UNIVERSITY OF CALGARY

SENG 401 – SOFTWARE ARCHITECTURE

REFACTORING

Dr. Ronnie de Souza Santos
https://www.drdesouzasantos.ca/

SOFTWARE EVOLUTION

THE PROCESS OF DEVELOPING, MAINTAINING, AND UPDATING SOFTWARE FOR VARIOUS REASONS. THIS PROCESS IS COMPLEX AND ENTAILS INHERENT RISKS.

MOTIVATION
Inherent and continuous aspect of the software development lifecycle

OPPORTUNITY

CHALLENGING

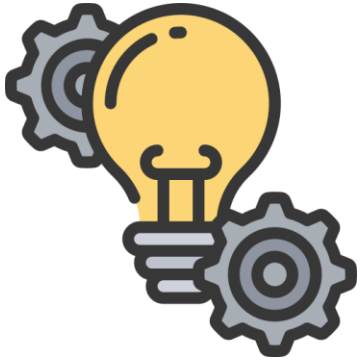
QUALITY

REFACTORING

THE PROCESS OF RESTRUCTURING AN EXISTING CODE, ALTERING ITS INTERNAL STRUCTURE WITHOUT CHANGING ITS EXTERNAL BEHAVIOR.

MOTIVATION
Improve the overall quality, maintainability, and readability of code.

REQUIREMENTS CHANGE

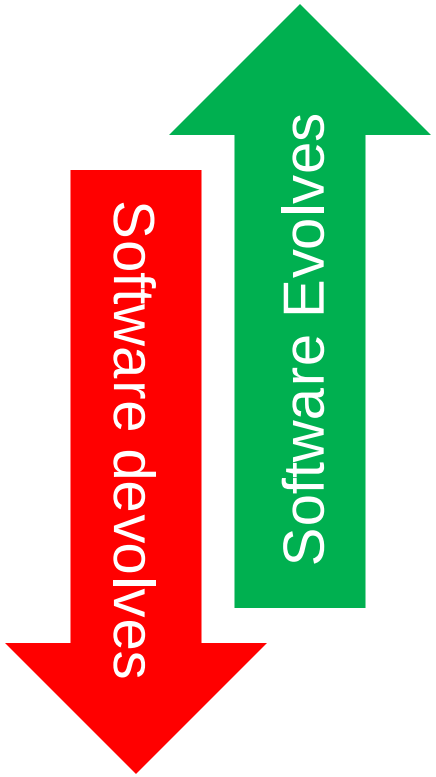


DESIGN CHANGES

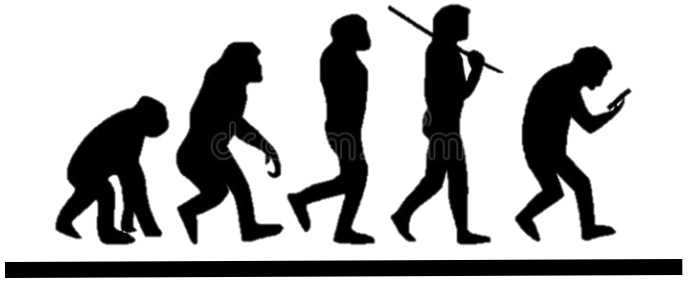
IMPLEMENTATION CHANGES

SOFTWARE NEEDS REFACTORING

QUALITY IMPROVES

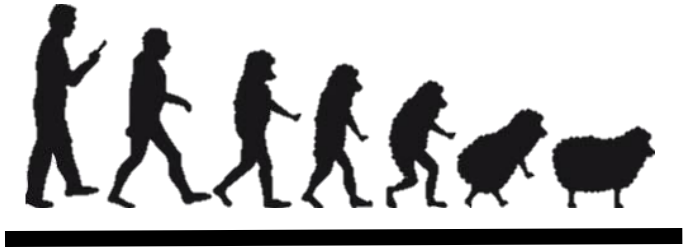


QUALITY DEGRADES



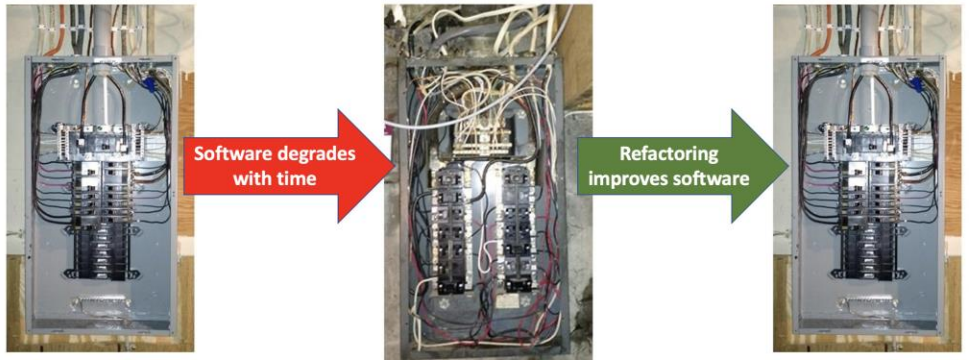
SOFTWARE EVOLVES

- Bugs are fixed
- Useless code removed
- Bad code improved
- Complexity is reduced



SOFTWARE DEVOLVES

- Band-aid fixes
- Short-cuts
- Poorly thought-out design
- Undocumented changes



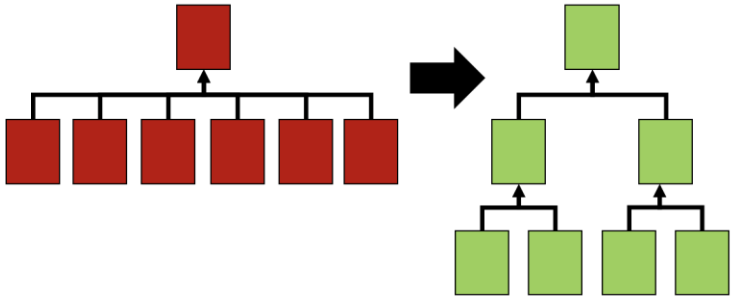
REFACTORING EXAMPLES

2000 → MAX.VALUE

i → districtIndex

IMPROVING READABILITY

Removing magic constants
Making variable names more informative
Decomposing overly long methods



IMPROVING UNDERSTANDABILITY

Reorganizing the class hierarchy
Removing dead or unused code
Fixing design

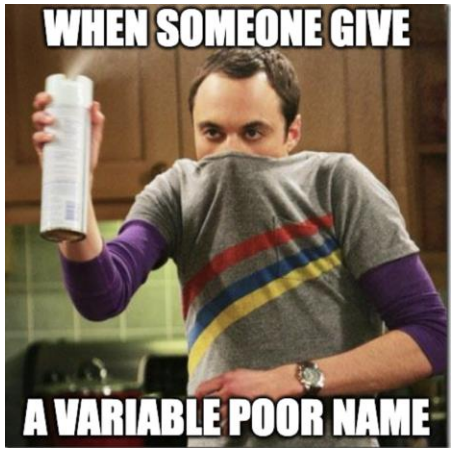
IMPROVING FLEXIBILITY

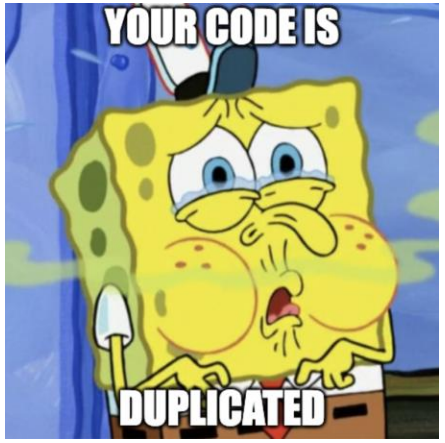
Improving interfaces
Improving APIs
Changing architecture

WHEN IS REFACTORING NEEDED?

Code is duplicated	A method is too long	A loop is too long or too deeply nested	A parameter list has too many parameters
A method uses more features of another class than of its own class	A method has a poor name	Data members are public	Comments are not used to explain difficult code
Many global variables are used	A class has poor cohesion	Changes require parallel modifications to multiple classes	Inheritance hierarchies must be modified in parallel
	A class doesn't do very much	One class is overly intimate with another	

CODE SMELLS



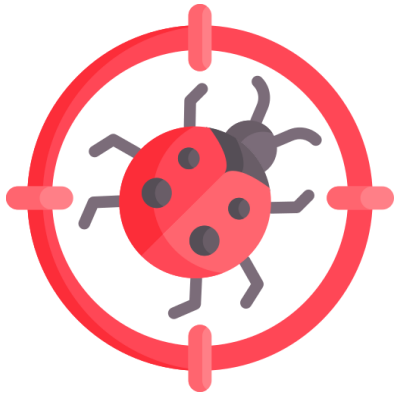


CODE SMELLS

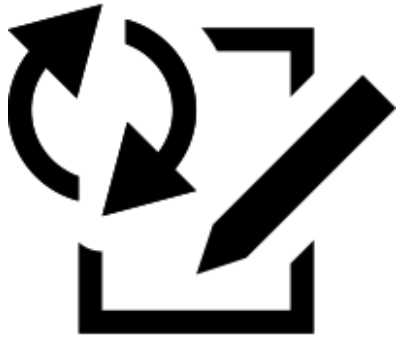
INDICATORS OF POTENTIAL ISSUES OR AREAS OF IMPROVEMENT IN SOFTWARE CODE.

MOTIVATION

Code smells suggest areas where refactoring or restructuring the code could enhance maintainability, readability, and overall software quality.



NOT BUGS



FREQUENT CHANGES

MANAGEMENT ISSUES

COMMON CODE SMELL: DUPLICATED CODE

```java
public class Order {
    public double calculateTotal(double price, int quantity) {
        return price * quantity * 1.10; // 10% tax
    }
}

public class Invoice {
    public double computeInvoiceTotal(double price, int quantity) {
        return price * quantity * 1.10; // 10% tax (duplicate)
    }
}
```

```java
public class TaxCalculator {
    public static double calculateTotal(double price, int quantity) {
        return price * quantity * 1.10; // Centralized tax calculation
    }
}

public class Order {
    public double getTotal(double price, int quantity) {
        return TaxCalculator.calculateTotal(price, quantity);
    }
}

public class Invoice {
    public double getTotal(double price, int quantity) {
        return TaxCalculator.calculateTotal(price, quantity);
    }
}
```

COMMON CODE SMELL: LONG METHOD

```java
public void processOrder(Order order) {
    System.out.println("Processing order: " +
    order.getId());

    if (order.getItems().size() == 0) {
        System.out.println("Order is empty");
    }

    int i = 0;
    while (i < order.getItems().size()) {
        System.out.println("Processing item: " +
        order.getItems().get(i).getName());
        i++;
    }

    System.out.println("Total price: " +
    order.getTotal());
}
```

```java
public void processOrder(Order order) {
    printOrderDetails(order);
    validateOrder(order);
    processItems(order);
}

private void printOrderDetails(Order order) {
    System.out.println("Processing order: " +
    order.getId());
}

private void validateOrder(Order order) {
    if (order.getItems().size() == 0) {
        System.out.println("Order is empty");
    }
}

private void processItems(Order order) {
    int i = 0;
    while (i < order.getItems().size()) {
        System.out.println("Processing item: " +
        order.getItems().get(i).getName());
        i++;
    }
}
```

COMMON CODE SMELL: LONG METHOD

```java
public void processOrder(Order order) {
    System.out.println("Processing order: " +
    order.getId());

    if (order.getItems().size() == 0) {
        System.out.println("Order is empty");
    }

    int i = 0;
    while (i < order.getItems().size()) {
        System.out.println("Processing item: " +
        order.getItems().get(i).getName());
        i++;
    }

    System.out.println("Total price: " +
    order.getTotal());
}
```

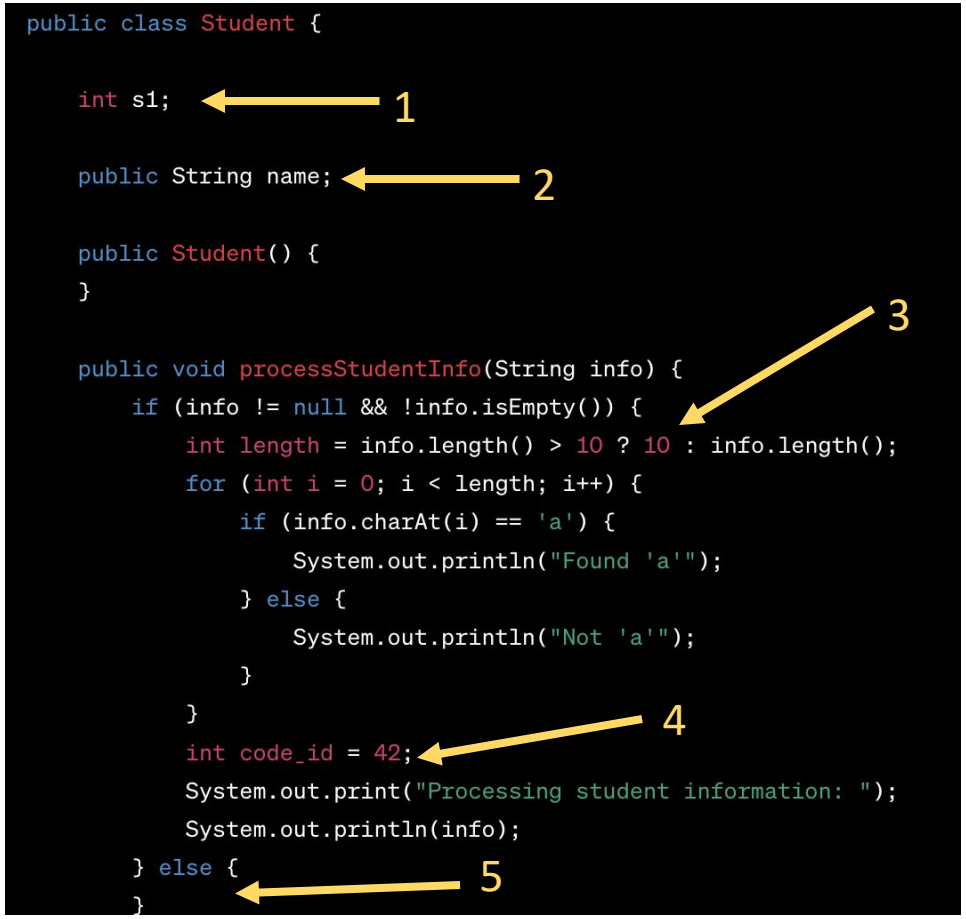
```java
public void processOrder(Order order) {
    printOrderDetails(order);
    validateOrder(order);
    processItems(order);
}

private void printOrderDetails(Order order) {
    System.out.println("Processing order: " +
    order.getId());
}

private void validateOrder(Order order) {
    if (order.getItems().size() == 0) {
        System.out.println("Order is empty");
    }
}

private void processItems(Order order) {
    int i = 0;
    while (i < order.getItems().size()) {
        System.out.println("Processing item: " +
        order.getItems().get(i).getName());
        i++;
    }
}
```

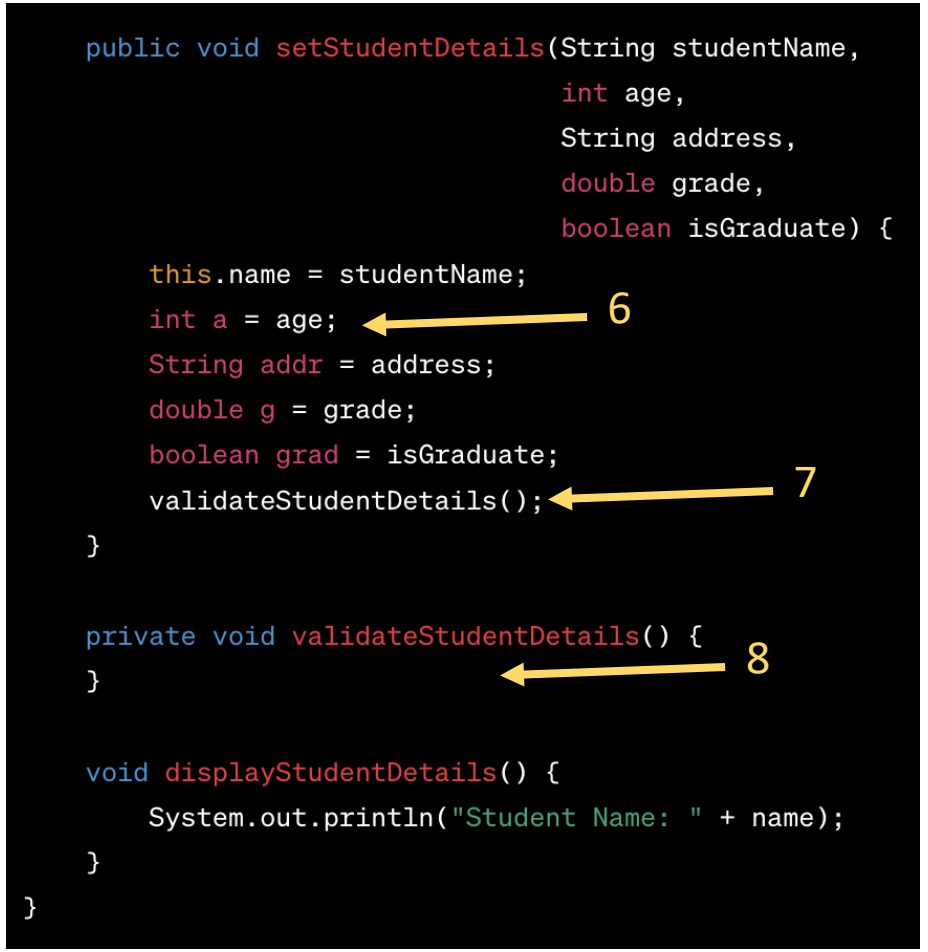
COMMON CODE SMELL: POOR COHESION

```
public class UserManager {
    public void addUser() { }
    public void removeUser() { }
    public void sendEmail() { }
    public void logActivity() { }
}
```

```
public class UserService {
    public void addUser() { }
    public void removeUser() { }
}

public class EmailService {
    public void sendEmail() { }
}

public class LoggingService {
    public void logActivity() { }
}
```

FINDING CODE SMELLS





1. Naming
2. Public attribute
3. Magic number
4. Unused variable
5. Empty method
6. Naming
7. Possible dependence
8. Empty method

BLACK HISTORY MONTH: KATHERINE JOHNSON

KATHERINE JOHNSON
(1918 – 2020)

Her mathematical work laid the foundation for the later use of computers and software in space missions. Her accomplishments also contributed to breaking down racial and gender barriers in the scientific and engineering fields.

REFACTORING AT THE DATA LEVEL

INVOLVES RESTRUCTURING THE WAY DATA IS ORGANIZED OR REPRESENTED IN THE CODE.

EXAMPLES

Rename variables
Encapsulate variables
Initialize variables
Replace Magic Number
Replace Array with Object

```
// Before
public int age;

// After
private int age;

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}
```

```
    // Before: Using a magic number
    public int multiplyByFiveBefore(int number) {
        return 7 * number;
    }

    // After: Introducing a symbolic constant for the magic number
    private static final int MULTIPLICATION_CONSTANT = 7;

    public int multiplyByFiveAfter(int number) {
        return MULTIPLICATION_CONSTANT * number;
    }
}
```

```
// Before
String[] personInfo = new String[3];
personInfo[0] = "John";
personInfo[1] = "Doe";
personInfo[2] = "25";

// After
Person person = new Person("John", "Doe", 25);
```

REFACTORING AT THE STATEMENT LEVEL

FOCUSES ON IMPROVING INDIVIDUAL STATEMENTS OR EXPRESSIONS WITHIN METHODS.

EXAMPLES

Extract command
Remove fragments
Split statements
Decompose expressions

```
// Before
if (head == null) {
    head = node;
    tail = node;
} else {
    tail.next = node;
    tail = node;
}

// After
if (head == null) {
    head = node;
} else {
    tail.next = node;
}
tail = node;
```

```
// Before
if ((res == null) || (b == null) ||
    (b.getHeight() != height) || (b.getWidth() != width) ||
    (res.getHeight() != height) || (res.getWidth() != width)) {
  return null;
}

// After
paramsAreNull = (res == null) || (b == null);
bIsNotSameSize = (b.getHeight() != height) || (b.getWidth() != width);
resIsNotSameSize = (res.getHeight() != height)
                || (res.getWidth() != width);

if (paramsAreNull || bIsNotSameSize || resIsNotSameSize) {
  return null;
}
```

REFACTORING AT THE ROUTINE LEVEL

INVOLVES RESTRUCTURING ENTIRE METHODS OR ROUTINES FOR IMPROVED READABILITY AND MAINTAINABILITY.

EXAMPLES

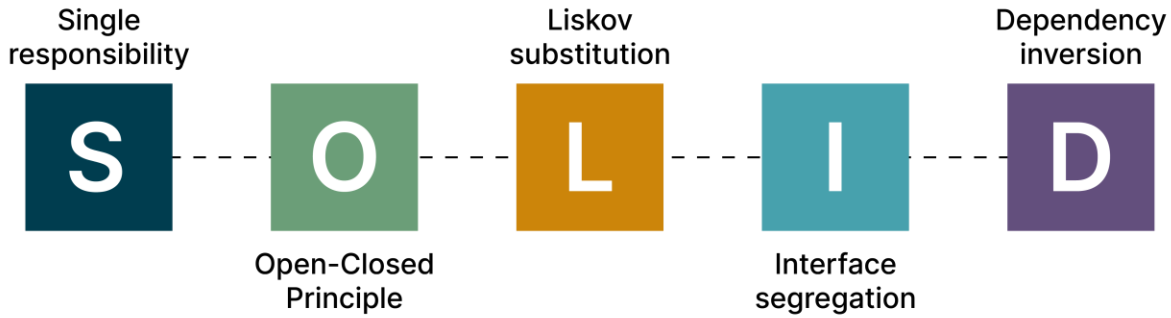
Extract method
Collapse method
Move method
Delete method

```
// Before

public class Shape {
  ...
}

public class Triangle extends Shape {
  private Color color;
  ...
  public void changeColor(Color color) {
    ...
  }
}

public class Circle extends Shape {
  private Color color;
  ...
  public void changeColor(Color color) {
    ...
  }
}
```

```
// After

public class Shape {
  private Color color;
  ...
  public void changeColor(Color color) {
    ...
  }
}

public class Triangle extends Shape {
  public void changeColor(Color color) {
    ...
  }
}

public class Circle extends Shape {
  public void changeColor(Color color) {
    ...
  }
}
```

REFACTORING AT THE CLASS LEVEL

INVOLVES RESTRUCTURING AND IMPROVING THE ORGANIZATION OF CODE WITHIN A CLASS



EXAMPLES

Split class
Combine classes
Extract abstract interface
Restrict access
Change Hierarchy

Creational	Structural	Behavioral
• Factory Method	• Adapter	• Interperter
• Abstract Factory • Builder • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Facade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Momento • Observer • State • Strategy • Visitor

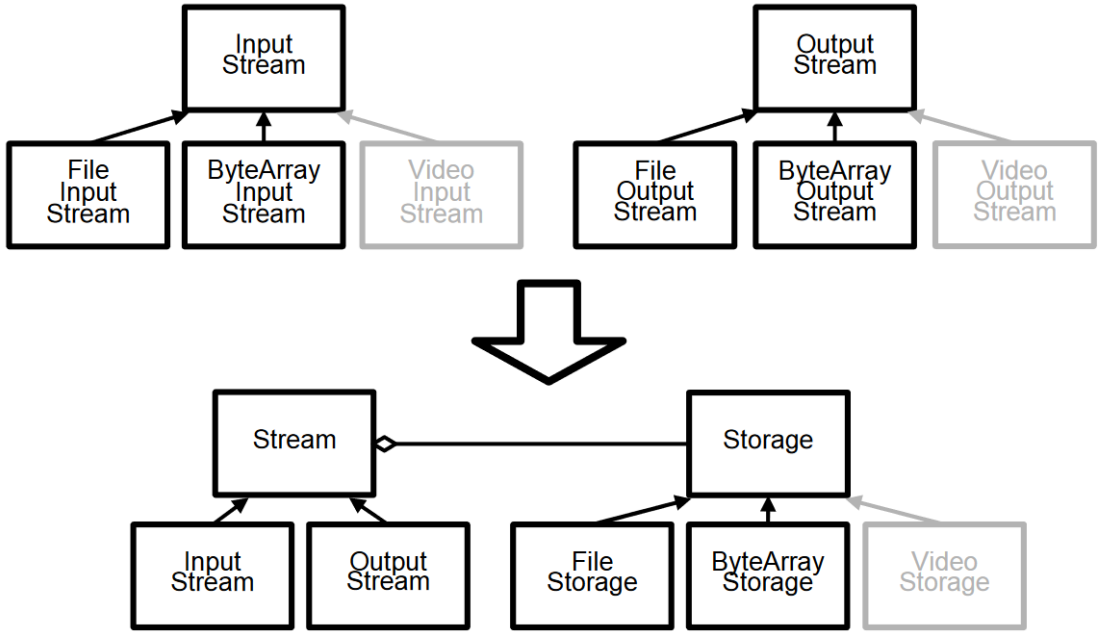
DESIGN PATTERNS

REFACTORING AT THE SYSTEM LEVEL

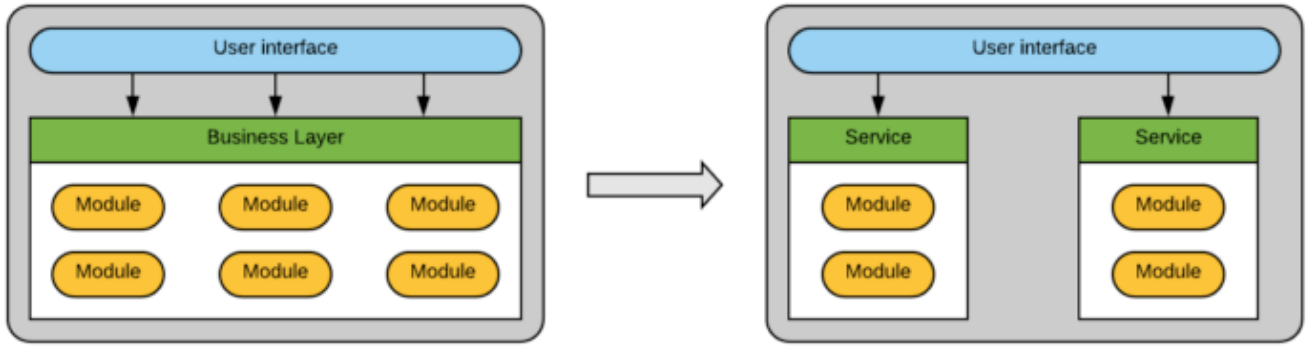
INVOLVES MAKING CHANGES TO THE OVERALL ARCHITECTURE AND STRUCTURE COMPONENTS OF A SOFTWARE.

EXAMPLES

Architecture change
Data migration
Security enhancements
Legacy System Migration
CI/CD implementation



SYSTEM LEVEL: ARACHTECTURE REFACTORING



1. Refactoring Layered Architecture → Modular Services

- Layered Architecture: dependencies between layers slow down development.
- **Solution?** Break down layers into independent service components and introduce dependency inversion to reduce coupling.

2. Refactoring Pipeline Architecture → Microservices

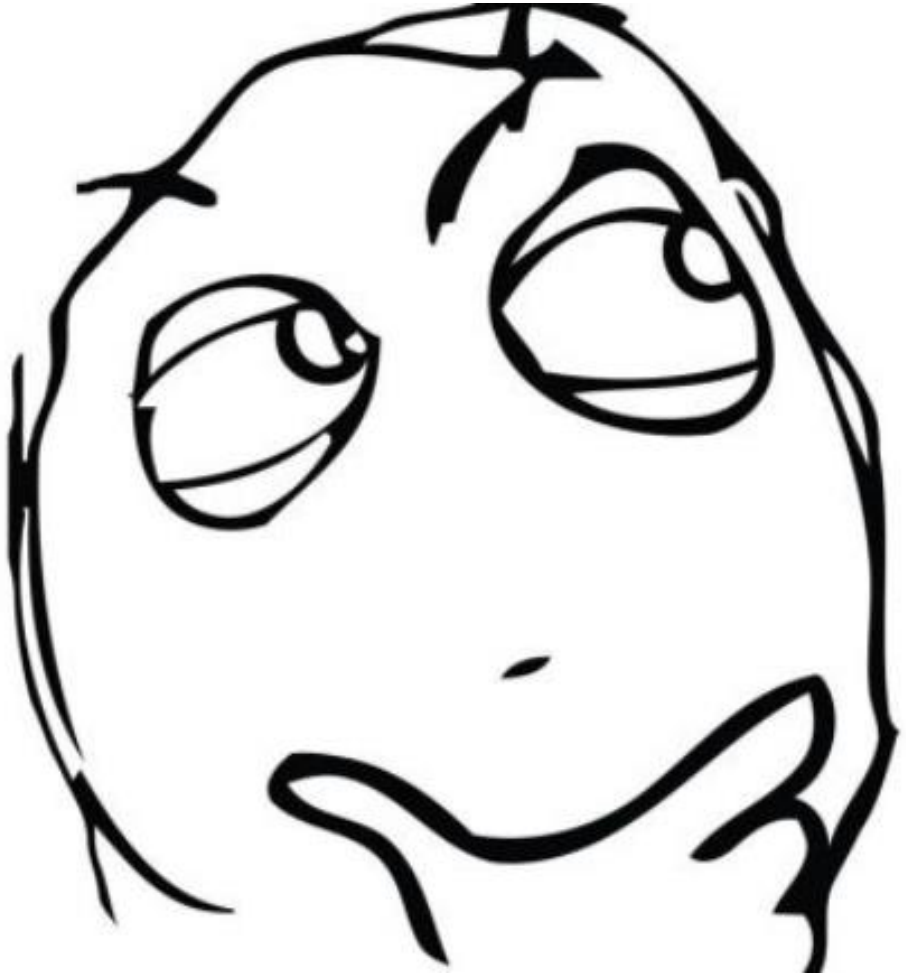
- Pipeline Architecture: rigid sequence of steps, performance bottlenecks, single point of failure.
- **Solution?** Break pipeline stages into independent microservices.

3. Refactoring Microservices → Event-Driven Architecture

- Microservice Architecture: rely on synchronous API calls, increasing latency.
- **Solution?** Adopt event sourcing to track state changes to allow asynchronous communication.

Image Source: https://medium.com/architecture-refactoring-how-to-sell-it-client/architecture-refactoring-and-design-refactoring-how-to-sell-it-client-alex-and-vira-9dd118fa4544

QUESTIONS?



UNIVERSITY OF CALGARY

SENG 401 – SOFTWARE ARCHITECTURE

REFACTORING

Dr. Ronnie de Souza Santos
https://www.drdesouzasantos.ca/