CHAPTER 9 DESIGN ENGINEERING

Overview

A software design is a meaningful engineering representation of some software product that is to be built. Designers must strive to acquire a repertoire of alternative design information and learn to choose the elements that best match the analysis model. A design can be traced to the customer’s requirements and can be assessed for quality against predefined criteria. During the design process the software requirements model is transformed into design models that describe the details of the data structures, system architecture, interfaces, and components. Each design product is reviewed for quality (i.e., identify and correct errors, inconsistencies, or omissions) before moving to the next software engineering action.

Design Engineering

- Encompasses the set of principles, concepts, and practices that lead to the development of a high quality system or product
- Design principles establish and overriding philosophy that guides the designer as the work is performed
- Design concepts must be understood before the mechanics of design practice are applied
- Goal of design engineering is to produce a model or representation that is bug free (firmness), suitable for its intended uses (commodity), and pleasurable to use (delight)

Design Specification Models

- Data/Class design - created by transforming the analysis model class-based elements (class diagrams, analysis packages, CRC models, collaboration diagrams) into classes and data structures required to implement the software
- Architectural design - defines the relationships among the major structural elements of the software, it is derived from the class-based elements and flow-oriented elements (data flow diagrams, control flow diagrams, processing narratives) of the analysis model
- Interface design - describes how the software elements, hardware elements, and end-users communicate with one another, it is derived from the analysis model scenario-based elements (use-case text, use-case diagrams, activity diagrams, swim lane diagrams), flow-oriented elements, and behavioral elements (state diagrams, sequence diagrams)
- Component-level design - created by transforming the structural elements defined by the software architecture into a procedural description of the software components using information obtained from the analysis model class-based elements, flow-oriented elements, and behavioral elements
Software Quality Attributes

A good design must

- implement all explicit requirements from the analysis model and accommodate all implicit requirements desired by the user
- be readable and understandable guide for those who generate code, test components, or support the system
- provide a complete picture (data, function, behavior) if the software from an implementation perspective

Design Quality Guidelines

A design should

- exhibit good architectural structure
- be modular
- contain distinct representations of data, architecture, interfaces, and components (modules)
- lead to data structures that are appropriate for the objects to be implemented and be drawn from recognizable design patterns
- lead to components that exhibit independent functional characteristics
- lead to interfaces that reduce the complexity of connections between modules and with the external environment
- be derived using a reputable method that is driven by information obtained during software requirements analysis

FURPS Quality Factors

- Functionality
- Usability
- Reliability
- Performance
- Supportability

Generic Design Task Set

1. Select an architectural pattern appropriate to the software based on the analysis model
2. Partition the analysis model into design subsystems, design interfaces, and allocate analysis functions (classes) to each subsystem
3. Examine information domain model and design appropriate data structures for data objects and their attributes
4. Create a set of design classes
   - Translate analysis class into design class
   - Check each class against design criteria and consider inheritance issues
o Define methods and messages for each design class
o Select design patterns for each design class or subsystem after considering alternatives
o Revise design classes and revise as needed
5. Design user interface
   o Review task analyses
   o Specify action sequences based on user scenarios
   o Define interface objects and control mechanisms
   o Review interface design and revise as needed
6. Conduct component level design
   o Specify algorithms at low level of detail
   o Refine interface of each component
   o Define component level data structures
   o Review components and correct all errors uncovered
7. Develop deployment model

Design Concepts

- Abstraction - allows designers to focus on solving a problem without being concerned about irrelevant lower level details (procedural abstraction - named sequence of events, data abstraction - named collection of data objects)
- Software Architecture - overall structure of the software components and the ways in which that structure provides conceptual integrity for a system
- Design Patterns - description of a design structure that solves a particular design problem within a specific context and its impact when applied
- Modularity - the degree to which software can be understood by examining its components independently of one another
- Information Hiding - information (data and procedure) contained within a module is inaccessible to modules that have no need for such information
- Functional Independence - achieved by developing modules with single-minded purpose and an aversion to excessive interaction with other models
- Refinement - process of elaboration where the designer provides successively more detail for each design component
- Refactoring - process of changing a software system in such a way internal structure is improved without altering the external behavior or code design
- Objects - encapsulate both data and data manipulation procedures needed to describe the content and behavior of a real world entity
- Class - generalized description (template or pattern) that describes a collection of similar objects
- Inheritance - provides a means for allowing subclasses to reuse existing superclass data and procedures; also provides mechanism for propagating changes
- Messages - the means by which objects exchange information with one another
- Polymorphism - a mechanism that allows several objects in an class hierarchy to have different methods with the same name (instances of each subclass will be free to respond to messages by calling their own version of the method)
Design Classes - refine analysis classes by providing detail needed to implement the classes and implement a software infrastructure to support the business solution (i.e., user interface classes, business domain classes, process classes, persistent classes, system classes)

Modular Design Method Evaluation Criteria

- Modular decomposability - provides systematic means for breaking problem into subproblems
- Modular composability - supports reuse of existing modules in new systems
- Modular understandability - module can be understood as a stand-alone unit
- Modular continuity - side-effects due to module changes minimized
- Modular protection - side-effects due to processing errors minimized

Effective Modular Design

- Functional independence - modules have high cohesion and low coupling
- Cohesion - qualitative indication of the degree to which a module focuses on just one thing
- Coupling - qualitative indication of the degree to which a module is connected to other modules and to the outside world

Design Class Characteristics

- Complete and sufficient
- Primitiveness - each class method focuses on providing one service
- High cohesion
- Low coupling

Design Model

- Process dimension - indicates design model evolution as design tasks are executed during software process
  - Architecture elements
  - Interface elements
  - Component-level elements
  - Deployment-level elements
- Abstraction dimension - represents level of detail as each analysis model element is transformed into a design equivalent and refined
  - High level (analysis model elements)
  - Low level (design model elements)
- Many UML diagrams used in the design model are refinements of diagrams created in the analysis model

Data Design
• High level model depicting user's view of the data or information
• Design of data structures and operators is essential to creation of high-quality applications
• Translation of data model into database is critical to achieving system business objectives
• Reorganizing databases into data warehouse enables data mining or knowledge discovery that can impact success of business itself

Architectural Design

• Derived from
  ◦ Information about the application domain relevant to software
  ◦ Relationships and collaborations among specific analysis model elements
  ◦ Availability of architectural patterns and styles
• Usually depicted as a set of interconnected systems that are often derived from the analysis packages

Interface Design

• Interface is a set of operations that describes the externally observable behavior of a class and provides access to its operations
• Important elements
  ◦ User interface (UI)
  ◦ External interfaces to other systems
  ◦ Internal interfaces between various design components
• Modeled using UML collaboration diagrams

Component-Level Design

• Describes the internal detail of each software component
• Defines
  ◦ Data structures for all local data objects
  ◦ Algorithmic detail for all component processing functions
  ◦ Interface that allows access to all component operations
• Modeled using UML component diagrams, UML activity diagrams, and pseudocode (PDL)

Deployment-Level Design

• Indicates how software functionality and subsystems will be allocated within the physical computing environment
• Modeled using UML deployment diagrams

Design Pattern Template

• Name
• Intent
• Aliases
• Motivation
• Applicability
• Structure
• Participants
• Collaborators
• Consequences
• Related patterns

Using Patterns in Design

• Architectural patterns - define overall structure and relationships of software components (classes, packages, subsystems)
• Design patterns - address specific design elements (component aggregations, component relationships, or communication)
• Idioms (coding patterns) - language specific implementations for algorithms or communications mechanisms

Frameworks

• Collection of "plug points" hooks or slots that enable it to be adapted to a specific problem domain
• Plug points enable designers to integrate problem specific functionality within the skeleton
• In an object-oriented context a skeleton is a collection of cooperating classes