Design Engineering

Overview

- What is software design?
- How to do it?
- Principles, concepts, and practices
- High-level design
- Low-level design

Design Engineering

- The process of making decisions about HOW to implement software solutions to meet requirements
- Encompasses the set of concepts, principles, and practices that lead to the development of high-quality systems

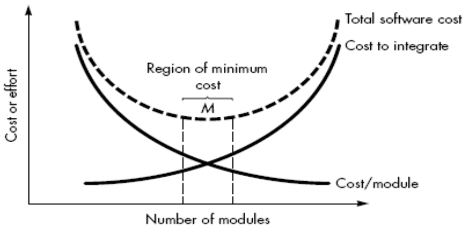
Concepts in Software Design

- Modularity
- Cohesion & Coupling
- Information Hiding
- Abstraction & Refinement
- Refactoring

Modularity

- Software is divided into separately named and addressable components, sometimes called modules, that are integrated to satisfy problem requirements
- Divide-and-conquer

Modularity and Software Cost



Cohesion & Coupling

- Cohesion
  - The degree to which the elements of a module belong together
  - A cohesive module performs a single task requiring little interaction with other modules
- Coupling
  - The degree of interdependence between modules
- High cohesion and low coupling

Information Hiding

- Do not expose internal information of a module unless necessary
  - E.g., private fields, getter & setter methods

Abstraction & Refinement

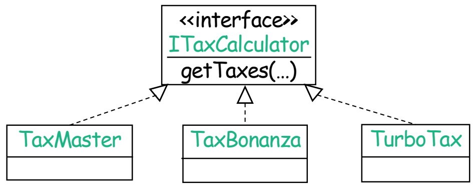
- Abstraction
  - To manage the complexity of software,
  - To anticipate detail variations and future changes
- Refinement
  - A top-down design strategy to reveal low-level details from high-level abstraction as design progresses

Abstraction to Reduce Complexity

- We abstract complexity at different levels
  - At the highest level, a solution is stated in broad terms, such as "process sale"
  - At any lower level, a more detailed description of the solution is provided, such as the internal algorithm of the function and data structure

Abstraction to Anticipate Changes

- Define interfaces to leave implementation details undecided
- Polymorphism



Software Design Practices Include:

- Two stages
  - High-level: Architecture design
    - Define major components and their relationship
  - Low-level: Detailed design
    - Decide classes, interfaces, and implementation algorithms for each component

How to Do Software Design?

- Reuse or modify existing design models
  - High-level: Architectural styles
  - Low-level: Design patterns, Refactorings
- Iterative and evolutionary design
  - Package diagram
  - Detailed class diagram
  - Detailed sequence diagram

Software Architecture

- "The architecture of a system is comprehensive framework that describes its form and structure -- its components and how they fit together"
  --Jerrold Grochow

What is Architectural Design?

- Design overall shape & structure of system
  - the components
  - their externally visible properties
  - their relationships
- Goal: choose architecture to reduce risks in SW construction & meet requirements

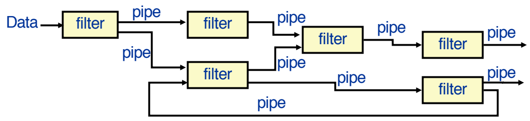
SW Architectural Styles

- Architecture composed of
  - Set of components
  - Set of connectors between them
    - Communication, co-ordination, co-operation
  - Constraints
    - How can components be integrated?
  - Semantic models
    - What are the overall properties based on understanding of individual component properties?

Architecture Patterns

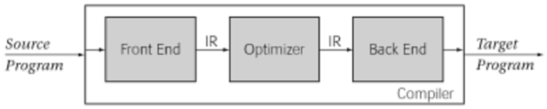
- Common program structures
  - Pipe & Filter Architecture
  - Event-based Architecture
  - Layered Architecture

Pipe & Filter Architecture

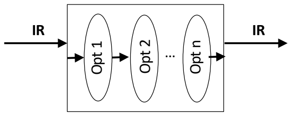


- A pipeline contains a chain of data processing elements
  - The output of each element is the input of the next element
  - Usually some amount of buffering is provided between consecutive elements

Example: Optimizing Compiler



Compiler Structure

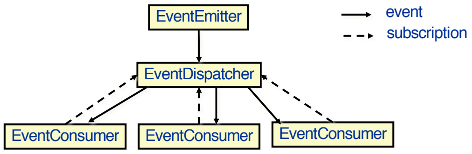


Compiler Optimization
[Engineering a Compiler, K. D. Cooper, L. Torczon]

Pros and Cons

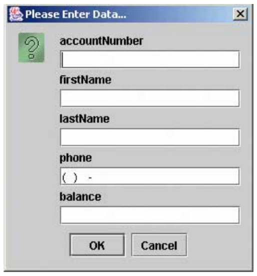
- Other examples
  - UNIX pipes, signal processors
- Pros
  - Easy to add or remove filters
  - Filter pipelines perform multiple operations concurrently
- Cons
  - Hard to handle errors
  - May need encoding/decoding of input/output

Event-based Architecture



- Promotes the production, detection, consumption of, and reaction to events
- More like event-driven programming

Example: GUI


Please Enter Data...

accountNumber

firstName

lastName

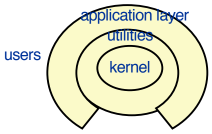
phone
( ) -

balance

Pros and Cons

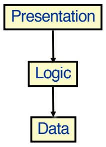
- Other examples:
  - Breakpoint debuggers, phone apps, robotics
- Pros
  - Anonymous handlers of events
  - Support reuse and evolution, new consumers easy to add
- Cons
  - Components have no control over order of execution

Layered/Tiered Architecture



- Multiple layers are defined to allocate responsibilities of a software product
- The communication between layers is hierarchical
- Examples: OS, network protocols

3-layer Architecture



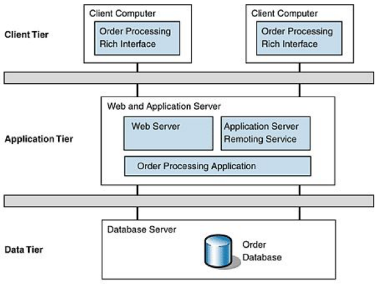
Presentation

Logic

Data

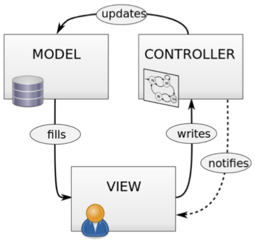
- Presentation: UI to interact with users
- Logic: coordinate applications and perform calculations
- Data: store and retrieve information as needed

Example: Online Ordering System



http://www.cardisoft.gr/frontend/article.php?aid=87&cid=96

Model-View-Controller



Design of Finite State Machine Drawing Tool

https://commons.wikimedia.org/wiki/File:MVC_Diagram_(Model-View-Controller).svg

Key Points about MVC

- View layer should not handle system events
- Controller layer has the application logic to handle events
- Model layer only respond to data operation

Layered Architecture: Pros and Cons

- Pros
  - Support increasing levels of abstraction during design
  - Support reuse and enhancement
- Cons
  - The performance may degrade
  - Hard to maintain