



ebook

SOFTWARE ENGINEERING DSE

**DESIGN CONCEPTS ALONG WITH
SOFTWARE DEVELOPMENT**

Software Engineering

Lesson 09 – Design Concepts along with Software Development

Software Design

Software Design sits at the technical kernel of software engineering. It is a problem solving process whose objectives to find and describe a way.

System Analysis determine **WHAT** has to be done and System Design determine **HOW** to do it.

Why Software Design is Important?

- Single word QUALITY.
- Design is the place where quality is fostered in software engineering.
- Design provides you with representations of software that can be accessed for quality.
- Without design, you risk building an unstable system – one that will fail when small changes are made, one that may be difficult to test.

Object Oriented Design Concepts

- **Encapsulation:** That idea of encapsulation is to hide how a class does its business, while allowing other classes to make requests of it.
- **Inheritance:** Allow Classes to inherit commonly used State and Behavior from other Classes.
- **Polymorphism:** It is an ability of an object to take on many forms.
- **Abstraction:** Abstraction is the process of refining away all the unneeded/unimportant attributes of an object and keep only the characteristics best suitable for your domain.
- **User Interface Classes:** Define all abstraction that are necessary for human computer interaction.
- **Business Domain Classes:** The classes identify the attributes and services that are required to implement some element of the business domain.

Software Engineering

- **Process Classes:** Lower level business abstractions required to fully manage the business domain classes.
- **Persistent Classes:** Represent data stores that will persist beyond the execution of the software.
- **System Classes:** Implement software management and control functions that enable the system to operate and communicate within its computing environment and with the outside world.

Design Patterns

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

As per the design pattern reference book **Design Patterns - Elements of Reusable Object-Oriented Software**, there are 23 design patterns which can be classified in three categories:

Creational, Structural and Behavioral patterns

1. Creational Design Patterns

These design patterns are all about class instantiation. These design patterns provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator.

- **Abstract Factory**
Creates an instance of several families of classes
- **Builder**
Separates object construction from its representation
- **Factory Method**
Creates an instance of several derived classes

Software Engineering

- **Object Pool**

Avoid expensive acquisition and release of resources by recycling objects that are no longer in use

- **Prototype**

A fully initialized instance to be copied or cloned

- **Singleton**

A class of which only a single instance can exist

2. Structural Design Patterns

These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.

- **Adapter**

Match interfaces of different classes.

- **Bridge**

Separates an object's interface from its implementation.

- **Composite**

A tree structure of simple and composite objects.

- **Decorator**

Add responsibilities to objects dynamically.

- **Facade**

A single class that represents an entire subsystem.

- **Flyweight**

A fine-grained instance used for efficient sharing.

- **Private Class Data**

Restricts accessor/mutator access

Software Engineering

- **Proxy**
An object representing another object

3. Behavioral Design Patterns

These design patterns are all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

- **Chain of responsibility**
A way of passing a request between a chain of objects.
- **Command**
Encapsulate a command request as an object.
- **Interpreter**
A way to include language elements in a program.
- **Iterator**
Sequentially access the elements of a collection.
- **Mediator**
Defines simplified communication between classes.
- **Memento**
Capture and restore an object's internal state.
- **Null Object**
Designed to act as a default value of an object
- **Observer**
A way of notifying change to a number of classes
- **State**
Alter an object's behavior when its state changes
- **Strategy**
Encapsulates an algorithm inside a class
- **Template method**
Defer the exact steps of an algorithm to a subclass

Software Engineering

- **Visitor**

Defines a new operation to a class without change

Creational Design Patterns – Abstract Factory

- Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
- The purpose of the Abstract Factory is to provide an interface for creating families of related objects, without specifying concrete classes.

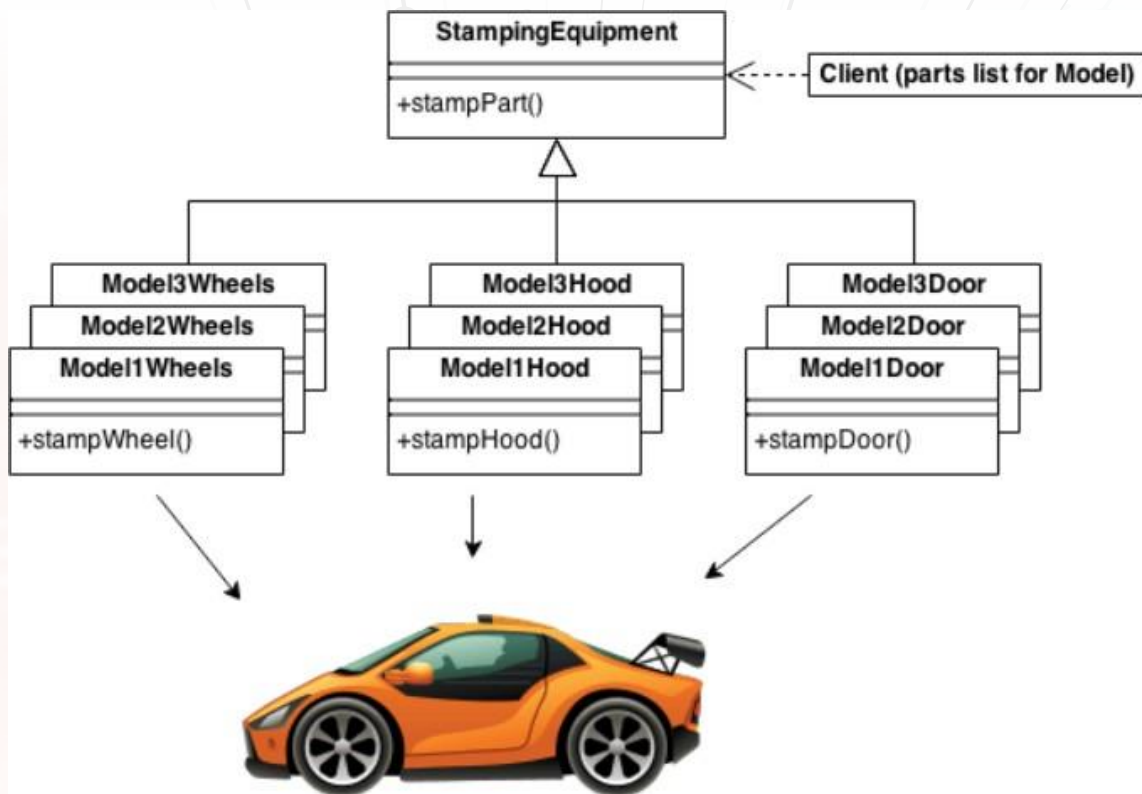


Figure 10.0.1 Abstract Factory

Software Engineering

Creational Design Patterns – Object Pool

- Object pooling can offer a significant performance boost; it is most effective in situations where the cost of initializing a class instance is high, the rate of instantiation of a class is high, and the number of instantiations in use at any one time is low.
- The Object Pool lets others "check out" objects from its pool, when those objects are no longer needed by their processes, they are returned to the pool in order to be reused.
- By keeping reusable instances of objects in a resource pool, and doling them out as needed.

Creational Design Patterns – Singleton

- Ensure a class has only one instance, and provide a global point of access to it.
- Encapsulated "just-in-time initialization" or "initialization on first use".
- A single constructor, that is private and parameter less.

Behavioral Design Patterns – Chain of Responsibility

- Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.
- Launch-and-leave requests with a single processing pipeline that contains many possible handlers.
- An object-oriented Linked List with recursive traversal. Use Pointers concept.

MVC Architecture

Model View Controller: Is a software architecture, which separates the logic from the user interface. This is achieved by separating the application into three parts Model, View and Controller. MVC is separation of concern. MVC is also a design pattern.

- **Model:** Represents the logical behavior of data in the application. It represents applications business logic. Interactions with Database. Model notifies view and controller whenever there is change in state.

Software Engineering

- **View:** Provides the user interface of the application. A view is the one which transforms the state of the model into readable HTML.
- **Controller:** Accepts inputs from the user and instructs the view and model to perform the action accordingly.

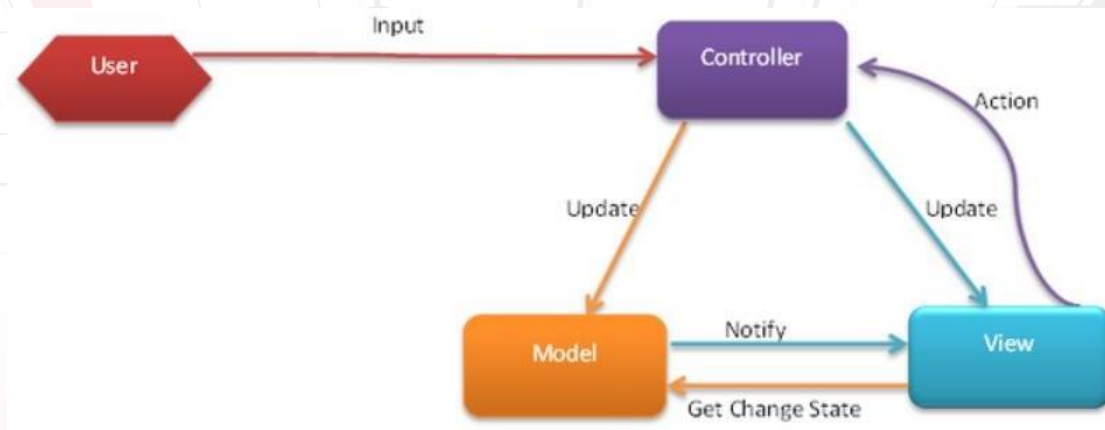


Figure 10.0.2 MVC Architecture

It should also be noted here, that your Controller acts as a bridge between your Model and the View. Because they, as them self, cannot perform any action.

The user must not be allowed to interact with Model himself, instead a Controller must be used to connect to the Model to get the data for the user's View that would be shown to him.

Software Engineering

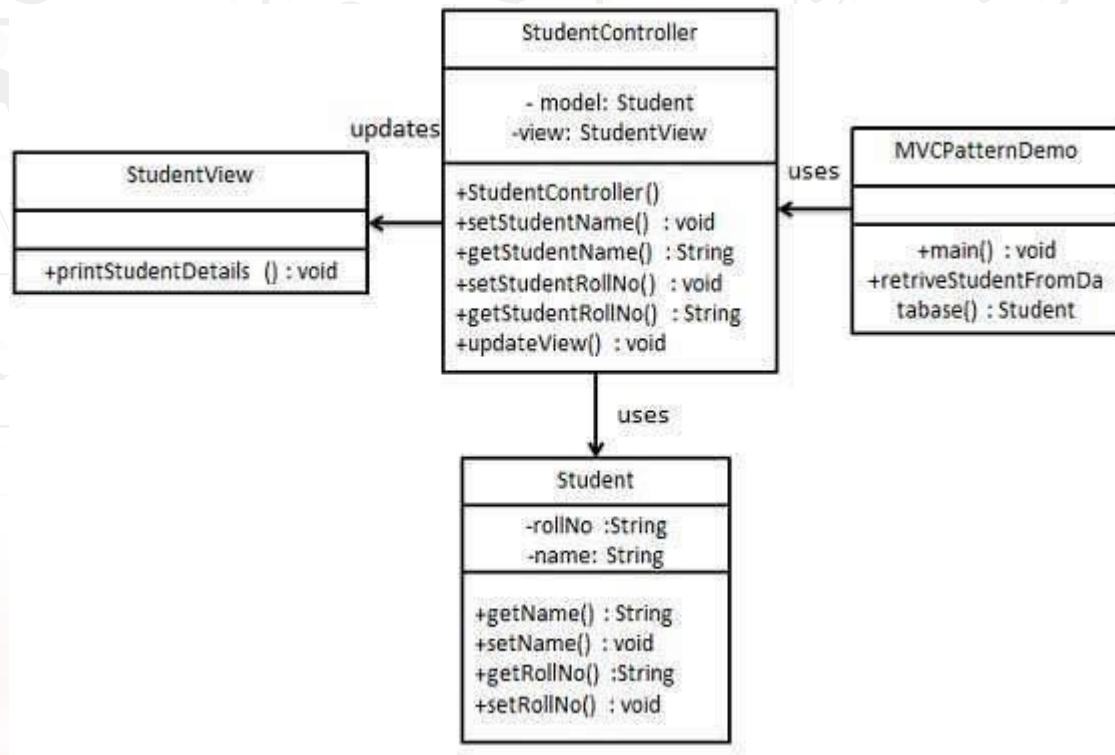


Figure 10.0.3 Model View Controller

MVC specially used in Web Development

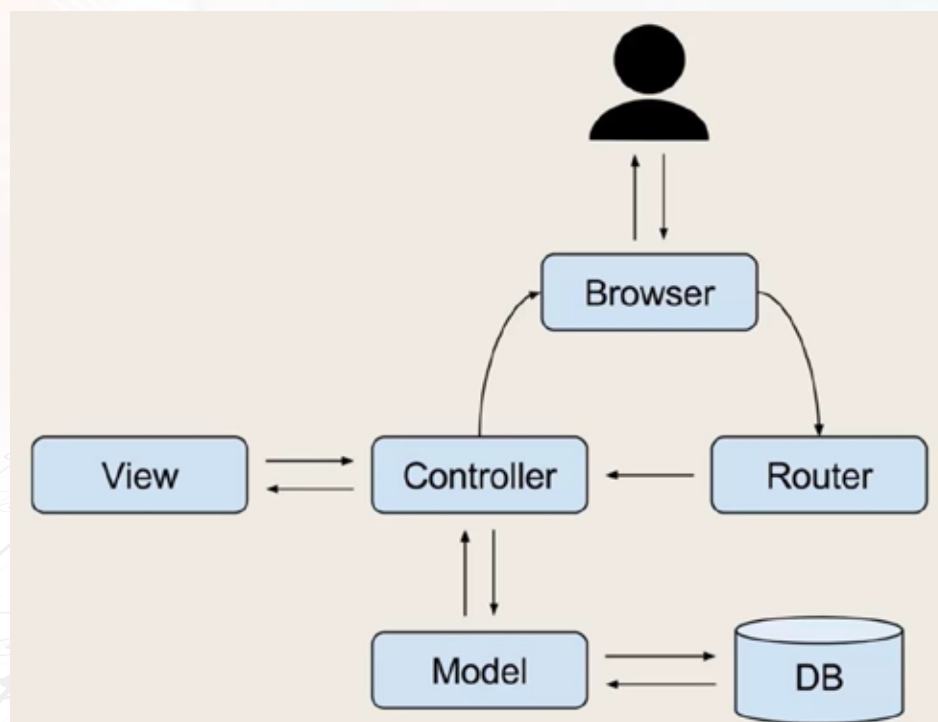


Figure 10.0.4 MVC in web development

Software Engineering

User Interface Design

Creates an effective communication medium between a human and a computer. Following a set of interface design principles, design identifies interface objects and actions and then creates a screen layout that forms the basis for a user interface prototype.

User Interface Design Principles (The Golden Rules)

1. Place the user in control.
2. Reduce the user's memory load.
3. Make the interface consistent.

1) Place the user in control: User wanted a system that reacted to his/her needs and helped his/her get things done.

- **Define interaction modes in a way that does not force a user into unnecessary or undesired actions:** Ex: Spell Check in Word Document.
- **Provide for flexible interaction:** Different users have different interaction preferences. Ex: Keyboard commands, Mouse movements, a multi touch screen.
- **Allow user interaction to be interruptible and undoable:** Ex: Undo any action.
- **Streamline interaction as skill levels advance and allow the interaction to be customized:** Ex: It is worthwhile to design a macro mechanism.
- **Hide Technical internals from the casual user:** The user should not be aware of the operating systems or other computer technology.

2) Reduce the User's memory Load: The more user has to remember, the more error-prone the interaction with the system will be.

- **Reduce demand on short-term memory:** The interface should be designed to reduce the requirement to remember past actions, inputs and results.
- **Establish meaningful defaults:** The initial set of defaults should make sense for the average user.
- **Define shortcuts that are intuitive:** Ex: CTRL+ P to invoke the print function.

Software Engineering

- **Disclose information in a progressive fashion:** Ex: After user pick the underlined function then all underlining options are presented.

3) Make the Interface Consistent: The interface should present and acquire information in a consistent fashion.

- **Allow the user to put the current task into a meaningful context:** It is important is provide the indicators.
- **Maintain consistency across a family of applications:** A set of applications should all implement the same design.
- **If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so:** Ex: CTRL+S for save, the user expects that in every application he encounters.

To build an effective user interface “all design should begin with an understanding of the intended user, including profiles of their age, gender, physical abilities, education motivation, goals and personality”.

Pay attention to what users do.

“Know the User, Know the Tasks”