Unit-3 Design

3.6- Function-Oriented Design

3.7- Object Oriented Design

3.8- Component Base Design

APPROACHES TO SOFTWARE DESIGN

There are two fundamentally different approaches to software design that are in use today—

1. Function-oriented design
2. Object-oriented design

- Though these two design approaches are radically different, they are complementary rather than competing techniques.
- The object oriented approach is a relatively newer technology and is still evolving.
- For development of large programs, the object- oriented approach is becoming increasingly popular due to certain advantages that it offers.
- On the other hand, function-oriented designing is a mature technology and has a large following.

Function Oriented Design

- A system is viewed as something that performs a set of functions. Starting at this high-level view of the system, each function is successively refined into more detailed functions.
- For example, consider a function create-newlibrary- member which essentially creates the record for a new member, assigns a unique membership number to him, and prints a bill towards his membership charge. This function may consist of the following subfunctions:
- assign-membership-number
- create-member-record
- print-bill

Each of these sub-functions may be split into more detailed subfunctions and so on.

- The system state is centralized and shared among different functions, e.g. data such as member-records is available for reference and updation to several functions such as:
- create-new-member
- delete-member
- update-member-record

Object-oriented design

- In the object-oriented design approach, the system is viewed as collection of objects (i.e. entities). The state is decentralized among the objects and each object manages its own state information.
- For example, in a Library Automation Software, each library member may be a separate object with its own data and functions to operate on these data. In fact, the functions defined for one object cannot refer or change data of other objects.
- Objects have their own internal data which define their state. Similar objects constitute a class. In other words, each object is a member of some class. Classes may inherit features from super class. Conceptually, objects communicate by message passing.

Function-oriented vs. object-oriented design approach

- Unlike function-oriented design methods, in OOD, the basic abstraction are not real-world functions such as sort, display, track, etc, but real world entities such as employee, picture, machine, radar system, etc. For example in OOD, an employee pay-roll software is not developed by designing functions such as update-employee-record, getemployee- address, etc. but by designing objects such as employees, departments, etc.

- Function-oriented techniques such as SA/SD group functions together if, as a group, they constitute a higher-level function. On the other hand, object-oriented techniques group functions together on the basis of the data they operate on.

Example: Fire-Alarm System

- The owner of a large multi-stored building wants to have a computerized fire alarm system for his building.
- Smoke detectors and fire alarms would be placed in each room of the building.
- The fire alarm system would monitor the status of these smoke detectors.
- Whenever a fire condition is reported by any of the smoke detectors, the fire alarm system should determine the location at which the fire condition is reported by any of the smoke detectors, the fire alarm system should determine the location at which the fire condition has occurred and then sound the alarms only in the neighboring locations.
- The fire alarm system should also flash an alarm message on the computer console.
- Fire fighting personnel man the console round the clock.
- After a fire condition has been successfully handled, the fire alarm system should support resetting the alarms by the fire fighting personnel.

- **Function-Oriented Approach:**
- /* **Global data (system state) accessible by various functions */**

```
BOOL detector_status[MAX_ROOMS];
int detector_locs[MAX_ROOMS];
BOOL alarm_status[MAX_ROOMS];  /* alarm activated when status is set */
int alarm_locs[MAX_ROOMS];   /* room number where alarm is located */
int neighbor-alarm[MAX_ROOMS][10];  /* each detector has at most 10
   neighboring locations*/
```

The functions which operate on the system state are:

```
interrogate_detectors();
get_detector_location();
determine_neighbor();
ring_alarm();
reset_alarm();
report_fire_location();
```

Object-Oriented Approach:

class detector

attributes**:**

status, location, neighbors

operations**:**

create, sense_status, get_location, find_neighbors

class alarm

attributes**:**

location, status

operations**:**

create, ring_alarm, get_location, reset_alarm

Component Based Design

SOFTWARE REUSE

- A possible way to reduce development cost is to reuse parts from previously developed software.
- In addition to reduced development cost and time, reuse also leads to higher quality of the developed products since the reusable components are ensured to have high quality.

WHAT CAN BE REUSED?

- Almost all artifacts associated with software development, including project plan and test plan can be reused. However, the prominent items that can be effectively reused are:

1. Requirements specification
2. Design
3. Code
4. Test cases
5. Knowledge

BASIC ISSUES IN ANY REUSE PROGRAM

- The following are some of the basic issues that must be clearly understood for starting any reuse program:

1. Component creation.
2. Component indexing and storing.
3. Component search.
4. Component understanding.
5. Component adaptation.
6. Repository maintenance.

1. Component creation: For component creation, the reusable components have to be first identified.

- Selection of the right kind of components having potential for reuse is important.

2. Component indexing and storing. Indexing requires classification of the reusable components so that they can be easily searched when we look for a component for reuse.

3. Component searching: The programmers need to search for right components matching their requirements in a database of components.

- To be able to search components efficiently, the programmers require a proper method to describe the components that they are looking for.

4. Component understanding: The programmers need a precise and sufficiently complete understanding of what the component does to be able to decide whether they can reuse the component.

- To facilitate understanding, the components should be well documented and should do something simple.

5. Component adaptation: Often, the components may need adaptation before they can be reused, since a selected component may not exactly fit the problem at hand.

- However, tinkering with the code is also not a satisfactory solution because this is very likely to be a source of bugs.

6. Repository maintenance

- A component repository once is created requires continuous maintenance.
- New components, as and when created have to be entered into the repository.
- The faulty components have to be tracked.
- Further, when new applications emerge, the older applications become obsolete. In this case, the obsolete components might have to be removed from the repository.

REUSE APPROACH

- A promising approach that is being adopted by many organizations is to introduce a building block approach into the software development process.
- For this, the reusable components need to be identified after every development project is completed.
- The reusability of the identified components has to be enhanced and these have to be cataloged into a component library.
- It must be clearly understood that an issue crucial to every reuse effort is the identification of reusable components.
- Domain analysis is a promising approach to identify reusable components. In the following subsections, we discuss the domain analysis approach to create reusable components.

Domain Analysis

- The aim of domain analysis is to identify the reusable components for a problem domain.
- **Reuse domain:** A reuse domain is a technically related set of application areas.
- A body of information is considered to be a problem domain for reuse, if a deep and comprehensive relationship exists among the information items as characterized by patterns of similarity among the development components of the software product.
- A reuse domain is a shared understanding of some community, characterized by concepts, techniques, and terminologies that show some coherence.
- Just to become familiar with the vocabulary of a domain requires months of interaction with the experts.

- Often, one needs to be familiar with a network of related domains for successfully carrying out domain analysis.
- Domain analysis identifies the objects, operations, and the relationships among them.
- During domain analysis, a specific community of software developers get together to discuss community-wide solutions.
- Analysis of the application domain is required to identify the reusable components.
- The actual construction of the reusable components for a domain is called *domain engineering*.

- **Evolution of a reuse domain:** The ultimate results of domain analysis is development of problem oriented languages.
- The problem-oriented languages are also known as *application generators*.
- These application generators, once developed form application development standards.
- The domains slowly develop.
- As a domain develops, we may distinguish the various stages it undergoes:

Stage 1 : There is no clear and consistent set of notations. Obviously, no reusable components are available. All software is written from scratch.

Stage 2 : Here , only experience from similar projects are used in a development effort. This means that there is only knowledge reuse.

Stage 3: At this stage, the domain is ripe for reuse. The set of concepts are stabilized. and the notations standardized. Standard solutions to standard. problems are available. There is both knowledge and component reuse.

Stage 4: The domain has been fully explored. The software development for the domain can largely be automated.

Programs are not written in the traditional sense any more. Programs are written using a domain specific language, which is also known as an *application generator*.

Component Classification

Components need to be properly classified in order to develop an effective indexing and storage scheme.

- We have already remarked that hardware reuse has been very successful. If we look at the classification of hardware components for clue, then we can observe that hardware components are classified using a multilevel hierarchy.
- At the lowest level, the components are described in several forms—natural language description, logic schema, timing information, etc.
- The higher the level at which a component is described, the more is the ambiguity.
- This has motivated the Prieto-Diaz's classification scheme.

Prieto-Diaz's classification scheme:

- Each component is best described using a number of different characteristics or facets.
- For example, objects can be classified using the following:

1. Actions they embody.
2. Objects they manipulate.
3. Data structures used.
4. Systems they are part of, etc.

Searching

- A popular search technique that has proved to be very effective is one that provides a web interface to the repository.
- Using such a web interface, one would search an item using an approximate automated search using key words, and then from these results would do a browsing using the links provided to look up related items.
- The approximate automated search locates products that appear to fulfill some of the specified requirements.
- The items located through the approximate search serve as a starting point for browsing the repository.
- These serve as the starting point for browsing the repository.
- The developer may follow links to other products until a sufficiently good match is found. Browsing is done using the keywordto-keyword, keyword-to-product, and product- to-product links.
- Finding a satisfactory item from the repository may require several iterations of approximate search followed by browsing.

Repository Maintenance

- Repository maintenance involves entering new items, retiring those items which are no more necessary, and modifying the search attributes of items to improve the effectiveness of search.
- Also, the links relating the different items may need to be modified to improve the effectiveness of search.
- As patterns requirements emerge, new reusable components are identified, which may ultimately become more or less the standards. However, as technology advances, some components which are still reusable, do not fully address the current requirements.
- On the other hand, restricting reuse to highly mature components, can sacrifice potential reuse opportunity

Reuse without Modifications

- Once standard solutions emerge, no modifications to the program parts may be necessary.
- One can directly plug in the parts to develop his application.
- Application generators have significant advantages over simple parameterized programs.
- The biggest of these is that the application generators can express the variant information in an appropriate language rather than being restricted to function parameters, named constants, or tables.
- The other advantages include fewer errors, easier to maintain, substantially reduced development effort, and the fact that one need not bother about the implementation details.
- Application generators have been applied successfully to data processing application, user interface, and compiler development.
- Application generators are less successful with the development of applications with close interaction with hardware such as real-time systems.