Software Design – What is it?

- Software design is an iterative process through which requirements are translated into a “blueprint” for constructing the software.
- During the design process the software requirements model is transformed into design models that describe the details of the data structures, system architecture, interface, and components.
- Each design product is reviewed for quality before moving to the next phase of software development.
Design Models

- Data design - created by transforming the analysis information model into data structures
- Architectural design - defines the relationships among the major structural elements of the software
- Interface design - describes how the software elements communicate with each other, with other systems, and with human users
- Component-level design - created by transforming the structural elements defined by the software architecture into procedural descriptions of software components
Design Principles

- should not suffer from tunnel vision
- should be traceable to the analysis model
- should not reinvent the wheel
- should minimize intellectual distance between the software and the problem as it exists in the real world
- should exhibit uniformity and integration
- should be structured to accommodate change
- should be assessed for quality as it is being created
- should be reviewed to minimize conceptual (semantic) errors

Design Concepts

- Abstraction - allows designers to focus on solving a problem without being concerned about irrelevant lower level details
  - Data abstraction - named collection of data objects
  - Procedural abstraction - named sequence of events
  - Control abstraction – program control mechanisms
- Refinement - process of elaboration where the designer provides successively more detail for each design component
  - Abstraction and refinement are complementary concepts

Data Abstraction

A door implemented as a data structure:

- manufacturer
- model number
- type
- swing direction
- lights
- type
- number
- weight
- opening mechanism

Implemented as a data structure
Procedural Abstraction

Implemented with a "knowledge" of the object that is associated with enter:

Stepwise Refinement

Repeat until door opens
- turn knob clockwise;
- if knob doesn't turn, then take key out;
- find correct key;
- insert in lock;
- pull/push door;
- move out of way;
- end repeat.

Modularity – software is divided into separately named and addressable components, called modules, that are integrated to satisfy problem requirements

- easier to build
- easier to change
- easier to fix
Design Concepts (Contd)

- Modularity

![Diagram showing the relationship between number of modules, cost per module, total software cost, region of minimum cost, and cost to integrate.]

Design Concepts (Contd)

- Software architecture - overall structure of the software components and the ways in which that structure provides conceptual integrity for a system
- Data structure - representation of the logical relationship among individual data elements
  - Examples: scalar, array, linked list
- Software procedure - processing details of each module individually
  - Event sequences, decision points, repetitive operations, data organization/structure
- Control hierarchy - represents the module organization and implies hierarchy of control, but does not represent the procedural aspects of the software (e.g. event sequences)

Design Concepts (Contd)

- Information hiding - information (data and procedure) contained within a module is inaccessible to modules that have no need for such information
- Patterns – design structures that provide proven solutions to recurring problems
- Refactoring – process of changing a software system in such a way that it does not alter the external behavior of the design (or code), yet improves its internal structure
Modular Design

• Reduces complexity - provides systematic means for breaking problem into sub-problems
• Facilitates change – critical aspect of software maintainability
• Results in easier implementation by encouraging parallel development of different parts of a system
• Supports reuse of existing modules in new systems

Modular Design (Contd)

• Functional independence – each module addresses a specific sub-function of requirements and has a simple interface when viewed from the other parts of the program structure
• Independence is measured using two qualitative criteria
  • 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
• Good design has high cohesion and low coupling

Cohesion

Seven categories or levels of cohesion (non-linear scale)
1. Coincidental cohesion Bad
2. Logical cohesion
3. Temporal cohesion
4. Procedural cohesion
5. Communicational cohesion
6. Functional cohesion
7. Informational cohesion Good
1. Coincidental Cohesion

- A module has coincidental cohesion if it performs multiple, completely unrelated actions
  - print next line, reverse string of characters comprising second parameter, add 7 to fifth parameter, convert fourth parameter to floating point
- Arise from rules like “Every module will consist of between 35 and 50 statements”
- Why is coincidental cohesion so bad?
  - Degrades maintainability
  - Modules are not reusable
- How to fix
  - Break into separate modules each performing one task

2. Logical Cohesion

- A module has logical cohesion when it performs a series of related actions, one of which is selected by the calling module
- Example
  - Module performing all input and output
- Why is logical cohesion so bad?
  - The interface is difficult to understand
  - Code for more than one action may be intertwined
  - Difficult to reuse

3. Temporal Cohesion

- A module has temporal cohesion when it performs a series of actions related in time
- Example
  - open old master file, new master file, transaction file, print file, initialize sales district table, read first transaction record, read first old master record (a.k.a. perform initialization)
- Why is temporal cohesion so bad?
  - Actions of this module are weakly related to one another, but strongly related to actions in other modules.
  - Not reusable
4. Procedural Cohesion

- A module has procedural cohesion if it performs a series of actions related by the procedure to be followed by the product.

- Example
  - read part number and update repair record on master file

- Why is procedural cohesion bad?
  - Actions are still weakly connected
  - Module is unlikely to be reusable

5. Communicational Cohesion

- A module has communicational cohesion if it performs a series of actions related by the procedure to be followed by the product, but in addition all the actions operate on the same data.

- Example 1
  - update record in database and write it to audit trail

- Example 2
  - calculate new coordinates and send them to terminal

- Why is communicational cohesion bad?
  - Still lack of reusability

6. Functional Cohesion

- Module with functional cohesion performs exactly one action

- Examples
  - calculate sales commission

- Why is functional cohesion good?
  - More reusable
  - Corrective maintenance easier
  - Easier to extend product
7. Informational Cohesion

- A module has informational cohesion if it performs a number of actions, each with its own entry point, with independent code for each action, all performed on the same data structure.

- Major difference between logical and information cohesion:
  - Actions in the module with logical cohesion are intertwined.
  - Actions in the module with information cohesion are completely independent.

- Why is informational cohesion good?
  - All the advantages of using abstract data type are gained.

---

Example of Information Cohesion

![Diagram showing a flowchart with multiple entries and exits, each with a description of actions and data regions.]

---

Coupling

Five categories or levels of coupling (non-linear scale):

1. Content coupling: Bad
2. Common coupling
3. Control coupling
4. Stamp coupling
5. Data coupling: Good
1. Content Coupling

- Two modules are content coupled if one directly references contents of the other
- Example
  - Module a branches into local label of module b
- Why is content coupling bad?
  - Almost any change to b, even recompiling b with new compiler or assembler, requires change to a

2. Common Coupling

- Two modules are common coupled if they have write access to global data
- Example
  - Modules a and b can access and change the value of global variable
- Why is common coupling bad?
  - Contradicts the spirit of structured programming
  - Modules can have side-effects
  - Entire module must be read to find out what it does
  - Difficult to reuse
  - Module exposed to more data than necessary

3. Control Coupling

- Two modules are control coupled if one passes an element of control to the other, that is, one module explicitly controls the logic of the other
- Example - If module p calls module q and q passes back
  - "I am unable to complete my task", then q is passing data
  - "I am unable to complete my task; write error message ABC123", then p and q are control coupled
- Why is control coupling bad?
  - Modules are not independent; module q (the called module) must know internal structure and logic of module p. This affects reusability
  - Usually is associated with modules of logical cohesion
4. Stamp Coupling

- Two modules are stamp coupled if a data structure is passed as a parameter, but the called module operates on some but not all of the individual components of the data structure.
- Why is stamp coupling bad?
  - It is not clear, without reading the entire module, which fields of a record are accessed or changed.
  - Difficult to understand.
  - Unlikely to be reusable.
  - More data than necessary is passed.

5. Data Coupling

- Two modules are data coupled if all parameters are homogeneous data items, that is, every argument is either a simple argument, or data structure in which all elements are used by the called module.
- Why is data coupling good?
  - The difficulties of content, common, control, and stamp coupling are not present.
  - Maintenance is easier.