Software Design: An Overview

Guy Tremblay and Anne Pons

Major: Computer System 06

200614159

Presenter: 박동민

Introduction

- Software Design?

  - SW development life cycle activity . Requirements are analyzed to produce a description of the SW's internal structure that will be the basis for it's construction

  - Also process of breaking down , organizing into components, and interfacing between components to a level of detail that enables their construction

  - Blueprint of the solution to be implemented: various models are analyzed, evaluated, and resulting model can be used as a starting point for subsequent development activity

2. Software Design Concepts

2.1 General Design Concepts:

- Design can be understood in 5 key concepts”

  - goals

  - constraints

  - alternatives

  - representations

  - solutions

2. Software Design Concepts

2.2 Software Design Context:

- SW development activities coupled with SW design are:
  - SW requirements analysis
  - SW coding and testing
  - SW integration and qualification testing

- Life cycle models: activities coupled with one another model are:
  - Linear models
  - Incremental models

2. Software Design Concepts

2.3 Software Design Process

- 2 activities that fits between SW requirement analysis and SW construction in standard listing of SW life cycle processes such as ISO/IEC 12207, Software Life Cycle Processes[ISO95] are :

1. Software architectural design:

- Sometimes called top-level design, describes how the system is broken down and organized into components the software architecture

2. Software detailed design:

- Describes specific behavior of the various components identified by the software architecture

3. Software Structure and Architecture

- Definition of software architecture:

  "The fundamental organization of a system embodies in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" , according to IEEE Standard1471

- Board meaning of software architecture may also be:

  - Architectural Design (ex. architectural styles)
  - Detailed Design (ex. Design patterns)
  - Families of Systems(ex. Product lines)

3. Software Structure and Architecture

3.1 Architectural Structures and Views

- Key purpose of SW architecture description:

  - communication among stakeholders(managers, testers, analysts..etc) involved in the development of the SW

- Views:

  a. different people, different roles and needs. Thus different perspectives of a software design should be described documented.

  b. A view represent a collection of models that represent one aspect of an entire system.

  c. IEEE Standard 1016, Recommended Practice for Software Design Description(SDD) is composed of a number of design views each of which contains a subset of the various attributes describing design entities

  - IEEE design views : Decomposition, Dependency, Interface , detailed description

3. Software Structure and Architecture

Continued from last slide

- RUP(Rational Unified Process)'s 4+1 view model includes:

a. logical view

b. Implementation

c. process view

d. deployment

e. use case

- 3 categories of views according to Clements et al, called view types:

a. Module viewtype

b. Component-and-connector viewtype

c. Allocation viewtype

3. Software Structure and Architecture

3.2 Macro/Microarchitectural Patterns: Architectural Styles Versus Design Patterns

- A pattern is a common solution to a common problem in a given context
- Key goal is to codify and document recurring solutions to typical problems
- System of patterns described by following attributes in Buschmann et al:
    a. name of the pattern
    b. The key situations in which the pattern may apply
    c. An example illustrating the need for the pattern
    d. The general problem, its essence that the pattern tries to solve
    e. The solution underlying the pattern
    f . Guideline for pattern's implementation

Patterns can be classified into 3 categories depending on their scope and level of abstraction:

a. Architectural Styles
b. Design patterns
c. Coding idioms

Architectural Style

- Architectural Styles – An architectural style expresses a fundamental structural organization schema for software systems. It provides a rich set of predefined subsystems, specifies their responsibilities, and includes rules and guideline for organizing the relationship between them
- Major Architectural Styles:
  - General structure(ex. Layers, pipes, and filters)
  - Distributed Systems(ex. Client-server, three-tiers, broker)
  - Interactive Systems(ex. Model-view-controller, presentation-abstraction-control)
  - Adaptable Systems(ex. Microkernel, reection)
  - Other styles(ex. Batch, interpreters, process control, rule-based)

- The choice of a particular Architectural style depends on the quality attributes that must be satisfied. One style may help certain quality attribute , it may also hinder other. Heterogeneous styles are also possible

Design Patterns(Microarchitectural Patterns)

- Design patterns are used to describe details at a lower and local level(Microarchitecture)

  - Gamma et al:
    a. Creational patterns
    b. Structural patterns
    c. Behavioral patterns
  - Buschmann et al:
    a. Structural decomposition patterns
    b. Organization of work patterns
    c. Access control patterns
    d. Management patterns
    e. Communication patterns

3. 3 Design of Families of Systems and Frameworks

An important goal of software design has always been to allow for the reuse of software elements and recent approaches toward that goal are based on software product lines and software components.

- Software Product line

  - a collection of systems sharing a common set of core software assets

  - Building a common set of software assets involves identifying the key commonalities encountered among the various members of the possible family of product- done through domain analysis as well as accounting for their possible variability - done by identifying and defining reusable and customizable components

  - In an object-oriented context, a related notion is that of framework, a partially complete software subsystem that can be extended by instantiating specific plug-ins or hot spots

4. Software Design Quality Analysis and Evaluation

- Software Quality: the totality of features and characteristics of a software product that bear on its ability to satisfy stated needs.
- 6 properties of quality : functionality, reliability, usability, efficiency, maintainability, and portability.

4.1 Design quality attributes:

- Run-time quality – observable only while the system is functioning(ex. Functionality, usability, performance…etc)
- Development-time quality – have impact on work of development, not observable at run time(ex. Modifiability, portability, reusability, testability)
- Conceptual Integrity – Architecture that reflects one single set of design ideas leading to simplicity, consistency, and elegance.

4. Software Design Quality Analysis and Evaluation

4.2 Measures:(Function-oriented, object oriented measures)

- Can be defined to obtain quantitative estimates of a design's size, structure, or quality. Used to measure certain quality attribute

- Function-oriented(structured) measures: the design's structure, obtained through functional decomposition is represented as a structure chart on which measures can be computed(ex. fan-in/out)

- Object-orient measures: the design structure is represented as class diagrams, can be computed(ex. Number of children, depth of inheritance tree)

4.3 Quality Analysis and Evaluation Tools

- Many quality attributes are hard to quantify so special techniques must be used to evaluate the quality of a design

- Software Design Reviews- (informal, semiformal, group-based) techniques used to verify the quality of a artifacts.

- Simulation and prototyping – dynamic techniques used to evaluate a design

5. Software Design notations and Documentation

- Many Different notations exist to represent SW design

- Budgen categorization of design notations:
  - Black box – concerned with the external properties of the elements of a design model
  - White box – largely concerned with describing some aspect of the detailed realization of a design element

- Alternative categorization to distinguish between:
  - Describing structural(static)properties – a design's structural organization
  - Describing behavioral(dynamic)properties – the behavior of the software components.

5. Software Design notations and Documentation

5.1 A Selection of Design Notations(UML)

- UML – Unified Modeling Language has become an almost de facto standard for software development notations

- Structural aspects of a SW design notations:

  - Class and object diagrams – used to represent a set of classes and objects and their relationship.

  - Component diagrams – used to model static implementation view of a system which are physical things such as executables, libraries, tables, files, and documents.

  - Deployment diagrams – used to model the static deployment view of a system. Can be used to represent distribution aspects(to model embedded, client/server or distributed systems)

  - Structural charts – used to describe the calling structure of programs

  - Structure(Jackson)diagrams – used to describe data structures manipulated by a program in terms of sequence , selection and iteration.

5. Software Design notations and Documentation

5.1 A Selection of Design Notations(UML)

- Behavioral Descriptions(Dynamic view)

  - Activity diagrams – used to show the control flow from activity to activity

  - Interaction diagrams(sequence and collaboration diagram) – used to show interactions among a group of objects

  - Data Flow diagrams – used to show the data flow among a set of processes

  - State transition diagrams and statechart diagrams – used to show the control flow from state to sate in a state machine

  - Pseudo code and program design language – languages used to describe generally at the detailed design stage, the behavior of a procedure or method

5. Software Design notations and Documentation

5.2 Design Documentation

- Coherency of design depends on the type of SW, the SW development method being used, stakeholders...etc
- Key practice is an selection of appropriate set of view to meet different needs of stakeholders
- Project manager would need detailed allocation views
- Developers would need mostly detailed module and component and connector view
- A key characteristic of any interface specification : must be two-way which means what the element provides and what it requires
- Another key idea: design should be presented and documented in a rational way: the rationale behind the key decisions should also recorded

ex. The design alternatives that were considered and rejected should be described

6. Software Design Strategies and Methods

6.1 General Strategies and Enabling Techniques

- can be described in terms of enabling technique that is independent of specific SW development method are :

a. Abstraction
b. Coupling and cohesion
c. Divide and conquer
d. Information hiding and encapsulation
e. Sufficiency, completeness and primitiveness

6.2 Function-oriented(Structured)Design

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

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

6. Software Design Strategies and Methods

- 2 key strategies to help derive a SW architecture from DFD:
  - a. Transaction analysis
  - b. Transformation analysis

- Couple and cohesion – key concepts of structured design, which characterize a design of good quality

- Additional Heuristics to improve the quality of resulting design:
  - fan in/out – high fan in is considered good/low
  - Decision splitting – recognition of a condition and execution of t he associated actions are not kept with the same module
  - Balanced systems – top level module deal with logical and abstract data

- Structured design made important contributions to SW design with its ability to integrate with an appropriate analysis method

6. Software Design Strategies and Methods

6.3 Object-oriented Design

- Notion of object – tied to the notions of data abstraction, encapsulation and abstract data type

- Object characteristics: created/destroyed, mutable, immutable

- Early OOD approaches – inheritance was not used(object based)

- Later OOD approaches – inheritance and polymorphism used(object oriented)

- UML – not an OO design method, neutral to any specific design

- UP(unified process) – incorporates OO analysis and design

- 4 phases of UP: inception, **elaboration**, **construction**, and transition

- Class diagram vs. interaction diagrams(sequence, collaboration diagrams)

6. Software Design Strategies and Methods

6.4 Data-Structure-Oriented Design

- Known as Jackson Structured Programming(JSP), emphasis is on the data that a program manipulates rather than the functions it performs

- Motivated by stability in data rather than the functions it perform(data is less like to be changed)

- Restricted to the design of data-processing programs using sequential (batch-style) files and processes

- Jackson System Development (JSD) : similar to OOD ,Jackson introduced to address more complex interacting processes

Conclusion

Software design is a rich and still evolving field

Software Design plays an important role in developing a software system.

Software design is both the process of defining the architecture, components, interfaces and other characteristics of a system or component and the result of that process

In Software Design, there are various models and methods can be used, but there is no such thing as “all around one” one and each one has different purpose for specific environment

advantage and disadvantage exist depending on different needs, purpose, and situation