COMP 3700: Software Modeling and Design

(Introduction to OO Analysis & Design)

Course Materials, Syllabus, Schedule

- Course Description (Objectives, Textbooks, Grading, Course Requirements), Class Schedule, Course Outline and Resources.
- Topics and Guidelines for Design Projects
- Schedule

Topics

- Computational Thinking
- Why Software Design?
- Software Engineering Life Cycles
  - Traditional Life Cycles
  - The Unified Process (UP)
- Structured vs. OO Design
- Visual Modeling with UML

Computational Thinking

- Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone.

Computational Thinking

- Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science.

Computational Thinking

- Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation.

Computational Thinking

- Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system.
- It is separation of concerns.
- It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable.

Computational Thinking

- It is using invariants to describe a system's behavior succinctly and declaratively. It is having the confidence we can safely use, modify, and influence a large complex system without understanding its every detail

Computational Thinking

- Computer science is the study of computation—what can be computed and how to compute it.
- Computational thinking thus has the following characteristics:

Computational Thinking

- ***Conceptualizing***, **not programming. Computer science is not computer programming. Thinking like a computer scientist means more than being able to program a computer. It requires thinking at multiple levels of abstraction**

Computational Thinking

- ***Fundamental, not rote skill.*** **A fundamental skill is something every human being must know to function in modern society. Rote means a mechanical routine**

Computational Thinking

- A way that humans, not computers, think.

  Computational thinking is a way humans solve problems; it is not trying to get humans to think like computers. Computers are dull and boring; humans are clever and imaginative.

Computational Thinking

- ***Complements and combines mathematical and engineering thinking.***
- **Computer science inherently draws on mathematical thinking, given that, like all sciences, its formal foundations rest on mathematics.**

Computational Thinking

- Computer science inherently draws on engineering thinking, given that we build systems that interact with the real world.
- The constraints of the underlying computing device force computer scientists to think computationally, not just mathematically.
- *Being free to build virtual worlds enables us to engineer systems beyond the physical world;*

Computational Thinking

- ***Ideas, not artifacts*. It's not just the software and hardware artifacts we produce that will be physically present everywhere and touch our lives all the time, it will be the computational concepts we use to approach and solve problems, manage our daily lives, and communicate and interact with other people**

Computational Thinking

- **Intellectually challenging and engaging scientific problems remain to be understood and solved. The problem domain and solution domain are limited only by our own curiosity and creativity.**

Computational Thinking

- One can major in computer science and do anything. One can major in English or mathematics and go on to a multitude of different careers. Ditto computer science.
- One can major in computer science and go on to a career in medicine, law, business, politics, any type of science or engineering, and even the arts

Characteristics of Software Design

- Design is conscious
- Design keeps human concerns in the center
- Design is a conversation with materials
- Design is creative
- Design is communication
- Design has social consequences
- Design is a social activity

Why Software Design

- **Writing a *program* is** easy
- **Developing a *software product* is** hard
- **High-quality software products are robust, efficient and effective**
- **High-quality software products are easy to understand, modify and compose with other high-quality software products**
- **Software engineers are skilled professionals who follow "best practices"**

High quality software

- The importance of high quality software
- What are the characteristics of high quality software?
- The need for precision in the specification of software

Top 10 excuses for low quality software

Top 10 Replies by Programmers when their programs do not work:

- **10. "That's weird..."**
- **9. "It's never done that before."**
- **8. "It worked yesterday."**
- **7. "It must be a hardware problem."**
- **6. “I haven't touched that module in weeks!"**
- **5. "You must have the wrong version."**
- **4. "Somebody must have changed my code."**
- **3. "Did you check for a virus on your system?"**
- **2. "You can't use that version on your system."**
- **And the Number One Reply by Programmers when their programs don't work:**

- **1. "I thought I fixed that."**

The need to produce correct software systems

- As computers become cheaper, smaller, and more powerful, their spread through our technological society becomes more pervasive.

Why study software design?

- In an article in the January/February 1997 issue of *I.E.E.E. Software*, authors cite staggering cost estimates of software development failures at $81 billion for 1995, and $100 *billion* for 1996.
- Several highly visible failures:
  - The cancellation of IBM's $8 *billion* contract with the FAA
  - The DOD cancellation of a $2 billion contract with IBM to modernize its information systems,
  - The failure of the software for delivering real-time sports data at the 1996 Olympics,
  - The one and one-half year delay in the United Airlines automated baggage handling system at the new Denver airport at a cost of $1.1 million per day, and the list could go on.
- A reading of Peter Neumann's book, *Computer Related Risks*, reveal deaths which resulted from radiation overdoses from a computer-based radiation-therapy system in the mid-1980s
- Aids to precision and cross-checking are essential, and this is precisely the objective of software design and modeling.

Modeling as a Design Technique

- A **model** is an abstraction of something for the purpose of understanding it before building it.

- To build complex systems
  - Abstract different views of the system,
  - Build models using precise notations,
  - Verify that the models satisfy requirements,
  - Gradually add detail to transform models into implementation

- What are the purposes of models?

Why model?

- Testing a physical entity before building it.
- Communication with customers.
- Visualization
- Reduction of complexity

Customer Myth

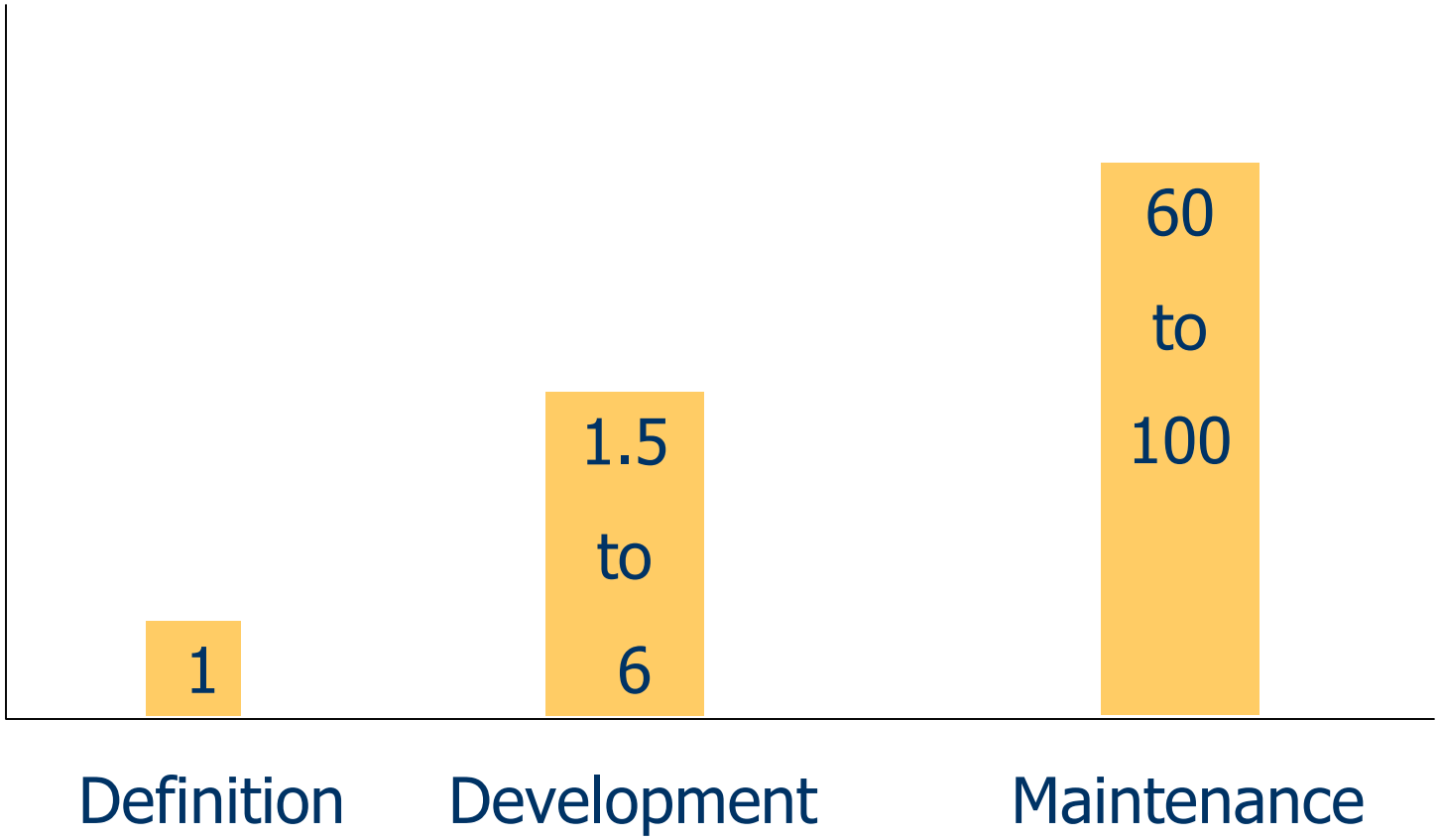
Myth: A general statement of objectives is sufficient, we will fill in the details later.

Reality: Poor up-front definition is the major cause of failed software efforts.

Customer Myth

- Myth: Project requirements continually change, but change can easily be accommodated because software is flexible.
- Reality: It is true that software requirements do change, but the impact of change varies with the time the change is introduced.

Cost of Change



Practitioner Myth

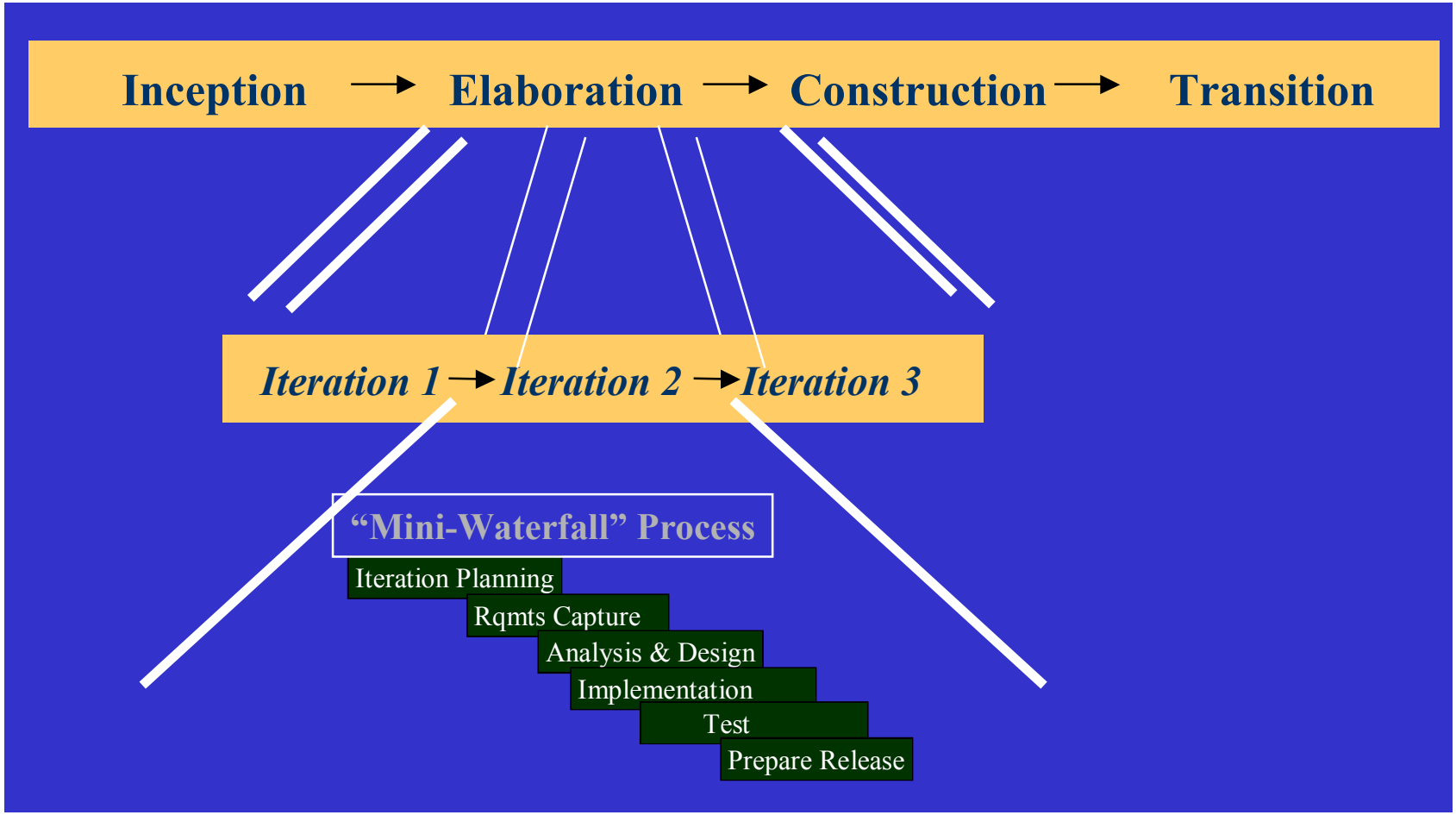
Myth: The only deliverable for a successful project is the working program.

Reality: A working program is only one part of the software configuration. Documentation forms the foundation for a successful development and provides guidance for the software maintenance task.

Building the system

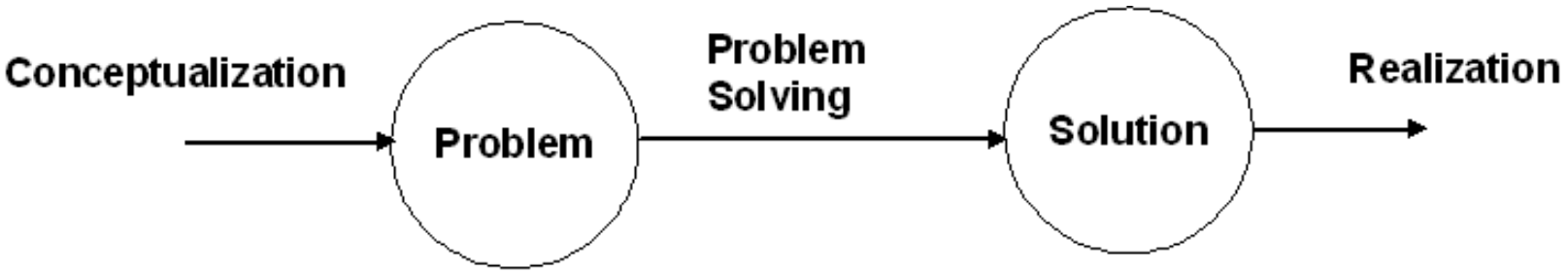
- Determine the requirements
- Create a system design
- Design individual programs
- Test the programs in pieces
- Test the programs together
- Deliver the system

Iterative Development with the Unified Process (UP)



Analysis & Design

- **Analysis** emphasizes the investigation of the domain and requirements
- **Design** emphasizes a conceptual solution that fulfills the requirements



What is Object-Orientation (OO)?

- OO means that we organize software as a collection of discrete objects that incorporate both data structure and behavior.
- Characteristics:
  - Identity: Discrete, distinguishable entities, called objects.
  - Classification: Objects with same data structure and behavior are grouped into a class.
  - Inheritance: Sharing of attributes and operations among classes based on a hierarchical relationship.
  - Polymorphism: Same operation may behave differently for different classes.

OO Principles

- **Information Hiding:** the principle of information hiding is the hiding of *design decisions* in a computer program that are most likely to change, thus protecting other parts of the program from change if the design decision is changed. Protecting a design decision involves providing a stable interface which shields the remainder of the program from the implementation

OO Principles

- **Encapsulation:** In modern programming languages, the principle of information hiding manifests itself in a number of ways, including encapsulation.
  - Abstract data type (ADT) is a specification of a set of data and the set of operations that can be performed on the data.
- **Abstraction** refers to focusing on essential aspects of an application while ignoring details.

Definition
Object-Oriented Analysis

A method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain.

Definition
Object-Oriented Design

A method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design.

Definition
Object-Oriented Programming

A method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships.

OO Methodology

- **System Conception** involves business analysts or users conceiving an application and tentative requirements.
- **Analysis** restates requirements by constructing models.
  - Domain model: A description of the real-world objects. (e.g., stock, bond, trade, commission in a stockbroker application).
  - Application model: Parts of the application system that are visible to the user (e.g., object that control the execution and present the results).

OO Methodology

- **System Design** involves devising a high-level strategy – the system architecture and interaction design.
- **Class design** adds details to the analysis model in accordance with the system design strategy.
  - Interface design
  - Algorithm and data structure design.
- **Implementation** involves translating class design onto a programming language

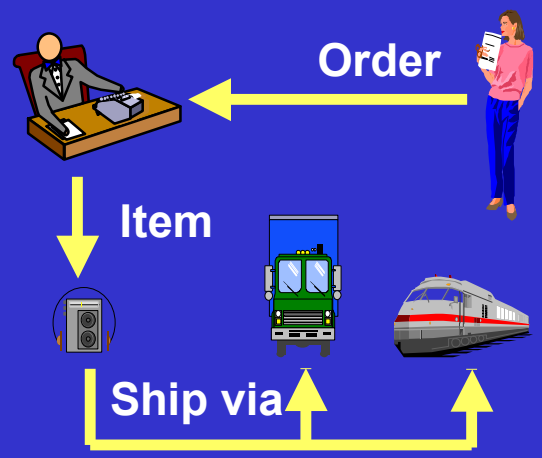
Three Models

- The class model describes the static structure of the objects and their relationships (Class diagram).
  - The class model defines the context for software development.
  - The class diagram is a graph whose nodes are classes and whose arcs are relationships.
- The state model describes aspects that change over time.
  - Specifies and implements control with state diagrams.

Three Models

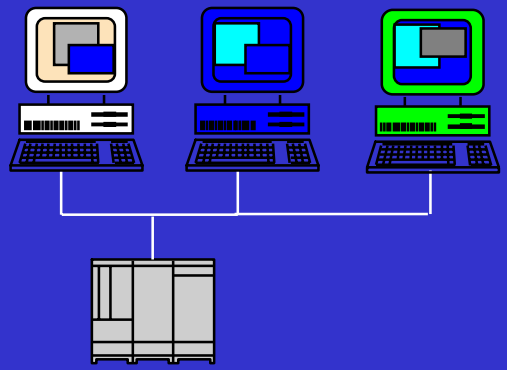
- The interaction model describes how objects in a system cooperate to achieve broader results.
  - Use cases focus on functionality from the perspective of the users.
  - The sequence and collaboration diagrams show the objects that interact and the time sequence of their interactions.
  - The activity diagram elaborates control and workflow that depict processing steps.

Visual Modeling with UML



Business Process

"Modeling captures essential parts of the system."
Dr. James Rumbaugh

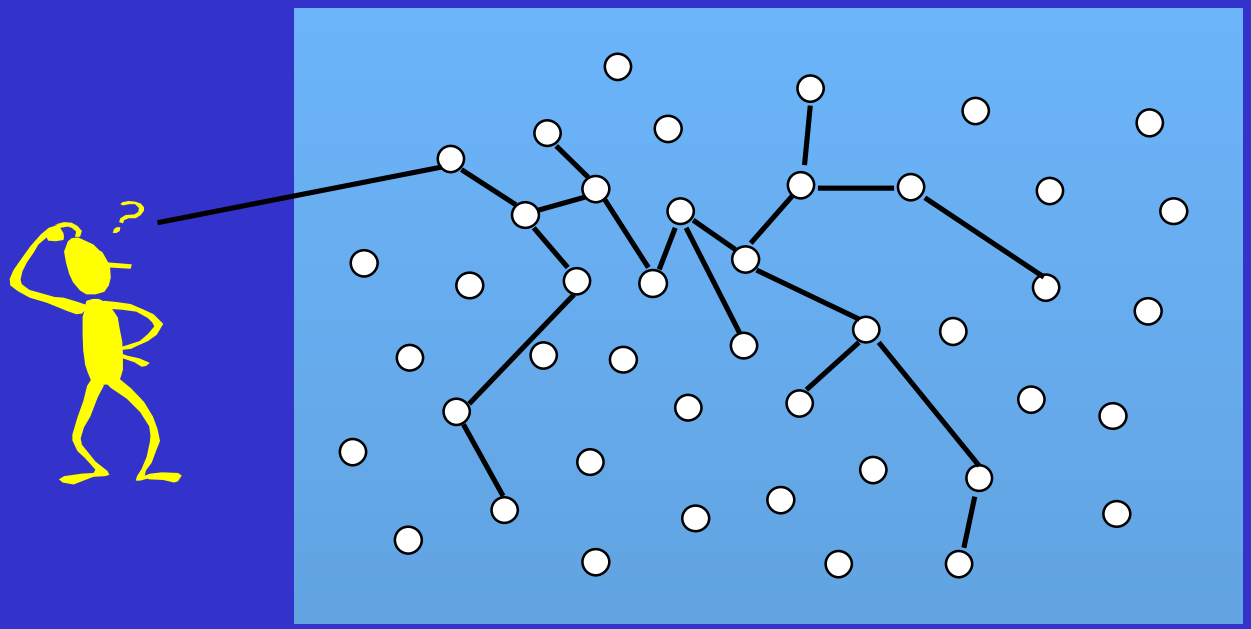


Computer System

Visual Modeling is modeling using standard graphical notations

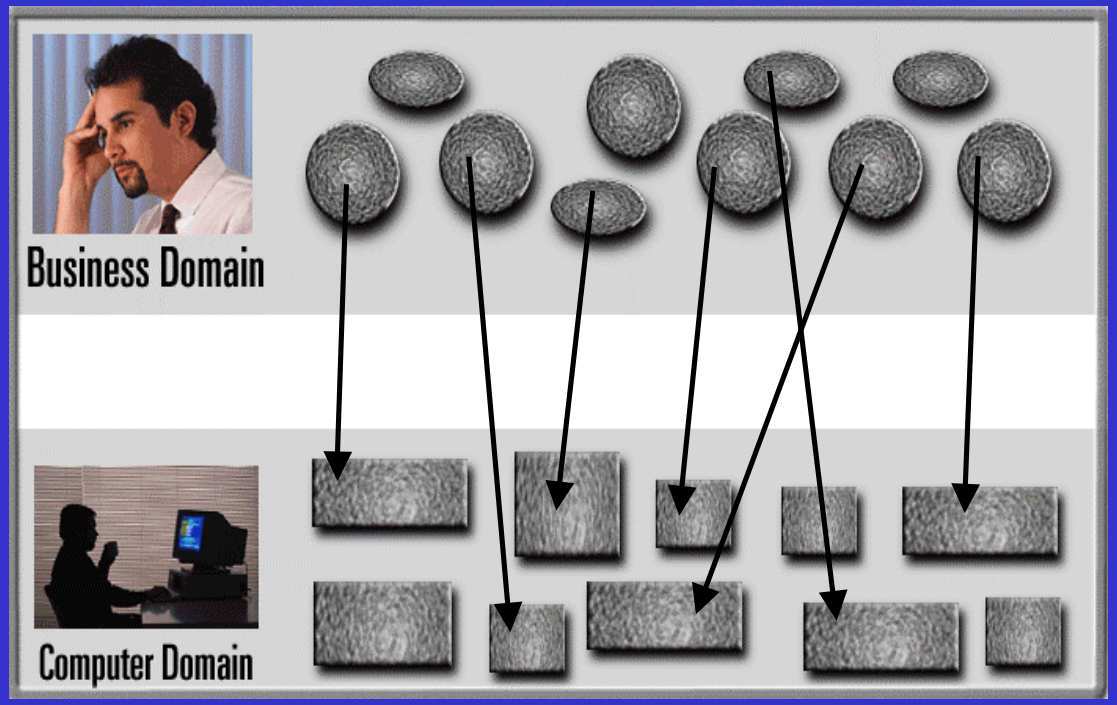
Visual Modeling Captures Business Process

Use Case Analysis is a technique to capture business process from user's perspective



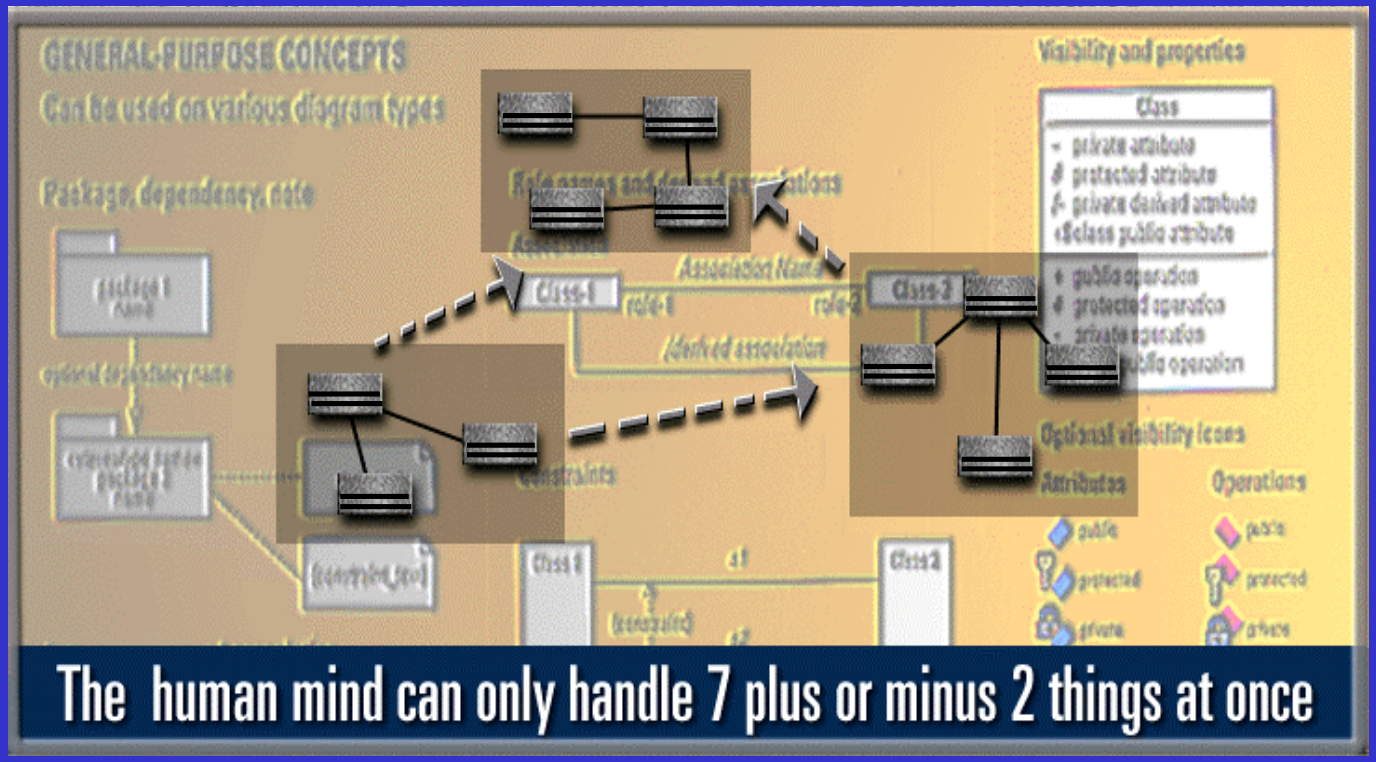
Visual Modeling is a Communication Tool

Use visual modeling to capture business objects and logic

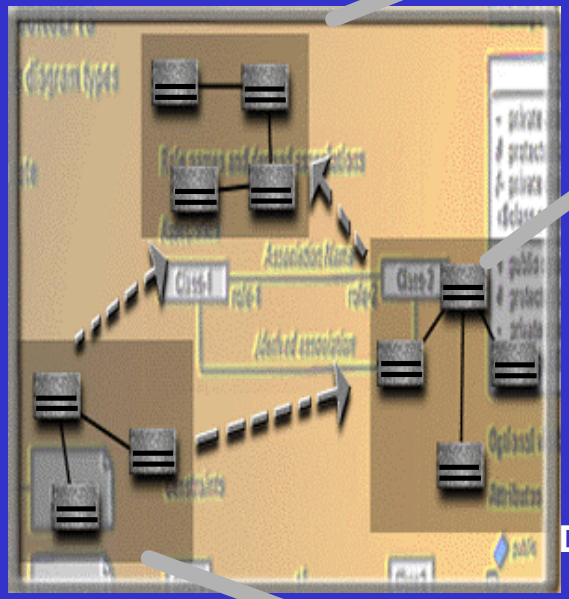


Use visual modeling to analyze and design your application

Visual Modeling Manages Complexity

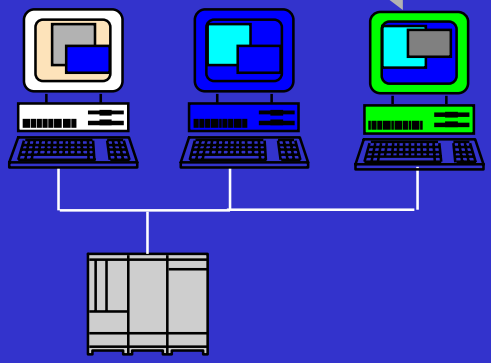


Visual Modeling Defines Software Architecture



User Interface
(Visual Basic,
Java)

Business Logic
(C++, Java)



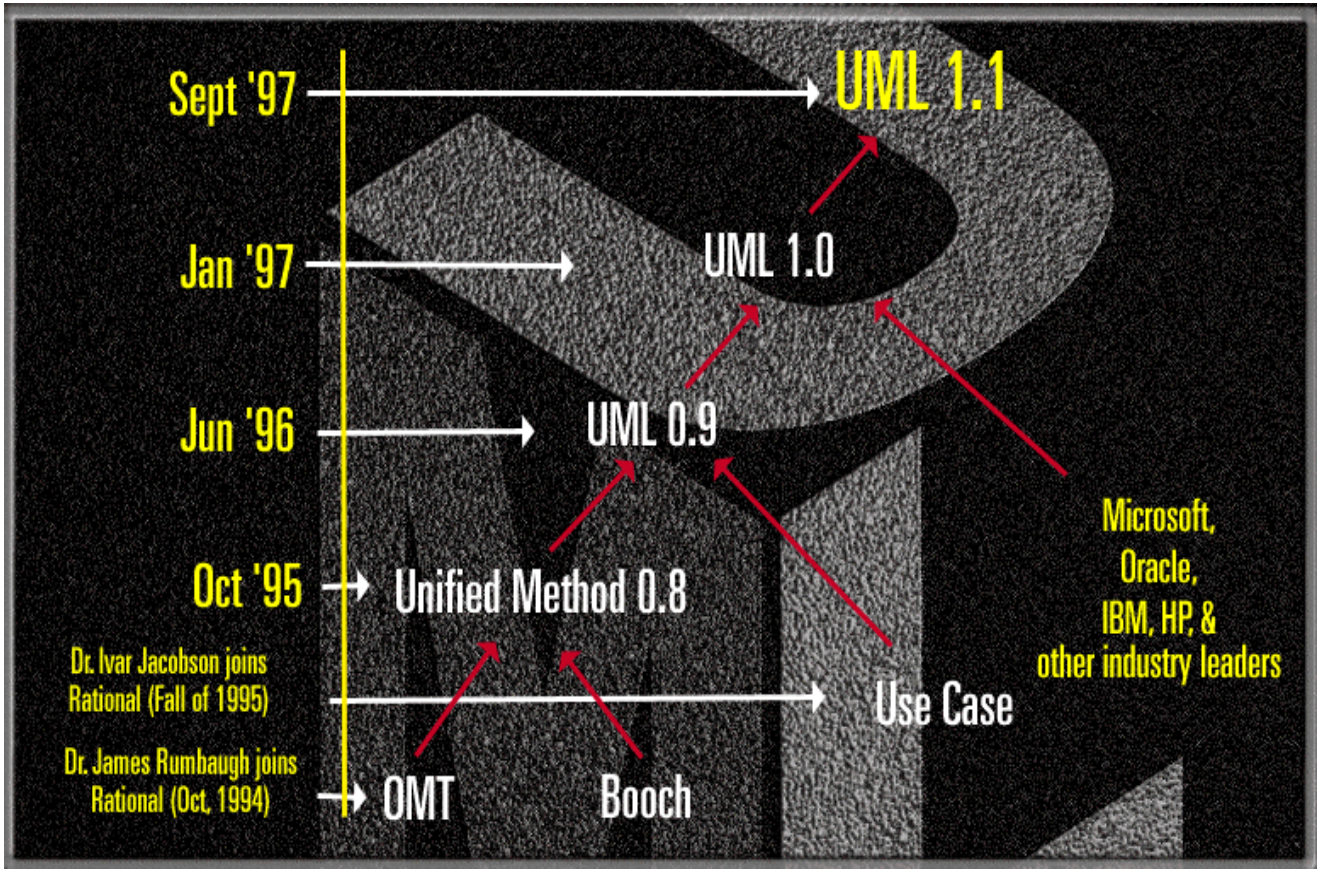
Database Server
(C++ & SQL)

Model your system independent of implementation language

What is the UML?

- UML stands for Unified Modeling Language
- The UML combines the best of the best from
  - Data Modeling concepts (Entity Relationship Diagrams)
  - Business Modeling (work flow)
  - Object Modeling
  - Component Modeling
- The UML is the standard language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system
- It can be used with all processes, across different implementation technologies

History of the UML



Design Paradigms

- **Function-Driven Paradigm**
- **Data-Driven Paradigm**

} **Structured Methods**

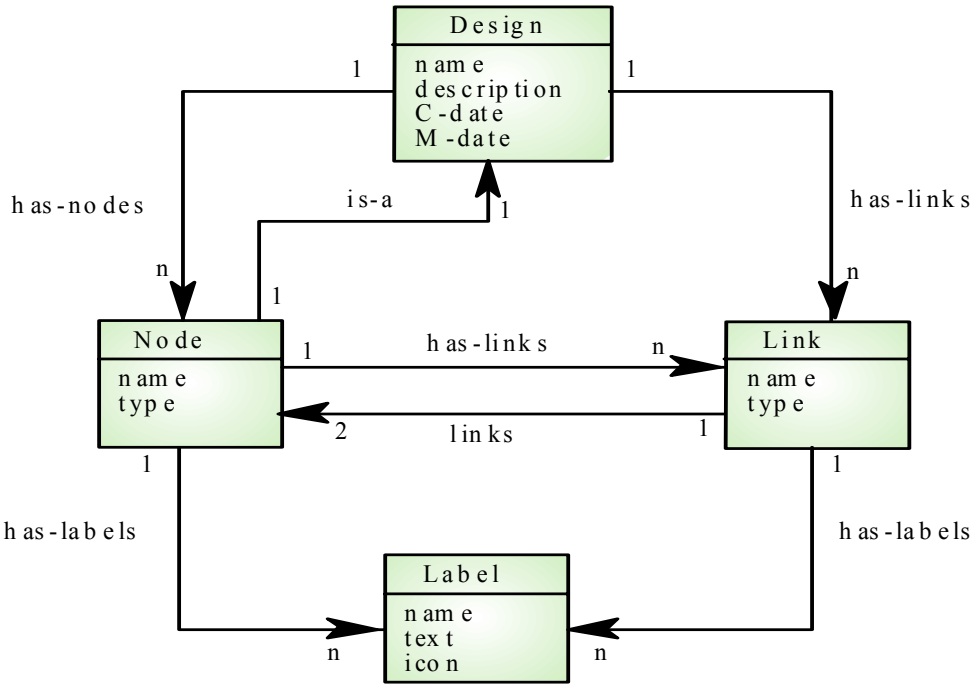
- **Object-Oriented Paradigm**

Function-driven Design

- **Process models show the overall process or the subprocesses that are supported by the system**

Data-Driven Paradigm

- Used to describe the logical structure of data processed by the system
- Entity-relation-attribute model sets out the entities in the system, the relationships between these entities and the entity attributes
- Widely used in database design. Can readily be implemented using relational databases
- No specific notation provided in the UML but objects and associations can be used



Question

What major problems have you encountered during past software projects? Estimate what percentage of your time you spend on analysis, design, coding, testing, debugging, and fixing?