

***Object-Oriented Software  
Development***  
**Goal and Scope**

**Koichiro Ochimizu  
Japan Advanced Institute of  
Science and Technologies  
School of Information Science**

**What is Software Engineering?**

## **Software Development is Challenging but Difficult to Achieve!**

- Software entities are more complex than most things people build like buildings, automobiles or VLSI.
- *Within only 30 years the amount of software in cars went from 0 to more than 10,000,000 lines of code. More than 2000 individual functions are realized or controlled by software in premium cars, today. 50-70% of the development costs of the software/hardware systems are software costs. (Manfred Broy, "Challenges in Automotive Software Engineering", ICSE2006, pp33-42,2006)*

## **Why is Software Development so difficult ? (F.Brooks,Jr)**

### **1. Complexity**

- Computer programs are complex by their nature: a huge amount of parts and their relationships.

### **2. Conformity**

- Software can not be created in isolation, but must conform to real-world constraints – pre-existing hardware , third party components, government regulations, legacy data formats, and so on.

## **Why is Software Development so difficult ? (F.Brooks,Jr)**

### **3. Changeability**

- Software is always evolving, as the outer environments of software change.

### **4. Invisibility**

- Software doesn't exist in a way that can be represented using geometric models, especially for representing the behavior of software.

## **Who makes such a complex software? ?**

- Human beings
- A group of human being should collaborate to complete the work within specified time and cost with producing high quality product.
- Difficult to deal with the following problems caused by human beings
  - **Variation**
  - **Uncertainty**
  - **Contingency**

## **Software Engineering can support their activities**

- **Software Engineering Technologies**
  - Provide us to control the problems specific to software developments
  - Support the team to proceed the work smoothly

## **Major Topics in Software Engineering**

- **Software Process Model (SPM)**

SPM provides for the strategy for software development
- **Project Management Technologies (PM)**

The application of knowledge, skills, tools and techniques to project activities to meet project requirement. Managing a project includes: Identifying requirements; Establishing clear and achievable objectives; Balancing the competing demands for quality, scope, time and cost (PMBOK).
- **Software Development Methodologies (SDM)**

SDM provides for the desirable structure of software and define the procedure how to form them

Several examples of structures :easy to verify correctness, easy to encapsulate the change impact, easy to divide the whole work into independent parts, easy to reuse, easy to evolve
- **Languages and Environments**

## History of SPM, SDM, PM

- Waterfall model (early in the 1970s)
- Development of Programming Methodologies (early in the 1970s)
- Development of Design Methodologies (late in the 1970s)
- Development of Requirement Engineering Technologies (late in the 1970s)
- Beginning of Technical Project Management (late in the 1970s to early in the 1980s)
- Improvement of Waterfall model (V model) (middle to late in the 1980s)
- Iterative Waterfall Model (mini waterfall, spiral) (early in the 1980s)
- Prototyping (early in the 1980s)
- Executable specifications and Formal Methods (middle in the 1980s)
- Process Programming (late in the 1980s)
- SPI (early in the 1990s)
- CASE tools (early in the 1990s)
- Architecture centric Development (middle in the 1990s)
- Object oriented software development technologies (after 1980s)
- Maturity of Software Assessment technologies (late in the 1990s)
- UML (late in the 1990s)
- Iterative Software Process Model(2000s)
- Agile (2000s)
- GORE, IR,COTS (middle of 2000s)

## Change of SPM

- Waterfall Model
  - Custom development
- V Model (System Engineering)
  - Outsourcing
- Iteration by Mini Waterfall Model or Spiral
  - Risk Management
- Prototyping
  - User involvement
- Iterative & Incremental SPM
  - Reduction of uncertainty by studying the project specific features

## Development of PM

- Various Measures
  - Cost-estimation
  - Detection of risky factors (Software complexity measures, V measure, E measure)
  - Decision support for terminating test activities (software reliability growth model)
- Measurement
  - Function Points
- CMM
  - Maturity Levels and Best Practices
- Software Assessment
  - Benchmark and Baseline
- PMBOK
  - Knowledge

## History of SDM

### What structures and How

- Structured Programming
  - easy to verify correctness a program, easy to divide the whole work into independent parts
- Information Hiding Module
  - Encapsulation of change impact
- Structured Analysis and Design
  - Encapsulation of change impact
- Requirement Engineering
  - Requirements definition
- Executable Specifications and Formal Methods
  - Verifying and proving some properties of a program, Generation of a program,
- Object-Orientation
  - easy to encapsulate the change impact, easy to reuse and easy to evolve a program
- Goal Oriented Requirement Engineering, Integrated Requirement Engineering, COTS
  - Shortening the development time

## Principles on Software Engineering

### 1. Rigor and Formality

Rigorous approach enables us to produce more reliable products, control their cost, and increase our confidence in their reliability. Formality is a stronger requirement than rigor; it requires the software process to be driven and evaluated by mathematical laws.

### 2. Separation of Concerns

To deal with different individual aspects of a problem and we can concentrate on each separately.

### 3. Modularity

Kind of Separation of Concerns. A complex system may be divided into simpler pieces called modules, allowing details of each module being handled in isolation.

### 4. Abstraction

Kind of Separation of Concerns; Separation of what from how. The we can identify the important aspects of a phenomenon and ignore its details.

### 5. Anticipation of Change

When likely changes are identified, special care must be taken to proceed in a way that will make future changes easy to apply.

### 6. Generality

Generalizing the problem to make the solution more potential one for being reused.

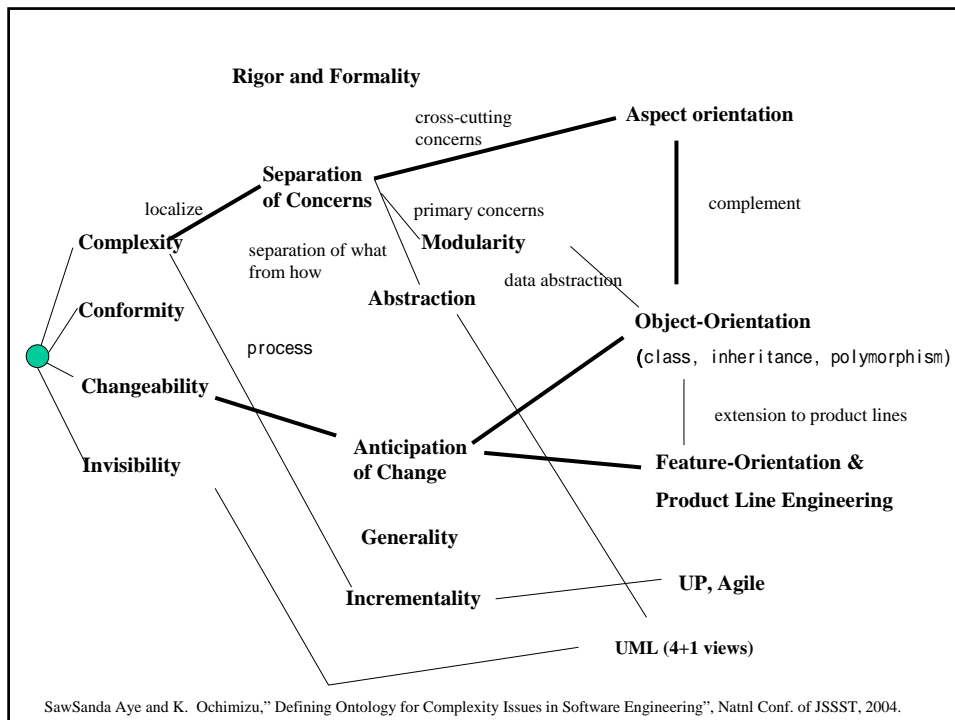
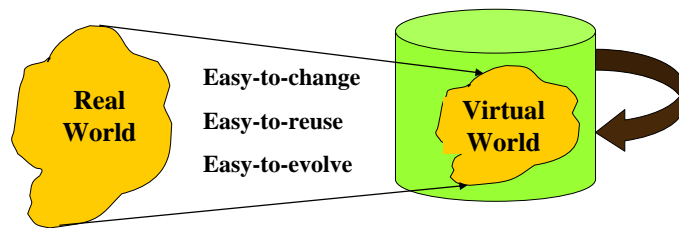
### 7. Incrementality

A process that proceeds in stepwise fashion, in increments, for risk reduction.

## Advantages of OOT

## OOT supports us to incorporate three major features into a system structure

- *Project the real world into the computer as you recognize and understand it.*
- *Virtual World can simulate the real world with some abstraction*
- *Maintain the virtual world constantly corresponding to mismatches between the real world and the virtual world and evolution of the real world.*



## Object-Oriented Technology

- **Mainly deal with Anticipation of Change**
  - **Class** for change of data structure
  - **Inheritance** for variation of properties over the several problem domains
  - **Polymorphism** for common signature of operations with different implementations

## Aspect Oriented Programming

- Aspect oriented programming complement the defects of object oriented programming. We can not encapsulate all of the functionality in objects (primary concerns). For an example, Codes related to logging spread over related objects (cross cutting concerns). Aspect can define a cross cutting concerns separately.

## Feature Modeling and Product Line Engineering

- **New technology for Anticipation of Change**  
Common features and Product Specific features are arranged by Variation Point Analysis Techniques

## Scope and Goal

- **Goal** enable you to understand basic principles and concepts of OOT and to apply them for practical use in software development.
- **Content**
  - Basic Principles and Concepts
  - Modeling Language(**UML**) and Programming Languages(**Java**)
  - Object-oriented Software Development Method  
(Use case-driven approach: **Unified Process, COMET**)
  - Case Study: **Elevator Control System**

## **Be Careful !**

- Just understanding
  - How to write UML diagrams
  - How to write Java programsis not enough

## **Please understand !**

- How to include several good features ;  
easy-to-change, easy-to-reuse, easy-to-evolve  
into a software structure through an object-oriented  
software development process
- Concepts and Principles independent of some  
specific languages like UML and Java.

## Important Concepts to be studied

- Class and Instance
  - Removal of redundant description
- Information hiding
  - Easiness of modifying a data structure
- Abstract Data Type
  - Both
- Inheritance
  - Reuse of classes by sub-classing
  - Easiness of extension of functions by sub-typing
- Polymorphism
  - Dynamic binding
- Use of the same concepts through analysis, design and programming
  - Simple correspondence among software artifacts

## Historical Survey of OOT

## Object-Oriented Programming

- 1967: Simula by O.J. Dahl **Class and Instance**
- 1972: Parnas Module by D.Parnas **Information hiding**
- 1972: Smalltalk72(Xerox PARC)
- 1974: CLU by B. Liskov **abstract data type**
- 1981: Smalltalk80 by Xerox **class library**
- 1986: Objective-C by Cox, C++ by Strusrup
- 1988: Eiffel by B. Meyer
- 1989: CLOS by Moon
- Now: Java, C++, C#

## Object-Oriented Technologies (Object Oriented Analysis and Design)

- 1986: OOD by G. Booch
- 1988: Shlare/Mellor,
- 1991: Coad/Yordon,
- 1991: OMT by J.Rumbaugh
- 1992: OOSE by Ivar Jacobson
- 1993-1994 Design Patterns by GoF
- 1997 CBSE by Szyperski
- 1997: UML
- 2004: UML 2 & MDA

## Causal Relationship of OO-Technologies

### Framework of Study on Software Engineering

- (1) Declare one's research interest definitely and briefly by observing the superficial problematic situations in a real world
- (2) Set up a hypothesis on a **Solution and its Effect** by abstracting an **Essential Problem**
- (3) Give a **name** both for an abstraction of problem and an effect of solution
- (4) Change your concerns to basic theoretical considerations apart from a real world. Concentrate your attentions on constructing a solution by using proper tools such as algebraic expressions, algorithms, languages, software tools and so on.
- (5) Evaluate effects of research results by performing a field test in a real world to find new requirements for technology evolution/revolution

## Information Hiding or Data Abstraction

- (1) The effects of modifying a data structure is scattered over a program. It makes modification of a program troublesome.
- (2) Effects of modified data structure is localized, if we can package both data structure and related operations in the same place of a source code.
- (3) Information Hiding or Data Abstraction, localization of change effects
- (4) Parnas's Module
- (5) Easiness of change of data structures was achieved.
- (6) The same or similar descriptions appear in a source code redundantly

## Abstract Data Type or Class

- (1) The same or similar descriptions appear in a source code redundantly, i.e. we should write code for each instance, if we realize the information hiding principle without type definition.
- (2) We can modify a code only once by defining a Parnas's Module as a type definition.
- (3) abstract data type (Class and Instances)
- (4) New Languages(e.g. CLU)
- (5) There are lot of similar class definitions in a source code.

## **(implementation) Inheritance**

- (1) There are lot of similar class definitions in a source code.
- (2) By arranging the similarity and the differences among classes as a tree structure, we can reuse an parent's implementation.
- (3) ( implementation ) Inheritance
- (4) Class Libraries for programming, UI, application domains
- (5) It is difficult to maintain class libraries especially for class libraries of some application domain, because inheritance exposes a subclass to details of its parent's implementation.

## **Interface inheritance (or sub-typing)**

- (1) Implementation Inheritance is not always useful for promoting reuse(super class sensibility)
- (2) Programming to an Interface, not an Implementation. single interface definition by abstract class and different method Implementation by concrete classes)
- (3) Interface inheritance(or sub-typing)
- (4) abstract class and concrete class
- (5) Interface definition is not so stable for a long time

## Advantages

- **Superiority of object oriented approach**
  - Localization of change effect( data abstraction or information hiding)
  - Removal of redundant description of Code by type definition( abstract data type)
  - Reuse by Sub Classing
  - Extensibility by Sub Typing

## Schedule(1/3)

- Feb. 20th
  - 13:00 Scope and Goal (History of SPMs and SDMs, History of OO-technologies)
  - 14:30 Basic Concepts on Representing the World (object, class, association, aggregation...)
- Feb. 21th
  - 13:00 Basic Concepts on Interaction (message passing, operation and method, polymorphism)
  - 14:30 Basic Concepts on Reuse (super class, class inheritance, interface inheritance)

## **Schedule(2/3)**

- Feb. 27th
  - 13:00 Introduction to Java Programming
  - 14:30 Outline of UML: Static Modeling  
(usecase modeling, details of class definition)
- Feb. 28th
  - 13:00 Outline of UML: Dynamic Modeling  
(state machine)
  - 14:30 Outline of UML: Dynamic Modeling  
(communication diagram, sequence diagram)

## **Schedule(3/3)**

- March 12
  - 13:00 Unified Process and COMET
  - 14:30 Case Study of Elevator Control System  
(problem definition, use case model)
- March 13
  - 13:00 Case Study of Elevator Control System  
(finding analysis classes by developing a consolidated communication diagram)
  - 14:30 Case Study of Elevator Control System  
(sub-system structuring and task structuring)
- March 14
  - 13:00 Case Study of Elevator Control System  
( performance analysis)
  - 14:30 UML2.0 and MDA

## References

- Hassan Gomaa, Designing Concurrent, Distributed And Real-Time Application with UML, Addison Wesley, (2000).
- James Rumbaugh et. al. : Object-Oriented Modeling and Design, Prentice-Hall Englewood, N.J., (1991).
- Ivar Jacobson et. al. : Object-Oriented Software Engineering -- A Use Case Driven Approach, ACM Press, (1992).
- Ivar Jacobson, James Rumbaugh, Grady Booch: The Unified Software Development Process, Addison-Wesley, (1999).
- James Rumbaugh, Ivar Jacobson, Grady Booch: The Unified Modeling Language Reference Manual, Addison-Wesley, (1999).
- James Rumbaugh, Ivar Jacobson, Grady Booch: The Unified Modeling Language Reference Manual, Second Edition, Addison-Wesley, (2005).
- Grady Booch, James Rumbaugh, Ivar Jacobson: The Unified Modeling Language User Guide, Addison-Wesley, (1999).

## References

- Hans-Erik Eriksson, Magnus Penker, "UML Toolkit", Wiley Publishing, Inc. 1998.
- Hans-Erik Eriksson, Magnus Penker, Brain Lyons, David Fado, "UML 2 Toolkit", Wiley Publishing, Inc. 2004.
- Hassan Gomaa, Designing Software Product Line with, Addison Wesley, (2004).
- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns", Addison-Wesley Publishing, 1995.
- Frank Bushmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, "Pattern-Oriented Software Architecture --- A System of Patterns" John Wiley & Sons, 1996
- Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Bushmann, "Pattern-Oriented Software Architecture Vol.2 --- Patterns for Concurrent and Networked Objects" John Wiley & Sons, Ltd, 2000.
- Martin Fowler, "Analysis Patterns: Reusable Object Models" Addison-Wesley Publishing Company, 1997.