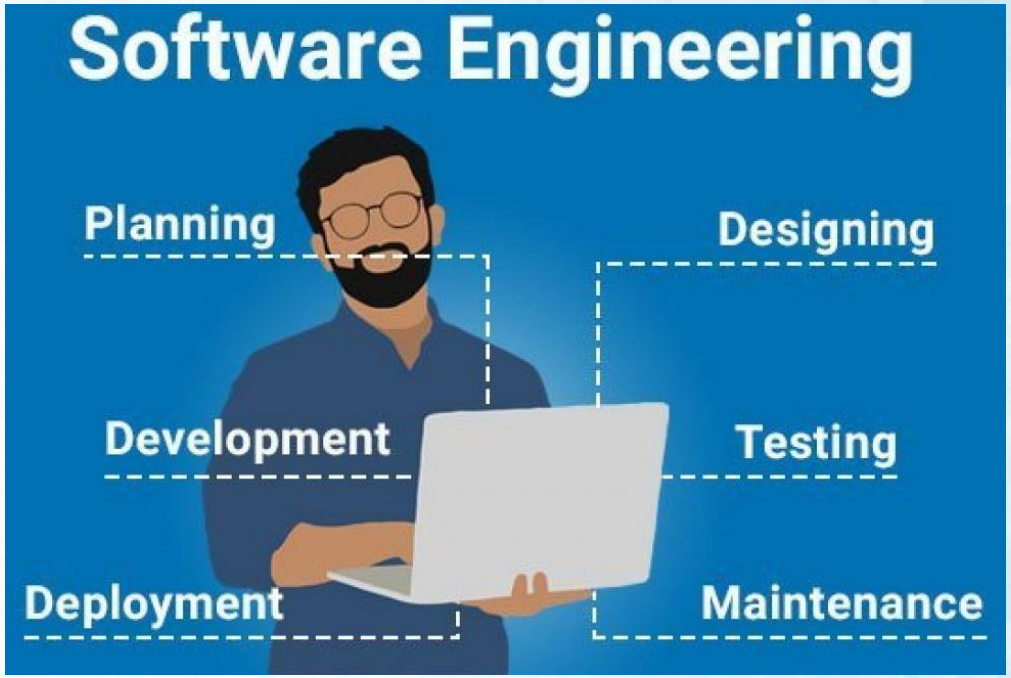
Basic software engineering concept



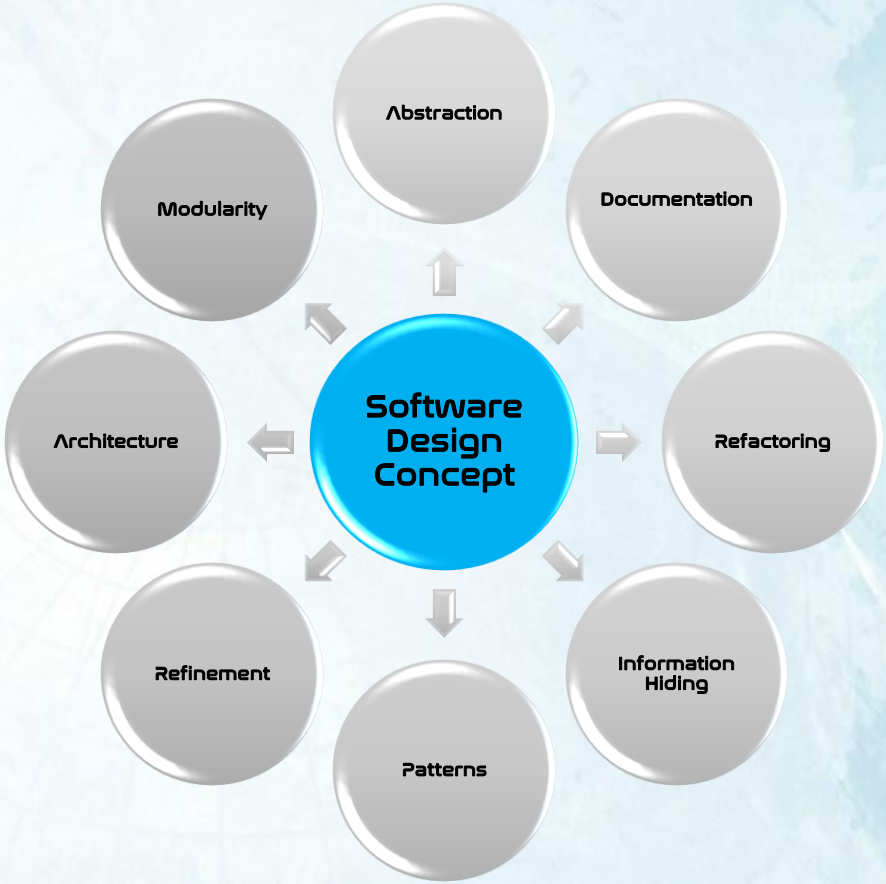
Prepared by—Raymond D. Neoh

Basic software engineering concept

It is important for every potential software engineers to understand the basic concept for designing any pieces of software. Whether they are designing a website, a e-commerce platform, automobile self driving system, or games, robotic, fintech applications, the followings design concept must be implemented in their project design stage and executed accordingly.

The software engineering design concept is important no matter what programming languages or tools are used. The end products reflects the thinking and planning that goes into design such a final product.

SRS (Software Requirements Precertification) documents dictate what the users require the final products will perform and deliver.



Basic software engineering concept

Abstraction- hide Irrelevant data

Abstraction simply means to hide the details to reduce complexity and increases efficiency or quality. Different levels of Abstraction are necessary and must be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solution. The solution should be described in broad ways that cover a wide range of different things at a higher level of abstraction and a more detailed description of a solution of software should be given at the lower level of abstraction.

Basic software engineering concept

Abstraction In OOP

In simple terms, abstraction “displays” only the relevant attributes of objects and “hides” the unnecessary details.

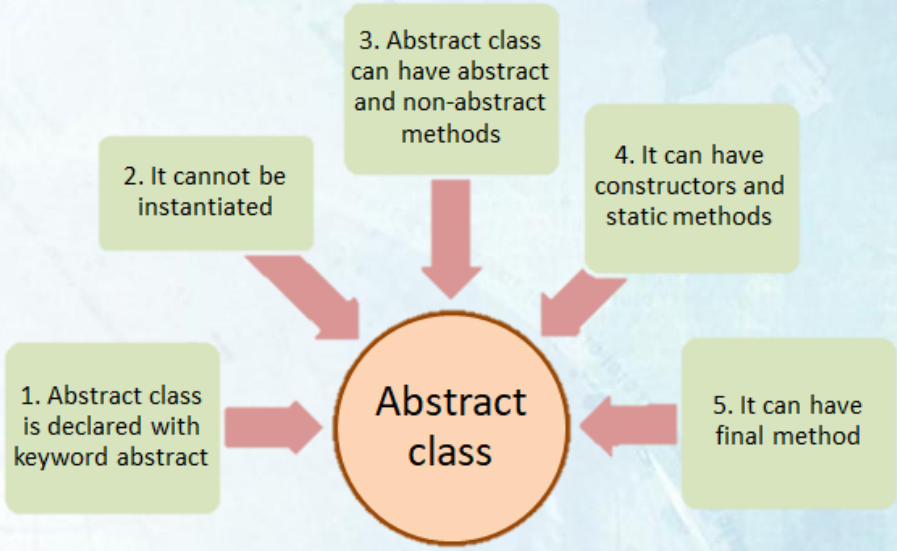
For example, when we are driving a car, we are only concerned about driving the car like start/stop the car, accelerate/ break, etc. We are not concerned about how the actual start/stop mechanism or accelerate/brake process works internally. We are just not interested in those details.

What we are concerned about is the “abstract” view of these operations that will help us to propel the car forward and reach our destination. This is a simple example of abstraction.

Thus the car has all the mechanisms and processes in place but from the end user’s perspective, i.e. car driver’s perspective he/she will be interested only in the abstract view of these processes.

Abstraction reduces the programming efforts and thereby the complexity. An end-user using the application need not be concerned about how a particular feature is implemented. He/she can just use the features as required.

Thus in abstraction, we deal with ideas and not the events. This means that we hide the implementation details from the user and expose only the functionality to the end-user. Thereby the user will only know “what it does” rather than “how it does”.



https://www.softwaretestinghelp.com/what-is-abstraction-in-java/

Basic software engineering concept

Abstraction In OOP

There are basically three types of abstraction
1.Procedural Abstraction
2.Data Abstraction
3.Control Abstraction

Procedural Abstraction: From the word itself, there are a series of procedures in form of functions followed by one after another in sequence to attain abstraction through classes.

Data Abstraction: From the word itself, abstraction is achieved from a set of data that is describing an object.

Control Abstraction: Abstraction is achieved in writing the program in such a way where object details are enclosed.

https://www.softwaretestinghelp.com/what-is-abstraction-in-java/
https://www.geeksforgeeks.org/difference-between-data-hiding-and-abstraction-in-java/

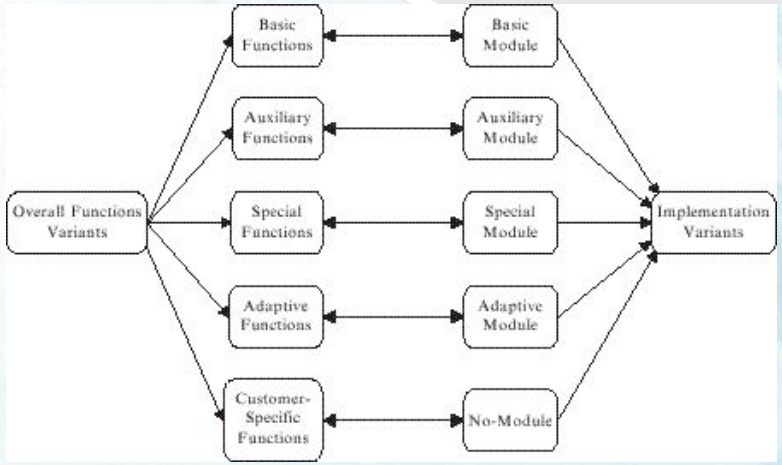
Basic software engineering concept

Modularity- subdivide the system

Modularity simply means dividing the system or project into smaller parts to reduce the complexity of the system or project. In the same way, modularity in design means subdividing a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions.

It is necessary to divide the software into components known as modules because nowadays there are different software available like Monolithic software that is hard to grasp for software engineers. So, modularity in design has now become a trend and is also important. If the system contains fewer components, then it would mean the system is complex which requires a lot of effort (cost) but if we are able to divide the system into components then the cost would be small.

Modularity

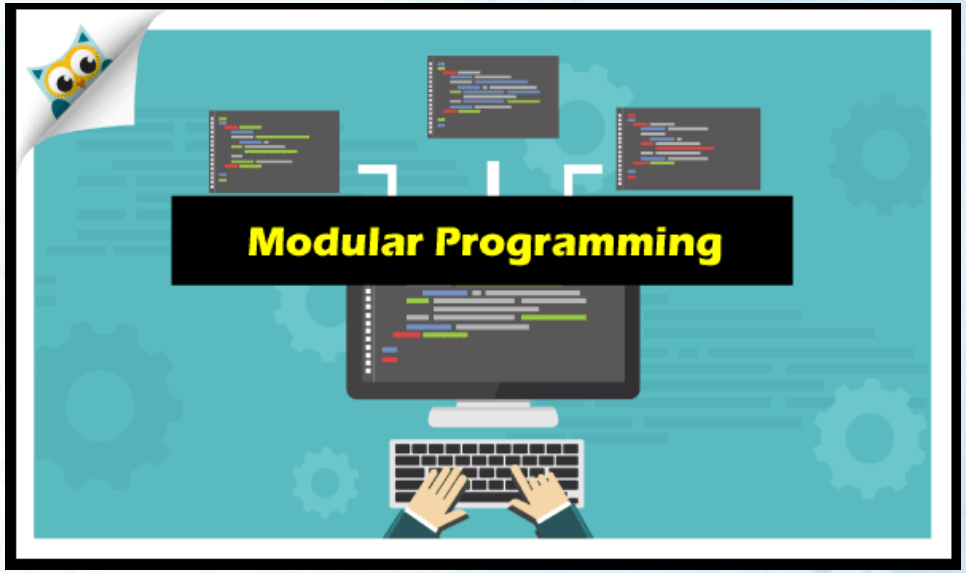


Basic software engineering concept

The concept of modular programming originated in the 1960s to help users. Programmers began to divide the more extensive programs into smaller parts. Though the concept of modular programming is six decades old, it is the most convenient programming method.

All the object-oriented programming languages like C++, Java, etc., are modular programming languages.

For a developer, one must learn to code in modules. There are times when we need to retrieve any code, make a dummy module for testing, and minimize the risk factors. Modular programming is bagged with such features making it essential.



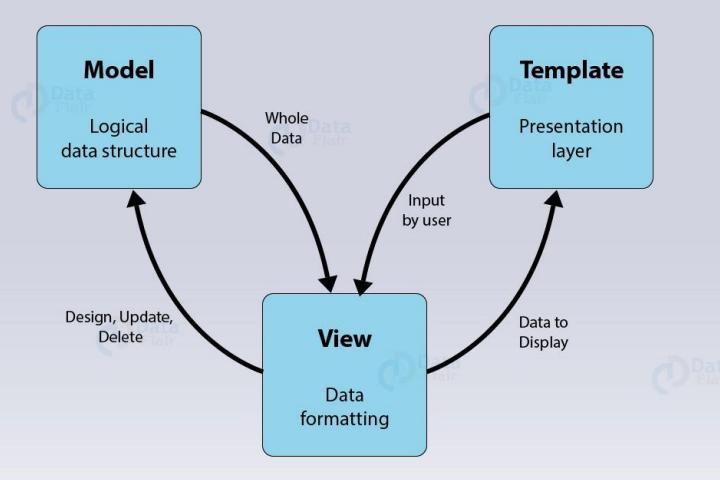
https://www.javatpoint.com/what-is-modular-programming

Basic software engineering concept

Architecture- design a structure of something

Architecture simply means a technique to design a structure of something. Architecture in designing software is a concept that focuses on various elements and the data of the structure. These components interact with each other and use the data of the structure in architecture.

Architecture

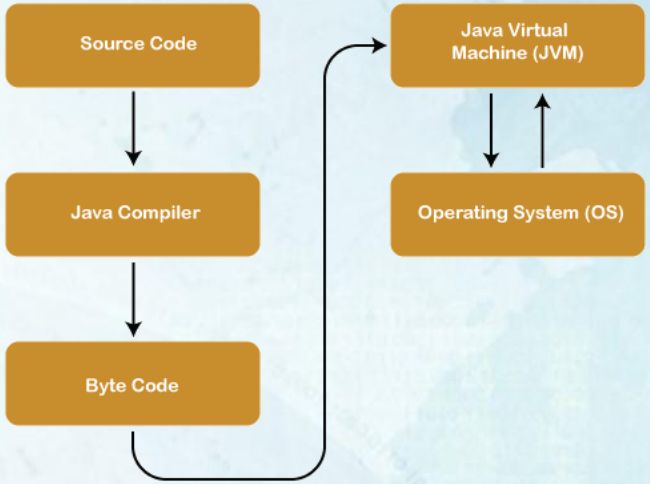


Basic software engineering concept

Java Architecture

The Java architecture includes the three main components:

- Java Virtual Machine (JVM)
- Java Runtime Environment (JRE)
- Java Development Kit (JDK)



Java Virtual Machine

The main feature of Java is **WORA**. WORA stands for **Write Once Run Anywhere**. The feature states that we can write our code once and use it anywhere or on any operating system. Our Java program can run any of the platforms only because of the Java Virtual Machine. It is a Java platform component that gives us an environment to execute java programs. JVM's main task is to convert byte code into machine code.

Java Runtime Environment

It provides an environment in which Java programs are executed. JRE takes our Java code, integrates it with the required libraries, and then starts the JVM to execute it.

Java Runtime Environment

It provides an environment in which Java programs are executed. JRE takes our Java code, integrates it with the required libraries, and then starts the JVM to execute it.

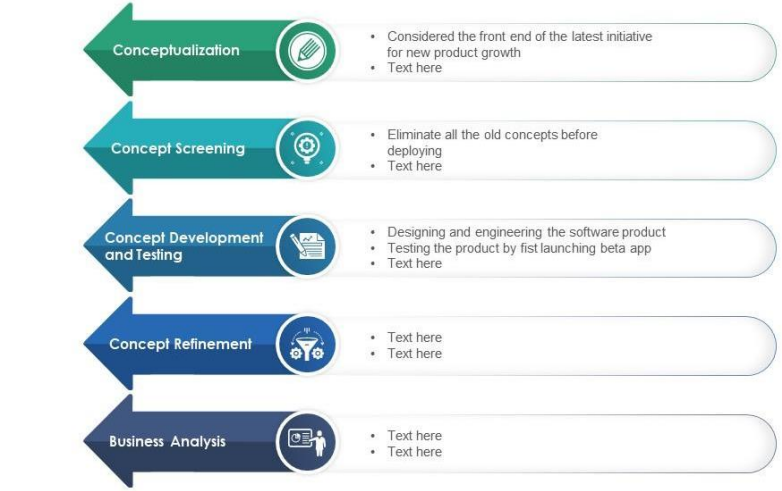
https://www.javatpoint.com/java-architecture

Basic software engineering concept

Refinement- removes impurities

Refinement simply means to refine something to remove any impurities if present and increase the quality. The refinement concept of software design is a process of developing or presenting the software or system in a detailed manner that means to elaborate a system or software. Refinement is very necessary to find out any error if present and then to reduce it.

Refinement



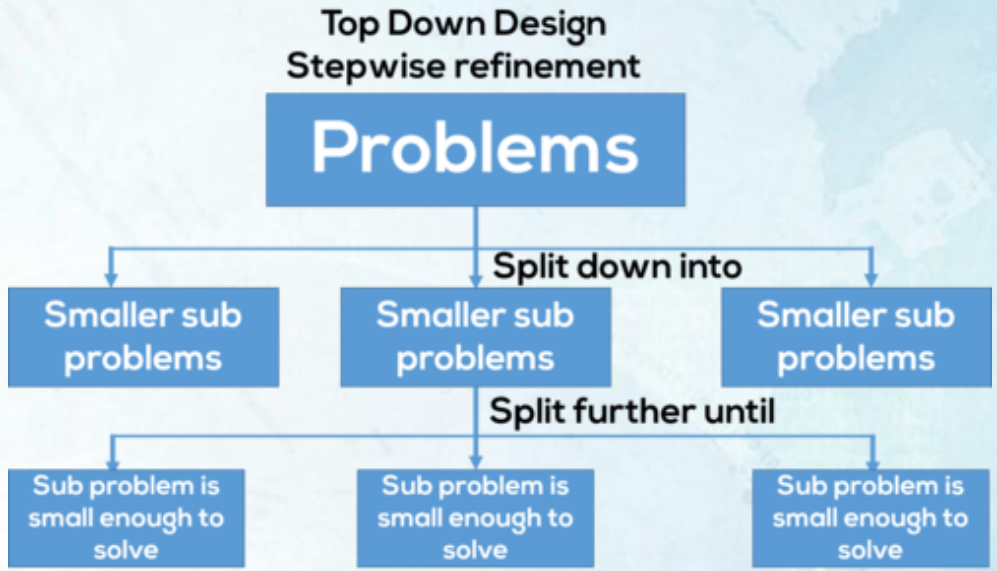
Basic software engineering concept

Stepwise refinement

Stepwise refinement refers to the progressive refinement in small steps of a program specification into a program. Sometimes, it is called top-down design.

The term *stepwise refinement* was used first in the paper titled *Program Development by Stepwise Refinement* by Niklaus Wirth, the author of the programming language Pascal and other major contributions to software design and software engineering.

Wirth said, "It is here considered as a sequence of design decisions concerning the decomposition of tasks into subtasks and of data into data structures."



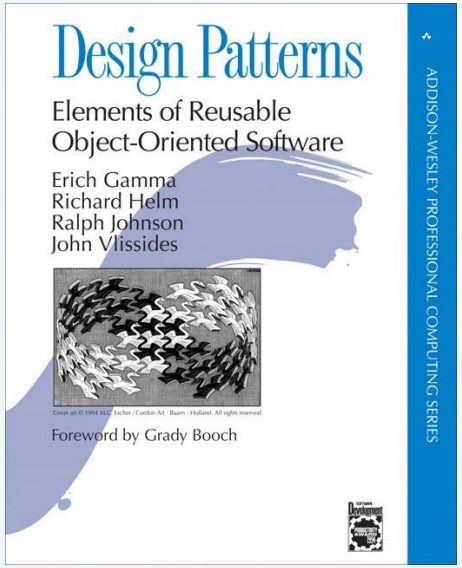
https://www.cs.cornell.edu/courses/JavaAndDS/stepwise/stepwise.html

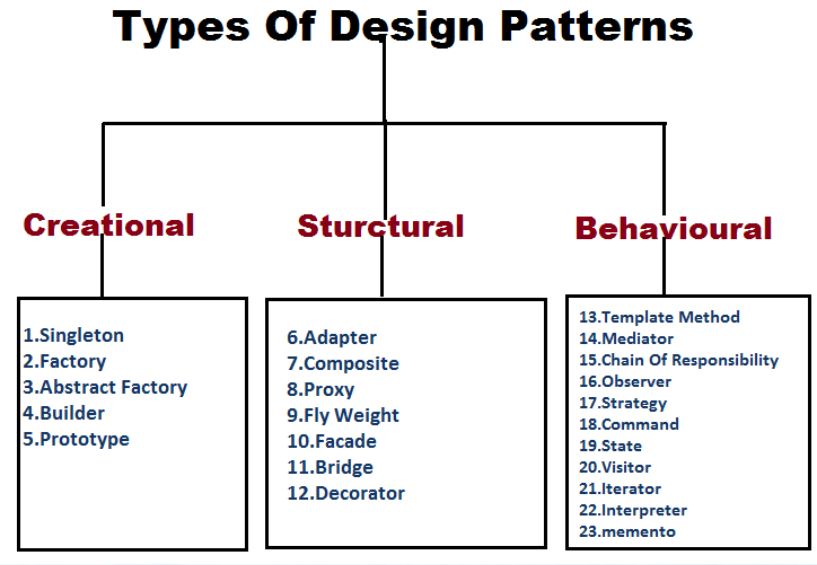
Basic software engineering concept

Pattern- a repeated form

The pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.

Pattern





Basic software engineering concept

23 Design Patterns in Software Development

There are 23 classic design patterns, although there are at least 26 design patterns discovered to date. These design patterns gained popularity after the publication of Design Patterns: Elements of Reusable Object-Oriented Software, a 1994 book published by the “Gang of Four” (GoF): Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Due to this, the 23 design patterns are often known as the gang of four design patterns.

Design patterns can be broken down into **three types**, organized by their intent into creational design patterns, structural design patterns, and behavioral design patterns.

https://www.netsolutions.com/insights/software-design-pattern/

Basic software engineering concept

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.

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.

Basic software engineering concept

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.

https://www.netsolutions.com/insights/software-design-pattern/

Basic software engineering concept

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.

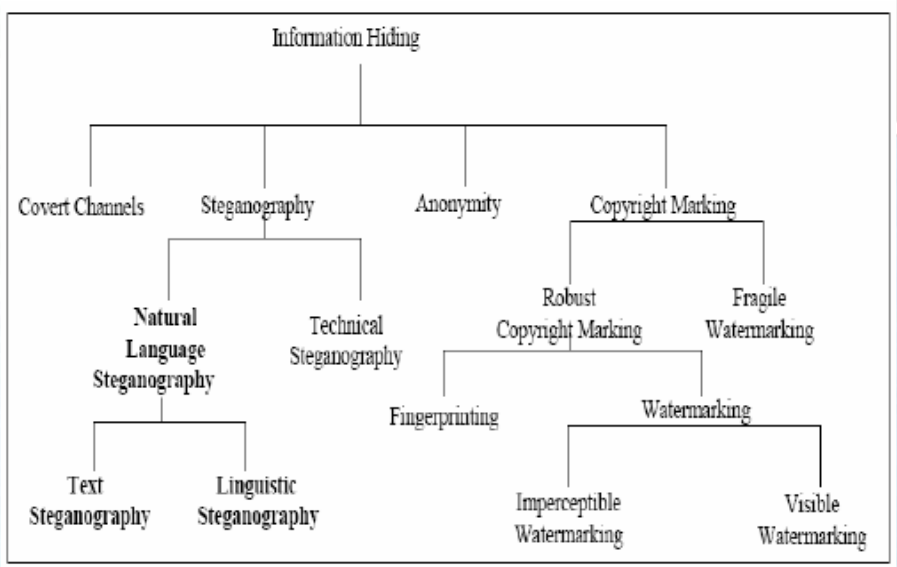
https://www.netsolutions.com/insights/software-design-pattern/

Basic software engineering concept

Information Hiding- hide the information

Information hiding simply means to hide the information so that it cannot be accessed by an unwanted party. In software design, information hiding is achieved by designing the modules in a manner that the information gathered or contained in one module is hidden and can't be accessed by any other modules.

Information Hiding



Basic software engineering concept

Data Hiding is hiding internal data from outside users. The internal data should not go directly that is outside person/classes is not able to access internal data directly. It is achieved by using an access specifier- a private modifier.

Concept involved in data Hiding: Getter and setter

```
class Account {
private double account_balance;
.....
.....
}
```

Here account balance of each say employee is private to each other being working in the same organization. No body knows account balance of anybody. In java it is achieved by using a keyword 'private' keyword and the process is called data hiding.

https://www.geeksforgeeks.org/difference-between-data-hiding-and-abstraction-in-java/

Basic software engineering concept

// Java Program showing working of data hiding

```
// Importing generic libraries
import java.io.*;

// Class created named Bank
class Bank {

    // Private data (data hiding)
    private long CurBalance = 0;

    // Bank_id is checked for
authentication
    long bank_id;
    String name;

    // Getter function to modify
private data
    public long get_balance(long Id)
    {

        // Checking whether the user is
        // authorised or unauthorised

        // Comparing bank_id of user and
the give Id
        // then only it will get access
        if (this.bank_id == Id) {

            // Return current balance
            return CurBalance;
        }

        // Unauthorised user
        return -1;
    }
    // Setter function
    public void set_balance(long balance,
long Id)
    {
        // Comparing bank_id of user and
the give Id
        // then only it will get access
        if (this.bank_id == Id) {
            // Update balance in current ID
            CurBalance = CurBalance +
balance;
        }
    }
}

// Another class created- Employee
public class Emp {
    public static void main(String[] args)
    {
        // Creating employee object of
bank type
        Bank _emp = new Bank();

        // Assigning employee object values
        _emp.bank_id = 12345;
        _emp.name = "Roshan";

        // _emp.get_balance(123456)
        _emp.set_balance(10000, 12345);
        // This will no get access as bank_id
is given wrong
        // so
        // unauthorised user is not getting
access that is
        // data hiding

        long emp_balance =
_emp.get_balance(12345);
        // As this time it is valid user it will
get access

        // Display commands
        System.out.println("User Name"
                + "  " + _emp.name);
        System.out.println("Bank_ID"
                + "  " + _emp.bank_id);
        System.out.println("Current
Balance"
                + "  " + emp_balance);
    }
```

Output:

User Name Roshan
Bank_ID 12345
Current Balance 10000

https://www.geeksforgeeks.org/difference-between-data-hiding-and-abstraction-in-java/

Basic software engineering concept

Refactoring- reconstruct something

Refactoring simply means reconstructing something in such a way that it does not affect the behavior of any other features. Refactoring in software design means reconstructing the design to reduce complexity and simplify it without affecting the behavior or its functions.
Fowler has defined refactoring as “the process of changing a software system in a way that it won’t affect the behavior of the design and improves the internal structure”.

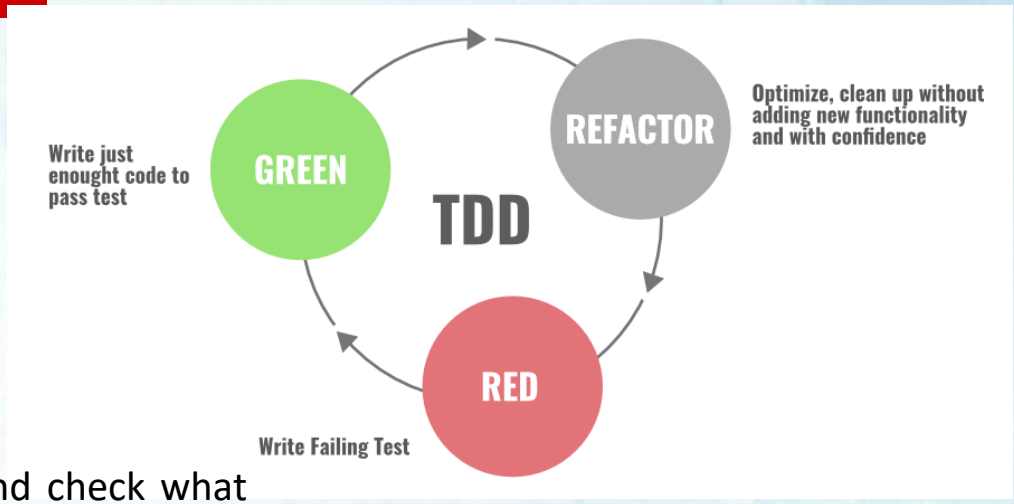
Refactoring

We will discuss some popular and common techniques to refactor the codes, such as:

•to perform code refactoring in small steps. Make tiny changes in your program, each of the small changes makes your code slightly better and leaves the application in a working state.
•Run the test TDD and CI after making small changes in the refactoring process. Without running these tests, you create a risk of introducing bugs.
•Do not create any new features or functionality during the refactoring process. You should refactor the code before adding any updates or new features in your existing code.
•Refactoring process can affect the testing outcomes so it’s good to get your QA and testing team involved in the refactoring process.
•You need to accept that you won’t be fully satisfied with your code. Your refactored code will be outdated in near future and you’ll have to refactor it again.

Most Common Code Refactoring Techniques

Basic software engineering concept



1. Red-Green Refactoring

Red-Green is the most popular and widely used code refactoring technique in the Agile software development process. This technique follows the "test-first" approach to design and implementation, this lays the foundation for all forms of refactoring. Developers take initiative for the refactoring into the test-driven development cycle and it is performed into the three district steps.

- **RED:** The first step starts with writing the failing "red-test". You stop and check what needs to be developed.
- **Green:** In the second step, you write the simplest enough code and get the development pass "green" testing.
- **Refactor:** In the final and third steps, you focus on improving and enhancing your code keeping your test green.

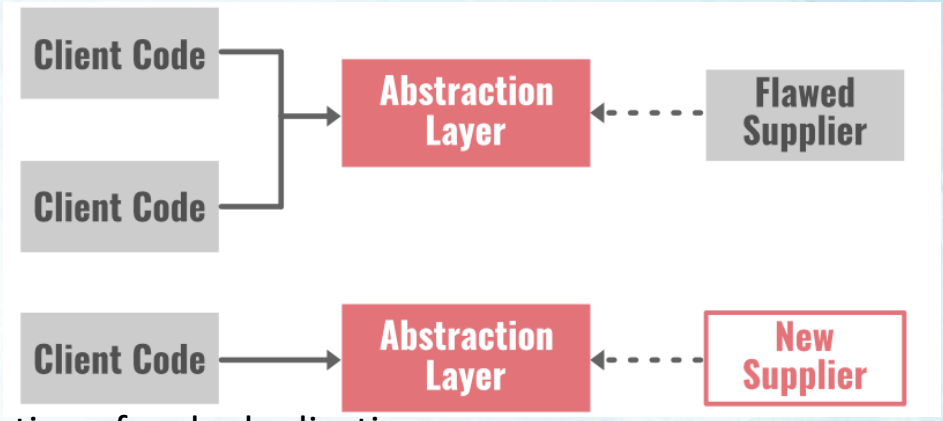
So basically, this technique has two distinct parts: The first part involves writing code that adds a new function to your system and the second part is all about refactoring the code that does this function. Keep in mind that you're not supposed to do both at the same time during the workflow.

https://www.geeksforgeeks.org/7-code-refactoring-techniques-in-software-engineering/

Basic software engineering concept

2. Refactoring By Abstraction

This technique is mostly used by developers when there is a need to do a large amount of refactoring. Mainly we use this technique to reduce the redundancy (duplication) in our code. This involves class inheritances, hierarchy, creating new classes and interfaces, extraction, replacing inheritance with the delegation, vice versa.



Pull-Up/Push-Down method is the best example of this approach.

- **Pull-Up method:** It pulls code parts into a superclass and helps in the elimination of code duplication.
- **Push-Down method:** It takes the code part from a superclass and moves it down into the subclasses.

Pull up the constructor body, extract subclass, extract superclass, collapse hierarchy, form template method, extract interface, replace inheritance with the delegation, replace delegation with Inheritance, push down-field all these are the other examples.

Basically, in this technique, we build the abstraction layer for those parts of the system that needs to be refactored and the counterpart that is eventually going to replace it.

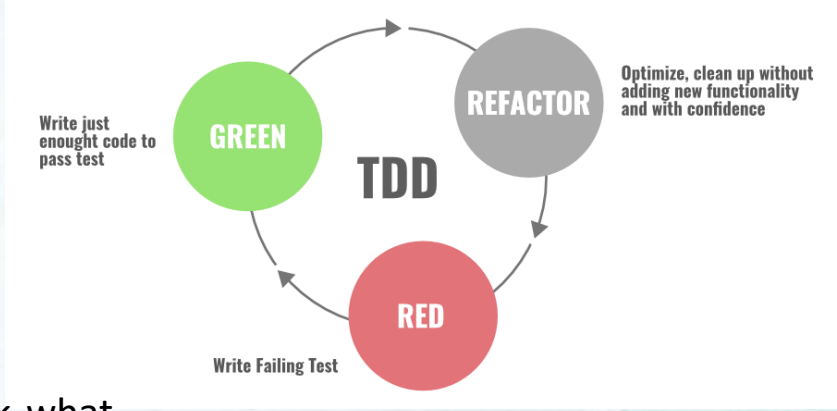
Two common examples are given below…

- **Encapsulated field:** We force the code to access the field with getter and setter methods.
- **Generalize type:** We create more general types to allow code sharing, replace type-checking code with the state, replace conditional with polymorphism, etc.

Basic software engineering concept

1. Red-Green Refactoring

Red-Green is the most popular and widely used code refactoring technique in the Agile software development process. This technique follows the “test-first” approach to design and implementation, this lays the foundation for all forms of refactoring. Developers take initiative for the refactoring into the test-driven development cycle and it is performed into the three district steps.



- **RED:** The first step starts with writing the failing “red-test”. You stop and check what needs to be developed.
- **Green:** In the second step, you write the simplest enough code and get the development pass “green” testing.
- **Refactor:** In the final and third steps, you focus on improving and enhancing your code keeping your test green.

So basically, this technique has two distinct parts: The first part involves writing code that adds a new function to your system and the second part is all about refactoring the code that does this function. Keep in mind that you’re not supposed to do both at the same time during the workflow.

https://www.geeksforgeeks.org/7-code-refactoring-techniques-in-software-engineering/

Basic software engineering concept

3. Composing Method

During the development phase of an application a lot of times we write long methods in our program. These long methods make your code extremely hard to understand and hard to change. The composing method is mostly used in these cases.

In this approach, we use streamline methods to reduce duplication in our code. Some examples are: extract method, extract a variable, inline Temp, replace Temp with Query, inline method, split temporary variable, remove assignments to parameters, etc.

Extraction: We break the code into smaller chunks to find and extract fragmentation. After that, we create separate methods for these chunks, and then it is replaced with a call to this new method. Extraction involves class, interface, and local variables.

Inline: This approach removes the number of unnecessary methods in our program. We find all calls to the methods, and then we replace all of them with the content of the method. After that, we delete the method from our program.

https://www.geeksforgeeks.org/7-code-refactoring-techniques-in-software-engineering/

Basic software engineering concept

4. Simplifying Methods

There are two techniques involved in this approach…let's discuss both of them.

•**Simplifying Conditional Expressions Refactoring:** Conditional statement in programming becomes more logical and complicated over time. You need to simplify the logic in your code to understand the whole program. There are so many ways to refactor the code and simplify the logic.

Some of them are: consolidate conditional expression and duplicate conditional fragments, decompose conditional, replace conditional with polymorphism, remove control flag, replace nested conditional with guard clauses, etc.

•**Simplifying Method Calls Refactoring:** In this approach, we make method calls simpler and easier to understand. We work on the interaction between classes, and we simplify the interfaces for them.

Examples are: adding, removing, and introducing new parameters, replacing the parameter with the explicit method and method call, parameterize method, making a separate query from modifier, preserve the whole object, remove setting method, etc.

https://www.geeksforgeeks.org/7-code-refactoring-techniques-in-software-engineering/

Basic software engineering concept

5. Moving Features Between Objects

In this technique, we create new classes, and we move the functionality safely between old and new classes. We hide the implementation details from public access.

Now the question is… when to move the functionality between classes or how to identify that it's time to move the features between classes?

When you find that a class has so many responsibilities and too much thing is going on or when you find that a class is unnecessary and doing nothing in an application, you can move the code from this class to another class and delete it altogether.

Examples are: move a field, extract class, move method, inline class, hide delegate, introduce a foreign method, remove middle man, introduce local extension, etc.

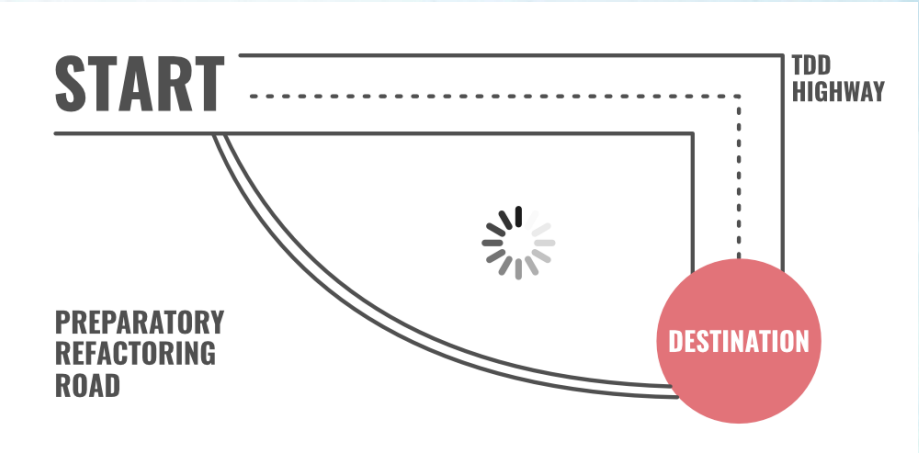
https://www.geeksforgeeks.org/7-code-refactoring-techniques-in-software-engineering/

Basic software engineering concept

6. Preparatory Refactoring

This approach is best to use when you notice the need for refactoring while adding some new features in an application. So basically it's a part of a software update with a separate refactoring process. You save yourself with future technical debt if you notice that the code needs to be updated during the earlier phases of feature development.

The end-user can not see such efforts of the engineering team eye to eye but the developers working on the application will find the value of refactoring the code when they are building the application. They can save their time, money, and other resources if they just spend some time updating the code earlier.



https://www.geeksforgeeks.org/7-code-refactoring-techniques-in-software-engineering/

Basic software engineering concept

Documentation—ensure what you write other understand

One of the greatest draw back for today software engineers are the only want to write codes but negated to document what they have written. As a result, people that follow up on the codes must guess or ask the engineers about the program that they are written. Many valuable time and important information are lost.

Therefore, not only the SRS are important for the documentation for the project, but well written are essential for debugging and maintenance of the program.

Basic software engineering concept

Conclusion

You have learned the fundamental of good osaftware development concepts, with from Java and to other OOP (Object Oriented Programming) languages.

All coding starts with the basic understanding of the design concept and techniques. Your understanding and mastering of these design concept and techniques will help your career in programming tremendously.