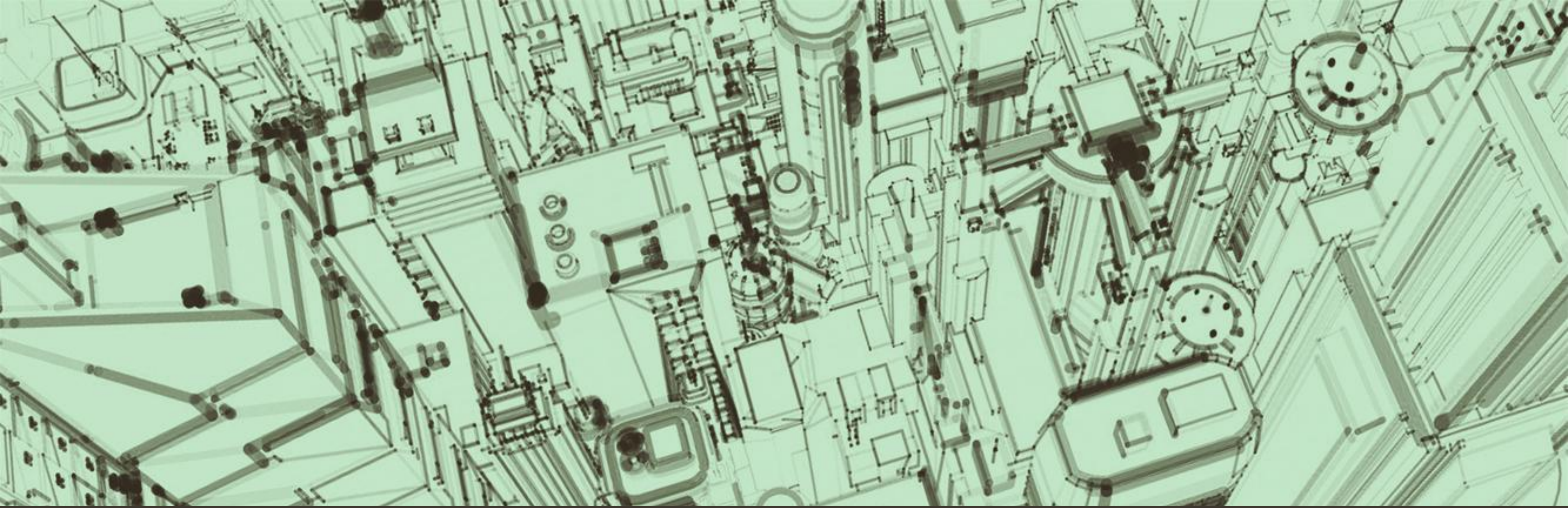


Software Engineering

Sobia Iftikhar
Sobia.Iftikhar@nu.edu.pk

Week 07

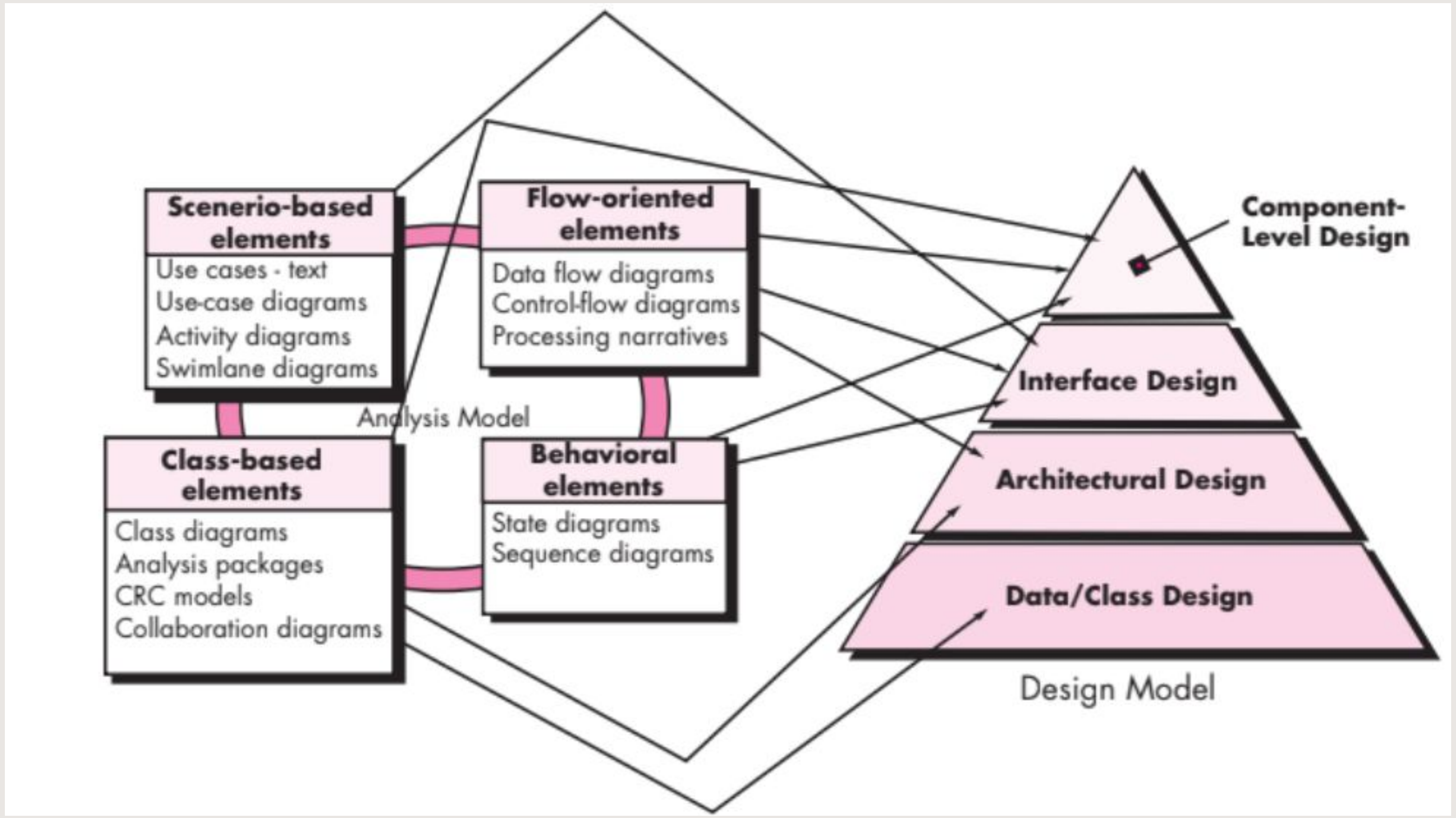
Class 16

24-March-2021

Software Design

- Software Design is the process to transform the user requirements into some suitable form, which helps the programmer in software coding and implementation.
- During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document.
- Hence the aim of this phase is to transform the SRS document into the design document.

Translating the requirements model into the design model



Design Process

- Software design is an iterative process through which requirements are translated into a “blueprint” for constructing the software.
- design iterations occur, subsequent refinement leads to design representations at much lower levels of abstraction.

1. Design Process-Software Quality Guidelines and Attributes

- Throughout the design process, the quality of the evolving design is assessed with a series of technical reviews.
- McGlaughlin [McG91] Suggests three characteristics that serve as a guide for the evaluation of a good design.

1. The design must implement all of the explicit requirements contained in the requirements model, and it must accommodate all of the implicit requirements desired by stakeholders.
2. The design must be a readable, understandable guide for those who generate code and for 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.

how is each of these goals achieved?

1. Quality Guidelines. software team must establish technical criteria for good design.
    1. A design should exhibit an architecture
    2. A design should be modular
    3. A design should contain distinct representations of data
    4. A design should lead to data structures
    5. A design should lead to interfaces that reduce the complexity of connections

how is each of these goals achieved?

1. Quality Attributes. The FURPS quality attributes represent a target for all software design:
   1. **Functionality** is assessed by evaluating the feature set and capabilities of the program
   2. **Usability** is assessed by considering human factors, usability is how easily a person can accomplish a given task with your product.
   3. **Reliability** is evaluated by measuring the frequency and severity of failure, the accuracy of output results, the mean-time-to-failure (MTTF), the ability to recover from failure, and the predictability of the program.
   4. **Performance** is measured by considering processing speed, response time, resource consumption, throughput, and efficiency.
   5. **Supportability** combines the ability to extend the program (extensibility), adaptability, serviceability.

2. The Evolution of Software Design

- Several design methods, growing out of the work just noted, are being applied throughout the industry.
- each software design method introduces unique heuristics and notation.
- All of these methods have a number of common characteristics:
  - (1) a mechanism for the translation of the requirements model into a design representation,
  - (2) a notation for representing functional components and their interfaces, (3) heuristics for refinement and partitioning.
  - (4) guidelines for quality assessment.

Software Design Concepts:

- The **software design concept** simply means the idea or principle behind the design.
- It describes how you plan to solve the problem of designing software, the logic, or thinking behind how you will design software.
- It allows the software engineer to create the model of the system or software or product that is to be developed or built.
- The software design concept provides a supporting and essential structure or model for developing the right software. There are many concepts of software design and some of them are given below:

Fundamental Concepts

Abstraction—data, procedure, control	Architecture—the overall structure of the software	Patterns—”conveys the essence” of a proven design solution	Separation of concerns—any complex problem can be more easily handled if it is subdivided into pieces
Modularity—compartmentalization of data and function	Hiding—controlled interfaces	Functional independence—single-minded function and low coupling	Refinement—elaboration of detail for all abstractions
Aspects—a mechanism for understanding how global requirements affect design	Refactoring—a reorganization technique that simplifies the design	OO design concepts—Appendix II	Design Classes—provide design detail that will enable analysis classes to be implemented

Class 17

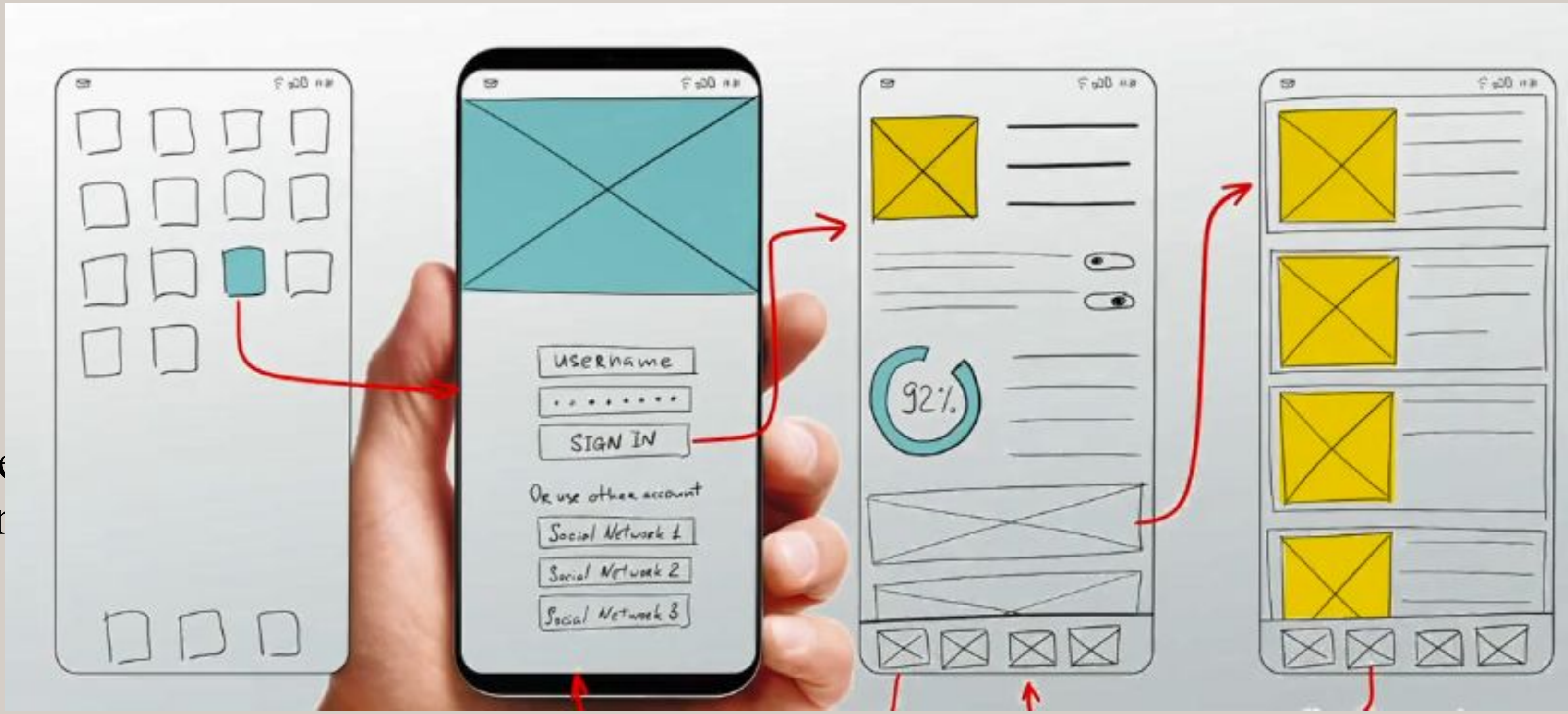
25-March-2021

There are many concepts of software design and some of them are given below:

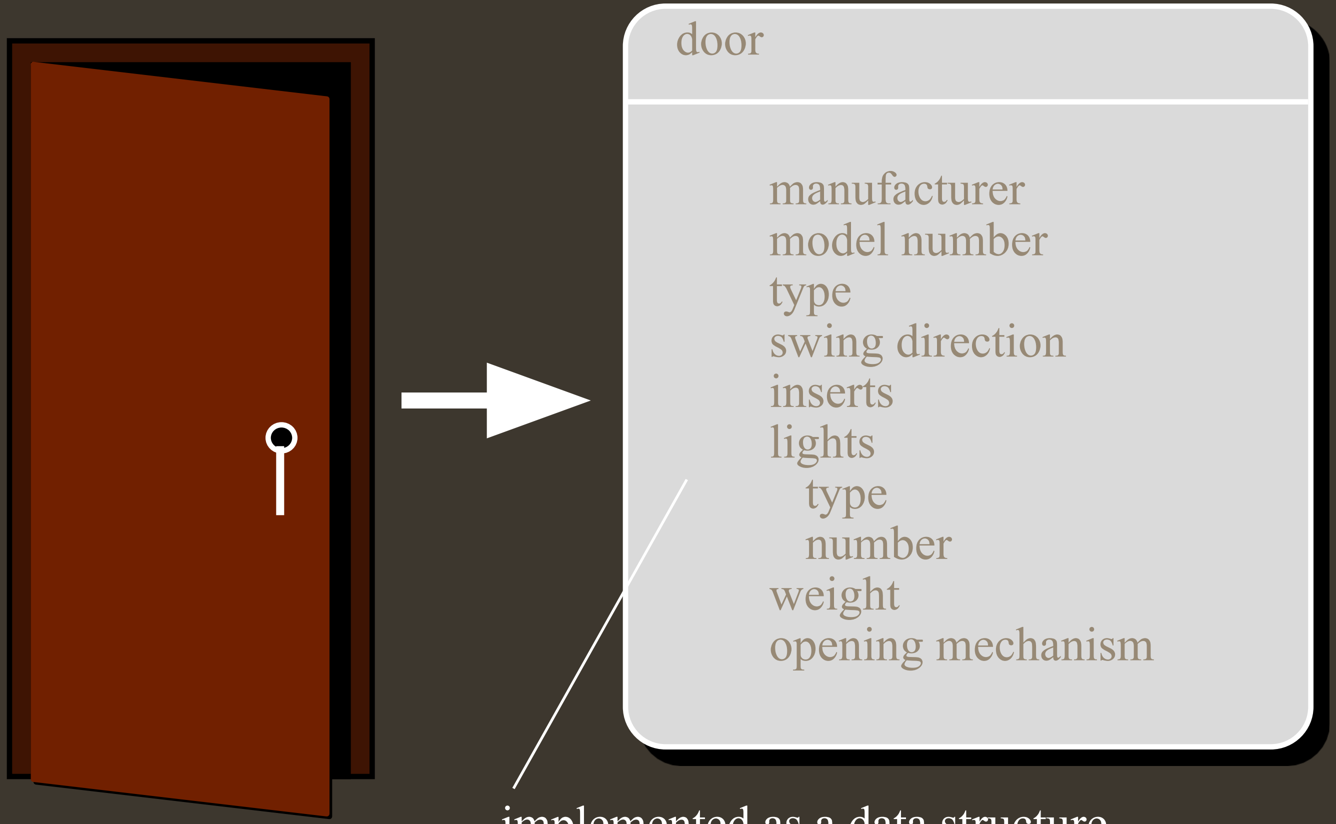
- Modularity
- Information
- Hiding
- Pattern
- Refinement
- Architecture
- Refactoring

1. Abstraction

- The process of hiding the complex properties or characteristic from the software it self to keep the things more simple.
- A sequence of instruction that contain a specific and limited function refers in a procedural abstraction.
- A collection of data that describe a data object is a data abstraction

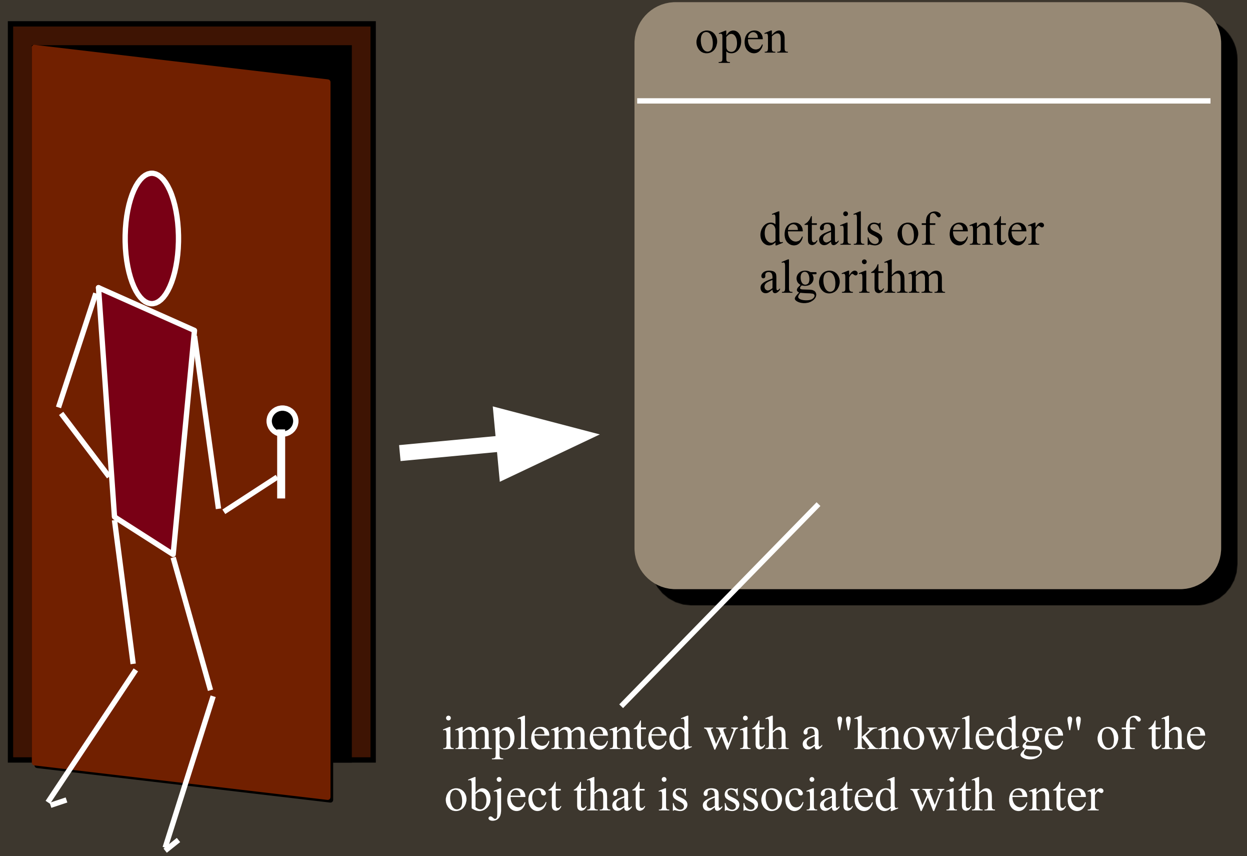


Data Abstraction



implemented as a data structure

Procedural Abstraction



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

2. Architecture

- The complete structure of the software is known as software architecture.
- Structure provides conceptual integrity for a system in a number of ways.

Structural properties. This aspect of the architectural design representation defines the components of a system (e.g., modules, objects, filters) and the manner in which those components are packaged and interact with one another. For example, objects are packaged to encapsulate both data and the processing that manipulates the data and interact via the invocation of methods

Extra-functional properties. The architectural design description should address how the design architecture achieves requirements for performance, capacity, reliability, security, adaptability, and other system characteristics.

Families of related systems. The architectural design should draw upon repeatable patterns that are commonly encountered in the design of families of similar systems. In essence, the design should have the ability to reuse architectural building blocks. a

3. Patterns

1. “A pattern is a named nugget of insight which conveys the essence of a proven solution to a recurring problem within a certain context amidst competing concerns
2. A design pattern can be characterized as “a three-part rule which expresses a relation between a certain context, a problem, and a solution
3. **Component pattern:** These address a specific element of the design such as an aggregation of components to solve some design problem, relationships among elements.
4. **Design Pattern:** A medium level pattern type that is used by developer to solve the problem in the design stage of development.

Patterns

Design Pattern Template

***Pattern name*—describes the essence of the pattern in a short but expressive name**

***Intent*—describes the pattern and what it does**

***Also-known-as*—lists any synonyms for the pattern**

***Motivation*—provides an example of the problem**

***Applicability*—notes specific design situations in which the pattern is applicable**

***Structure*—describes the classes that are required to implement the pattern**

***Participants*—describes the responsibilities of the classes that are required to implement the pattern**

***Collaborations*—describes how the participants collaborate to carry out their responsibilities**

***Consequences*—describes the “design forces” that affect the pattern and the potential trade-offs that must be considered when the pattern is implemented**

***Related patterns*—cross-references related design patterns**

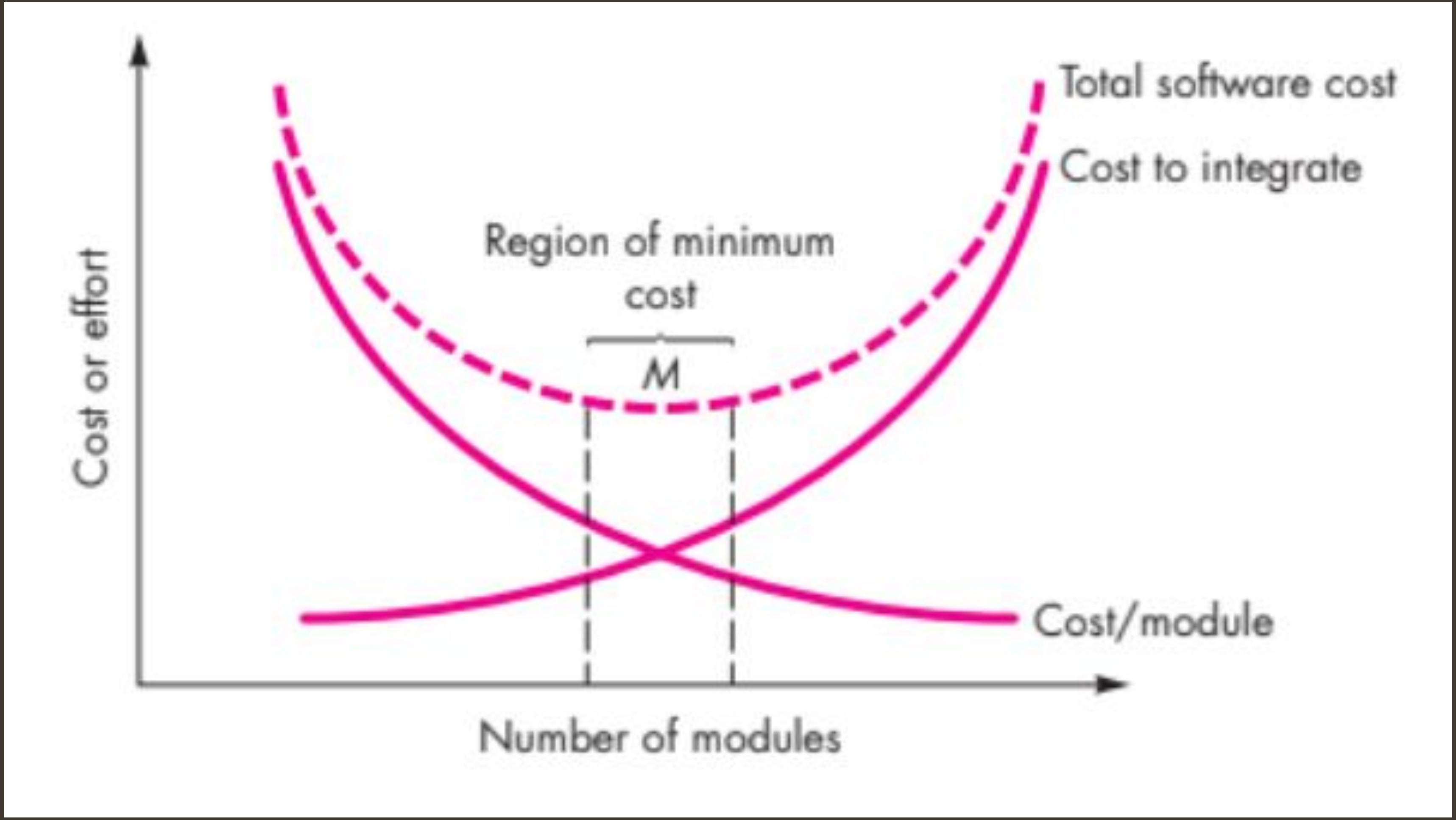
4. Modularity

Software is separately divided into name and addressable components. Sometimes they are called as modules which integrate to satisfy the problem requirements.

Modularity is the single attribute of a software that permits a program to be managed easily.

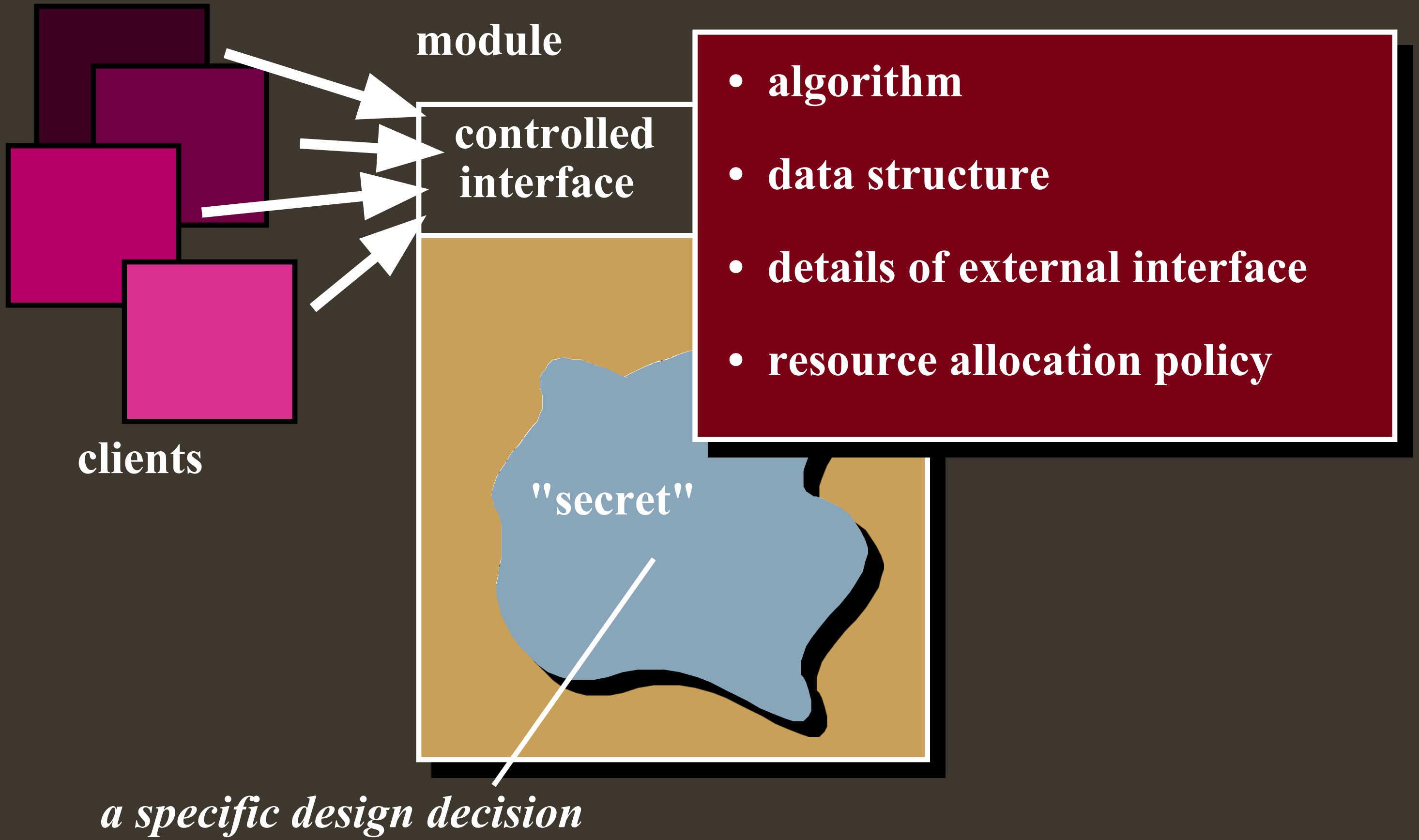
Monolithic software (i.e., a large program composed of a single module)

Modularity and Software Cost



5. Information Hiding

Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

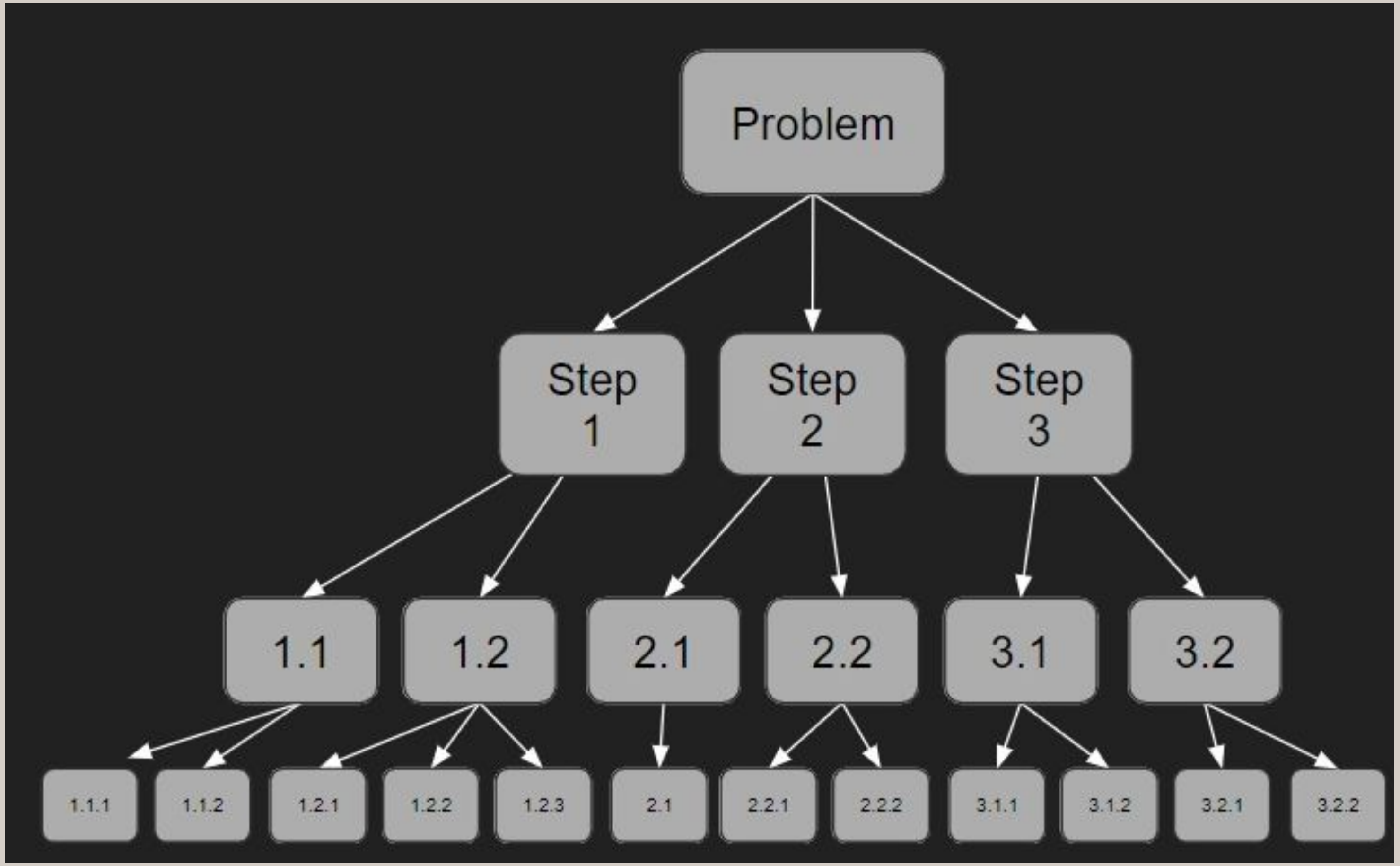

module
controlled interface
• algorithm
• data structure
• details of external interface
• resource allocation policy
clients
"secret"
a specific design decision

6. Functional Independence

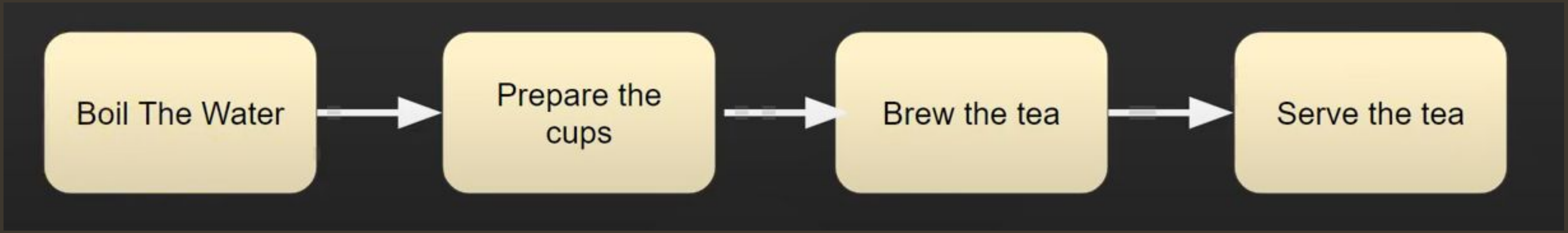
- Functional independence is achieved by developing modules with “single-minded” function and an “aversion” to excessive interaction with other modules.
  - Cohesion is an indication of the relative functional strength of a module.
  - Coupling is an indication of the relative interdependence among modules.

7. **Refinement** is a top-down design approach.

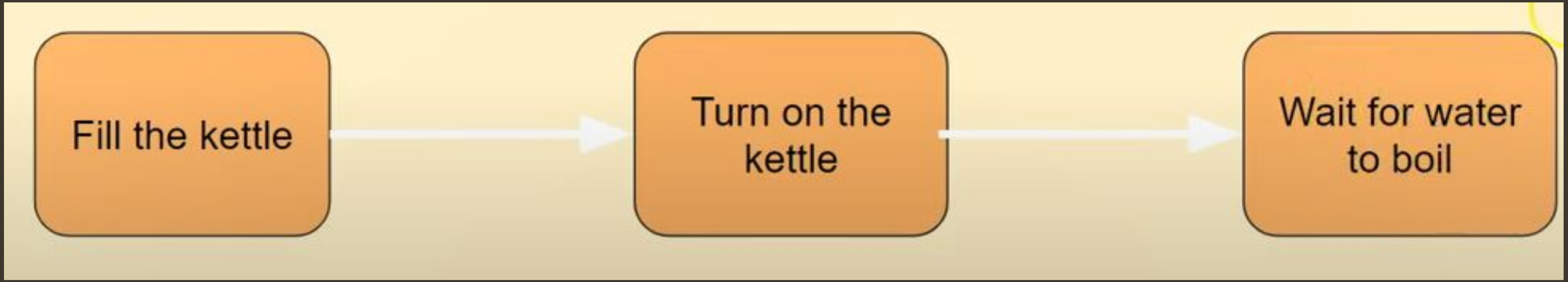
- Stepwise Refinement is the process of breaking down a programming problem into a series of steps. You start with a general set of steps to solve the problem, defining each in turn.
- Once you have defined each of the steps you then break the problem down into a series of smaller sub-steps.
- You keep on going until you have described the problem in such a level of detail that you can code a solution to the problem.



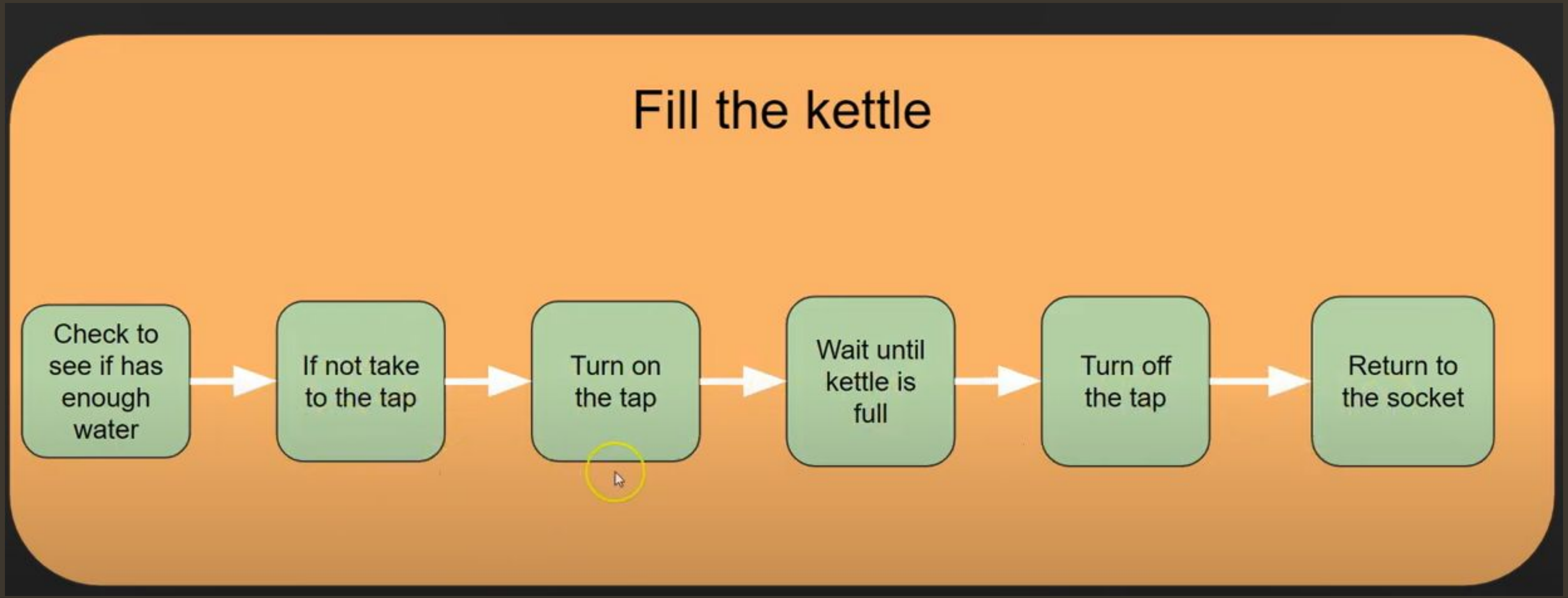
Example- refinement

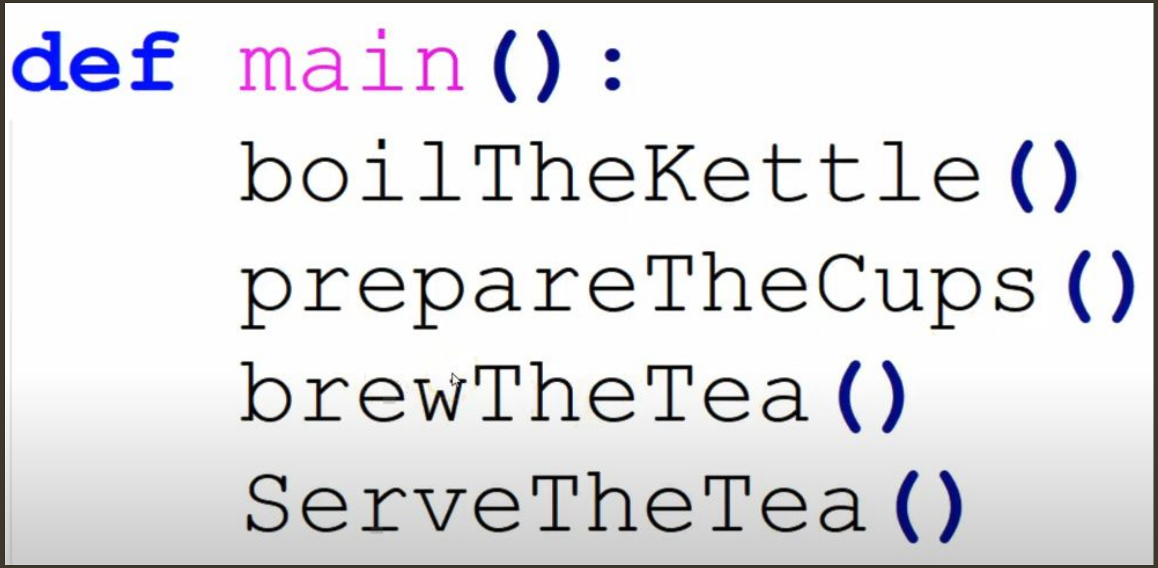


Step-01 boil water

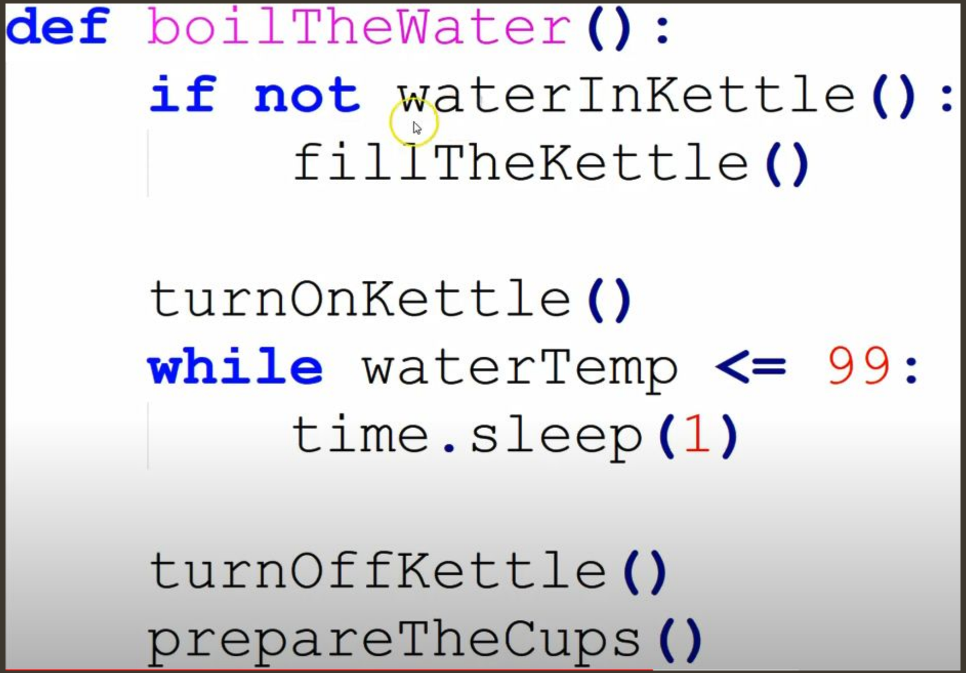


Step-02





```
def main():
    boilTheKettle()
    prepareTheCups()
    brewTheTea()
    ServeTheTea()
```



```
def boilTheWater():
    if not waterInKettle():
        fillTheKettle()

    turnOnKettle()
    while waterTemp <= 99:
        time.sleep(1)

    turnOffKettle()
    prepareTheCups()
```

Refactoring

Refactoring is a reorganization technique that simplifies the design (or internal code structure) of a component without changing its function or external behaviour.

It removes redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures, or any other design failures.

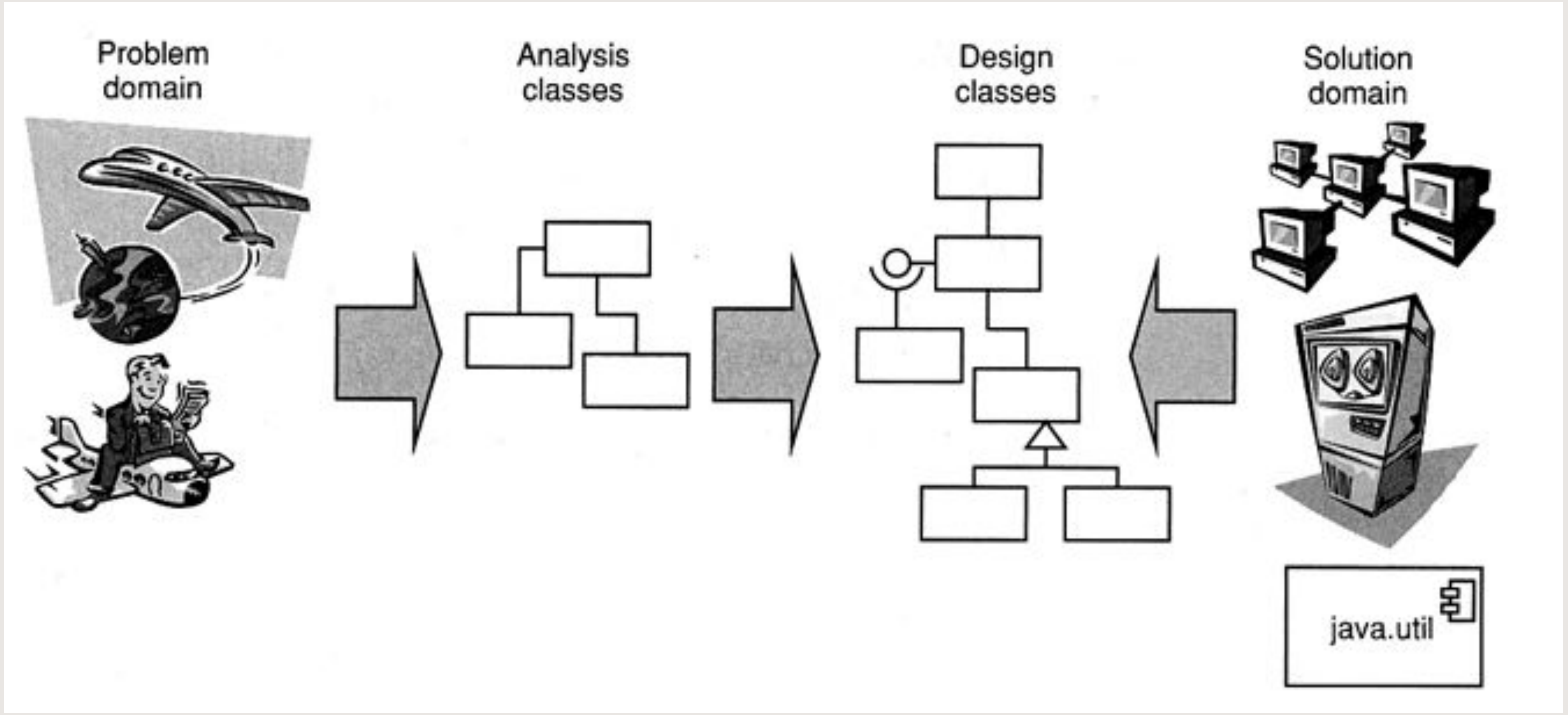
OO Design Concepts

- Design classes
  - Entity classes
  - Boundary classes
  - Controller classes
- Inheritance—all responsibilities of a superclass is immediately inherited by all subclasses
- Messages—stimulate some behavior to occur in the receiving object
- Polymorphism—a characteristic that greatly reduces the effort required to extend the design

Class 18
26-March-2021

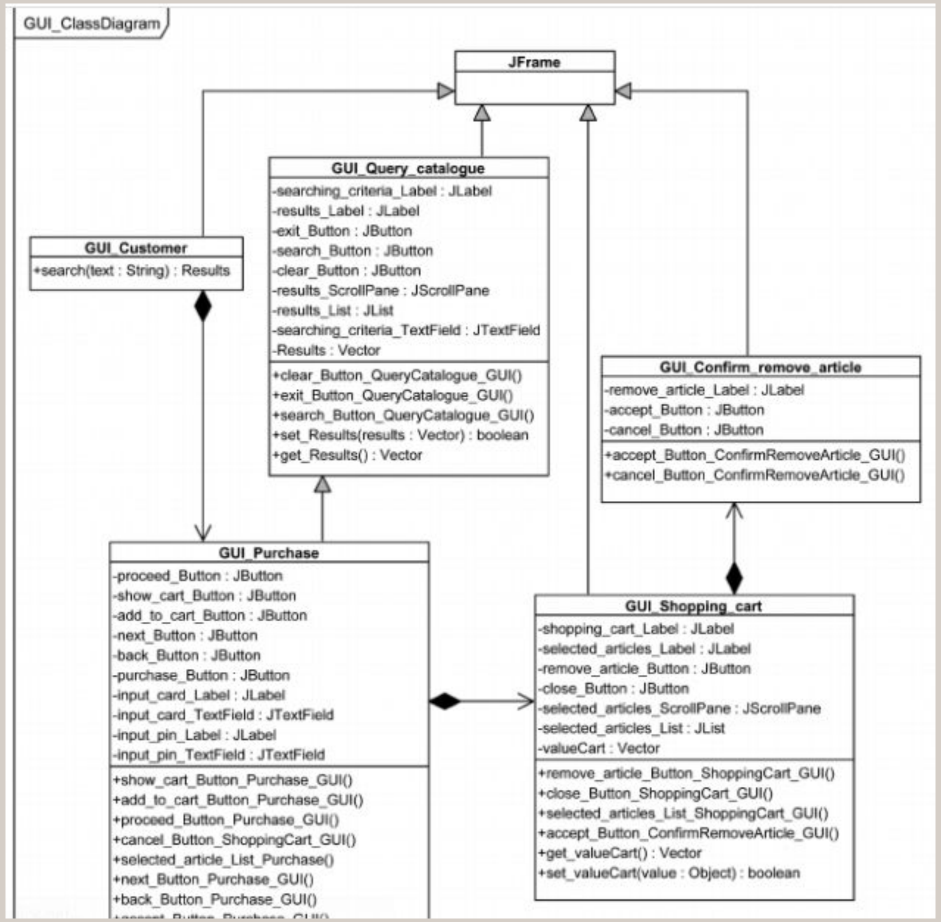
Design Classes

- The requirements model defines a set of analysis classes.



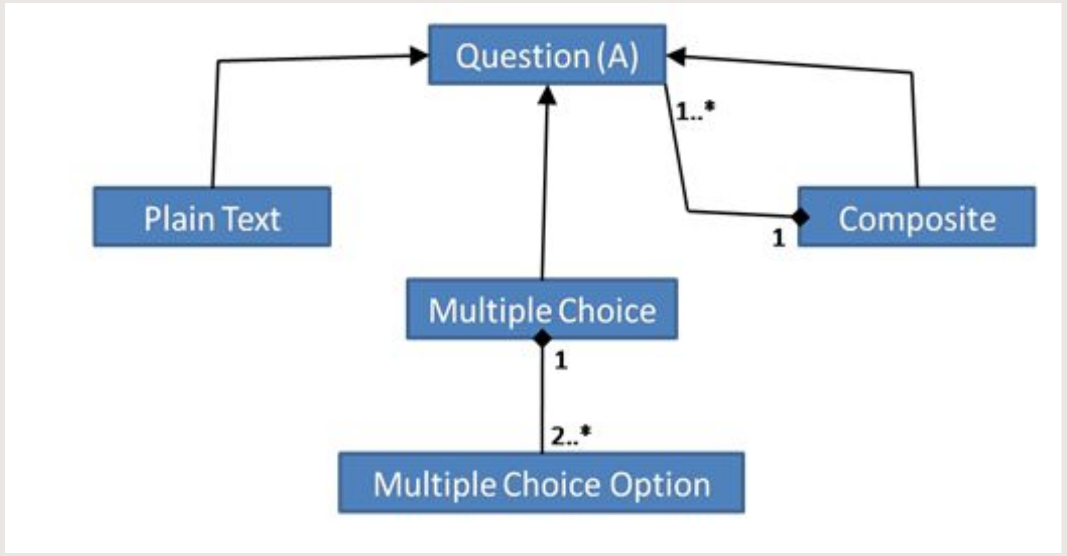
- Five different types of design classes, each representing a different layer of the design architecture
  - User Interface Classes
  - Business domain classes
  - Process classes
  - Persistent classes
  - System classes

User interface classes


GUI_ClassDiagram
JFrame
GUI_Query_catalogue
-searching_criteria_Label : JLabel
-results_Label : JLabel
-exit_Button : JButton
-search_Button : JButton
-clear_Button : JButton
-results_ScrollPane : JScrollPane
-results_List : JList
-searching_criteria_TextField : JTextField
-Results : Vector
+clear_Button_QueryCatalogue_GUI()
+exit_Button_QueryCatalogue_GUI()
+search_Button_QueryCatalogue_GUI()
+set_Results(results : Vector) : boolean
+get_Results() : Vector
GUI_Customer
+search(text : String) : Results
GUI_Confirm_remove_article
-remove_article_Label : JLabel
-accept_Button : JButton
-cancel_Button : JButton
+accept_Button_ConfirmRemoveArticle_GUI()
+cancel_Button_ConfirmRemoveArticle_GUI()
GUI_Purchase
-proceed_Button : JButton
-show_cart_Button : JButton
-add_to_cart_Button : JButton
-next_Button : JButton
-back_Button : JButton
-purchase_Button : JButton
-input_card_Label : JLabel
-input_card_TextField : JTextField
-input_pin_Label : JLabel
-input_pin_TextField : JTextField
+show_cart_Button_Purchase_GUI()
+add_to_cart_Button_Purchase_GUI()
+proceed_Button_Purchase_GUI()
+cancel_Button_ShoppingCart_GUI()
+selected_article_List_Purchase()
+next_Button_Purchase_GUI()
+back_Button_Purchase_GUI()
GUI_Shopping_cart
-shopping_cart_Label : JLabel
-selected_articles_Label : JLabel
-remove_article_Button : JButton
-close_Button : JButton
-selected_articles_ScrollPane : JScrollPane
-selected_articles_List : JList
-valueCart : Vector
+remove_article_Button_ShoppingCart_GUI()
+close_Button_ShoppingCart_GUI()
+selected_articles_List_ShoppingCart_GUI()
+accept_Button_ConfirmRemoveArticle_GUI()
+get_valueCart() : Vector
+set_valueCart(value : Object) : boolean

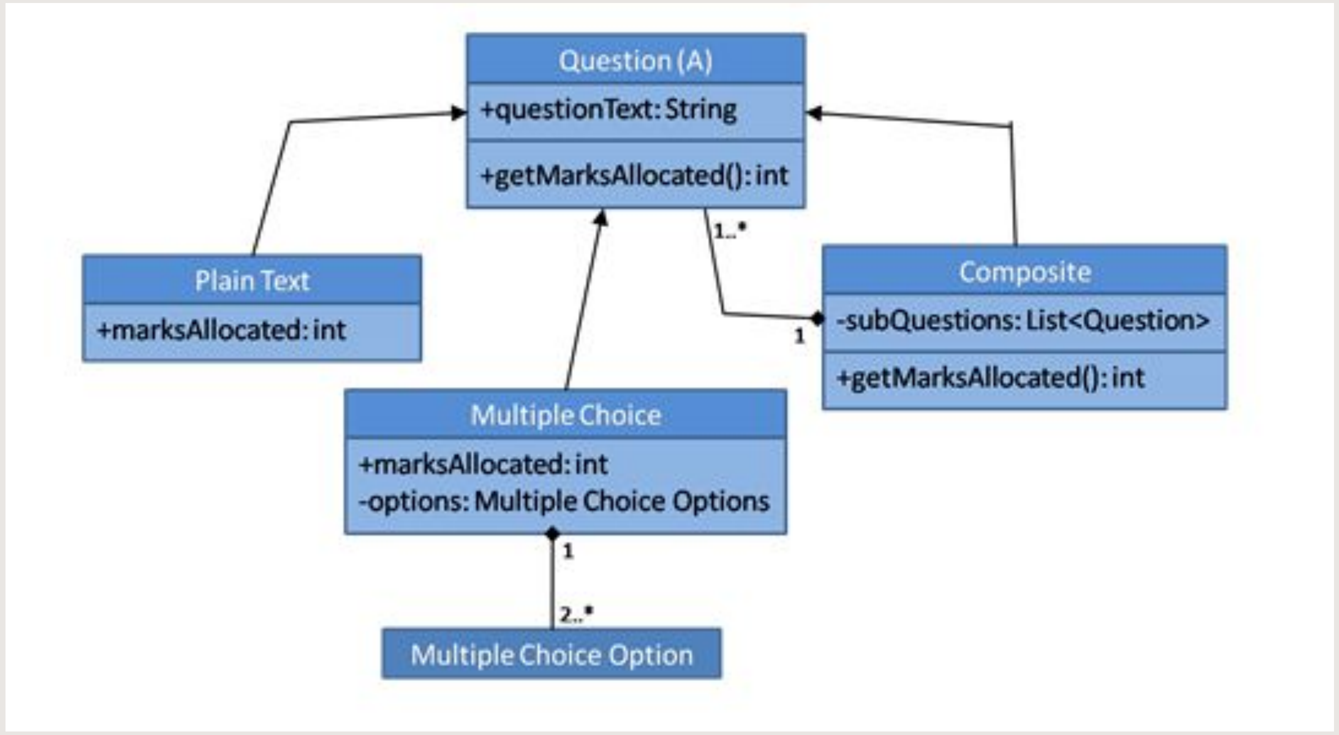
1. These classes are designed for Human Computer Interaction(HCI).
2. These interface classes define all interfaces which is required for Human Computer Interaction(HCI).

Business domain classes



- These classes are commonly refinements of the analysis classes.
- These classes are recognized as attributes and methods which are required to implement the elements of the business domain.

Process classes



- It implement the lower-level business abstraction which is needed to completely manage the business domain class.

- **4. Persistence classes**
  - It shows data stores that will persist behind the execution of the software.
  - Persistent classes are those java classes whose objects have to be stored in the database tables.
- **System Classes**
  - System classes implement software management and control functions that allow to operate and communicate in computing environment and outside world
  - System class belongs to the package **java.lang**.
  - **java.lang.System**

Design Classes-They define four characteristics of a well-formed design class:

- Complete and sufficient.
  - . A design class should be the complete encapsulation of all attributes and methods that can reasonably be expected (based on a knowledgeable interpretation of the class name) to exist for the class.
  - For example, the class Scene defined for video-editing software is complete only if it contains all attributes and methods that can reasonably be associated with the creation of a video scene. Sufficiency ensures that the design class contains only those methods that are sufficient to achieve the intent of the class, no more and no less.

Primitiveness

- Methods associated with a design class should be focused on accomplishing one service for the class. Once the service has been implemented with a method, the class should not provide another way to accomplish the same thing.

High cohesion.

Class should be focus on one kind of thing

A cohesion design class has a small and focused set of responsibilities.

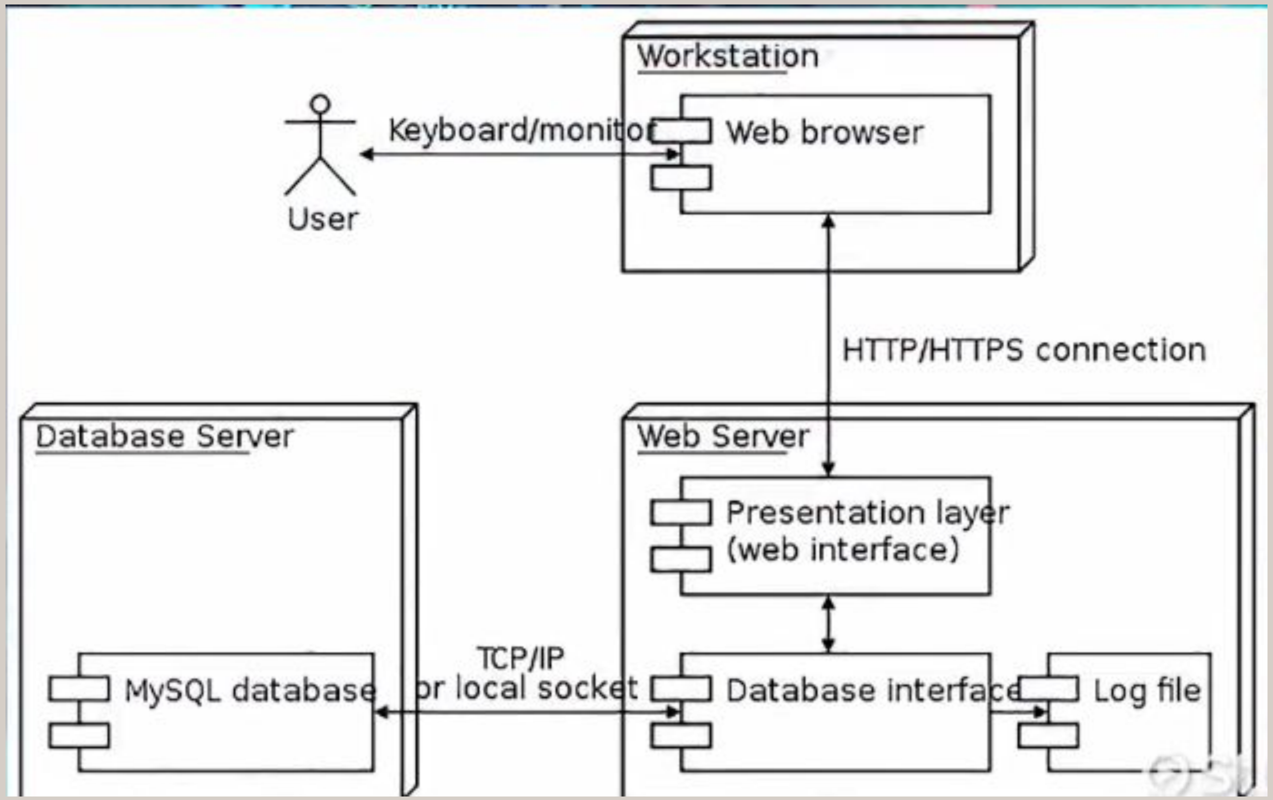
For implementing the set of responsibilities the design classes are applied single-mindedly to the methods and attribute

Low coupling.

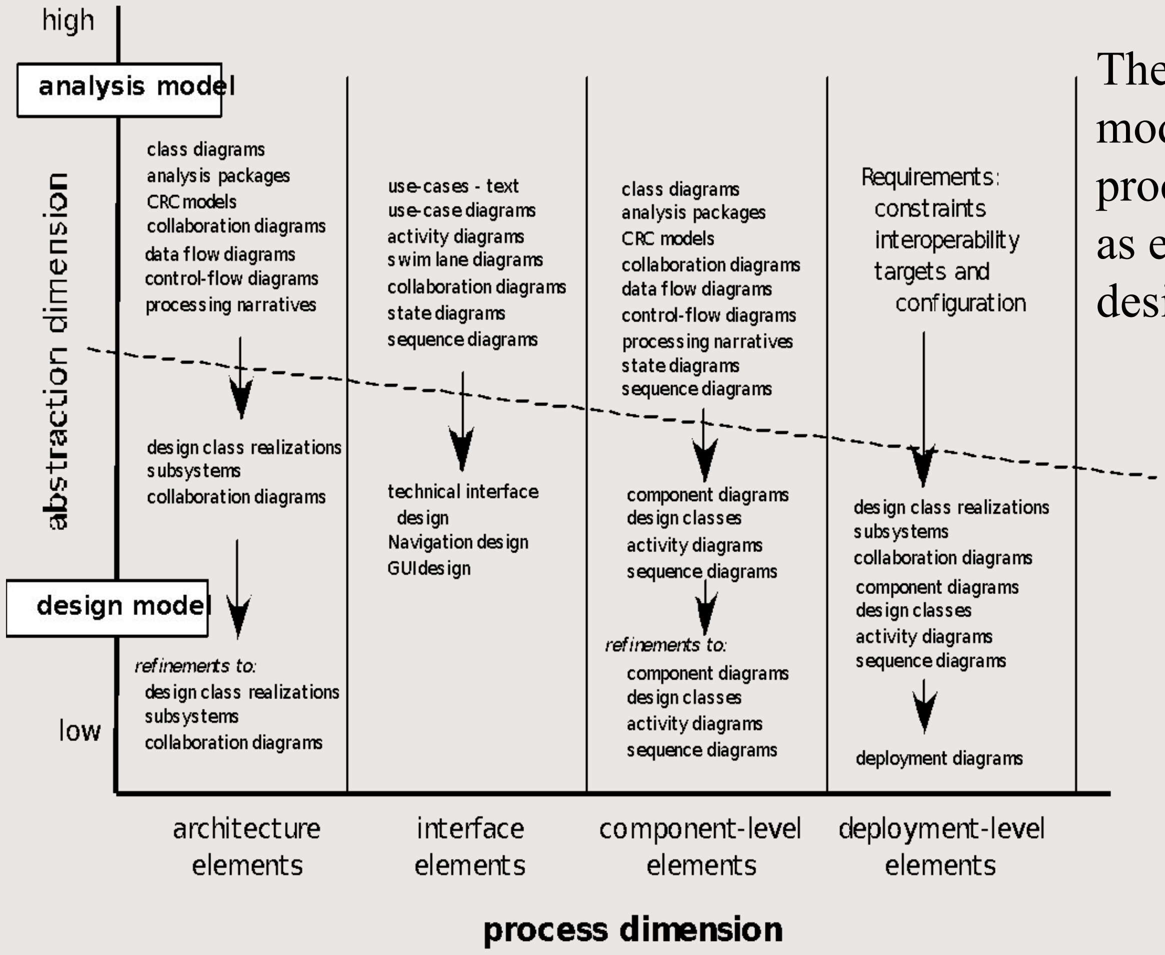
- All the design classes should collaborate with each other in a design model.
- The minimum acceptable of collaboration must be kept in this model.
- If a design model is highly coupled then the system is difficult to implement, to test and to maintain over time.
- .
- This restriction, called the Law of Demeter [Lie03], suggests that a method should only send messages to methods in neighboring classes.

The Design Model

A **design model** in software engineering is an object-based picture or pictures that represent the use cases for a system



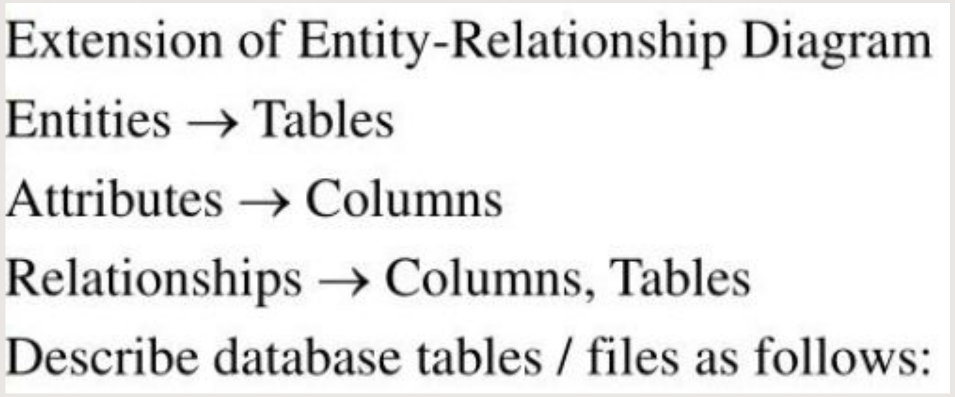
The Design Model



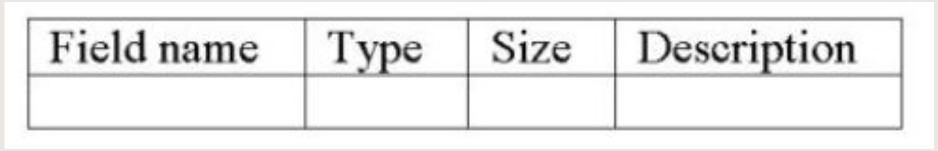
The process dimension indicates the evolution of the design model as design tasks are executed as part of the software process. The abstraction dimension represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively

Design Model Elements

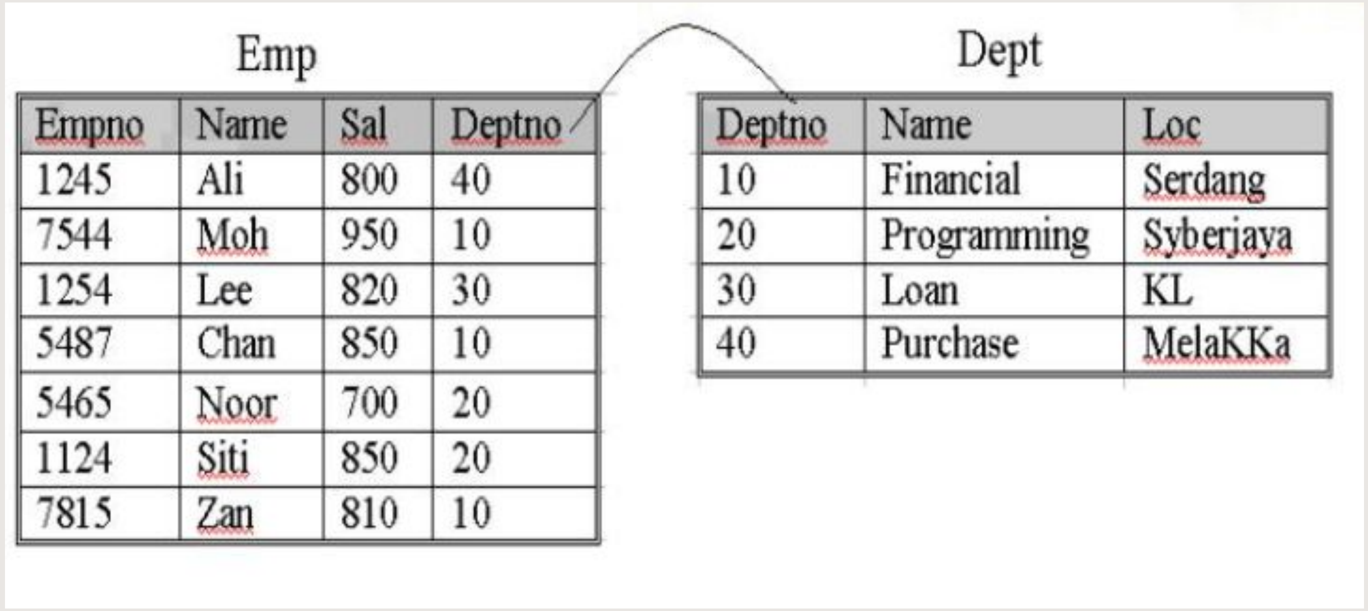
- Select the logical representation of data objects



Extension of Entity-Relationship Diagram
Entities → Tables
Attributes → Columns
Relationships → Columns, Tables
Describe database tables / files as follows:



Field name	Type	Size	Description



Emp

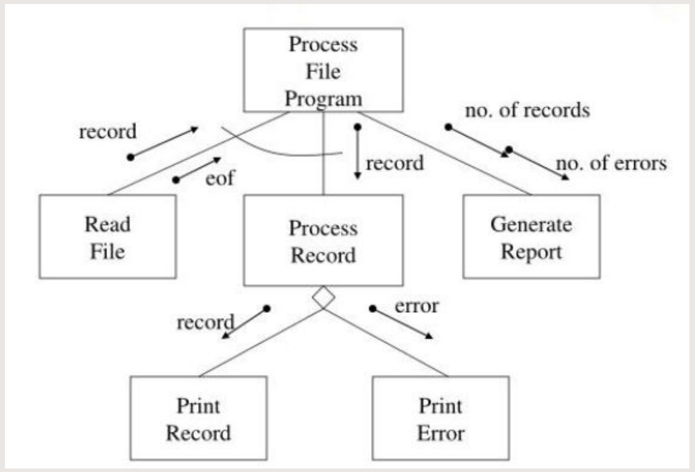
Empno	Name	Sal	Deptno
1245	Ali	800	40
7544	Moh	950	10
1254	Lee	820	30
5487	Chan	850	10
5465	Noor	700	20
1124	Siti	850	20
7815	Zan	810	10

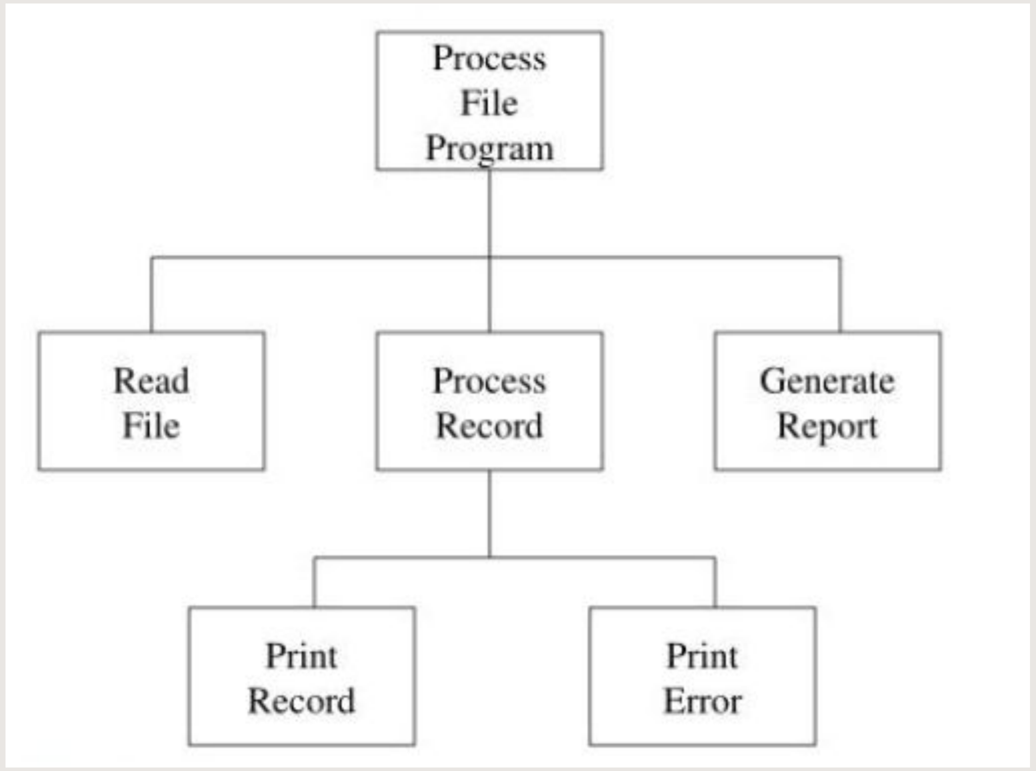
Dept

Deptno	Name	Loc
10	Financial	Serdang
20	Programming	Syberjaya
30	Loan	KL
40	Purchase	MelaKKa

Architectural Design Elements

- The architecture design elements provides us overall view of the system.
- The architectural design element is generally represented as a set of interconnected subsystem that are derived from analysis packages in the requirement model.

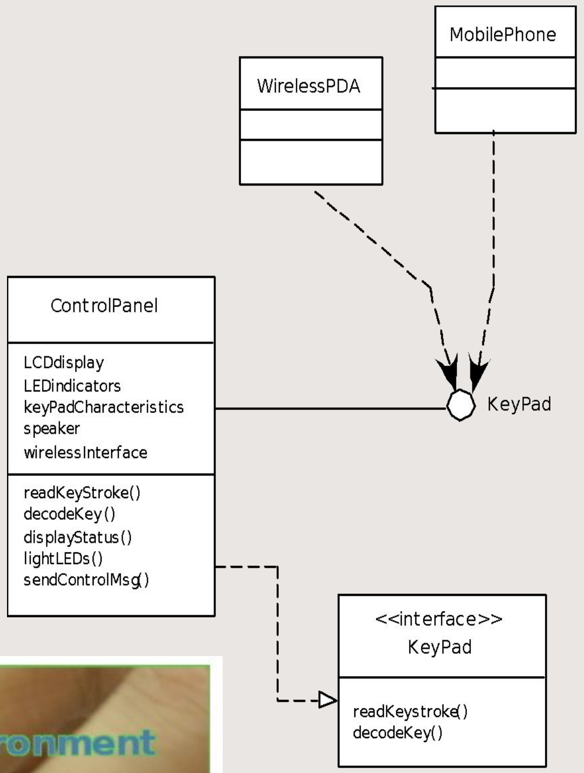




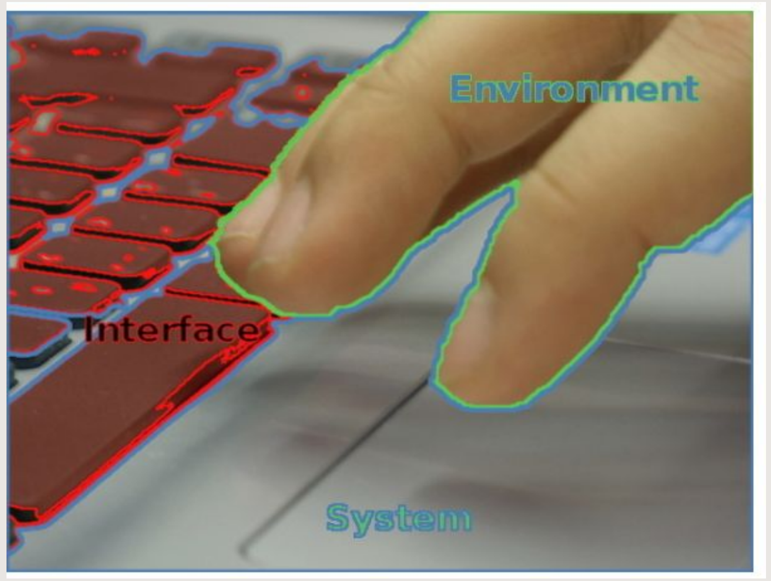
Interface Design Elements

- They communicate between the components defined as part of architecture.

Example: They tell us where the doorbell is located

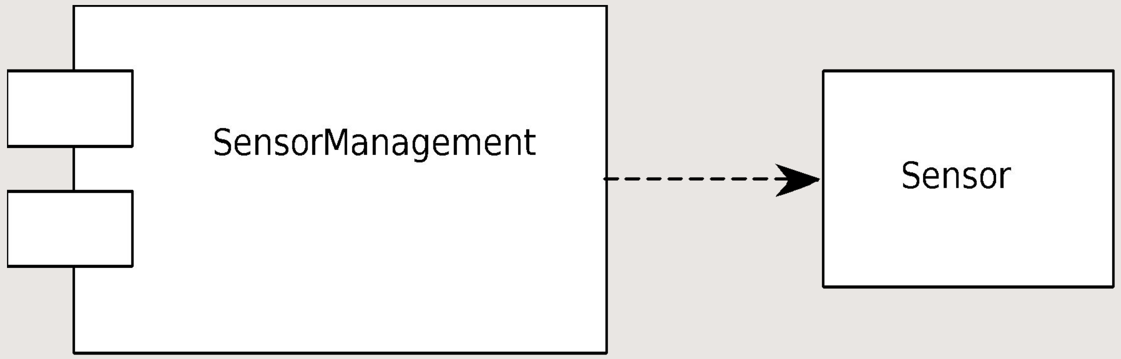


There are three important elements of interface design:
(1) the user interface (UI);
(2) external interfaces to other systems, devices, networks, or other producers or consumers of information;
(3) internal interfaces between various design components.



Component Elements

- The component level design for software is similar to the set of detailed specification of each room in a house.
- The processing of data structure occurs in a component and an interface which allows all the component operations.
- This level of design defines the interface, algorithms, data structure, and communication methods of each component.a



Deployment Elements

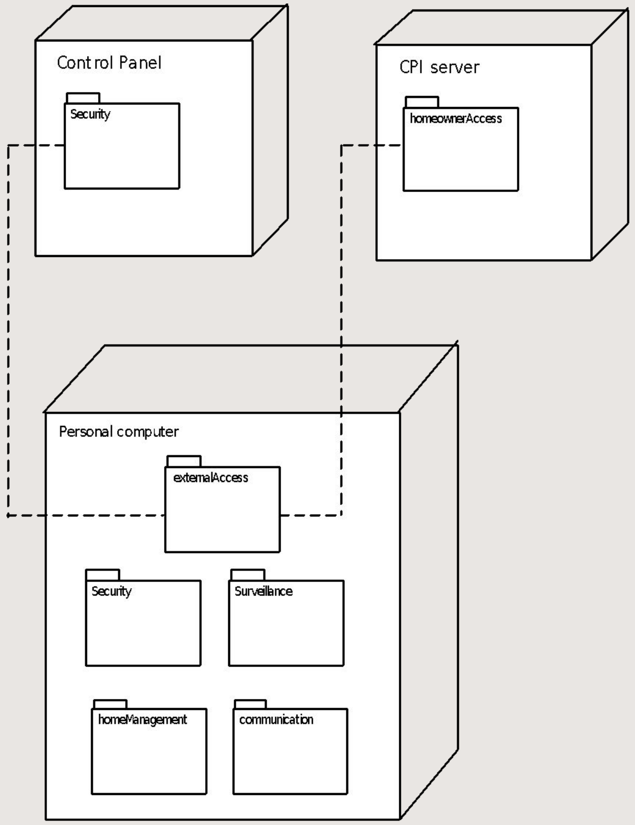
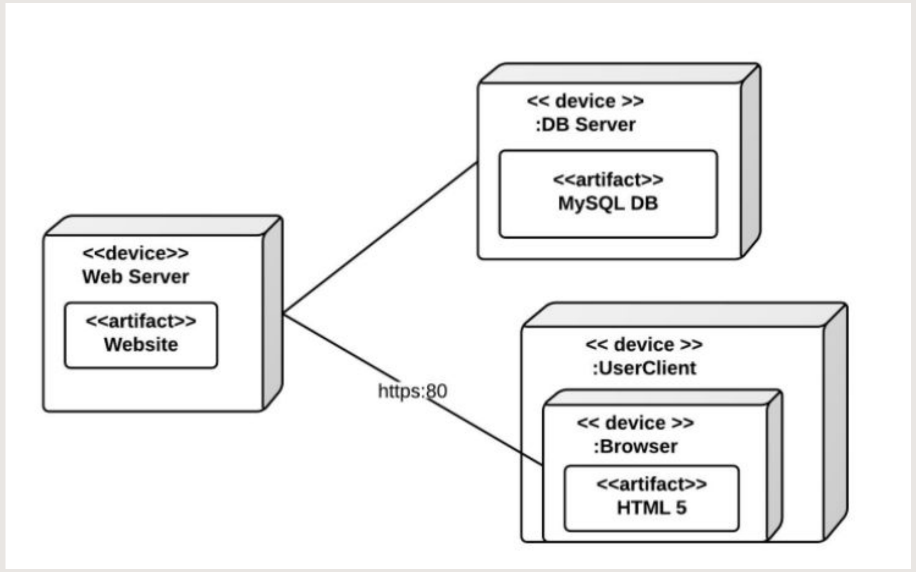


Figure 9.8 UML deployment diagram for *SafeHome*

it describes an aspect of the system itself. In this case, the deployment diagram describes the physical deployment of information generated by the software program on hardware components. The information that the software generates is called an artifact.



Thankyou