Design Patterns

- Design patterns are *proven, reusable solutions* to common problems that occur in software design.
- They provide a *standardized way to structure code* to make it more **flexible, maintainable, and scalable**.
- Rather than reinventing the wheel each time a common problem arises, developers can rely on these tried-and-tested approaches.

- The concept of design patterns was popularized by the book *“Design Patterns: Elements of Reusable Object-Oriented Software”* (1994) by the *Gang of Four* (GoF): **Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides**.
- This book outlined **23 classic design patterns**, grouped into three categories:
- Creational Patterns(5) - Concerned with the process of object creation, ensuring flexibility and reuse.
- Structural Patterns(7) - Focused on organizing classes and objects to form larger structures while keeping them efficient and flexible.
- Behavioral Patterns(11) - Deal with communication between objects, ensuring that interactions are efficient and understandable.

Why Design Patterns Matter

- They provide **common vocabulary and understanding** among developers.
- Improve **code readability** and **reduce complexity**.
- Promote **code reuse and robustness**.
- Serve as **best practices** in solving recurring design problems.

- As the name suggests, however, a software design pattern is not code – rather, software design patterns act as a guide or paradigm to help software engineers create products following best practices.
- Design patterns are used to support object-oriented programming (OOP), a paradigm that is based on the concepts of both objects (instances of a class; data with unique attributes) and classes (user-defined types of data).

- Pattern designing requires a combination of skills, including creativity, attention to detail, and technical knowledge. A successful pattern designer must be able to visualize the finished product and translate that vision into a precise and accurate pattern.
- The pattern typically shows relationships and interactions between classes or objects.
- The idea is to speed up the development process by providing well-tested, proven development/design paradigms.
- Design patterns are programming language-independent strategies for solving a common problem.
- That means a design pattern represents an idea, not a particular implementation.

- It’s not mandatory to always implement design patterns in your project.
- Design patterns are not meant for project development. Design patterns are meant for common problem-solving.
- Whenever there is a need, you have to implement a suitable pattern to avoid such problems in the future.

- A design patterns are **well-proved solution** for solving the specific problem/task.
- **Problem Given:**
  Suppose you want to create a class for which only a single instance (or object) should be created and that single object can be used by all other classes.
- **Solution:**
  Singleton design pattern is the best solution of above specific problem. So, every design pattern has **some specification or set of rules** for solving the problems.

- Advantage of design pattern:

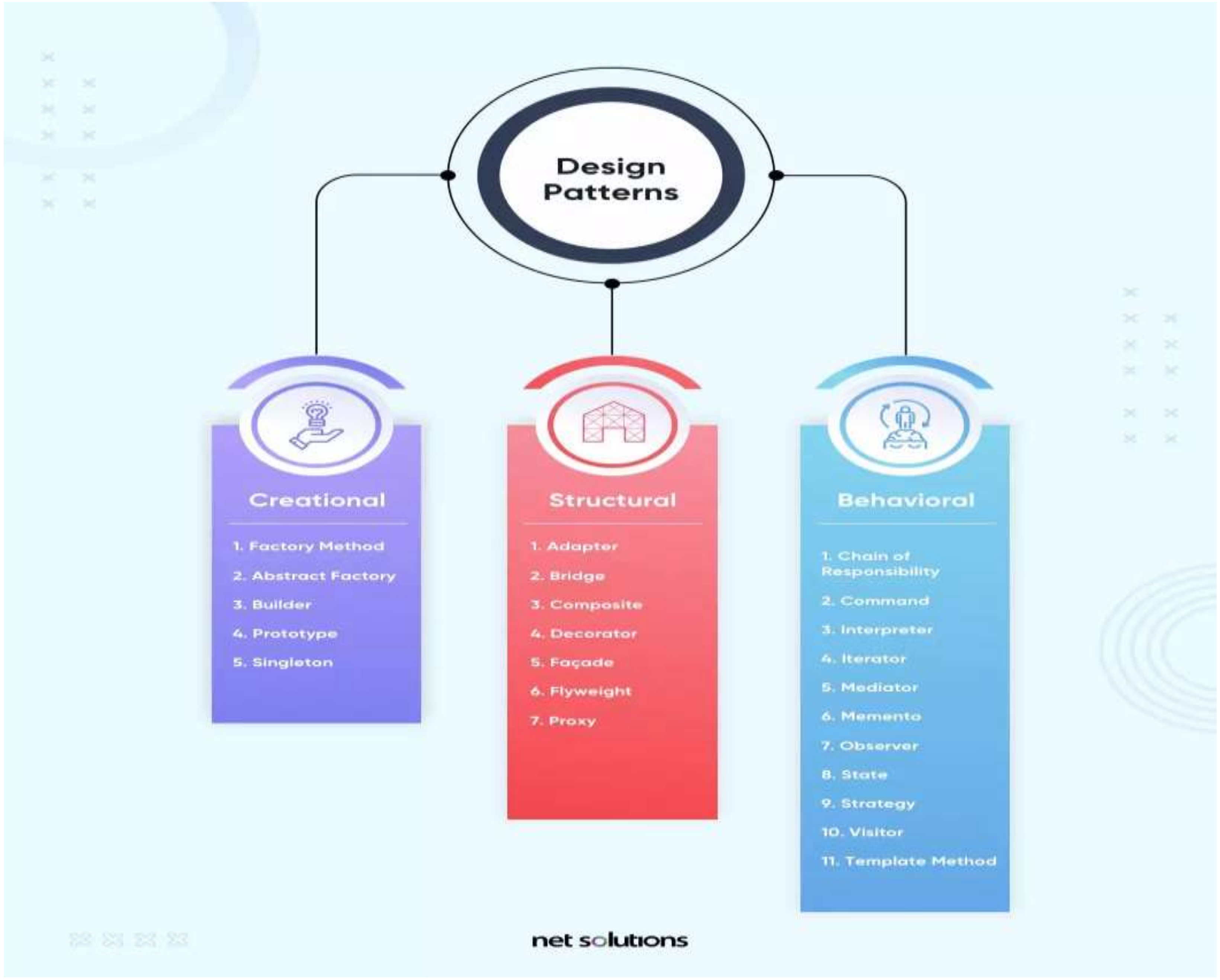
1. They are reusable in multiple projects.
2. They provide the solutions that help to define the system architecture.
3. They capture the software engineering experiences.
4. They provide transparency to the design of an application.
5. They are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.
6. Design patterns dont guarantee an absolute solution to a problem. They provide clarity to the system architecture and the possibility of building a better system.

- **Disadvantages of Pattern Designing:**

1. Cost: Pattern designing can be expensive, particularly if it involves specialized software or equipment.
2. Skill: Pattern designing requires specialized knowledge and skills, which may not be available to everyone.
3. Time-consuming: Creating a pattern can be a time-consuming process, requiring attention to detail and numerous adjustments to ensure a proper fit.
4. Limited creativity: Working within the confines of a pattern can limit the designer's creativity, making it difficult to create unique or innovative designs.
5. Sustainability: The production of patterns, particularly on paper, can contribute to waste and harm to the environment.

- When should we use the design patterns?
- We must use the design patterns **during the analysis and requirement phase of SDLC**(Software Development Life Cycle).
- Design patterns ease the analysis and requirement phase of SDLC by providing information based on prior hands-on experiences.

Types of Design Patterns


Design Patterns
Creational
1. Factory Method
2. Abstract Factory
3. Builder
4. Prototype
5. Singleton
Structural
1. Adapter
2. Bridge
3. Composite
4. Decorator
5. Façade
6. Flyweight
7. Proxy
Behavioral
1. Chain of Responsibility
2. Command
3. Interpreter
4. Iterator
5. Mediator
6. Memento
7. Observer
8. State
9. Strategy
10. Visitor
11. Template Method
net solutions

1. Creational Design Patterns

- A creational design pattern deals with object creation and initialization, providing guidance about which objects are created for a given situation. These design patterns are used to increase flexibility and to reuse existing code.
- **Factory Method**: Creates objects with a common interface and lets a class defer instantiation to subclasses.
- **Abstract Factory**: Creates a family of related objects.
- **Builder**: A step-by-step pattern for creating complex objects, separating construction and representation.
- **Prototype**: Supports the copying of existing objects without code becoming dependent on classes.
- **Singleton**: Restricts object creation for a class to only one instance.

- **1. Creational**
- These design patterns are all about class instantiation or object creation. These patterns can be further categorized into Class-creational patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.
- Creational design patterns are the *Factory Method, Abstract Factory, Builder, Singleton, Object Pool, and Prototype.*

Use case of creational design pattern-

- 1) Suppose a developer wants to create a simple DBConnection class to connect to a database and wants to access the database at multiple locations from code, generally what the developer will do is create an instance of DBConnection class and use it for doing database operations wherever required. This results in creating multiple connections from the database as each instance of DBConnection class will have a separate connection to the database.
- In order to deal with it, we create DBConnection class as a singleton class, so that only one instance of DBConnection is created and a single connection is established. Because we can manage DB Connection via one instance, we can control load balance, unnecessary connections, etc.

2. Structural Design Patterns

- A structural design pattern deals with class and object composition, or how to assemble objects and classes into larger structures.
- **Adapter**: How to change or adapt an interface to that of another existing class to allow incompatible interfaces to work together.
- **Bridge**: A method to decouple an interface from its implementation.
- **Composite**: Leverages a tree structure to support manipulation as one object.
- **Decorator**: Dynamically extends (adds or overrides) functionality.
- **Façade**: Defines a high-level interface to simplify the use of a large body of code.
- **Flyweight**: Minimize memory use by sharing data with similar objects.
- **Proxy**: How to represent an object with another object to enable access control, reduce cost and reduce complexity.

- **2. Structural**
- These design patterns are about organizing different classes and objects to form larger structures and provide new functionality. Structural design patterns are *Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Private Class Data, and Proxy.*
- Use Case Of Structural Design Pattern-
- 1) When 2 interfaces are not compatible with each other and want to establish a relationship between them through an adapter it’s called an adapter design pattern. The adapter pattern converts the interface of a class into another interface or class that the client expects, i.e adapter lets classes work together that could not otherwise because of incompatibility. so in these types of incompatible scenarios, we can go for the adapter pattern.

3. Behavioral Design Patterns

- A behavioral design pattern is concerned with communication between objects and how responsibilities are assigned between objects.
- **Chain of Responsibility**: A method for commands to be delegated to a chain of processing objects.
- **Command**: Encapsulates a command request in an object.
- **Interpreter**: Supports the use of language elements within an application.
- **Iterator**: Supports iterative (sequential) access to collection elements.
- **Mediator**: Articulates simple communication between classes.
- **Memento**: A process to save and restore the internal/original state of an object.
- **Observer**: Defines how to notify objects of changes to other object(s).
- **State**: How to alter the behavior of an object when its stage changes.
- **Strategy**: Encapsulates an algorithm inside a class.
- **Visitor**: Defines a new operation on a class without making changes to the class.
- **Template Method**: Defines the skeleton of an operation while allowing subclasses to refine certain steps.

- **3. Behavioral**
- Behavioral patterns are about identifying common communication patterns between objects and realizing these patterns. Behavioral patterns are *Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Null Object, Observer, State, Strategy, Template method, Visitor*
- Use Case of Behavioral Design Pattern-
- 1) The template pattern defines the skeleton of an algorithm in an operation deferring some steps to sub-classes. The template method lets subclasses redefine certain steps of an algorithm without changing the algorithm structure.
- For example, in your project, you want the behavior of the module to be able to extend, such that we can make the module behave in new and different ways as the requirements of the application change, or to meet the needs of new applications. However, no one is allowed to make source code changes to it, i.e you can add but can't modify the structure in those scenarios a developer can approach template design patterns.

Why Do We Need Design Patterns?

- Design patterns offer a best practice approach to support object-oriented software design, which is easier to design, implement, change, test and reuse. These design patterns provide best practices and structures.
- 1. Proven Solution
- Design patterns provide a proven, reliable solution to a common problem, meaning the software developer does not have to “reinvent the wheel” when that problem occurs.

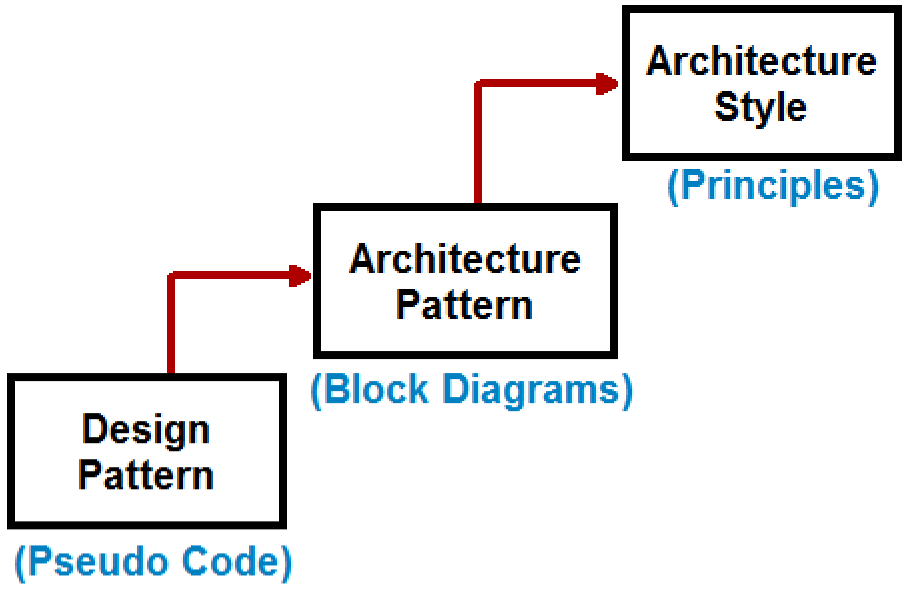
- 2. Reusable
- Design patterns can be modified to solve many kinds of problems – they are not just tied to a single problem.
- 3. Expressive
- Design patterns are an elegant solution.
- 4. Prevent the Need for Refactoring Code
- Since the design pattern is already the optimal solution for the problem, this can avoid refactoring.
- 5. Lower the Size of the Codebase
- Each pattern helps software developers change how the system works without a full redesign. Further, as the “optimal” solution, the design pattern often requires less code.

7 Best Software Design Patterns

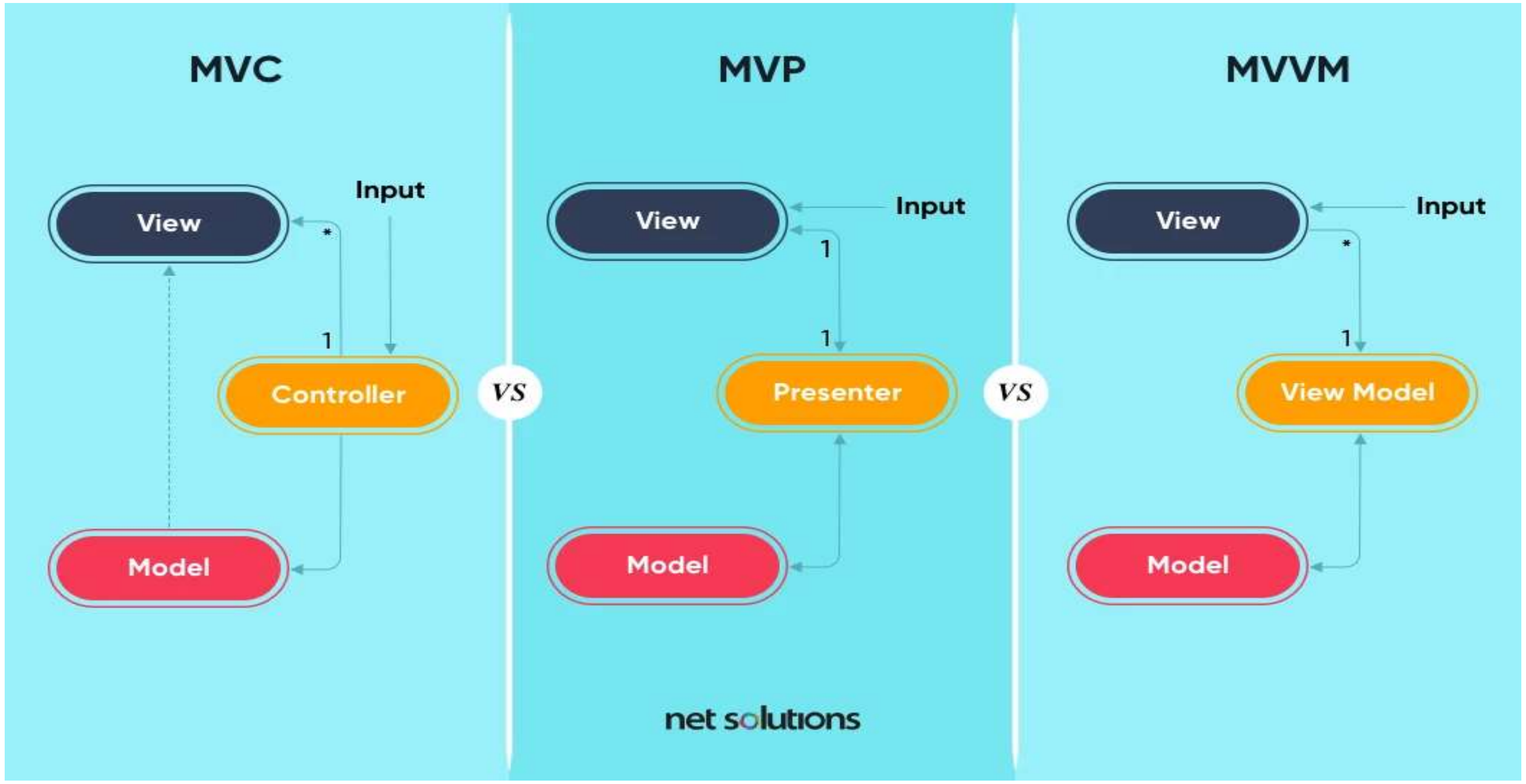
- 1. Singleton Design Pattern
- 2. Factory Method Design Pattern
- 3. Facade Design Pattern
- 4. Strategy Design Pattern
- 5. Observer Design Pattern
- 6. Builder Design Pattern
- 7. Adapter Design Pattern

Architectural pattern vs Design pattern

- Design patterns provide very specific software related tasks where as Architectural pattern are solutions for business problems.
- In other words, Architectural pattern focuses more on the abstract view of idea while Design pattern focuses on the implementation view of idea.


Design Pattern
(Pseudo Code)
Architecture Pattern
(Block Diagrams)
Architecture Style
(Principles)

Popular Software Architectural Patterns



1. MVC Design Pattern

- The model-view-controller (MVC) design pattern is the earliest architectural pattern that is made up of three parts:
- **Model** – the backend business logic and data
- **View** – the interface components to display the data. Leverages the Observer pattern to update with Model and display the updated model when necessary.
- **Controller** – Input is directed here first, processing the request through the model and passing it back to view
- The MVC design pattern is important because it provides separation of concern (SoC), separating the front and backend code into distinct parts to make it easier to update and scale the application without interference or interruption. The MVC model also allows multiple developers to work on different parts of the application at the same time. The risk, however, is that exposing the model to view can introduce security and performance concerns.
- MVC is common for web apps, libraries, and user interfaces.

2. MVP Design Pattern

- The model-view-presenter (MVP) design pattern is derived from MVC but replaces the controller with the presenter and really focuses only on modeling the presentation layer.
- **Model** – the backend business logic and data
- **View** – input begins here and the requested action is presented here
- **Presenter** – One-to-one listening to the views and models, processing the request through the model and passing it back to view
- In this model, the presenter acts as a mediator between the view and the model, supporting a more loosely coupled model. MVP is ideally suited to make views reusable and to support unit testing.
- MVP is commonly used for websites, web apps, and mobile apps (particularly Android).

3. MVVM Design Pattern

- In the model view view-model (MVVM) design pattern, there is two-way data binding between view and view-model (replacing presenter in the MVP design pattern), more cleanly separating the user interface and application logic:
- **Model** – the backend business logic and data
- **View** – input begins here and the requested action is presented here
- **View-Model** – has no reference to view, its only purpose is to maintain the state of view and manipulate the model as the actions of view change
- MVVM allows for view-specific subsets of a model to be created with the state and logic bound to the view, requiring less logic in the code to run the view. MVVM is ideally suited to help improve performance and allow for greater customization and personalization of the view.
- MVVM is also commonly used for mobile apps (growing use in Android) where bi-directional data binding is important.

- In software engineering, there are advantages and disadvantages to using software design patterns.
- Knowing when to use software design patterns – and when not – and how best to implement each pattern comes down to having an experienced team.