Fakultas Informatika
School of Computing
Telkom University

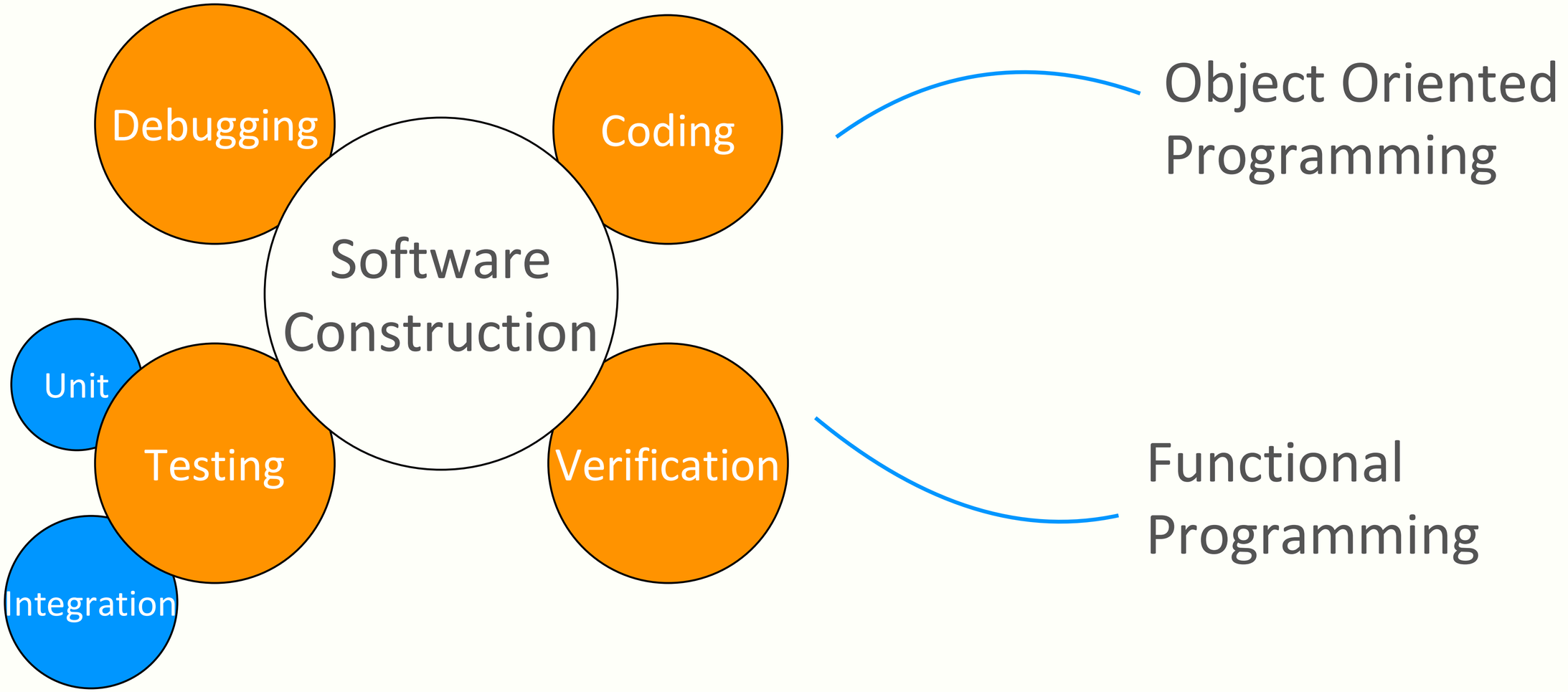
Yayasan Pendidikan Telkom

Object Oriented Programming
Functional Programming

Perancangan Aplikasi Sains Data
(Week 5 – 6)

PROGRAM STUDI SAINS DATA
TELKOM UNIVERSITY

Introduction to OOP and FP



Brief History of OOP and FP

Tracing the Development of Programming Paradigms

Origins of OOP: Developed in the 1960s with Simula; popularized by Smalltalk, C++, and Java

Origins of FP: Rooted in lambda calculus (1930s); adopted in Lisp, Haskell, and modern Python.

Modern Trends: Hybrid approaches integrate OOP and FP (e.g., Python, Scala, and JavaScript)

Understanding The Paradigm

OOP vs. FP: Understanding the Distinctions

OOP; State & Objects: OOP focuses on objects that store state and behavior, making it ideal for modular applications

FP; Pure Functions & Immutability: FP emphasizes stateless functions and immutable data, enabling parallelism and predictability

Common Misconceptions: OOP is not always slower than FP, and FP can be used for complex applications, not just small scripts

Object Oriented in Data Science

Building Reusable and Scalable Code

Core OOP concepts: Encapsulation, inheritance, polymorphism

OOP in Data Science Enhances **code modularity**, **reusability**, and **maintainability** in machine learning pipelines.

Example: Scikit-learn; Scikit-learn models use OOP principles (e.g., fit), predict).

Functional in Data Science

A Declarative Approach to Data Processing

Core FP concepts: Referential Transperency, immutability, higher-order functions.

Facilitates **parallelism**, **reduces side effects**, and **simplifies complex transformations**.

Example: Pandas & PySpark ; Data manipulation using map()', 'filter()', 'apply()' for scalable processing.

Functional

Principles

Immutability

Referential Transperency

Higher Order Functions

Functional

Immutability

In functional programming, everything is immutable. This means that **once a variable is set, its value cannot be changed**. If `x=3` at the beginning of a program, x will always have the value of 3 for the rest of the program.

```
x = -5
y = x + 1
```

This code have immutable concept . Why?. Give another example!

So...

How do we get any work done if we can't change any variable?

Functional

Referential Transperency

In functional programming, referentially transparent if we can replace it with its value anywhere in the code. If you can replace the call of a function with its actual value, then the function is referentially transparent

```
def add(a,b):
    return a + b
Print(add(2,3))
```

This code have referential transperency concept . Why?. Give another example!

Functional

Higher Order Function

In Functional Programming, functions are first-class citizens, meaning they can be passed as arguments, returned from other functions, and stored in variables. A higher order function is a function that takes another function as an argument or returns a function. Higher order functions also allow us to minimise duplicate code.

```
def apply_twice(func, x):
    return func(func(x))

def increment(num):
    return num + 1

print(apply_twice(increment, 3))
```

This code have referential Higher Order Function concept . Why?. Give the results and another example!

Functional Task

Analyze this code and explain the concept of functional programming we discussed before (Immutability, referential transparency, and higer-order functions)!

```
def increment_list(numbers):
    return [num + 1 for num in numbers]

def square(x):
    return x * x

def apply_function(func, data):
    return [func(x) for x in data]

# Sample data
numbers = [1, 2, 3, 4, 5]

# Apply functions
incremented_numbers = increment_list(numbers)
squared_numbers = apply_function(square, numbers)

# Print results
print("Original:", numbers)
print("Incremented:", incremented_numbers)
print("Squared:", squared_numbers)
```

Home Work

Make the source code that demonstrates Immutability, Referential Transparency and Higer Order Function in single running program.

End Of Week 5