# SOFTWARE RE-ENGINEERING

**SE-409**

# Contents

- Object Oriented Re-Engineering Patterns
  - Refactoring
    - Techniques
    - Discuss Reasons for Why Code Refactoring is Important in Software Development

# Refactoring

- Refactoring is the process of restructuring code, while not changing its original functionality.
- The goal of refactoring is to improve internal code by making many small changes without altering the code's external behavior.
- Refactoring improves code readability and reduces complexities. Refactoring can also help software developers find bugs or vulnerabilities hidden in their software.

# When to Refactor

Refactoring should be done on an ongoing basis as part of the software development process.

However, there are certain situations when refactoring becomes particularly important:

- **Code review feedback** - When code is reviewed by other developers, they may identify areas where the code could be improved. Refactoring can help to address these issues and improve the quality of the code.
- **Changes in requirements** - When the requirements for a project change, it may be necessary to refactor the code to accommodate the new requirements. Refactoring can help to ensure that the code remains maintainable and adaptable over time.

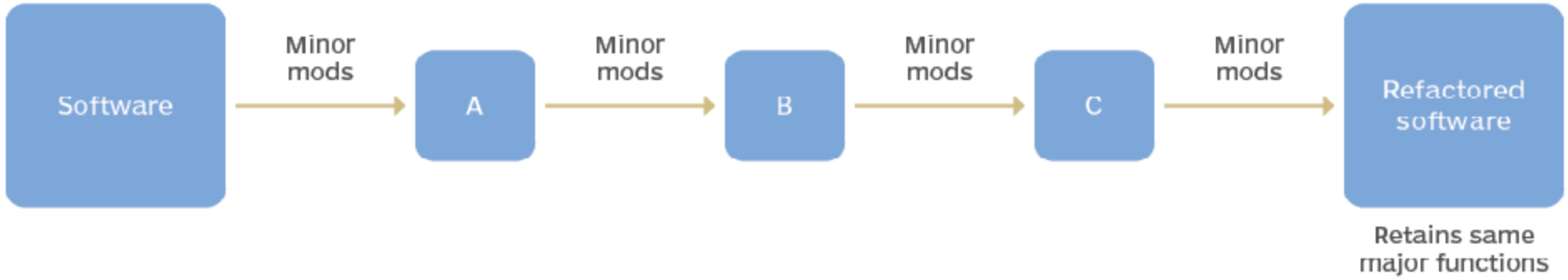
# When to Refactor

- Performance issues - When code performance is slow, it may be necessary to refactor the code to improve its efficiency. Refactoring can help to identify and remove bottlenecks in the code.

❖ You should not refactor:

- Stable code that won't need to change.
- Someone else's code , unless the other person agrees to it or no copyrights issue.

# the process of making several small code refactoring



# Refactoring techniques

There are many different refactoring techniques that developers can use to improve the design and structure of their code.

Here are some commonly used ones:

- Extract Method
- Inline Method
- Rename Method/Class
- Introduce Parameter Object
- Introduce Null Object etc.

# Extract Method

- Extract Method is a refactoring technique that involves taking a block of code within a method and moving it into a separate method with a descriptive name.
- The goal of Extract Method is to improve the readability, maintainability, and reusability of the code by breaking down complex or lengthy methods into smaller, more focused ones.
- This can make the code more modular , easier to understand and reusable.

# Extract Method

**Before Refactoring:**

```
def print_info(name, age):
    print("Name: " + name)
    print("Age: " + str(age))
    if age >= 18:
        print("You are an adult.")
    else:
        print("You are a minor.")
```

**After Refactoring:**

```
def print_info(name, age):
    print("Name: " + name)
    print("Age: " + str(age))
    print_age_status(age)

def print_age_status(age):
    if age >= 18:
        print("You are an adult.")
    else:
        print("You are a minor.")
```

In the example, the print_info method contains some conditional logic that can be extracted into a separate method print_age_status to improve readability and maintainability.

# Inline Method

- The opposite of Extract Method, this technique involves taking a method and moving its contents back into the calling code, removing unnecessary abstraction.
- Inline Method is a refactoring technique used to improve the readability and maintainability of code.
- The goal of Inline Method is to simplify the code by reducing unnecessary abstraction. It is used when a method is only called in one place and its contents are simple and straightforward.

# Inline Method

**Before Refactoring:**

```
class Calculator:
    def add(self, a, b):
        return self.increment(a, b)

    def increment(self, x, y):
        return x + y + 1
```

**After Refactoring:**

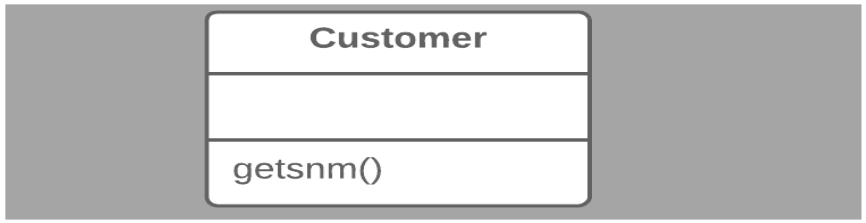
```
class Calculator:
    def add(self, a, b):
        return a + b + 1
```

- In this example, the add method calls the increment method to add two numbers and increment the result by one.
- However, the increment method is only called in one place and its contents are simple and straightforward.
- Therefore, we can safely inline the increment method into the add method to simplify the code and eliminate unnecessary abstraction.

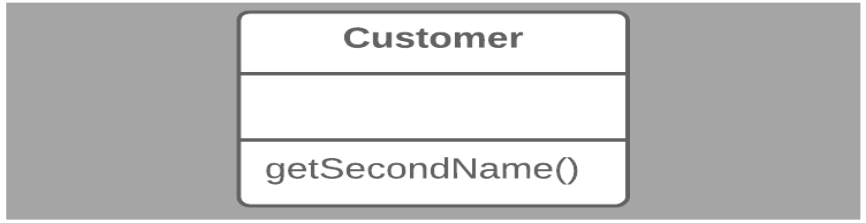
# Rename Method/Class

- Rename Method/Class is a refactoring technique that involves changing the name of a method or a class to make it more meaningful, descriptive, or consistent with the naming conventions used in the codebase.
- **Problem: The name of a method doesn't explain what the method does.**

**Before Refactoring:**




**After Refactoring:**




Rename the method getsnm() to getSecondName() to improve the meaning of the code.

# Introduce Parameter Object

- Introduce Parameter Object is a refactoring technique that involves grouping related parameters into a single object, which is then passed to the method instead of individual parameters.
- When a method has a long list of parameters, it can be difficult to understand what each parameter is used for and how they relate to each other.
- By introducing a parameter object, the related parameters can be grouped together into a single object, which can be given a meaningful name that describes what the parameters represent.
- This can help to simplify the method signature and make the code more readable.

# Introduce Parameter Object

## Before Refactoring:

```
def format_person(name, age, city):
    return f"{name} is {age} years old and lives in {city}."
```

- By using the Introduce Parameter Object technique, you have simplified the function signature and made the code more readable and maintainable.
- If you need to add more properties to the Person class in the future, you can do so without having to modify the function signature.

## After Refactoring:

```
class Person:
    def __init__(self, name, age, city):
        self.name = name
        self.age = age
        self.city = city

def format_person(person):
    return f"{person.name} is {person.age} years old and lives in {person.city}.“

person = Person(“Saad", 30, “Pakistan")
formatted_string = format_person(person)
print(formatted_string)
```

# Introduce Null Object

- Introduce Null Object is a refactoring technique that involves creating a null object that can be used in place of a null reference.
- This can help to simplify the code and prevent null reference errors.

**Before Refactoring:**

```
def print_person_name(person):
    print(person.name)

# If person is None or null, this function will raise a NullPointerException.
```

**After Refactoring:**

```
class NullPerson:
    def __init__(self):
        self.name = "Unknown"

def print_person_name(person):
    if person is None:
        person = NullPerson()
    print(person.name)
```

# Introduce Null Object

```
# call the print_person_name function with a Person object
or a None value without worrying about null reference errors

person = Person(“Saad")
print_person_name(person)  # prints “Saad"

person = None
print_person_name(person)  # prints "Unknown"
```

By using the Introduce Null Object technique, you have prevented null reference errors and made the code more robust and reliable.

# Reasons for Why Code Refactoring is Important in Software Development

Code refactoring is an essential part of software development because it helps to improve the quality and maintainability of the code.

Some of the reasons why code refactoring is important:

- Improves code quality
- Enhances maintainability
- Increases code reuse
- Facilitates team collaboration
- Improves performance

# Reasons for Why Code Refactoring is Important in Software Development

- Code refactoring is important in software development or Re-engineering because it helps to improve the quality, maintainability, and performance of the code, while reducing technical debt and facilitating team collaboration.
- By making the code easier to understand and modify, refactoring can also save time and effort in the long run.

# Comparison of Object-Oriented Re-Engineering Patterns and Refactoring in Adapting to Changing Requirements

| Aspect | Object-Oriented Patterns | Refactoring |
|---|---|---|
| **Focus** | System architecture/design | Code structure and readability |
| **Scope** | Broad and strategic | Localized and tactical |
| **Goal** | Address systemic issues and scalability | Improve code clarity and maintainability |
| **When Used** | Major requirement shifts (e.g., new features, paradigms) | Adapting code to minor changes or simplifying logic |
| **Example** | Introduce design patterns (e.g., Strategy, Adapter) | Rename variables, extract/rewrite methods |