Software Engineering Principles

Kasaiah Meruva

14 May 2018

Agenda

Speaker Introduction

Session Goal

1. Software Engineering Principles
2. Thinking About (Coding) the Solution
3. Engineering Best Practices
4. Thinking About the Process

Q & A

Speaker – Introduction

Name: Kasaiah Meruva

Qualification: Master Of Computer Applications

Experience: 14 years of experience in software industry

Working currently as Software Development Manager in PayPal Pte Ltd for last 10 years.

On the job, part of enabling Bank Payments through PayPal in around 40+ new countries

Our goal for this lesson isn't for you to remember all these principles and acronyms...

... but it's for you to know what it means to be an engineer and how engineering "works".

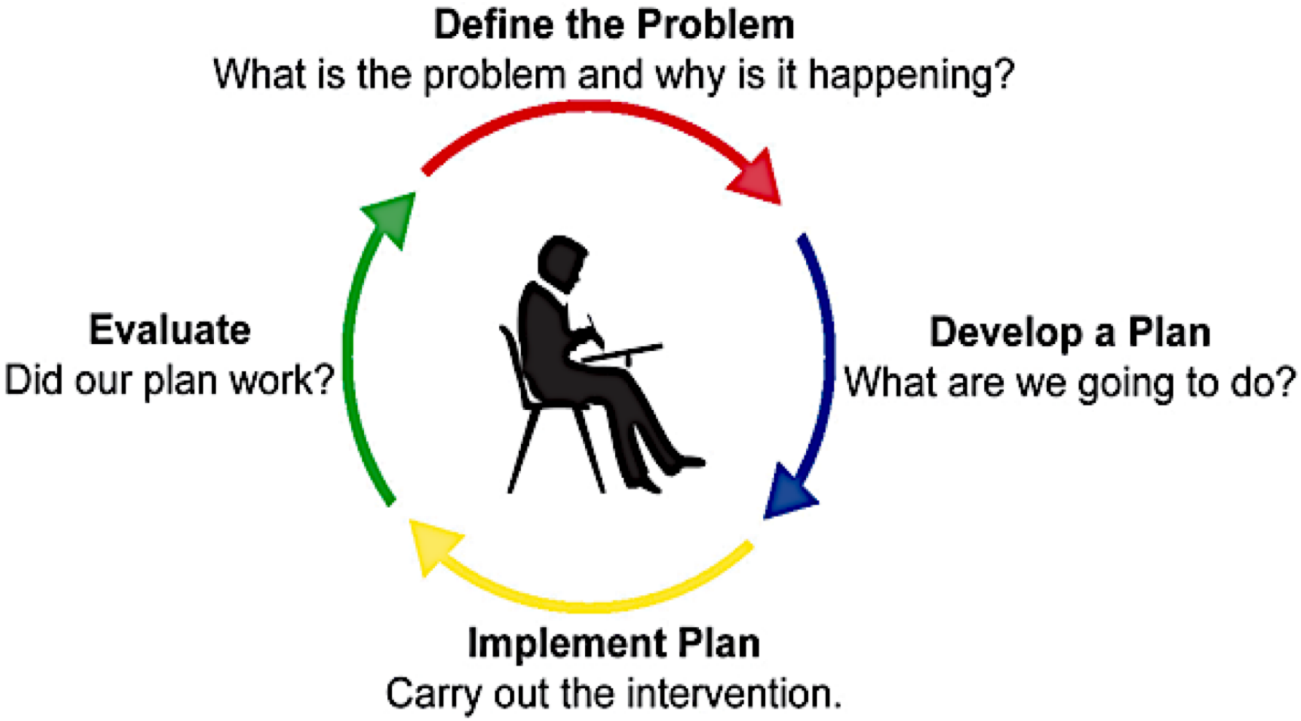
Session Goal

What is Software Engineering?

Software engineering is all about **finding** and **applying** the best ways to solve technical problems with software (which is why it's so much fun).

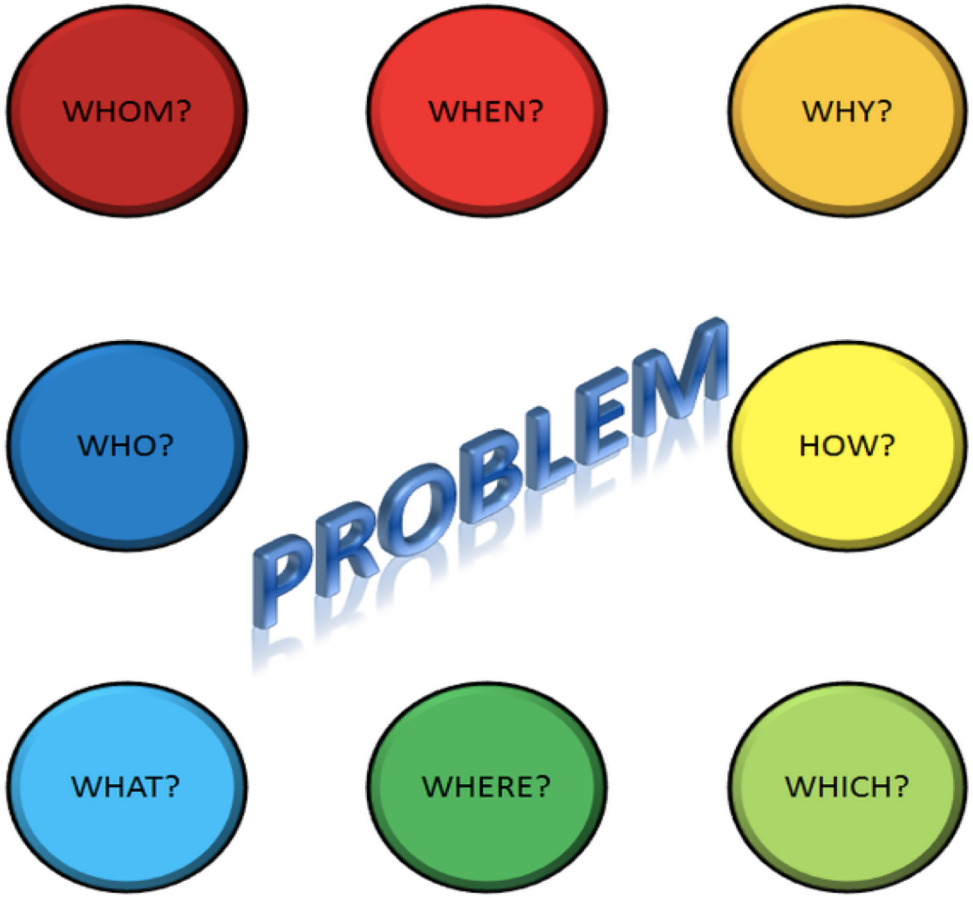
How an Engineer Solves a Problem?

There's no major mystery to engineering – you just apply a systematic process for creatively solving problems. What that looks like is this:



Software Engineering Principles

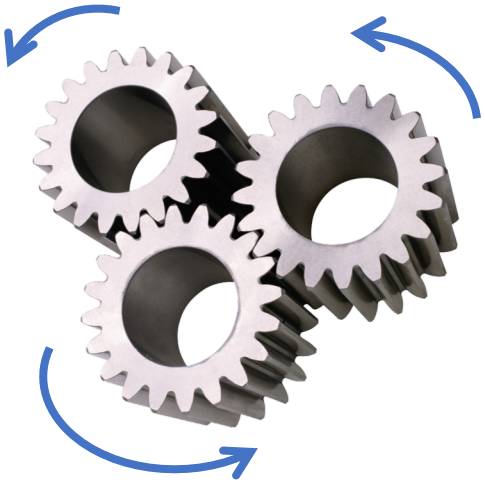
1. Think through the problem



2. Modularization

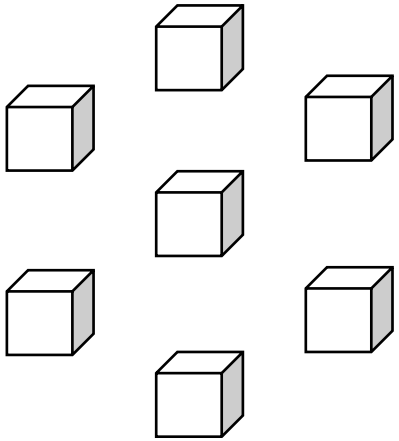
The process of continuous **decomposition** of the software system until **fine-grained components** are created.

Monolithic System



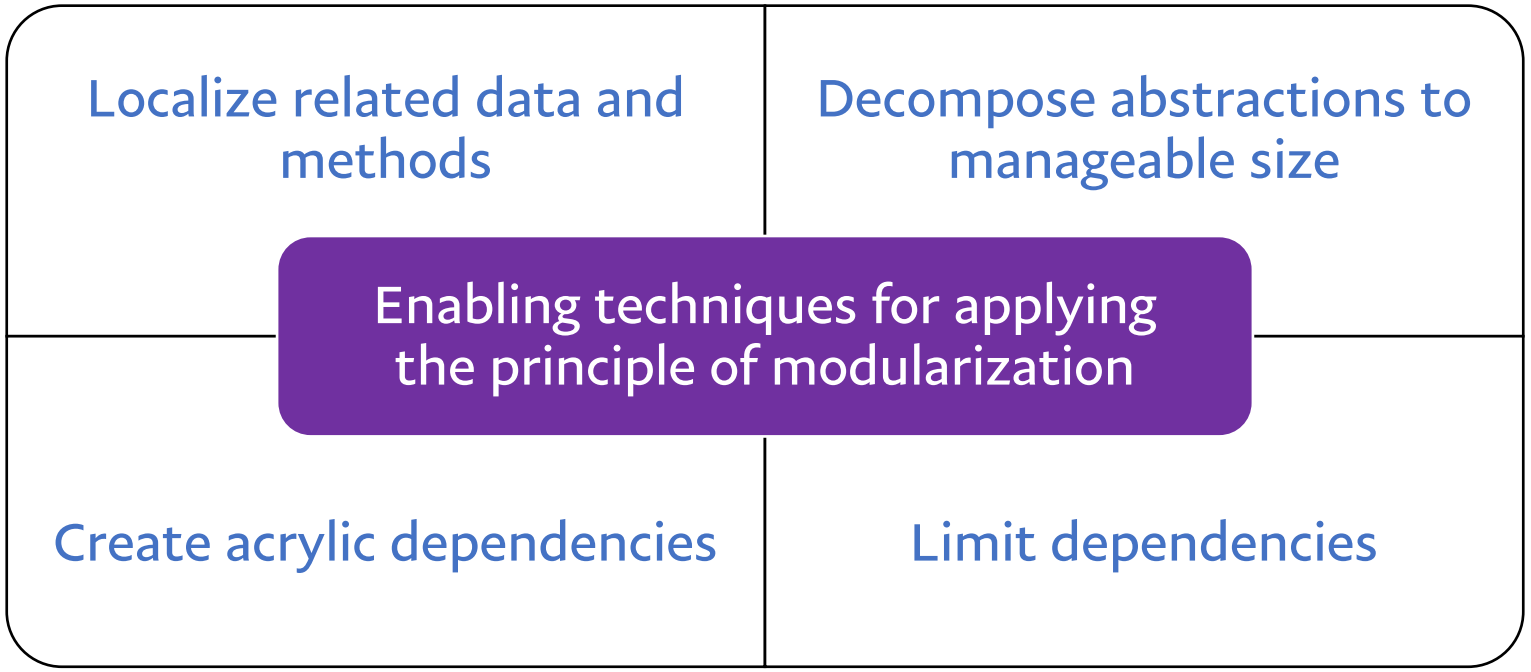
Modularization Engine

Modularized Units



When you modularize the design, you are also modularizing the requirements, code and test cases.

2. Modularization



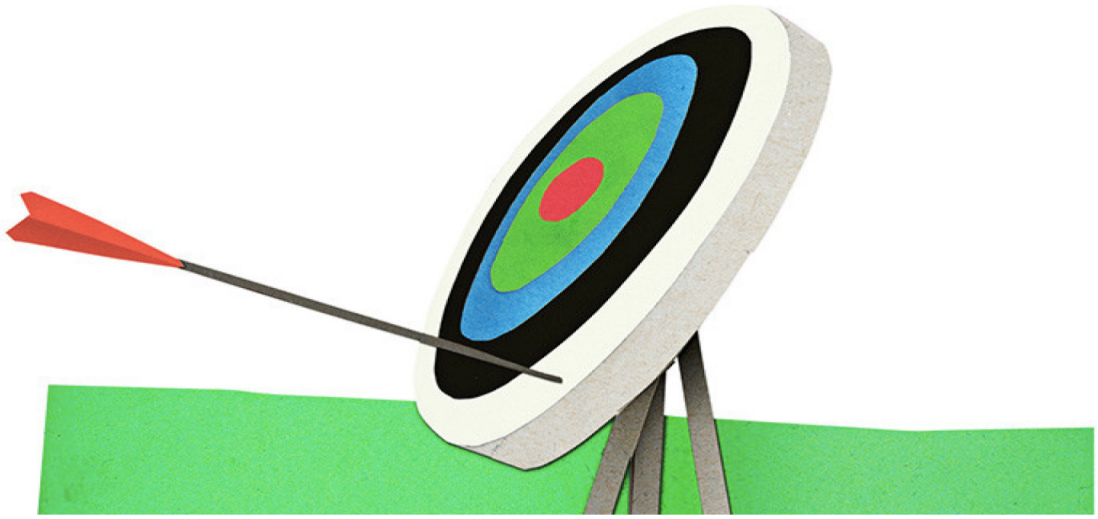
3. KISS (Keep It Simple, Stupid)

Typical violations of KISS

- Over-engineering (need a bike, but build an aircraft)
- Using design patterns when not needed
- Using complex frame work for simple tasks

4. Learn from mistakes

- Accept you will make mistakes – we all make mistakes
- Have the self-confidence to admit to them, take responsibility for them, act on them.
- Learn from them and be courageous about making changes – try to avoid repeating mistakes.



Thinking About (Coding) the Solution

1. YAGNI (You Ain't Gonna Need It!)

2. DRY (Don't Repeat Yourself)

Both code fragments returns the same output. Which code will you prefer?

$$1 + 2 + 3 \ \ldots \ + n = \frac{n(n+1)}{2}$$

```
input = 20000;
sum = 0;

sum = sum + 1;
sum = sum + 2;
    :
    :
sum = sum + 20000;

return sum;
```

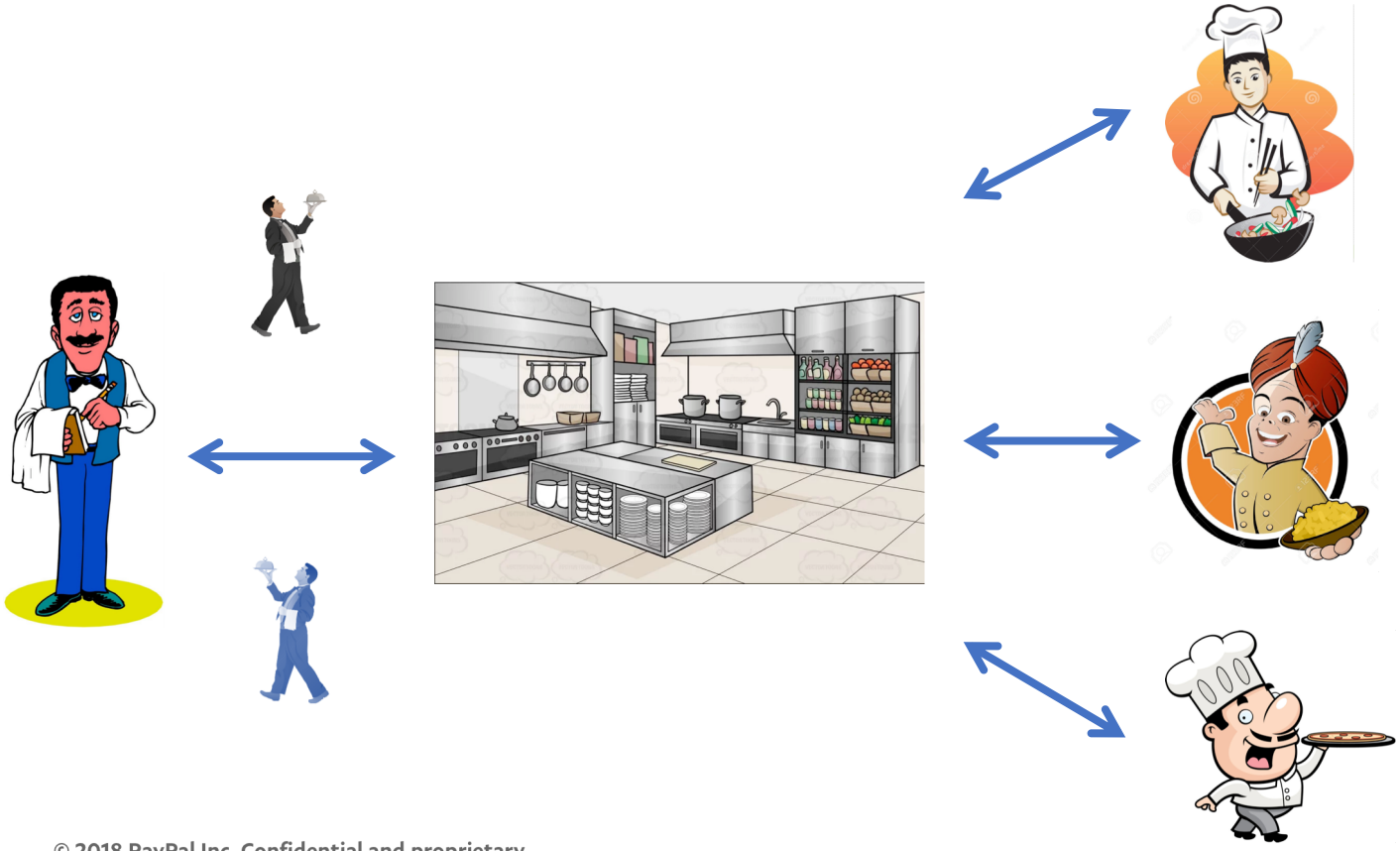
20000 times

```
input = 20000;




return (input * (input + 1)) / 2;
```

3. Embrace abstraction

Example: Abstract Factory Pattern - delegating the object creation to factories

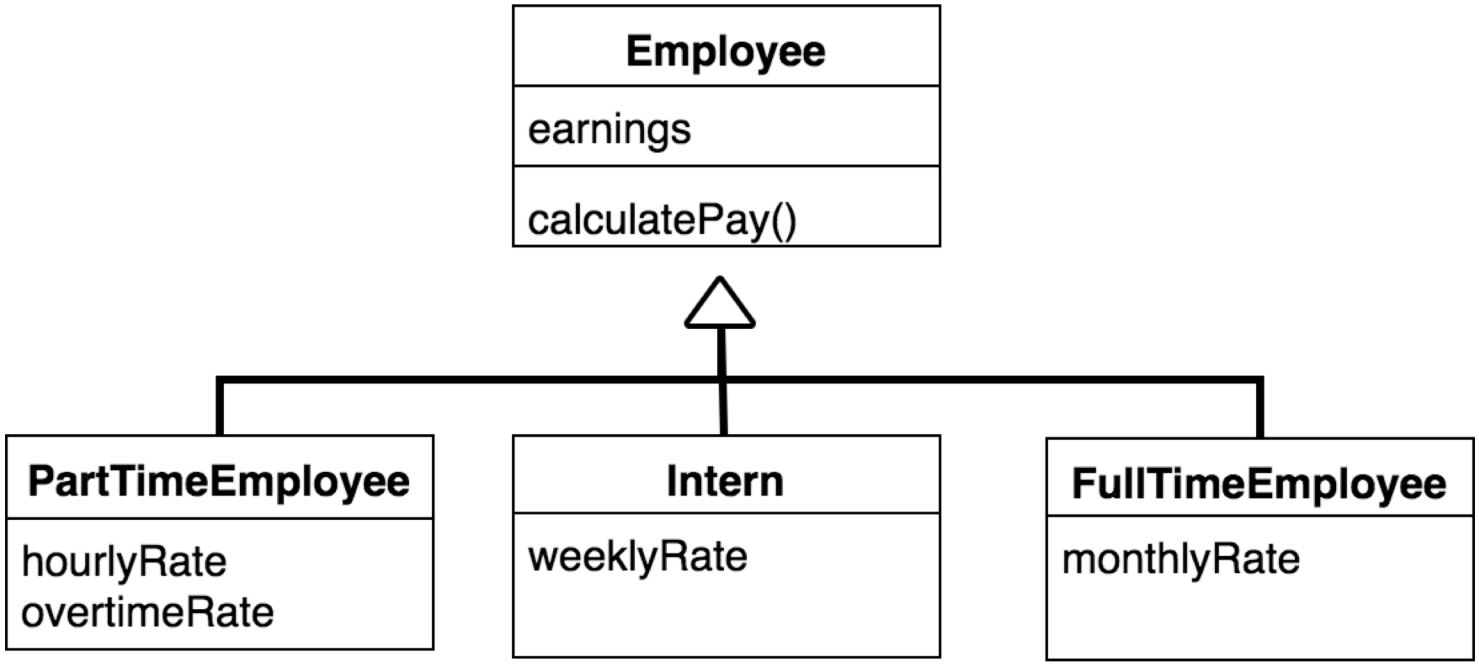
Restaurant orders are received by a kitchen, which are assigned to chefs like Chinese, Indian and Continental



3. Embrace abstraction

Abstract Class: `Employee`

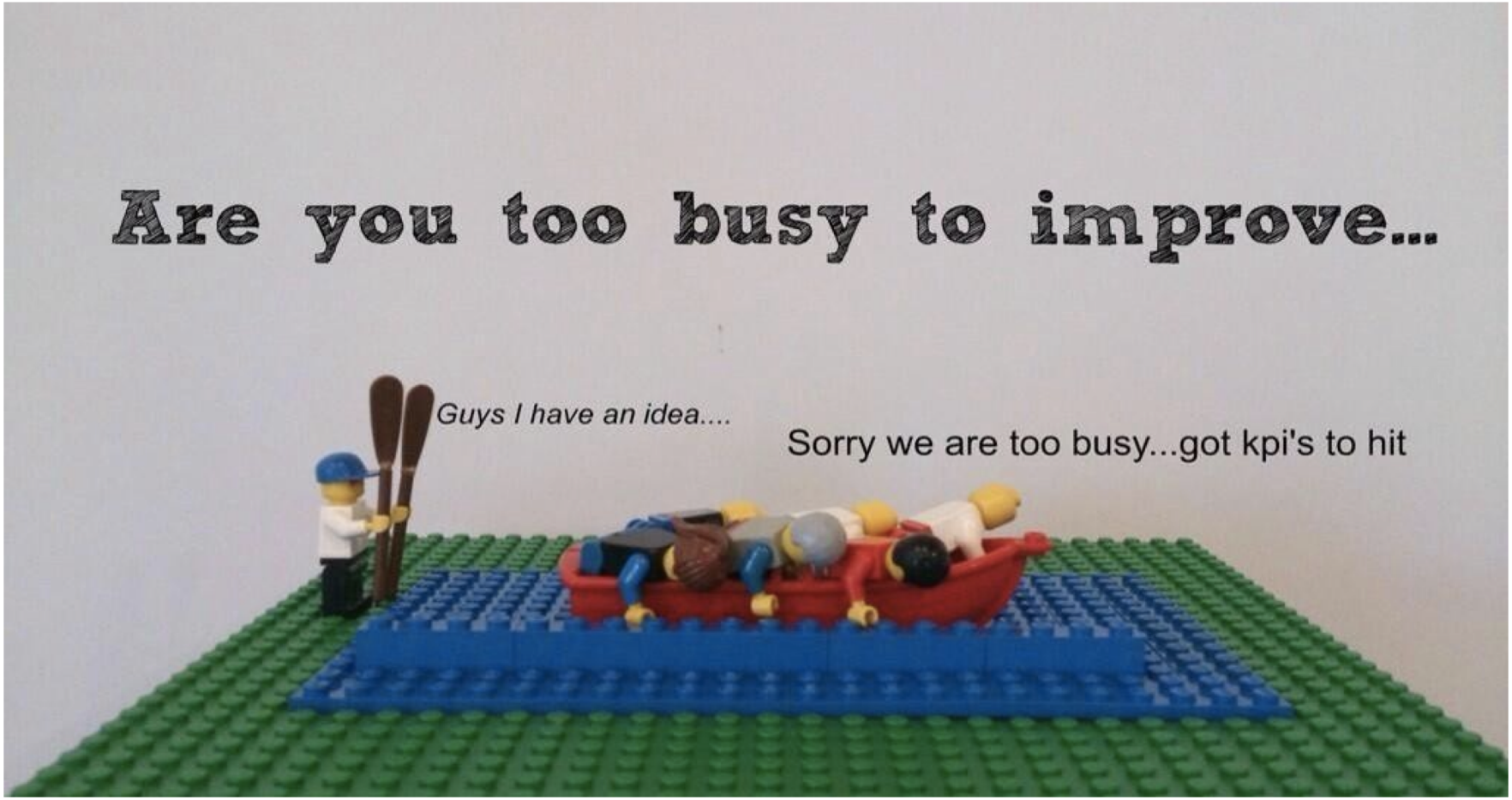
Concrete Class: `PartTimeEmployee`, `Intern`, `FullTimeEmployee`



4. Don’t Re-Invent the Wheel

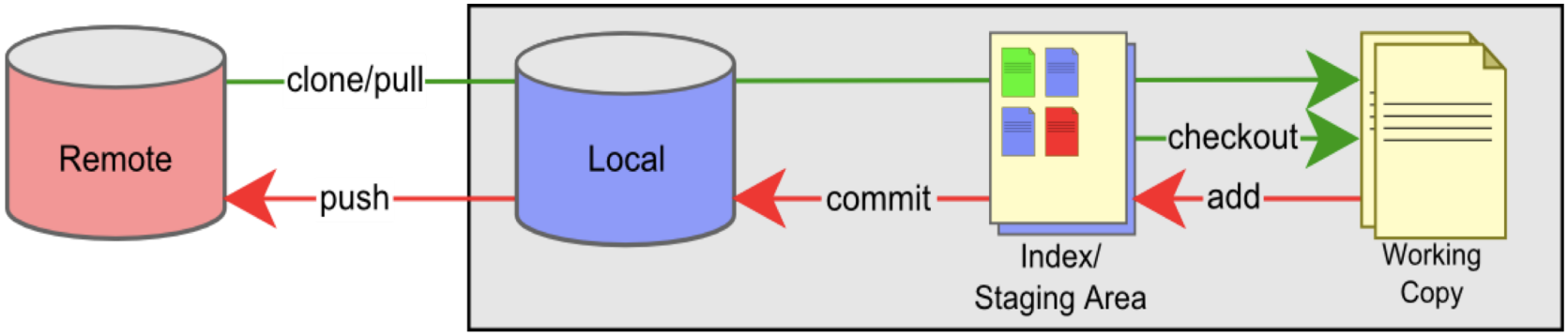
5. Debugging is harder than writing code

6. Kaizen (leave it better than when you found it)



Engineering Best Practices

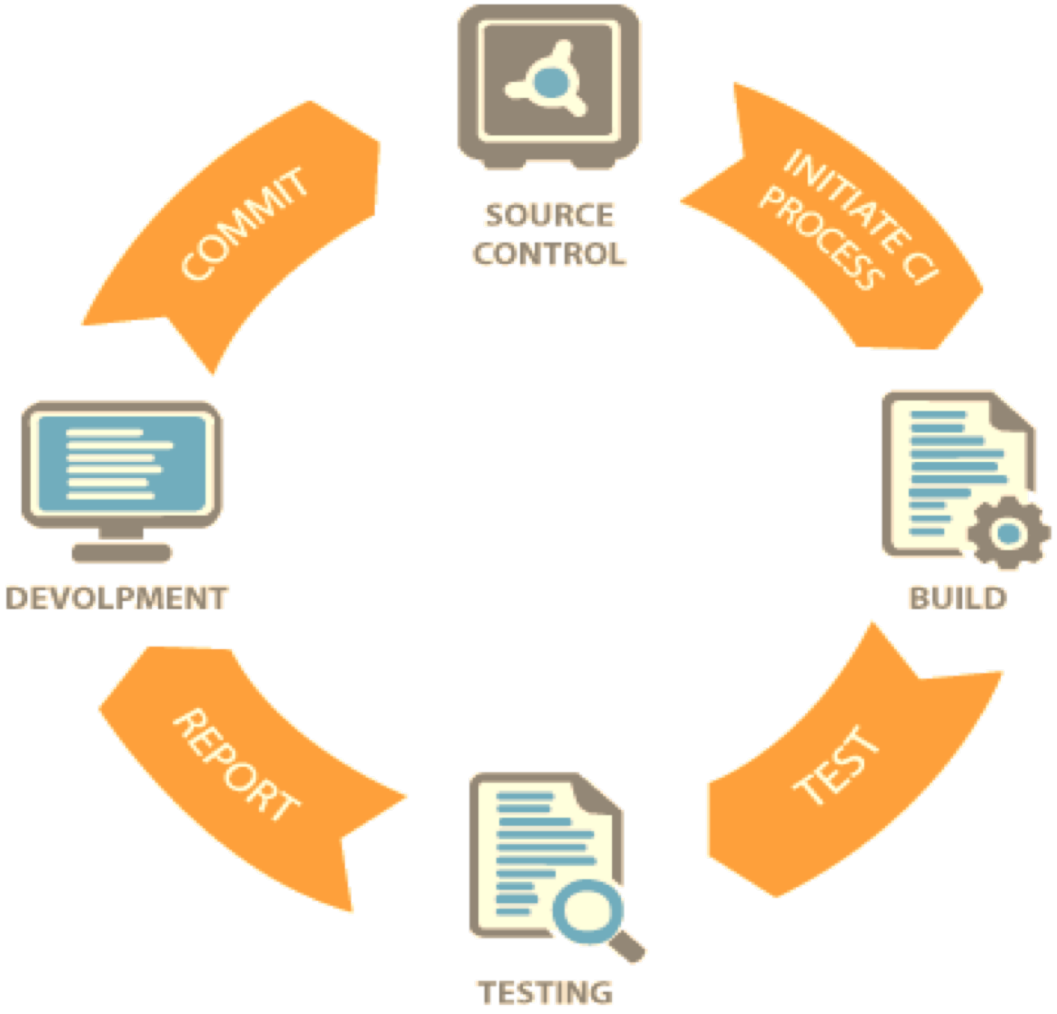
1. Source Control



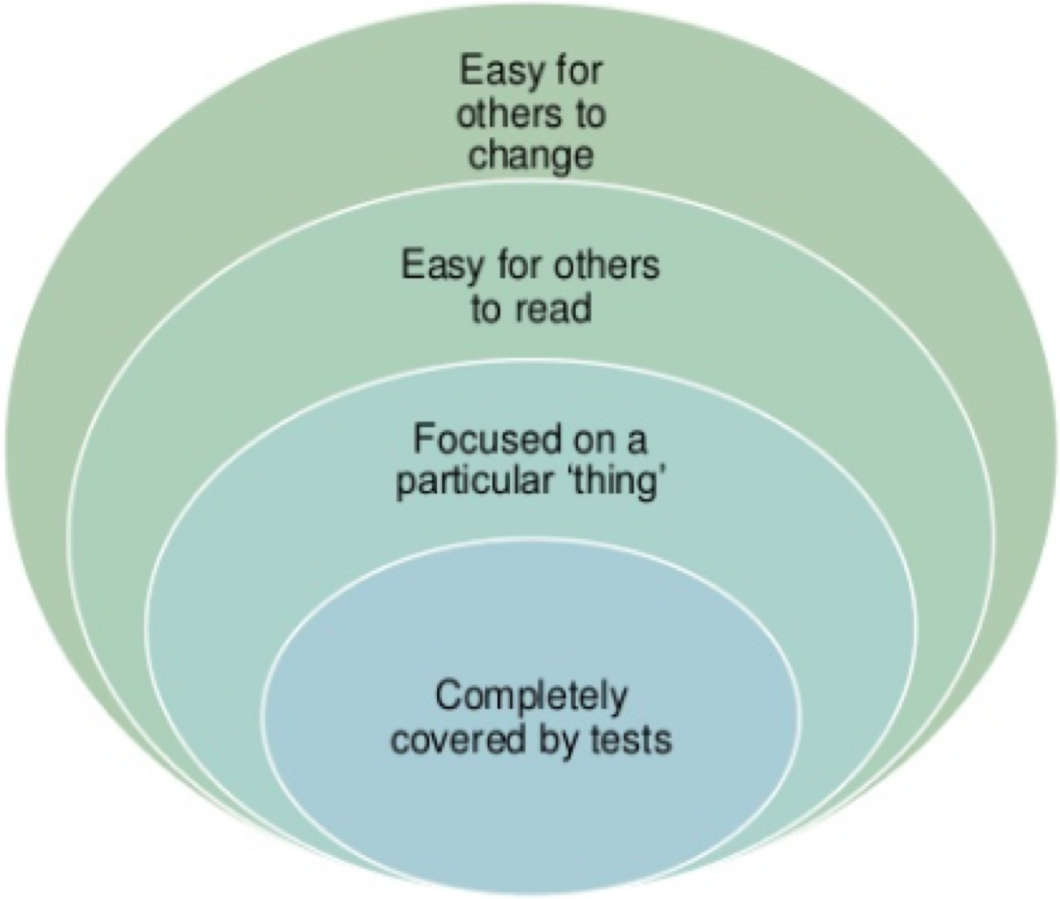
2. Automated Testing

- To get early and instant feedback
- Regression tests only
- Safety net to save time which is reinvested into manual testing
- NOT to replace manual testing

3. Continuous Integration (CI)



4. Clean Code



5. Refactoring

A disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.

5. Refactoring – Code

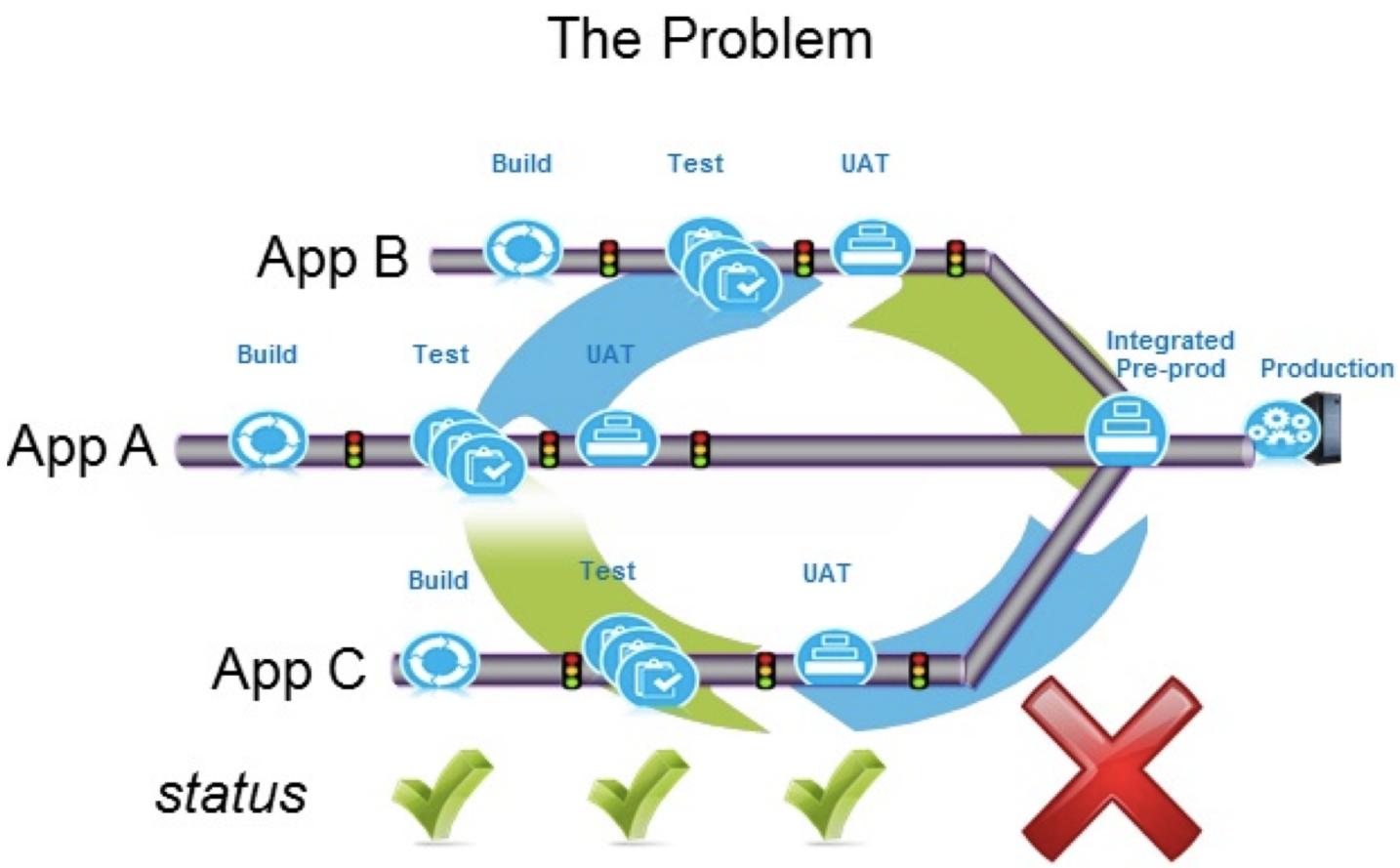
What is the difference between the 2 code segments?

```
void printDebt() {
    printBanner();
    // Print details
    System.out.println("name: " + name);
    System.out.println("amount: " + getOutstanding());
}
```

```
void printDebt() {
    printBanner();
    printDetails(getOutstanding());
}


void printDetails(int amount) {
    System.out.println("name: " + name);
    System.out.println("amount: " + amount);
}
```

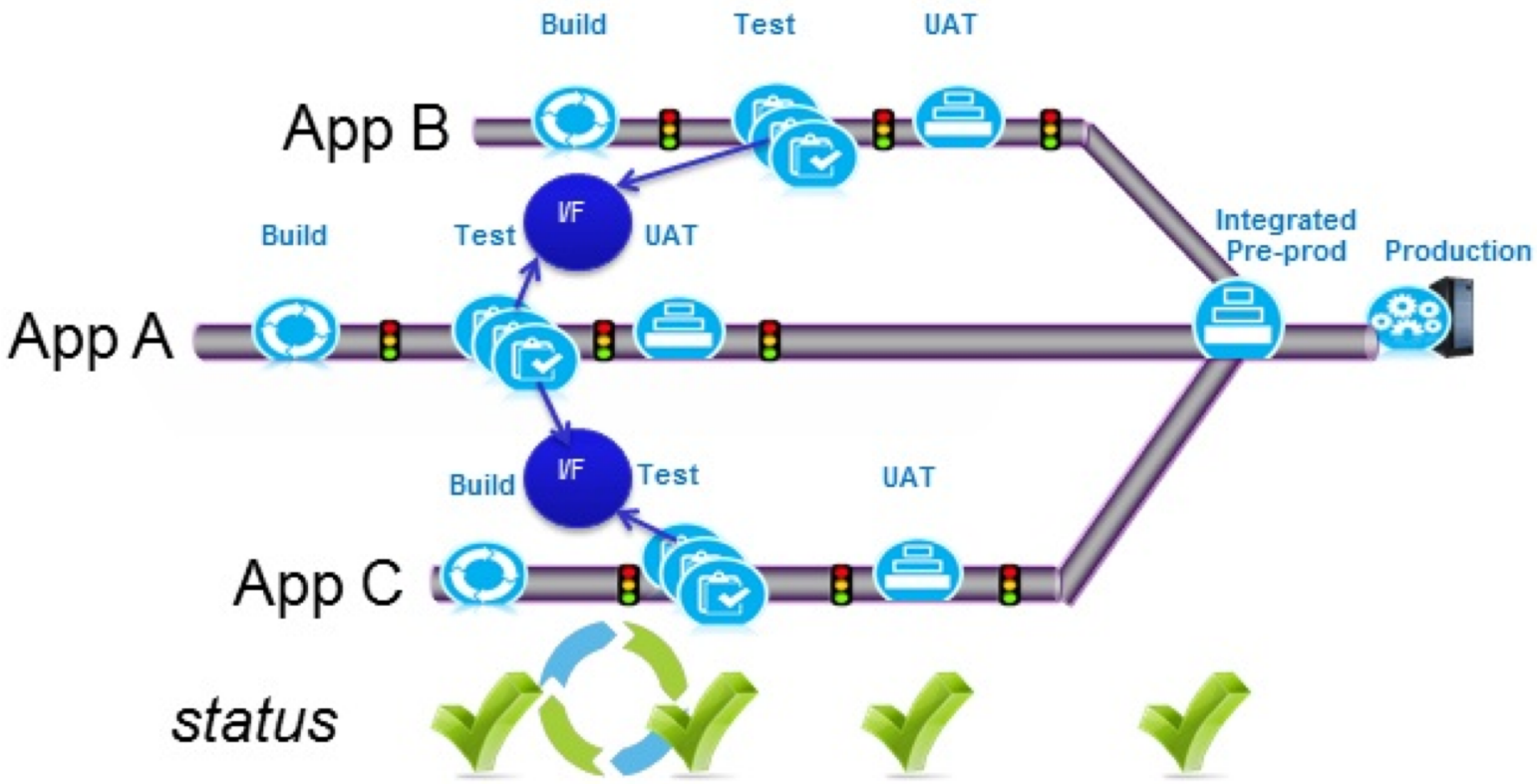
4. Integration Testing

Without Integration Testing:



4. Integration Testing

With Integration Testing:

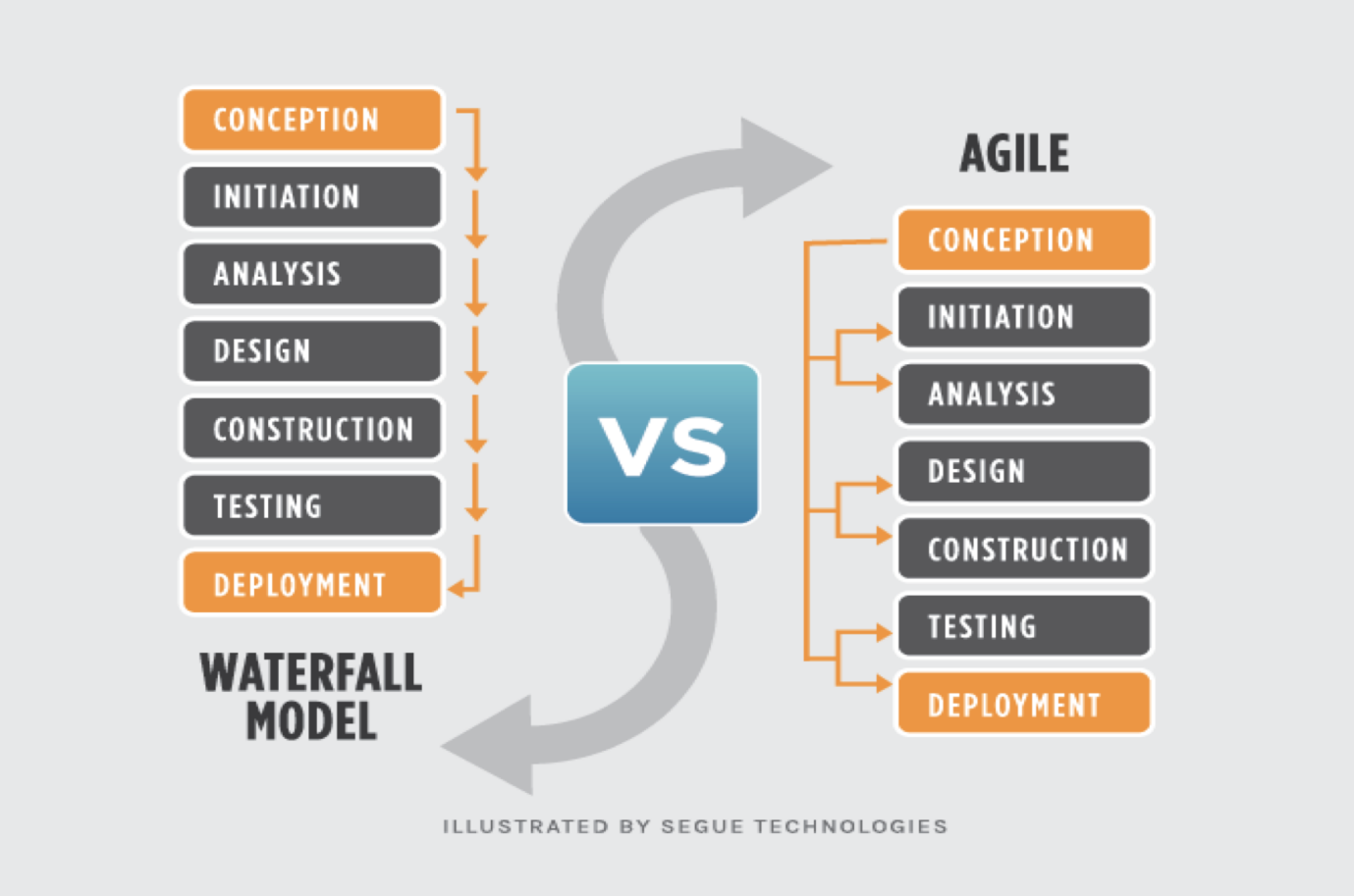


Thinking About the Process

1. Have a Clear Vision

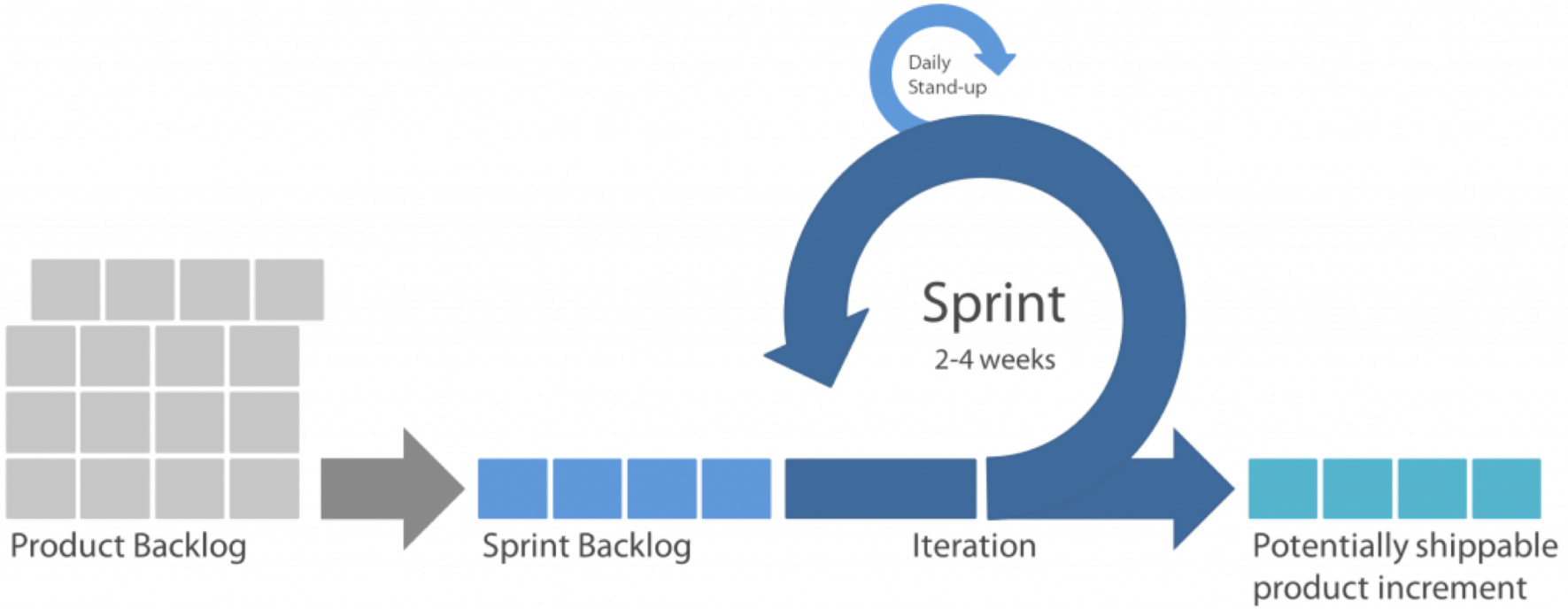
2. Choose the Development Process

Differences between **Waterfall** vs **Agile Development**:

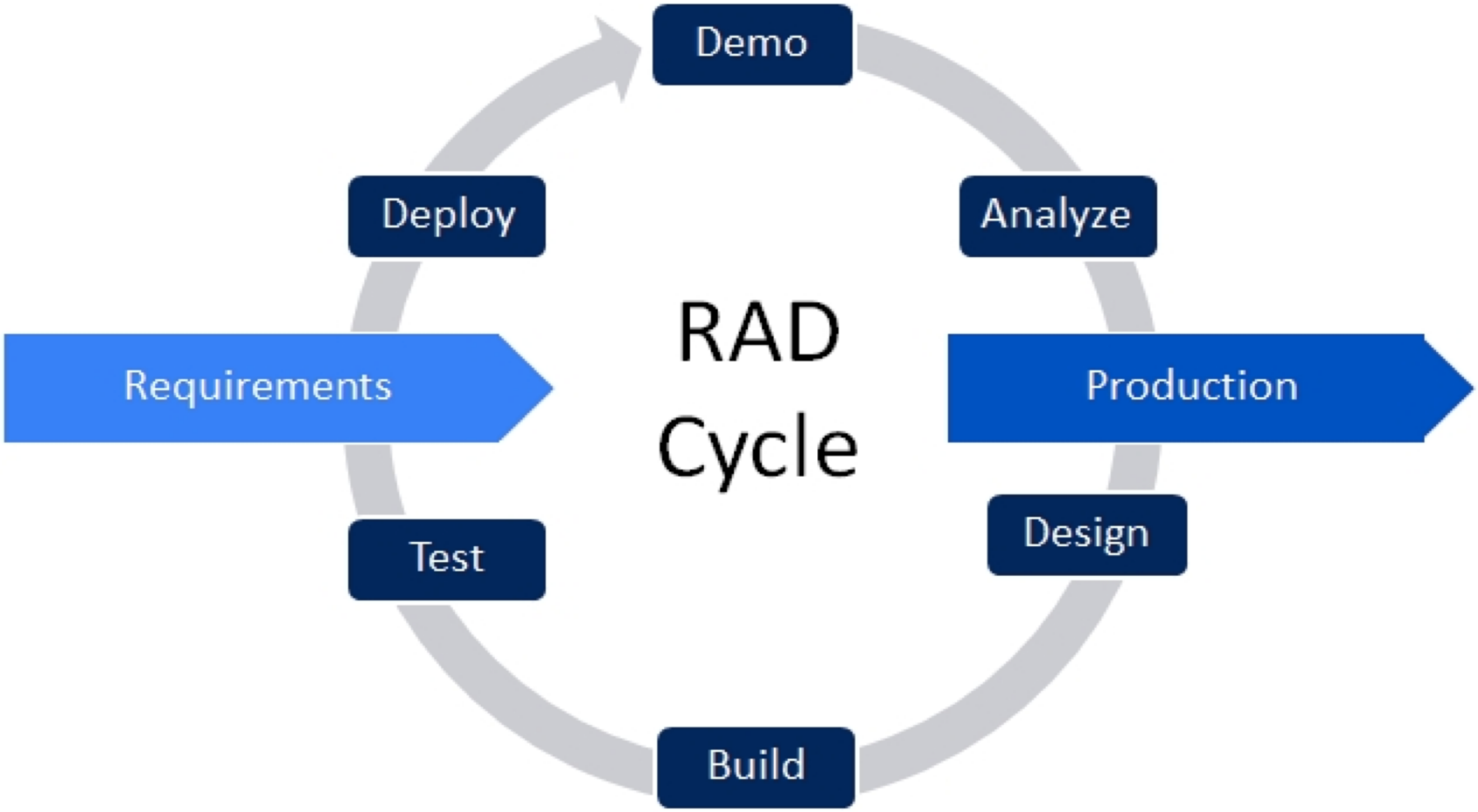


2. Choose the Development Process

A bird's-eye view on **Agile Development**:



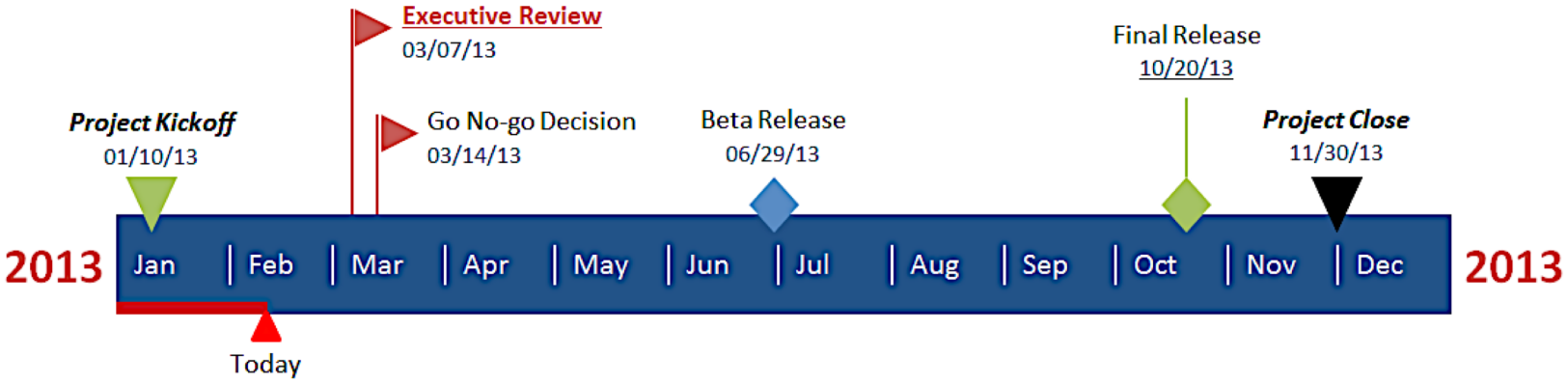
3. Develop Applications Rapidly



3. Develop Applications Rapidly

4. Rigorous Process

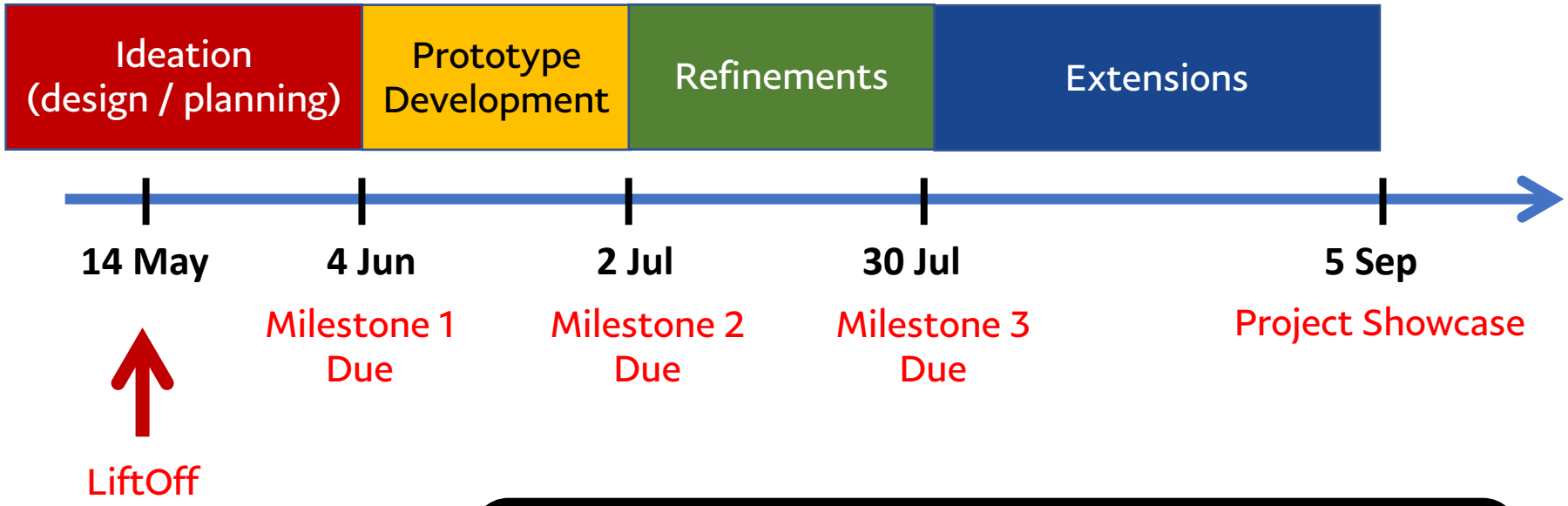
5. Project Plan



Requirements	56 Days	01/05/13 - 03/01/13
Prototype	41 Days	02/02/13 - 03/14/13
Specs	50 Days	03/14/13 - 05/02/13
Development	210 Days	03/25/13 - 10/20/13
Stabilize	26 Days	10/21/13 - 11/15/13
Launch	35 Days	11/16/13 - 12/20/13

5. Project Plan

Orbital Program Plan



What type of model is similar to the Orbital workflow?

Why?

Q & A