SOFTWARE DESIGN

- Software design is an iterative process through which requirements are translated into a “blue print” for constructing the software.
- There are **3** distinct types of *activities* in software design:

i) External Design:

- Involves conceiving, planning and specifying the externally observable characteristics of a software product.
- The characteristics include:
  i) User displays and report formats
  ii) External data sources and data sinks
  iii) Functional characteristics
  iv) Performance requirements
  v) High level process structure of the product.

ii) Internal Design:

- Involves conceiving planning and planning out specifying the internal structure and the processing details of the software product.
- The details include
  i) Specify internal structure and processing data

ii) To record design decisions
iii) To elaborate test plan
iv) provides a blue print for implementation, testing and maintenance activities.

iii) Detailed Design:

i) Specification of algorithm that implements the function
ii) Concrete data structures that implement the data stores
ii) Actual interconnections among function and data structures.

SOFTWARE DESIGN PROCESS

- Software design is an iterative process through which requirements are translated into a “blue-print” for constructing software.
- The design is represented at a high-level of abstraction.
- High-level of abstraction is a level that can be directly traced to the specific system objective and more detail data functional and behavioral requirements.

Design and Software quality:

Three characteristics that serve as a guide the evaluation of a good design are:

1. The design must implement all the explicit requirements contained in the analysis model and it must accommodate all of the implicit requirements desired by the customer.
2. The design must be a readable understandable guide for those who generate code and those who test and subsequently support the software.
3. The design should provide a complete picture of the software, addressing the data, functional and behavioral domains from an implementation perspective.

Characteristics of Good Design:

i) A design should exhibit an architecture that has been created using recognizable design patterns. That composed of components that exhibit good design characteristics. This can be implemented in an

evolutionary fashion, there by facilitating implementation and testing.

ii) A design should be modular (i.e.,) the software should be logically partitioned into elements that perform specific functions and sub functions.

iii) A design should contain distinct representations of data architecture interfaces and components.

iv) A design should lead to data structures

v) A design should lead to components that exhibit independence functional characteristics.

vi) A design should lead to interfaces that reduce the complexity of connections between modules and with the external environment.

vii) A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.

viii) A design should be represented using a notation that effectively communicates its meaning.

Quality Attributes:

The software quality attribute are

* Functionality
* Reliability
* Performance
* Supportability [FURPS].

The FURPS quality attributes represent a target for all software design:

1. **Functionality:**

   It is accessed by evaluating the feature set and capabilities of the program, the generality of the functions that are delivered the security of the overall system.

2. **Usability:**

   It is accessed by considering human factors, aesthetics, consistency and documentation.

3. **Reliability**:

   It is evaluated by measuring the frequency and severity of failure, the accuracy of output result, the mean-time-to-failure [MTTF], the ability to recover from failure and the predictability of the program.

4. **Performance:**

   It is measured by processing speed, response time, resource consumption, throughput and efficiency.

5. **Supportability:**

   It combines the ability to extend the program extensibility, adaptability, serviceability.

Design Concepts

Software design concepts provide the necessary framework for "getting it right".

1. Abstraction:

- Abstraction is the intellectual tool that allows us to deal with concepts apart from particular instances of those concepts.
- At the highest level of the abstraction a solution is stated in broad terms using the language of the problem environment.
- At the lowest level of the abstraction a more detailed description of the solution is provided.

Different types of abstraction are

2. Procedural abstraction or Functional abstraction

- refers to a sequence of instructions that have a specific and limited function.
- E.g:"the word 'open' for a door"
- Open includes a long sequence of procedural steps. (e.g.: walk to the door, reach out and grasp knob. Turn knob & pull door etc).

3. Data abstraction:

- collection of data that describes a data object.
- eg: we can define a data abstraction called door. (eg: door type, swing direction, opening mechanism, weight dimensions etc).

4. Architecture.

The architectural design can be given using one or more of a no of different models

i) Structural models

Represent architecture as a organized collection of program component.

ii) Framework models

Increase the level of design abstraction by attempting to identify the repeatable architectural design framework, which is encountered in similar types of applications.

iii) Dynamic models

Address the behavioral aspects of the program architecture, indicating how the system or structure configuration may change as a function of external events.

iv) Process models

Focus on the design of the business or technical process that the system must accommodate.

v) Functional models

Can be used to represent the functional hierarchy of the system.

5. Patterns:

A pattern is defined as in which conveys the essence of a proven solution to a recurring problem within a certain context in competing concerns".

The goal of the each design pattern is to provide a description that enables a designer to determine.

- Whether the pattern is applicable to the current work.

- Whether the pattern can be reused.
- Whether the pattern can serve as a guide for developing a similar but functionality or structurally different pattern.

6. Modularity:

- Software is divided into separately named and addressable components called modules.
- These are integrated to satisfy problem requirements.
- Modularity is a single attribute of software that allows a program to be intellectually manageable.

PROPERTIES

i) Each processing abstraction is a well-defined subsystem that is potentially useful in other applications.

ii) Each function in each abstraction has a single well defined purpose

iii) Each function manipulates no more than one major data structure.

iv) Functions share global data selectively. It is easy to identify all routines that share a major data structure.

v) Functions that manipulate instances or abstract data types are encapsulated with the data structure being manipulated.

7. Information hiding:

- Each module in the system hides the internal details of its processing activities.

8. Functional independence

- It is a direct outgrowth of modularity and the concepts of abstraction & information hiding.
- It is achieved by developing modules with "single-minded", and an "aversion" to excessive interaction with other modules.

Independence is assessed using two methods

a. Cohesion:

- It is an indication of the relative functional strength of a module.
- It is a natural extension of information hiding.
- A cohesive module performs a single task, requiring little interaction with other components .

b. Coupling:

- Coupling is an indication of interconnection among modules in a software structure.
- Coupling depends on the interface complexity between modules.
- The degree of coupling is lowest for data communication, higher for control communication and highest for the modules that modify other modules.

9. Refinement:

- It is a process of elaboration.
- Step-wise refinement is a top-down design strategy.
- A program is developed is by successively refining levels of procedural detail.
- Abstraction enables a designer to reveal low-level details as design progresses.

10. Refactoring:

- It is a reorganization techniques that simplifies the design (or code) of a component without changing its behavior.
- Defined as Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure.