CONTENTS

CHARACTERISTICS OF A GOOD DESIGN

FIRMNESS

- A program should not have any bugs that inhabit this function.

COMMODITY

- A program should be suitable for the purpose for which it was intended.

DELIGHT

- The experience of using the program should be pleasurable and easy to use.

DESIGN PROCESS

- Software designing is an art.
- It is a creative process.
- Software design is an iterative process through which requirements are translated into a blueprint for constructing the software.
- It is the most difficult and challenging task.
- Throughout the design process the quality of evolving design is assessed with series of technical reviews

STAGES OF DESIGN

1. Understand the problem.
2. Identify one or more solutions.
3. Solution Abstraction.

PROCESS OF DESIGN ENGINEERING

1. Software specifications are transformed into design models.
2. Models describe the details of data structures system architecture, interface, and components.
3. Design productive stand reviewed for quality before moving to the next phase of software development.
4. A design model and specification documents are produced containing models, architecture interfaces, and components.

DESIGN ELEMENTS

- Data/class design
  - Creates a model of data and objects that is represented at a high level of abstraction
- Architectural design
  - Depicts the overall layout of the software
- Interface design
  - Tells how information flows into and out of the system and how it is communicated among the components defined as part of the architecture
  - Includes the user interface, external interfaces, and internal interfaces
- Component-level design elements
  - Describes the internal detail of each software component by way of data structure definitions, algorithms, and interface specifications
- Deployment-level design elements
  - Indicates how software functionality and subsystems will be allocated within the physical computing environment that will support the software

DESIGN CONCEPT

ABSTRACTION

- Data abstraction is the programming process of creating a data type, usually a class, that hides the details of the data representation to make the data type easier to work with.
- Extracting relevant details only.

ARCHITECTURE

- Overall structure of a software.
  1. Structural Properties: The manner in which the components (Modules, Objects etc.) are packed and interact with one another.
  2. Extra-Functional properties: How the design achieves requirements for performance, reliability, security, and other system characteristics.
  3. Families of related system: Ability to reuse architectural building blocks.

PATTERN

- A structure that solves design problem.
- A pattern must be guaranteed to work so that it may be reused many times over, but it also must be relevant to the current project at the same time.
- If the said pattern does not fit into the overall design function of the current project, it might be possible to reuse it as a guide to help create a new pattern that would be more fitting to the situation.

1. Architectural Pattern: High-level pattern type that can be defined as the overall formation and organization of the software system itself.
2. Design Pattern: Medium-level pattern type that is used by the developers to solve problems in the design stage of development. Can affect how objects or components interact with one another.
3. Coding Pattern: Low-level pattern type, often known as coding patterns, they are used as a workaround means of setting up and defining how components will be interacting with the software itself without being dependent on the programming language. There are many different programming languages all with different syntax rules, making this a requirement to function on a variety of platforms.

MODULARITY

- Modularity refers to the idea of using predetermined code to increase overall efficiency and management of the current project.
- The software components will usually be divided into unique items known as modules.
- Their specific functions divide these modules.
- Modularity makes the systems easy to manage.

INFORMATION HIDING

- Data hiding allows modules to pass only required information between themselves without sharing the internal structures and processing.
- The specific purpose of hiding the internal details of individual objects has several benefits: Increased efficiency, in general, allows for much lower-level error and high-quality software.

FUNCTIONAL INDEPENDENCE

- Modules that have a “Single-Minded” function and an aversion to excessive interaction with other modules.
- High Cohesion: A module performs only a single task
- Low Coupling: A module has lowest amount of connection needed with other modules.

COHESION

- Cohesion is the measure of functional strength of a module.
- In general, it measures the relationship strength between 2 pieces of functionality within a given module.

1. High Cohesion: In a good module, various parts having high cohesion is preferable due to its reliability, usability, robustness, and understandability.

2. Low Cohesion: It is associated with undesirable traits including difficulty in maintaining, reusing, and understanding.

COUPLING

- How closely two modules interact with each other and how interdependent they are.
- The degree of coupling between two modules depends on their interface complexity.
- If the system has low coupling it is a sign of well-structured computer system and a great design.
- A system with low coupling and high cohesion supports the mission of higher readability and maintainability.
- The coupling is an indication of strength of interconnection between all the components in system.

STEPWISE REFINEMENT

- A process of elaboration.
- It is a top-down design strategy.
- Development of a program by successively refining levels of procedure detail.
- For example:
    1. Open the Door.
    2. Walk to door; Reach the Knob; Open Door; Walk through; Close door.

REFACTORING

- A reorganization technique that simplifies the design of a component without changing its functional or external behavior.
- When software is refactored, the existing design is examined for redundancy, unused design elements, insufficient or unnecessary algorithm, poorly constructed or inappropriate data structure, or any other design failures.

OBJECT ORIENTED DESIGN CONCEPTS

DESIGN CLASS

- Entity classes.
- Boundary classes.
- Controller classes.

INHERITANCE

- All responsibility of superclass immediately inherited by all subclasses.

POLYMORPHISM

- A characteristic that greatly reduces the effort required to extend the design.

DESIGN CLASSES

ENTITY CLASSES

- Analysis classes as refined during design to become entity classes.

BOUNDARY CLASSES

- These are developed during design to create the interface that the user sees and interacts with as the software is used.

CONTROLLER CLASSES

- These classes are designed to manage the creation or update of entity objects.
- Complex communication between set of objects.