# Software design : an overview

Guy Tremblay and Anne Pons

200711468 장재호

# Contents

- 1. INTRODUCTION
- 2. SOFTWARE DESIGN CONCEPT
- 3. SOFTWARE STRUCTURE AND ARCHITECTURE
- 4. SOFTWARE DESIGN QUALITY ANALYSIS AND EVALUATION
- 5. SOFTWARE DESIGN NOTATIONS AND DOCUMENTATION
- 6. SOFTWARE DESIGN STRATEGIES AND METHOD
- 7. CONCLUSION

# 1. INTORODUCTION

- “Design”에 대한 IEEE의 정의

  - the process of defining the architecture, components, interfaces, and other characteristics of a system or component.

  - the result of [that] process.

# 1. INTORODUCTION

- Viewed as a process
  - Software design은 소프트웨어 개발의 life-cycle activity이다.

- More precisely
  - software design은 소프트웨어의 architecture와 interface를 말한다.

# 2. SOFTWARE DESIGN CONCEPT

## 2.1. General design Concepts

- 일반적인 정의

  - design은 문제를 해결하는 활동이다.
    하지만 이것의 역은 항상 사실일 필요는 없다.
    즉, 문제를 해결하는 활동이 모두 design은 아니다.

# 2. SOFTWARE DESIGN CONCEPT

## 2.1. General design Concepts

- Design problem은 일반적으로 여러가지 특성으로 특징지어진다. 예를 들면 한가지 문제에 여러 가지 가능한 solution이 존재한다.

- 따라서 wicked problem이 발생

# 2. SOFTWARE DESIGN CONCEPT

## 2.1. General design Concepts

- Design의 일반적인 정의에서의 5가지 키포인트

  - goals
  - constraints
  - alternatives
  - representations
  - solutions

# 2. SOFTWARE DESIGN CONCEPT

## 2.2. Software Design Context

- Software development life cycle

  - Software requirements analysis.

  - Software coding and testing
    (also known as software construction)

  - Software integration and qualification testing

# 2. SOFTWARE DESIGN CONCEPT

## 2.2. Software Design Context

- 2 main type of Life cycle model
  - Linear model
    ex) waterfall model

  - Incremental models
    ex) the spiral model, the iterative development approach

# 3. SOFTWARE STRUCTURE AND ARCHITECTURE

- Software architecture의 일반적인 정의
  - The internal structure

- Oxford 영어사전
  structure : the way in which something is constructed
  or organized

# 3. SOFTWARE STRUCTURE AND ARCHITECTURE

## 3.1. Architectural Structures and Views

- Kruchtens's "4+1 view model"

1. The local view
   - describes how the functional requirements are satisfied.

2. The implementation view
   - describes how the design is broken down into implementation units.

3. The process view
   - addresses issues related to concurrency and distribution.

4. The deployment view
   - shows how the runtime units and components are distributed onto the various processing nodes

5. The use-case view
   - which consists of a small number of use cases, ties together the other views, illustrating how the all work together

# 3. SOFTWARE STRUCTURE AND ARCHITECTURE

## 3.1. Architectural Structures and Views

- Other sets of view that classified into three categories, called viewtypes:
  1. Module viewtype
     - describe the units of implementation.
  2. Component-and-connector viewtype
     - the units of execution, that is, elements having a run-time presence.
  3. Allocation viewtype
     - The relationships between a system and its development and execution environment.

# 3. SOFTWARE STRUCTURE AND ARCHITECTURE

## 3.2.Macro/Microarchitectural Patterns

- The key idea behind patterns
  - 소프트웨어 개발자들이 되풀이되는 많은 문제와 해결을 관찰했고, pattern의 목적은 이런 일상적으로 되풀이 되는 전형적인 문제의 solution이다.

# 3. SOFTWARE STRUCTURE AND ARCHITECTURE

## 3.2.Macro/Microarchitectural Patterns

- 범위, 개념의 단계에 따라 분류되는 pattern의 세가지 항목
- 1. Architectural style
- 2. Design patterns
- 3. Coding idioms

# 3. SOFTWARE STRUCTURE AND ARCHITECTURE

## 3.2.Macro/Microarchitectural Patterns

- Architectural style
  - Macroachitectural Patterns
  - “a set of constraint on an architecture [that] define a set or family of architectures that satisfy them”

# 3. SOFTWARE STRUCTURE AND ARCHITECTURE

## 3.2.Macro/Microarchitectural Patterns

- Architectural style

- 다양한 저자들이 정의한 몇 가지 major architectural style
  - ▶ General structure(ex. layers, pipes and filters, blackboard)
  - ▶ Distributed system(ex. client-server, three-tiers, broker)
  - ▶ Interactive system(ex. model-view-controller)
  - ▶ Adaptable system(ex. Microkernel, reection)
  - ▶ Other style(ex. Batch, interpreters, process control)

# 3. SOFTWARE STRUCTURE AND ARCHITECTURE

## 3.2.Macro/Microarchitectural Patterns

- Design Patterns
  - Microachitectural Patterns
  - the high-level organization of software system

# 3. SOFTWARE STRUCTURE AND ARCHITECTURE

## 3.2.Macro/Microarchitectural Patterns

- Design Patterns
  - Gamma et al.
    - Creational patterns(ex. Builder, factory, prototype)
    - Structural patterns(ex. Adapter, bridge, composite)
    - behavioral patterns(ex. Command, interpreter, iterator)

# 3. SOFTWARE STRUCTURE AND ARCHITECTURE

## 3.2.Macro/Microarchitectural Patterns

- Design Patterns
  - Buschmann et al.
    - Structural decomposition patterns
    - Organization of work patterns
    - Access control patterns
    - Management patterns
    - Communication patterns

# 3. SOFTWARE STRUCTURE AND ARCHITECTURE

## 3.3 Design of Families of Systems and Frameworks

- Resent approaches toward that goal of software design  based on software product lines and software components.

- Software production line : A collection of systems sharing a managed set of features constructed from a common set of core software asset

# 4. SOFTWARE DESIGN QUALITY ANALYSIS AND EVALUATION

- Software Quality:

  ①명세서와의 일치 정도처럼 주어진 요구를 만족시키기 위한 능력에 영향을 미치는 소프트웨어 제품의 모든 특성과 속성들.
- ②소프트웨어가 요구되는 속성들의 조합을 갖고 있는지의 정도.
- ③해당 소프트웨어가 기대에 어느 정도 부응하는지의 정도.
- ④현재 사용 중인 소프트웨어가 고객의 기대에 어느 정도 부응하는지의 정도를 결정하는 소프트웨어의 제한 속성.
- Properties of Software Quality
  - Functionality
  - Usability
  - Efficiency
  - Maintainability
  - portability

# 4. SOFTWARE DESIGN QUALITY ANALYSIS AND EVALUATION

## 4.1 Design Quality Attributes

- Run-time qualities
  -ex) functionality, usability, performance

- Development-time qualities
  -ex) integrability, modifiablility, portability

# 4. SOFTWARE DESIGN QUALITY ANALYSIS AND EVALUATION

## 4.2 Measures (quantitative estimates)

선택된 디자인 접근 방식에 따라 분류됨

▶ Function-oriented (structured) measures
- structure chart로 표현
- ex) fan-in/fanout, cyclomatic complexity

▶ Objected-oriented measures
- class diagram으로 표현
- ex) weighted method per class, depth of inheritance tree, number of children

# 4. SOFTWARE DESIGN QUALITY ANALYSIS AND EVALUATION

## 4.3 Quality Analysis and Evaluation Tools

- 품질 특성을 계량하기 어려울 때 사용하는 다른 기법
  - Software design reviews
    ex) architecture reviews, design reviews and inspections
  - Simulation and prototyping
    ex) simulation-based performance or reliablility analysis

# 5 Software design notations and documentation

- Budgen categorization

▶ Black-box notation :
- design model의 외부 요소의 속성

▶ White-box notation :
- 디자인 요소의 구체적인 실현의 일부 측면을 나타냄

- Alternative categorization

▶ Structural/Static properties

▶ Behavioral/dynamic properties

# 5 Software design notations and documentation

## 5.1 A selection of Design Notations

- Class and object diagrams
- Component diagrams
- Deployment diagrams
- Structure charts
- Structure (Jackson) diagrams

# 5 Software design notations and documentation

## 5.2 Behavioral Descriptions (Dynamic view)

- 시스템, 구성요소의 동적 동작을 설명하는 데 사용
  - Activity diagrams
  - Interaction diagrams : sequence and collaboration diagrams
  - Data flow diagrams
  - State transition diagrams and statechart diagrams
  - Pseudo code and program design languages (PDLs)

# 6 Software Design strategies and methods

## 6.1 General Strategies and Enabling Techniques

- Abstraction
- Coupling and cohesion
- Divide and conquer
- Information hiding and encapsulation
- Sufficiency, completeness, primitiveness

# 6 Software Design strategies and methods

## 6.2 Function-oriented (Structured) Design

Divide and conquer approach toward identifying major system function in a top-down approach.

Structured analysis produces DFDs of the various system functions together with associated process descriptions, that is, descriptions of the processing performed by each subtask, usually using informal Pseudocode.

Entity-relationship diagrams describing the data stores can also be used.

- Key strategies to help derive a software architecture from a DFD
  - Transaction analysis: triggers
  - Transformation analysis : identifying the central transform, structure chart
- Key concept of structured design are those of coupling and cohesion: restrict coupling to normal types: data, stamp, control coupling. Avoid *common* and *contest* coupling

# 6 Software Design strategies and methods

## 6.3 Object-oriented Design

- Object based (no inheritance or polymorphism) Vs object oriented design
- OO design (solution domain) Vs requirement analysis ( problem domain)
- Class diagrams Vs Integration diagrams (sequence or elaboration diagrams)

# 6 Software Design strategies and methods

## 6.4 Data-structured-oriented Design

- Emphasis is on the data that a program manipulates rather than the functions it performs
- Motivated by stability in data rather then functions that need to be performed
- Restricted to the design of data-processing programs using sequential ( batch-style) files and processes
- Jackson System Development (JSD) : approach similar to OOD to address more complex interacting processes

# 6 Software Design strategies and methods

## 6.4 Data-structured-oriented Design

- Object based (no inheritance or polymorphism) Vs object oriented design
- OO design (solution domain) Vs requirement analysis ( problem domain)
- Class diagrams Vs Integration diagrams (sequence or elaboration diagrams)

# 7. CONCLUSION