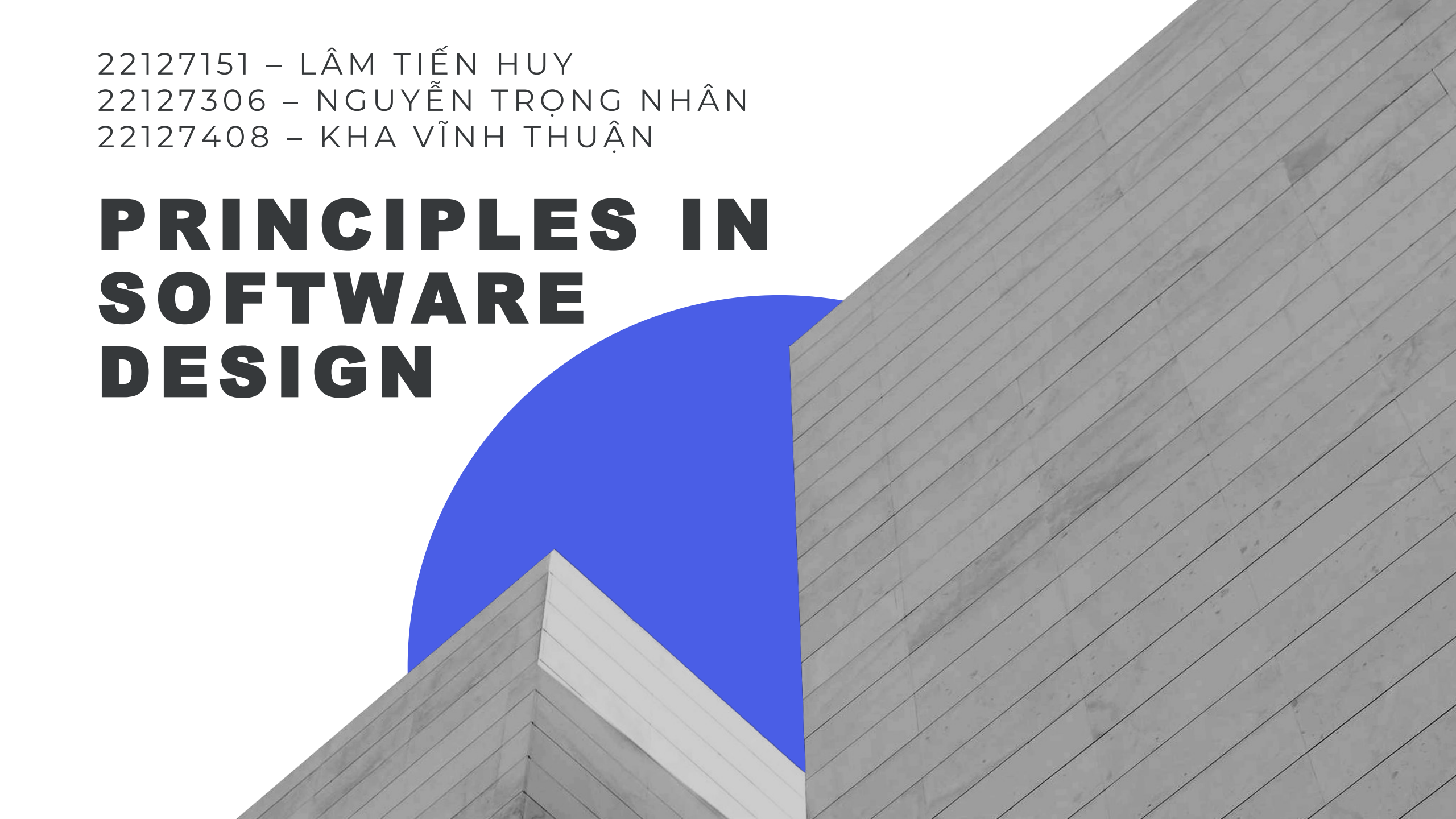
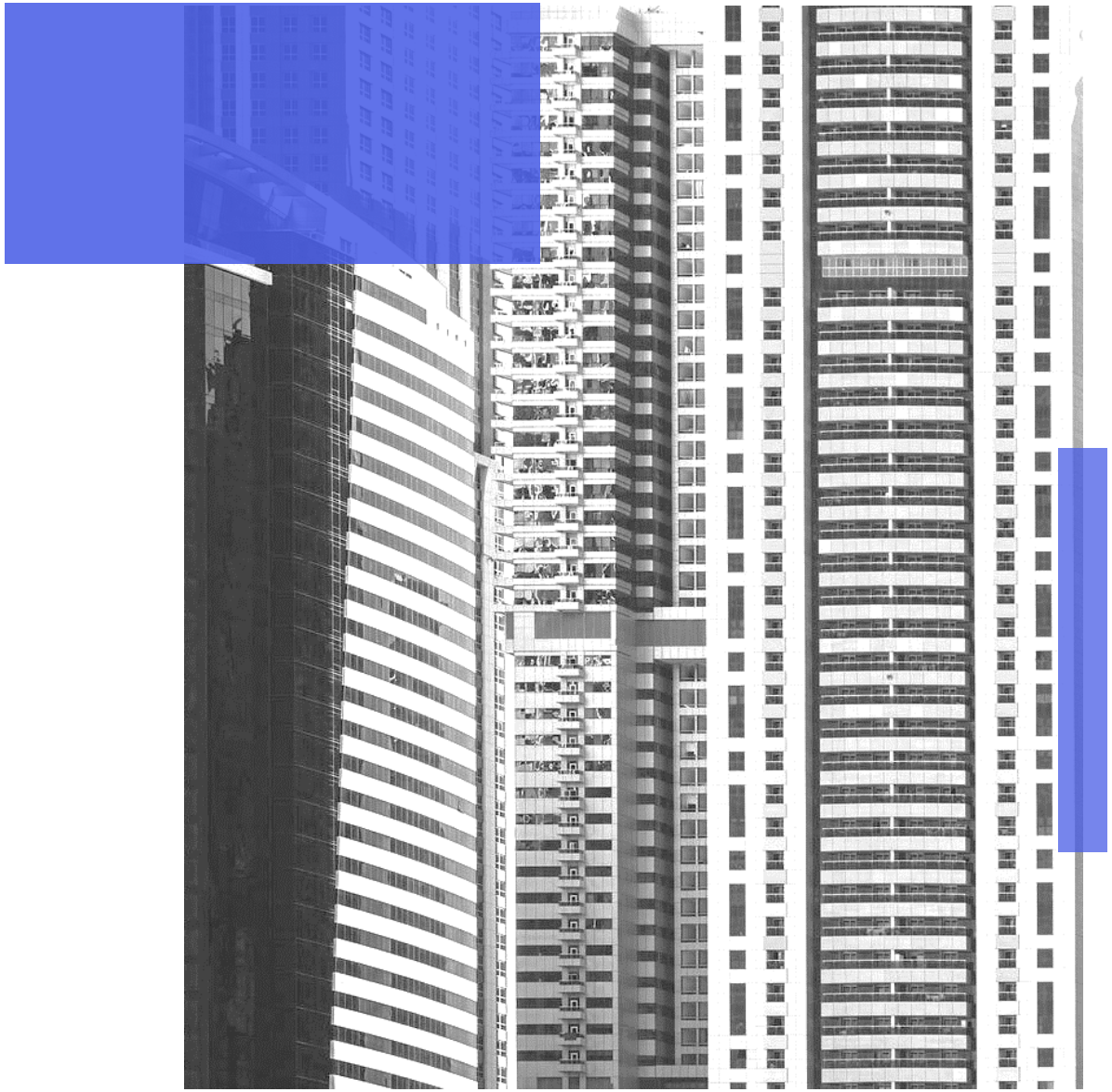
22127151 – LÂM TIẾN HUY
22127306 – NGUYỄN TRỌNG NHÂN
22127408 – KHA VĨNH THUẬN

PRINCIPLES IN SOFTWARE DESIGN





SOLID PRINCIPLES

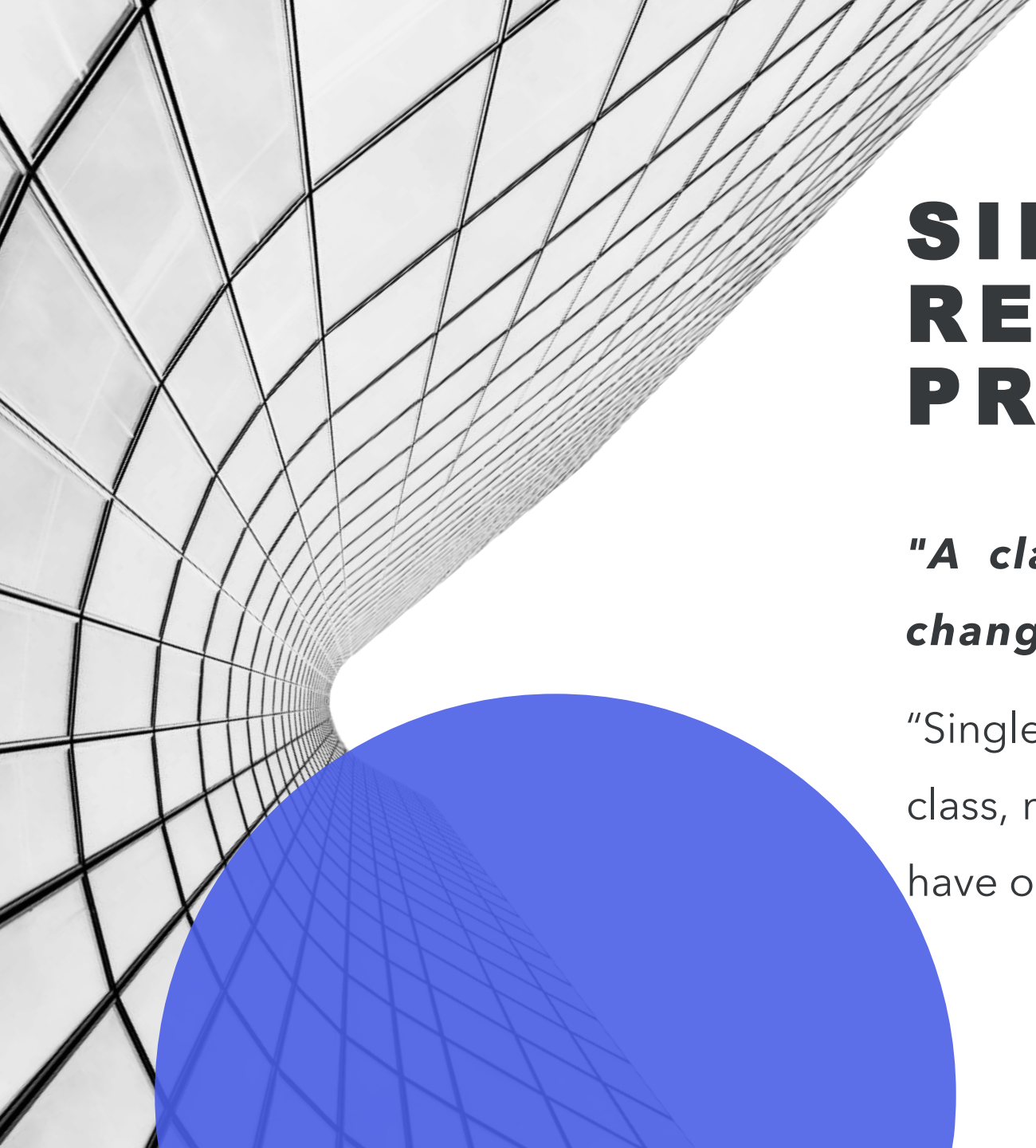
S – SINGLE RESPONSIBILITY PRINCIPLE

O – OPEN CLOSED PRINCIPLE

L – LISKOV SUBSTITUTION PRINCIPLE

I – INTERFACE SEGREGATION PRINCIPLE

D – DEPENDENCY INVERSION PRINCIPLE



SINGLE RESPONSIBILITY PRINCIPLE

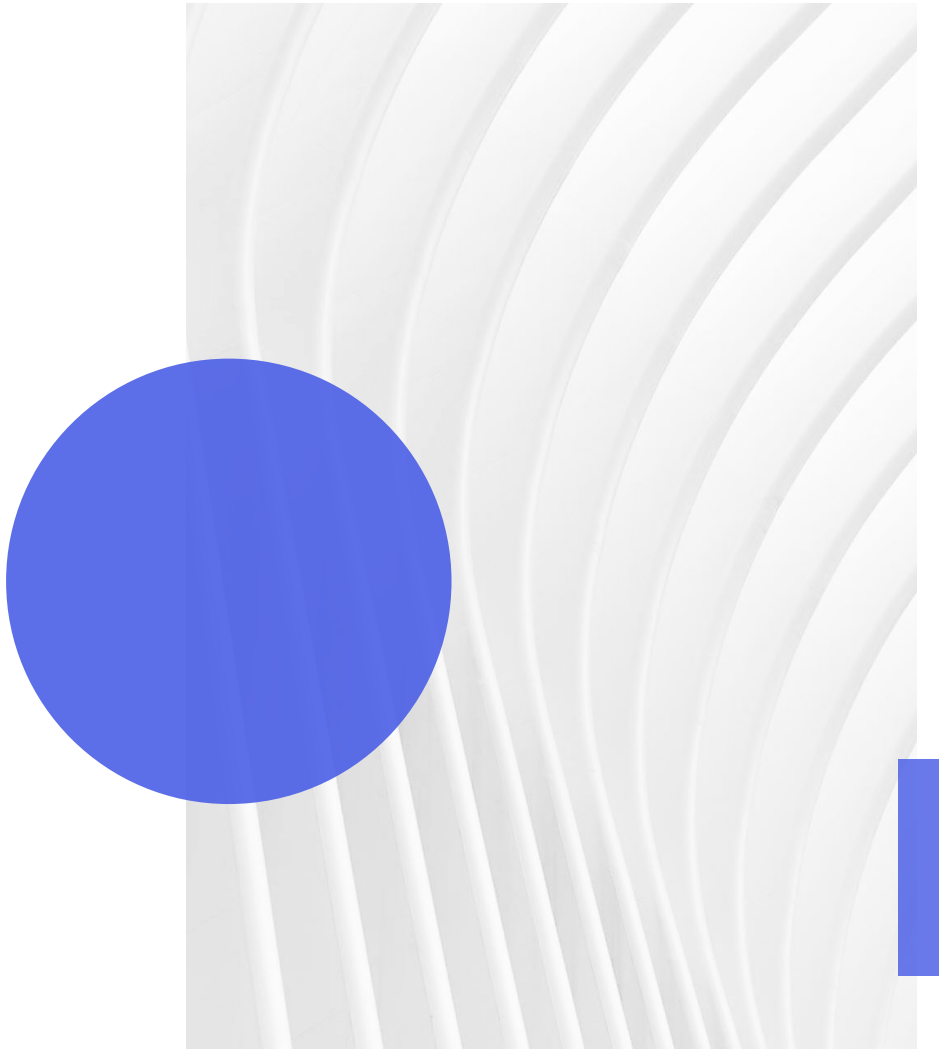
"A class should have only one reason to change."

"Single Responsibility Principle" (SRP) shows every class, module, or function in a program should only have one responsibility or purpose in a program.

KEY IDEA OF SRP

When designing your logic in either class or method, you should not be writing **all kinds of responsibilities** in **one place**. This will make your code quite complex and unmanageable.

It will also be **difficult** to **adjust new changes** later as there are high chances it will affect the other functionality and you will end up testing all the functionalities even though it is a smaller change.



SRP EXAMPLE

```
class Invoice
{
        void AddInvoice() {}
        void DeleteInvoice() {}
        void GenerateReport() {}
        void EmailReport() {}
}
```

Invoice class means that it should be only for **handling invoices.**

These two methods have only **one purpose** each, both **handling invoices.**

These two methods also have only **one purpose** each, but they are **handling reports**, not invoices.

CLASS INVOICE VIOLATED THE SINGLE RESPONSIBILITY PRINCIPLE!

SRP SOLUTION

```
class Invoice {
        public void AddInvoice() {}
        public void DeleteInvoice() {}
}
class Report {
        void GenerateReport() {}
}
class Email {
        void EmailReport() {}
}
```

Invoice class is now only for **handling invoices.**

Report class is created for **handling reports**, storing the separated method.

Email class is created for **handling emails**, storing the separated method.

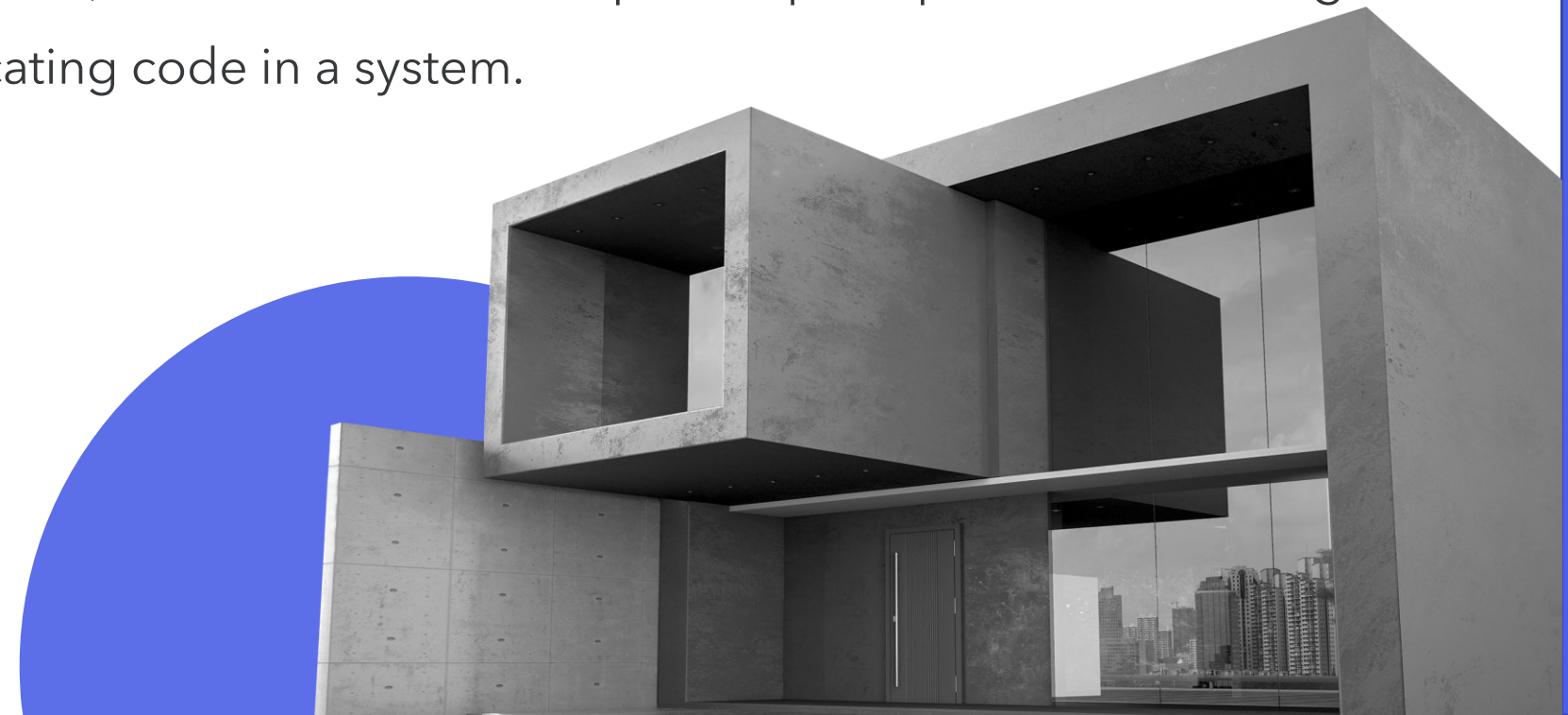
SRP ADVANTAGES

- Better Maintainability
- Easier Debugging & Testing
- Improved Readability & Understanding
- Less Coupling, More Flexibility
- Encourages Reusability

DON’T REPEAT YOURSELF

"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system."

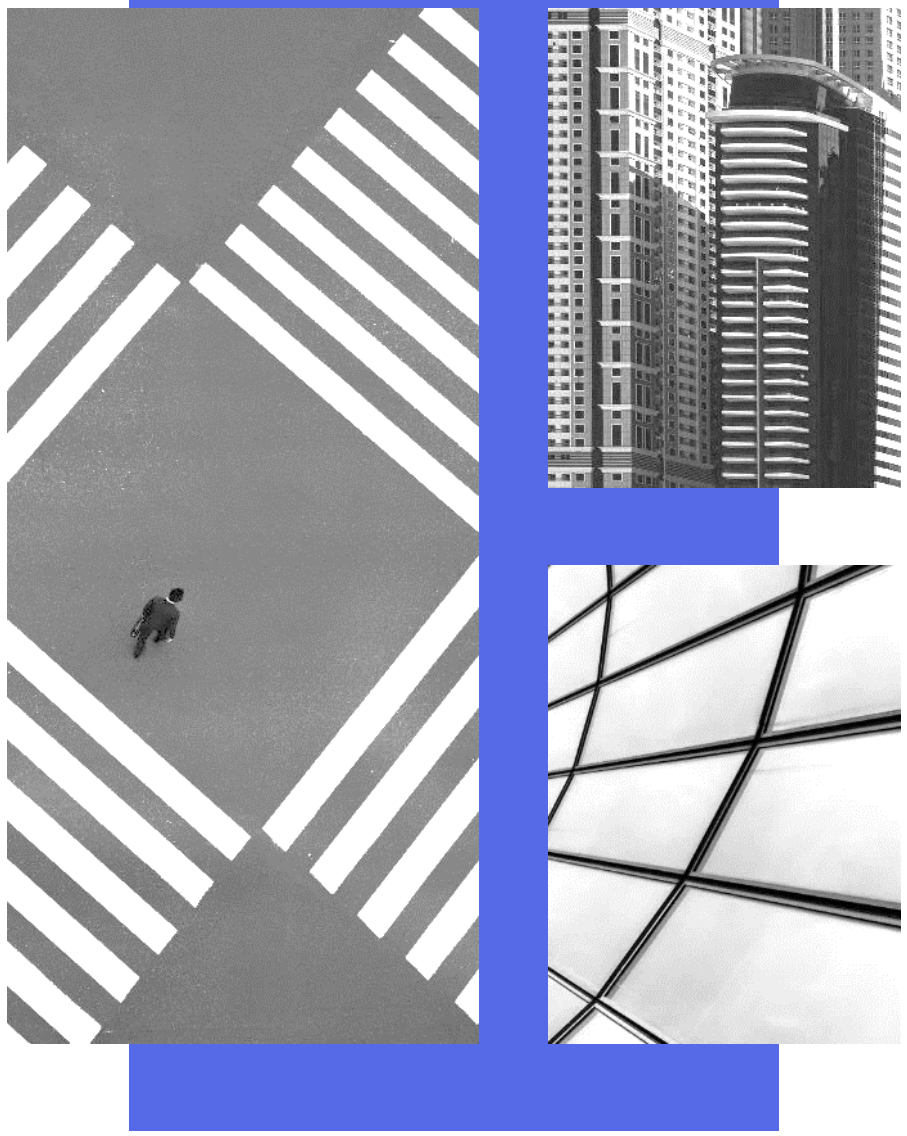
"Don't Repeat Yourself" (DRY) is a software development principle that encourages developers to avoid duplicating code in a system.



KEY IDEA OF DRY

The main idea behind DRY is to reduce redundancy and promote efficiency by ensuring that **a particular piece of knowledge or logic exists in only one place** within a codebase.

The DRY principle aim to create **reusable components, functions, or modules** that can be utilized in **various parts** of the codebase. This not only makes the code more maintainable but also minimizes the chances of errors since changes or updates only need to be made in one location.



DRY EXAMPLE

```
def calculate_discount(price):
    return price * 0.9

def calculate_premium_discount(price):
    return price * 0.85
```

This function calculates a **10% discount**.

This function calculates a **15% discount**.

Both functions **repeat** a **similar logic**: **calculating discounts.**

BOTH FUNCTIONS VIOLATED THE DON'T REPEAT YOURSELF PRINCIPLE!

DRY SOLUTION

```
def apply_discount(price, discount_rate):
    return price * (1 - discount_rate)

# Usages:
apply_discount(100, 0.1)
apply_discount(100, 0.15)
```

This function **calculates a discount** from a given discount price and rate.

The usages of the above function inputs a **10% discount** and a **15% discount**, respectively, just from **one function.**

DRY ADVANTAGES

- Reduces Code Duplication
- Easier Maintenance
- More Readable & Clean Code
- Improved Scalability
- Encourages Modular Design

SRP AND DRY

DRY addresses the issue of code duplication and avoiding repetition of logic or information. It is more about implementation details, aims to minimize the repetition of code to avoid inconsistencies and tough maintenance. DRY operates at a more granular level, dealing with specific lines of code, functions, or modules that may be duplicated.

SRP addresses classes or modules with a clear and singular responsibility. It is more about design principles, encapsulating a single responsibility or concern. SRP operates at a higher level of abstraction, focusing on the design of classes or modules and their responsibilities within the overall architecture.

DRY and SRP are often used together to create clean, maintainable, and efficient code. Applying both contributes to the development of robust and maintainable software systems.

THANKS FOR WATCHING!

22127151 – LÂM TIẾN HUY
22127306 – NGUYỄN TRỌNG NHÂN
22127408 – KHA VĨNH THUẬN

REFERENCES

GeeksforGeeks. (2023, October 31). Single responsibility in SOLID design principle. GeeksforGeeks. https://www.geeksforgeeks.org/single-responsibility-in-solid-design-principle/

Abba, I. (2022, April 26). SOLID Definition – The SOLID principles of Object-Oriented design explained. freeCodeCamp.org. https://www.freecodecamp.org/news/solid-principles-single-responsibility-principle-explained/

GeeksforGeeks. (2024a, February 22). Don't repeat yourself(DRY) in Software Development. GeeksforGeeks. https://www.geeksforgeeks.org/dont-repeat-yourselfdry-in-software-development/

Blogs, P. (2022, January 6). Understanding the DRY (Don't Repeat Yourself) Principle. Plutora. https://www.plutora.com/blog/understanding-the-dry-dont-repeat-yourself-principle