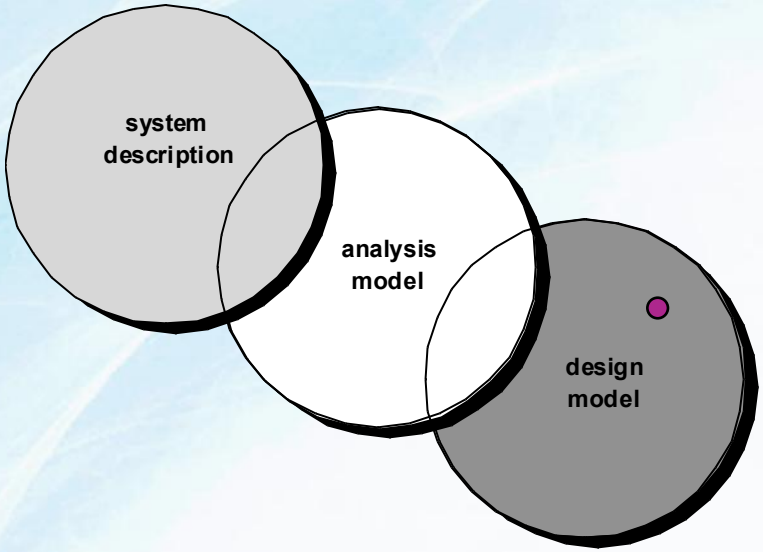
Software Design

By

Purvi D. Sankhe

A Bridge

Writing the Software Specification



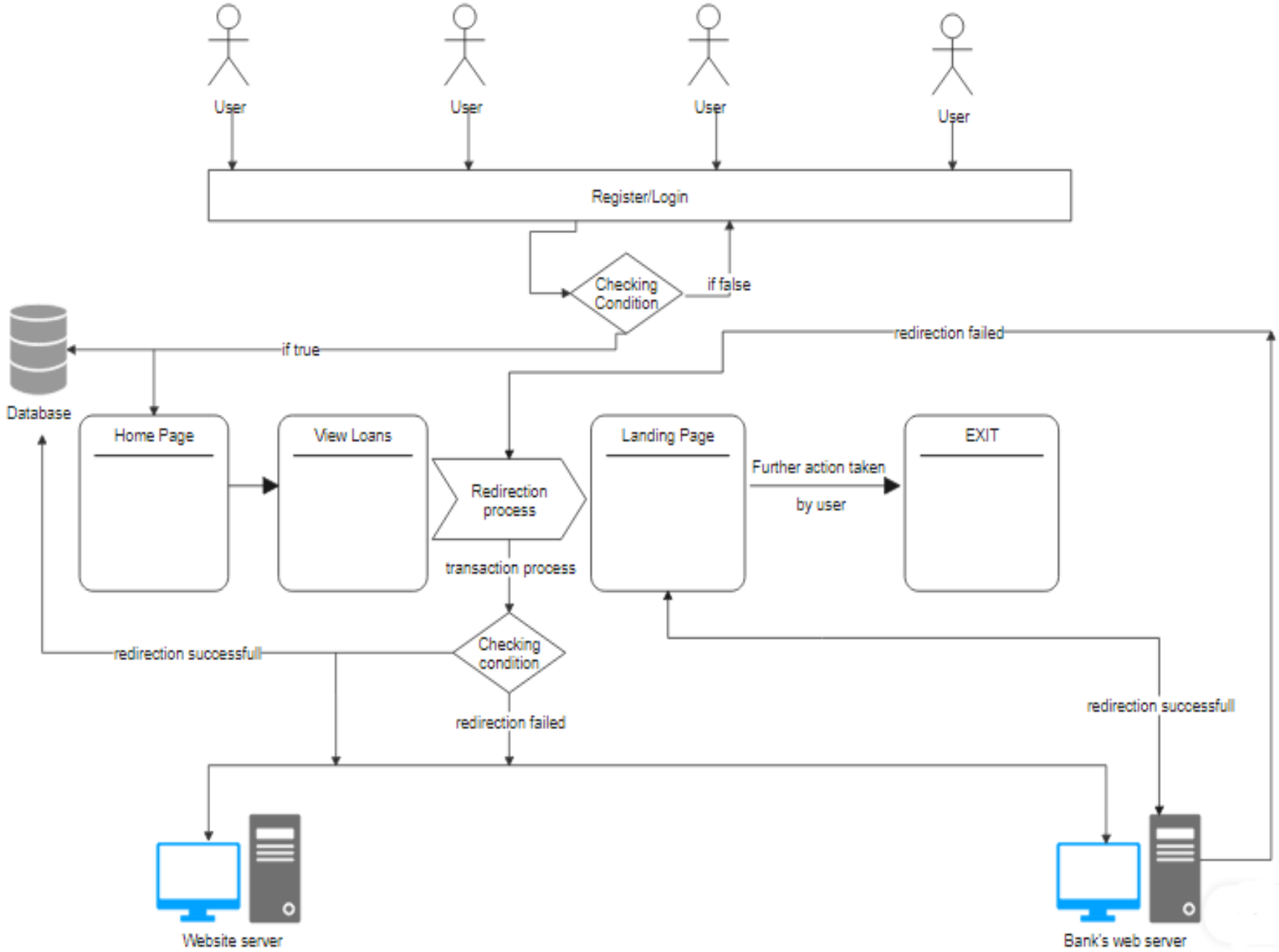
Design Modeling in Software Engineering

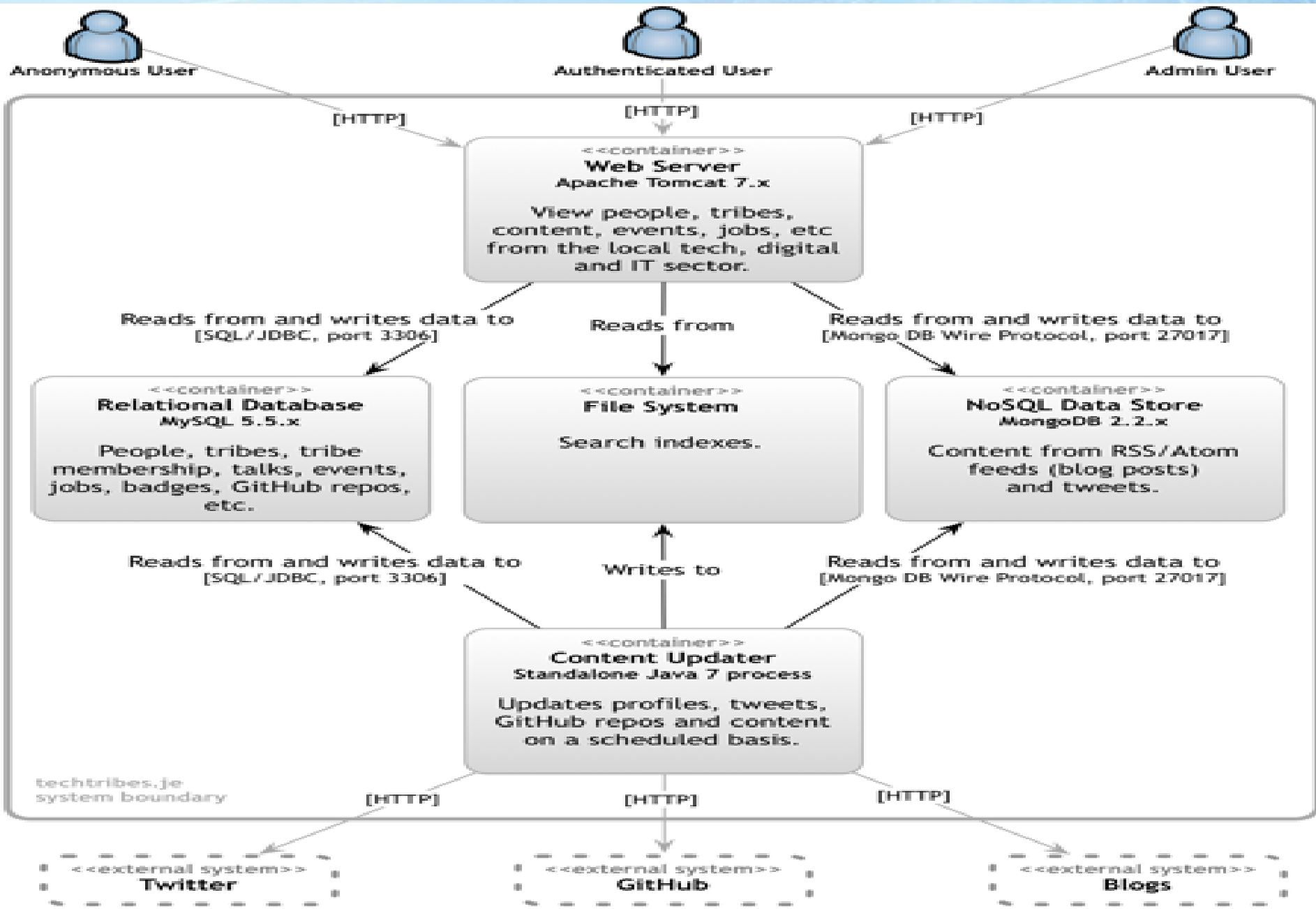
Data Design

- **The data design action translates data defined as part of the analysis model into data structures.**
- **The data objects, attributes, and relationships depicted in entity relationship diagrams and the information stored in data dictionary provide a base for data design activity.**

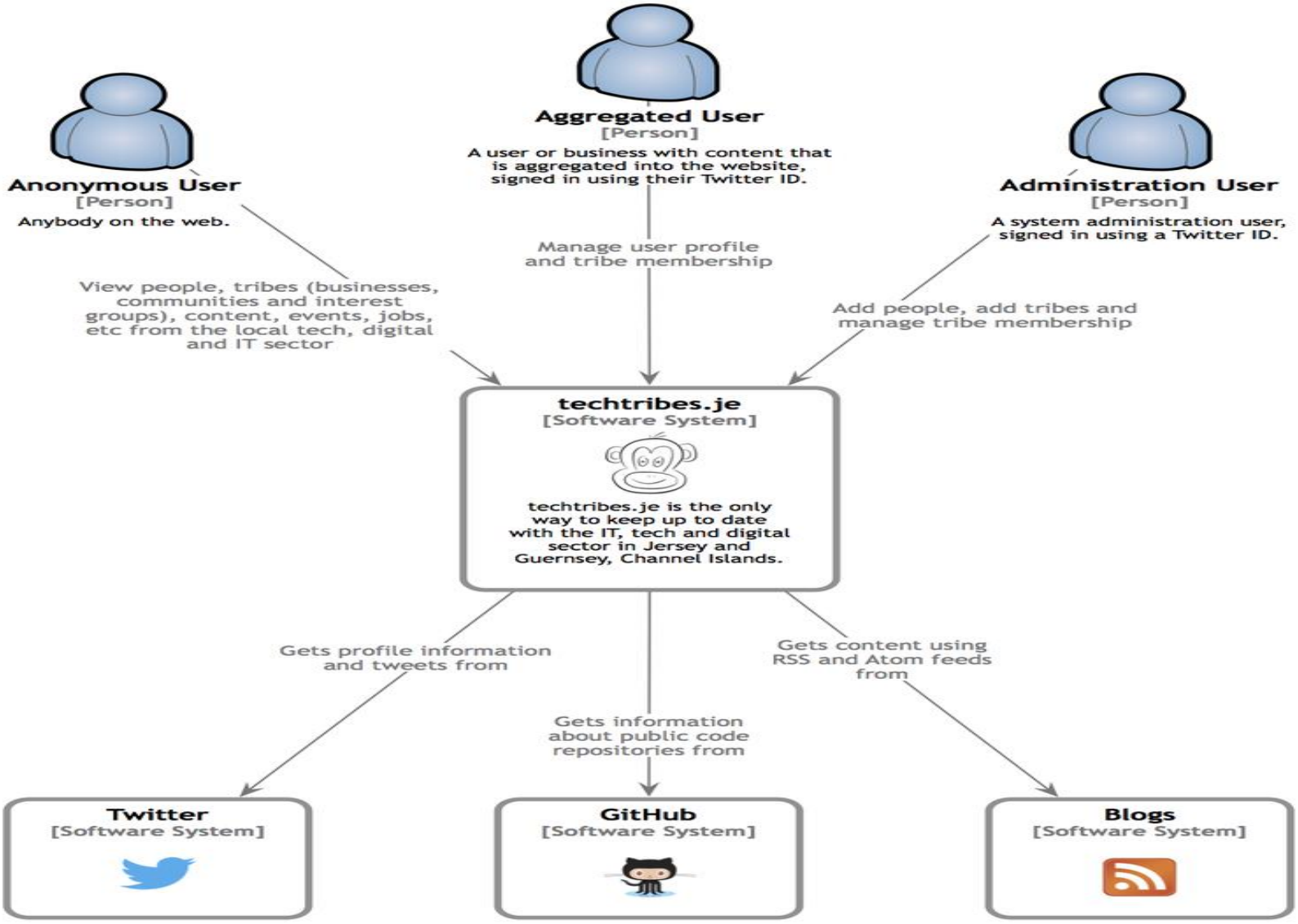
Architectural design

- **Requirements of the software should be transformed into an architecture that describes the software's top-level structure and identifies its components. This is accomplished through architectural design (also called system design),**
- **Acts as a preliminary 'blueprint' from which software can be developed.**
- **IEEE defines architectural design as 'the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.'**


User
User
User
User
Register/Login
Checking Condition
if false
if true
Database
redirection failed
Home Page
View Loans
Redirection process
Landing Page
Further action taken by user
EXIT
transaction process
Checking condition
redirection successfull
redirection failed
redirection successfull
Website server
Bank's web server



techtribes.je - Containers



techtribes.je - Context

UI Design

- **User interface is the front-end application view to which user interacts in order to use the software.**
- **User can manipulate and control the software as well as hardware by means of user interface.**
- **User interface is part of software and is designed such a way that it is expected to provide the user insight of the software.**
- **UI provides fundamental platform for human-computer interaction.**

Component Level Design

- **Detailed design phase that focuses on defining the internal structure, behavior, and interactions of individual software components.**
- **A component is a modular, reusable, and replaceable part of a system that encapsulates a specific functionality or set of functionalities.**
- **Component-level design bridges the gap between high-level architectural design and actual code implementation.**
- **This design phase ensures that each component is well-defined, adheres to architectural guidelines, and can be integrated seamlessly with other components to form the complete system.**

How the Four Elements Work Together

- **Data Design** provides the foundation for how data is stored and managed.
- **Architectural Design** defines the high-level structure and organization of the system.
- **Interface Design** ensures smooth interaction between users, components, and external systems.
- **Component-Level Design** provides the detailed implementation plan for each component.

Example: Design Model for an Online Shopping System

1. Data Design

- **Entities:** User, Product, Order, Payment.
- **Relationships:** A User can place multiple Orders, and an Order contains multiple Products.
- **Database:** Relational database with tables for Users, Products, Orders, and Payments.

2. Architectural Design

- **Presentation Layer:** User interface for browsing products and checking out.
- **Business Logic Layer:** Handles shopping cart, payment processing, and order management.
- **Data Layer:** Manages data storage and retrieval.

Example: Design Model for an Online Shopping System

3. Interface Design

- **UI:** Screens for product search, cart management, and payment.
- **APIs:** RESTful APIs for adding products to the cart, processing payments, and fetching order history.

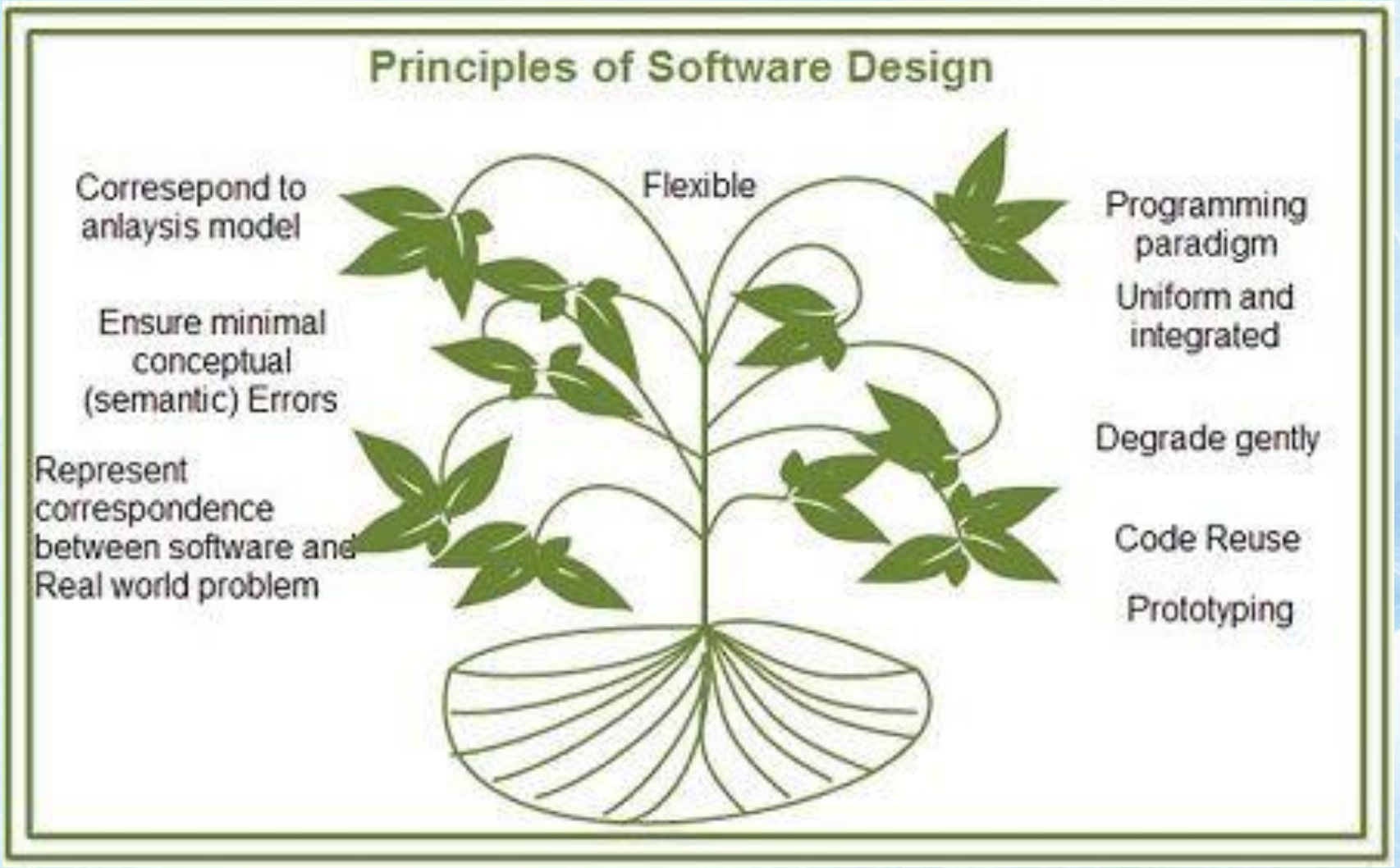
4. Component-Level Design

- **Authentication Component:** Login(username, password), Register(userDetails).
- **Payment Component:** ProcessPayment(cardDetails, amount), RefundPayment(transactionID).
- **Search Component:** SearchProduct(keyword), GetProductDetails(productID).

Why Design?

- Design is the place where ***quality*** is fostered in software engineering.
- Design is the only way that we can accurately translate a customers requirements into a finished software product or system.
- Without design we risk building an unstable system.

Design Principles



Design Principles

- The design process should not suffer from ‘tunnel vision.’
- The design should be traceable to the analysis model.
- The design should not reinvent the wheel.
- The design should “minimize the intellectual distance” [DAV95] between the software and the problem as it exists in the real world.
- The design should exhibit uniformity and integration.

Design Principles (Continue)

- The design should be structured to accommodate change.
- The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered.
- Design is not coding, coding is not design.
- The design should be assessed for quality as it is being created, not after the fact.
- The design should be reviewed to minimize conceptual (semantic) errors.

Design Concepts

- **Abstraction :** it permits one to concentrate on a problem at some level of generalization without regard to irrelevant low level details**. (Data, Procedure, Control)**
- **Refinement:** elaboration of detail for all abstractions
- **Modularity:** compartmentalization of data and function
- **Architecture:** overall structure of the software
  - Styles and patterns

Design Concepts (Continue)

- **Control Hierarchy:** represents the organisation of program components (modules) and implies a hierarchy of control.
- **Structural Partitioning:** the program structure can be partitioned both horizontally & Vertically.
- **Data Structure:** Data structure is a representation of the logical relationship among individual elements of data.

Design Concepts (Continue)

- **Software Procedure**: the algorithms that achieve function

- **Hiding**: controlled interfaces

Abstraction

It permits one to concentrate on a problem at some level of generalization without regard to irrelevant low-level details.

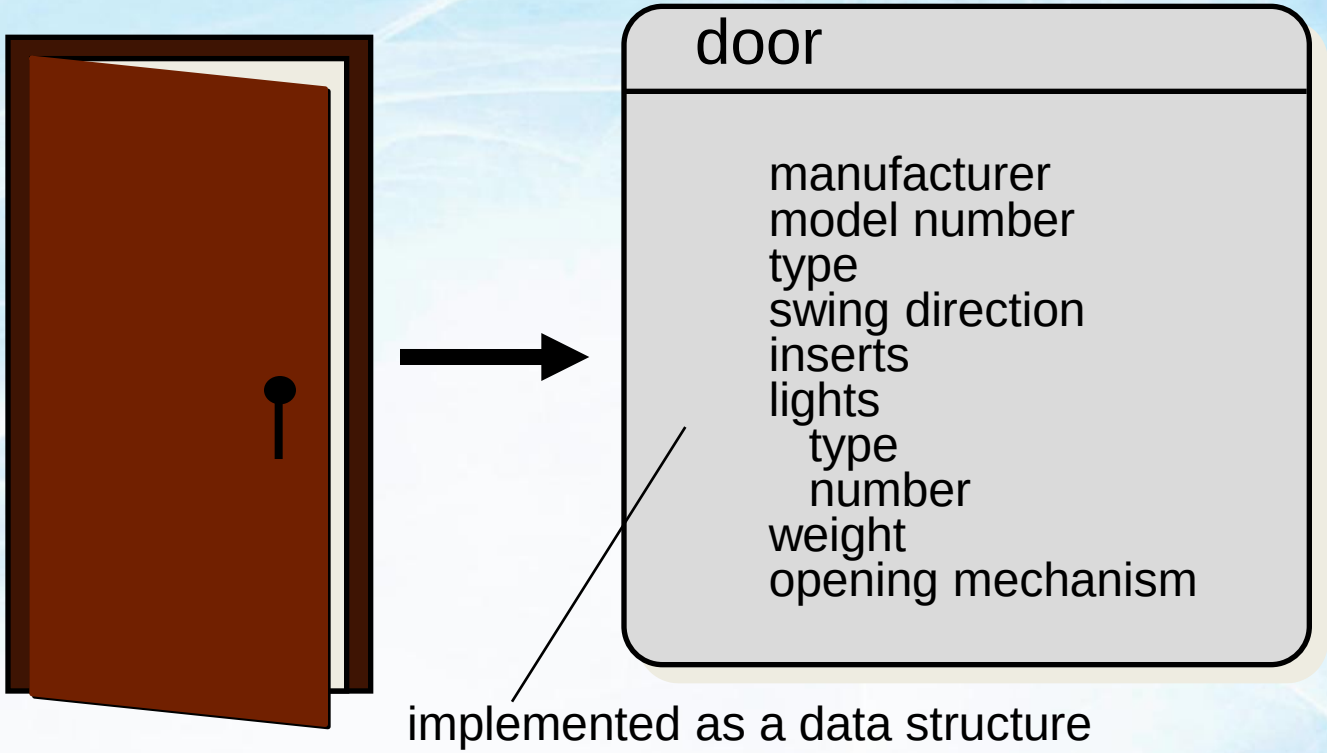
(Data, Procedure, Control)

IEEE defines abstraction as 'a view of a problem that extracts the essential information relevant to a particular purpose and ignores the remainder of the information.'

At the highest level: an outline of the solution

at the lower levels, the solution to the problem is presented in detail.

Data Abstraction

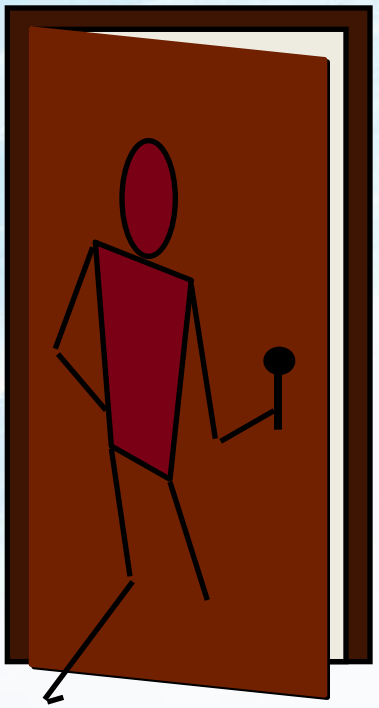


Data Abstraction:

•This refers to representing data in a way that hides its complexity and provides a simplified view for users or other systems.

•**Example**: In object-oriented programming (OOP), classes and objects abstract the details of data storage and manipulation, providing a clean interface for users or other components to interact with the data.

Procedural Abstraction



open

Details of enter algorithm

Walk to door, Reach out & grasp knob, turn knob & pull door, step away from moving door

implemented with a "knowledge" of the object that is associated with enter

Refers to the practice of defining procedures (functions or methods) to perform specific tasks while hiding the implementation details. It allows software systems to be designed in a **modular, reusable, and maintainable** way.

Procedure abstraction is like a **restaurant menu**:
- You order a dish (call a function).
- The kitchen prepares it (executes the function).
- You don't need to know how it's cooked (implementation is hidden).

Control Abstraction

It involves hiding the implementation details of control structures (like loops, conditionals, or function calls) so that the user or other components do not need to understand the intricate details of how tasks are performed.

•**Example**: In software design, a service layer may abstract the complex logic of interacting with a database or external APIs, offering a simplified interface to other parts of the system.

Stepwise Refinement:

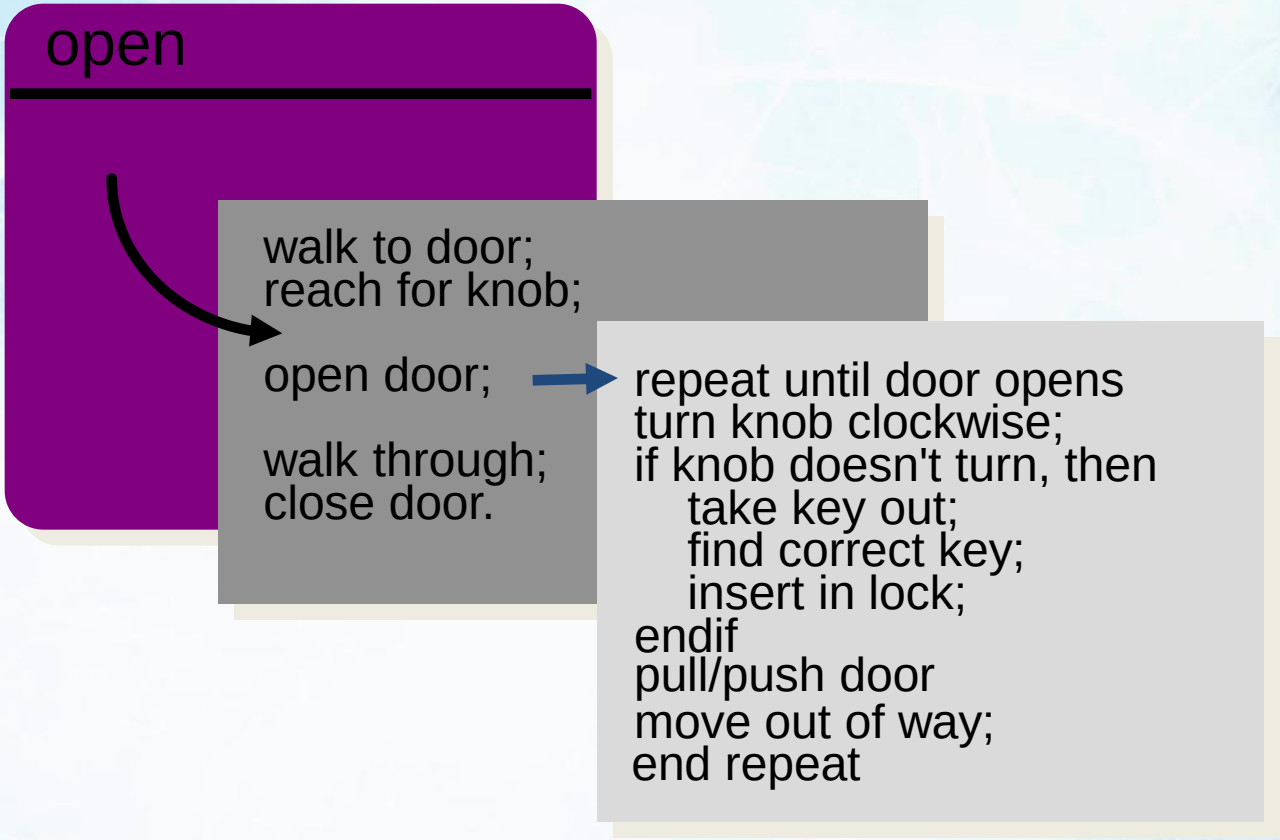
elaboration of detail for all abstractions

Abstraction & Refinement are complementary concepts.

Abstraction enables a designers to specify procedure & data & yet suppress low level details.

Refinements helps the designer to reveal low level details as design progress.

Stepwise Refinement



Modularity: compartmentalization of data and function

- S/w divided into separately named &addressable components..called modules.
- Monolithic software cannot be easily grasped by readers.
- One observation of human problem solving :
- Divide n conquer mechanism

Note: undermodularity or overmodularity should be avoided.

Modular Design

easier to build, easier to change, easier to fix ...

Key Characteristics of Modular Design

- **Decomposition:** The system is divided into smaller, independent modules.
- **Encapsulation:** Each module hides its internal details and exposes only necessary functionality.
- **Interchangeability:** Modules can be modified or replaced without affecting the entire system.
- **Reusability:** Modules can be reused across different projects or parts of the same application.
- **Scalability:** New features can be added as new modules without altering the existing system.
- **Maintainability:** Debugging and updating become easier as changes in one module do not impact others significantly.

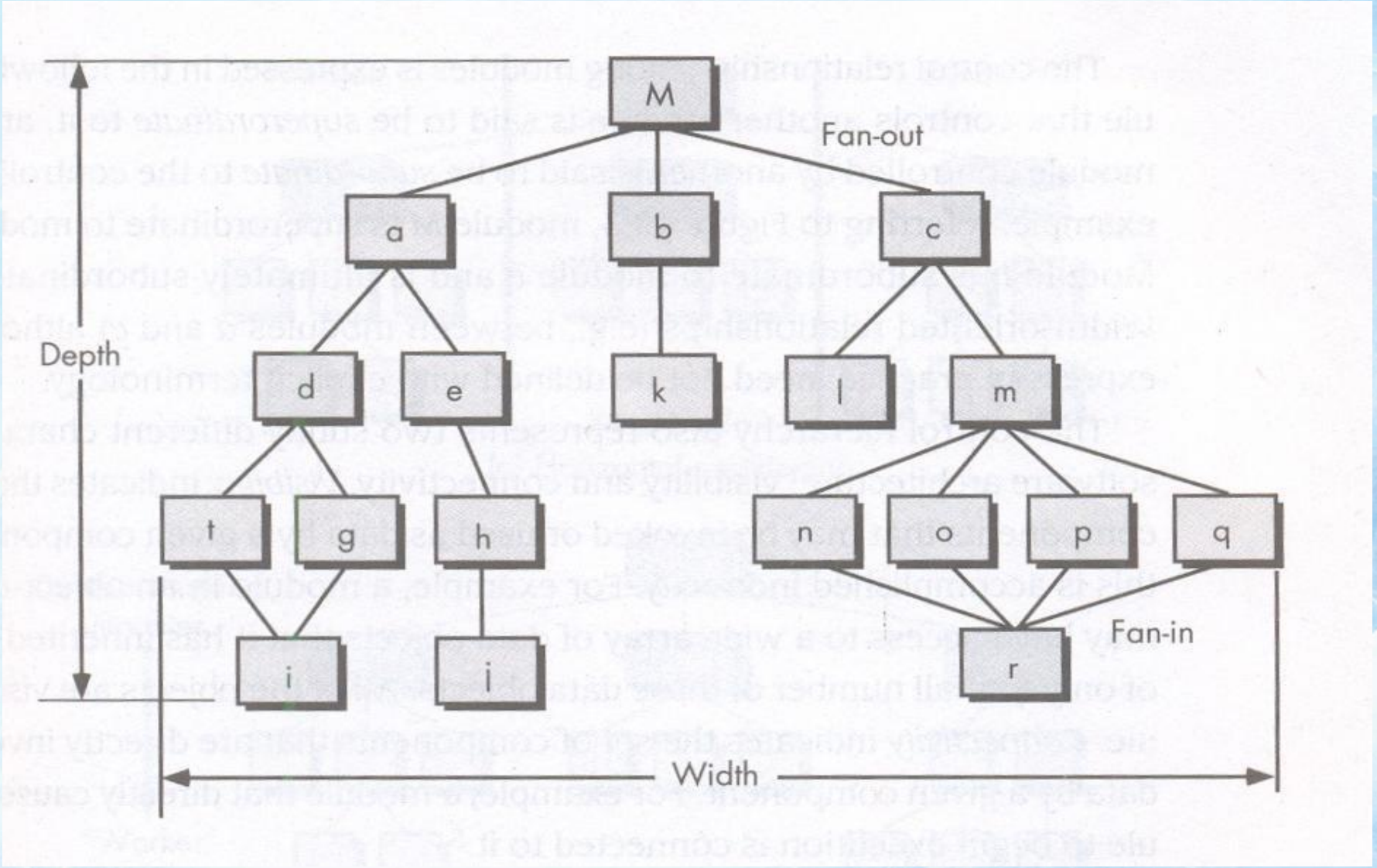
Software Architecture: overall structure of the software

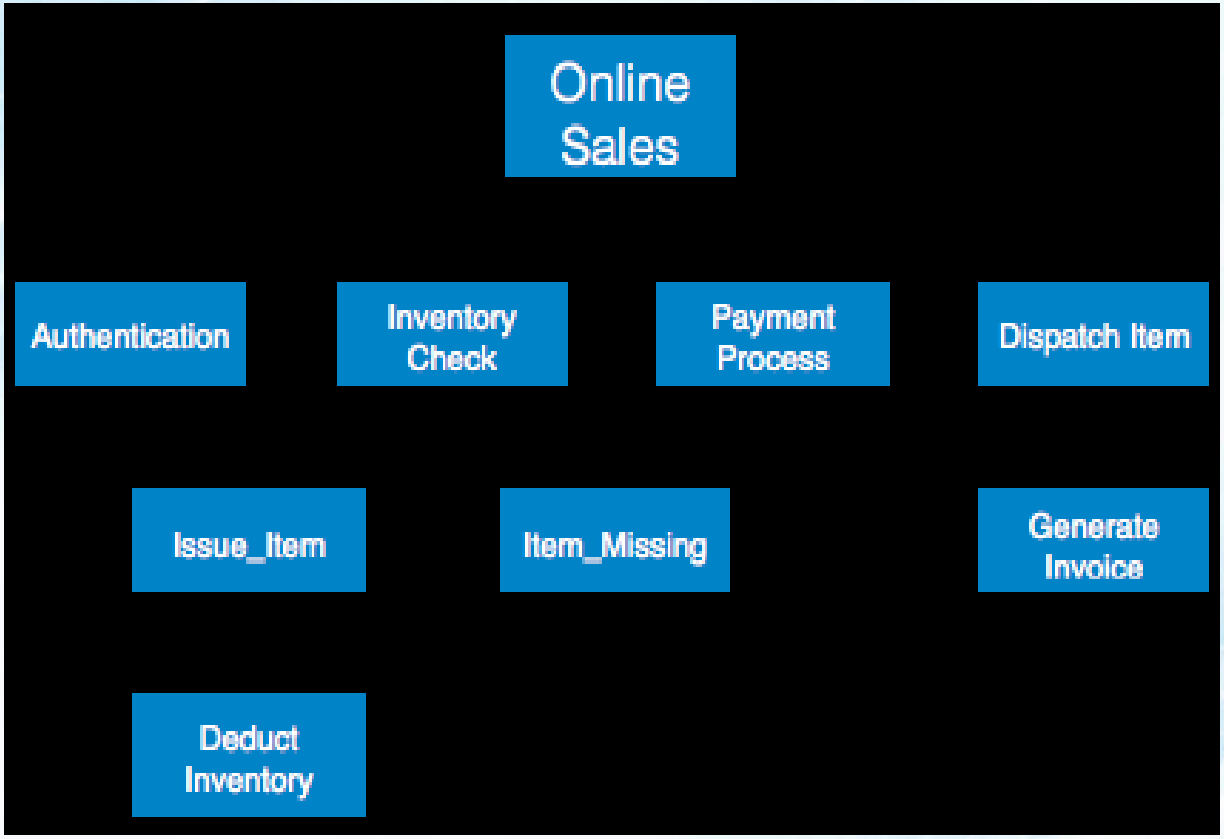
- Provides the overall structure of the software & the ways in which that structure provides conceptual integrity for a system.
- Properties of Software Architecture:
    1. Structural properties: components like modules, objects..
    2. Extra functional properties: performance, capacity, reliability, security etc.
    3. Families of related systems: reusability

Control Hierarchy: represents the organisation of program components (modules) and implies a hierarchy of control.

- Program structure represents organization of program components & implies hierarchy of control.
- Doesn't represent procedural aspects of s/w.
- Depth& width….level & span of control
- Fan-out: measure of no. of modules thar are directly controlled by other modules.
- Fan-in: how many modules directly control a given module.
- Superordinate & subordinate.

Control Hierarchy




Online Sales
Authentication
Inventory Check
Payment Process
Dispatch Item
Issue_Item
Item_Missing
Generate Invoice
Deduct Inventory

Software Design

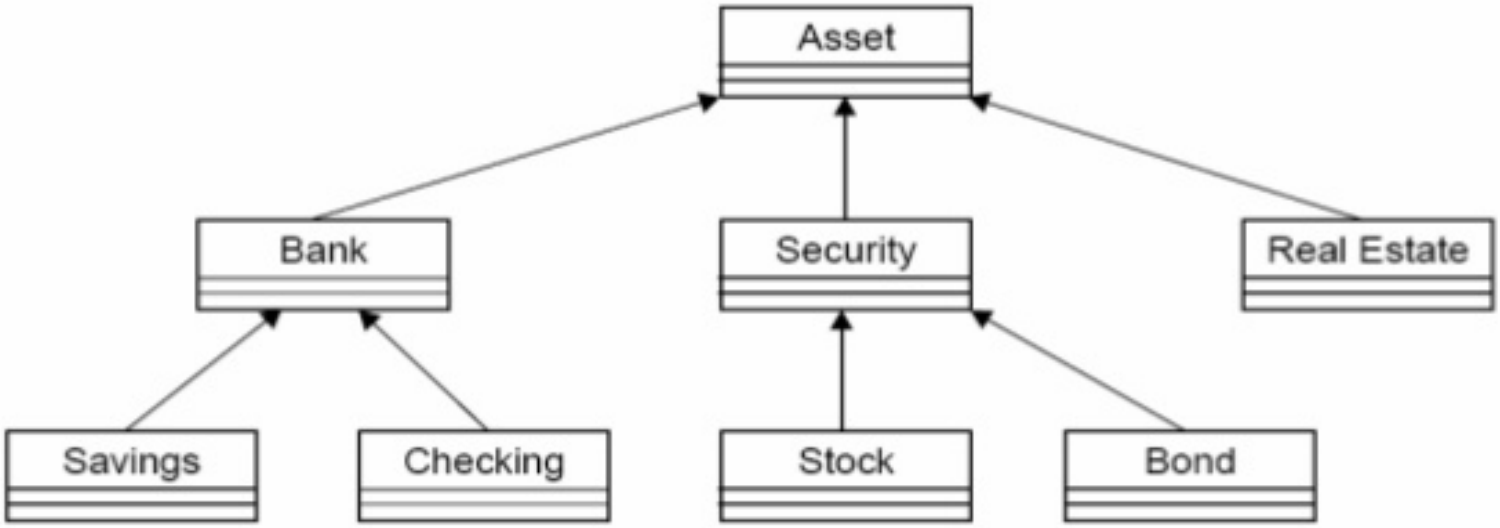
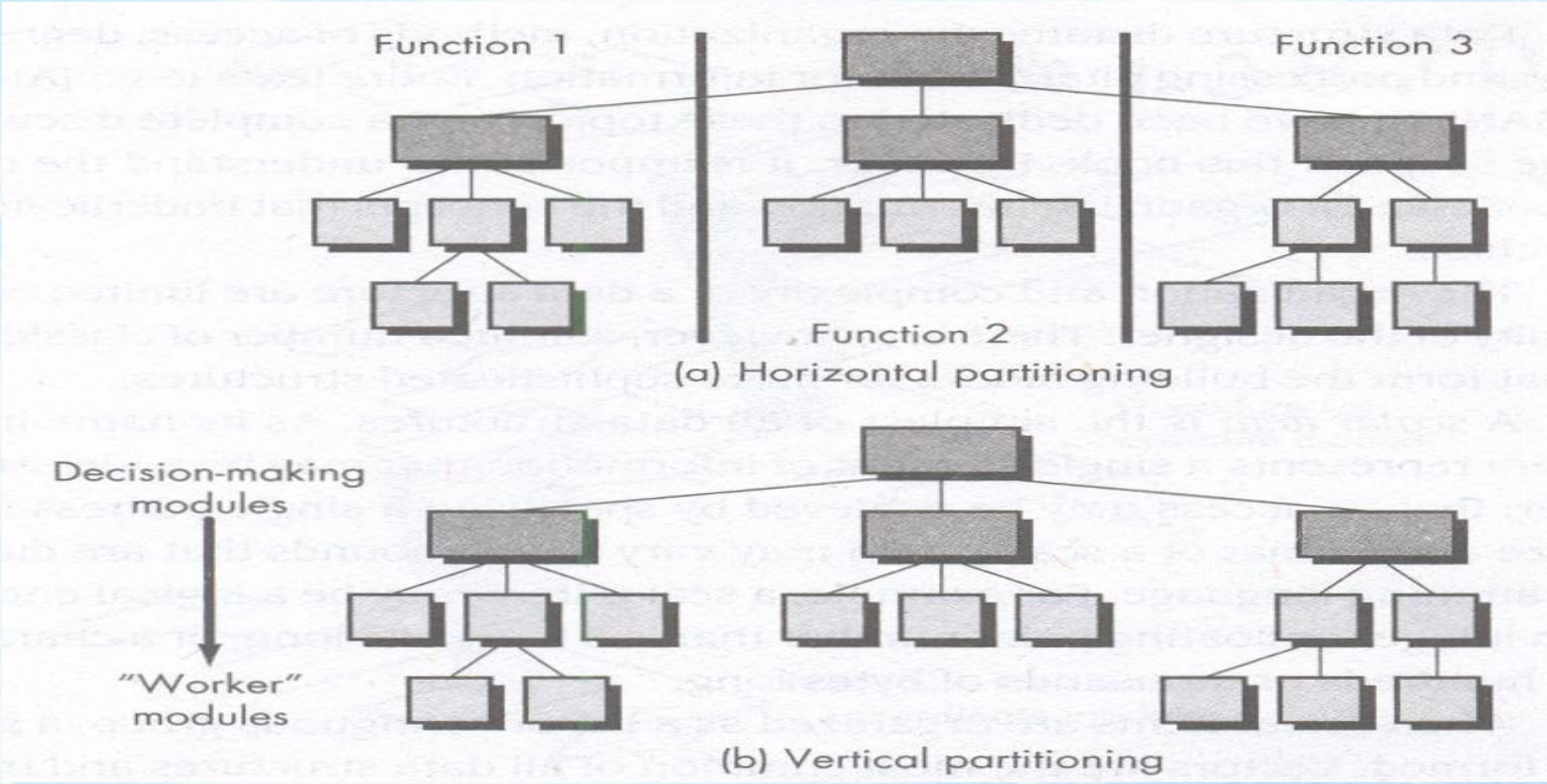


Fig. 24: Hierarchy

Structural Partitioning: the program structure can be partitioned both horizontally & Vertically.

If architectural style is hierarchical, the program structure can be partitioned both way.

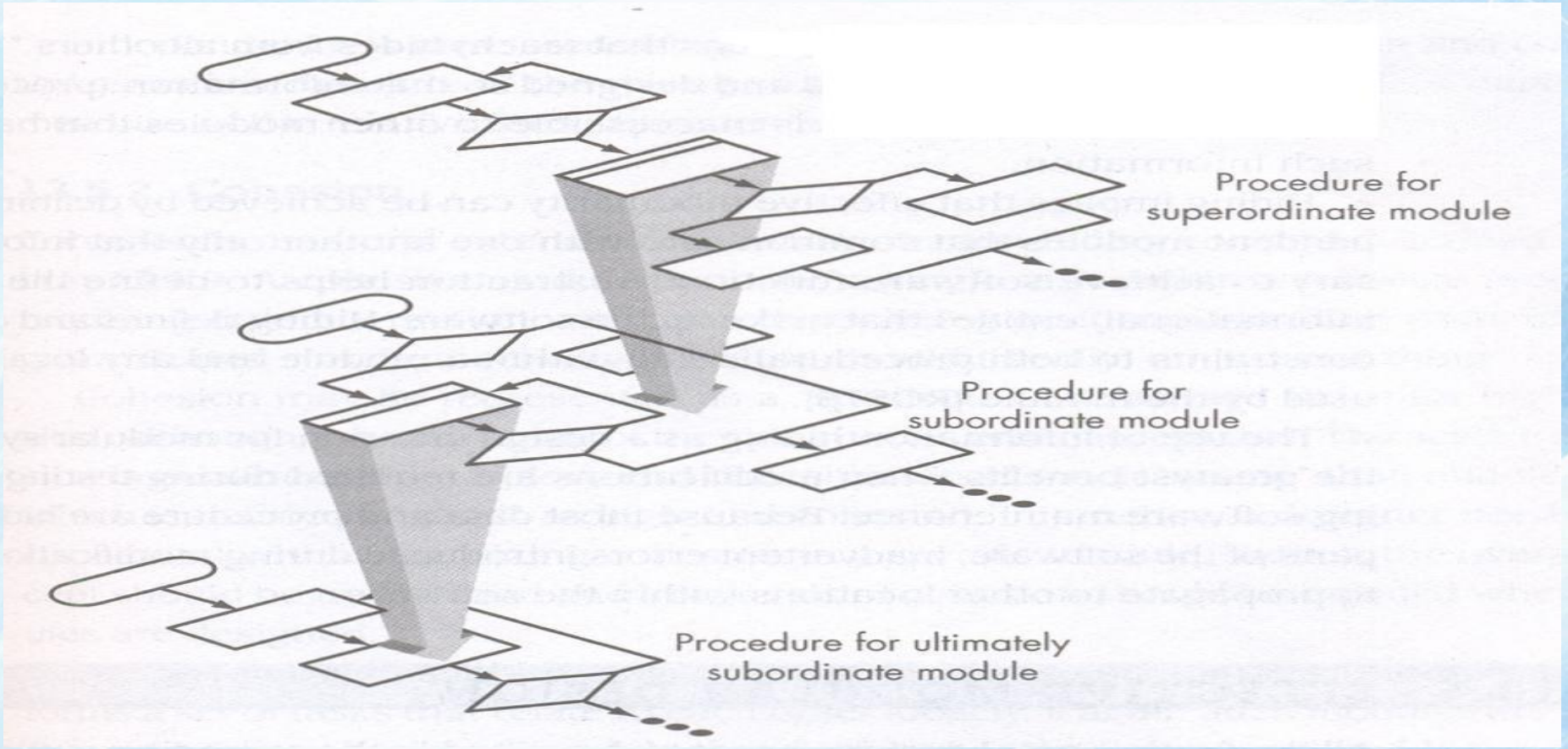


Data structure: Data structure is a representation of the logical relationship among individual elements of data.

- Data structure is a representation of the logical relationship among individual elements of data.
- Sequential Vector.
- Linked list
- Hierarchical data structure .

Software Procedure: the algorithms that achieve function

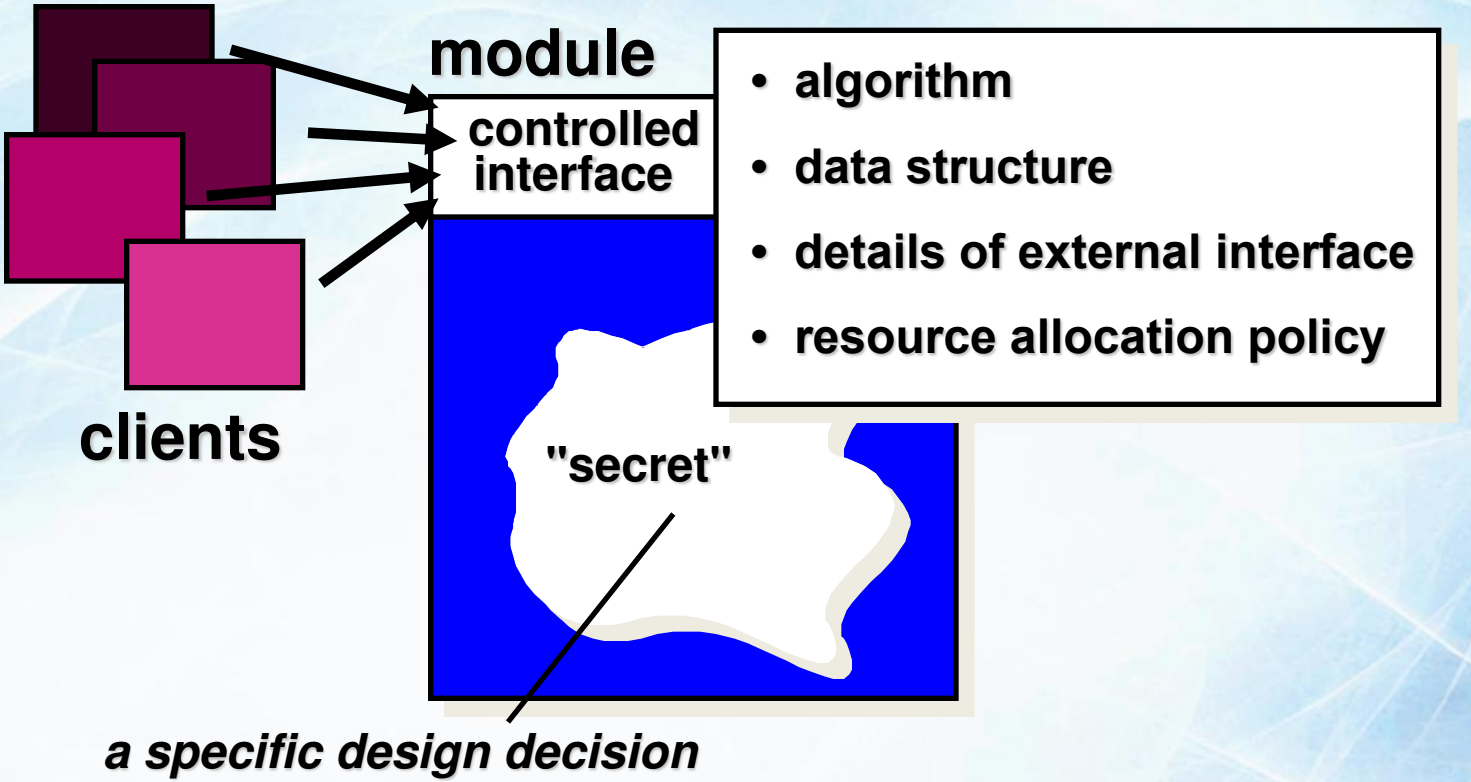
- Focuses on processing details of each module individually.
- Should provide specification including sequence of events, exact decision points, repetitive operations, data organization & structure.

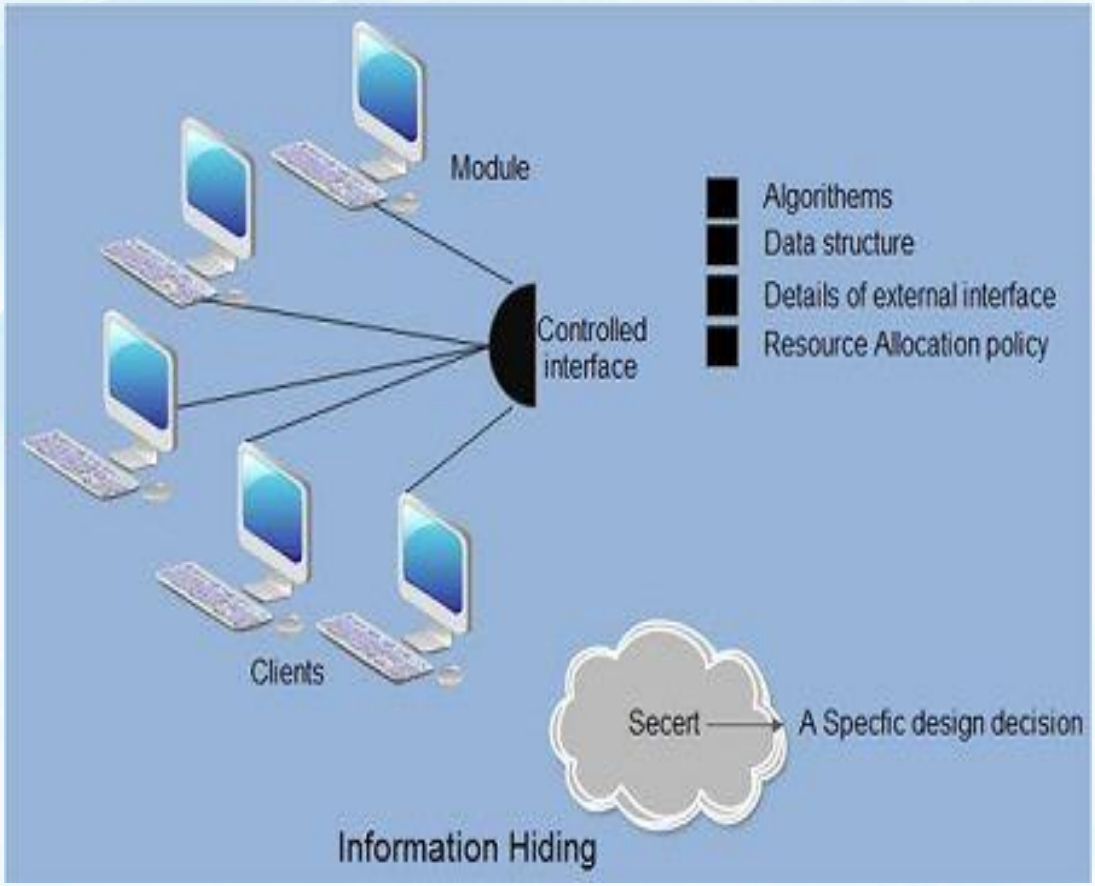


Information Hiding

- Modules should be specified and designed in such a way that the data structures and processing details of one module are not accessible to other modules.
- They pass only that much information to each other, which is required to accomplish the software functions.
- The way of hiding unnecessary details is referred to as information hiding.

Information Hiding: controlled interfaces




Module
Controlled interface
Algorithems
Data structure
Details of external interface
Resource Allocation policy
Clients
Secert
A Specfic design decision
Information Hiding

Why Information Hiding?

- Reduces the likelihood of “side effects”
- emphasizes communication through controlled interfaces
- discourages the use of global data
- results in higher quality software

Example: Consider a Bank Account class in an object-oriented programming language. The internal details, such as account balance and transaction history, should not be directly accessible to external code. Instead, controlled access is provided through methods.