

Software Quality Engineering

Week 4

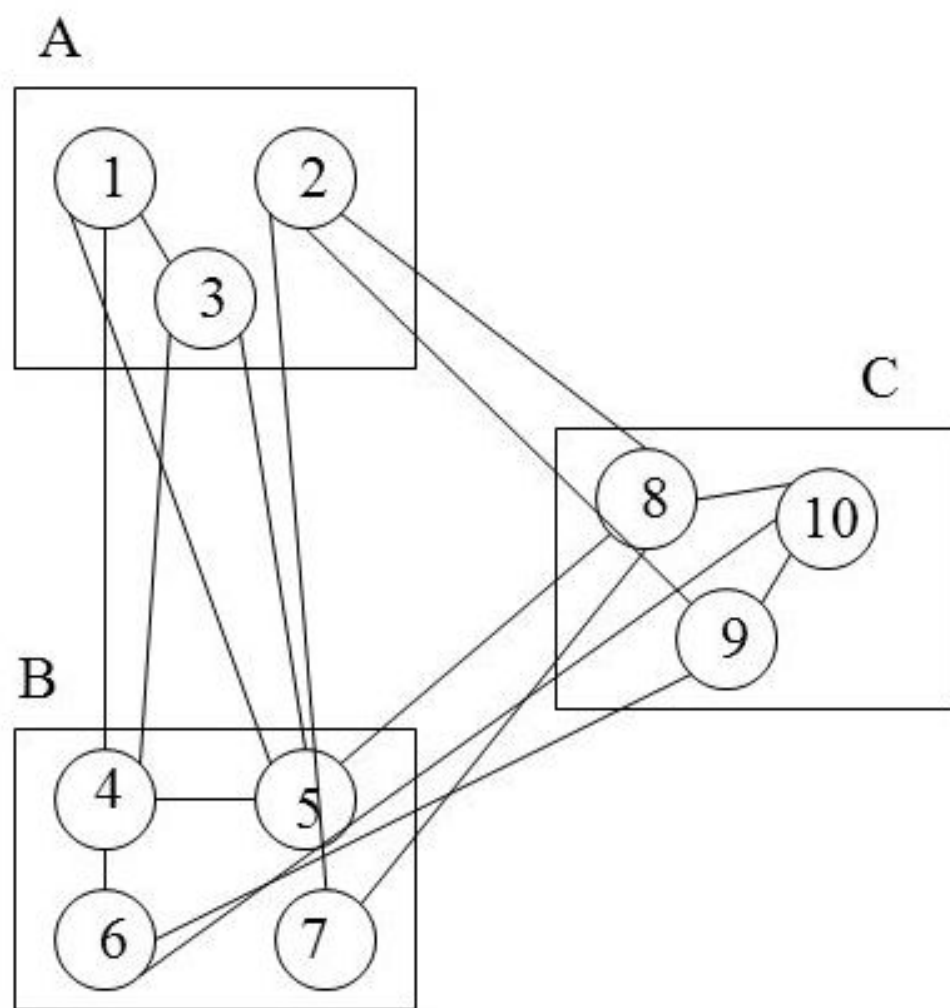
Engr. Muhammad Umer Haroon

FOLLOW THE RULES

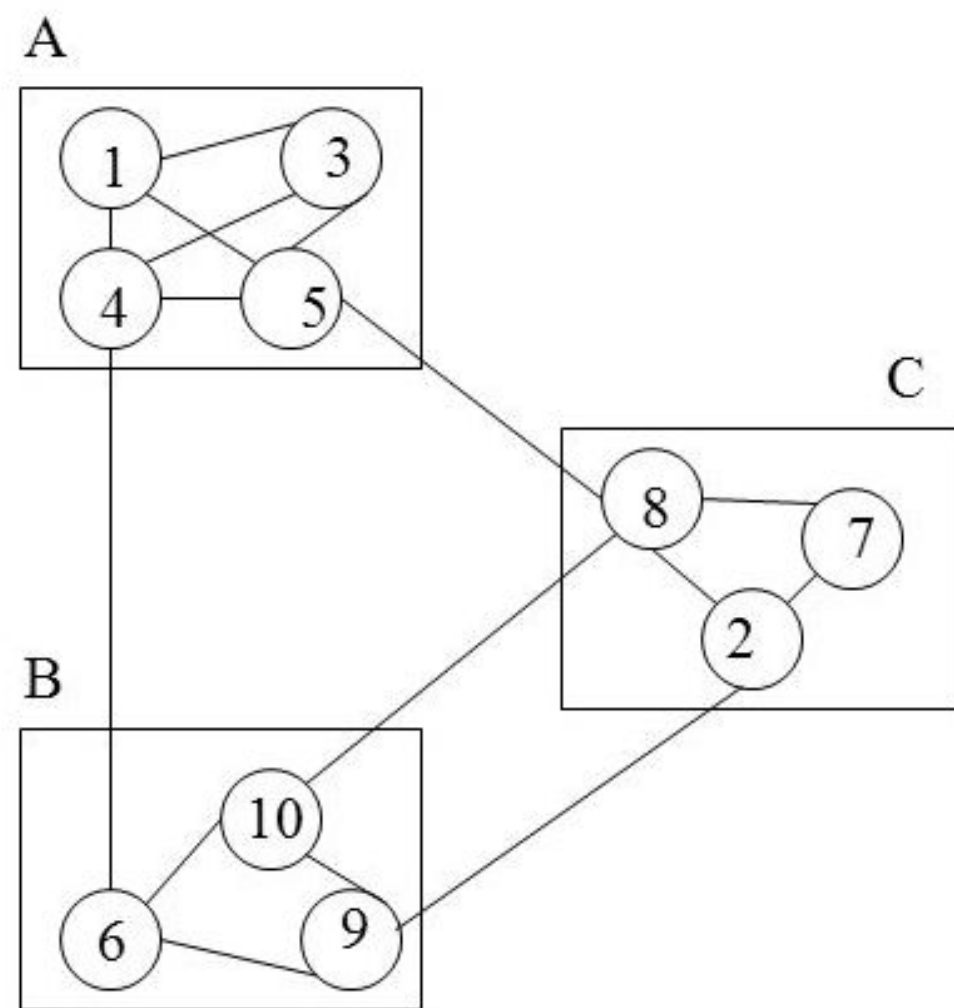
- ▶ Use of Mobile (no)
- ▶ Late coming (no)
- ▶ Short Attendance (no)
- ▶ Plagiarism (no)
- ▶ Cheat (no)
- ▶ Disrespect (no)

SOFTWARE DESIGN AND QUALITY

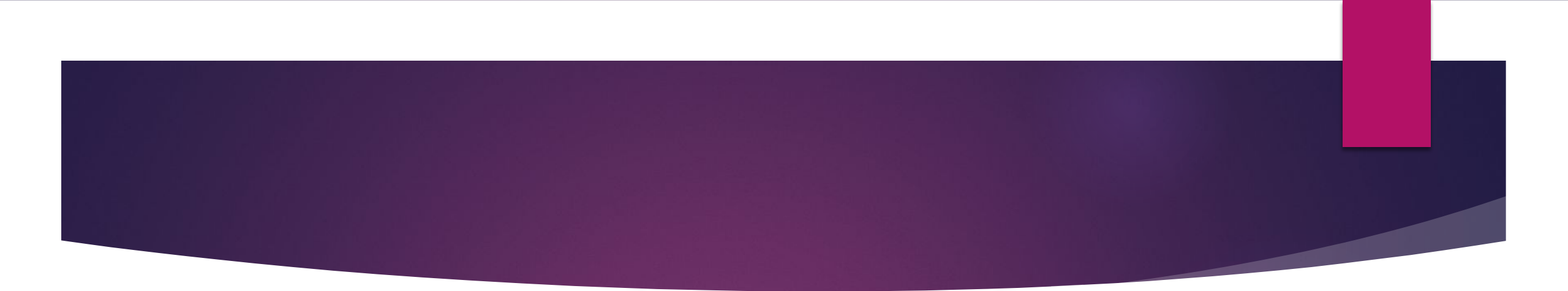
CONT....



Bad modularization:
low cohesion, high coupling



Good modularization:
high cohesion, low coupling

- 
- ▶ The quality of any software application depends mainly on the extent of coupling in the application.
 - ▶ Hence it is imperative that you measure the degree of coupling between the components of your application.

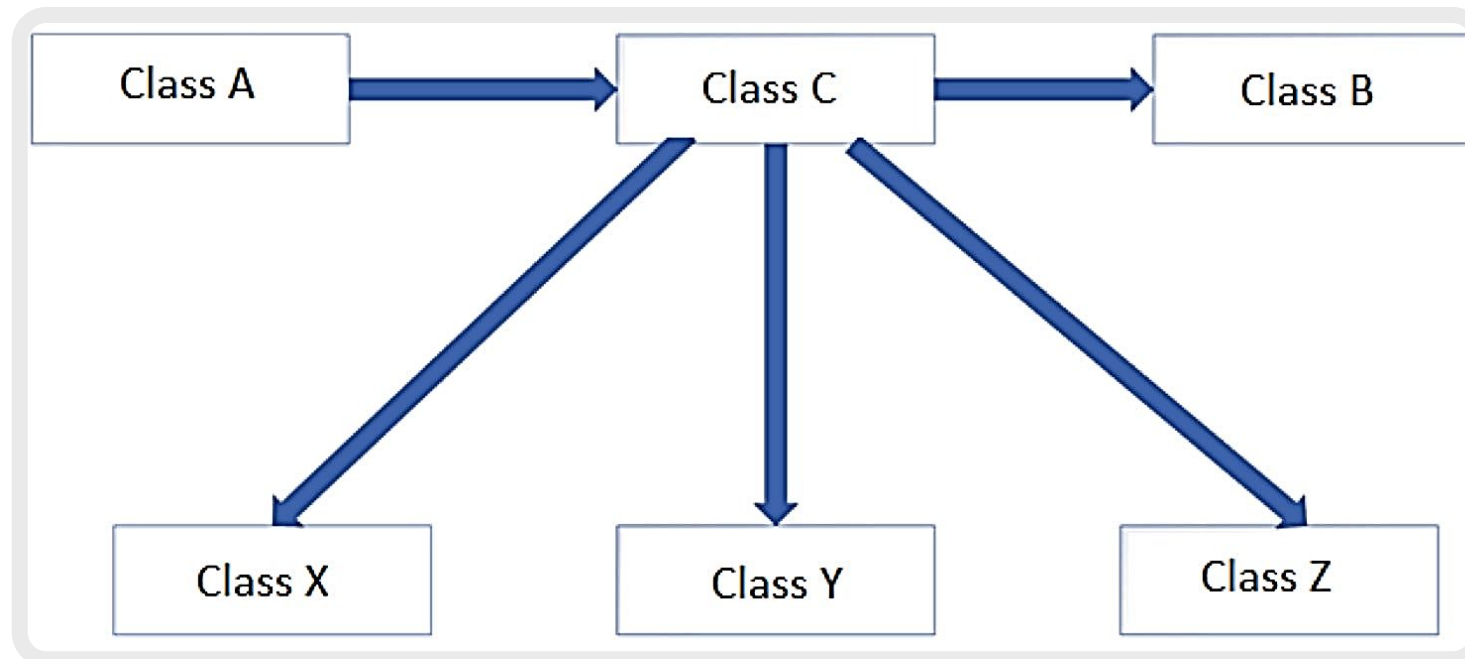
- 
- ▶ One measure of coupling is instability.

Instability

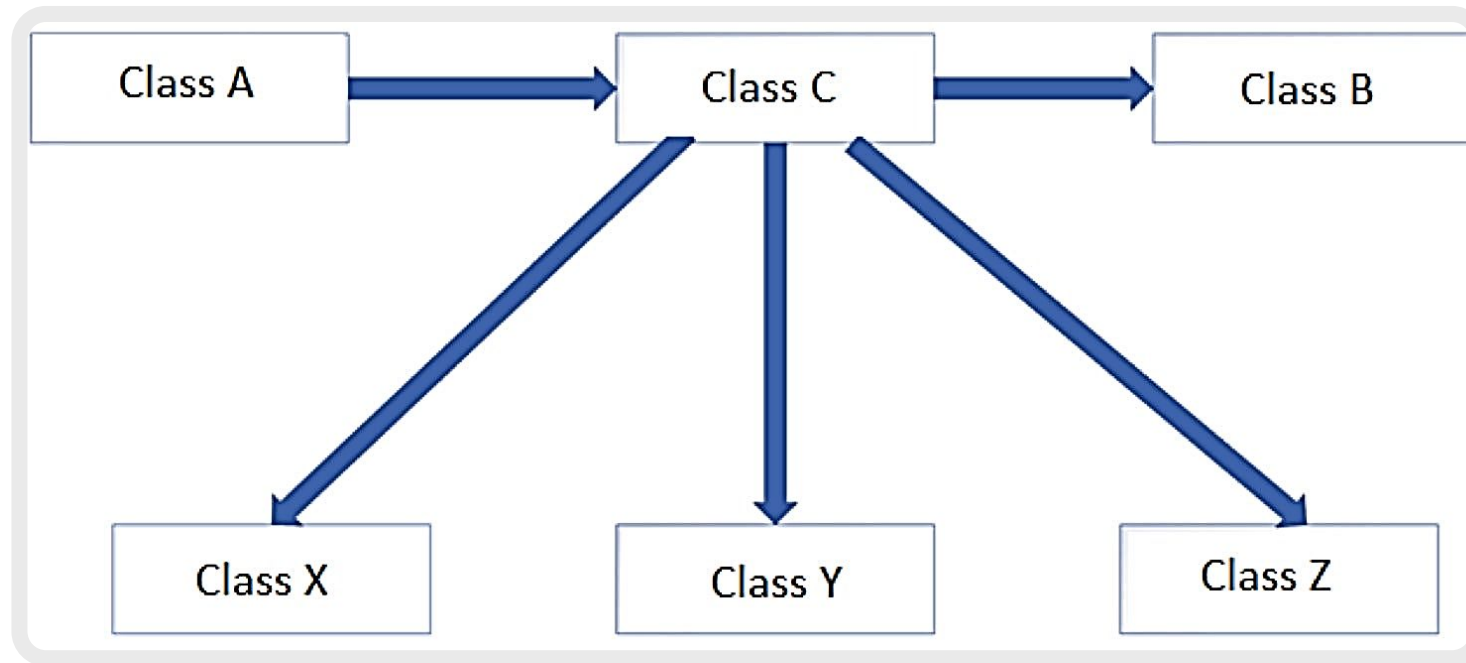
- ▶ Instability is a measure of how change propagates through the system. It's how external change affects the class.
- ▶ However, unlike many measures where one end of the scale is good and the other end is bad, instability at either extreme is good, and the middle is what might indicate an issue.
- ▶ **It's measured by the two forms of coupling, incoming dependency, called *afferent coupling*, and outgoing dependencies called *efferent coupling*.**

Efferent coupling

Efferent coupling (denoted by C_e) is a measure of the number of classes this class depends on, i.e., it is a measure of the number of outgoing dependencies of the class or the interrelationships between the classes.



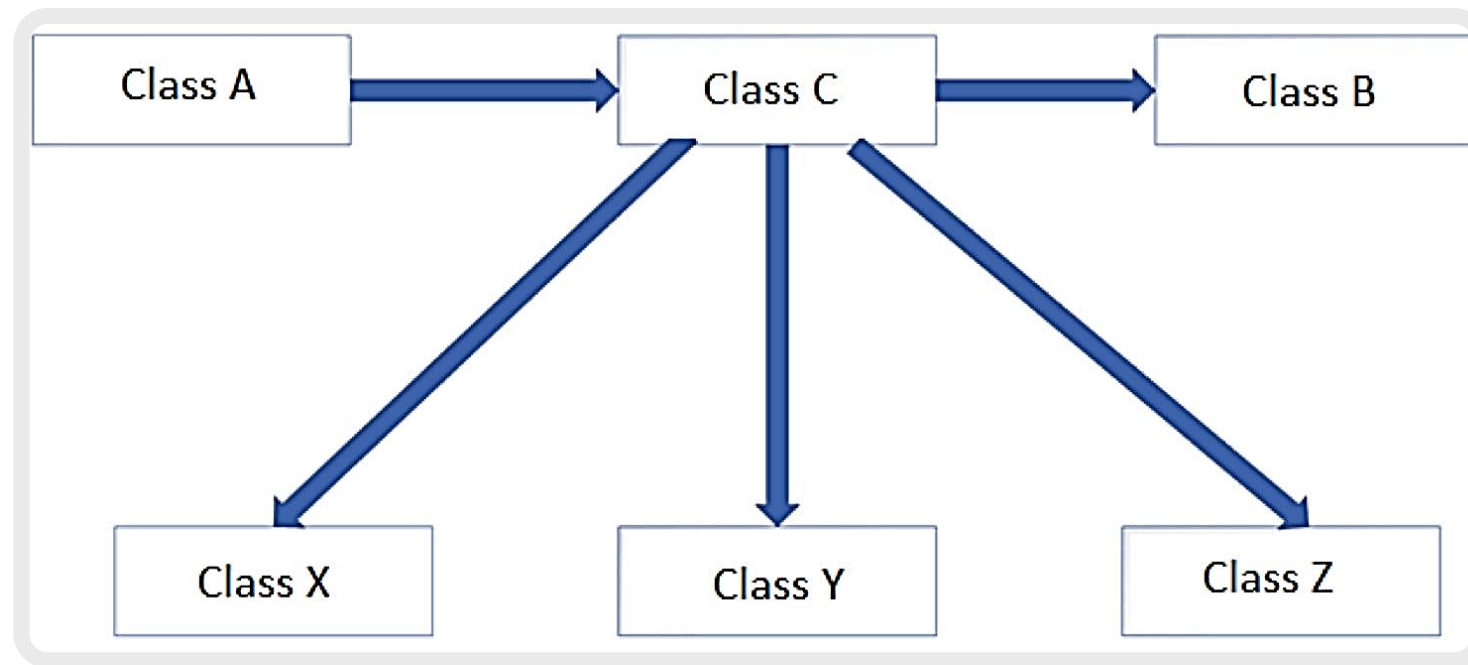
- ▶ As evident from the above figure, class C has four outgoing dependencies (classes B, X, Y, and Z) and one incoming dependency (class A). Hence the value of C_e for class C is 4.



Afferent coupling

Afferent coupling (denoted by C_a) measures the number of classes that depend on or use this class.

The value of C_a for class C is 1.

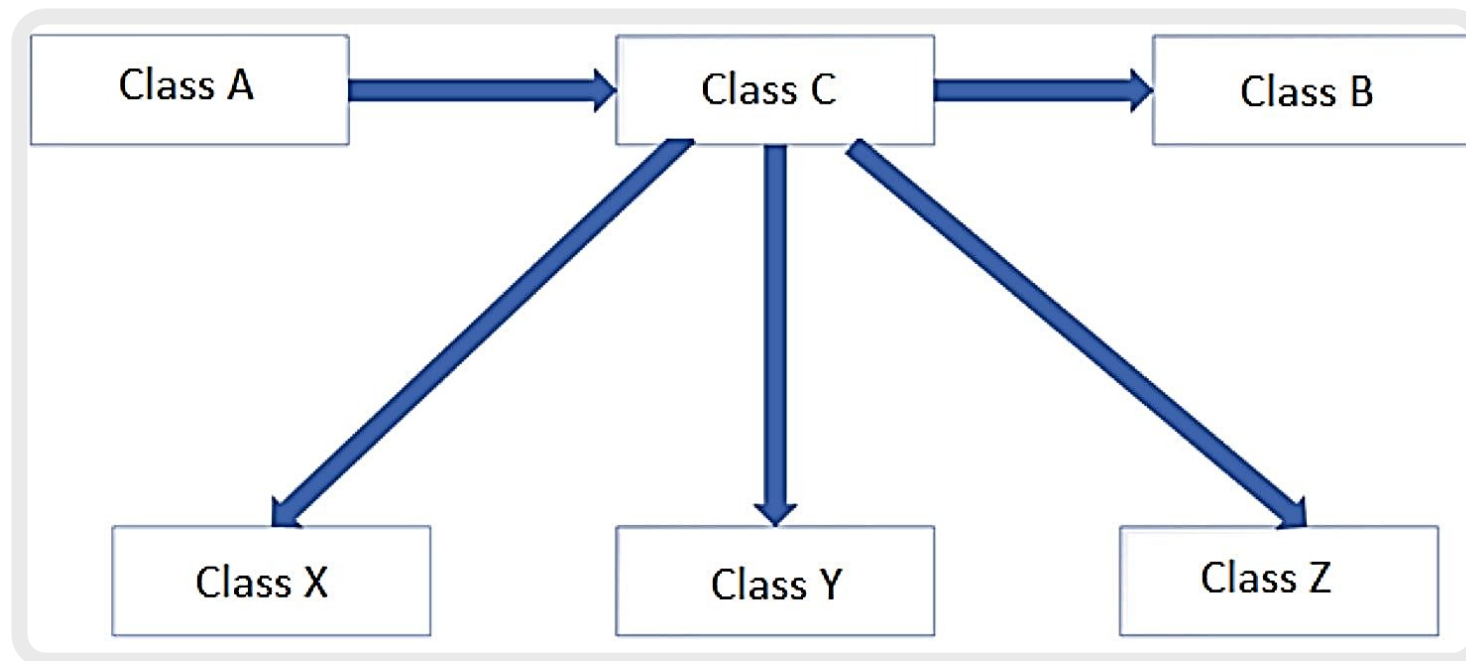


Instability... again

- ▶ Instability is a measure of how change propagates through the system. It's how external change affects the class.
- ▶ However, unlike many measures where one end of the scale is good and the other end is bad, instability at either extreme is good, and the middle is what might indicate an issue.
- ▶ It's measured by the two forms of coupling, incoming dependency, called *afferent coupling*, and outgoing dependencies called *efferent coupling*.
- ▶ Instability is the ratio of outgoing dependencies, that efferent coupling, to all dependencies related to the class.

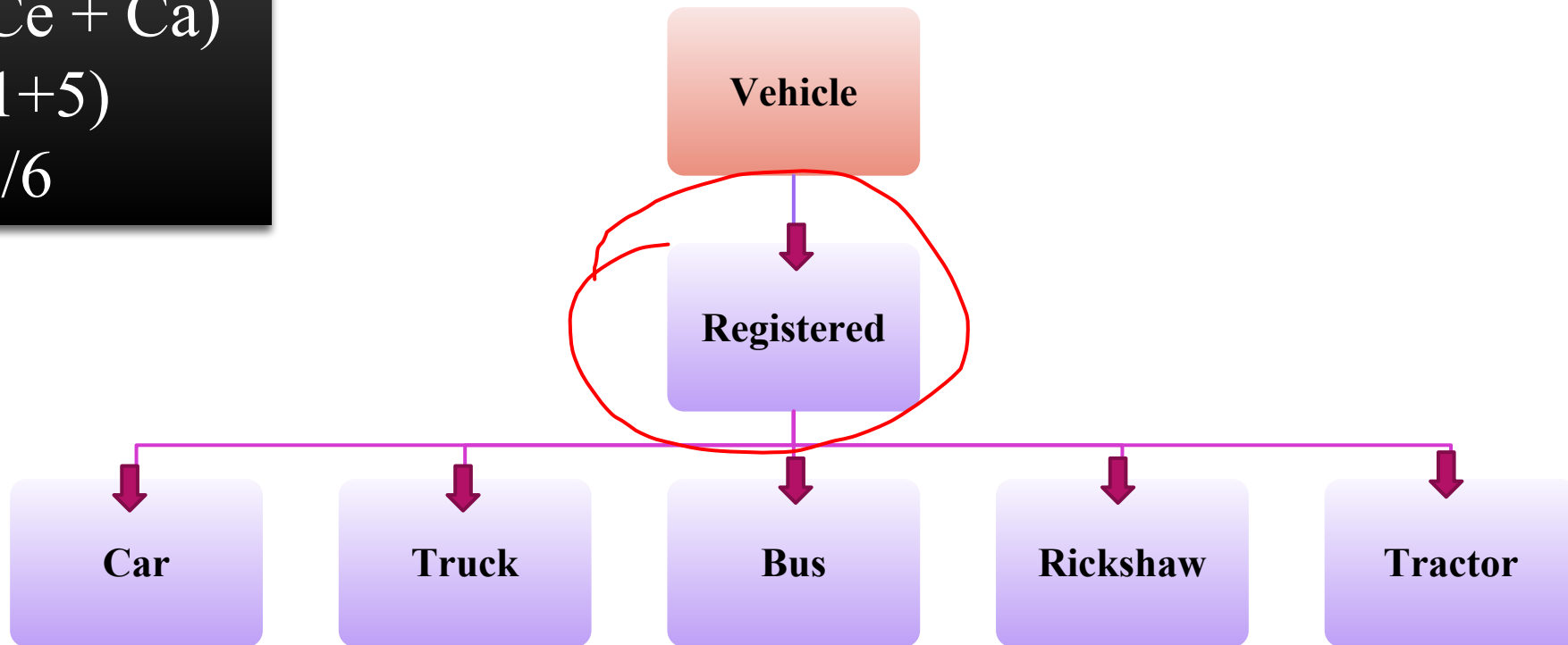
Instability

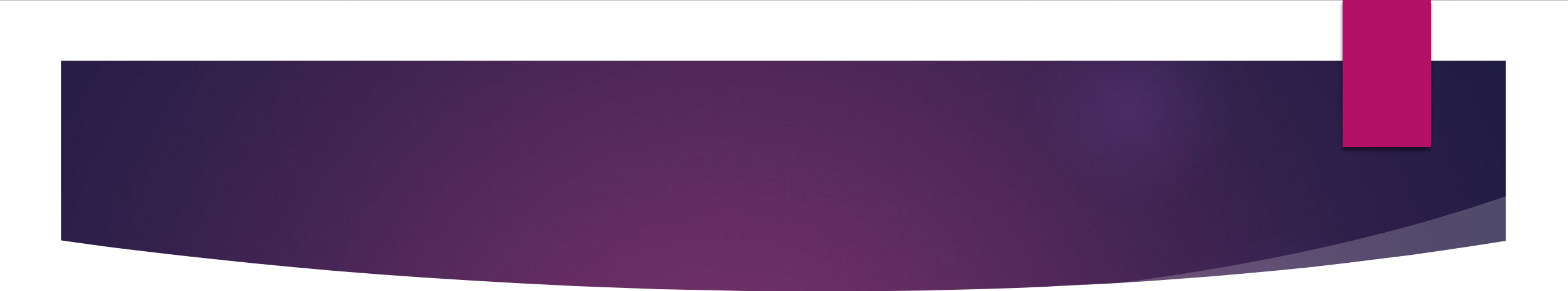
$$I = Ce / (Ce + Ca)$$



High instability

$$I = Ce / (Ce + Ca)$$
$$I = 5 / (1 + 5)$$
$$I = 5/6$$



- 
- ▶ Any change to any of the complex classes in the bottom row might mean change in the registered class. **But this might not necessarily be bad.** What we're doing is making use of this registered class to protect the vehicle code from the numerous changes in the complex classes and processes.
 - ▶ So, rather than vehicle talking directly to all five of those classes, we put the registered in the middle to take care of any of the changes.

- ▶ Imagine you are building a prototype for a new social media platform. In order to rapidly iterate and experiment with different features and user experiences, you may want to design your system with high instability. For example, you may have a module responsible for handling user authentication and another module responsible for displaying user profiles. Both of these modules may have many dependencies on other parts of the system, in order to enable rapid experimentation and iteration.

ANOTHER EXAMPLE

- ▶ By allowing high instability in your system, you can quickly test different user experiences and functionality, without worrying too much about maintaining a stable or maintainable codebase. Once you have settled on the desired functionality and user experience, you can then start to refactor your code and reduce instability, in order to improve the maintainability, reliability, and scalability of your system.

Low instability is generally considered to be a desirable quality, as it can make the system easier to maintain, change, and evolve over time. Low instability is achieved by reducing the number of dependencies that a module or component has on other parts of the system, and by ensuring that changes to one part of the system do not have unintended effects on other parts of the system.

Person

Teacher

Student

Staff

$$I = Ce / (Ce + Ca)$$
$$I = 0 / (0 + 4)$$
$$I = 0 / 4$$

Teaching Assistant

```
graph TD; Person --> TA[Teaching Assistant]; Teacher --> TA; Student --> TA; Staff --> TA;
```

The diagram illustrates a system architecture where four entities (Person, Teacher, Student, and Staff) all depend on a single component, the Teaching Assistant. Each entity is represented by a light blue box at the top, and the Teaching Assistant is represented by a light orange box at the bottom. Four purple arrows point from each of the top boxes down to the Teaching Assistant box. The Teaching Assistant box is enclosed in a dark red oval, and the entire diagram is set against a white background with a dark blue header and footer area.

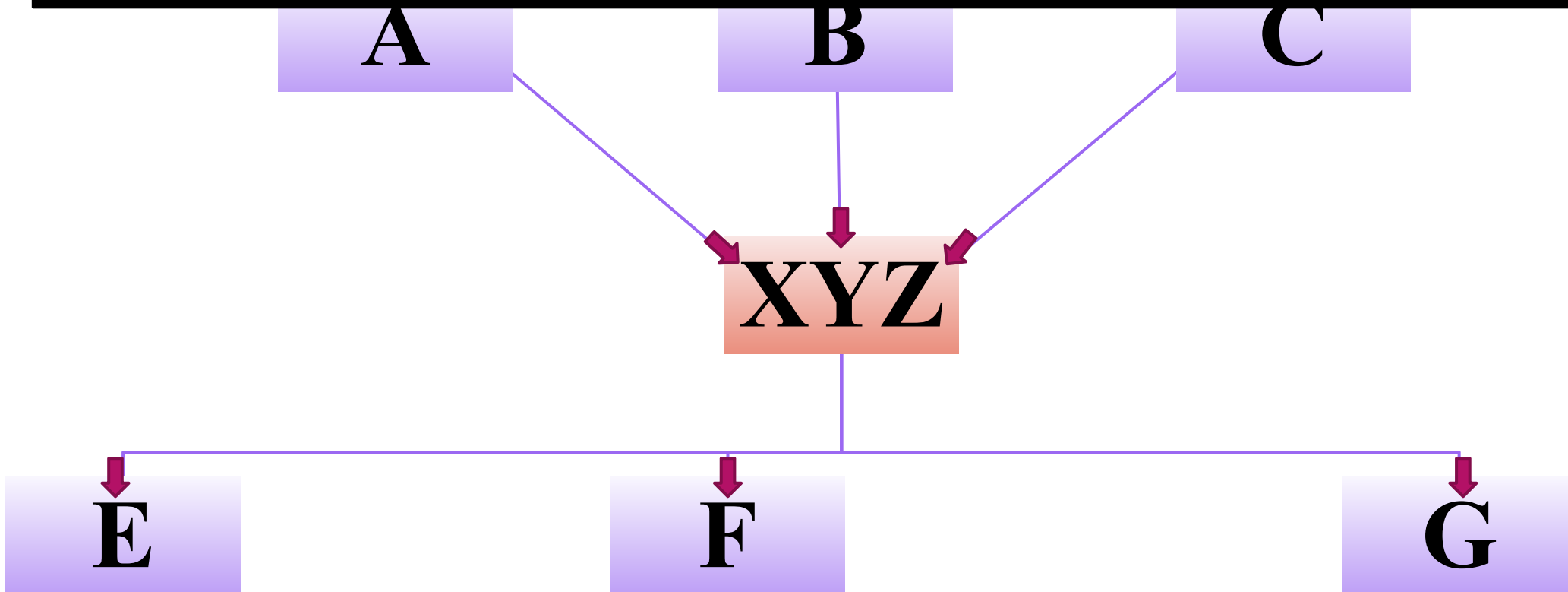
- ▶ Imagine you are designing a system that has a module responsible for calculating the total price of a shopping cart. This module is dependent on several other modules, including a module for retrieving product prices, a module for applying discounts, and a module for calculating tax.

EXAMPLE

- ▶ If the module responsible for calculating the total price has high instability, it means that it has many dependencies on other parts of the system. For example, if the module for retrieving product prices changes, it may have unintended effects on the module responsible for calculating the total price, and require significant changes to the codebase.
- ▶ On the other hand, if the module responsible for calculating the total price has low instability, it means that it has fewer dependencies on other parts of the system. For example, if the module for retrieving product prices changes, it may have little to no impact on the module responsible for calculating the total price, and require minimal changes to the codebase.
- ▶ By reducing the number of dependencies between the different modules, and ensuring that the module responsible for calculating the total price has low instability, you can make the system easier to maintain, modify, and evolve over time. This can lead to a more flexible and adaptable system, that can be easily tested and updated to meet changing business requirements.

Similar amount of AF & EC

It may be more practical to focus on reducing the overall coupling between different parts of the system, rather than trying to achieve perfect balance.



```
<?php
```

```
// This function calculates the area of a  
rectangle
```

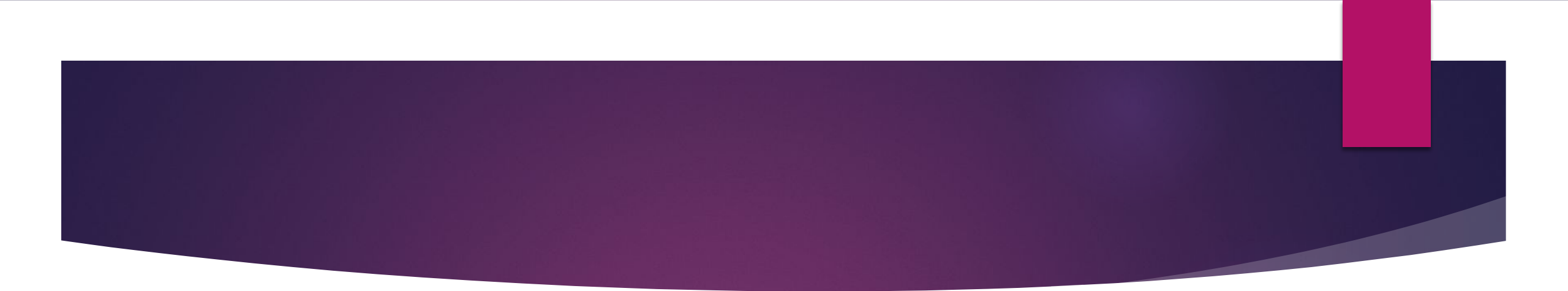
```
function calculate_area($width, $height) {  
  
    return $width * $height;  
  
}
```

```
// This function calculates the perimeter of a  
rectangle
```

```
function calculate_perimeter($width,  
$height) {  
  
    return 2 * ($width + $height);  
  
}
```

```
// This function prints the dimensions and  
properties of a rectangle
```

```
function print_rectangle($width, $height) {  
  
    $area = calculate_area($width, $height);  
  
    $perimeter = calculate_perimeter($width,  
$height);  
  
    echo "Width: $width\n";  
  
    echo "Height: $height\n";  
  
    echo "Area: $area\n";  
  
    echo "Perimeter: $perimeter\n";  
  
}  
  
?>
```

- 
- ▶ In this example, we have **three functions** that perform related tasks of calculating the area and perimeter of a rectangle, and printing its properties.
 - ▶ The **print_rectangle** function is tightly coupled with the **calculate_area** and **calculate_perimeter** functions. Any changes to these functions would require updates to the **print_rectangle** function as well. This tight coupling can lead to maintenance problems and increase the risk of introducing bugs in the software. If we were to add more functionality related to rectangles, the code would become more complex, harder to read, and harder to maintain.

- 
- ▶ An attempt to reduce coupling

```
<?php
```

```
// This function calculates the area of a  
rectangle
```

```
function calculate_area($width, $height) {  
  
    return $width * $height; }  

```

```
function calculate_perimeter($width, $height) {  
  
    return 2 * ($width + $height); }  

```

```
// This function returns an array containing the  
dimensions and properties of a rectangle
```

```
function get_rectangle_info($width, $height) {  
  
    $area = calculate_area($width, $height);  
  
    $perimeter = calculate_perimeter($width,  
$height);
```

```
return array(  

```

```
    'width' => $width,  

```

```
    'height' => $height,  

```

```
    'area' => $area,  

```

```
    'perimeter' => $perimeter ); }  

```

```
// This function prints the dimensions and properties of a  
rectangle
```

```
function print_rectangle_info($rectangle_info) {  

```

```
    echo "Width: {$rectangle_info['width']}\n";  

```

```
    echo "Height: {$rectangle_info['height']}\n";  

```

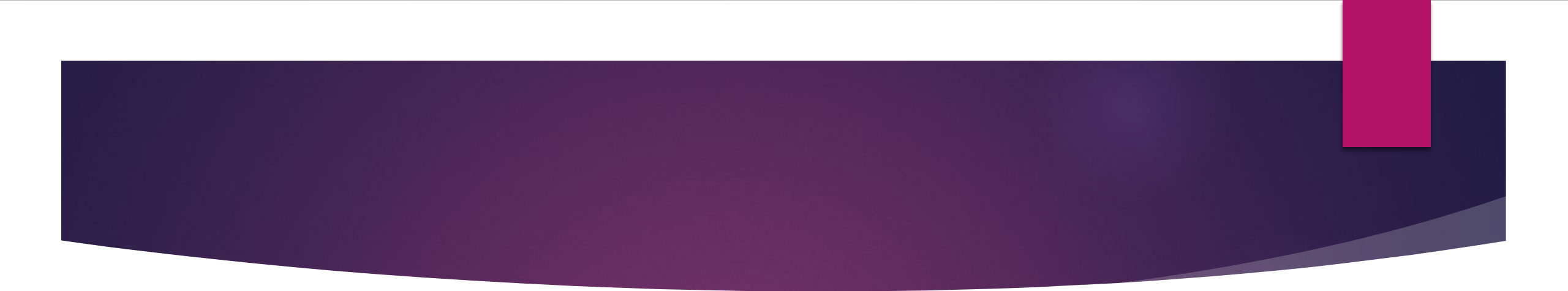
```
    echo "Area: {$rectangle_info['area']}\n";  

```

```
    echo "Perimeter: {$rectangle_info['perimeter']}\n";  

```

```
} ?>
```

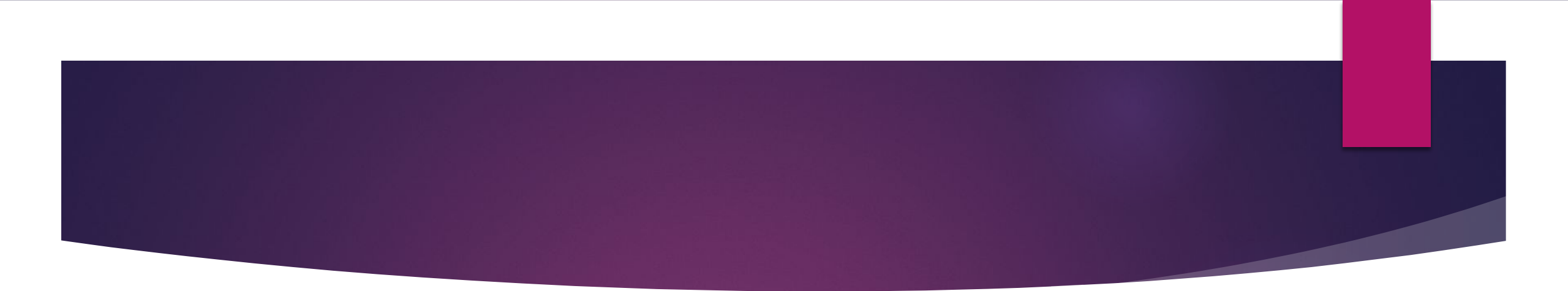
- 
- ▶ In this modified code, we added a new function called `get_rectangle_info` that calculates the area and perimeter of a rectangle and returns an array containing its properties. We also modified the `print_rectangle_info` function to accept the array returned by `get_rectangle_info` as an argument and print its properties.
 - ▶ By doing this, we reduced the coupling between these functions, making the code more modular, easier to read, and easier to maintain.

Measuring abstractness

- ▶ The degree of abstraction of a module or a component is also an indicator of software quality. The ratio of abstract types (i.e., abstract classes and interfaces) in a module or component to the total number of classes and interfaces indicates its degree of abstraction.
- ▶ This metric has a range from 0 to 1. If the value of this metric is zero, then it indicates an entirely concrete component or module. And if the value is one, it indicates that the module or component being analyzed is entirely abstract.

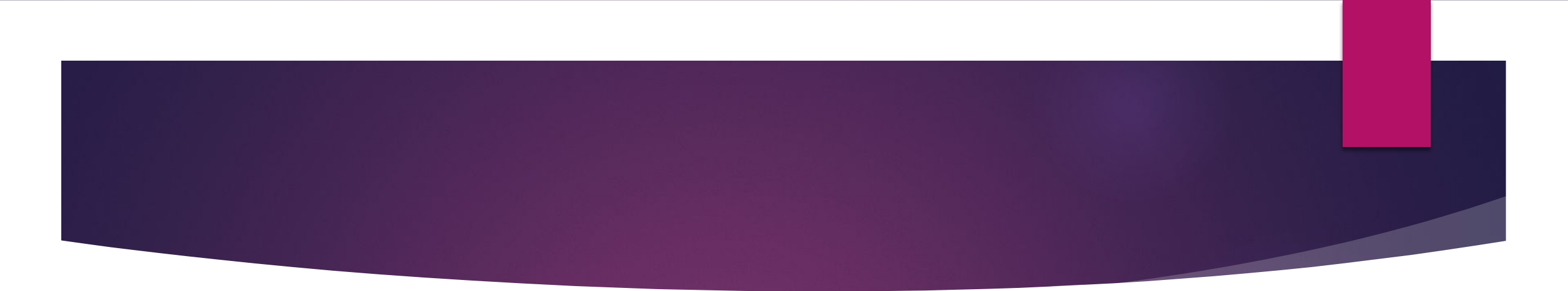
Measuring abstractness....

- ▶ Let us now suppose that T_a is the number of abstract classes present in a component or module, and T_c is the number of concrete classes. Then the degree of abstractness denoted by A is given by the following equation:
- ▶ $A = T_a / (T_a + T_c)$

- 
- ▶ The quest to develop software with high cohesion and low coupling increases the burden on software developers. It can also increase the complexity of the software application. The pursuit of low coupling in a software application must always be balanced against these other considerations.
 - ▶ By taking advantage of the above metrics for coupling, instability, and abstractness, you can work to improve the quality of your applications without introducing too much complexity in the software or putting undue stress on your developers.

The Abstractness Metric (A)

- ▶ The value of the Abstractness Metric can range from 0 to 1, where a value of 0 means that the component has no abstract classes or interfaces, and a value of 1 means that the component contains only abstract classes and interfaces, and no concrete types.
- ▶ If the value of the Abstractness Metric is less than 0, it indicates that the component has more concrete types than abstract types, which means that the component is less abstract. This is typically not desirable, as highly concrete components tend to be more tightly coupled and less flexible, making them more difficult to modify and maintain over time.

- 
- ▶ $A = \text{Number of abstract classes and interfaces} / \text{Total number of types}.$
 - ▶ I is instability value
 - ▶ The Abstractness vs Instability (A/I) metric can be calculated by dividing the Abstractness Metric by the Instability Metric. Components with a high A/I metric are considered to be highly abstract and stable, while those with a low A/I metric are considered to be less abstract and less stable.

```
<?php
```

```
// Define a sample abstract class
```

```
abstract class Animal {  
    public abstract function makeSound();  
}
```

```
// Define a sample concrete class that implements the  
abstract class
```

```
class Dog extends Animal {  
    public function makeSound() {  
        echo "Woof! \n";  
    }  
}
```

```
// Define a sample interface
```

```
interface Vehicle {  
    public function start();  
    public function stop();  
}
```

```
// Define a sample concrete class that implements  
the interface
```

```
class Car implements Vehicle {  
    public function start() {  
        echo "Starting the car! \n";  
    }  
    public function stop() {  
        echo "Stopping the car! \n";  
    }  
}
```

```
// Calculate the Abstractness Metric (A)
```

```
$abstractness = (2 / 4); // There are 2 abstract  
classes/interfaces out of a total of 4 types
```

- ▶ The Dog class depends on the Animal abstract class (outgoing dependency).
- ▶ The Car class depends on the Vehicle interface (outgoing dependency).
- ▶ The Animal abstract class is depended on by the Dog class (incoming dependency).
- ▶ The Vehicle interface is depended on by the Car class (incoming dependency).

$$I = C_e / (C_e + C_a)$$

$$C_e = 2$$

$$C_a = 2$$

$$I = 2 / (2 + 2)$$

$$I = 0.5$$

- ▶ The Dog class
- ▶ The Car class
- ▶ The Animal abstract class
- ▶ The Vehicle interface class

There are 2 abstract classes/interfaces out of a total of 4 types.

$A = \text{Number of abstract classes and interfaces} / \text{Total number of types}.$

So

$$A = 2/4$$

$$A = 0.5$$

(A/I) metric

- ▶ (A/I) metric ?

$$I = Ce / (Ce + Ca)$$

$$Ce=2$$

$$Ca=2$$

$$I=2/(2+2)$$

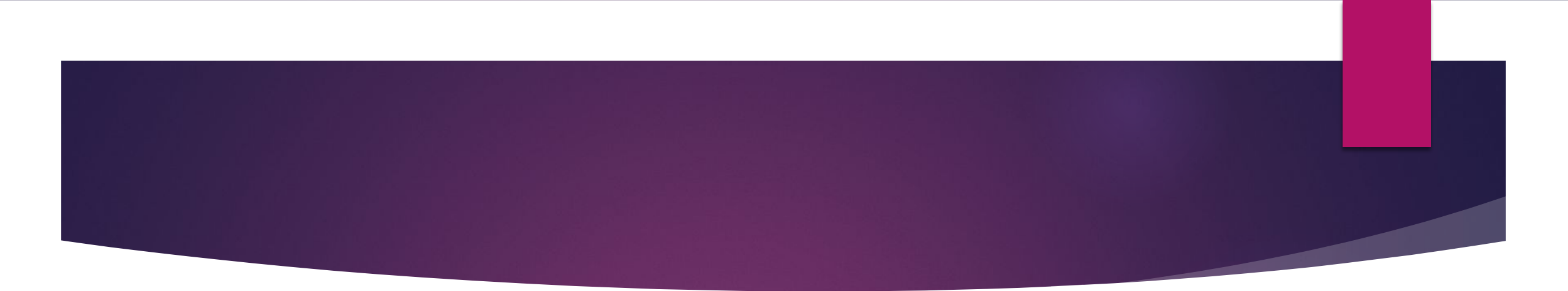
$$I=0.5$$

$$A=2/4$$

$$A= 0.5$$

(A/I) metric

- ▶ The Abstractness vs Instability (A/I) metric ranges from 0 to 1, where 0 indicates a completely unstable and completely concrete system, and 1 indicates a completely stable and completely abstract system.
- ▶ When the A/I metric is less than 1, it indicates that the system is balanced between stability and abstraction.

- 
- ▶ The closer the metric is to 0, the more unstable and concrete the system is, indicating that changes to the system are likely to have a significant impact on other parts of the system. The closer the metric is to 1, the more stable and abstract the system is, indicating that changes to the system are likely to have minimal impact on other parts of the system.
 - ▶ If the A/I metric is greater than 1, it indicates that the system is unbalanced, with an emphasis on abstraction over stability. In this case, changes to the system may be more difficult to make or may have unintended consequences on other parts of the system.

Coupling? Metric Cel? What is this??



EASY

- ▶ A **high A/I metric** value indicates that the system is stable and abstract, with a clear separation of concerns between different components, which can result in better quality software. This is because changes made to one component are less likely to have a significant impact on other parts of the system, which can make the system easier to maintain and evolve over time.
- ▶ On the other hand, a **low A/I metric** value indicates that the system is unstable and concrete, with many interdependencies between different components. This can make the system more difficult to maintain and evolve over time, which can lead to lower quality software.

VERY IMPORTANT

- ▶ In general, a well-designed software system should have a balanced A/I metric, with a reasonable level of abstraction and stability that meets the needs of the system's users and stakeholders.
- ▶ However, it's important to note that the A/I metric is just one of many metrics that can be used to evaluate software quality, and it should be considered in the context of other factors such as functionality, reliability, usability, maintainability, and performance.

2. اَللّٰهُمَّ صَلِّ عَلٰى مُحَمَّدٍ وَعَلٰى آلِ مُحَمَّدٍ كَمَا
صَلَّيْتَ عَلٰى اِبْرَاهِيْمَ وَعَلٰى آلِ اِبْرَاهِيْمَ اِنَّكَ حَمِيْدٌ
مَّجِيْدٌ اَللّٰهُمَّ بَارِكْ عَلٰى مُحَمَّدٍ وَعَلٰى آلِ مُحَمَّدٍ كَمَا
بَارَكْتَ عَلٰى اِبْرَاهِيْمَ وَعَلٰى آلِ اِبْرَاهِيْمَ اِنَّكَ
حَمِيْدٌ مَّجِيْدٌ.

اے اللہ! اپنی رحمت نازل فرما محمد پر اور آل محمد پر جیسا کہ تو نے اپنی رحمت نازل فرمائی ابراہیم پر اور آل
ابراہیم پر۔ بیشک تو تعریف والا اور بزرگی والا ہے۔ اے اللہ! برکت نازل فرما محمد پر اور آل محمد پر جیسا
کہ تو نے برکت نازل فرمائی ابراہیم پر اور آل ابراہیم پر۔ بیشک تو تعریف والا اور بزرگی والا ہے۔