**Chapter 1**

**UML(Unified Modeling Language)**

1. It is a standard diagramming notation.
2. It is not OOA/D or a method It is just diagramming notation
3. **Thinking in Objects is more important**
4. It is a **visual** language for specifying, constructing and documenting the artifacts of systems.

**- Three ways to apply UML**

1. Sketch – Informal and Incomplete diagram
2. Blueprint(청사진) – reverse engineering to visualize and better understanding existing code in UML diagram  
   - code generation
3. Programming language – Automatically generated. (아직 개발중)d

**- Three Perspectives to Apply UML**

1. Conceptual – diagram are interpreted as describing in a situation of the real world or domain of interest
2. Specification (software) 명세 – describe software abstraction or components with specification and interfaces. **- Use X**
3. Implementation – Such as java, diagram describe software implementation **- Use**

**The meaning of “Class” in Different Perspectives**

1. Conceptual – real world concept or thing. UP Domain Model
2. Software – regardless of the process or method, Class representing a specification or implementation perspective of a software component.
3. Implementation – Such as Java, implemented a specific OO language

Responsibility-driven design

1. How should objects collaborate?
2. What classes should do what?

Case Studies

1. ongoing case studies

Use cases

1. requirement analysis

Iterative Development

Agile Modeling

Agile Up

**Responsibility assignment**

**A critical ability in OO development is to skillfully assign responsibilities to software objects.**

1. robustness(견고함)
2. maintainability(유지보수성)
3. reusability(재사용성)

Analysis & Design

- Analysis

1. investigation of the problem
2. requirements
3. Broad term : requirement analysis ( an investigation of the requirments) OR   
   **objective oriented analysis**( an investigation of the domain objects)

- Design

1. Conceptual solution(개념적 해결) more important implementation
2. Broad term : objective-oriented design, database design

**objective oriented analysis**  
- finding, and describing the object  
- Ex) Plane, Flight, Pilot

**objective-oriented design**

**-** defining software objects

- how they collaborate to fulfill the requirements

- Ex) tailNumber Attr / getFightHistory Method

**Example**

**1) Define Use Cases**

- Stories and scenarios of how people use the application.

- They aren’t an object-oriented artifact(산출물) But they are popular tool in requirement analysis

**2) Define Domain Model**

- Conceptual object model(개념적 객체 모델)

- shows the noteworthy domain concepts or objects

**3) Assign Object Responsibilities and Draw Iteration Diagram**

- A common notation to illustrate these collaborations is the sequence diagram.(UML)

- Note white board

- Software object designs and programs do take some inspiration form read-world domain, **but they are not direct models or simulations of the real world**

**4) Define Design Class Diagram**

- This illustrates the attributes and methods of the classes

- A lower representation gap between the software components and our mental models of a domain

**Chapter 2.**

**Iterative and evolutionary Development ( = iterative and incremental development** )

- early programming and testing of a partial system

- in repeating cycle.

1) short quick development step

2) feedback

3) adaptation to clarify the requirements and design

- mini project

- tested, integrated(통합시키다), executable

- requirement analysis , design, implementation, testing activities

- output is a production grade subset of the final system

- Better to resolve and prove the **risky and critical design decisions** **early** rather than late

- date slippage is illegal(기간 연장 X)

↔

**Sequential, Waterfall lifecycle**

1) big up front speculative requirements

2) design steps before programming

Software Development process

- building, deploying, possibly maintain software

**UP(Unified Process) → Widely Use RUP(Rational Unified Process)**

- Popular iterative software development

- Very flexible and open, lightweight and agile approach ( agile mena is 재빠른)

- Encourage including skillful practices from other iterative methods, (XP, TDD, refactoring, continuous integration (통합적인 개발))

Up is best practices

1. iterative lifecycle
2. risk driven development
3. cohesive(응집력)
4. well-documented process description

UP

- It’s necessary to introduce the subject in the context of some process

Iteration ( 2 ~ 6 weeks )

- subset of the requirements

- quickly designing

- implementing

- testing

Benefit Iterative

- Less project failure, better productivity and lower defect rates

- early visible progress

- early feedback

- managed complexity

How to do It And Ev Analysis and Desing

1. Two-days work shop ( 10% deeply analyze, 90% high level )   
   (1) architecturally significant implemented  
   (2) high business value  
   (3) high risk
2. 위의 사항에대해서 planning meeting four-week timeboxed iteration. design, build, test
3. Do Iteration 3~4 weeks  
   (1) two-day modeling, design work pair sketching whiteboard  
   (2) programming, testing  
   (3) Unit, acceptance, load, usability etc  
   (4) One week before the end, next To do list   
   (5) Last week Tuesday code finish. Iteration baseline create and total test  
   (6) Wednesday demo show stackholders and feedback
4. Second workshop
5. Iteration meeting
6. Similar step do
7. 80 ~ 90 % finish 반복
8. 20% elaboration phase( 얼마나 진행됬는지 판단)
9. tododododo