# Object Oriented Programming

Binnur Kurt
*kurt@ce.itu.edu.tr*

Istanbul Technical University
*Computer Engineering Department*


Version 0.1.2

# About the Lecturer

- **BSc**

  İTÜ, Computer Engineering Department, 1995

- **MSc**

  İTÜ, Computer Engineering Department, 1997

- Areas of Interest
  - Digital Image and Video Analysis and Processing
  - Real-Time Computer Vision Systems
  - Multimedia: Indexing and Retrieval
  - Software Engineering
  - OO Analysis and Design

# Welcome to the Course

## Important Course Information

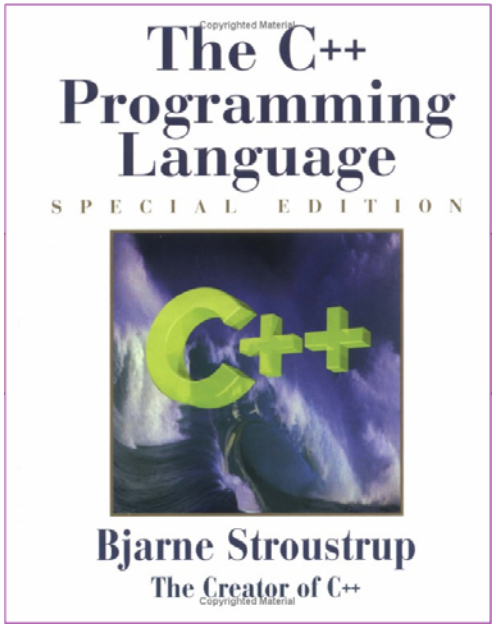
- Course Hours
  - 10:00-13:00 Thursday
- Course Web Page
  - http://www.cs.itu.edu.tr/~kurt/courses/blg252e
- Join to the group
  - http://groups.yahoo.com/group/blg252e
  - blg252e@yahoogroups.com
- E-mail Kurt@ce.itu.edu.tr

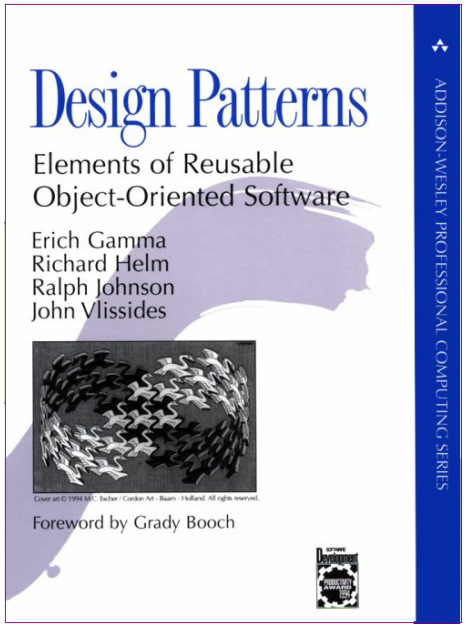
# Grading Scheme

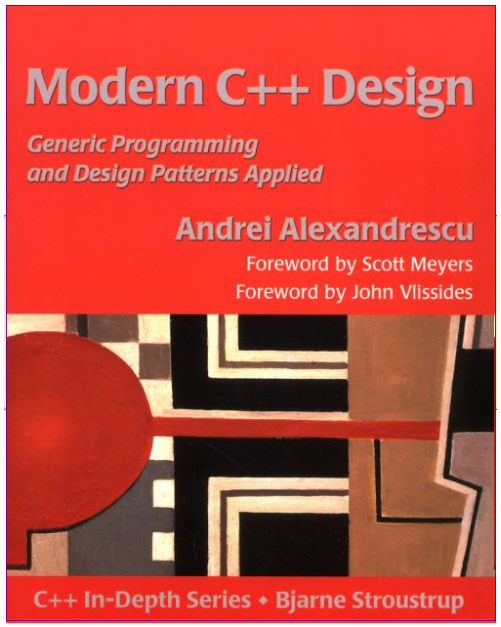
- 3 Homeworks (5% each)
- 2 Midterm Exams (20%,25%)
- A final exam (40%)
- You must follow the official Homework Guidelines

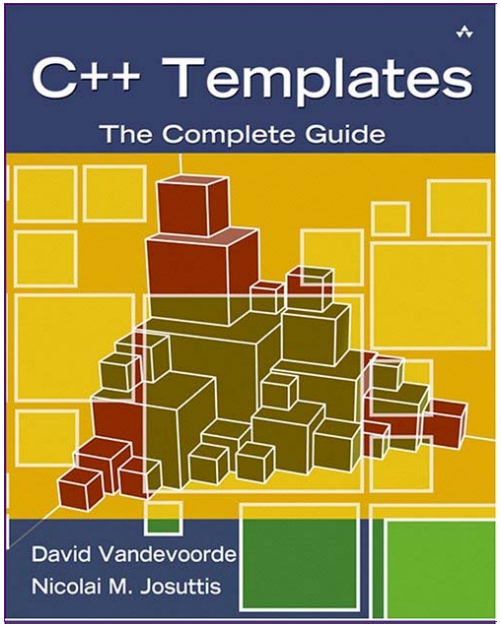
  (http://www.ce.itu.edu.tr/lisans/kilavuz.html).
- Academic dishonesty including but not limited to cheating, plagiarism, collaboration is unacceptable and subject to disciplinary actions. Any student found guilty will have grade F. Assignments are due in class on the due date. Late assignments will generally not be accepted. Any exception must be approved. Approved late assignments are subject to a grade penalty.

# References




The presentation is based on

Asst.Prof.Dr. Feza Buzlaca's Lecture Notes

*Tell me and I forget.*
*Show me and I remember.*
*Let me do and I understand.*

—Chinese Proverb

There is no time for lab sessions
On the course web page you will find lab files for each week. You should do the lab sessions on your own.
Just follow the instructions on these documents.

# Purpose of the Course

- To introduce several programming paradigms including **Object-Oriented Programming, Generic Programming, Design Patterns**
- To show how to use these programming schemes with the C++ programming language to build "**good**" programs.

# Course Outline

1. Introduction to Object Oriented Programming.
2. C++: A Better C.
3. Classes and Objects
4. Constructors and Destructors
5. Operator Overloading
6. Inheritance
7. Pointers to Objects
8. Polymorphism
9. Exceptions

# Course Outline

10. Templates

11.The Standard Template Library - STL

# How to Use the Icons

## Demonstration

## Discussion

## Reference

## Exercise

# 1 INTRODUCTION

# Content

- Introduction to Software Engineering
- Object-Oriented Programming Paradigm

# Software

- Computer Software is the product that software engineers design and build.
- It encompasses
  - programs that execute within a computer of any size and architecture,
  - documents that encompass hard-copy and virtual forms,
  - data that combine numbers and text but also includes representations of pictorial, video and audio information.

# History

- Common problems:
  - Why does it take so long?
  - Why are development costs so high?
  - Why can't find all faults before delivery?
  - Why can't we measure the development?

# History

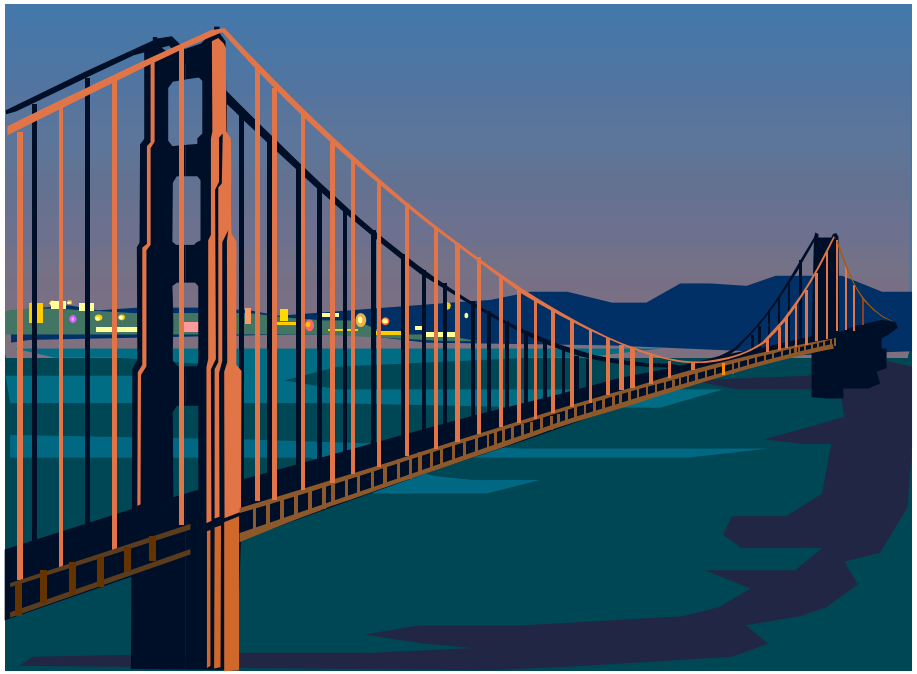
- Software Engineering: 1967, NATO Study Group, Garmisch/GERMANY
- 1968, NATO Software Engineering Conference: Software Crisis
  - Low quality
  - Not met deadlines and cost limits

# After 35 years

- Still softwares are
  - Late
  - Over budget
  - With residual faults
- Means
  - SW has own unique properties and problems
  - Crisis >>>>> Depression

# Is SW An Engineering?

- May be?
- Bridge – Operating System
  - After collapse, redesign & rebuild
  - Inspect similar bridges
  - Perfectly engineered
  - Experience
  - Maintaining

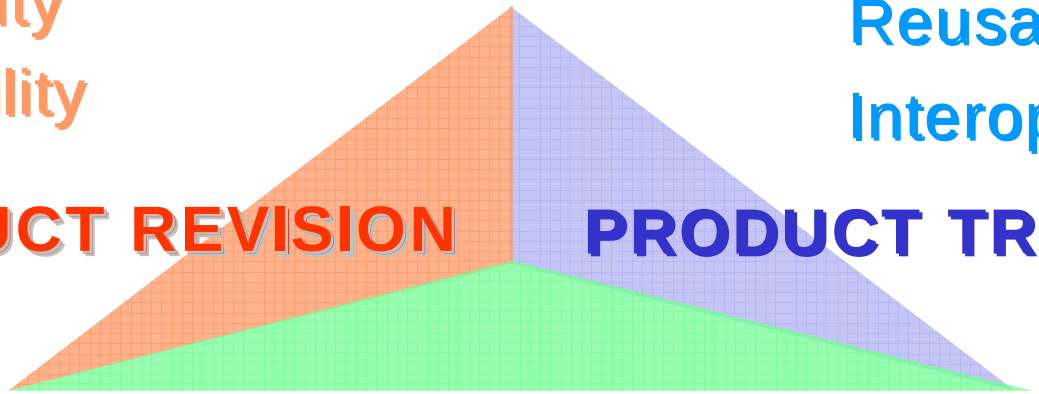


# McCall Quality Triangle



Maintainability
Flexibility
Testability

Portability
Reusability
Interoperability

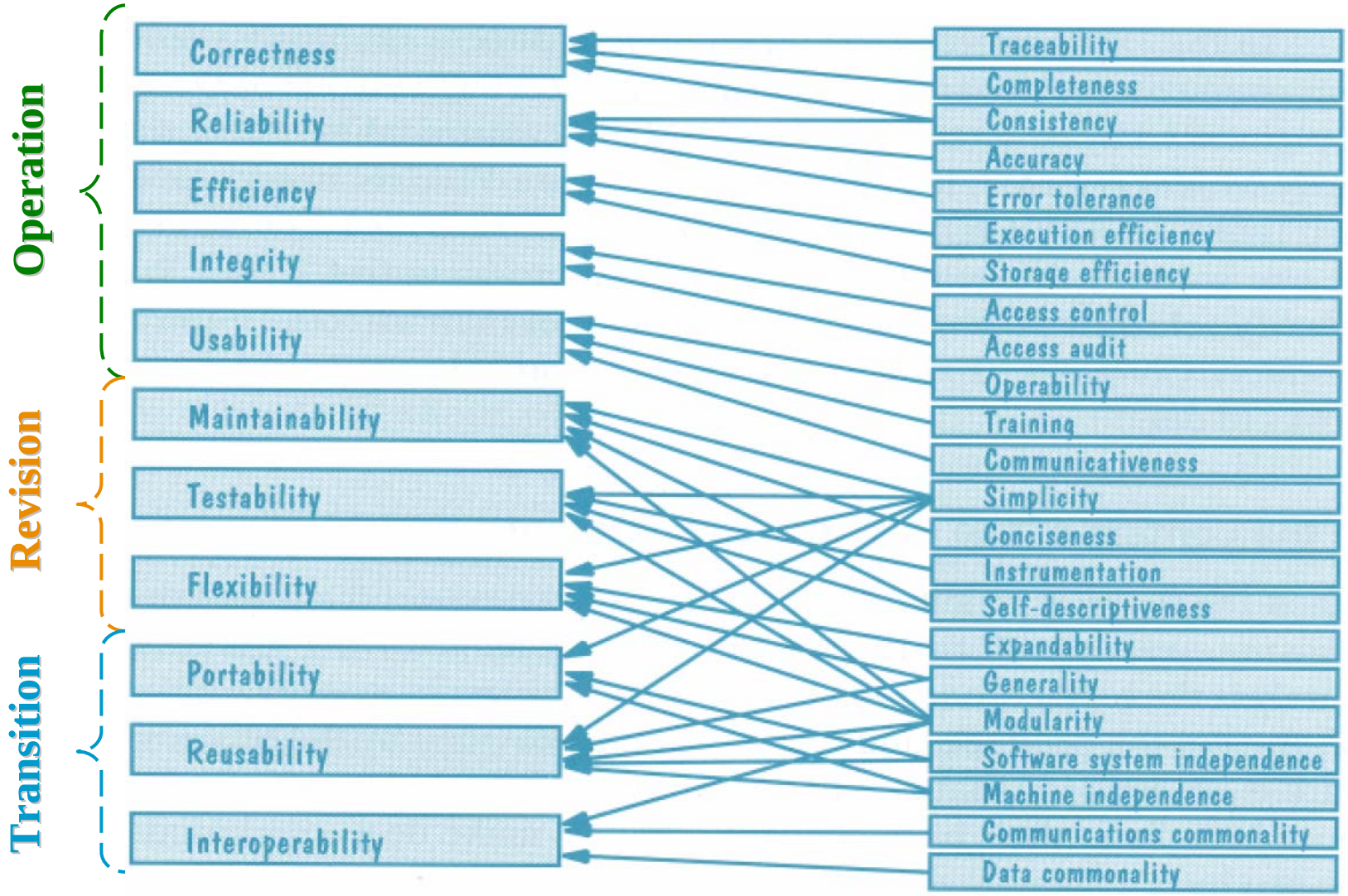


PRODUCT REVISION

PRODUCT TRANSITION

PRODUCT OPERATION

Correctness
Reliability
Usability
Integrity
Efficiency


Operation
Revision
Transition
Correctness
Reliability
Efficiency
Integrity
Usability
Maintainability
Testability
Flexibility
Portability
Reusability
Interoperability
Traceability
Completeness
Consistency
Accuracy
Error tolerance
Execution efficiency
Storage efficiency
Access control
Access audit
Operability
Training
Communicativeness
Simplicity
Conciseness
Instrumentation
Self-descriptiveness
Expandability
Generality
Modularity
Software system independence
Machine independence
Communications commonality
Data commonality

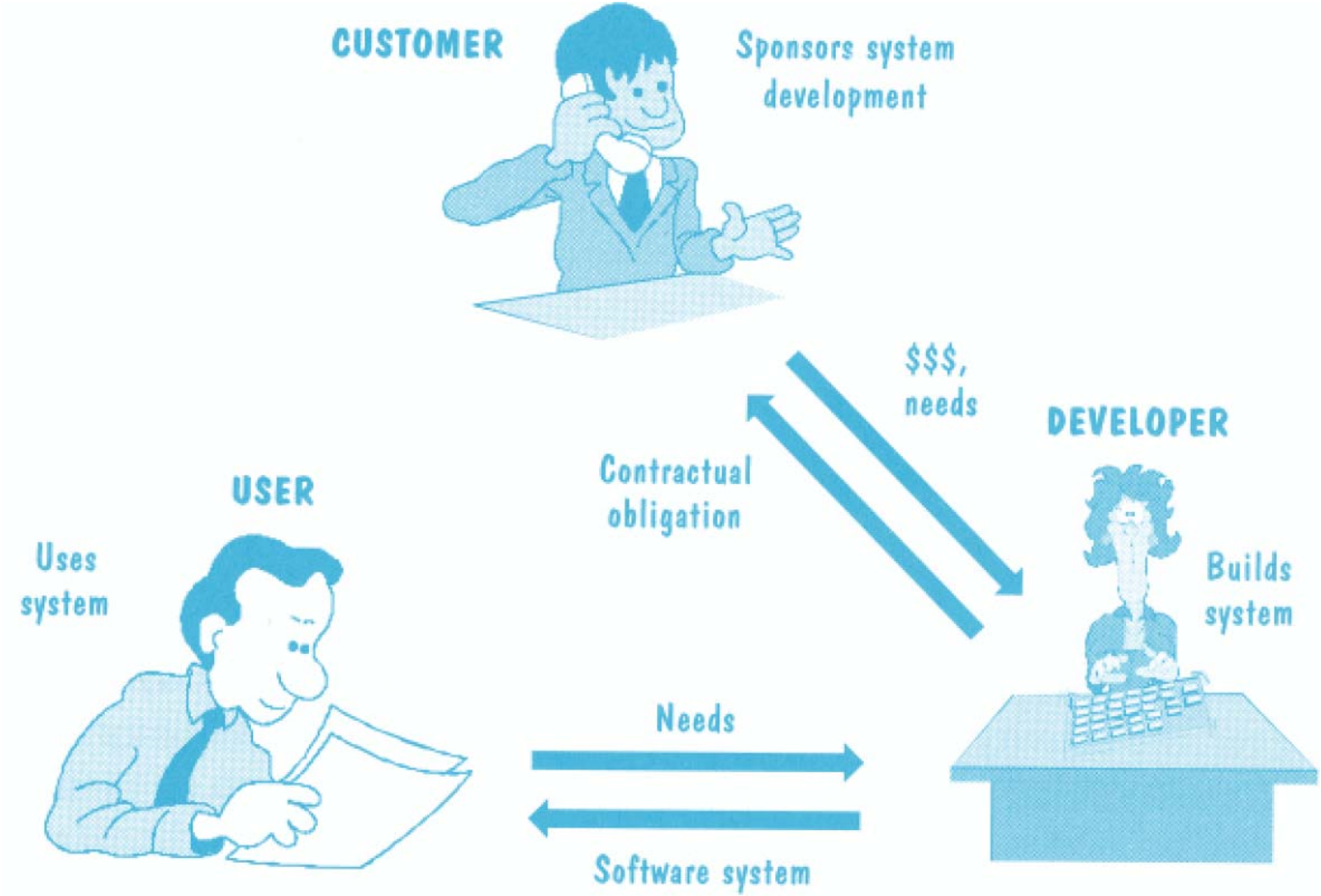
# McCall Quality Triangle

- Correctness: The extent to which a program satisfies its specification and fulfills the customer's mission objectives
- Reliability: The extent to which a program can be expected to perform its intended function with required precision
- Efficiency: The amount of computing resources and code required by a program to perform its function
- Integrity: Extent to which access to software or data by unauthorized persons can be controlled
- Usability: Effort required to learn, operate, prepare input and interpret output of a program
- Maintainability: Effort required to locate and fix an error in a program

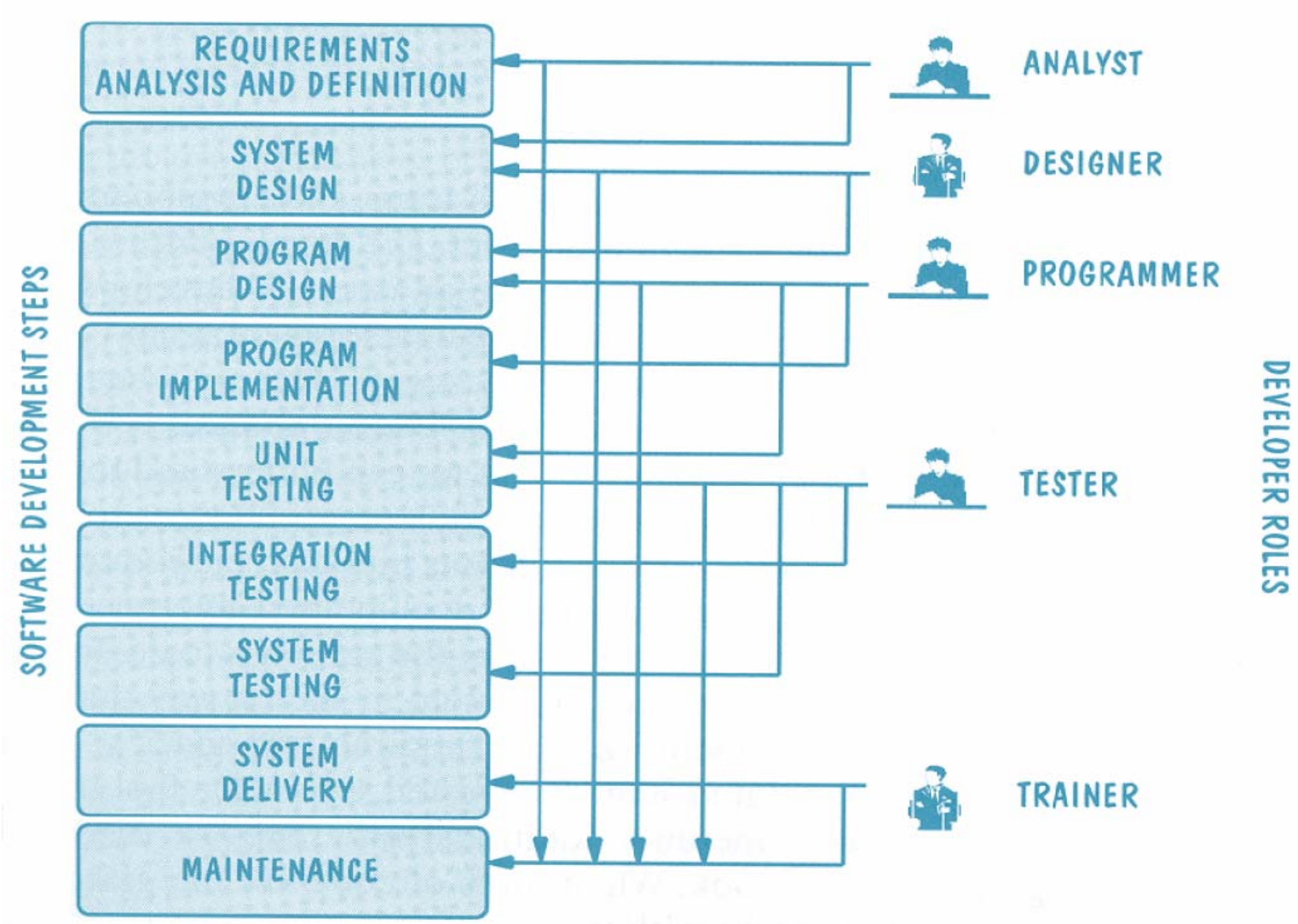
# McCall Quality Triangle

- Flexibility: Effort required to modify an operational program
- Testability: Effort required to test a program to ensure that it performs its intended function
- Portability: Effort required to transfer the program from one hardware and/or software system environment to another
- Reusability: Extent to which a program can be reused in other applications
- Interoperability: Effort required to couple one system to another

# Customer-User-Developer



# Development Team



# Software Life Cycle

- Requirements Phase
- Specification Phase
- Design Phase
- Implementation Phase
- Integration Phase
- Maintenance Phase
- Retirement Phase

# Requirements Phase

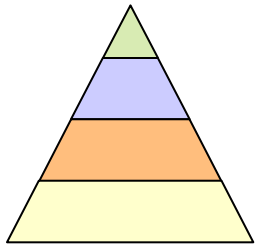
- Defining constraints
  - Functions
  - Due dates
  - Costs
  - Reliability
  - Size
- Types
  - Functional
  - Non-Functional

# Specification Phase

- Documentation of requirements
  - Inputs & Outputs
  - Formal
  - Understandable for user & developer
  - Usually functional requirements. (what to do)
  - Base for testing & maintenance
- The contract between customer & developer ?

# Design Phase

- Defining Internal structure (how to do)
- Has some levels (or types of docs)
  - Architectural design
  - Detailed design
  - ...



- Important
  - To backtrack the aims of decisions
  - To easily maintain

# Implementation Phase

- Simply coding
- Unit tests
  - For verification

# Integration Phase

- Combining modules
- System tests
  - For validation
- Quality tests

# Maintenance Phase

- Corrective
- Enhancement
  - Perfective
  - Adaptive
- Usually maintainers are not the same people with developers.
- The only input is (in general) the source code of the software?!?

# Retirement Phase

- When the cost of maintenance is not effective.
  - Changes are so drastic, that the software should be redesigned.
  - So many changes may have been made.
  - The update frequency of docs is not enough.
  - The hardware (or OS) will be changed.

# Why Object Technology?

- Expectations are,
- Reducing the effort, complexity, and cost of development and maintenance of software systems.
- Reducing the time to adapt an existing system (quicker reaction to changes in the business environment). Flexibility, reusability.
- Increasing the reliability of the system.

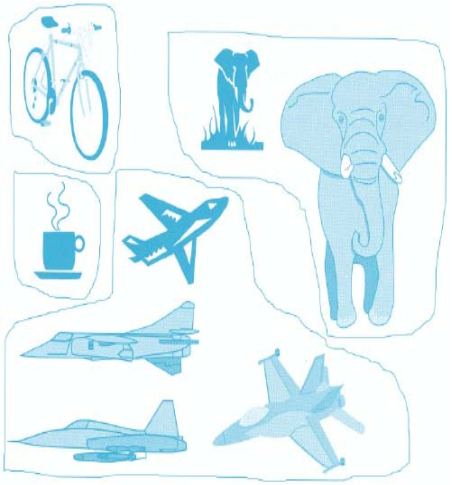
# Why C++

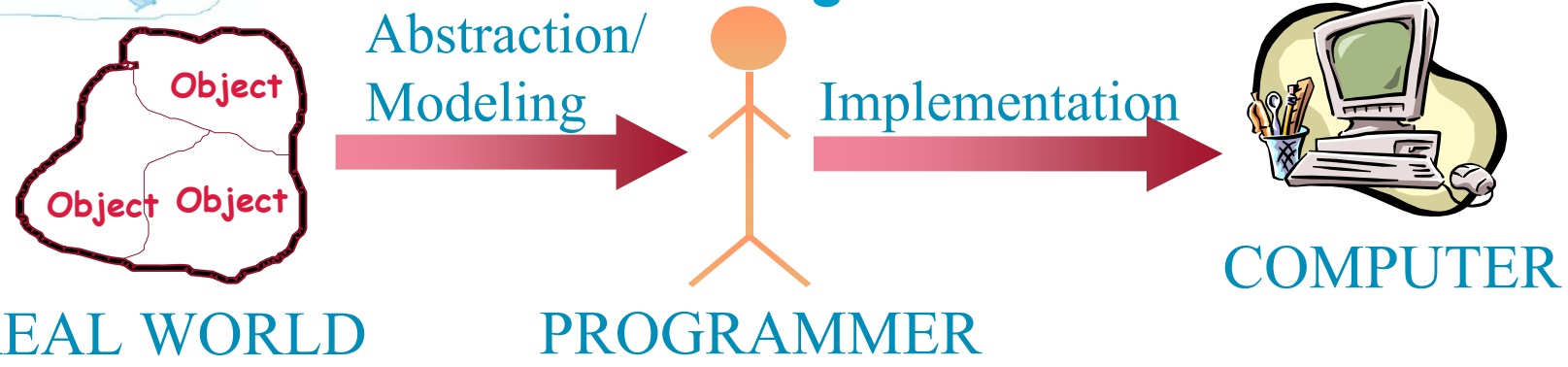
- C++ supports writing high quality programs (supports OO)
- C++ is used by hundreds of thousands of programmers in every application domain.
  - This use is supported by hundreds of libraries, hundreds of textbooks, several technical journals, many conferences.
- Application domain:
  - Systems programming: Operating systems, device drivers. Here, direct manipulation of hardware under real-time constraints are important.
  - Banking, trading, insurance: Maintainability, ease of extension, ease of testing and reliability is important.
  - Graphics and user interface programs
  - Computer Communication Programs

# What is Programming?

- Like any human language, a programming language provides a way to express concepts.
- Program development involves creating models of real world situations and building computer programs based on these models.
- Computer programs describe the method of implementing the model.
- Computer programs may contain computer world representations of the things that constitute the solutions of real world problems.

# What is Programming? (Con't)






- If successful, this medium of expression (the object-oriented way) will be significantly easier, more flexible, and efficient than the alternatives as problems grow larger and more complex.

# Learning C++

- Like human languages, programming languages also have many syntax and grammar rules.
- Knowledge about grammar rules of a programming language is not enough to write “good” programs.
- The most important thing to do when learning C++ is to focus on concepts and not get lost in language-technical details.
- Design techniques is far more important than an understanding of details; that understanding comes with time and practice.
- Before the rules of the programming language, the programming scheme must be understood.
- Your purpose in learning C++ must not be simply to learn a new syntax for doing things the way you used to, but to learn new and better ways of building systems

# Software Quality Metrics

- A program must do its job correctly. It must be useful and usable.
- A program must perform as fast as necessary (Real-time constraints).
- A program must not waste system resources (processor time, memory, disk capacity, network capacity) too much.
- It must be reliable.
- It must be easy to update the program.
- A good software must have sufficient documentation (users manual).

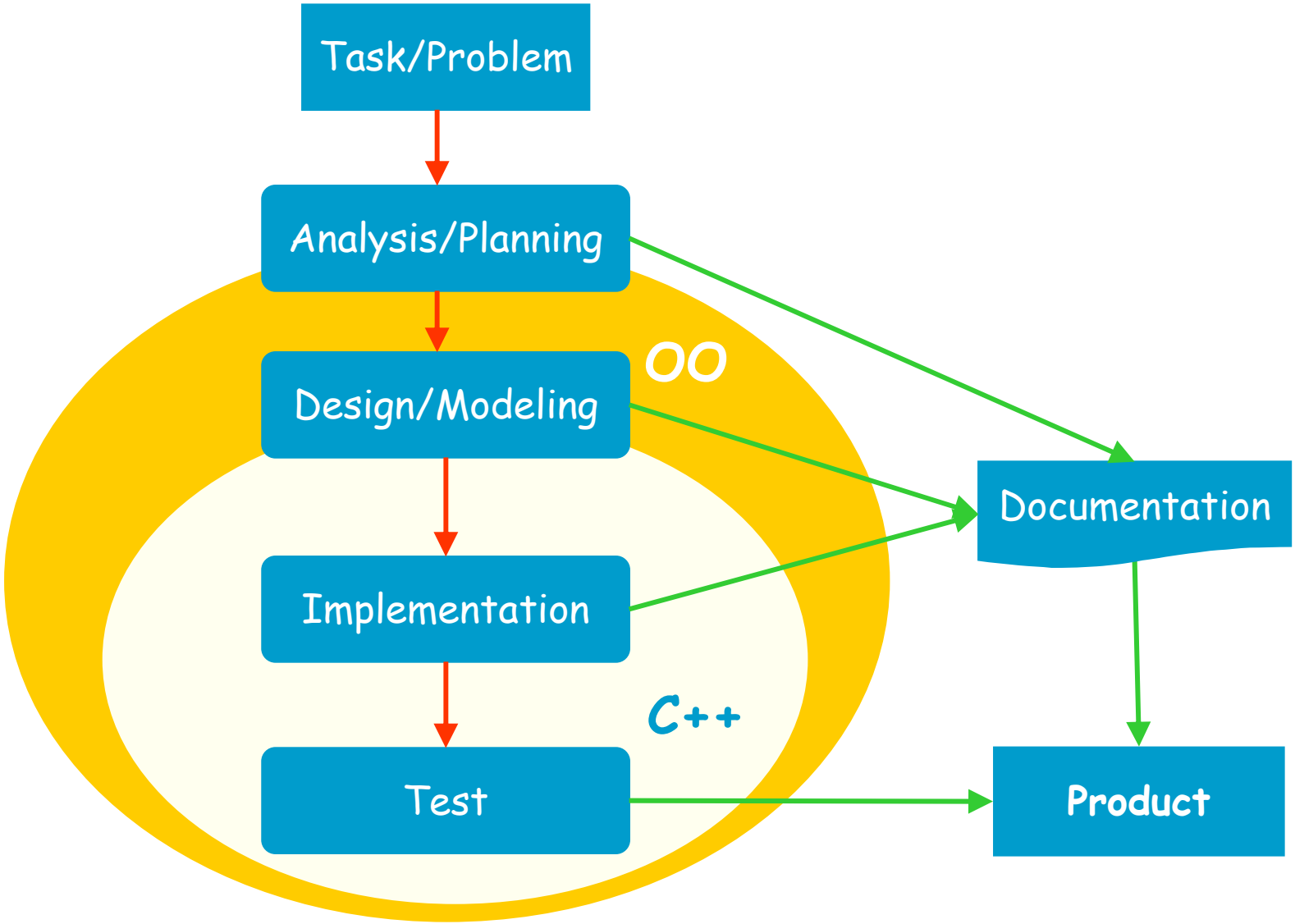
user

- Source code must be readable and understandable.
- It must be easy to maintain and update (change) the program.
- A program must consist of independent modules, with limited interaction.
- An error may not affect other parts of a program (Locality of errors).
- Modules of the program must be reusable in further projects.
- A software project must be finished before its deadline.
- A good software must have sufficient documentation (about development).

Software developer

Object-oriented programming technique enables programmers to build high-quality programs. While designing and coding a program, these quality metrics must be kept always in mind.

# Software Development Process



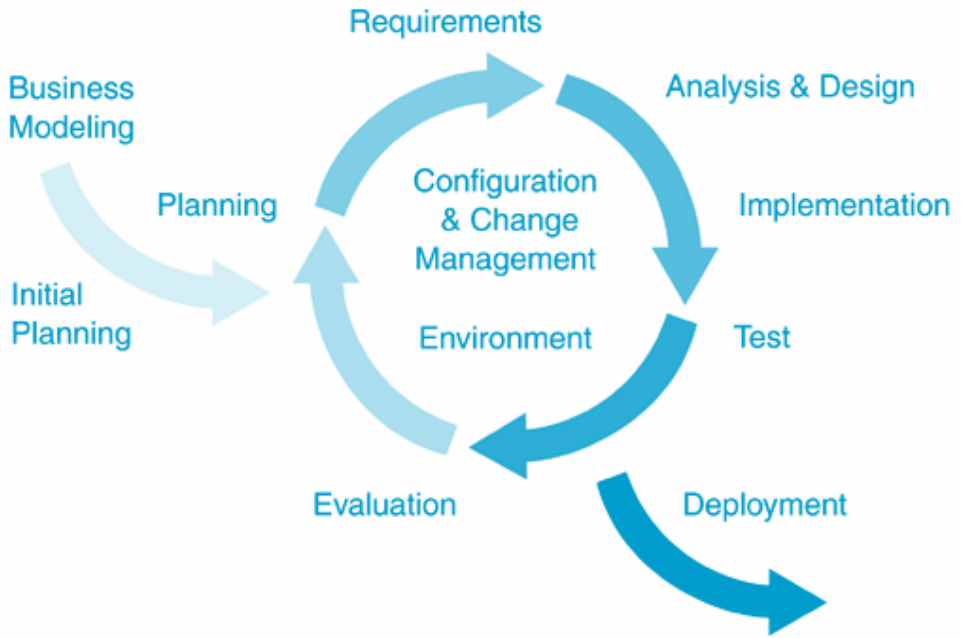
- Analysis: Gaining a clear understanding of the problem. Understanding requirements. They may change during (or after) development of the system!
- Building the programming team.
- Design: Identifying the key concepts involved in a solution. Models of the key concepts are created. This stage has a strong effect on the quality of the software. Therefore, before the coding, verification of the created model must be done.
- Design process is connected with the programming scheme. Here, our design style is object-oriented.
- Coding: The solution (model) is expressed in a program.
- Coding is connected with the programming language. In this course we will use C++.
- Documentation: Each phase of a software project must be clearly explained. A users manual should be also written.
- Test: the behavior of the program for possible inputs must be examined.

# UML

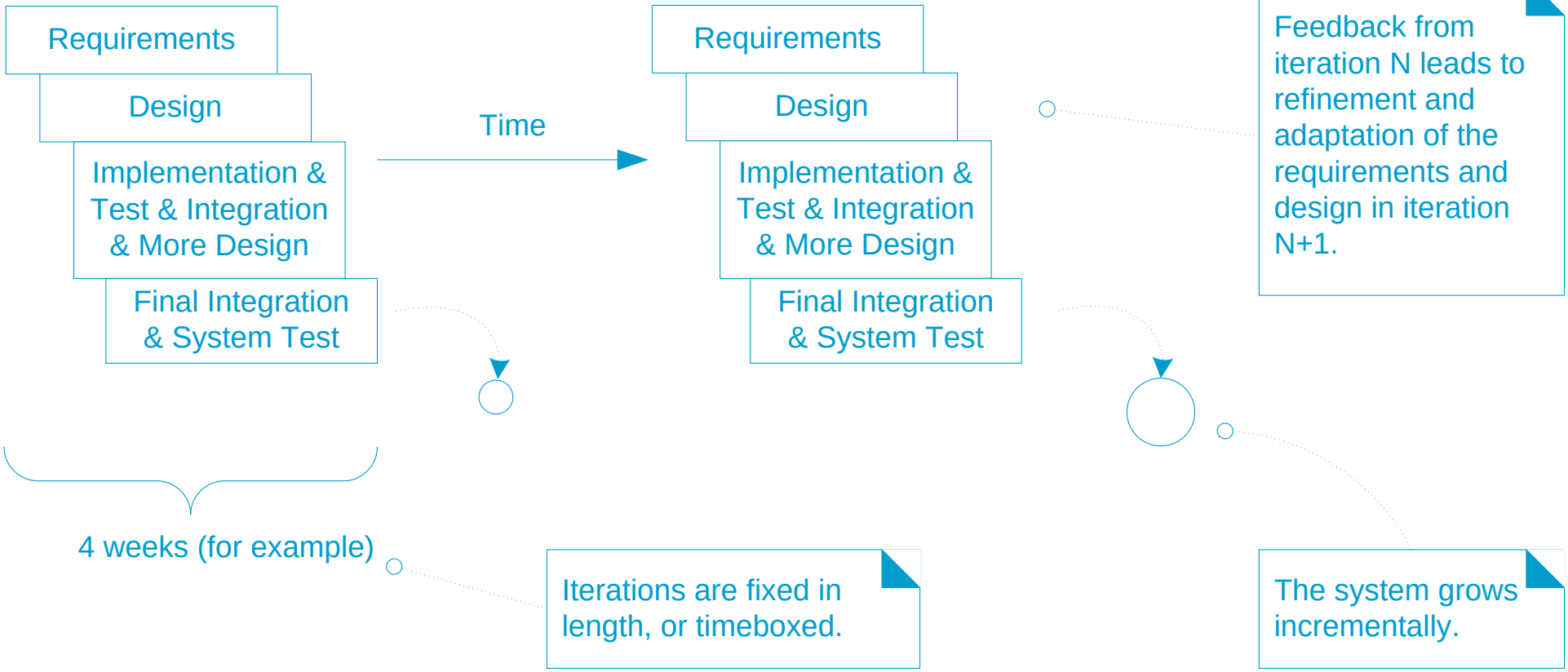
- They are important **design principles** and **design patterns**, which help us developing high-quality software. The Unified Modeling Language (**UML**) is useful to express the model.

# Unified Process (UP)

- The UP promotes several best practices.
- Iterative
- Incremental
- Risk-driven



# Unified Process (UP)

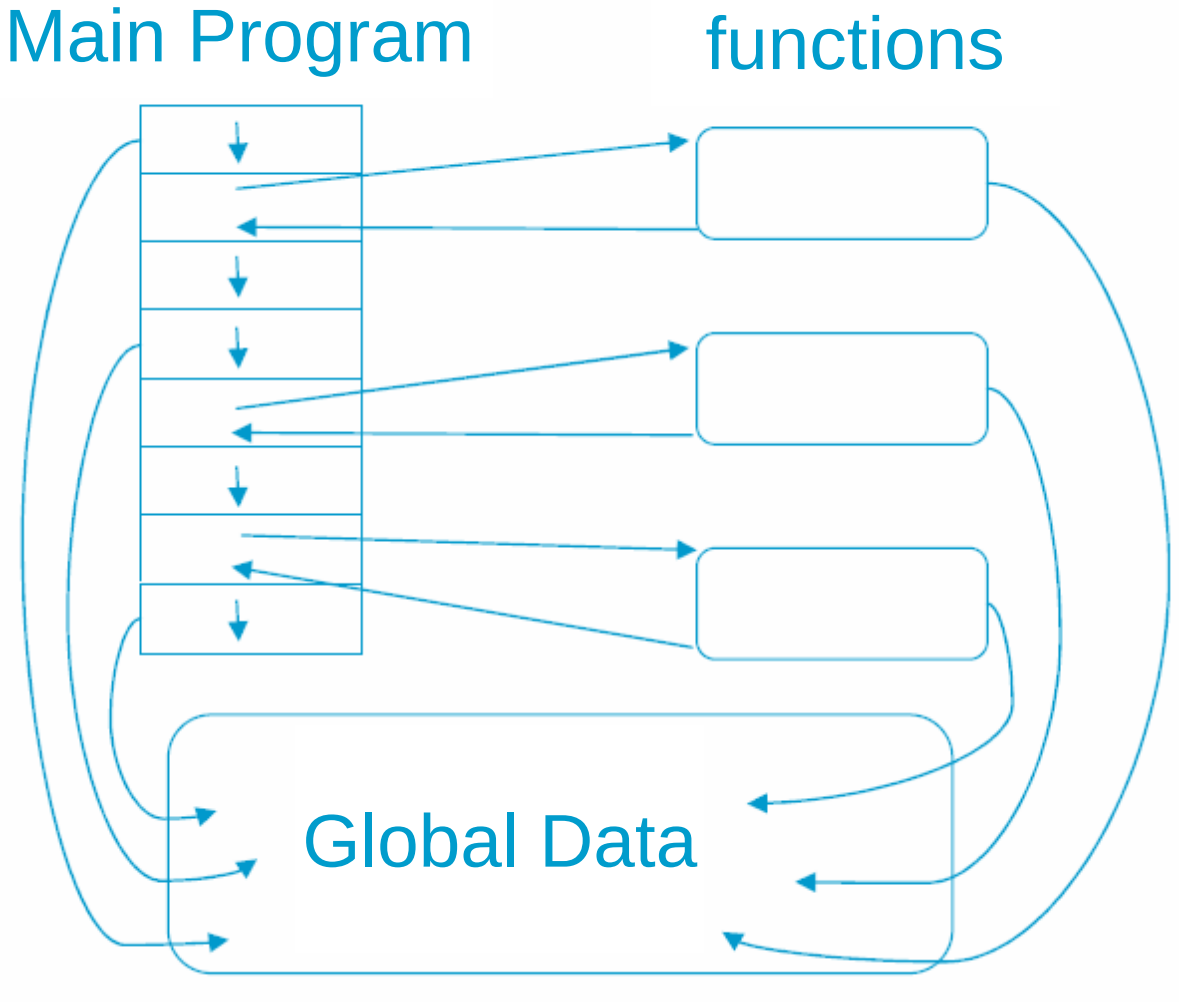


# Procedural Programming

- Pascal, C, BASIC, Fortran, and similar traditional programming languages are procedural languages. That is, each statement in the language tells the computer to do something.
- In a procedural language, the emphasis is on doing things (functions).
- A program is divided into functions and—ideally, at least—each function has a clearly defined purpose and a clearly defined interface to the other functions in the program.

# Procedural Programming





# Problems with Procedural Programming

- Data Is Undervalued
- Data is, after all, the reason for a program's existence. The important parts of a program about a school for example, are not functions that display the data or functions that checks for correct input; they are student, teacher data.
- Procedural programs (functions and data structures) don't model the real world very well. The real world does not consist of functions.
- Global data can be corrupted by functions that have no business changing it.
- To add new data items, all the functions that access the data must be modified so that they can also access these new items.
- Creating new data types is difficult.

# Besides...

- It is also possible to write good programs by using procedural programming (C programs).
- But object-oriented programming offers programmers many advantages, to enable them to write high-quality programs.

# Object Oriented Programming



The fundamental idea behind object-oriented programming is:

• The real world consists of objects. Computer programs may contain computer world representations of the things (objects) that constitute the solutions of real world problems.

• Real world objects have two parts:

  •*Properties* (or *state* :characteristics that can change),

  •*Behavior* (or *abilities* :things they can do).

•To solve a programming problem in an object-oriented language, the programmer no longer asks *how the problem will be divided into functions*, but **how it will be divided into objects**.

•The emphasis is on **data**

# Object Oriented Programming

- What kinds of things become objects in object-oriented programs?
  - Human entities: Employees, customers, salespeople, worker, manager
  - Graphics program: Point, line, square, circle, ...
  - Mathematics: Complex numbers, matrix
  - Computer user environment: Windows, menus, buttons
  - Data-storage constructs: Customized arrays, stacks, linked lists

# OOP : Encapsulation and Data Hiding



Thinking in terms of objects rather than functions has a helpful effect on design process of programs. This results from the close match between objects in the programming sense and objects in the real world.

To create software models of real world objects both *data* and the *functions* that operate on that data are combined into a single program entity. Data represent the properties (state), and functions represent the behavior of an object. Data and its functions are said to be *encapsulated* into a single entity.

An object's functions, called *member functions* in C++ typically provide the only way to access its data. The data is *hidden*, so it is safe from accidental alteration.

# OOP : Encapsulation and Data Hiding *Con't*

- ***Encapsulation*** and ***data hiding*** are key terms in the description of object-oriented languages.
- If you want to modify the data in an object, you know exactly what functions interact with it: the member functions in the object. No other functions can access the data. This simplifies writing, debugging, and maintaining the program.

# Example: A Point on the plane



A Point on a plane has two properties; x-y coordinates.

Abilities (behavior) of a Point are, moving on the plane, appearing on the screen and disappearing.

A model for 2 dimensional points with the following parts:

Two integer variables (**x** , **y**) to represent x and y coordinates

A function to move the point: **move** ,

A function to print the point on the screen: **print** ,

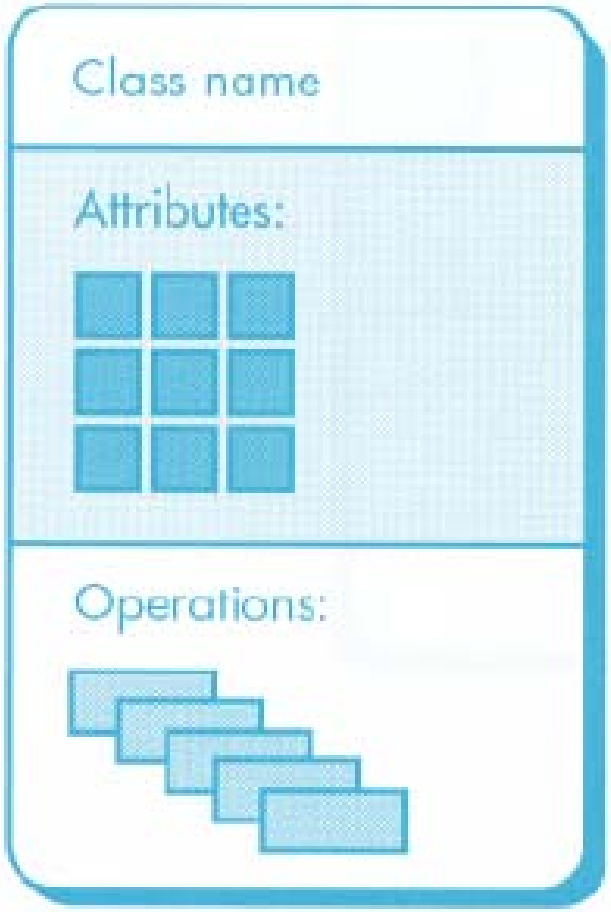
A function to hide the point: **hide** .

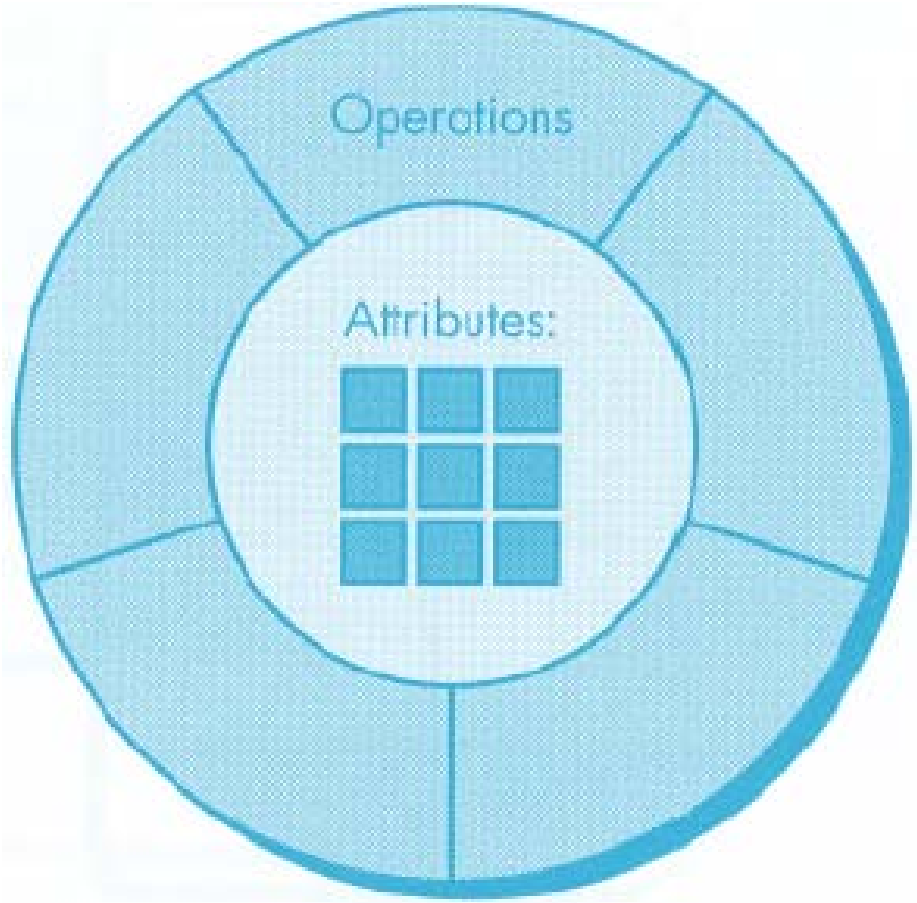
# Example: A Point on the plane *Con't*

Once the model has been built and tested, it is possible to create many objects of this model , in main program.

```
Point point1, point2, point3;
    :
point1.move(50,30);
point1.print();
```

# The Object Model



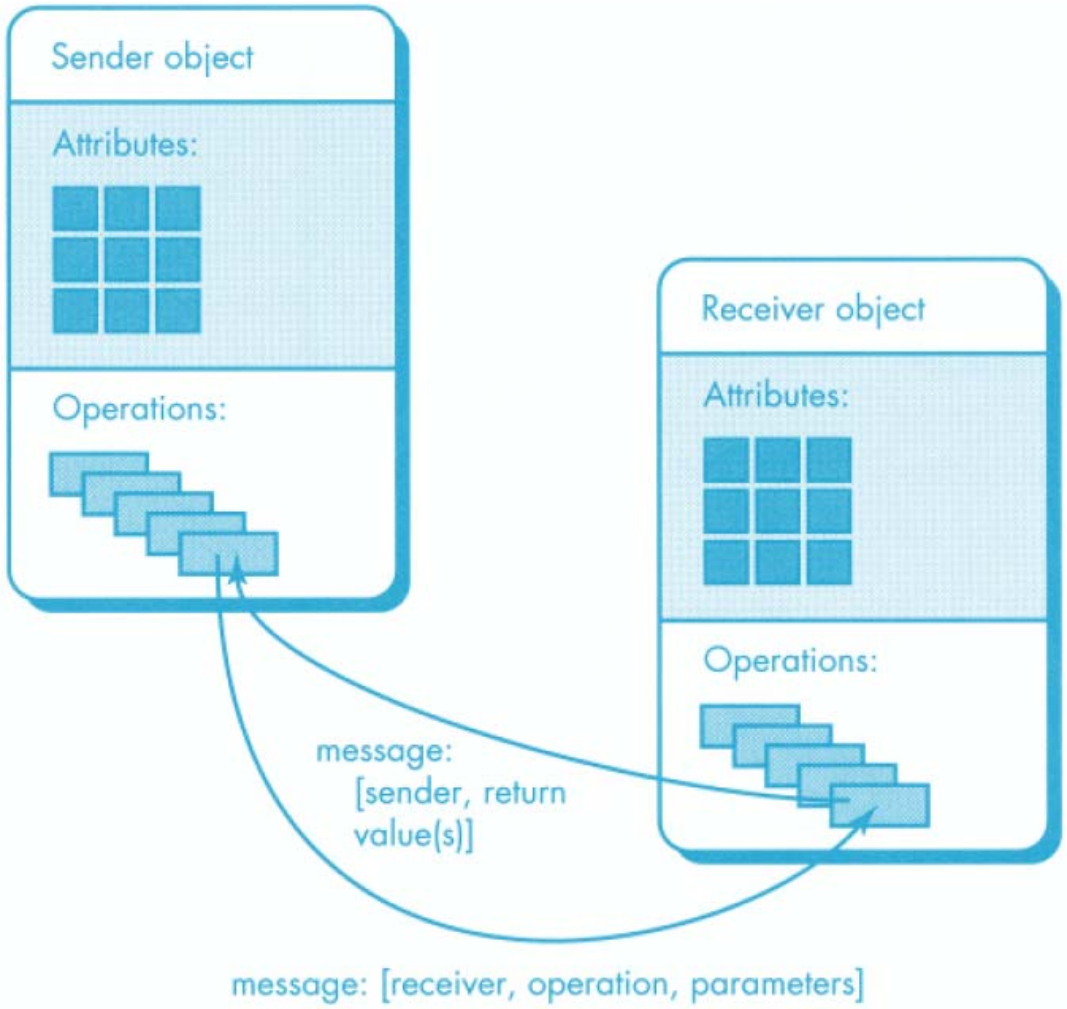




A C++ program typically consists of a number of objects that communicate with each other by calling one another's member functions.

# The Object Model

*Con't*





# OOP *vs.* Procedural Programming



Procedural Programming:

•Procedural languages still requires you to think in terms of the structure of the computer rather than the structure of the problem you are trying to solve.

•The programmer must establish the association between the machine model and the model of the problem that is actually being solved.

•The effort required to perform this mapping produces programs that are difficult to write and expensive to maintain. Because the real world thing and their models on the computer are quite different.

# Example: Procedural Programming *Con't*

- Real world thing: student
- Computer model: char *, int, float ...
- It is said that the C language is closer to the computer than the problem.

# OOP *vs.* Procedural Programming *Con't*



Object Oriented Programming

► The object-oriented approach provides tools for the programmer to represent elements in the problem space.

► We refer to the elements in the problem space and their representations in the solution space as "objects."

► The idea is that the program is allowed to adapt itself to the problem by adding new types of objects, so when you read the code describing the solution, you're reading words that also express the problem.

► OOP allows you to describe the problem in terms of the problem, rather than in terms of the computer where the solution will run.

# OOP *vs.* Procedural Programming *Con't*

- Benefits of the object-oriented programming:
  - Readability
  - Understandability
  - Low probability of errors
  - Maintenance
  - Reusability
  - Teamwork