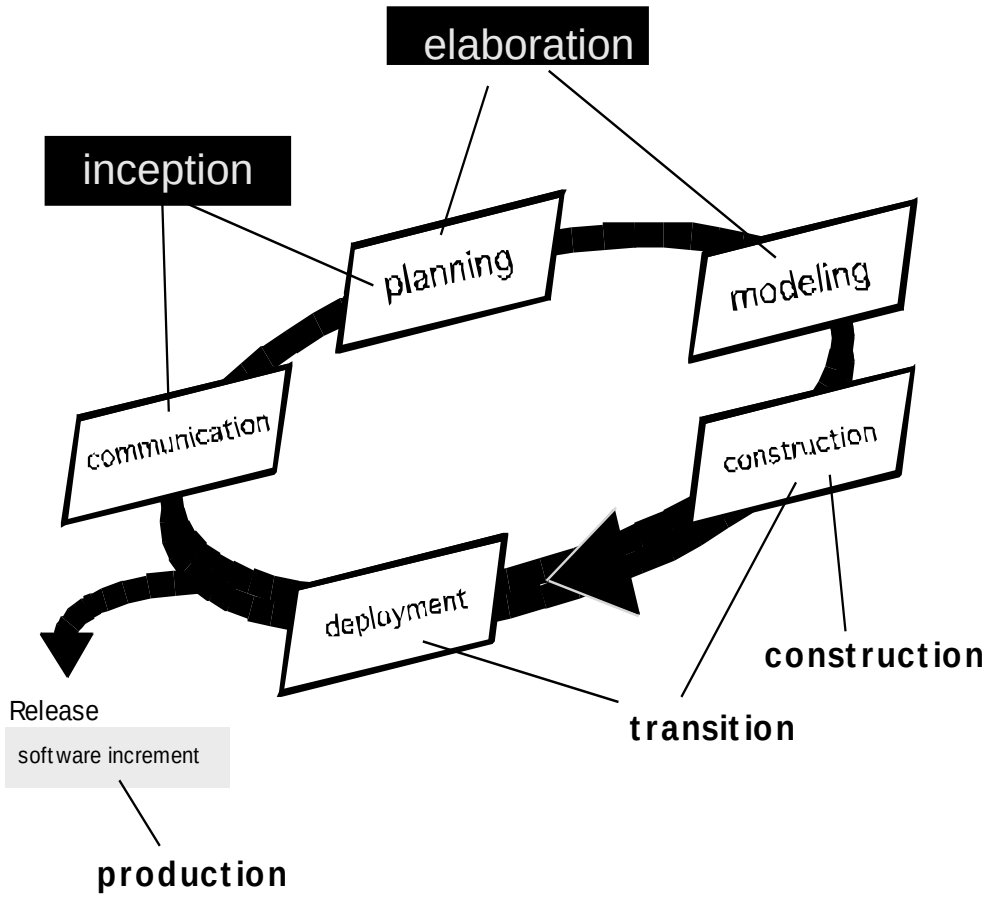
Spiral Model

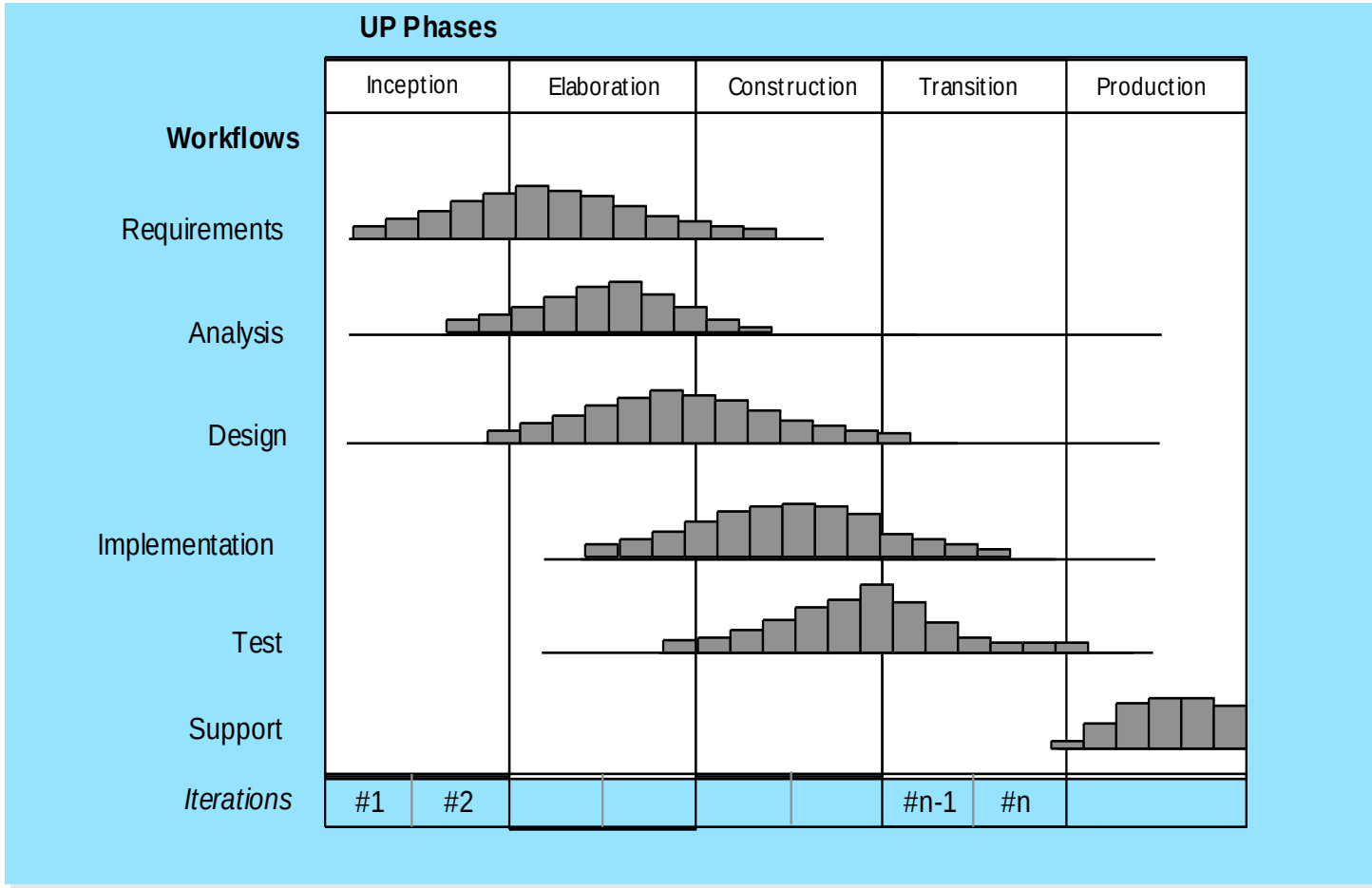


Concurrent Model



Unified Process





Question:

- What is demerit in the process models mentioned-above?

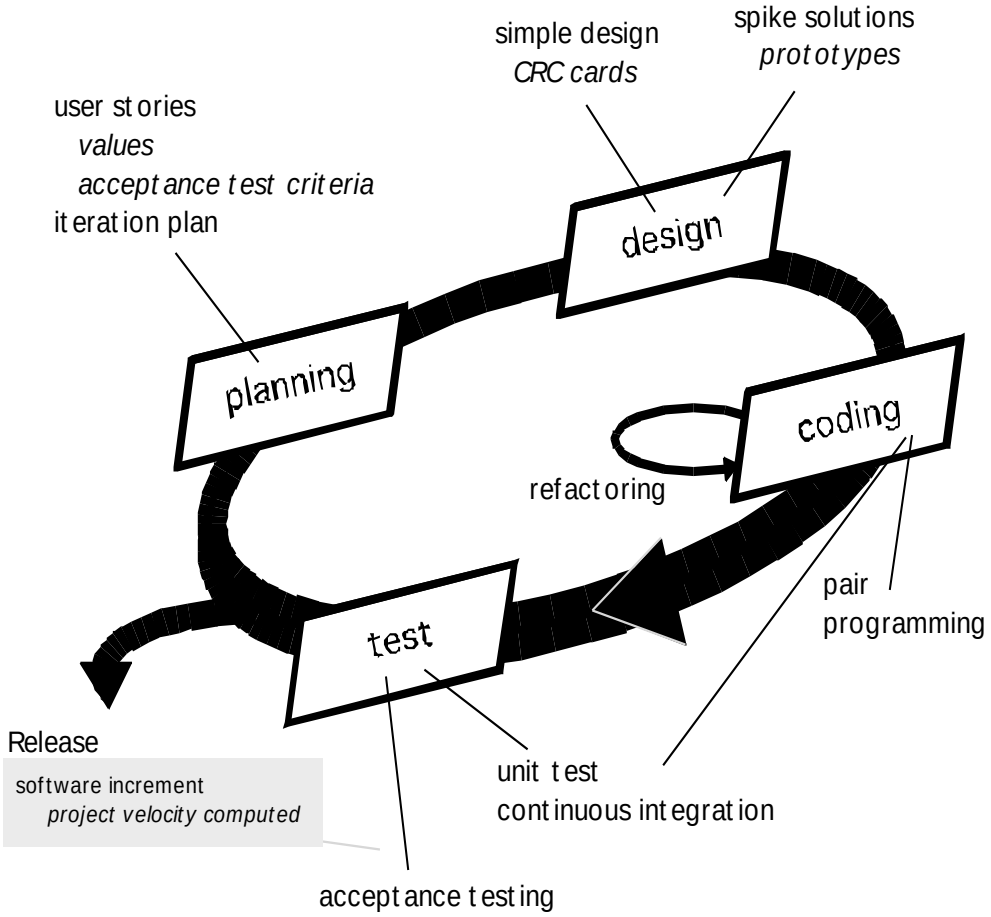
Disciplined ,Document-oriented !

Agile Process

- Why agile?
- Agile development manifesto :
  - ***Individuals and interactions* over processes and tools**
  - ***Working software* over comprehensive documentation**
  - ***Customer collaboration* over contract negotiation**
  - ***Responding to change* over following a plan**

Agile Process Model

- XP
- IXP

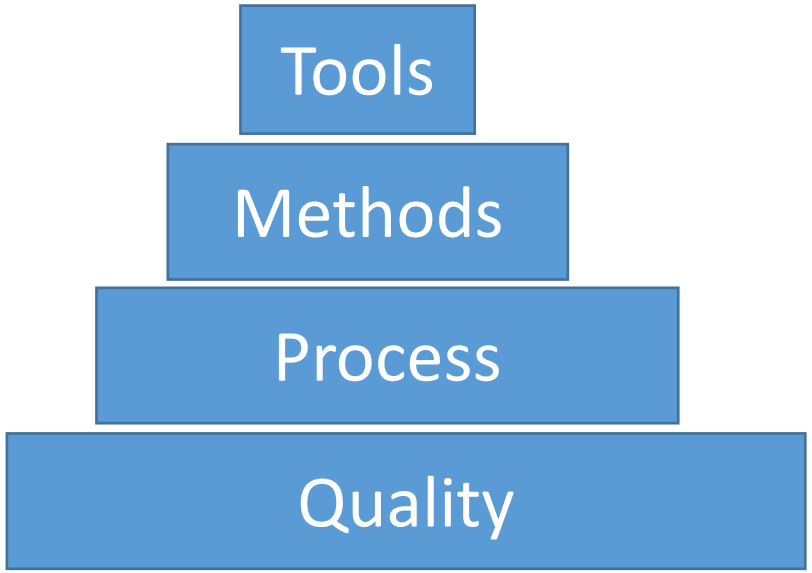


Other Agile Process Models

- Adaptive Software Development (ASD)
- Scrum
- Dynamic Systems Development Method (DSDM)
- Crystal
- Feature Drive Development (FDD)
- Lean Software Development (LSD)

Practice in Software Engineering

Generic Framework



- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
- Construction
  - Code generation
  - Testing
- Deployment

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

Principles that Guide Practice

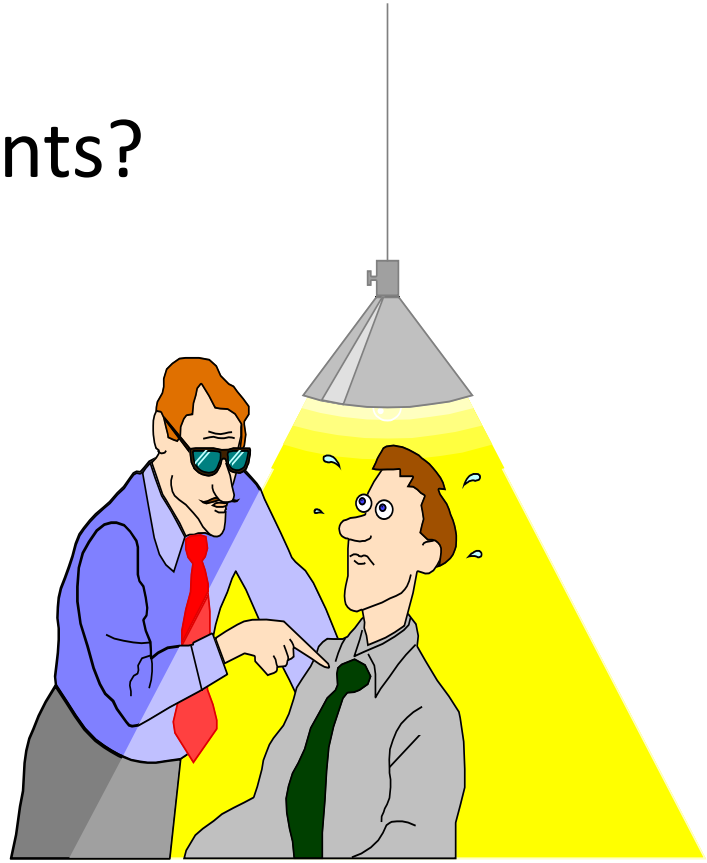
Requirement Engineering

- How to understand requirements?

  Functions

  Information

  Behavior



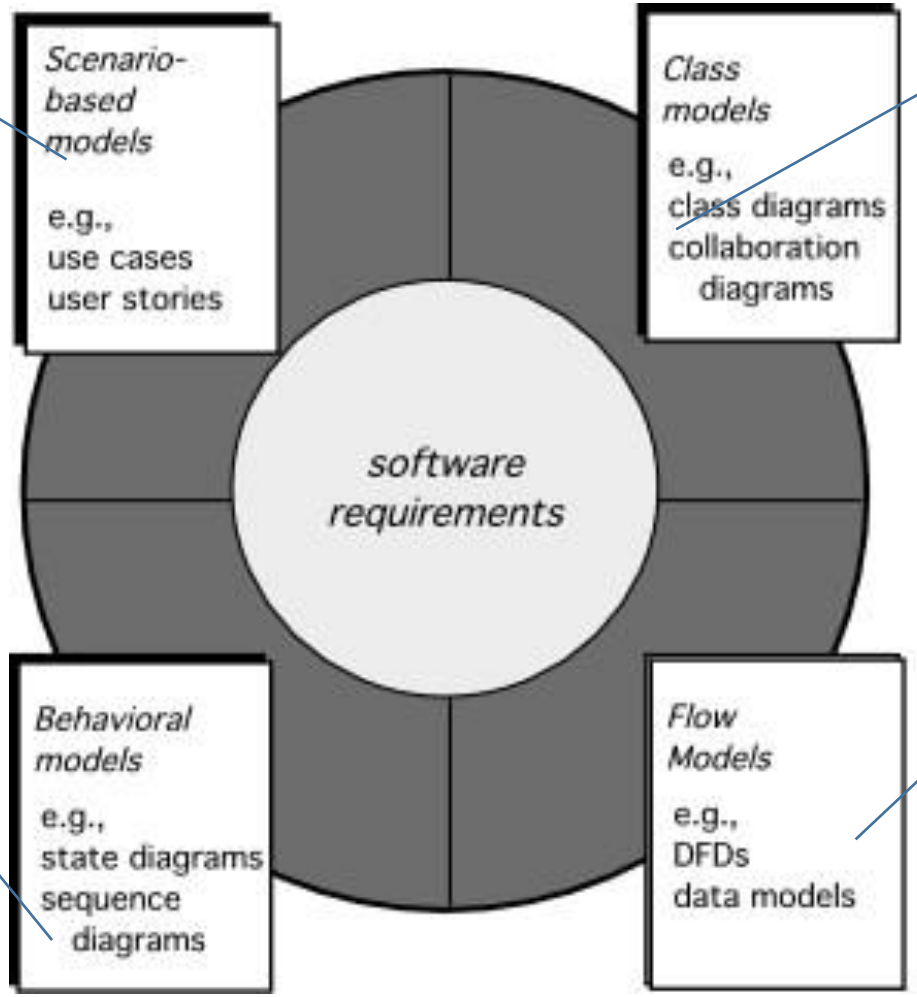
The broad spectrum of tasks and techniques that lead to an understanding of requirements

Requirement engineering tasks

- Inception
- Elicitation –gathering requirements
- Elaboration-requirement modeling
- Negotiation-win-win
- Specification-document， Model， prototype
- Validation-Quality assess
- Management-Change

Element of Requirement Modeling

Usecase Diagram
Activity Diagram
Swimlane
Usecase
Description

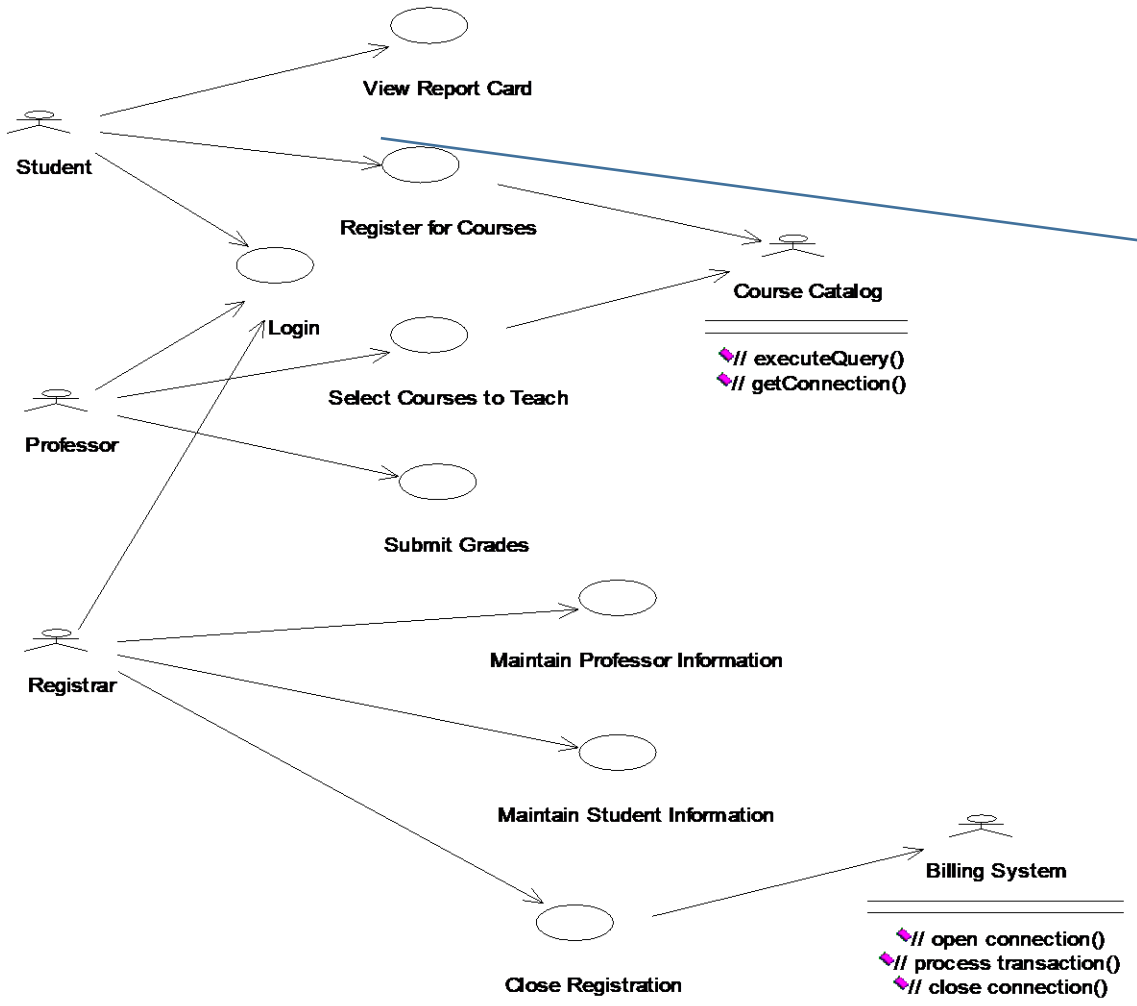


Class Diagrams
CRC
Collaboration Diagrams

STD
Sequence
Diagram

Data Flow
ERD

Use case Diagram Example



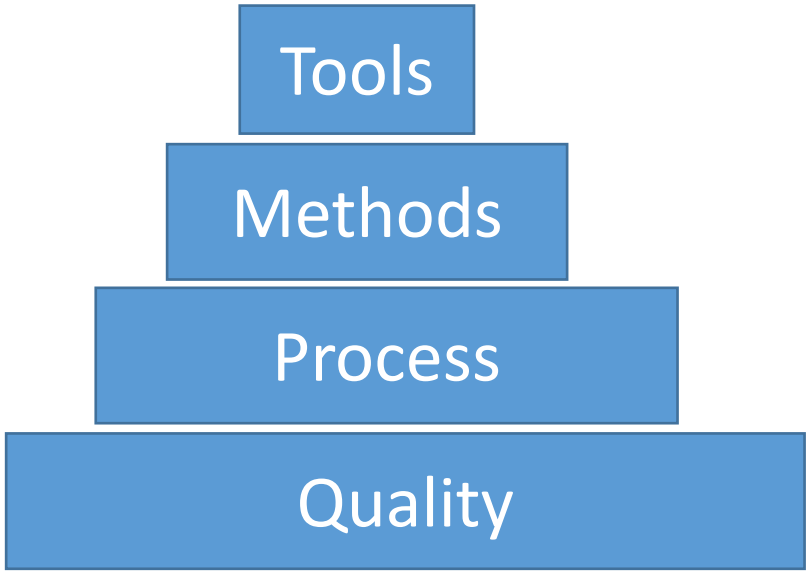
（1）Usecase specification Based on Template （textual）

（2）Visualizing： activity Diagram

（3） All the actors :system environment(person ,device, other system)

Software Engineering

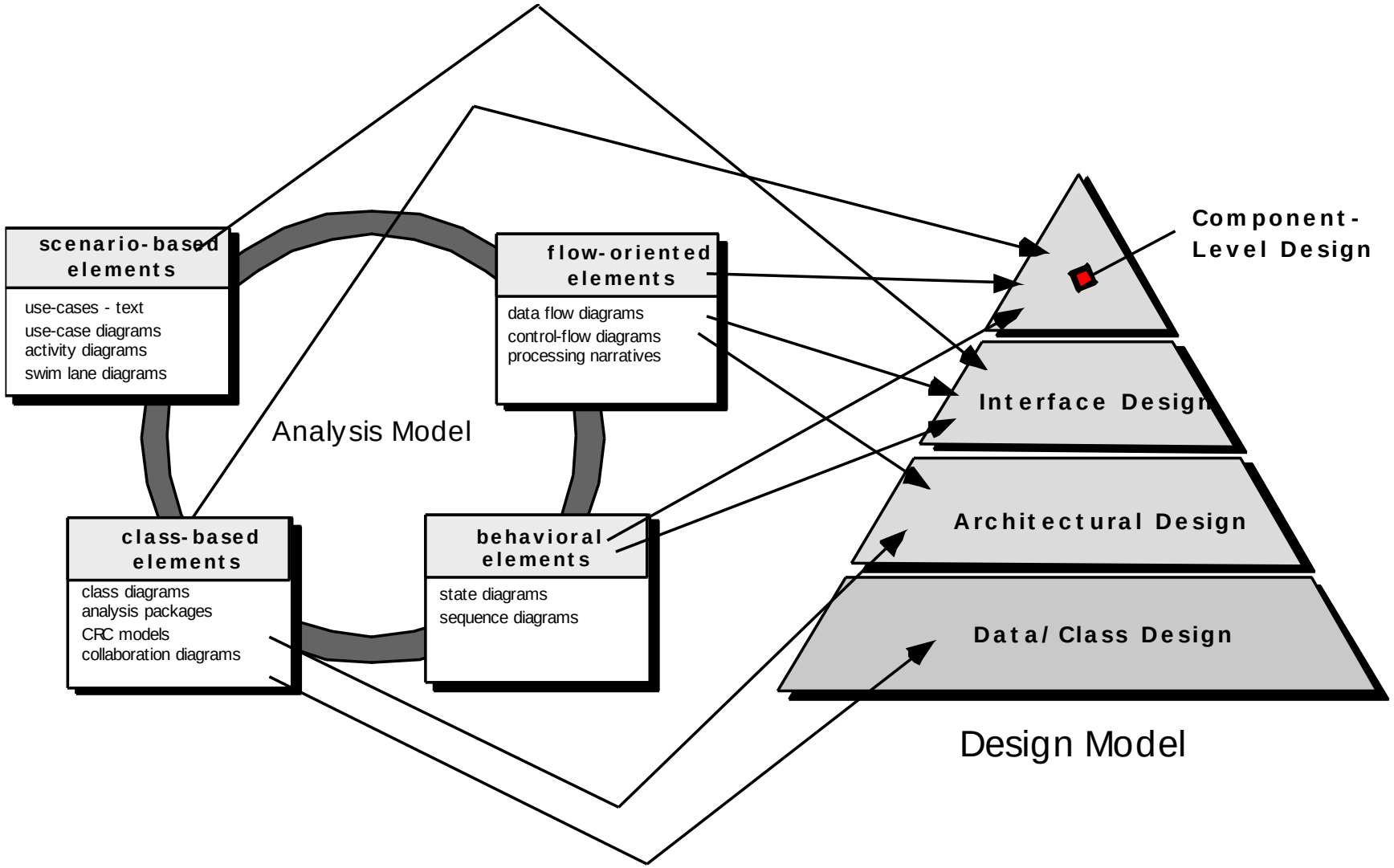
Generic Framework



- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
- Construction
  - Code generation
  - Testing
- Deployment

Principles that Guide Practice

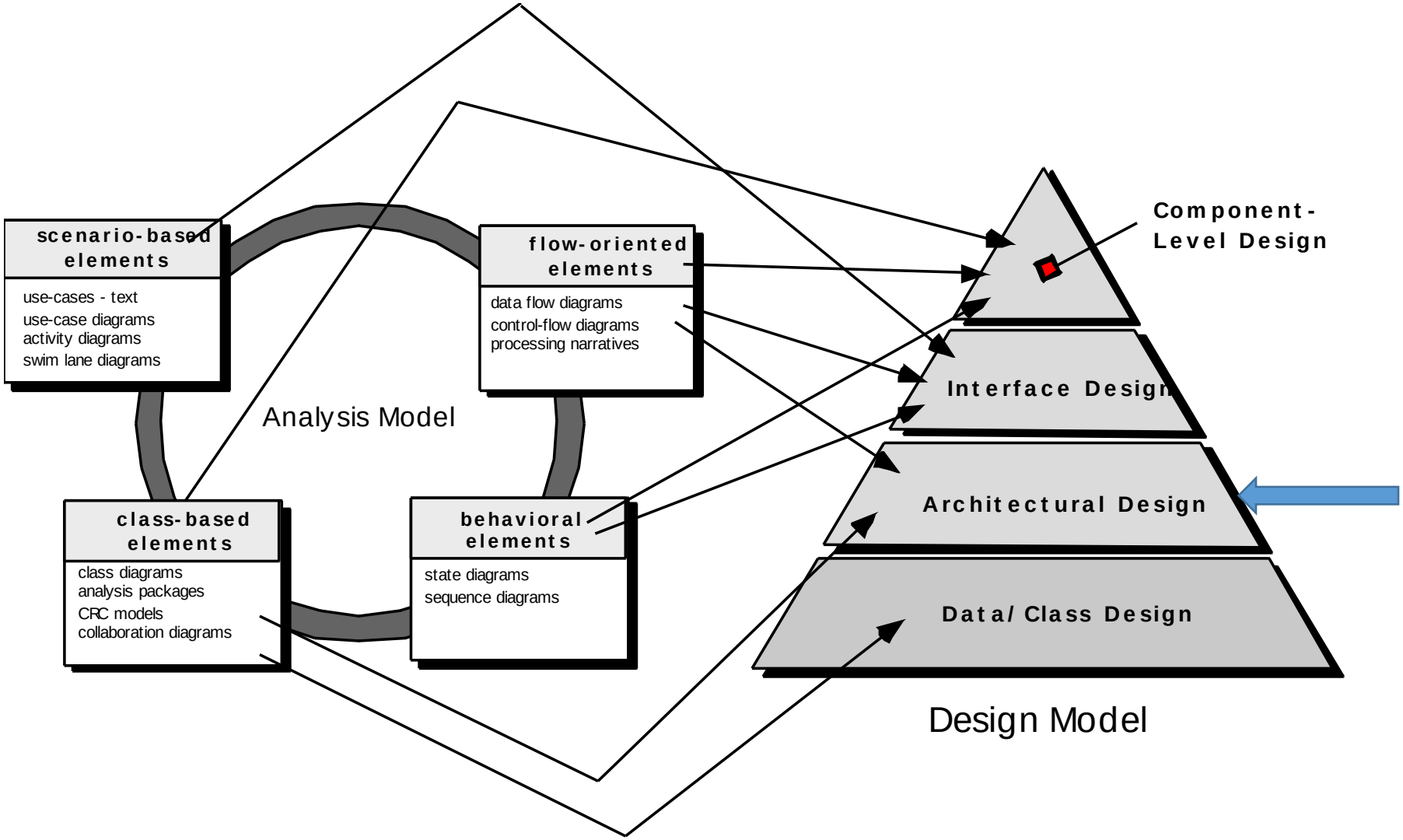
Software Design



Design Concepts

- Abstraction—data, procedure, control
- Architecture—the overall structure of the software
- Patterns—”conveys the essence” of a proven design solution
- Separation of concerns—any complex problem can be more easily handled if it is subdivided into pieces
- Modularity—compartmentalization of data and function
- Hiding—controlled interfaces
- Functional independence—single-minded function and low coupling
- Refinement—elaboration of detail for all abstractions
- Aspects—a mechanism for understanding how global requirements affect design
- Refactoring—a reorganization technique that simplifies the design
- OO design concepts： Design Class/Message/Inheritance/Polymophism
- Design Classes—provide design detail that will enable analysis classes to be implemented

Software Design



Architecture Design

- What is software Architecture?

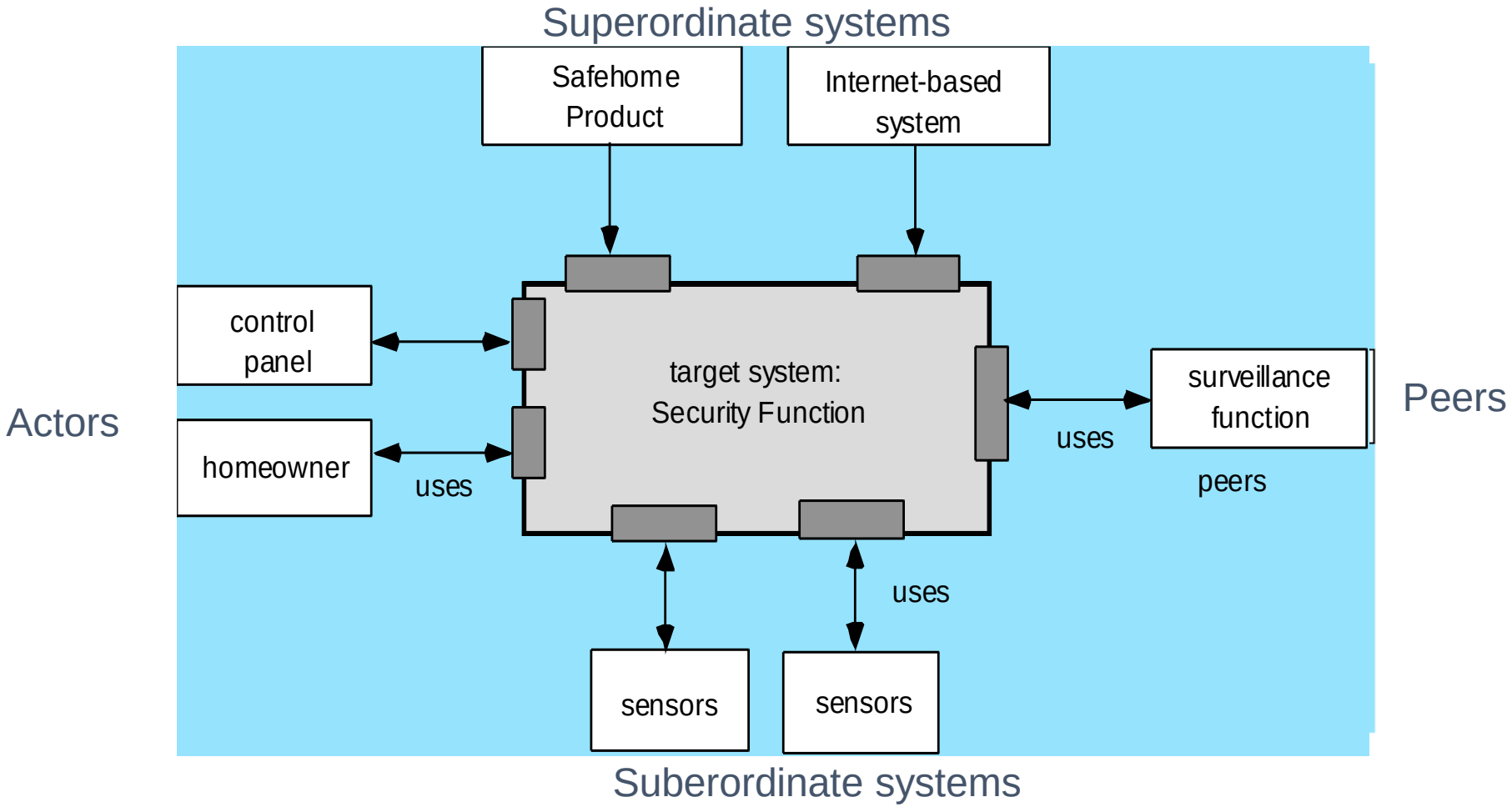
  Software Architecture is the structure or structures of the system, which comprise software components, the externally visible properties of those components and the relationships among them.

- Why software architecture is so important?
- Architecture styles:

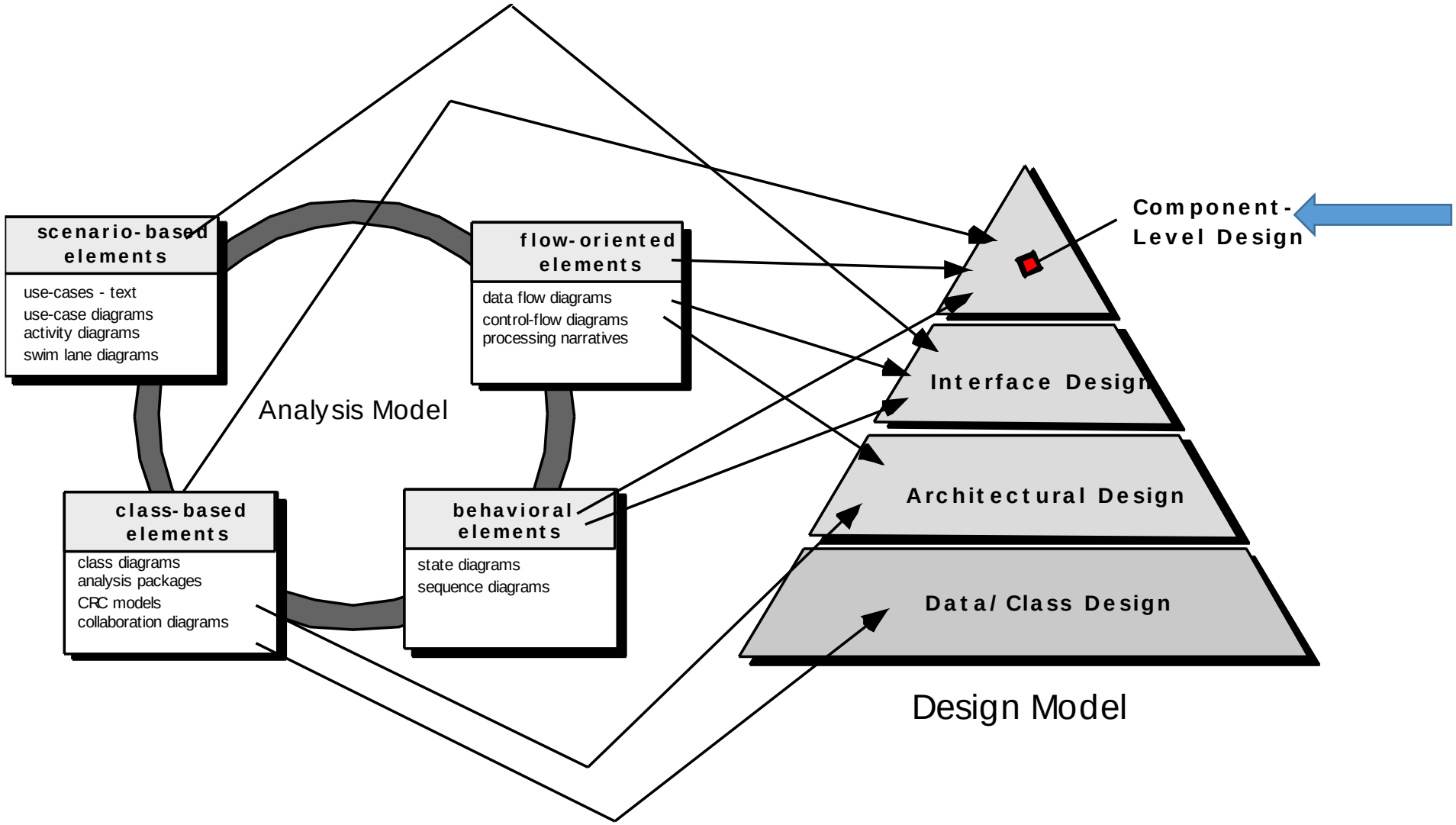
  Data flow,Data-centric,Layered,call-return,Object-Oriented

- Architecture Context Diagram(ACD)
- Software Architecture derived from DFD

Architecture Context Diagram



Software Design



Component-level Design

- What is Component?

a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces

- What is interface?

a group of externally visible (i.e., public) operations. The interface contains no internal structure, it has no attributes, no associations . .

Different View on Component

- Object-Oriented view:

  A component contains a set of collaborating classes

- Traditional View

  A component contains

  - *processing logic,*
  - *the internal data structures that are required to implement the processing logic, and*
  - *an interface that enables the component to be invoked and data to be passed to it.*

Design Principles

- Open-Closed Principle(OCP)
- Liskov Substitution Principle (LSP)
- Dependency Inversion Principle (DIP)
- The Interface Segregation Principle (ISP)

Packaging Principle

- The Release Reuse Equivalency Principle (REP)
- The Common Closure Principle (CCP)
- The Common Reuse Principle (CRP

Cohesion

- Conventional view

  the "single-mindedness" of a module

- OO view

  cohesion implies that a component or class encapsulates only attributes and operations that are closely related to one another and to the class or component itself

Coupling

- Conventional view

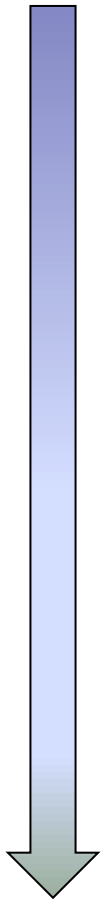
  The degree to which a component is connected to other components and to the external world

- OO view

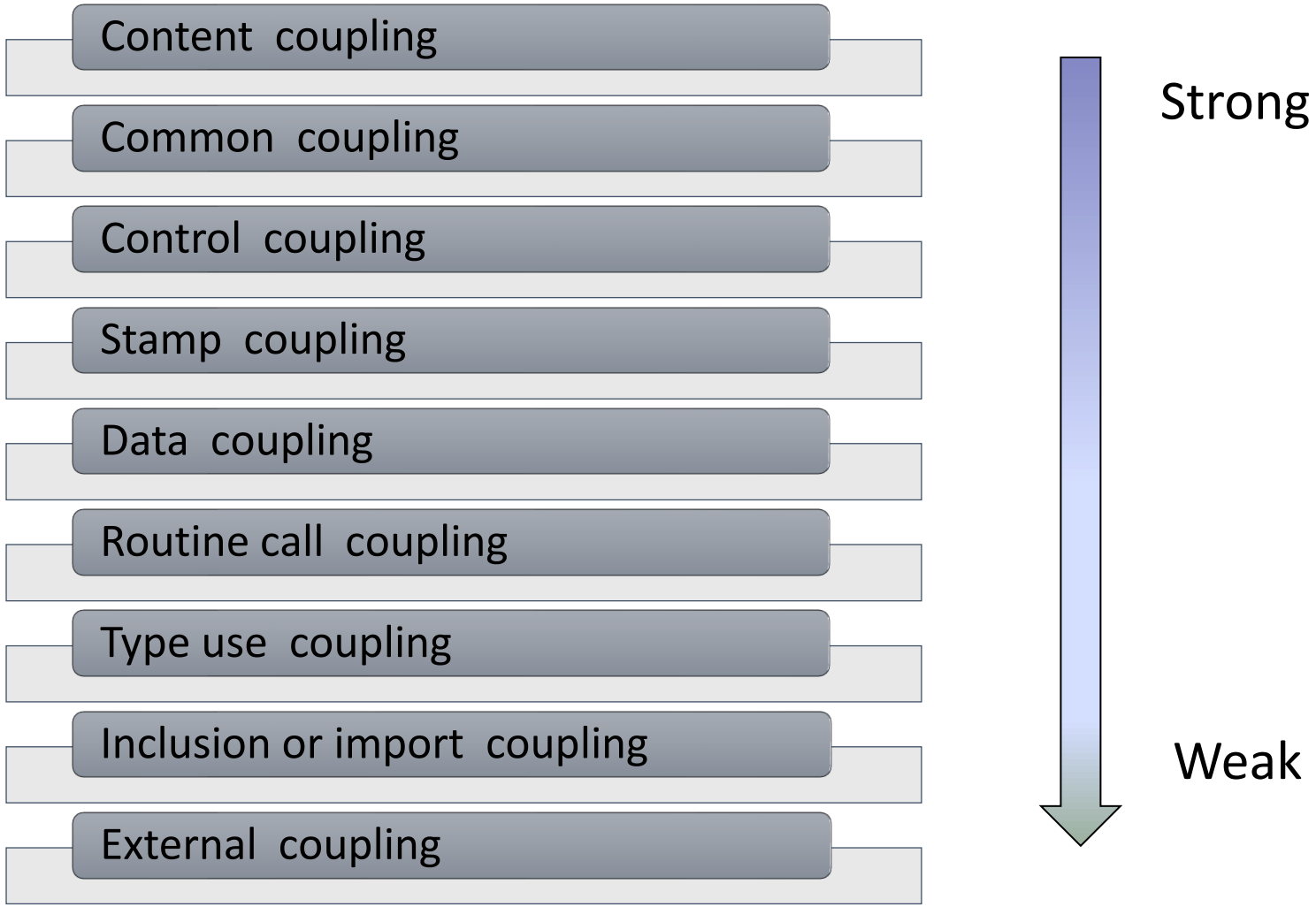
  a qualitative measure of the degree to which classes are connected to one another

Levels of Cohesion

- Functional Cohesion
- Layer Cohesion
- Communicational Cohesion
- Sequential Cohesion
- Procedural Cohesion
- Temporal Cohesion
- Utility Cohesion

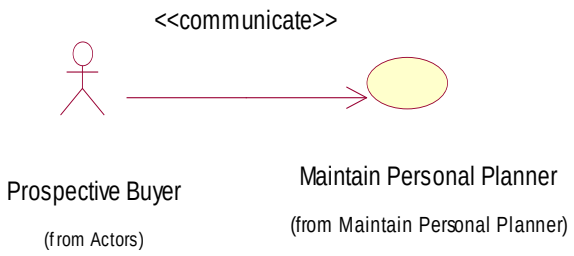


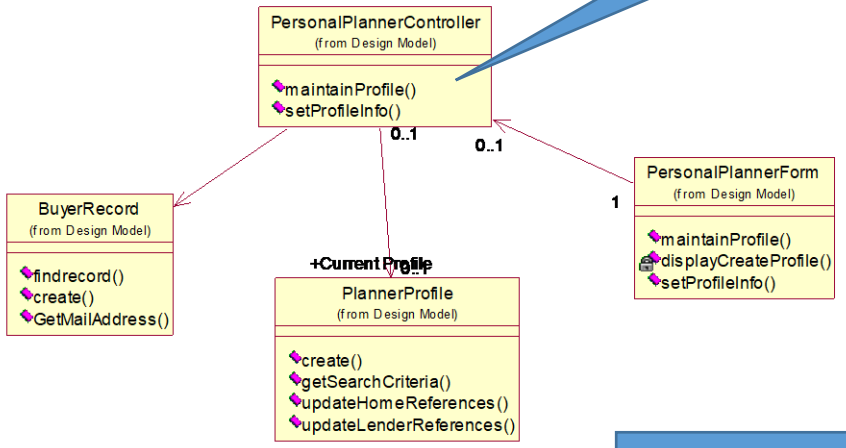
Levels of Coupling



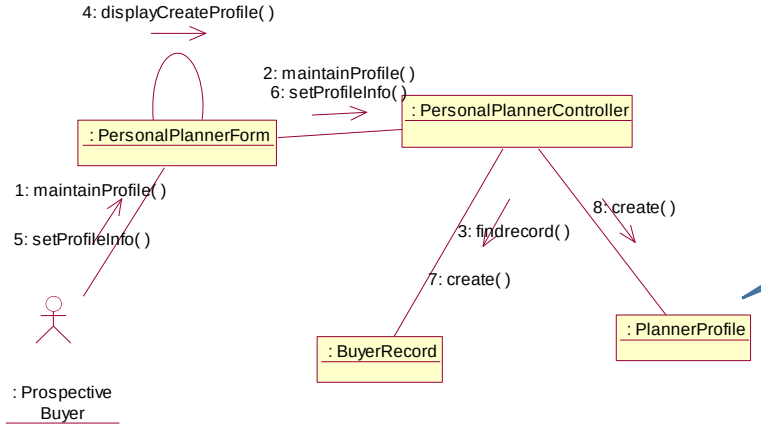
O-O View of Component Design

Class diagram
Class
elaboration





Collaboration
Diagram
/object diagram



Designing Traditional Components

- A traditional software component is often called module, procedure, or subroutine.
- Structured design uses a set of constrained logical constructs
  - *sequence*
  - *condition*
  - *Repetition*
- Design Tools:
  - *Program Flow Graph*
  - *PDL*
  - *Decision Table/Tree*

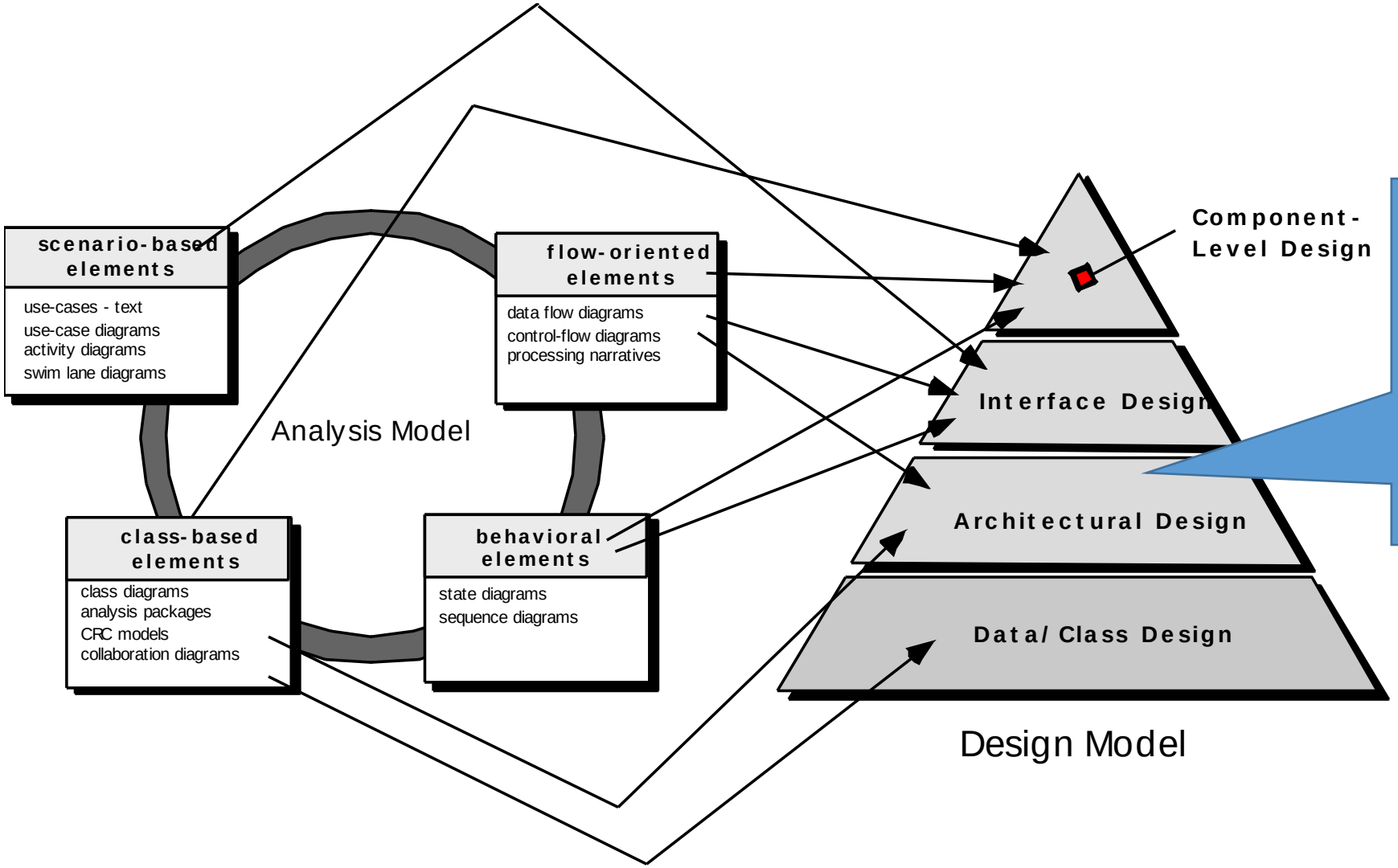
Component-Based Development

- Component-based Software Engineering

- Domain Engineering： *analysis, construction, and dissemination*

- Component Standard
  - OMG/CORBA
  - Microsoft COM and .NET
  - Sun JavaBeans

Software Design



(1) the user interface (UI)
(2) external interfaces
(3) internal interfaces

User Interface Design

- UI design (increasingly called *usability design*) is a major software engineering action.

- *usability*

  a qualitative measure of the ease and efficiency with which a human can employ the functions and features offered by the high-technology product

Golden Rules

- Place the user in control
- Reduce the user’s memory load
- Make the interface consistent

UI Analysis and Design Models

User Interface analysis Tasks

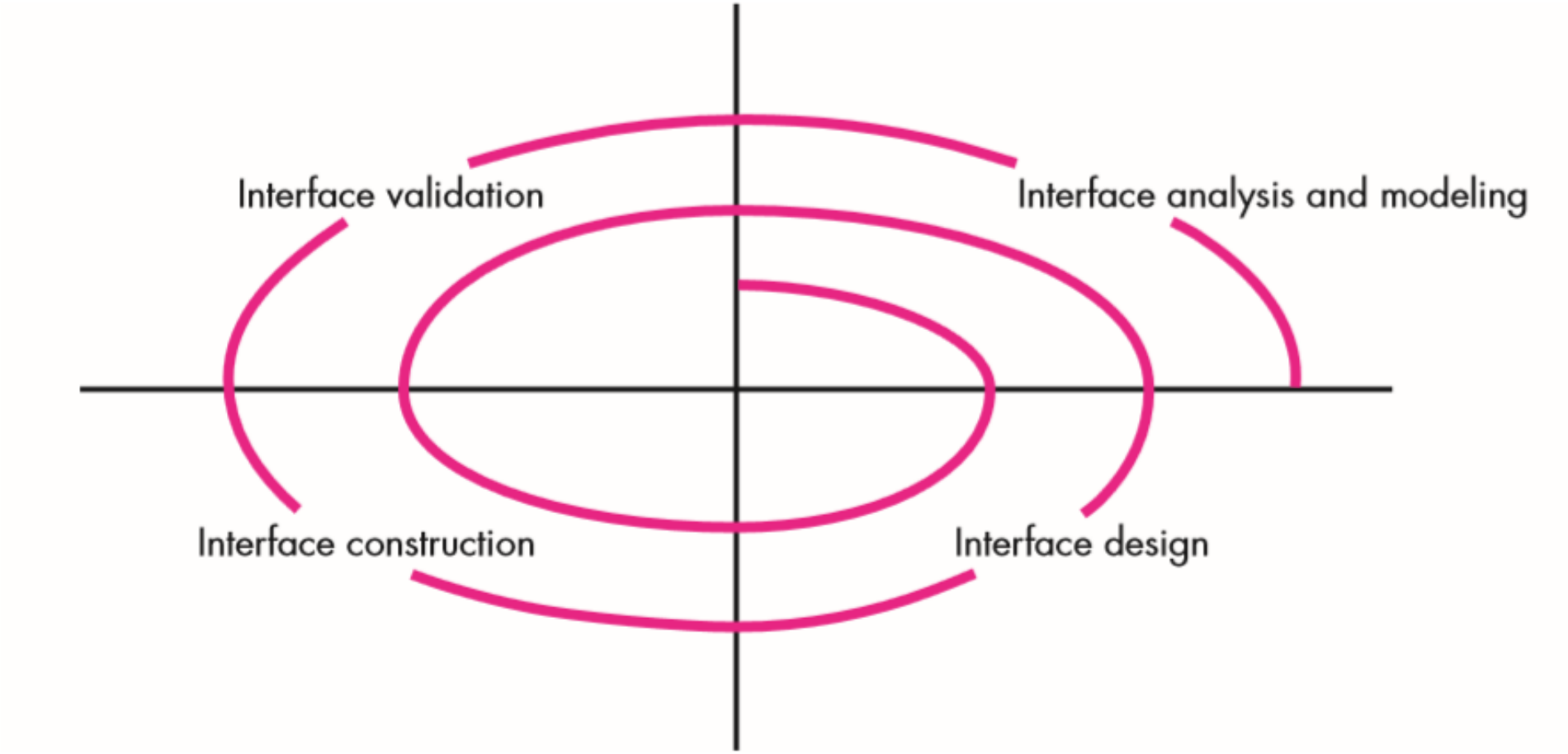
- User analysis
- Task analysis
- Display content analysis
- Environmental analysis

Design Models

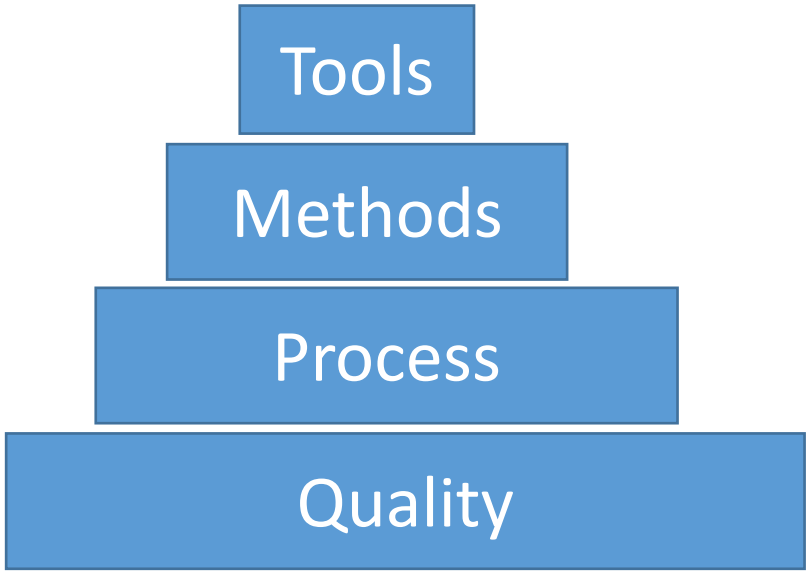
- User Model
- Design Model
- Mental Model(system Perception)
- Implement

Keep the above four models consistent

UI Analysis and Design Process



Software Engineering



Generic Framework

- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
- Construction
  - Code generation
  - Testing
- Deployment

Principles that Guide Practice

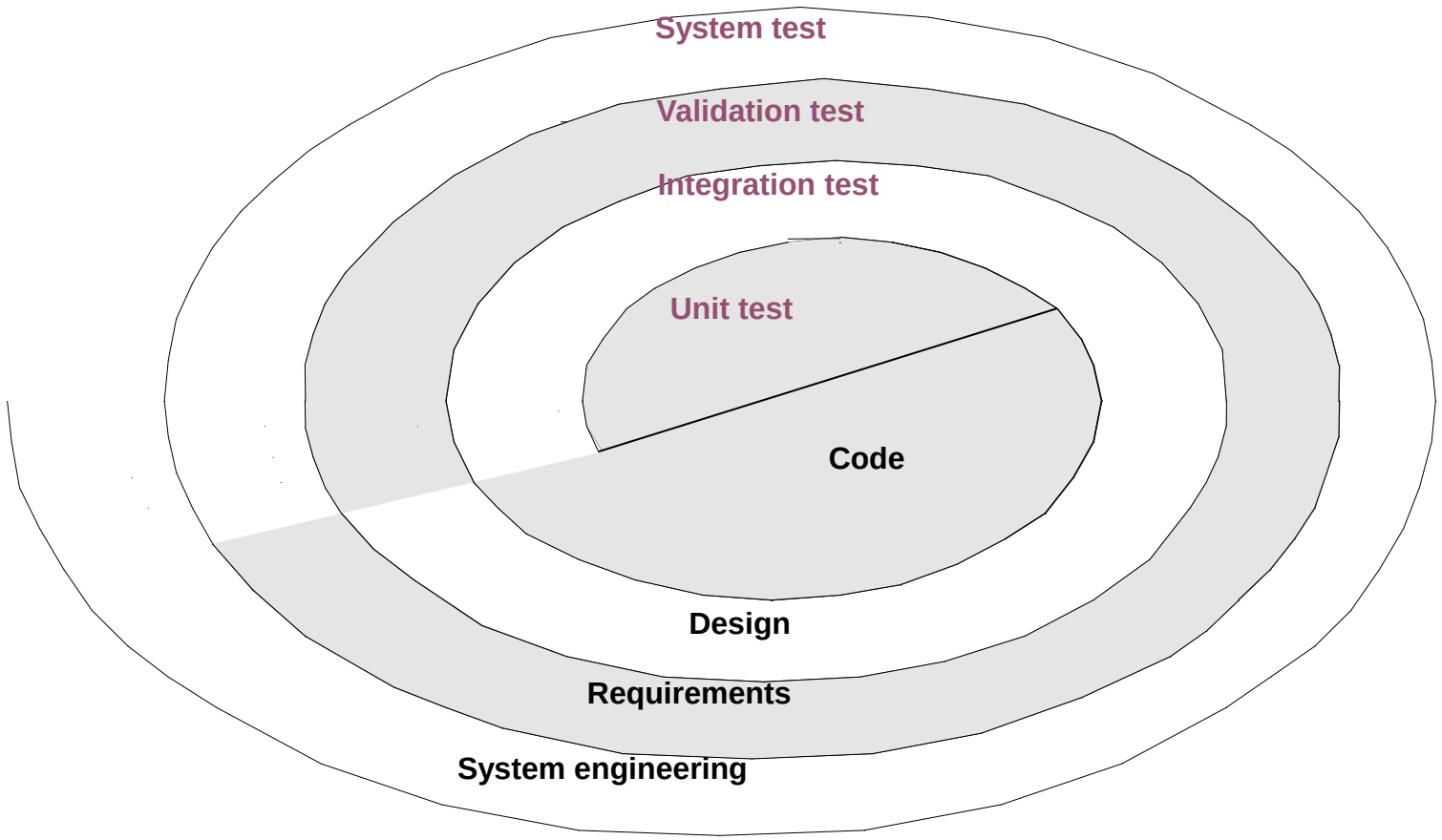
Software Testing

- What is testing?

Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user.

- Testing Strategies
- Testing Technique
- Debugging: A Diagnostic Process

Testing Strategies: From small toward big



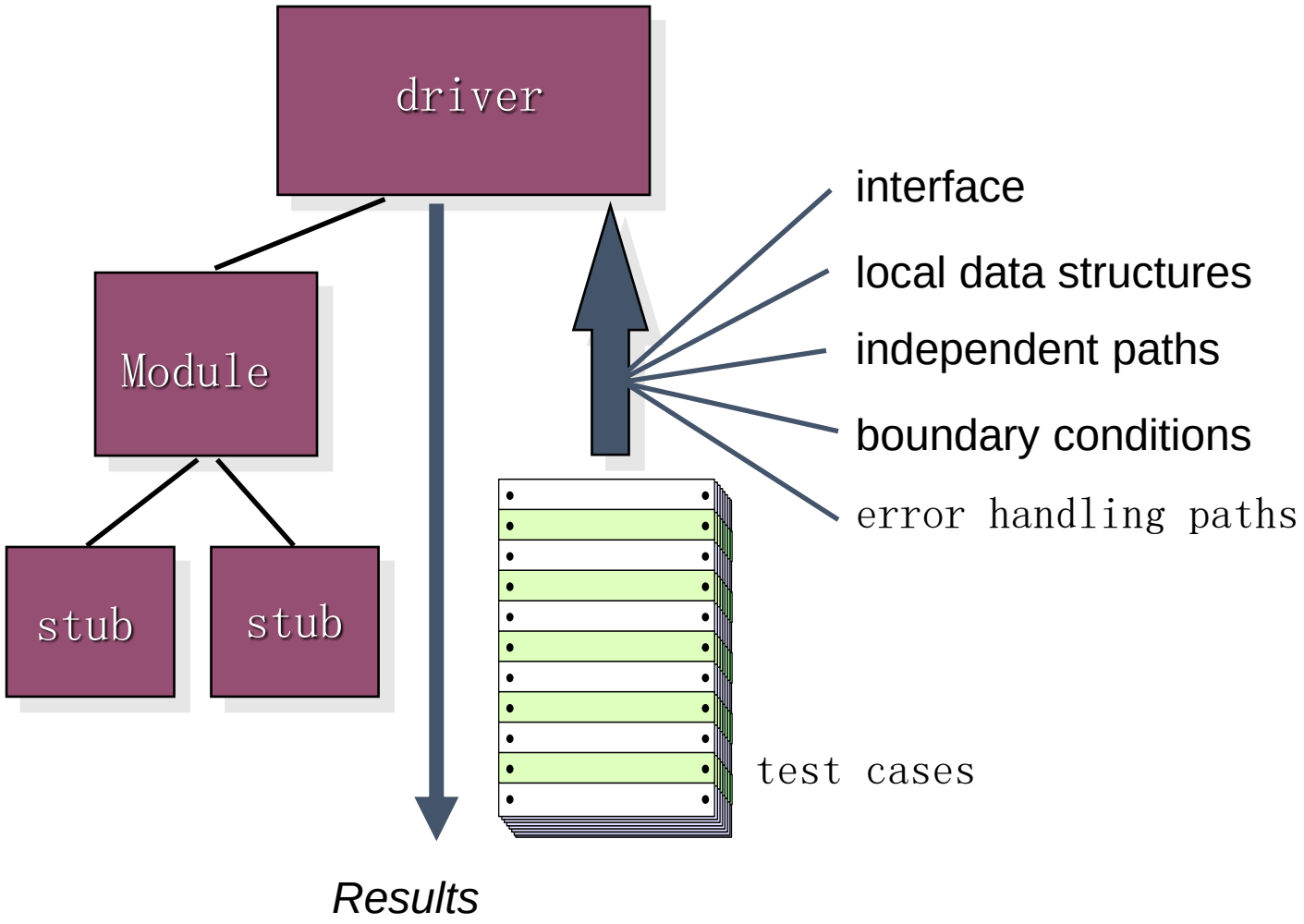
Strategy for conventional and o-o software

- *Module and class* is the smallest element in Conventional and o-o software respectively
- Different strategies for conventional and object-oriented software in unit testing and integration testing
- NO difference in Validation testing and system testing strategy

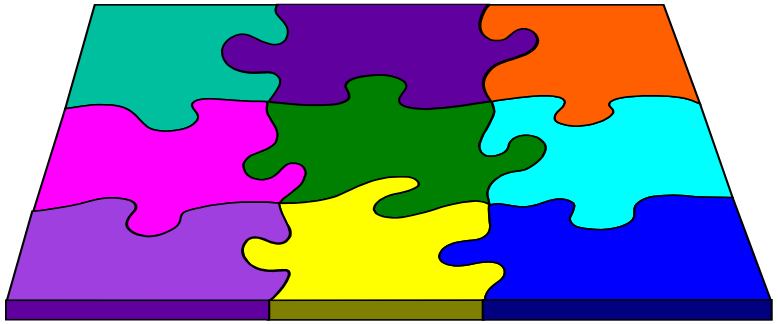
Unit Testing for Conventional software

- Focuses verification effort on the smallest unit of software design – the software component or module.
- Focuses on the internal processing logic and data structures within the boundaries of a component.
- Can be conducted in parallel for multiple components.

Unit Test Environment



Integration Testing for conventional software



- The objective of integration testing is to take unit-tested components and build a program structure that has been dictated by design.
- Integration strategy:
  - Bottom-up
  - Top-down
  - Sandwich

Regression Testing

- *Regression testing* is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.
- Regression testing helps to ensure that changes (due to testing or for other reasons) do not introduce unintended behavior or additional errors.
- Regression testing may be conducted manually, by
  - reexecuting a subset of all test cases or
  - using automated capture/playback tools.

Unit Testing in the OO Context

- Class testing for OO software is the equivalent of unit testing for conventional software.
  - operations within the class are tested
  - the state behavior of the class is examined

Integration Testing in the OO Context

- two different strategies
  - thread-based testing
    - Threads are sets of classes that respond to an input or event.
  - use-based testing
    - Begins the construction of the system by testing *independent classes*.
    - The next layer of classes, called *dependent classes*, that use the independent classes are tested.
- cluster testing
  - is one step in the integration testing of OO software
  - integrates the set of classes required to demonstrate one collaboration (to find collaboration error and verify CRC Model)

Validation Testing

- Validation Testing focuses on user-visible actions and user-recognizable output from the system, Which is achieved through a series of tests that demonstrate conformity with requirements

- Including
  - Alpha testing
  - Beta testing
  - Acceptance testing

System Testing

- All tests work to verify that system elements have been properly integrated and perform allocated functions

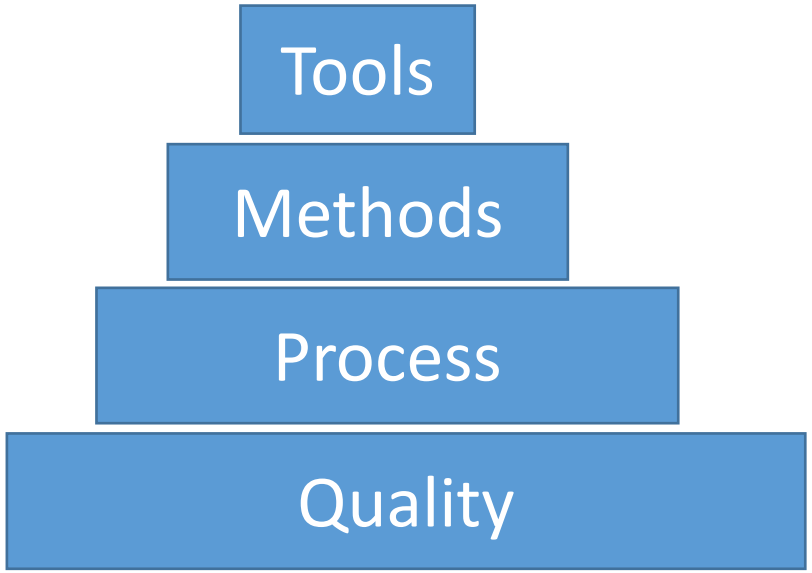
- Including
  - Recovery testing
  - Security testing
  - Stress testing
  - Performance Testing
  - Deployment testing

Testing Technique

- Software testing fundamentals -testability
- White-Box testing
  - Basic path testing （环形复杂度）
  - Control structure testing
- Block-Box testing
  - Equivalence partitioning
  - Boundary value analysis
- OO testing method
  - Testing methods at class level
  - Testing methods at inter-class level

Practice in Software Engineering

Generic Framework



- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
- Construction
  - Code generation
  - Testing
- Deployment

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

Principles that Guide Practice

Type

- Multiple choice, 20题X1
- T/F:10题X1
- Term explanation:5题*4
- QA:3题*8
- Analysis and design：26（4小题）

Thanks!

Good Luck in final test!