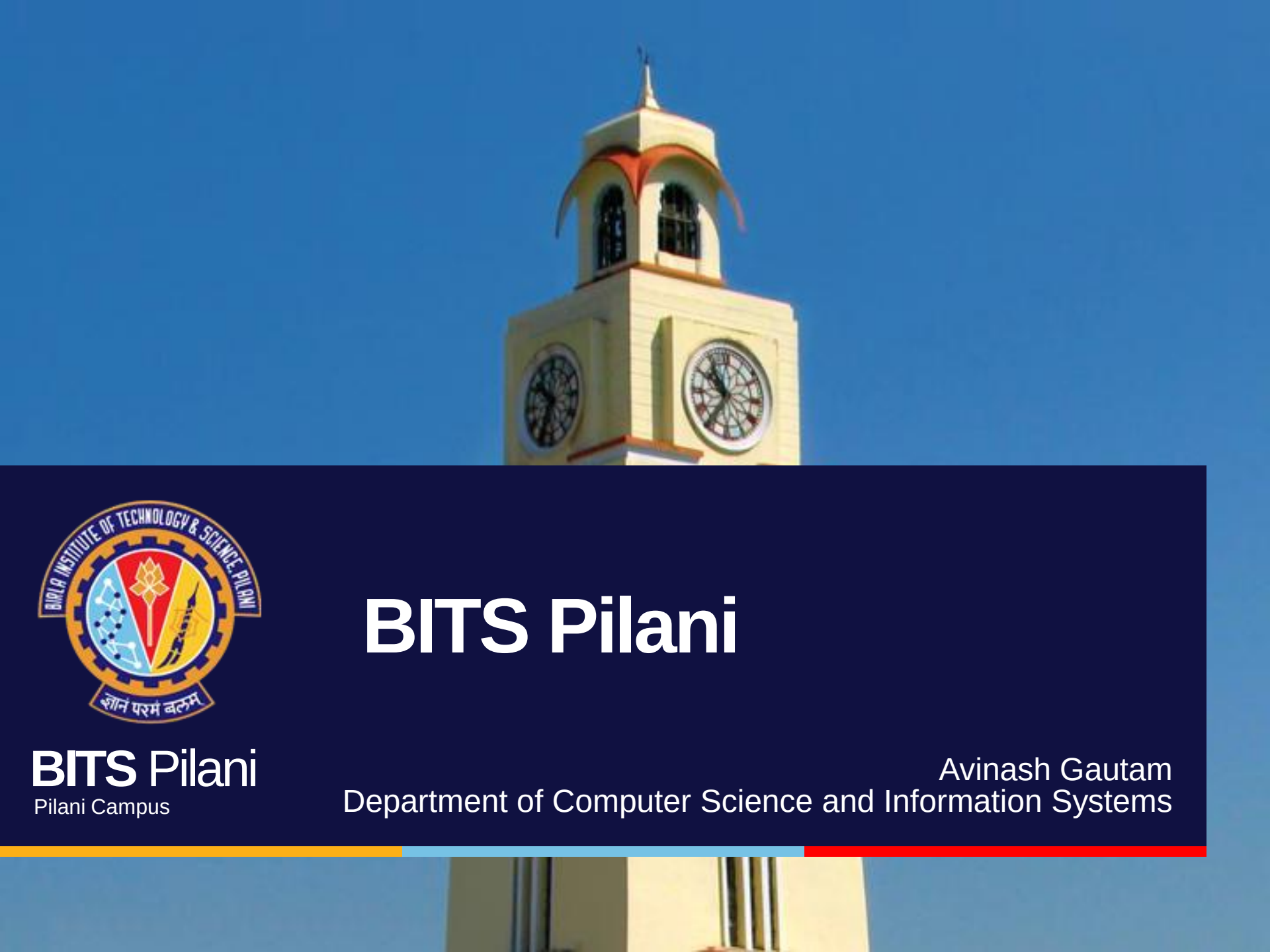


BITS Pilani

BITS Pilani
Pilani Campus

Avinash Gautam
Department of Computer Science and Information Systems

The Unified Process

- Use Case driven
  - A use case is a piece of functionality in the system that gives a user a result of value.
- Use cases capture functional requirements
- Use case answers the question: *What is the system supposed to do for the user?*

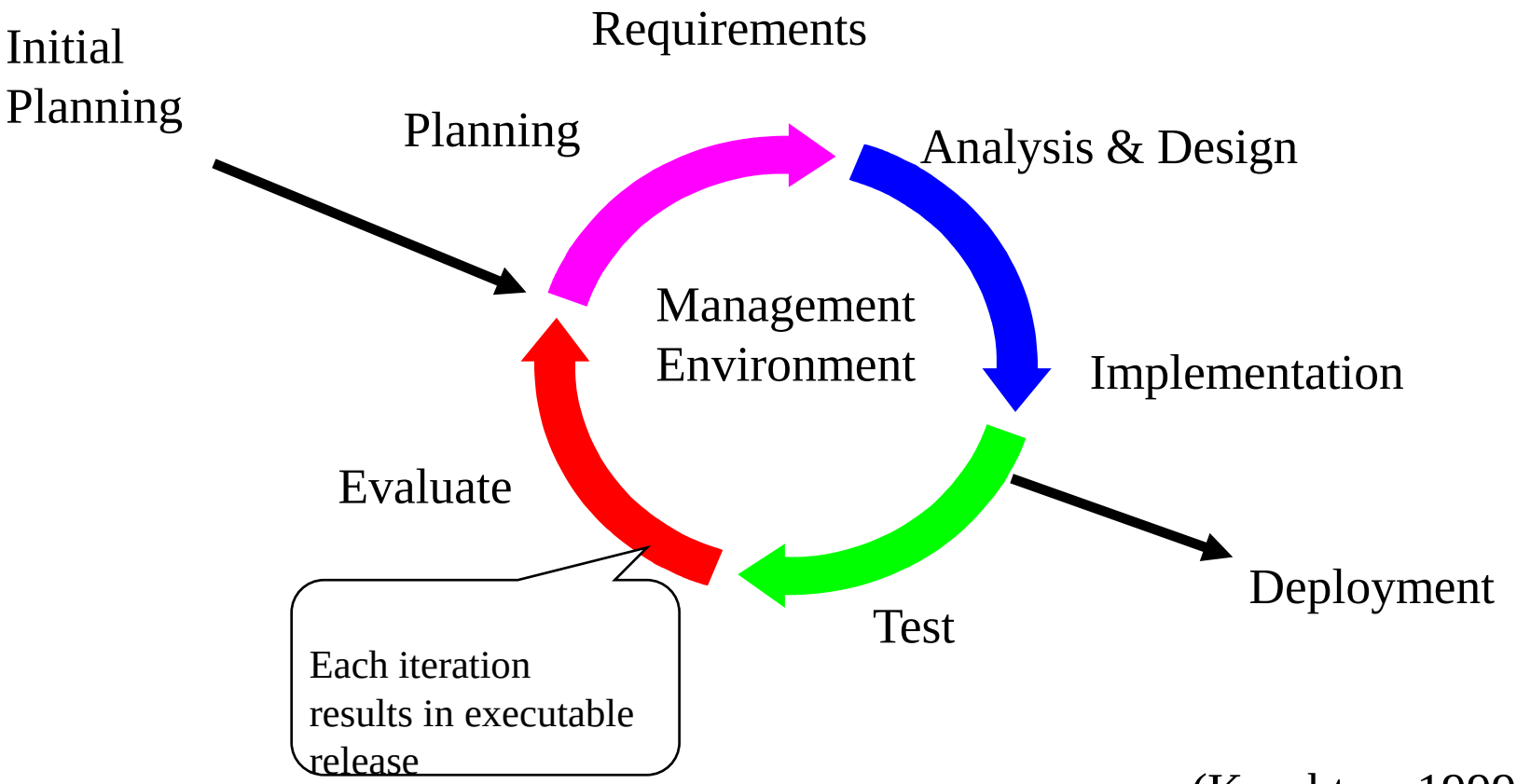
The Unified Process

- Architecture centric
  - similar to architecture for building a house
  - Embodies the most significant static and dynamic aspects of the system
  - Influenced by platform, OS, DBMS etc.
  - Related as ***function*** (use case) and ***form*** (architecture)
  - Primarily serves the realization of use cases
  - The form must allow the system to evolve from initial development through future requirements (i.e. the design needs to be flexible)

The Unified Process

- **Iterative and Incremental**
  - commercial projects continue many months and years
  - to be most effective - break the project into *iterations*
- Every iteration - identify use cases, create a design, implement the design
- Every iteration is a complete development process

An iterative and incremental process



(Kruchten, 1999)

Iterations

- Iterations must be selected & developed in a planned way i.e. in a logical order - early iterations must offer utility to the users
  - iteration based on a group of use cases extending the usability of the system developed so far
  - iterations deal with the most important risks first
  - not all iterations are additive - some replace earlier “superficial” developments with a more sophisticated and detailed one.

Benefits of an iterative approach

- Risks are mitigated earlier
- Change is more manageable
- Higher level of reuse
- Project team can learn along the way
- Better overall quality

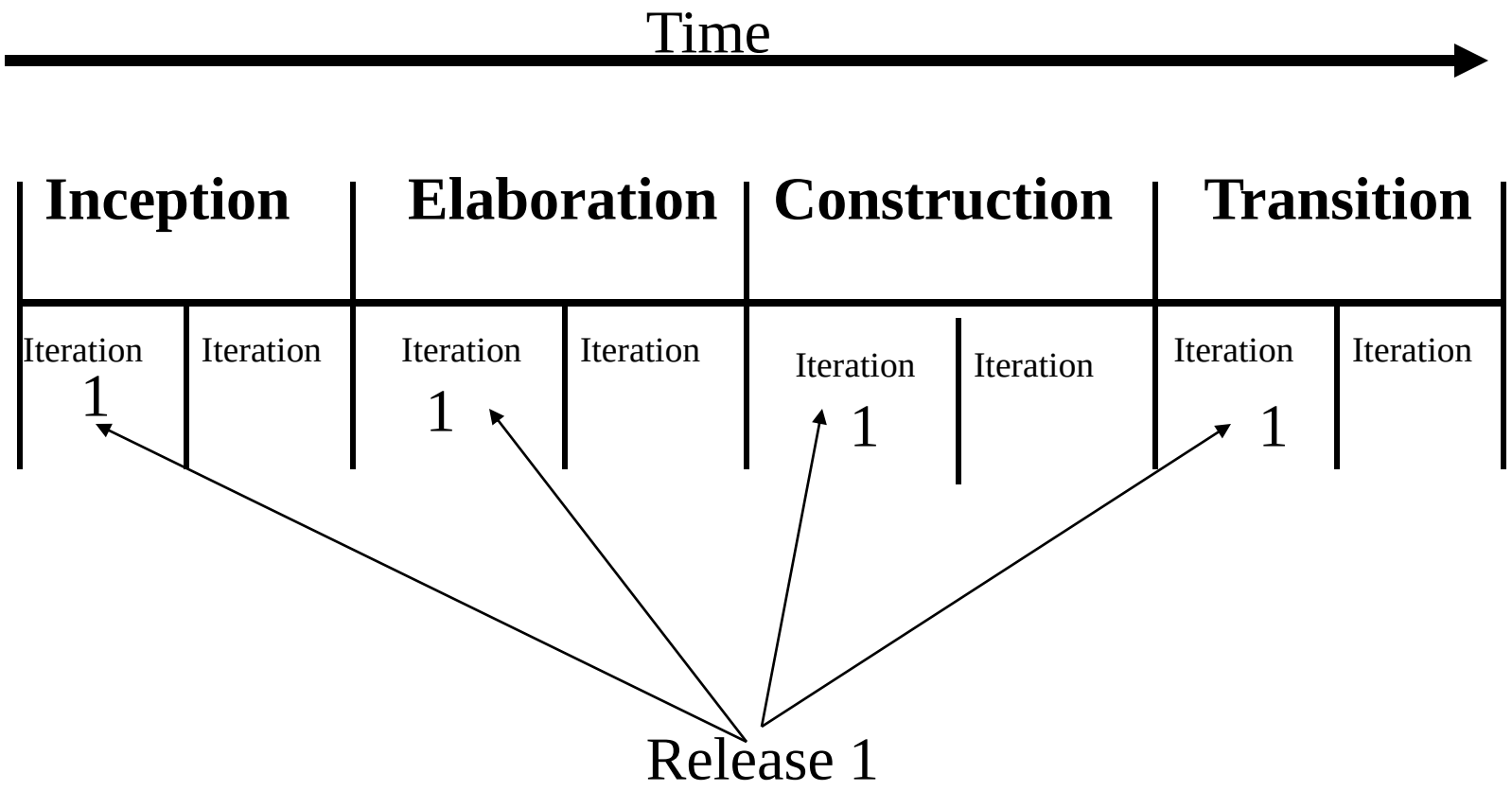
The Unified Process

- The Unified Software Development Process is a definition of a complete set of activities to transform users' requirements through a consistent set of artifacts into a software product
- Look at the whole process
  - Life cycle
  - Artifacts
  - Workflows
  - Phases
  - Iterations
- A process is described in terms of ***workflows*** where a workflow is a set of activities with identified artifacts that will be created by those activities

The Life of the Unified Process

- Unified process repeats over a series of cycles
- Each cycle concludes with a product *release*
- Each cycle consists of four phases:
  - inception- Define the scope of project
  - Elaboration - Plan project, specify features, baseline architecture
  - construction- Build the product
  - transition- Transition the product into end user community

The Life of the Unified Process



A cycle with its phases and its iterations

Inception ➔ Elaboration ➔ …

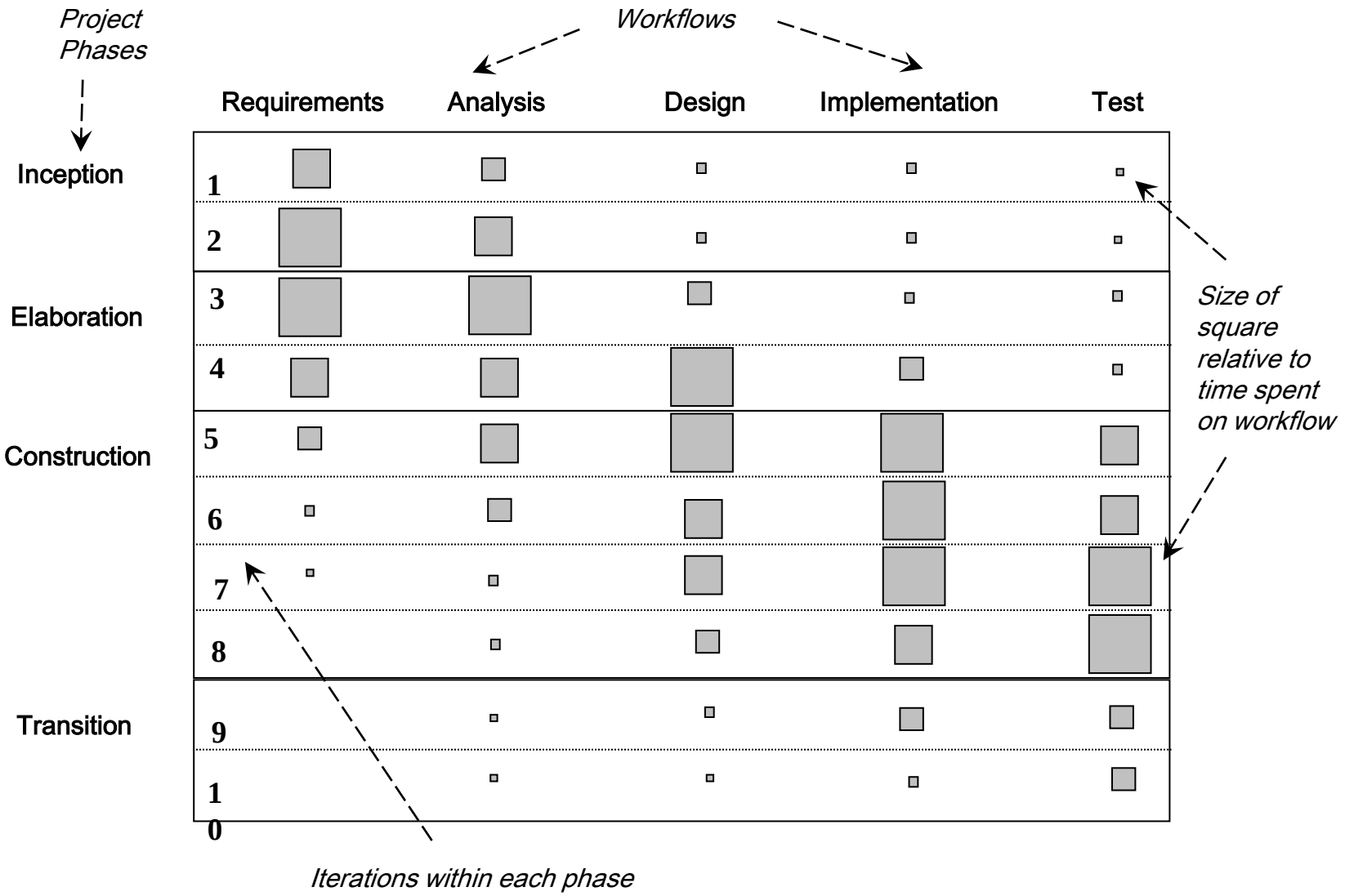
- During **inception**, establish business rationale and scope for project
  - Business case: how much it will cost and how much it will bring in?
  - Scope: try to get sense of size of the project and whether it's doable
  - Creates a *vision and scope document* at a high level of abstraction
- In **elaboration**, collect more detailed requirements and do high-level analysis and design
  - Inception gives you the go-ahead to start a project, elaboration determines the **risks**
    - Requirement risks: big danger is that you may build the wrong system
    - Technological risks: can the technology actually do the job? will the pieces fit together?
    - Skills risks: can you get the staff and expertise you need?
    - Political risks: can political forces get in the way?
  - Develop use cases, non-functional requirements & domain model

… ➔ Construction ➔ Transition

- **Construction** builds production-quality software in many increments, tested and integrated, each satisfying a subset of the requirements of the project
  - Delivery may be to external, early users, or purely internal
  - Each iteration contains usual life-cycle phases of analysis, design, implementation and testing
  - Planning is crucial: use cases and other UML documents
- **Transition** activities include beta testing, performance tuning (optimization) and user training
  - No new functionality unless it's small and essential
  - Bug fixes are OK

UP artifacts

- The UP describes work activities, which result in *work products* called *artifacts*
- Examples of artifacts:
  - Vision, scope and business case descriptions
  - Use cases (describe scenarios for user-system interactions)
  - UML diagrams for domain modeling, system modeling
  - Source code (and source code documentation)
  - Web graphics
  - Database schema


Project Phases
Workflows
Requirements
Analysis
Design
Implementation
Test
Inception
Elaboration
Construction
Transition
1
2
3
4
5
6
7
8
9
10
Size of square relative to time spent on workflow
Iterations within each phase

Unified Process - Milestones

innovate achieve lead

Milestone: a management decision point in a project that determines whether to authorize movement to the next iteration/phase

Inception phase - agreement among customers/developers on the system's life cycle objectives

Elaboration phase - agreement on the viability of the life cycle architecture, business case and project plan

Construction phase - agreement on the acceptability of the software product both operationally and in terms of cost

Transition phase - final agreement on the acceptability of the software product

Timeboxing

Management of a UP project.

Iterations are "timeboxed" or fixed in length.

Iteration lengths of between two to six weeks are recommended.

Each iteration period has its own development plan.

If all the planned activities cannot be completed during an iteration cycle, the completion date should not be extended, but rather tasks or requirements from the iteration should be removed and added to the next iteration cycle.

Best Practices and Key Concepts

- Tackle high-risk and high-value issues in early iterations.
- Continuously engage users for evaluation, feedback and requirements
- Build a cohesive, core architecture in early iterations
- Continuously verify quality; test early, often and realistically
- Apply use cases where appropriate
- Do some visual modeling (with the UML)
- Carefully manage requirements
- Practice change request and configuration management.

UML Introduction

Defining Models and Artifacts

- Objectives
  - analysis and design models
  - familiarize UML notations and diagrams
- Real world software systems are inherently complex
- Models provide a mechanism for decomposition and expressing specifications

What is a Model

- Like a map, a model represents something else
- A useful model has the right level of detail and represents only what is important for the task in hand
- Many things can be modelled: bridges, traffic flow, buildings, economic policy

Why Use a Model?

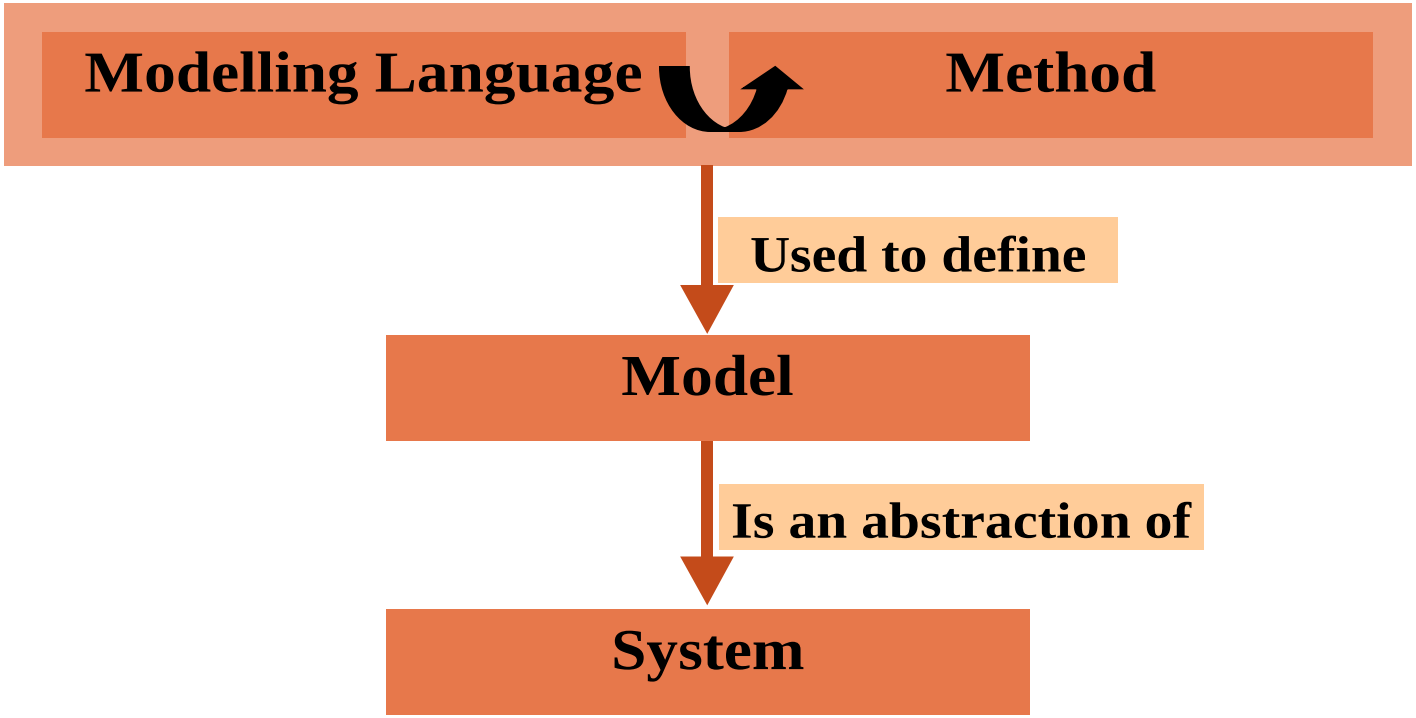
- A model is quicker and easier to build
- A model can be used in a simulation
- A model can evolve as we learn
- We can choose which details to include in a model
- A model can represent real or imaginary things from any domain

Why do we model?

- Provide structure for problem solving
- Experiment to explore multiple solutions
- Furnish abstractions to manage complexity
- Reduce time-to-market for business problem solutions
- Decrease development costs
- Manage the risk of mistakes

Some Terminology

- Model: A description or analogy used to help visualize something that cannot be directly observed. A model is an abstraction describing a system or a subset of a system
- Method: A means or manner of procedure; especially, a regular and systematic way of accomplishing anything.
- A view depicts selected aspects of a model
- A notation is a set of graphical or textual rules for representing views

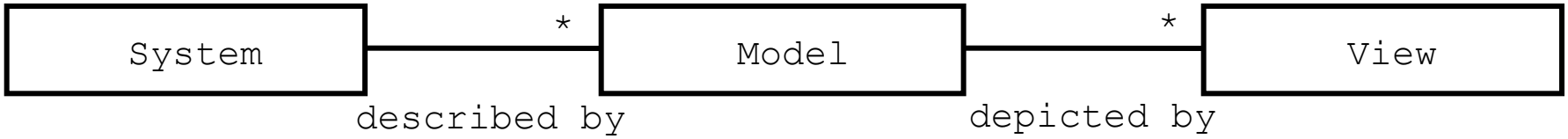


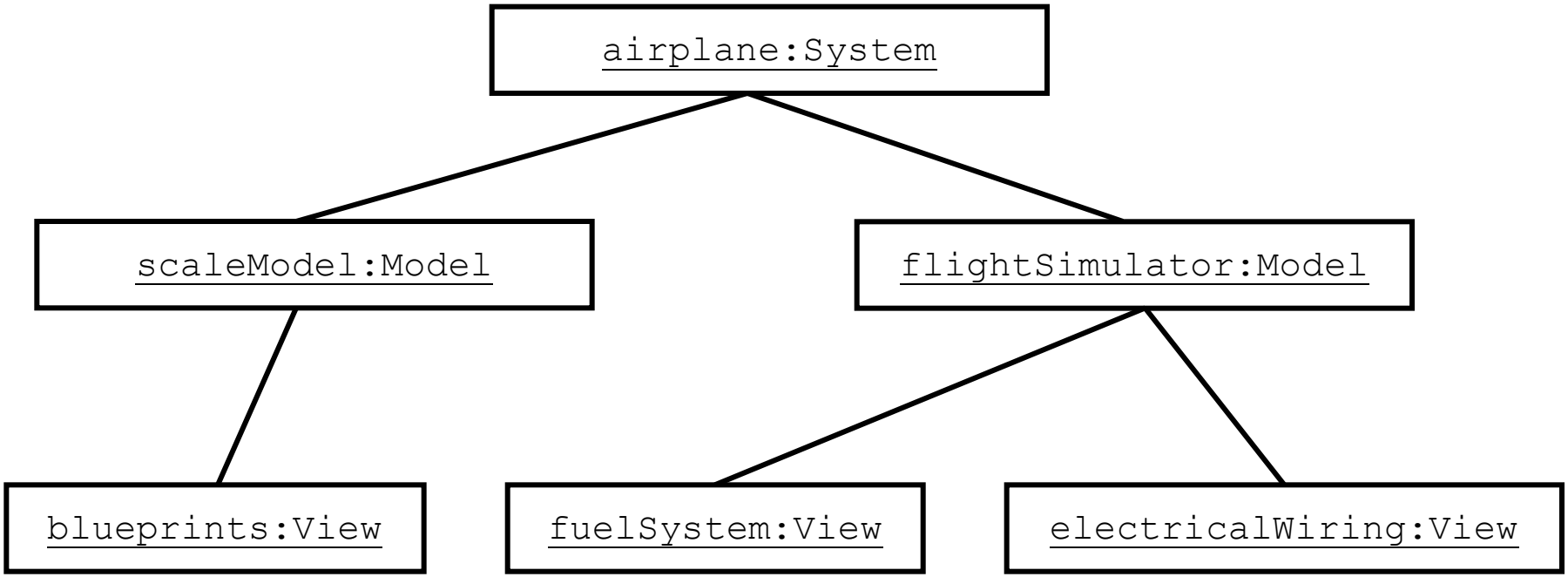
- A system is a collection of subsystems organized to accomplish a purpose and described by a set of models.
- A model is an abstraction of a system.

Examples

System: Aircraft
Models: Flight simulator, scale model
Views: All blueprints, electrical wiring, fuel system

Models, Views, and Systems as UML





Object-Oriented Modeling

- Why models?
  - To abstract reality
    - Show essential details; filter out non-essential details
  - To help us deal with complexity
    - Deal with human limitations in understanding complex things
  - To allow us to focus on the "big picture"
  - To promote understanding of requirements, cleaner design, more maintainable systems
- Why objects?
  - To more accurately reflect reality in a model
  - To reduce the "semantic gap" between reality and a model
  - To localize changes to the model
  - We can model reality as a number of interacting objects!

Object-Oriented Modeling Perspectives

Conceptual

- Do not consider any aspects of implementation of objects
- Focus on identifying the objects in the problem domain

Specification

- Consider interfaces of objects (but no internal aspects)
- Focus on how objects interact in the solution domain

Implementation

- Consider all details of objects (external and internal)
- Focus on how to code objects

Modeling proceeds from the conceptual to implementation

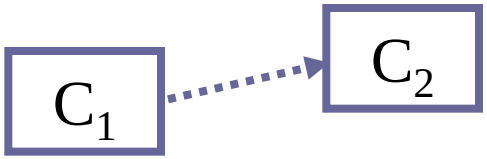
Same OO concepts can be used at all levels to build models

Analysis and Design models

- Analysis model - models related to an investigation of the domain and problem space (Use case model qualifies as an example)
- Design model - models related to the solution (class diagrams qualifies as an example)

Elements of Modelling Language

- Symbols: Standard set of symbols
- Syntax: Acceptable ways of combining symbols
- Semantics: Meaning given to language expressions



C_1 sends a message to C_2

Advanced Properties

- Expressiveness: What the language can say

OK: C_1 sends messages to C_2
Not OK: C_1 sends messages to C_2, after all messages of C_2 were recieved

- Methodology: Procedures to be followed

1. Model all classes
2. Model all relations
3. Model all inheritance

- Guidelines: Suggestions on how to build effective models

Try to model classes with a balanced number of associations

What UML is *NOT*

- UML is NOT a methodology
- UML is NOT a process
- UML is NOT proprietary (Now under the OMG)

UML *is strictly* Notations

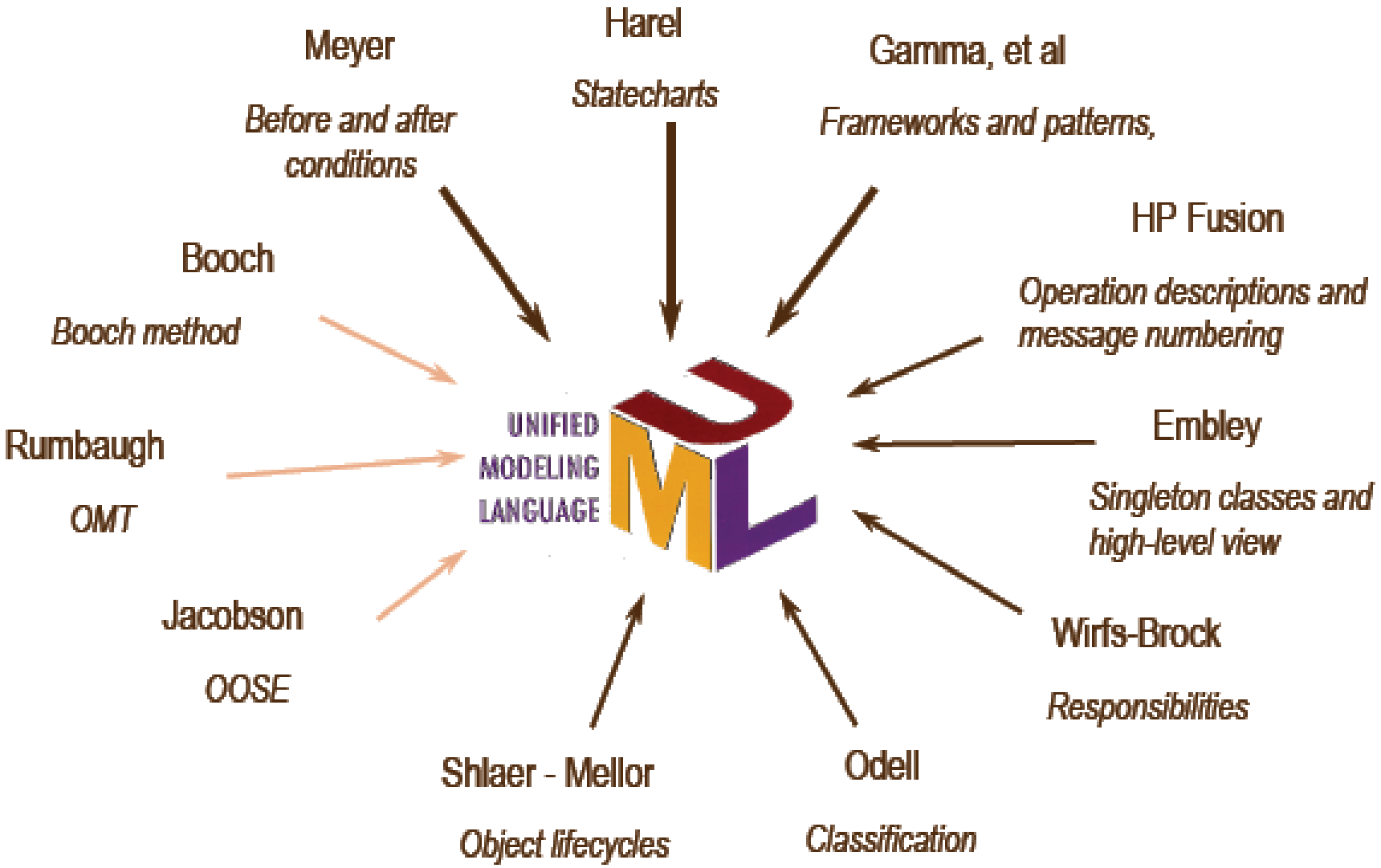
A collection of diagrams for system visualization of
- Software structure
- Behavior
- Physical organization

What is UML?

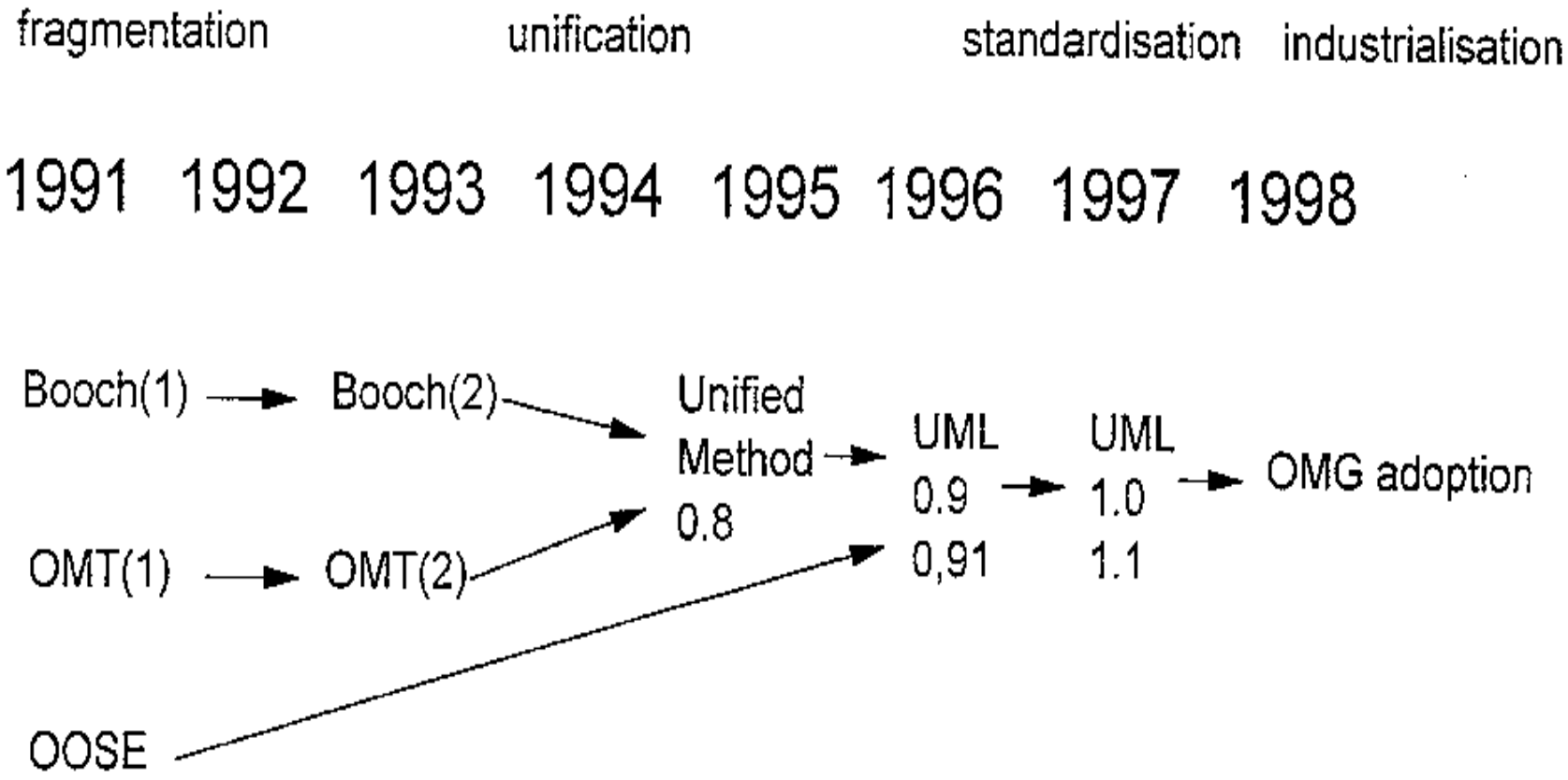
- The UML is a graphical language for
  - specifying
  - visualizing
  - constructing
  - documenting

  the artefacts of software systems
- Added to the list of OMG adopted technologies in November 1997 as UML 1.1 (Current Version is 2.0)
- To end the OO method wars
  - Lack of standardization
- Has become the formal and de facto standard

UML is ... unified



History of UML



General Goals of UML

- Model systems using OO concepts
- Establish an explicit coupling to conceptual as well as executable artifacts
- To create a modeling language usable by both humans and machines
- Models different types of systems (information systems, technical systems, embedded systems, real-time systems, distributed systems, system software, business systems, UML itself, ...)

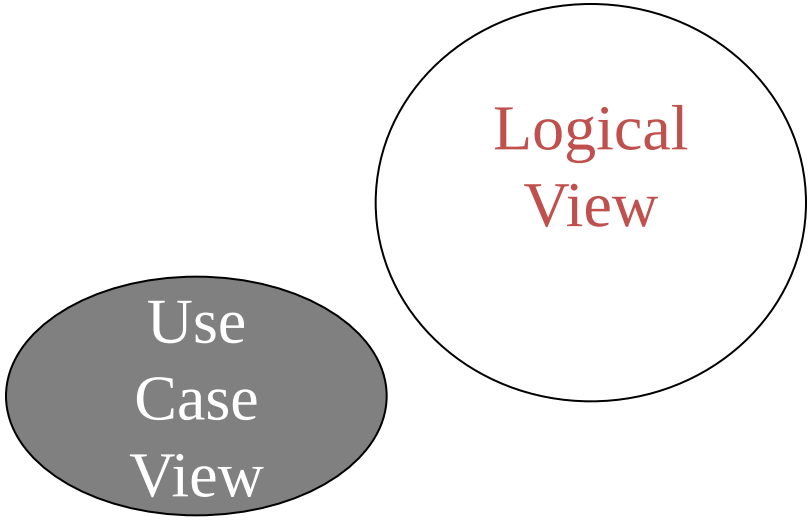
Parts of UML

- Views - shows different aspects of the system that are modeled, links the modeling language to the method/process chosen for development
- Diagrams - graphs that describe the contents in a view
- Model elements - concepts used in a diagram

Views in UML

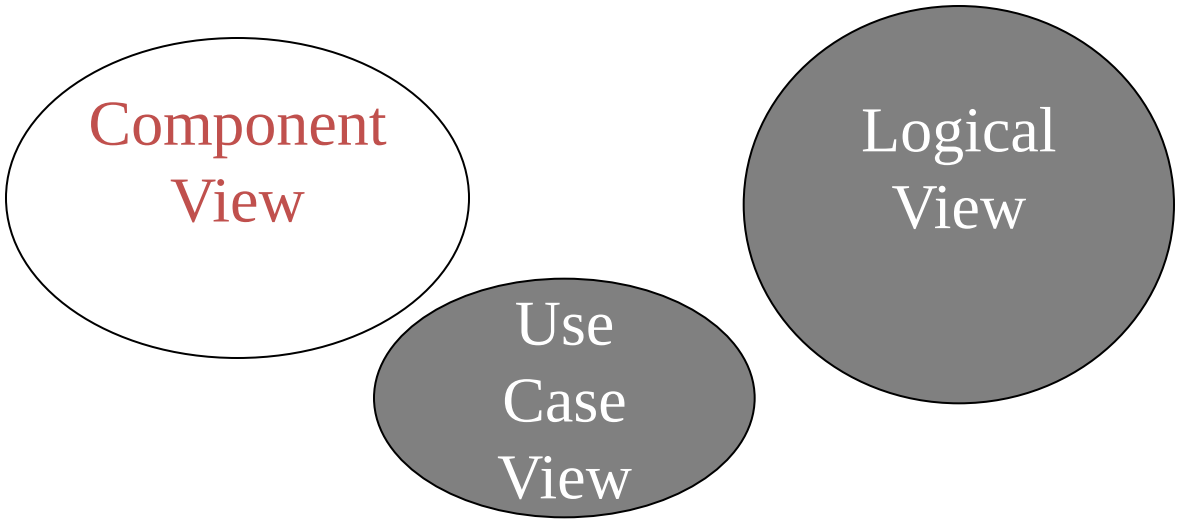
- Use-case view : *A view showing the functionality of the system as perceived by the external actors*

Views in UML



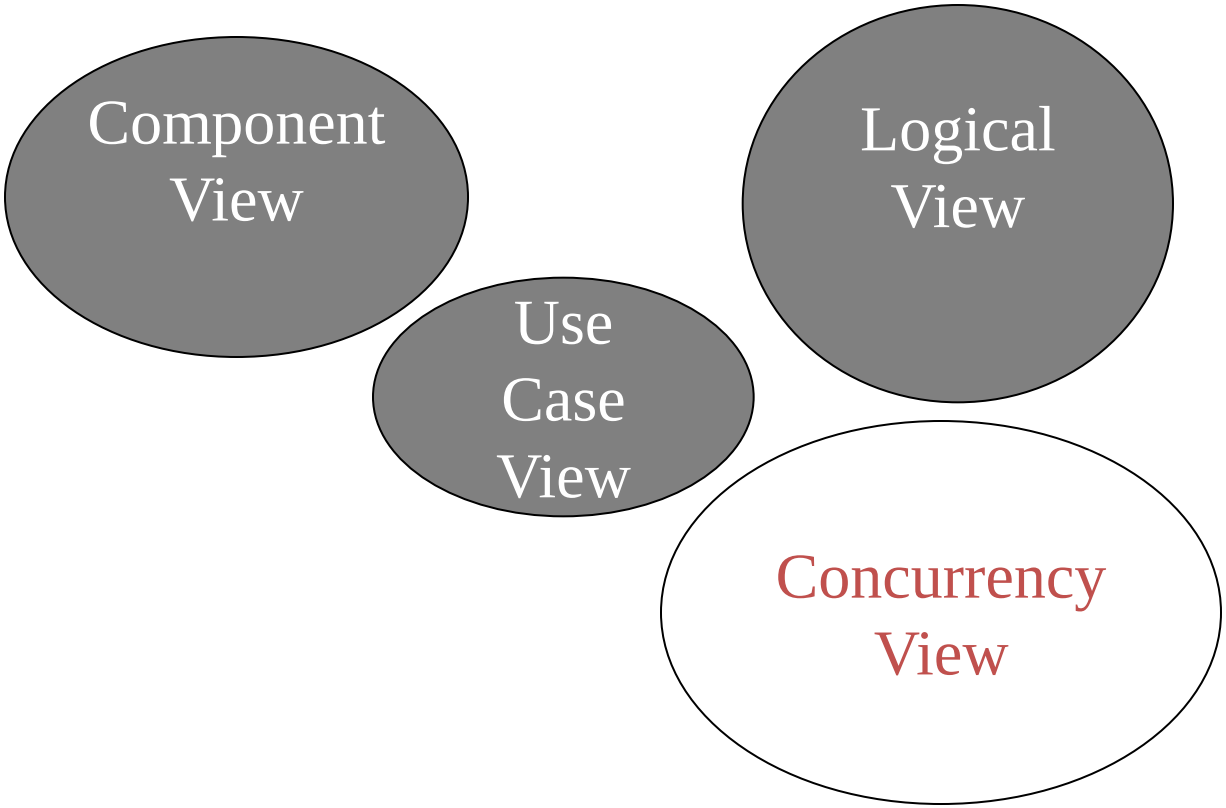
- Logical view: *A view showing how the functionality is designed inside the system, in terms of the static structure and dynamic behavior.*

Views in UML



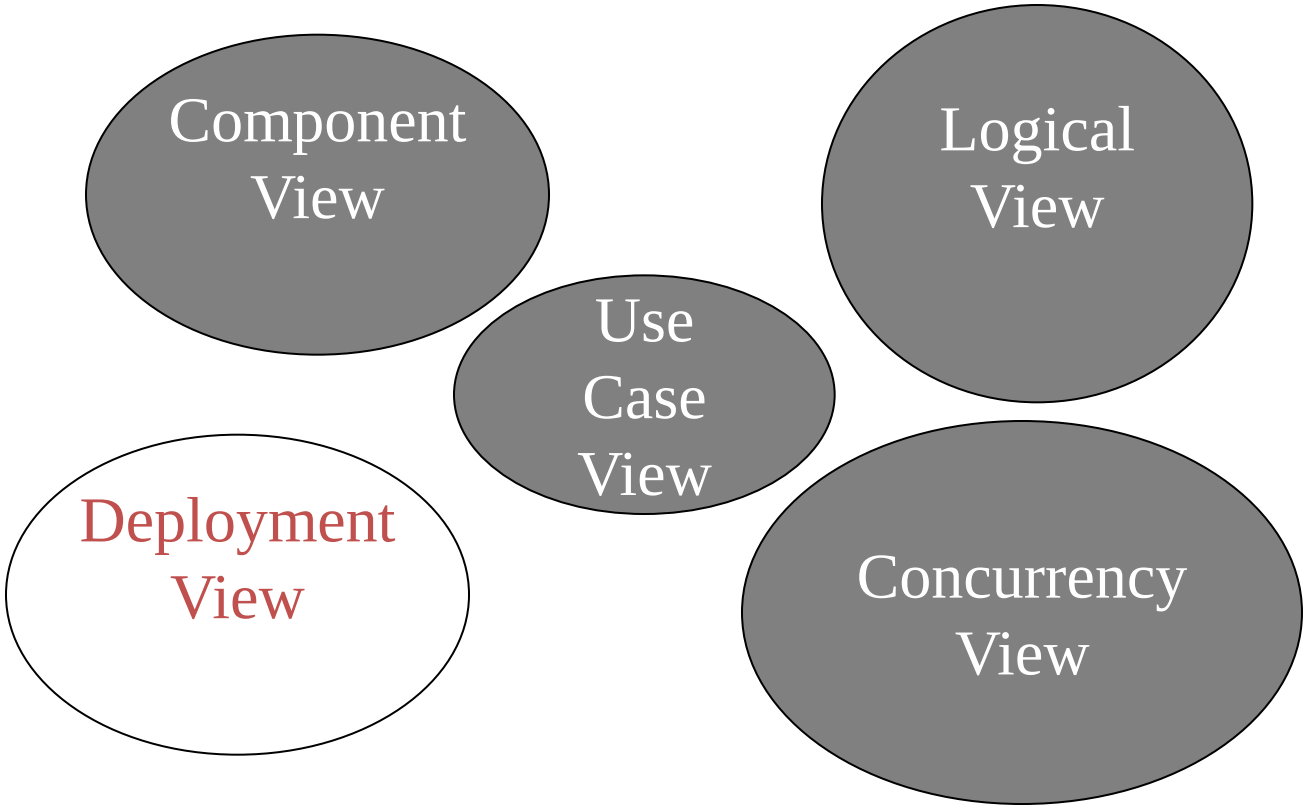
- Component view: *A view showing the organization of the core components*

Views in UML



- Concurrency view: *A view showing the concurrency of the system*

Views in UML



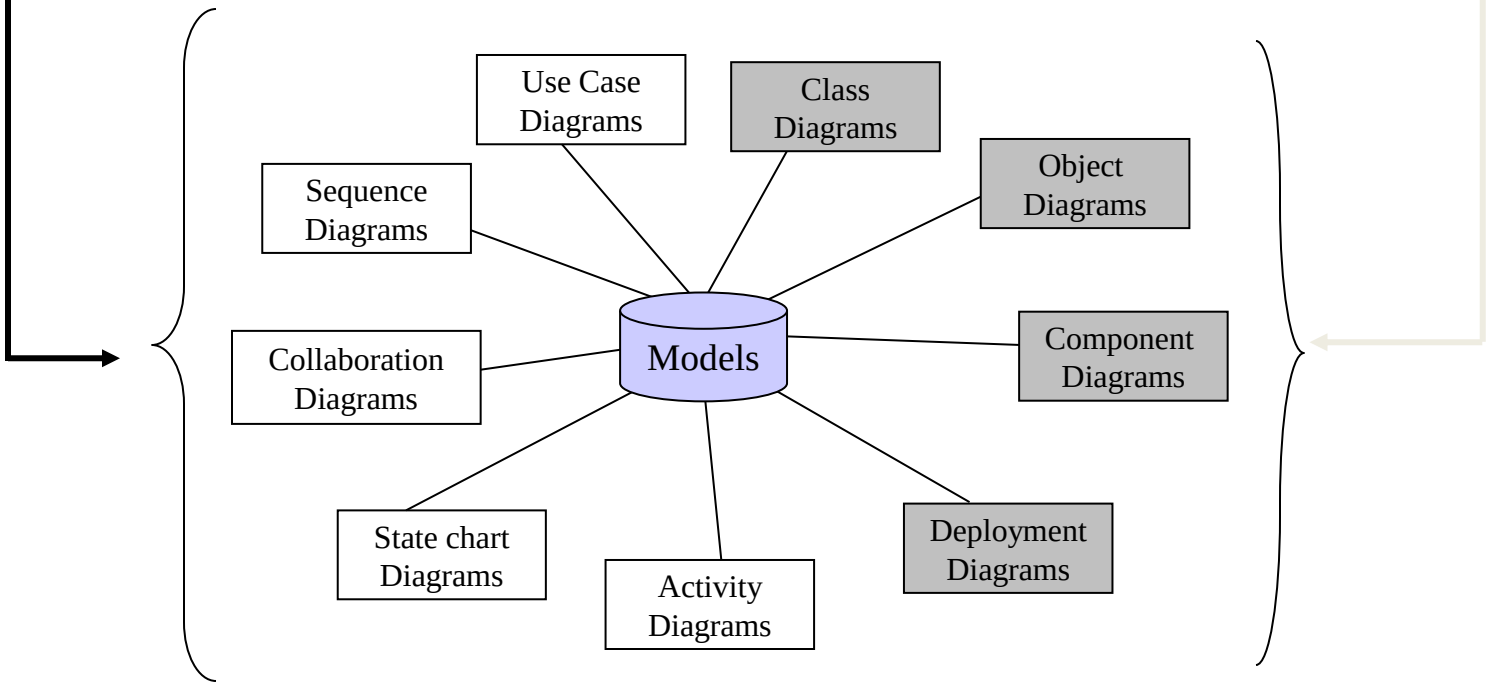
- Deployment view: *A view showing the deployment of the system in terms of the physical architecture*

Introduction to UML

- Model elements
  - Class
  - Object
  - State
  - Use case
  - Interface
  - Association
  - Link
  - Package ….

UML Diagrams

- Two types of diagrams:
  - Structural diagrams
  - Behavioural diagrams



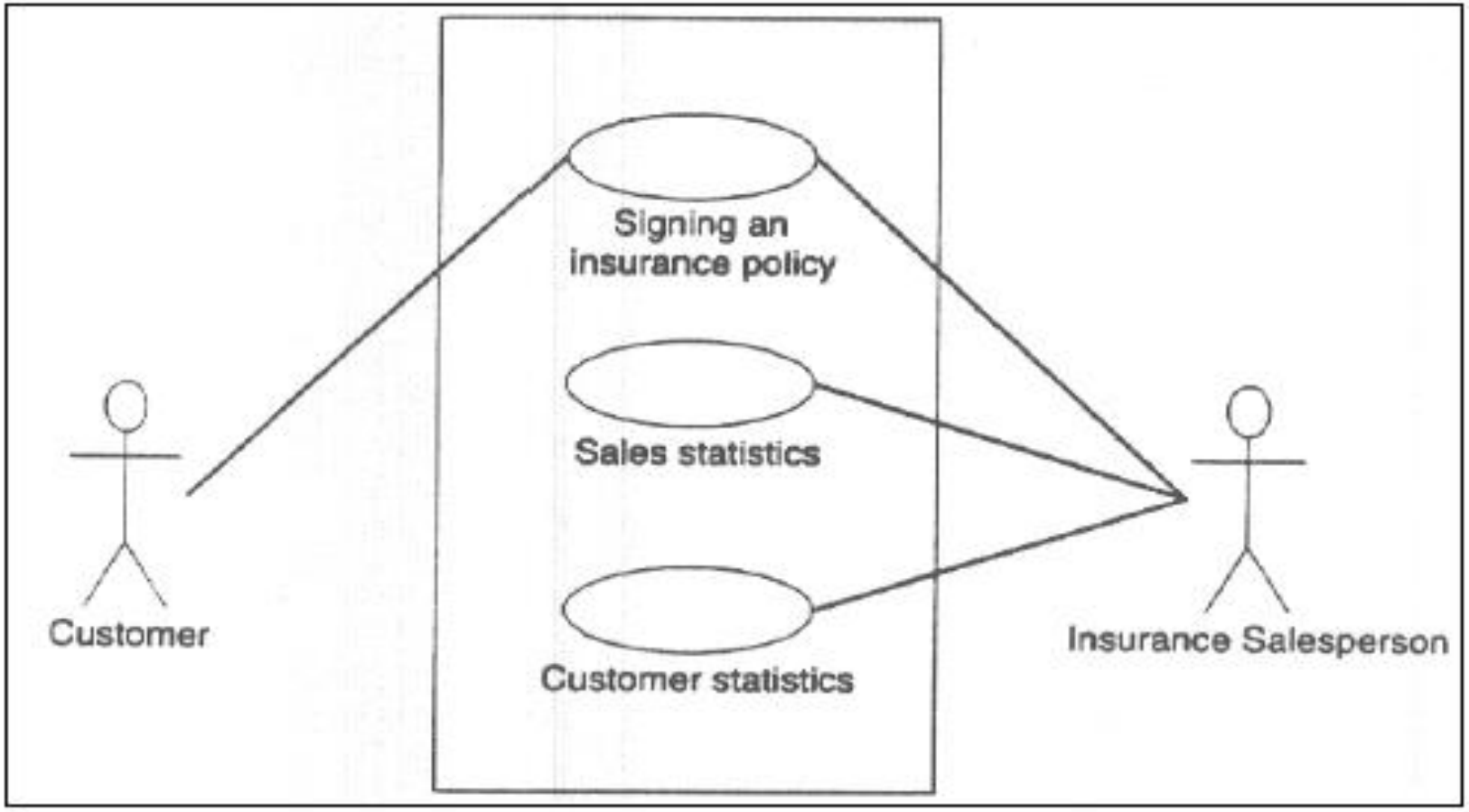
UML diagrams

- Use Case diagram: External interaction with actors
- Class/Object Diagram : captures static structural aspects, objects and relationships.
- State Diagram: Dynamic state behavior
- Sequence diagram: models object interaction over time
- Collaboration diagram: models component interaction and structural dependencies

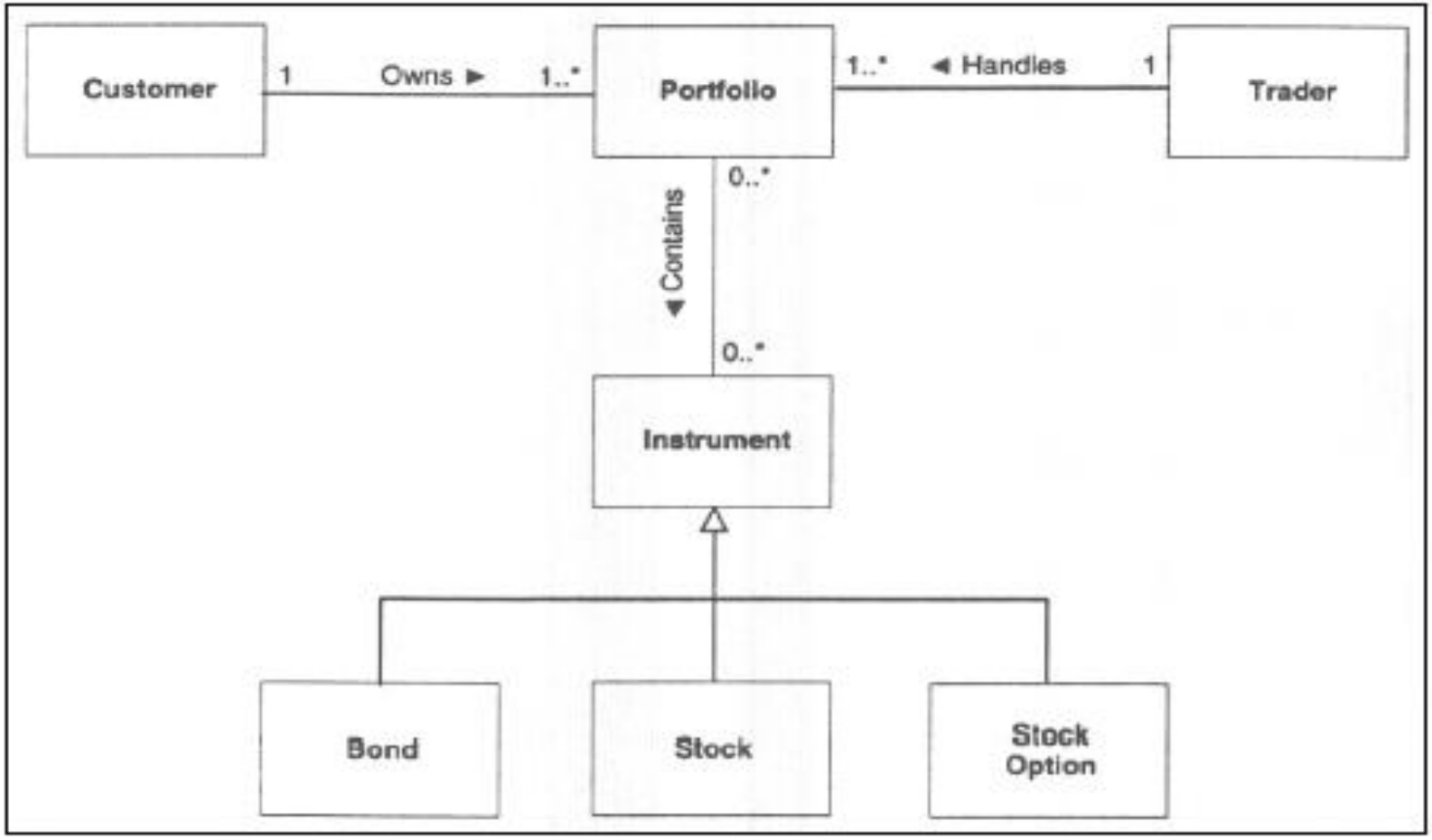
More UML diagrams

- Activity diagram: models object activities
- Deployment diagram: models physical architecture
- Component diagram: models software architecture

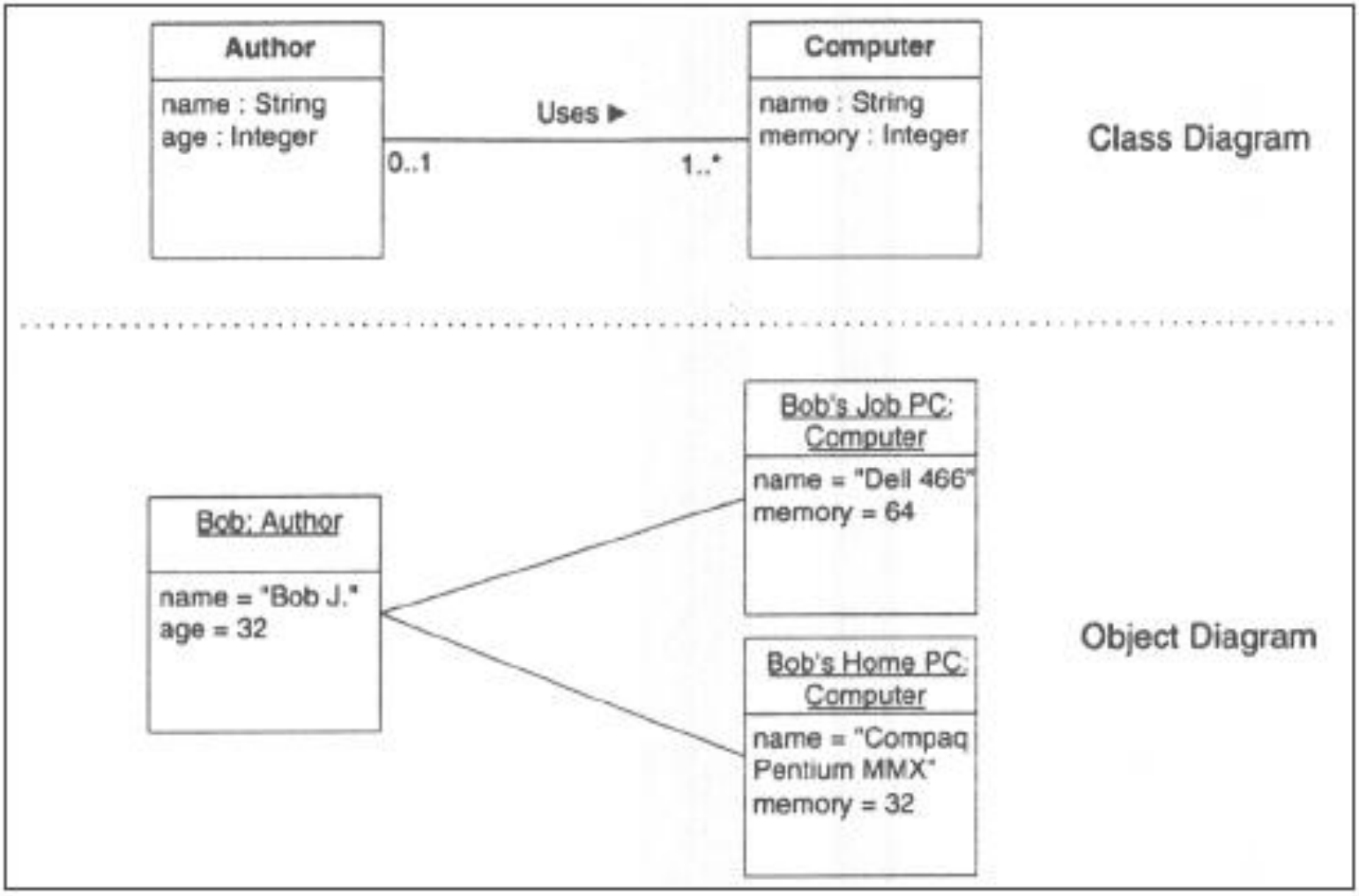
Use-Case Diagram



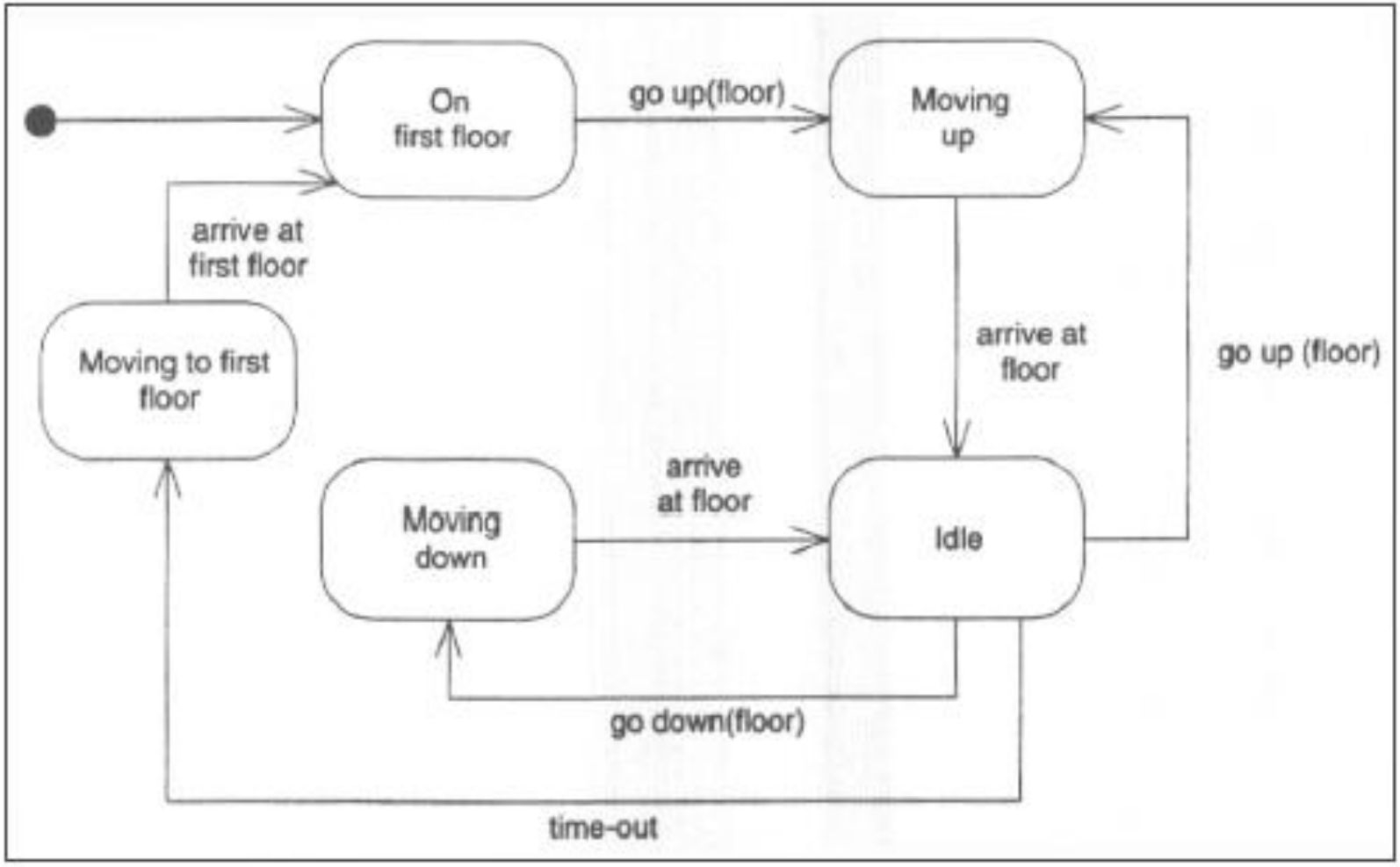
Class Diagram



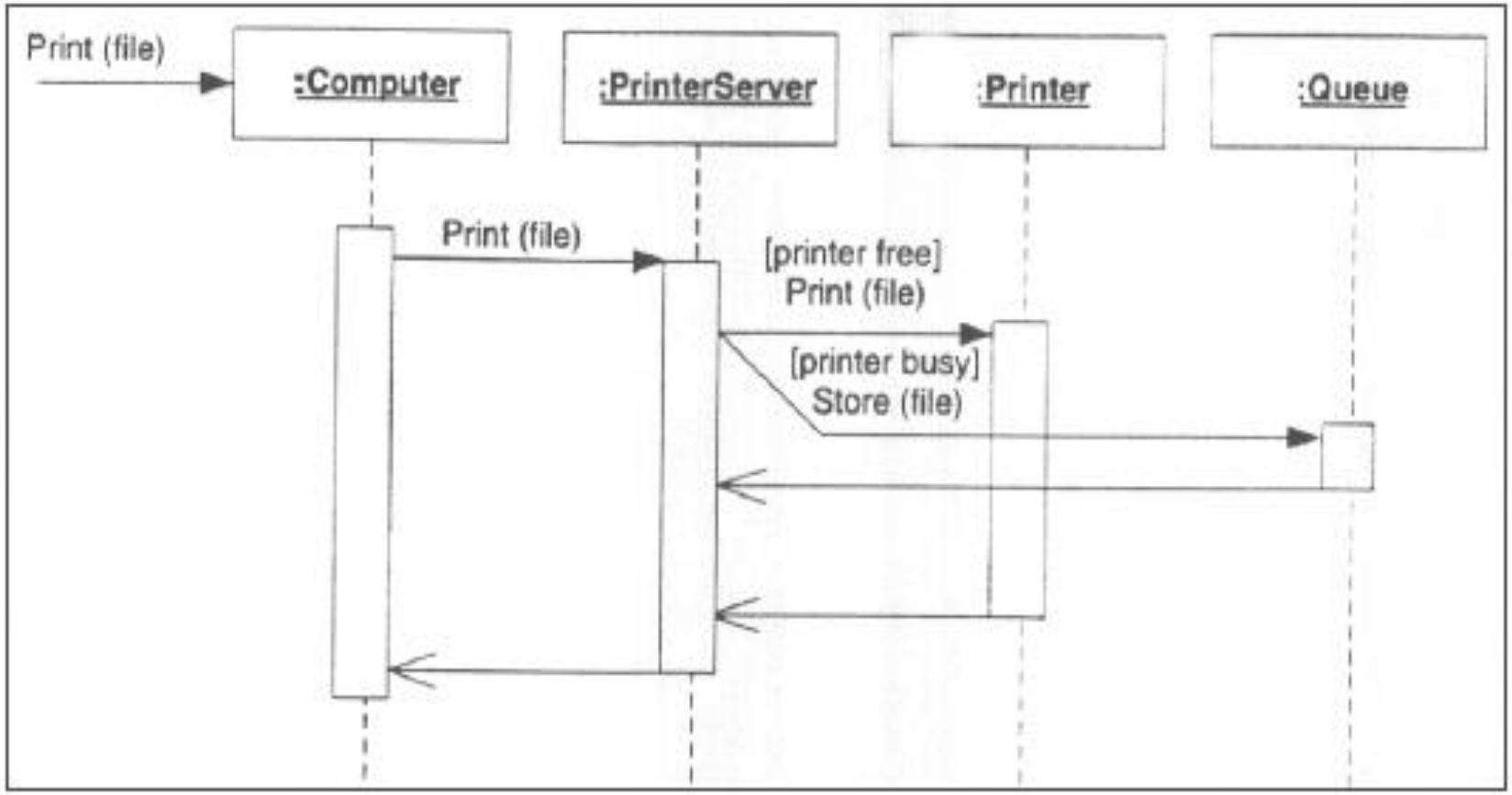
Object Diagram



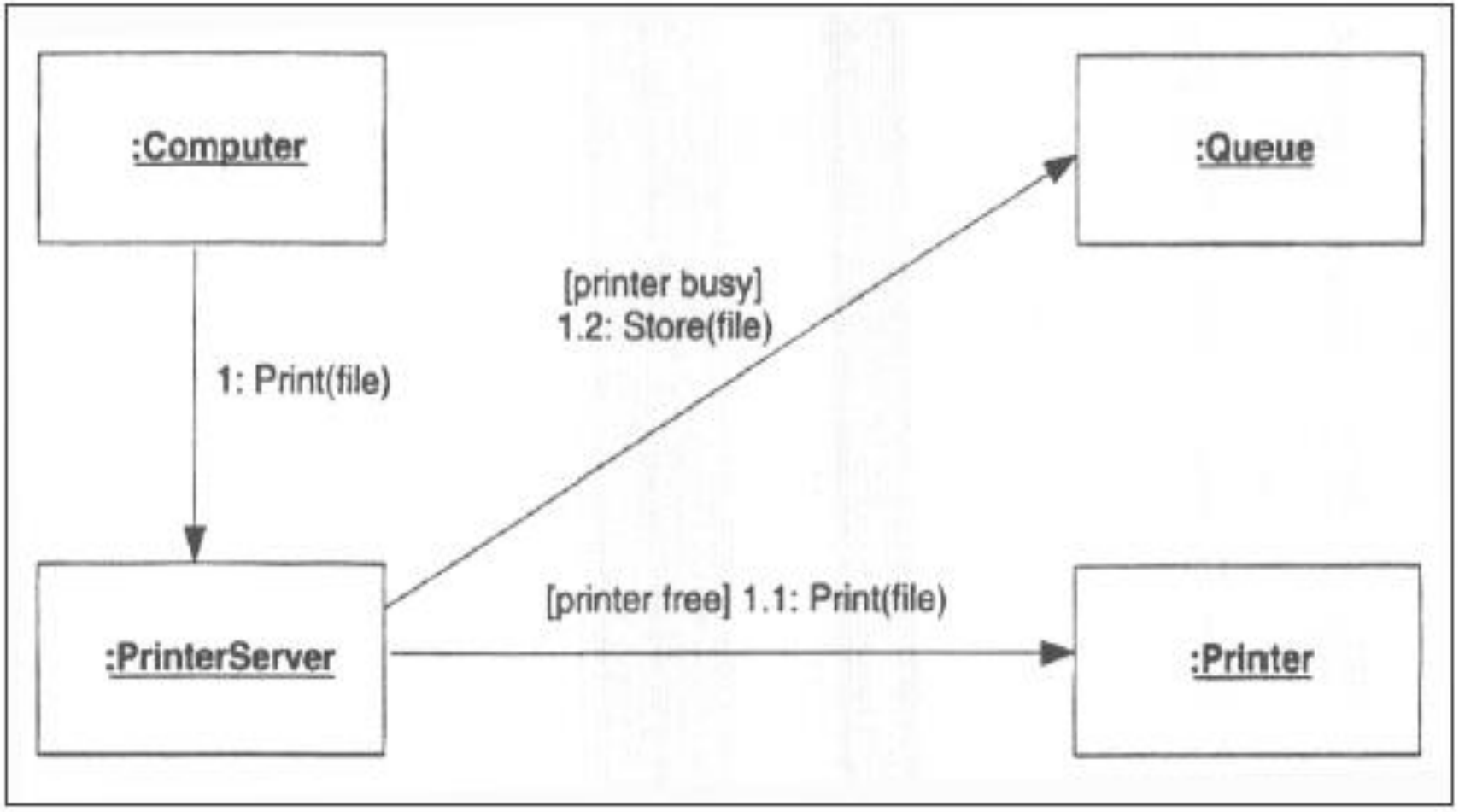
State Diagram



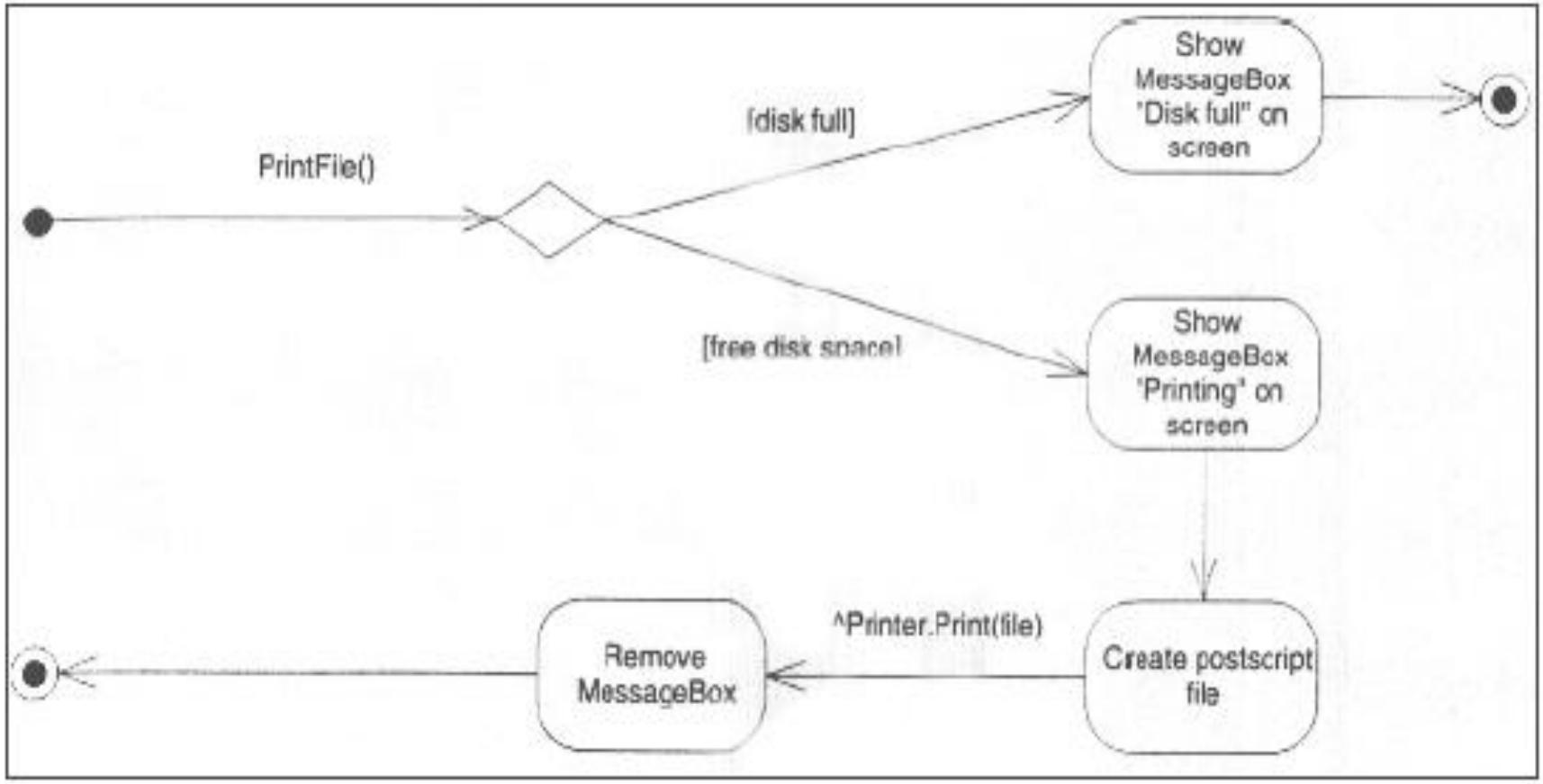
Sequence Diagram



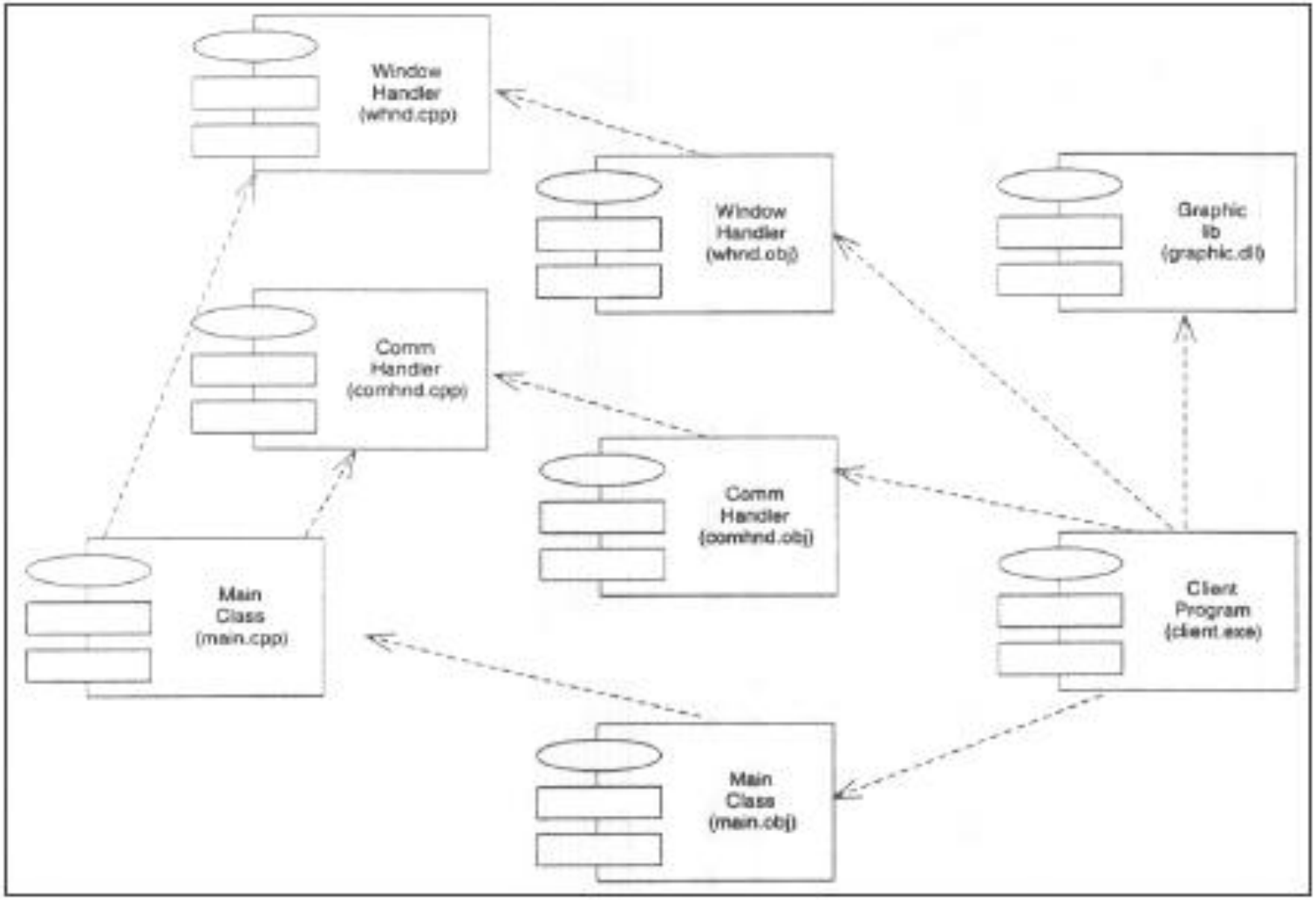
Collaboration Diagram



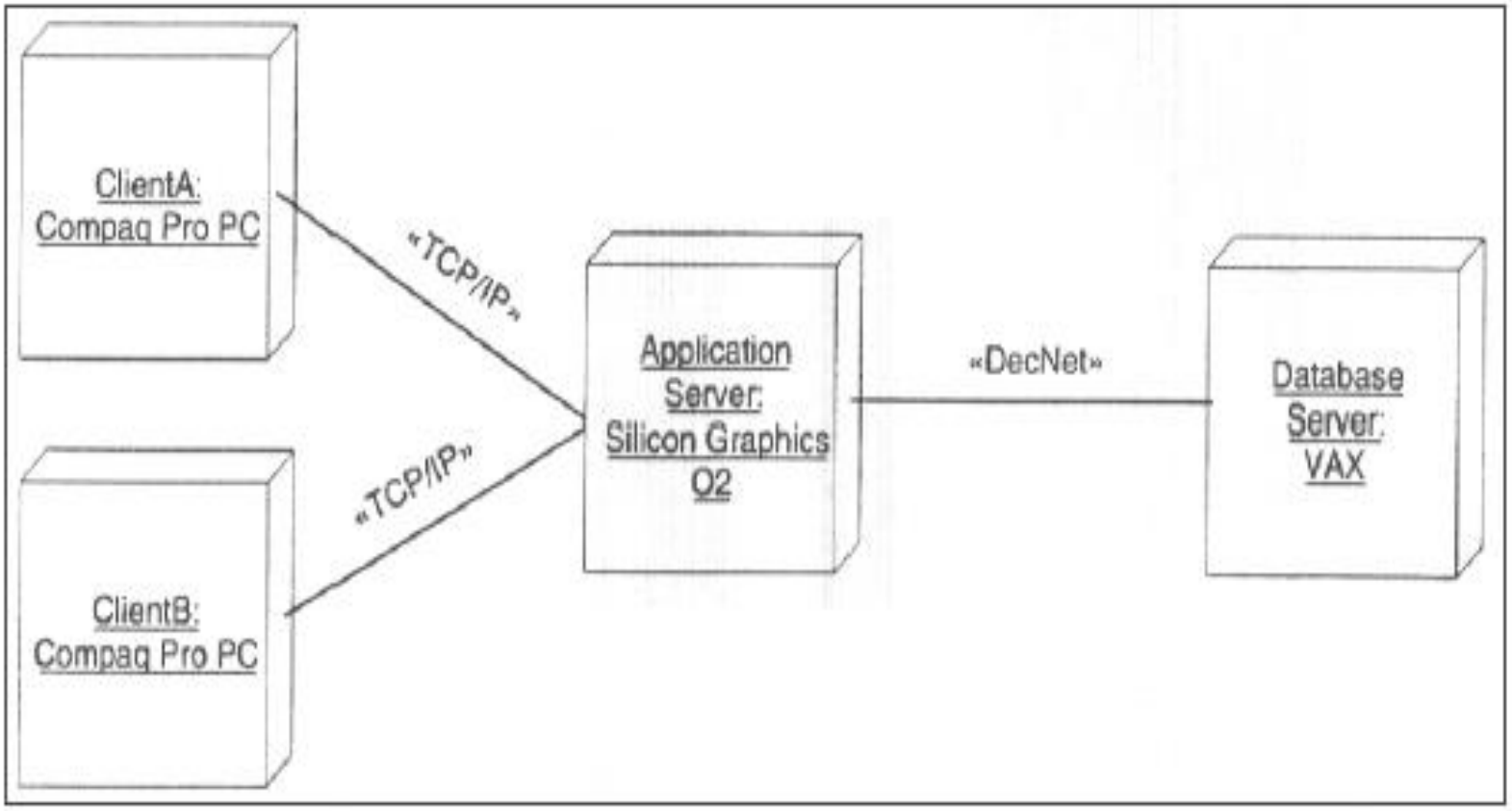
Activity Diagram



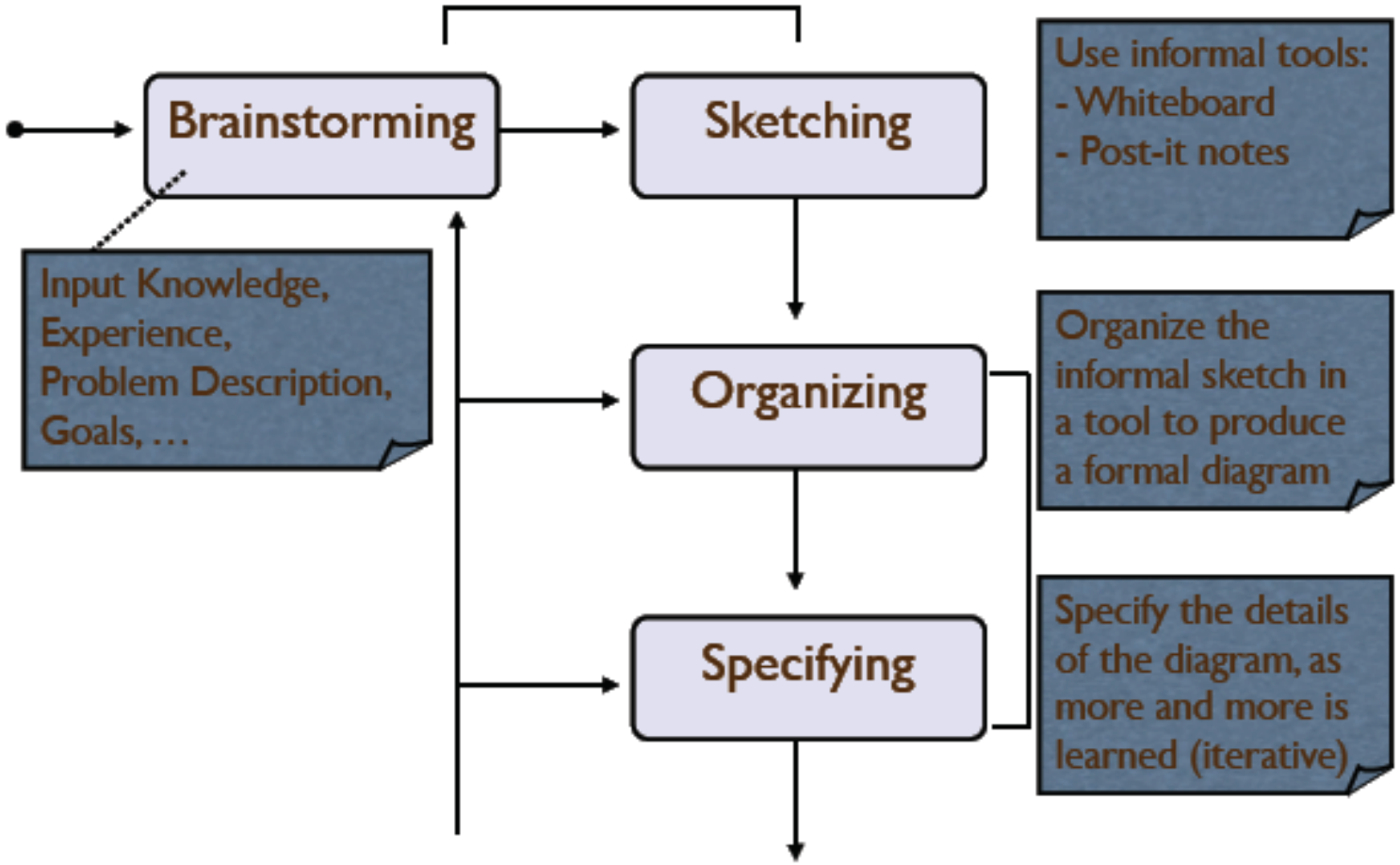
Component Diagram



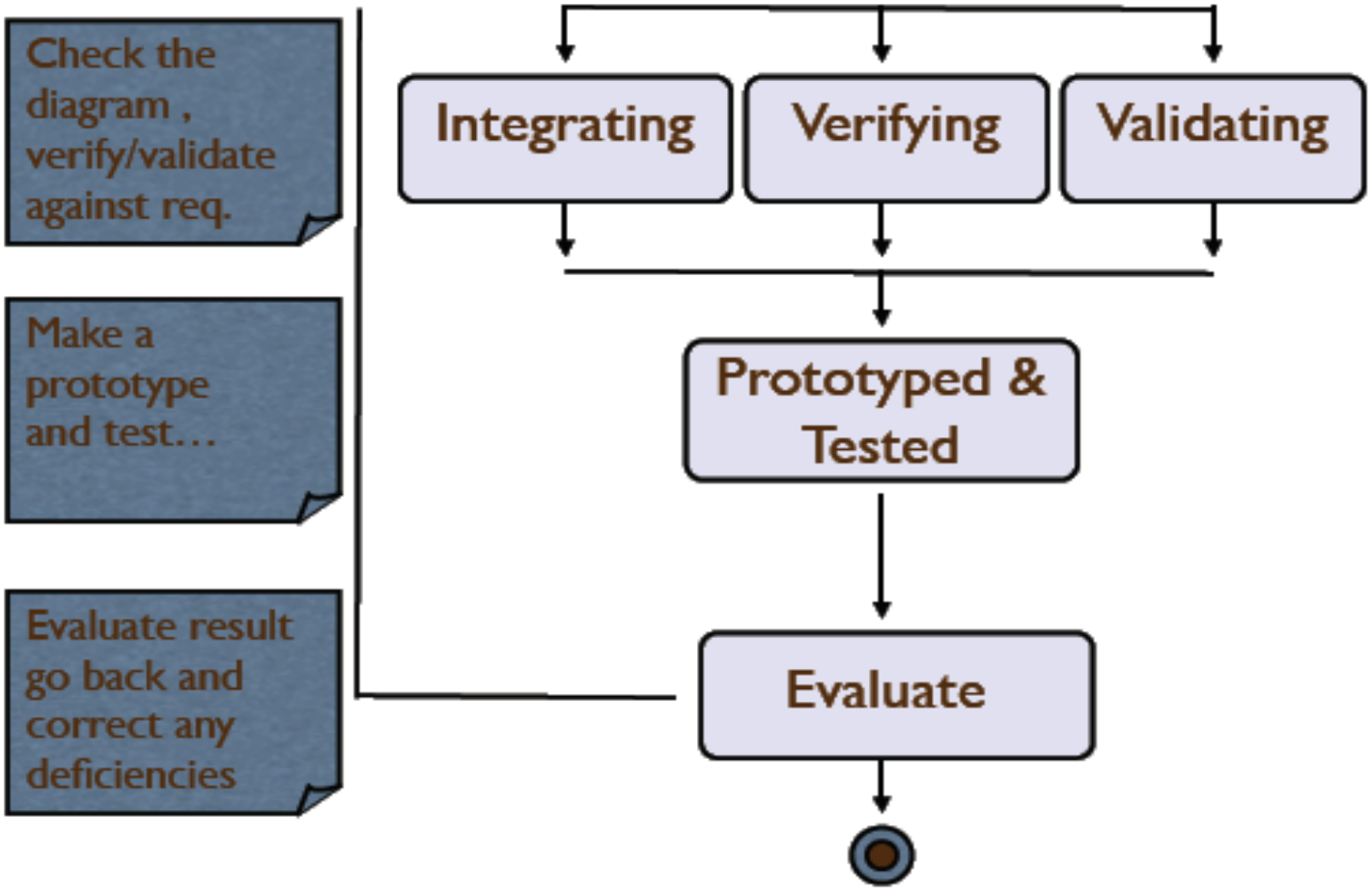
Deployment Diagram



A Process for Making Models...



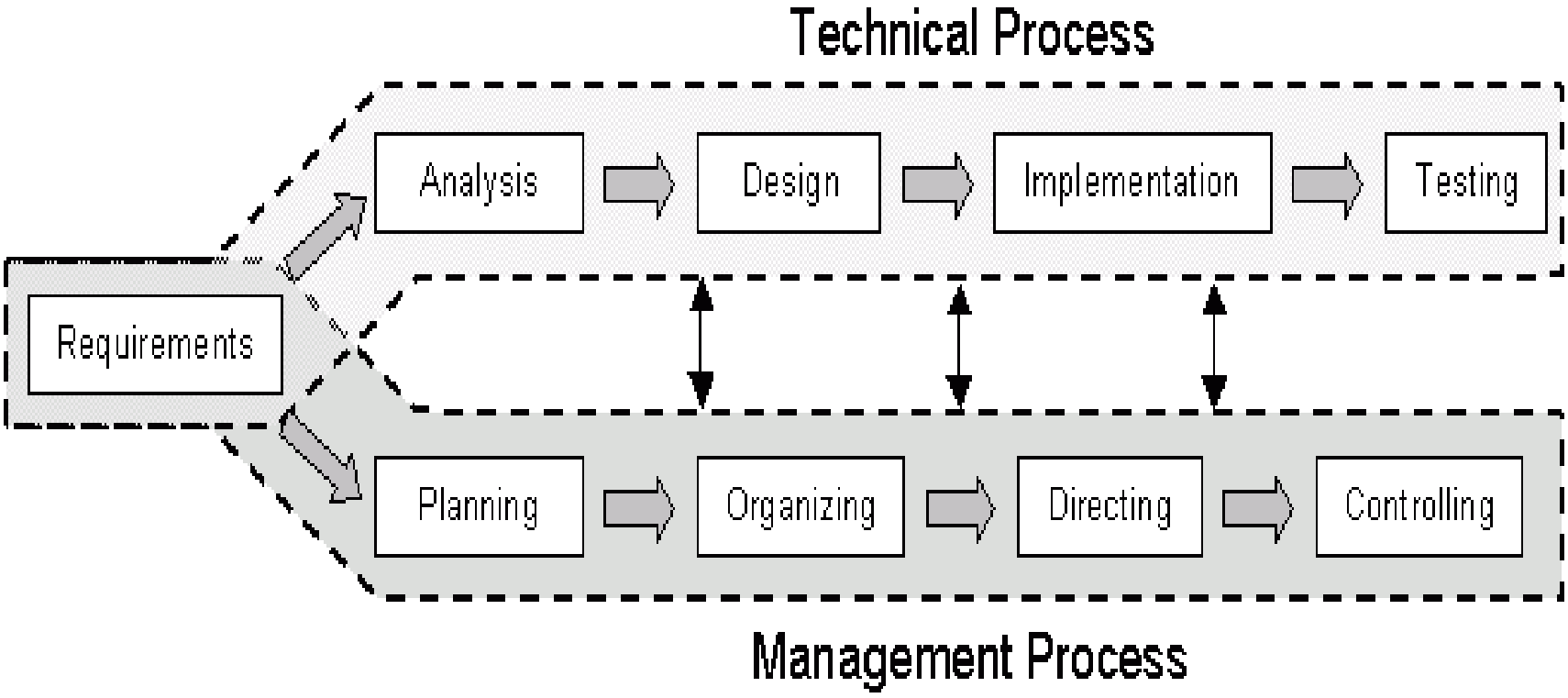
A Process for Making Models...



Requirements Engineering

- "The hardest single part of building a software system is deciding what to build. ..No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later."

Requirements drive the technical and management processes



What is a Software Requirement

- [IEEE 90] defines a requirement as:

  (1) A condition or capability needed by a user to solve a problem or achieve an objective.
  (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
  (3) A documented representation of a condition or capability as in (1) or (2).

What is this Phase For?

- Major misconception
  - determining what client wants

"I know you believe you understood what you think I said, but I am not sure you realize that what you heard is not what I meant!"

- Must determine client's & **user's** needs
  - chances for success slim if you don't figure this out!

Requirements-What are They?

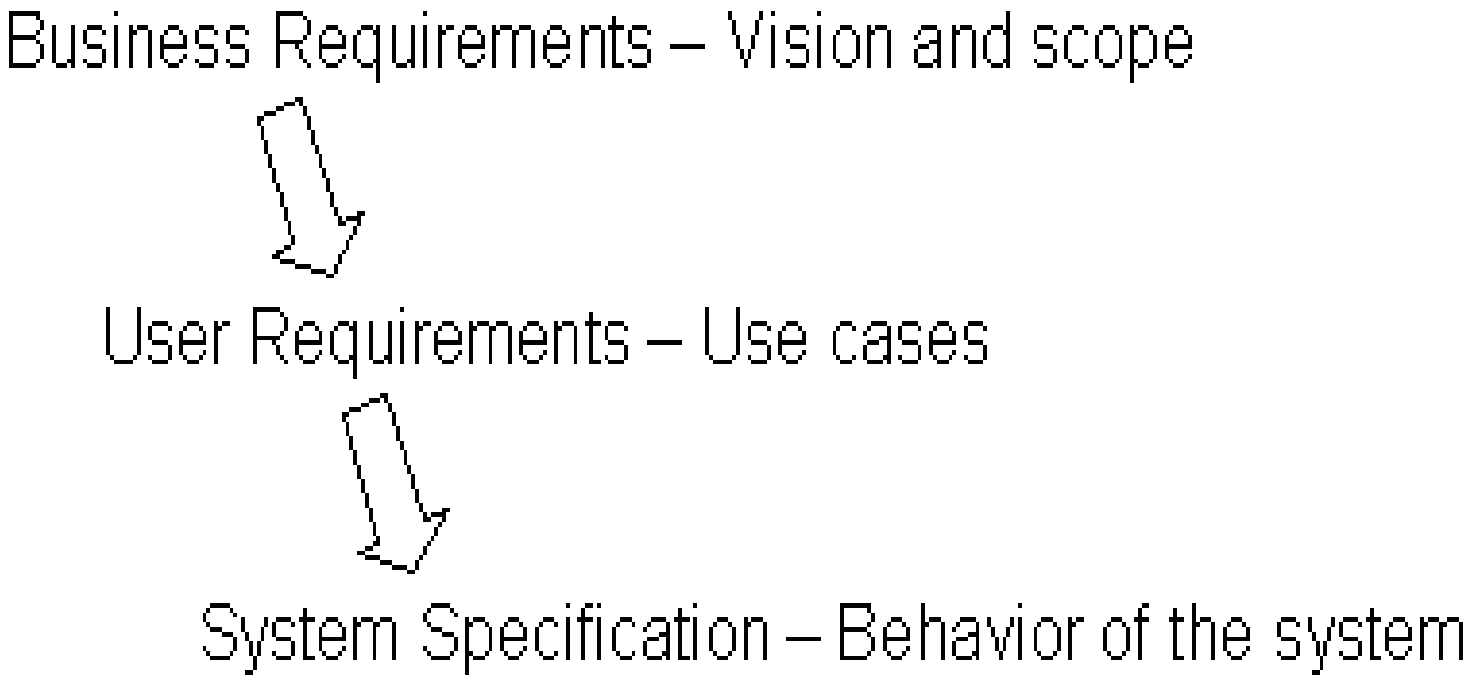
- A requirement is a description of a system feature, capability, or constraint.
- Classes of Requirements:
  - functional
  - nonfunctional (constraints)
- Requirements Priority
  - essential (“shalls”)
  - highly desirable (“shoulds”)
  - desirable but low priority

Types of Requirements

- Physical environment
- Interfaces
- Users and human factors
- functionality
- performance
- documentation/training
- data

- resources
- security
- reliability
- portability
- maintenance
- etc.

Levels of Requirements



Levels of Requirements

- ***Business requirements*** - objectives of the customer or user of the system
- ***User requirements*** - tasks the user needs to accomplish with the system.
- ***System specification*** (functional requirements) - system behavior that will allow users to perform their tasks.

Levels of Requirements-Examples

Business requirement = Hospital needs to get drug information in the hands of pharmacists and doctors so they can make better decisions about which drugs to prescribe for patients.

User Requirement = The pharmacist should be able to enter a drug name and get back information about side affects, dosage and drug interactions.

System specification = The system will keep a database of drug names including scientific name, generic name and brand name. The system will return all drug names that match a full or partial name. If more than one drug name matches the user will have the option of selecting one of the drugs in the list.

THANK YOU