Introduction to Abstraction

- Definition of Abstraction: Abstraction is a fundamental concept in software design where essential features are emphasized while irrelevant details are ignored.

- Importance: Abstraction simplifies problem-solving by focusing on the core aspects of a problem, aiding in better understanding and facilitating efficient design processes.

Procedural Abstraction

- Definition: Procedural abstraction involves breaking down complex problems into smaller, manageable subproblems and sequences of actions.

- Example: Consider a sorting module where the focus is on the output being sorted, without concerning about the specifics of the sorting algorithm initially.

- Top-Down Decomposition: This approach results in a hierarchical structure with the main problem at the top, followed by its decomposition into subproblems and primitive problems.

- Importance: Procedural abstraction aligns with human problem-solving tendencies and simplifies software development processes.

Data Abstraction

- Definition: Data abstraction focuses on organizing data structures in a hierarchical manner to simplify data handling and enhance adaptability.

- Primitive Data Structures: Programming languages offer basic data structures like integers, real numbers, and characters.

- Building Complicated Structures: These include more complex structures like stacks, binary trees, and application-oriented objects.

- Importance: Data abstraction simplifies data handling, making it easier to adapt and comprehend software systems.

Comparison: Procedural vs. Data Abstraction

- **Procedural Abstraction**: Focuses on controlling the sequence of actions and decomposing problems into smaller steps.

- **Data Abstraction:** Focuses on organizing and structuring data to simplify data handling and enhance adaptability.

- Example: Procedural abstraction deals with the sequence of actions in a program, while data abstraction abstracts data structures from implementation details.

Benefits of Abstraction

- Simplifies Problem-Solving: Abstraction breaks down complex problems into manageable parts, making them easier to tackle.

- Enhances Adaptability: Abstraction facilitates changes and updates in software design, as modifications can be made at higher levels without impacting lower-level details.

- Improves Comprehension: Abstraction makes code easier to understand and maintain, leading to better software quality and longevity.

SYSTEM STRUCTURE

System structure refers to the arrangement and organization of components within a software system, including modules, their relationships, and dependencies.

Definition of Modules and Dependencies:

- Modules are distinct units of software that encapsulate related functionality.

- Dependencies between modules define how they interact and rely on each other during execution.

Intermodule Relations:

- Various types of intermodule relations exist, including containment, sequencing, data delivery, and usage.

- These relations determine the complexity of module interactions and influence system design.

Call Graph Representation

Explanation of Call Graph:

- A call graph represents the outcome of a design process, with nodes representing modules and edges denoting their relations.

- It serves as a visual representation of module dependencies within a software system.

Types of Relations:

- Call graphs depict various intermodule relations, such as containment, sequencing, data delivery, and usage.

- Each relation illustrates how modules interact and communicate with one another.

Minimizing Knowledge Between Modules:

- Call graphs help identify and manage dependencies while keeping module interactions concise.

Call Graph Shapes

Different Shapes of Call Graphs:

- Call graphs can take different shapes, including directed graphs, hierarchies, layered schemes, and tree structures.

- Each shape reflects the organization and complexity of module dependencies within the system.

Measurable Attributes of Call Graphs

Attributes of Call Graphs:

- Measurable attributes of call graphs include size (number of nodes and edges), depth (longest path from root to leaf node), and tree impurity.

- These attributes provide insights into the complexity and structure of module dependencies.

Importance in Assessing Design Quality:

- They serve as parameters for qualitative assessment of design effectiveness.

Tree Impurity

Explanation of Tree Impurity:

- Tree impurity measures the deviation of a call graph from a pure tree structure.

- It indicates the extent to which the graph contains cycles or non-tree-like structures.