Software Modeling & Analysis

- Introduction -

정세진

Dependable Software Laboratory
KU KONKUK UNIVERSITY

T.A.

- 이름: 정세진
- 연구실: 신공학관 1219호
- E-mail: sjjung.dslab@gmail.com
  - [2019SMA]OOO~

An Introduction to Object-Oriented Development (OOD) & Software Engineering (SE)

Software Development

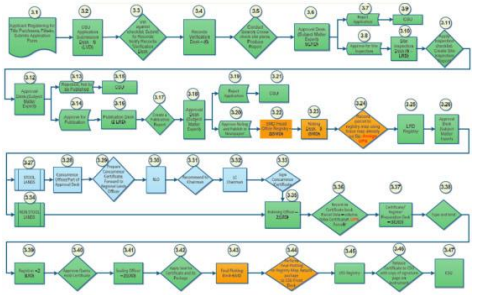
- Software Development ≈ Solving Problem with Software in Computer

Problems in real world

Natural Language
→ Descriptions of **Problems**
(through Identifying Requirements)

+

Business Process

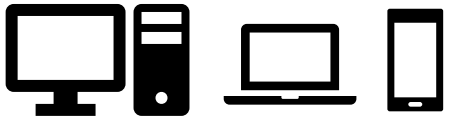


A Big Gap between Languages

Solutions in computer

Programming Language
→ Descriptions of **Solutions**
(through Designing Programs)

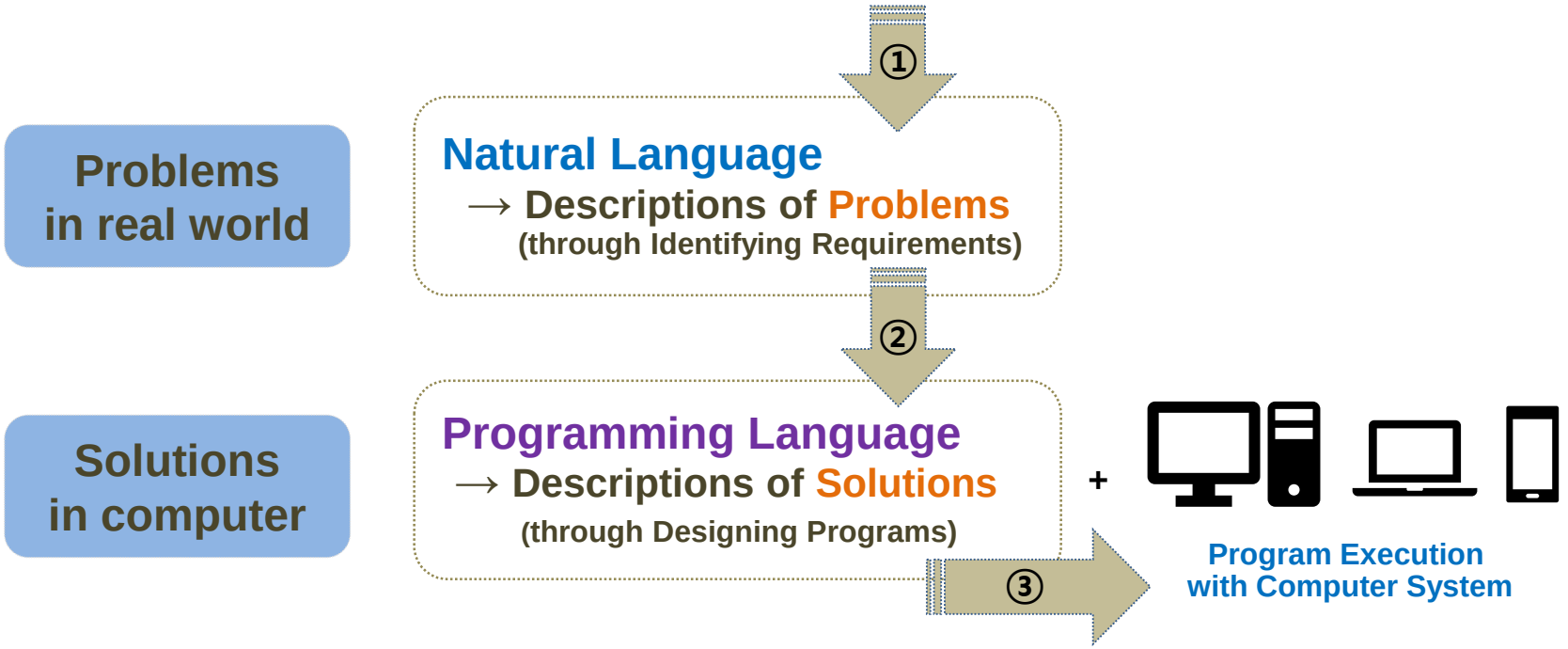
+



Program Execution with Computer System

Software Development

- Software Development ≈ Solving Problem with Software in Computer



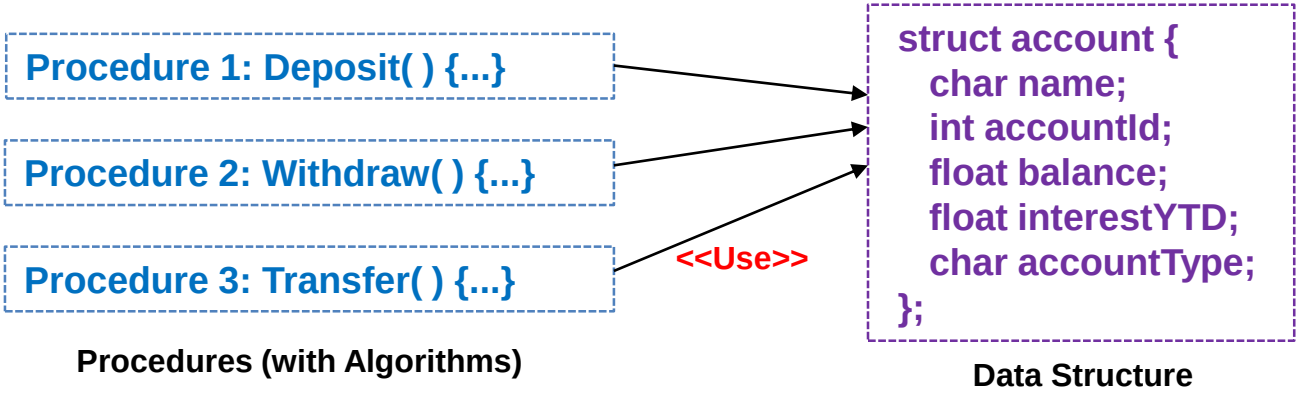
Software Development ≈ ①②③ → Procedural Programming → SASD

Object-Oriented Programming → OOAD

Computational Thinking ≈ ①②/2

Procedural Programming

- A program is organized with **procedures**.
  - **Procedure/Function**
    - building-block of procedural programs
    - **statements** changing values of **variables**
  - Focusing on data structures, algorithms, and sequencing of steps
    - **Algorithm** : a set of instructions for solving a problem
    - **Data structure** : a construct used to organize data in a specific way
  - Most computer languages (from FORTRAN to **C**) are procedural ones.

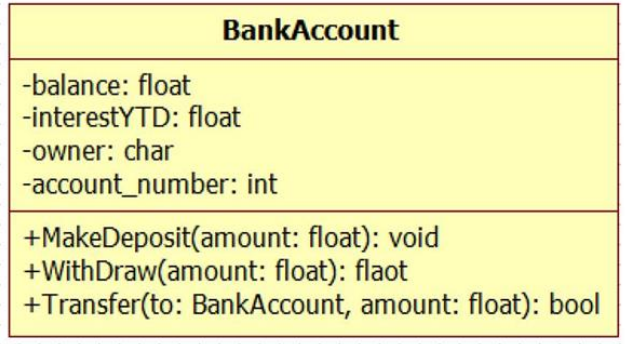


Object-Oriented Programming

- A program is organized with **objects**.
  - Focusing on objects and their communications.
    - **Object** : consisting of **data** and **operations** (functions)
    - **Object communication** : an object **calls** an operation of other objects with its data
  - Providing system functionalities through object communications
    - No explicit data flow
    - Only **communication sequences** among objects

data

operation




```
Class BankAccount {
  private:
    float balance;
    float interestYTD;
    char * owner;
    int account_number;
  public:
    void Deposit (float amount) {…}
    float WithDraw (float amount) {…}
    bool Transfer (BankAccount to, float amount) {…}
};
```

Object-Oriented Programming - OOAD

- **OOAD** (Object-Oriented Analysis and Design)
  - A software development methodology for Object-Oriented programs
  - OOA + OOD

- **Object-Oriented Analysis (OOA)**
  - Discover the domain concepts/objects (the objects of the problem domain)

- **Object-Oriented Design (OOD)**
  - Define software objects (static)
  - Define how they collaborate to fulfill the requirements (dynamic)

An OOAD Example - Dice Game

Define use cases → Define domain model → Define interaction diagrams → Define design class diagrams

OOA

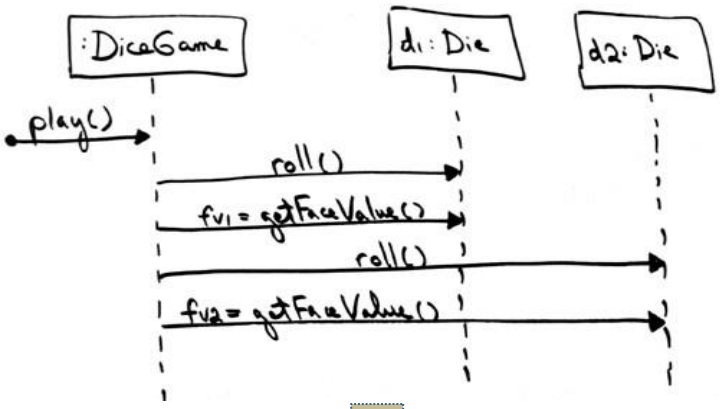
Use Case : Play a Dice Game

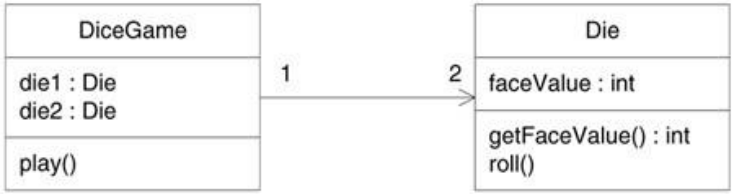
- Player requests to roll the dice.
- System presents results.
- If the dice's face value totals seven, player wins; otherwise, player loses.

Domain Model

OOD

Interaction Diagram





Design Class Diagram

Software Engineering

- Several definitions
  - The application of a systematic, disciplined, quantifiable approach to development, operation and maintenance of software [IEEE Standard 610.12]
  - The disciplined application of engineering, scientific, and mathematical principles and methods to the economical production of quality software [Watts Humphrey]
- Software Engineering is
  - All activities to develop and manage software well.
  - Theories, methods and tools for professional software development.

Software Engineering

- **Software engineering** is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use

- Engineering discipline
  - Using appropriate theories and methods to solve problems bearing in mind organizational and financial constraints.

- All aspects of software production
  - Not just technical process of development
  - Also project management and the development of tools, methods, etc. to support software production.

Importance of Software Engineering

- **A number of software projects fail due to**
  - Increasing system complexity
    - While new software engineering techniques help us to build larger and more complex systems, the demands change constantly.
    - Systems have to be built and delivered more quickly; larger, even more complex systems are required; systems have to have new capabilities that were previously thought to be impossible.
  - Failure to use software engineering methods
    - It is fairly easy to write computer programs without using software engineering methods and techniques.
    - Many companies do not use software engineering methods in their everyday work. Consequently, their software is often more expensive and less reliable than it should be.

- We need to be able to produce reliable and trustworthy systems economically and quickly(on time).

- It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project.

The Software Process

- A structured set of activities required to develop a software system.

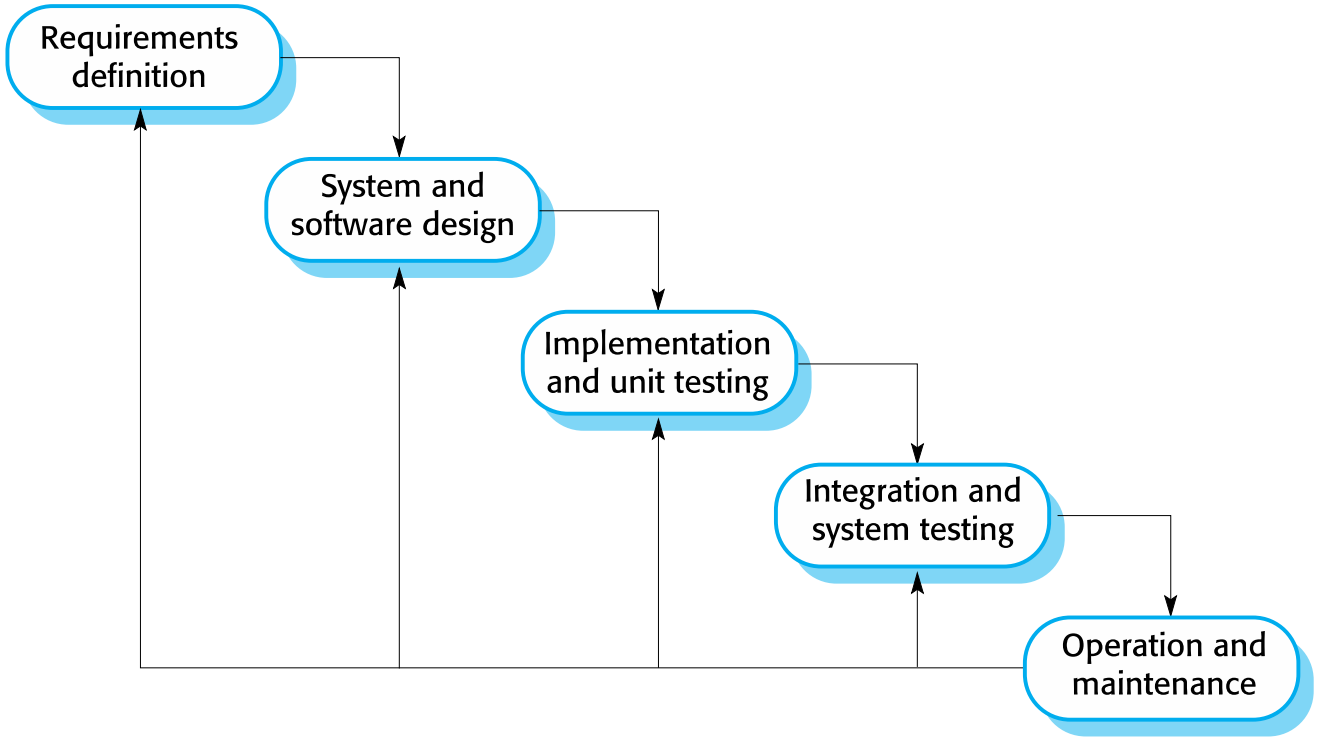
- Many different software processes, but all involve:
  - Specification : defining what the system should do
  - Design and implementation : defining the organization of the system and implementing the system.
  - Validation : checking that it does what the customer wants
  - Evolution : changing the system in response to changing customer needs

- A **software process model** is an abstract representation of a process.
  - Describes a process from some particular perspective.
    - Activities in the process
    - The ordering of these activities

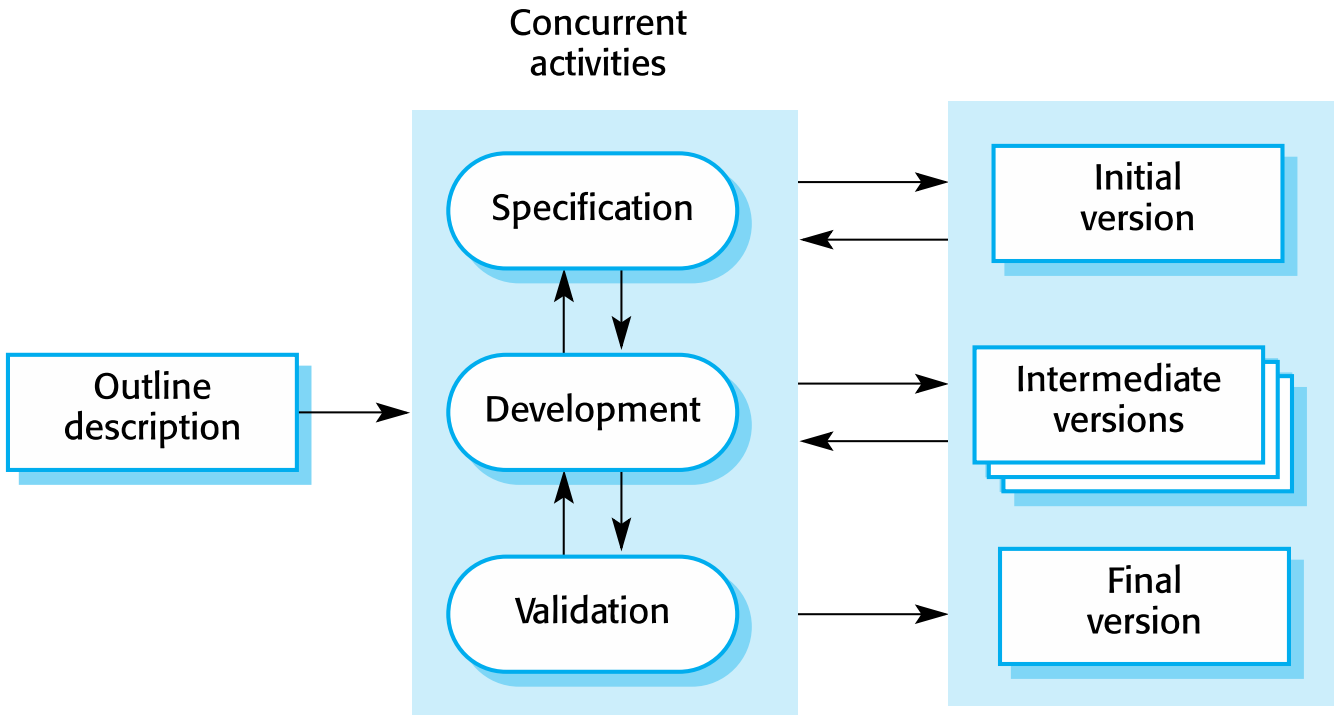
Software Process Models

- The waterfall model
  - Plan-driven model
  - Separate and distinct phases of specification and development
- Incremental development
  - Specification, development and validation are interleaved.
  - May be plan-driven or agile.
- Integration and configuration (Component-based Development)
  - The system is assembled from existing configurable components.
  - May be plan-driven or agile.
- In practice, most large systems are developed using a process that incorporates elements from all of these models.

The Waterfall Model

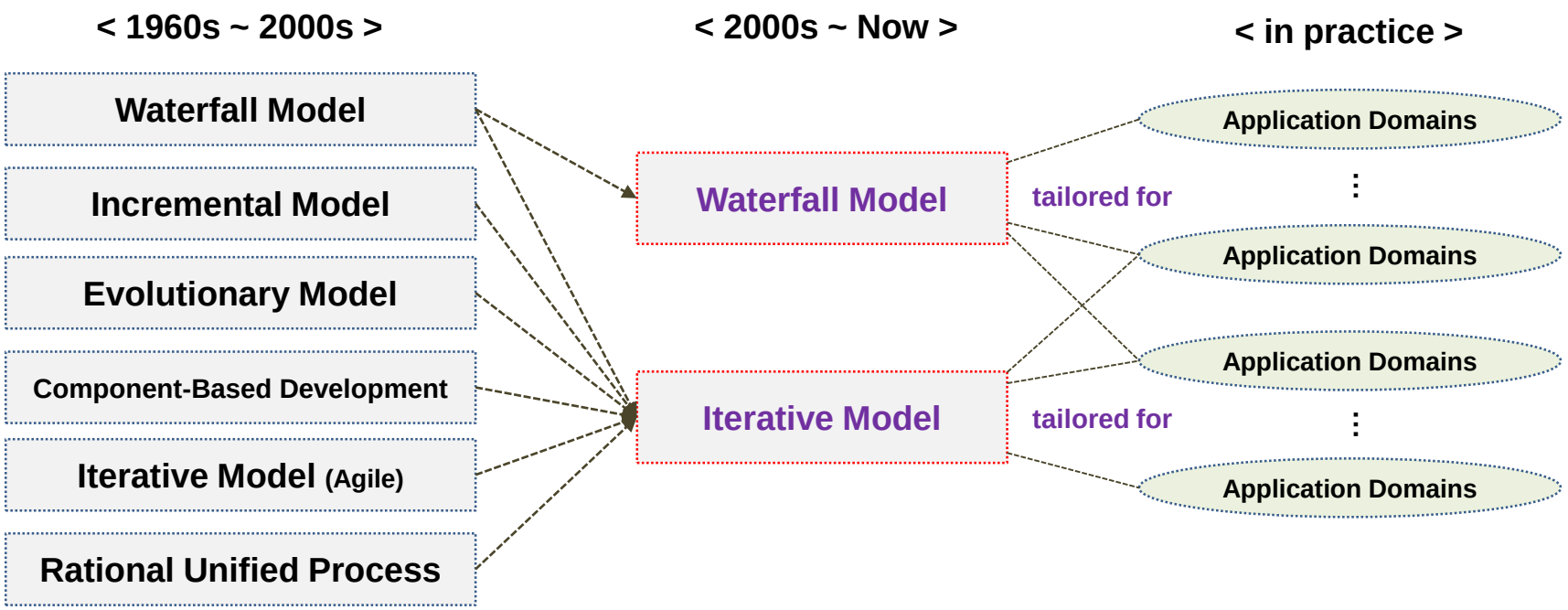


Incremental Development



Software Process Model

- **Software (Development) Process models**
  - Defining a distinct set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software, systematically.
  - Defining **Who** is doing **What**, **When** to do it, **How** to reach a certain goal.

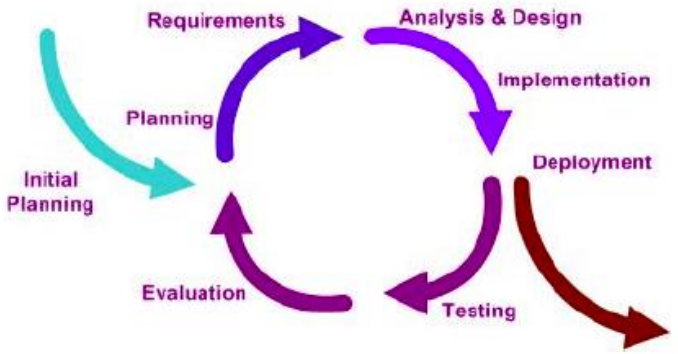


Iterative Model - Agile

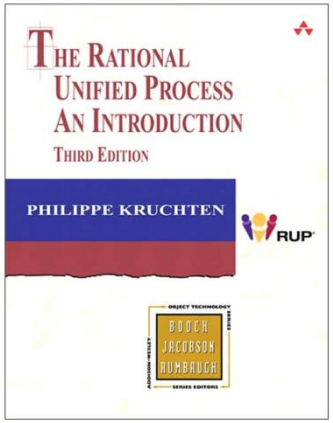
- **Agile development** is **an umbrella term** a group of methodologies weighting rapid prototyping and rapid development experiences.
  - Lightweight in terms of documentation and process specification
  - Example: XP(eXtreme Programming) , TDD(Test Driven Development)

- Agile methods attributes
  - Iterative (several cycles)
  - Incremental (not delivering the product at once)
  - Actively involve users to establish requirements

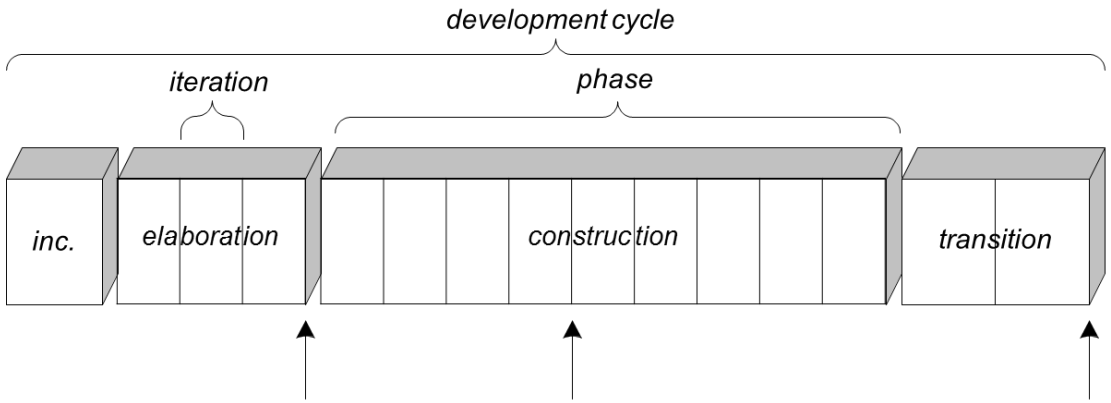
- Agile Manifesto
  - Individual over processes and tools
  - Working software over documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan



Iterative Model - UP



- **Rational Unified Process (RUP) or UP**
  - A Software development approach that is
    - **Iterative (Incremental, Evolutionary)**
      - Each iteration includes a small waterfall cycle.
    - **Risk-driven / Client-driven / Architecture-centric**
    - **Use-case-driven**
  - A Well-defined and well-structured software engineering process
    - 4 Phases and 9 Disciplines
  - A de-facto industry standard for developing OO software



An iteration end-point when some significant decision or evaluation occurs.

A stable executable subset of the final product. The end of each iteration is a minor release.

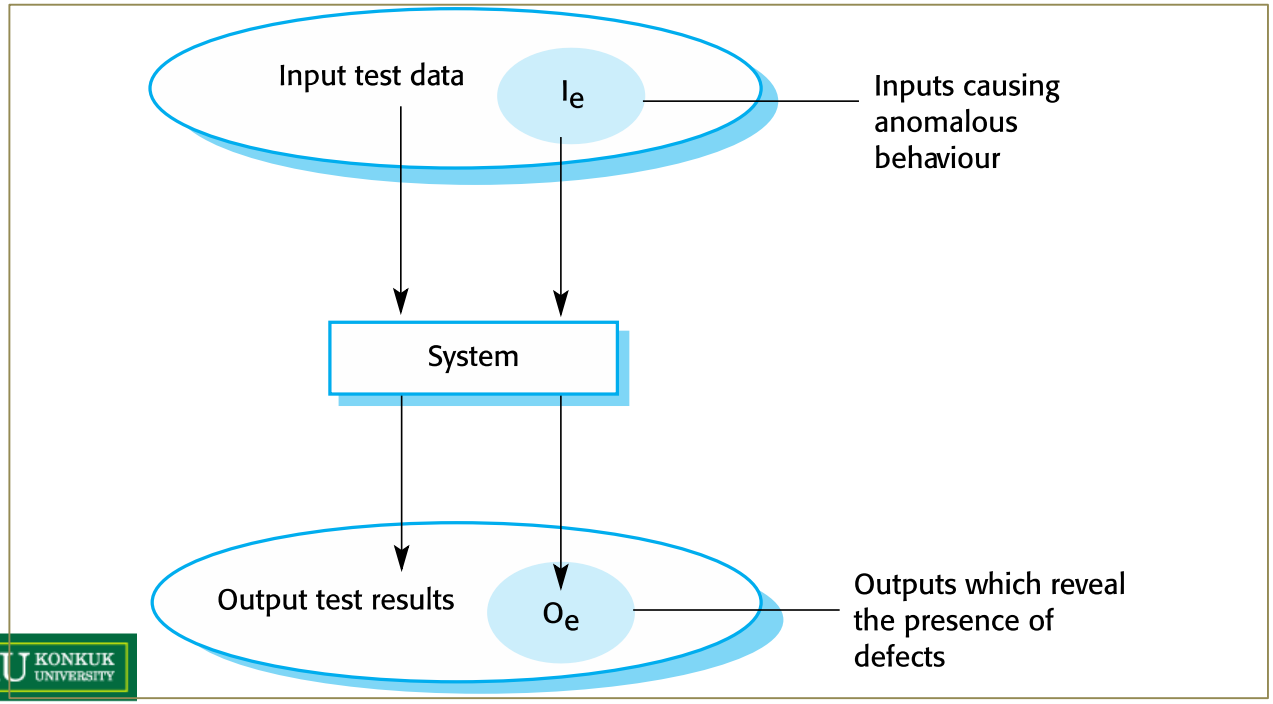
The difference (delta) between the releases of 2 subsequent iterations.

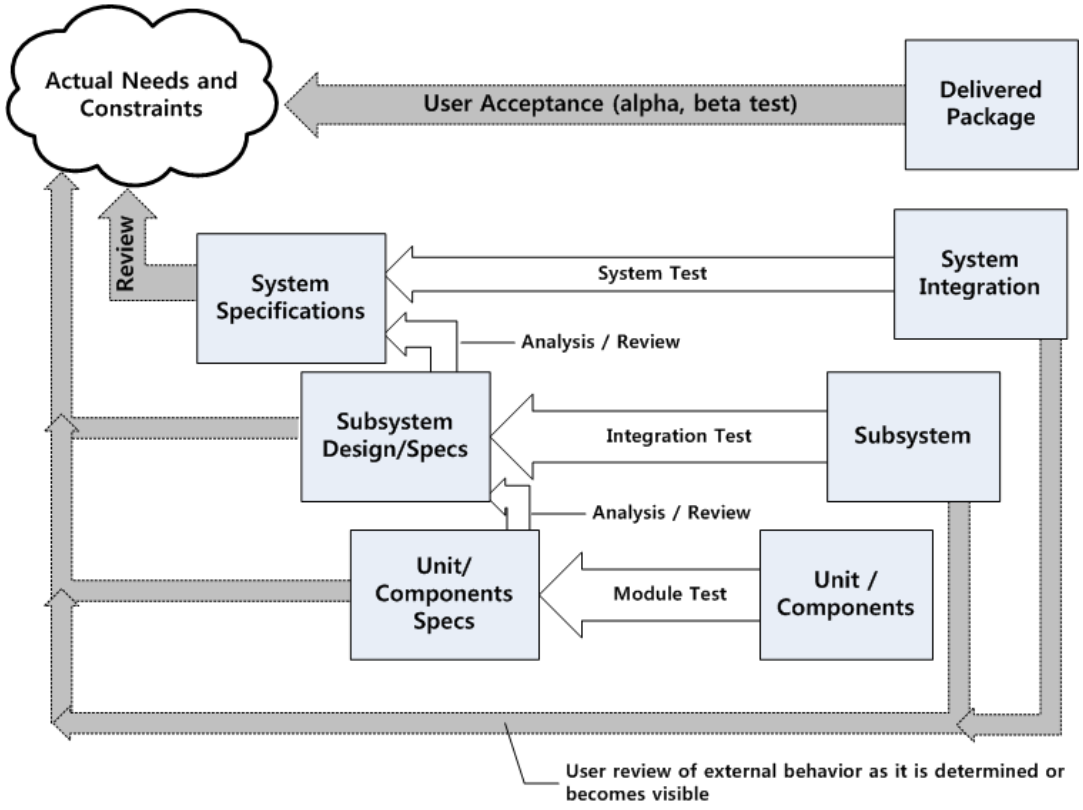
At this point, the system is released for production use.

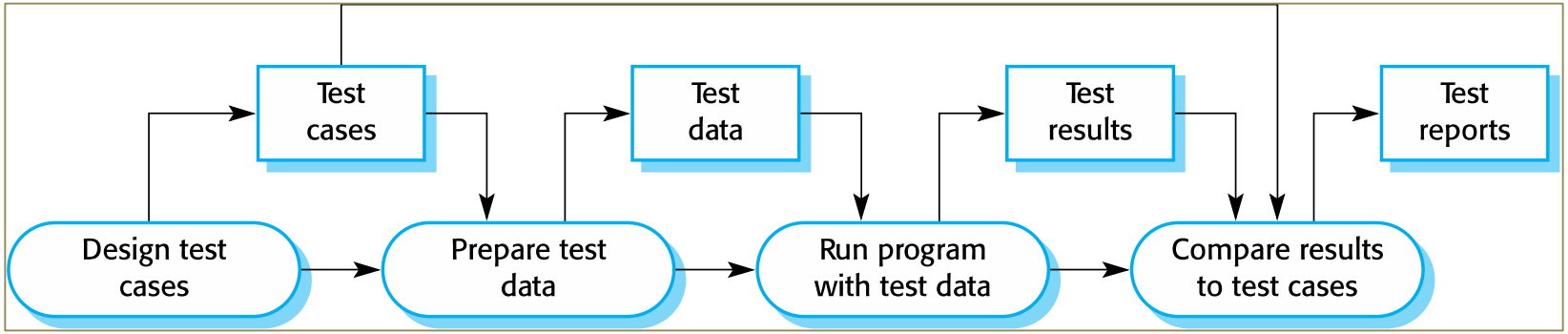
Verification & Validation

- **Validation testing**
  - To demonstrate to the developer and the system customer that the software meets its requirements.
  - A successful test shows that the system operates as intended.

- **Verification (Defect) testing**
  - To discover faults or defects in the software where its behavior is incorrect or not in conformance with its specification.
  - A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.




Actual Needs and Constraints
User Acceptance (alpha, beta test)
Delivered Package
Review
System Specifications
System Test
System Integration
Analysis / Review
Subsystem Design/Specs
Integration Test
Subsystem
Analysis / Review
Unit/ Components Specs
Module Test
Unit / Components
User review of external behavior as it is determined or becomes visible


Test cases
Test data
Test results
Test reports
Design test cases
Prepare test data
Run program with test data
Compare results to test cases

Stages of Testing

- Development testing
  - The system is tested during development to discover bugs and defects.
    - Unit testing
    - Integration testing
    - System testing
- Release testing
  - A separate testing team test a complete version of the system before it is released to users.
- User testing
  - Users or potential users of a system test the system in their own environment.

Development Testing

- **All testing** activities that are carried out by the team developing the system.
  - **Unit testing**
    - Individual program units or object classes are tested.
    - Unit testing should focus on testing the functionality of objects or methods.
  - **Component testing (= Integrated testing)**
    - Several individual units are integrated to create composite components.
    - Component testing should focus on testing component interfaces.
  - **System testing**
    - Some or all of the components in a system are integrated and the system is tested as a whole.
    - System testing should focus on testing component interactions.

Software Change

- **Software change** is inevitable.
  - New requirements emerge when the software is used.
  - The business environment changes.
  - Errors must be repaired.
  - New computers and equipment is added to the system.
  - The performance or reliability of the system may have to be improved.

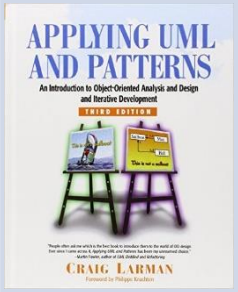
- A key problem for all organizations is to implement and manage the change to their existing software systems.

- Evolutionary Development (incremental, iterative model) is needed
  - **And traceability analysis**

Contents in Detail

대주제	차시	소주제	학습 목표	상세 내용
1. An Introduction to Object-Oriented Development	1 2	Object-Oriented Development	• '소프트웨어 개발'을 정의할 수 있다. • OOAD 와 SASD의 차이점을 구분할 수 있다. • 다양한 소프트웨어 개발 방법론/프로세스를 구분하고 정리할 수 있다.	• OOAD vs. SASD • Software Development Process
	3	Object-Oriented	• 객체지향 (Object-Oriented)을 정의할 수 있다.	• Object-Oriented
	4	Object-Oriented Principles	• 객체지향 Principles을 이해하고 적용할 수 있다.	• Object-Oriented Principles
	5 6	UML	• UML 2.0을 구성하는 13개 다이어그램들의 목적을 이해할 수 있다.	• 13 UML Diagrams

Contents in Detail

대주제	차시	소주제	학습 목표	상세 내용
2. Object-Oriented Analysis and Design 	7	Part I. Introduction	• OOAD 및 UP 기본개념을 정리할 수 있다. • 교재의 Case Study 내용을 확인할 수 있다.	• Chapter 1. Object-Oriented Analysis and Design • Chapter 2. Iterative, Evolutionary, and Agile • Chapter 3. Case Studies
	8 9	Part II. Inception	• UP 기반 OOAD의 첫 단계인 Inception 단계를 이해할 수 있다. • Inception 단계의 활동을 수행할 수 있다. • 기능/비기능 요구사항을 구별할 수 있다. • Use Case를 활용할 수 있다.	• Chapter 4. Inception is Not the Requirements Phase • Chapter 5. Evolutionary Requirements • Chapter 6. Use Cases • Chapter 7. Other Requirements
	10	Part III. Elaboration Iteration 1 – Basics - OOA	• Analysis 단계의 활동을 이해할 수 있다. • Domain model의 목적을 이해하고 활용할 수 있다.	• Chapter 8. Iteration 1 Basics • Chapter 9. Domain Models
	11	- OOA	• Sequence diagram의 목적을 이해하고 활용할 수 있다. • Operation contract의 목적을 이해할 수 있다.	• Chapter 10. System Sequence Diagram • Chapter 11. Operation Contracts
	12	- OOD	• Design 단계의 활동을 이해할 수 있다. • Package diagram의 목적을 이해하고 활용할 수 있다.	• Chapter 12. Requirements to Design Iteratively • Chapter 13. Logical Architecture and UML Package Diagrams
	13 14	- OOD	• Sequence diagram의 목적을 이해하고 활용할 수 있다.	• Chapter 14. On to Object Design • Chapter 15. UML Interaction Diagram
	15 16	- OOD	• Class diagram의 목적을 이해하고 활용할 수 있다.	• Chapter 16. UML Class Diagram
	17	- OOD	• GRASP 디자인 패턴의 목적과 효과적인 적용 방법을 이해할 수 있다.	• Chapter 17. GRASP: Designing Objects with Responsibilities
	18 19	- OOI	• OO Design에서 Implementation으로의 전환과정을 정확하게 이해할 수 있다. • 개발방법론의 장점을 확인할 수 있다.	• Chapter 19. Designing for Visibility • Chapter 20. Mapping Designs to Code

Contents in Detail

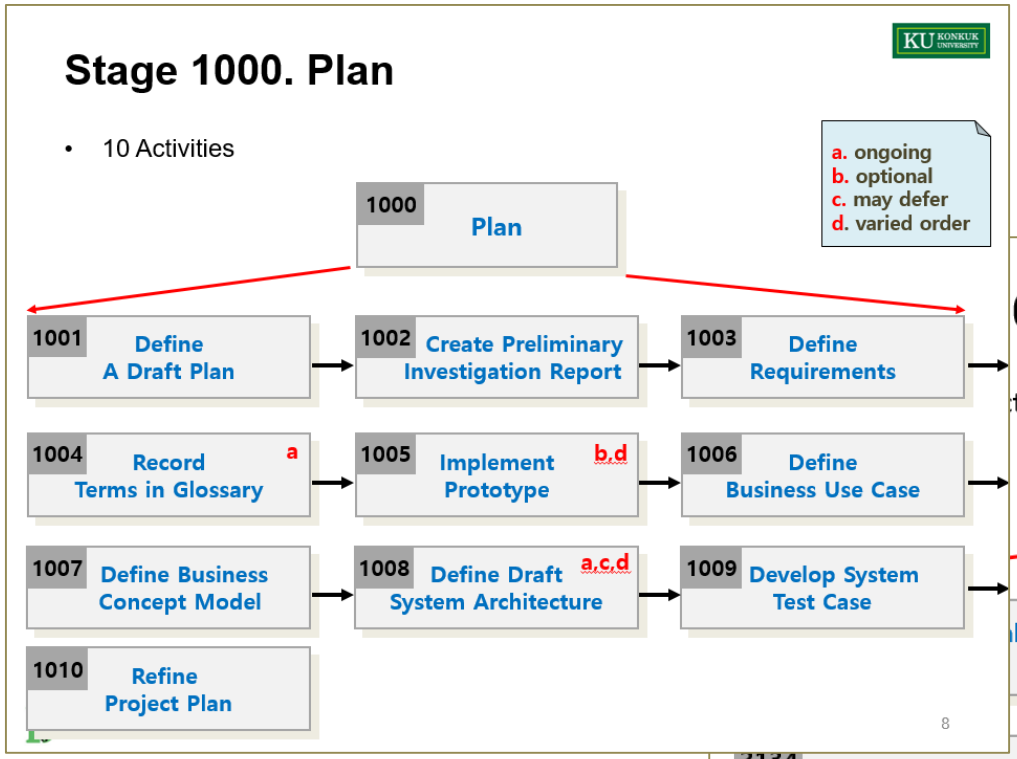
대주제	차시	소주제	학습 목표	상세 내용
3. Advanced Topics in UML	20 21	Statechart Diagram	• Statechart의 문법을 정확하게 이해하고, 이를 활용하여 모델링을 수행할 수 있다.	• Statechart Diagram
	22	Component Diagram	• Component Diagram을 이해하고 활용할 수 있다.	• Component Diagram
	23	Extension Mechanism of UML	• UML을 적절하게 확장하는 방법을 이해할 수 있다. • MOF의 개념을 이해할 수 있다.	• Extension Mechanism of UML

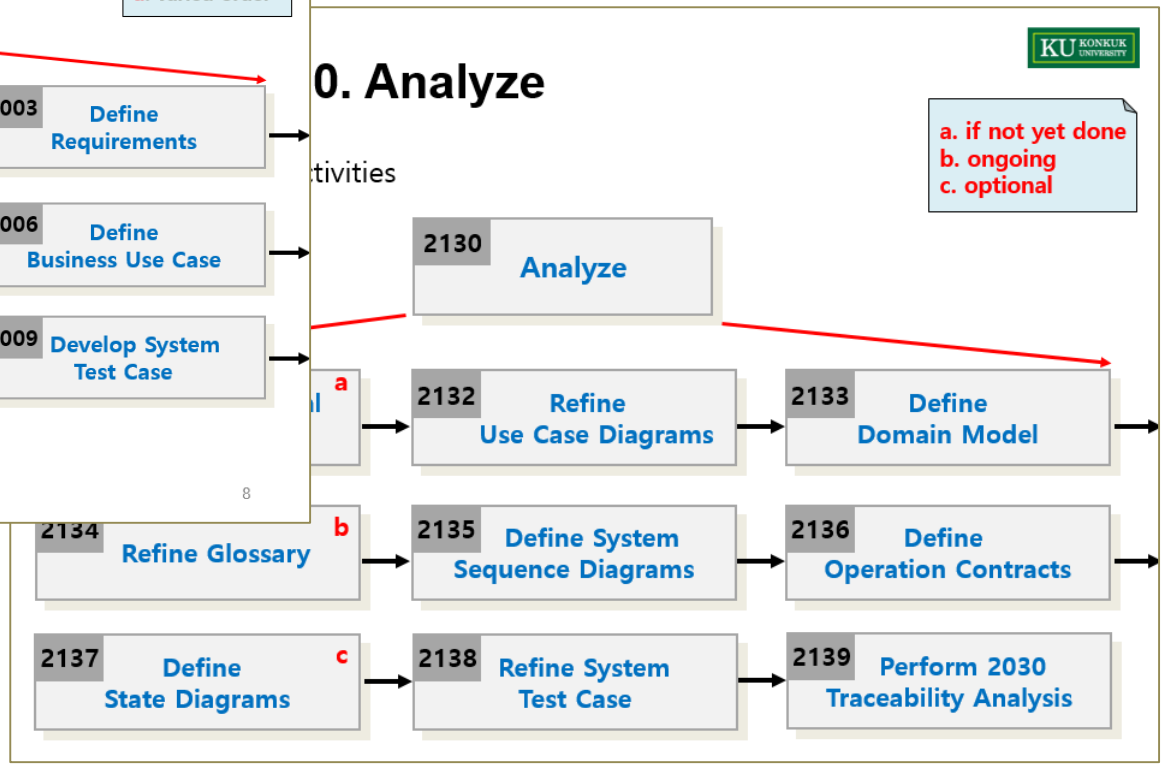
대주제	차시	소주제	학습 목표	상세 내용
4. Summary	24	OOAD Summary	• UML을 적절하게 사용하여, UP 기반의 OOAD를 수행할 수 있는 이론적인 배경을 갖춘다.	• OOAD Summary

Team Activity
OOPT - A New OOO Digital Watch

Team Project with OOPT

- A software development process based on RUP
- Tailored to software engineering classes in universities
- A revision of OSP (Object Space Process)
- Have been practiced and refined for 10 years





A New OOO Digital Watch

- Supposed to develop a new OOO digital watch

- Let's analyze and design your own new OOO digital watch.
  - OOAD development method : **OOPT**
  - Use a UML tool
    - Not use Statechart, Communication, Activity, Package, Deployment Diagrams, for now.
  - Basic Requirements & Assumption :
    - A set of predefined/fixed hardware (1 LCD, 4 buttons, 1 buzzer, 1 SW Controller)
    - Dynamic SW Configuration (4 among 6 functions)
      - **Time keeping, Timer, Stopwatch, Alarm (+2 function)**
    - 4 alarms
  - Instructions
    - Take care of the layered architecture of your system under development.
    - Take care of your system context – embedded system!
    - Make every assumptions clear, feasible and consistent.

- **Team activities:**
  1. Stage 1000 : Plan
  2. Stage 2000 > 2030 : Analyze
  3. Stage 2000 > 2040 : Design
  4. Stage 2000 > 2050 : Implementation

A New OOO Digital Watch

- Supposed to develop a new OOO digital watch

- Let's analyze an
  - OOAD develo
  - Use a UML t
    - Not use S
  - Basic Require
    - A set of p
    - Dynamic S
    - 4 alarms
  - Instructions
    - Take care
    - Take care
    - Make eve

, for now.

ller)

- **Team activities:**
  1. Stage 1000
  2. Stage 2000
  3. Stage 2000
  4. Stage 2000

Team Project

- 메일로 팀 구성해서 보내 주세요 (**학번, 이름**)
  - 4인 1팀

Schedule

Week	Date	월요일 (12:30~14:30) - 신공학관 1214호	금요일 (14:30~16:30) - 신공학관 1214호
1	03.04 / 03.08	Course Introduction - Lecture Note	**Lab. Orientation** *
2	03.11 / 03.15	1/2 Object-Oriented Development	3/4 Object Oriented & Principles
3	03.18 / 03.22	OOPT Stage 1000 - Plan & Elaboration * **Case Study - LMS** Case Study - PRINTER	5/6/7/8/9 UML , Domain Model
4	03.25 / 03.29	**Team Practice #1**	**Team Presentation #1 - OOPT Stage 1000**
5	04.01 / 04.05	10/11 System Sequence Diagram , Operation Contracts	OOPT Stage 2030 - Analyze * **Case Study - LMS** Case Study - PRINTER
6	04.08 / 04.12	**Team Practice #2**	**Team Presentation #2 - OOPT Stage 2030**
7	04.15 / 04.19	12/13 Logical Architecture	14/15 Interaction Diagrams
8	04.22	Midterm Exam. *	
9	04.29 / 05.03	16 Class Diagram	OOPT Stage 2040 - Design * **Case Study - LMS** Case Study - PRINTER
10	05.06 / 05.10	(공휴일)	**Team Practice #3 - CTIP 환경 전수** *
11	05.13 / 05.17	**Team Presentation #3 - OOPT Stage 2040**	OOPT Stage 2050 - Construct * OOPT Stage 2060 - Testing **Case Study - LMS** Case Study - PRINTER
12	05.20 / 05.24	17/19/20 GRASP , Visibility	22/23 Component Diagram , MOF
13	05.27 / 05.31	**Team Presentation #4 - 1st Cycle**	**Team Practice #4** *
14	06.03 / 06.07	**Team Presentation #5 - 2nd Cycle**	**Team Practice #5**
15	06.10 / 06.14	**Team Presentation #6 - 3rd Cycle**	(Reserved)
16	06.17	Final Exam.	