Software Engineering: Basic Programming

Pallat Anchaleechamaikorn

> Technical Coach
> Infinitas by KrungThai
> Arise by Infinitas

yod.pallat@gmail.com
https://github.com/pallat
https://dev.to/pallat

https://go.dev/tour (Thai)
https://github.com/uber-go/guide (Thai)

Preriquisite

install Go

https://go.dev/dl

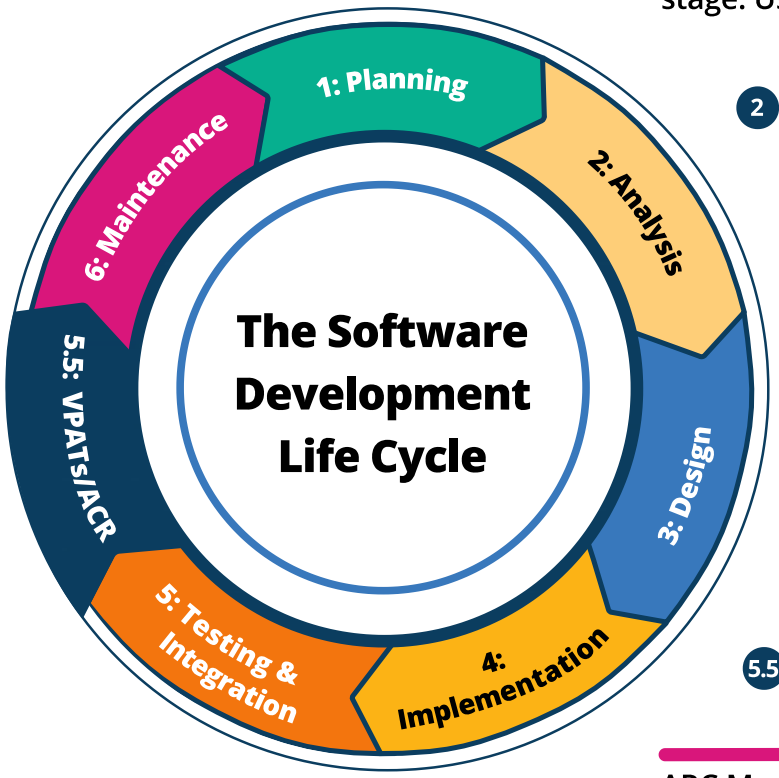
install kotlin

```
brew update
brew install kotlin
```

```
brew install ghc
```

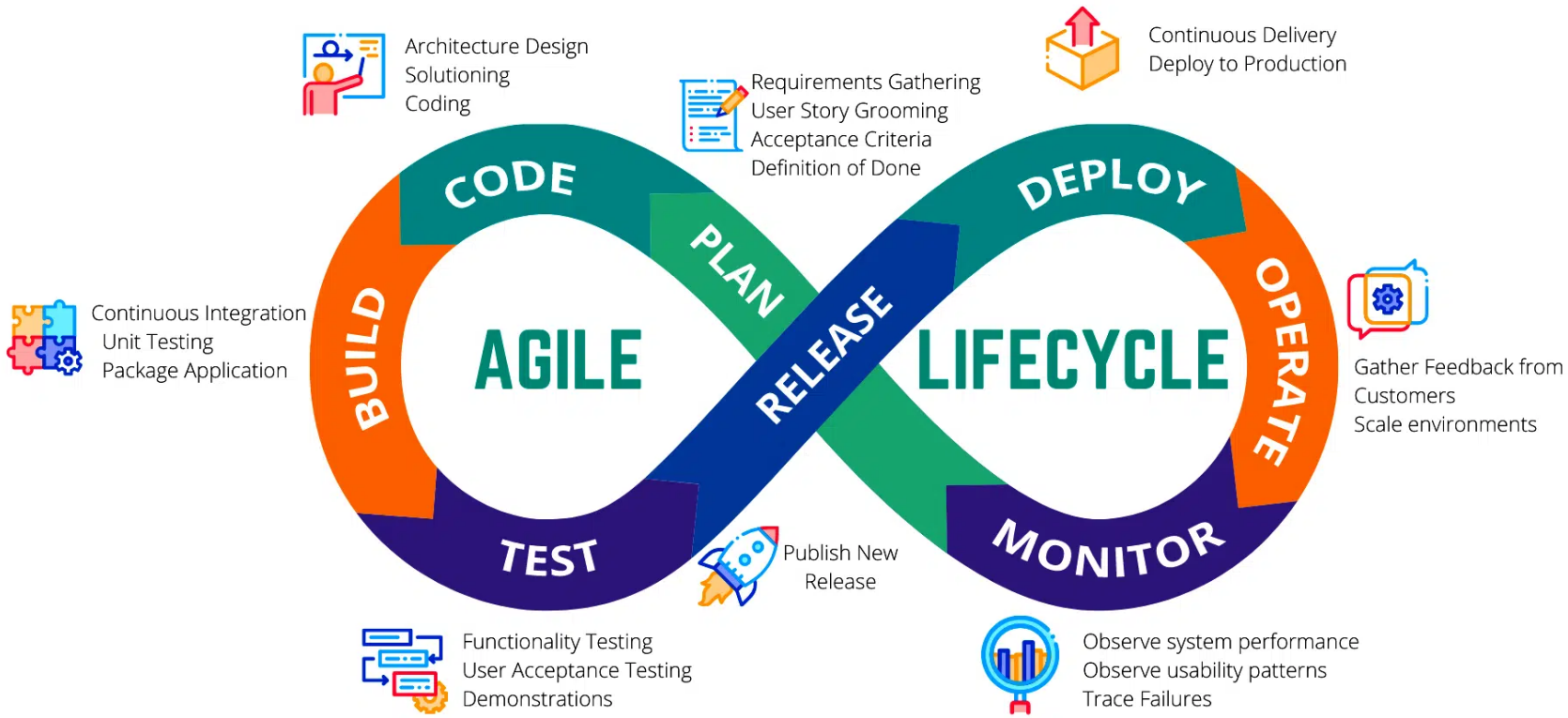
Software Engineering

SDLC

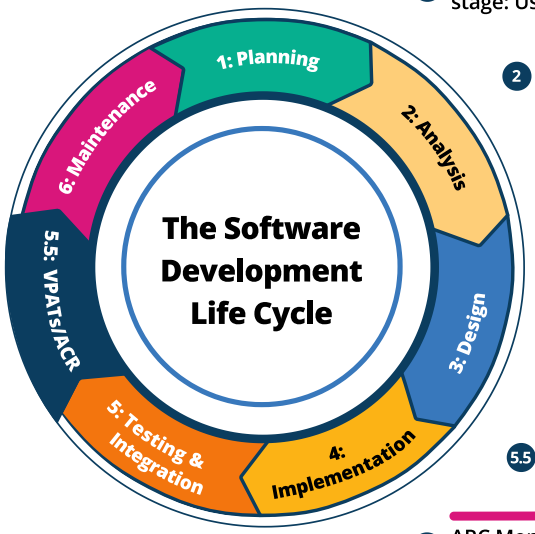


1. Integrating insights into planning stage: User Stories/Code
2. Accessibility Audits: Manual and Machine Detectable
3. Prioritization and integration of inputs into design phase
4. New feature/functionality builds: ARC KnowledgeBase & ARC Tutor, code annotations
5. ARC API Automated Testing and policy compliance
5.5. VPATs/Accessibility Conformance Reports
6. ARC Monitoring Monthly Scanning and User Generated Content

Agile Life Cycle



SDLC / Agile Life Cycle -> SW Engineering



1. Integrating insights into planning stage: User Stories/Code
2. Accessibility Audits: Manual and Machine Detectable
3. Prioritization and integration of inputs into design phase
4. New feature/functionality builds: ARC KnowledgeBase & ARC Tutor, code annotations
5. ARC API Automated Testing and policy compliance

5.5 VPATs/Accessibility Conformance Reports

6. ARC Monitoring Monthly Scanning and User Generated Content

Planing

Analysis

Development

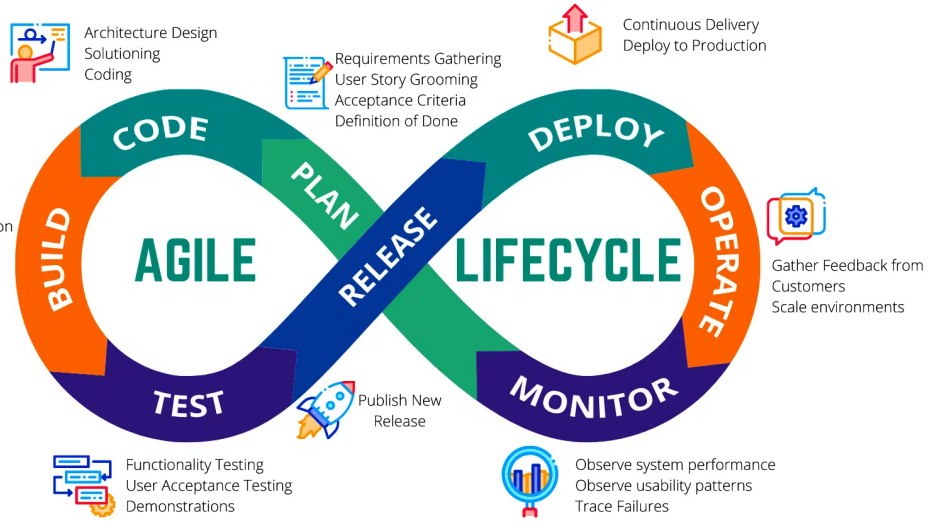
Implementation

Programming

Design

Testing

Validation and Verification



Conclusion

- Design
- Develop
- Secure (Automated)
- Testing (Automated)
- CI/CD (Automated)
- Monitoring (Automated)
- Disaster Recovery Plan
- Reconcile
- etc.

Agile LC

Code -> Code

Build -> Code (ci)

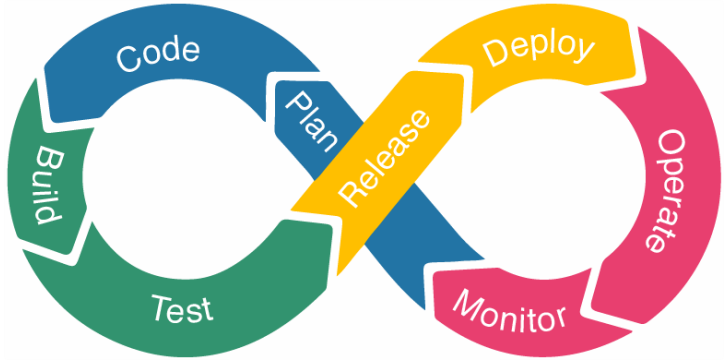
Test -> Code (ci)

Deploy -> Code (cd)

Release -> Code/Manual (cd)

Operate -> Code/Manual (Observe Abilities)

Monitor -> Code/Manual (Observe Abilities)



Principles

TDD

SOLID

KISS

YAGNI

DRY

CUPID

GRASP

KISS YAGNI DRY

https://www.boldare.com/blog/kiss-yagni-dry-principles/

KISS

According to the keep it simple, stupid principle the key to building a successful product is simplicity

1. Users don’t want to waste time. They expect a frictionless experience with straightforward, intuitive user flows, jargon-free naming, and quick results.
2. A simpler software structure makes testing, including also automated testing, easier and more effective.
3. Reduced codebase complexity makes maintenance and onboarding of new team members mid-project easier and faster.

https://todobackend.com/

Over-Engineered ToDo App to learn DDD, Hexagonal Architecture, CQRS, and Event Sourcing

https://blog.bitloops.com/over-engineered-todo-app-to-learn-ddd-hexagonal-architecture-cqrs-and-event-sourcing-74b53a6210fc

YAGNI (You Are Not Gonna Need It)

The main goal of the YAGNI principle is to avoid spending time and money on overengineering things that you think you will need later on

1. Better developer performance: The team focuses on delivering the current requirements effectively. They don’t spend time and effort on guesses.
2. More flexible codebase: You don’t have to find ways to use suboptimal solutions that you had already developed before you had the full picture.

YAGNI reasons

- too much thinking
- common lib
- framework
- edge case
- Over Engineered

DRY

DRY stands for don’t repeat yourself and recommends reducing the repetition of software patterns. By eliminating redundancies in process and logic, engineers lower technical debt and improve the maintainability of the code - both of which are important cost factors, especially in the long term. What’s important, DRY should be applied across the whole system, including not only in the codebase but also in testing and documentation.

DRY?

DRY of Logic

DRY of Process

Comparison of multi-paradigm programming languages

https://en.wikipedia.org/wiki/Comparison_of_multi-paradigm_programming_languages

Go Kotlin JS

Laguage	Number	Paradigm
Go	5	Concurrent,Imperative,Reflection,Pipeline,Generic
Kotlin	8	Concurrent,Functional,Meta,Generic,Imperative,Relflecti
Javascript	4(5)	Functional,Imperative,Relflection,ObjectOriented

The Hello World Collection

http://helloworldcollection.de/

Hello world Go

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
```

Hello world Kotlin

```
fun main() {
    println("Hello, World!")
}
```

Hello world Javascript

```
console.log('Hello, World!');
```

Hello world Huskell

```
main = do
  putStrLn "Hello, World!"
```

Naming Conventions

```
When in Rome, do as the Romans do
```

Java Naming Conventions

Interface start with `I`
ie: `IComparable`, `IEnumerable`

Method name should be in mixed case. Use verbs to describe what the method does
ie: `getSurname`

Constant name should be in uppercase
ie: `DEFAULT_WIDTH`

Kotlin style guide

https://developer.android.com/kotlin/style-guide

```
// Okay
package com.example.deepspace
// WRONG!
package com.example.deepSpace
// WRONG!
package com.example.deep_space

// WRONG!
fun `test every possible case`() {}
// OK
fun testEveryPossibleCase() {}
```

Haskell naming conventions

https://downloads.haskell.org/~ghc/5.02.3/docs/set/sec-library-naming-conventions.html

Actions creating new values arbitrary values of type X have the name newX, e.g. newIORef.

Golang Effective Go

https://go.dev/doc/effective_go

Best practice

The good way to do like them is to see what they did

Basic Function

Odd Even: Go

```go
func isEven(n int) bool {
 return n&1 == 0
}

func isOdd(n int) bool {
 return n&1 == 1
}
```

Odd Even: JS

```
let isEven = (n) => (n & 1) == 0;
let isOdd = (n) => (n & 1) == 1;
```

Odd Even: Haskell

```haskell
isEven :: Integer -> Bool
isEven 0 = True
isEven n = isOdd (n - 1)

isOdd :: Integer -> Bool
isOdd 0 = False
isOdd n = isEven (n - 1)
```

Odd Even: Kotlin

```
fun isEven(value: Int) = value and 1 == 0
fun isOdd(value: Int) = value and 1 == 1
```

FizzBuzz

```
n mod 3 = 0 // "Fizz"
n mod 5 = 0 // "Buzz"
n mod 5 = 0 && n mod 3 == 0 // "FizzBuzz"
Otherwise "n"
```

1 🗣️ "1"

3 🗣️ "Fizz"

5 🗣️ "Buzz"

7 🗣️ "7"

9 🗣️ "Fizz"

15 🗣️ "FizzBuzz"

Go

Haskell

Change

```
change(3,20) = [{10,1},{5,1},{2,1}]
change(2,100) = [{50,1},{20,2},{5,1},{2,1},{1,1}]
change(432,500) = [{50,1},{10,1},{5,1},{2,1}]
```

Change #2

```
price(3).pay(20).change() = [{10,1},{5,1},{2,1}]
price(2).pay(20).change() = [{50,1},{20,2},{5,1},{2,1},{1,1}]
price(432).pay(500).change() = [{50,1},{10,1},{5,1},{2,1}]
```

Change #3

```
pen := product{name: "pen", price: 3}
pay := cash{banknote:20, amount: 1}
sell(pen,pay) = []cash{{10,1},{5,1},{2,1}}

pencil := product{name: "pencil", price: 2}
pay := cash{banknote:20, amount: 1}
sell(pencil,pay) = []cash{{50,1},{20,2},{5,1},{2,1},{1,1}}

calculator := product{name: "calculator", price: 432}
pay := cash{banknote:500, amount: 1}
sell(calculator,pay) = []cash{{50,1},{10,1},{5,1},{2,1}}
```

Homework

https://go.dev/tour

1. Read CSV file convert to JSON file
2. Read json file insert into SQLite
3. Implement API to get information by LastName