

Objectives

- To introduce the process of software design
- To describe the different stages in this design process
- To show how object-oriented and functional design strategies are complementary
- To discuss some design quality attributes

CS 422 Software Engineering Principles
From Software Engineering by F. Schneider, 1996

Chapter 12
Slide 2

Topics covered

- The design process and design methods
- Design strategies including object-oriented design and functional decomposition
- Design quality attributes

CS 422 Software Engineering Principles
From Software Engineering by F. Schneider, 1996

Chapter 12
Slide 3

Stages of design

Problem understanding

Look at the problem from different angles to discover the design requirements

Identify one or more solutions

Evaluate possible solutions and choose the most appropriate depending on the designer's experience and available resources

Describe solution abstractions

Use graphical, formal or other descriptive notations to describe the components of the design

Repeat process for each identified abstraction until the design is expressed in primitive terms

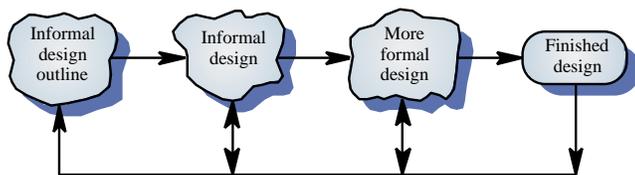
The design process

Any design may be modeled as a directed graph made up of entities with attributes which participate in relationships

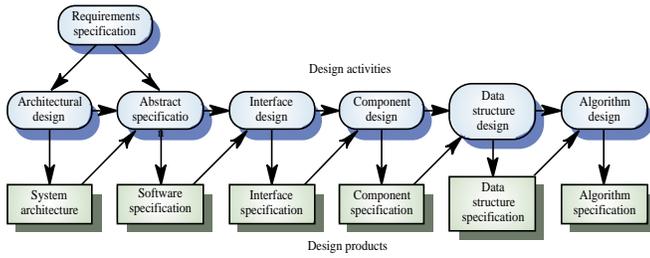
The system should be described at several different levels of abstraction

Design takes place in overlapping stages. It is artificial to separate it into distinct phases but some separation is usually necessary

From informal to formal design



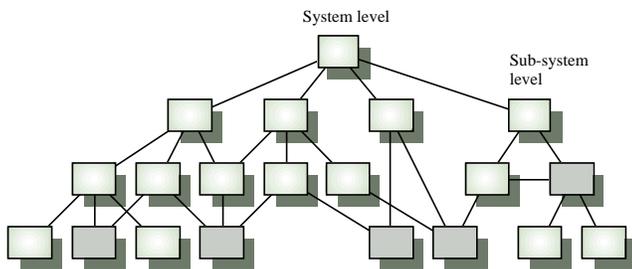
Phases in the design process



Design phases

- Architectural design* Identify sub-systems
- Abstract specification* Specify sub-systems
- Interface design* Describe sub-system interfaces
- Component design* Decompose sub-systems into components
- Data structure design* Design data structures to hold problem data
- Algorithm design* Design algorithms for problem functions

Hierarchical design structure



Top-down design

In principle, *top-down design involves starting at the uppermost components in the hierarchy* and working down the hierarchy level by level

In practice, *large systems design is never truly top-down*

Some branches are designed before others.

Designers **reuse** experience (and sometimes components) during the design process

Design methods

Structured methods are sets of notations for expressing a software design and guidelines for creating a design

Well-known methods include Structured Design (Yourdon), and JSD (Jackson Method)

Can be applied successfully because they support standard notations and ensure designs follow a standard form

Structured methods may be supported with CASE tools

Method components

Many methods support comparable views of a system

A data flow view (data flow diagrams) showing data transformations

An entity-relation view describing the logical data structures

A structural view showing system components and their interactions

Method deficiencies

They are guidelines rather than methods in the mathematical sense. Different designers create quite different system designs

They do not help much with the early, creative phase of design. Rather, they help the designer to structure and document his or her design ideas

Design description

Graphical notations. Used to display component relationships

Program description languages. Based on programming languages but with more flexibility to represent abstract concepts

Informal text. Natural language description.

All of these notations may be used in large systems design

Design strategies

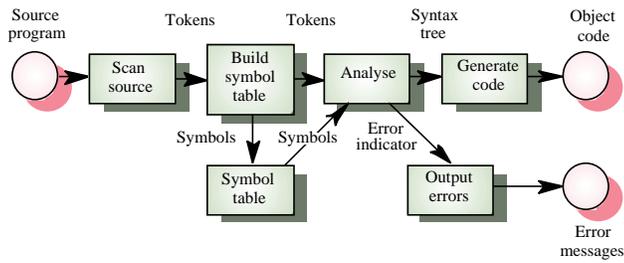
Functional design

The system is designed from a functional viewpoint. The system state is centralized and shared between the functions operating on that state

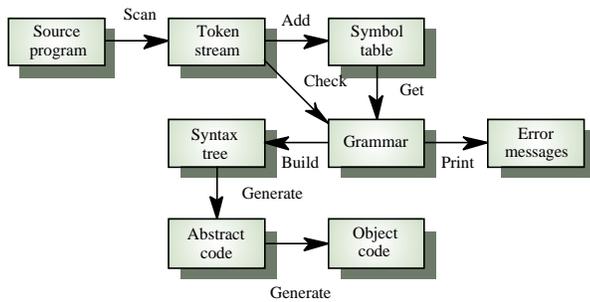
Object-oriented design

The system is viewed as a collection of interacting objects. The system state is de-centralized and each object manages its own state. Objects may be instances of an object class and communicate by exchanging methods

Functional view of a compiler



Object-oriented view of a compiler

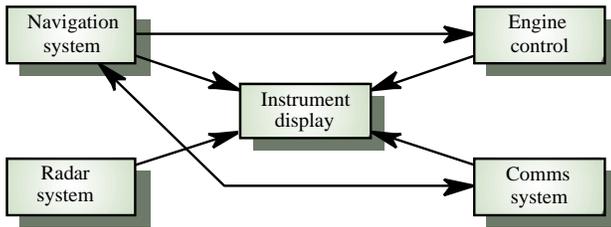


Mixed-strategy design

Although it is sometimes suggested that one approach to design is superior, in practice, an object-oriented and a functional-oriented approach to design are complementary

Good software engineers should select the most appropriate approach for whatever sub-system is being designed

Aircraft sub-systems



High-level objects

- The navigation system
- The radar system
- The communications system
- The instrument display system
- The engine control system

...

System functions (sub-system level)

- Display track (radar sub-system)
- Compensate for wind speed (navigation sub-system)
- Reduce power (engine sub-system)
- Indicate emergency (instrument sub-system)
- Lock onto frequency (communications sub-system)

...

Low-level objects

The engine status
The aircraft position
The altimeter
The radio beacon
...

Design quality

Design quality is an elusive concept. Quality depends on specific organizational priorities
A 'good' design may be the most efficient, the cheapest, the most maintainable, the most reliable, etc.
The attributes discussed here are concerned with the maintainability of the design
Quality characteristics are equally applicable to function-oriented and object-oriented designs

Cohesion

A measure of how well a component 'fits together'
A component should implement a single logical entity or function
Cohesion is a desirable design component attribute as when a change has to be made, it is localized in a single cohesive component
Various levels of cohesion have been identified

Cohesion levels

Coincidental cohesion (weak)

Parts of a component are simply bundled together

Logical association (weak)

Components which perform similar functions are grouped

Temporal cohesion (weak)

Components which are activated at the same time are grouped

Procedural cohesion (weak)

The elements in a component make up a single control sequence

Cohesion levels

Communicational cohesion (medium)

All the elements of a component operate on the same input or produce the same output

Sequential cohesion (medium)

The output for one part of a component is the input to another part

Functional cohesion (strong)

Each part of a component is necessary for the execution of a single function

Object cohesion (strong)

Each operation provides functionality which allows object attributes to be modified or inspected

Cohesion as a design attribute

Not well-defined. Often difficult to classify cohesion

Inheriting attributes from super-classes weakens cohesion

To understand a component, the super-classes as well as the component class must be examined

Object class browsers assist with this process

Coupling

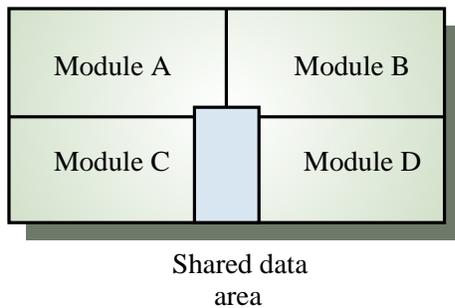
A measure of the strength of the inter-connections between system components

Loose coupling means component changes are unlikely to affect other components

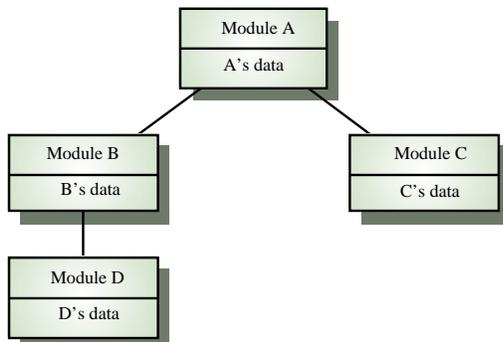
Shared variables or control information exchange lead to tight coupling

Loose coupling can be achieved by state decentralization (as in objects) and component communication via parameters or message passing

Tight coupling



Loose coupling



Coupling and inheritance

Object-oriented systems are loosely coupled because there is no shared state and objects communicate using message passing

However, an object class is coupled to its super-classes.

Changes made to the attributes or operations in a super-class propagate to all sub-classes.

.....Such changes must be carefully controlled!

Understandability

Related to several component characteristics

Cohesion. Can the component be understood on its own?

Naming. Are meaningful names used?

Documentation. Is the design well-documented?

Complexity. Are complex algorithms used?

Informally, high complexity means many relationships between different parts of the design

Hence it is hard to understand

Most design quality metrics are oriented towards complexity measurement and are of limited use

Adaptability

A design is adaptable if:

Its components are loosely coupled

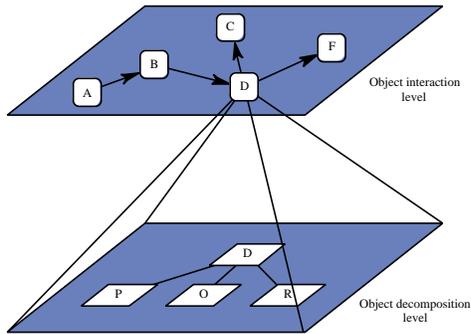
It is well-documented and the documentation is up to date

There is an obvious correspondence between design levels (design visibility)

Each component is a self-contained entity (tightly cohesive)

To adapt a design, it must be possible to trace the links between design components so that change consequences can be analyzed

Design traceability



Adaptability and inheritance

Inheritance dramatically improves adaptability.
Components may be adapted without change by deriving a sub-class and modifying that derived class
However, as the depth of the inheritance hierarchy increases, it becomes increasingly complex
It must be periodically reviewed and restructured

Key points

Design is a creative process
Design activities include architectural design, system specification, component design, data structure design and algorithm design
Functional decomposition considers the system as a set of functional units
Object-oriented decomposition considers the system as a set of objects
Decisions on parallelism should usually be detailed design decisions
