Contents(1)

- **Goal and Scope**
- **Basic Concepts on OOT**
  - Basic Concepts to represent the world
  - Basic Concepts for Reuse
  - Information Hiding Principle and Java Program
  - Superiority of OOT
- **Modeling Techniques**
  - Static Model: Class and Association
  - Dynamic Model: State Machine
  - Dynamic Model: Interaction Diagram
  - Concurrency Description: Active Object and Multi-thread Programming
  - Outline of UML2.0

Superiority of Object Oriented approach

Koichiro Ochimizu

School of Information Science

JAIST

JAIST Koichiro Ochimizu

Framework of Study on Software Engineering

- (1) Declare one's research interest definitely and briefly by observing and abstracting 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

Structured Programming

(1) Human beings do not have enough ability to develop the large-scaled software system systematically with assuring the correctness of the program during development

(2) **Pearl and Necklace** (Assume a virtual machine that can solve the problem directly using ideal instruction set and data structure. Construct lower level virtual machines which can simulate the instruction of the upper level virtual machine, by performing decomposition of instruction and data type exclusively) .
Stepwise Refinement using one-entry and one-exit control structure during decomposition to enable each refinement to do independently to control the complexity of the work.

(3) Structured Programming

(4) Pascal, C

(5) The effect of data type refinement is scattered over a program.

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 sensitibity)
  (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

Object Composition

- (1) Inheritance is not almighty for reuse. It is ridiculous to define sub classes for all possible combination in the case of “is-a-role-played-by”.
  (2) Prepare components(objects) with primitive functions. Facade combine them to response the original message from the other object(delegation)
  (3) Object Composition and Delegation
  (4) Providing component library and connector class
  (5) Third-party components are not always reliable

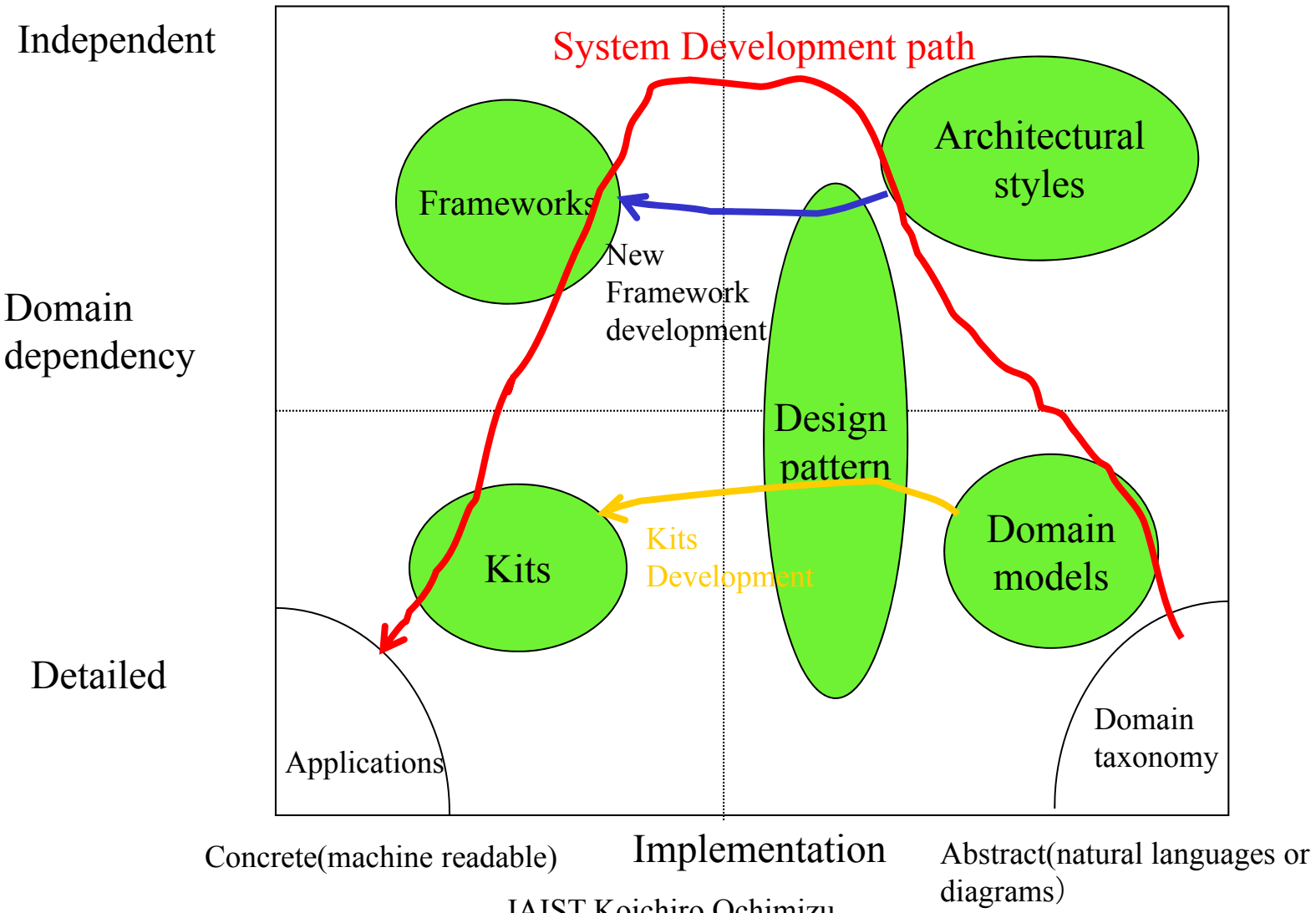
Design Patterns

- (1) Granularity of reuse is too small when we reuse classes. Experts on OOP use the recurring patterns, collaborations of several classes
  (2) Applying the Information Hiding Principles more generally. Reuse of collaboration of objects as micro architectures.
  (3) Design Patterns
  (4) Catalogue of Design Patterns
  (5) too many patterns

Achievement

- **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

Relationships



[38]W.M.Tepfenhart and J.J. Cusick, "A Unified Object Topology", IEEE Software Jan/Feb. pp.31-35,1997.