Process Models

Week 3 – session 2

Y. Raghu Reddy

Software Engineering Research Center
IIIT Hyderabad, India

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

What Is Agile Software Development?

- In the late 1990's several methodologies began to get increasing public attention. All emphasized:
  - Close collaboration between developers and business experts
  - Face-to-face communication (as more efficient than written documentation)
  - Frequent delivery of new deployable business value
  - Tight, self-organizing teams
  - Ways to craft the code and the team such that the inevitable requirements churn was not a crisis.

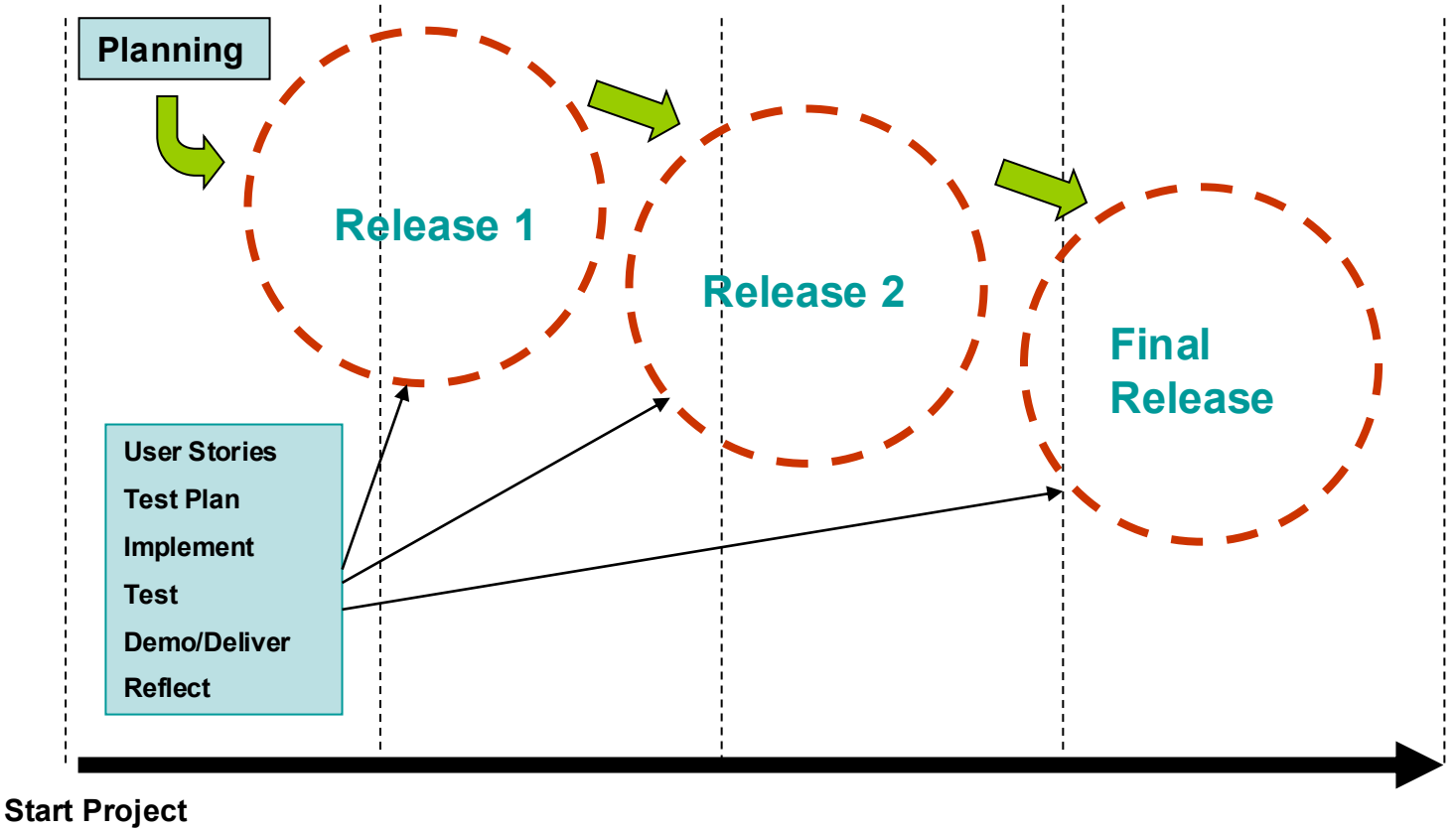
- 2001 : Workshop in Snowbird, Utah, Practitioners of these methodologies met to figure out just what it was they had in common. They picked the word "agile" for an umbrella term and crafted the Manifesto for Agile Software Development,

Applying Agility

Effective (rapid and adaptive) response to change
Effective communication among all stakeholders
Drawing the customer onto the team
Organizing a team so that it is in control of the work performed

Yielding …

Rapid, incremental delivery of software



Agile Characteristics

- Incremental development – several releases
- Planning based on user stories
- Each iteration touches all life-cycle activities
- Testing – unit testing for deliverables; acceptance tests for each release
- Flexible Design – evolution vs. big upfront effort
- Reflection after each release cycle
- Several technical and customer focused presentation opportunities

Key Agile Components

- User Stories
  - Requirements elicitation
  - Planning – scope & composition
- Evolutionary Design
  - Opportunity to make mistakes
- **Refactoring**
  - Small changes to code base to maintain design entropy
- **Test driven development**
  - Dispels notion of testing as an end of cycle activity
- **Continuous Integration**
  - Code (small booms vs big bang)
- Team Skills
  - Collaborative Development (Pair programming)
  - Reflections (process improvement)
- Communication/shared ownership
  - Interacting with customer / team members

Refactoring

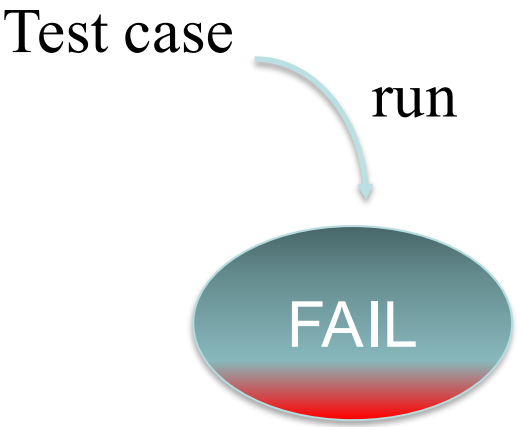
```
public void processOrder(Order order) {
    // 1. Validate Order
    if (order.getItems().isEmpty()) {
        throw new IllegalArgumentException("Order must have items");
    }

    // 2. Calculate Total
    double total = 0;
    for (Item item : order.getItems()) {
        total += item.getPrice() * item.getQuantity();
    }
    if (total > 100) {
        total *= 0.90;
    }
    order.setTotal(total);

    // 3. Save to Database
    database.save(order);

    // 4. Send Confirmation
    emailService.send("Order confirmed!",
order.getEmail());
}
```

Refactoring (Long Method)

```
public void processOrder(Order order) {
    validate(order);
    calculateTotal(order);
    saveOrder(order);
    sendConfirmation(order);
}

private void validate(Order order) {
    if (order.getItems().isEmpty()) {
        throw new IllegalArgumentException("Order must have items");
    }
}

private void calculateTotal(Order order) {
    double total = order.getItems().stream().mapToDouble(i -> i.getPrice() * i.getQuantity()).sum();
    if (total > 100) total *= 0.90;
    order.setTotal(total);
}
// ... other private methods for saveOrder and sendConfirmation
```

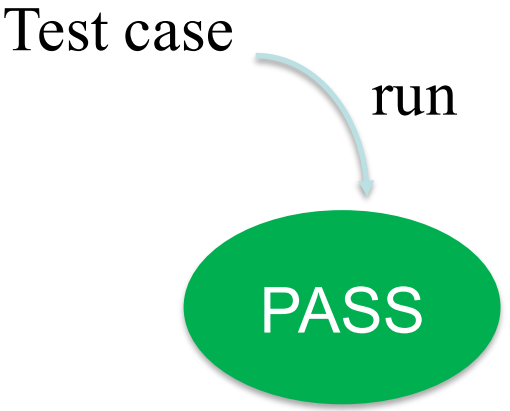
TDD - example

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class CalculatorTest {

    @Test
    public void testAddition() {
        Calculator calculator = new
Calculator();
        int result = calculator.add(3, 4);
        assertEquals(7, result);
    }
}
```




```
public class Calculator {

    public int add(int a, int b) {
        return a + b;
    }

}
```

Write Code;



subtraction… (repeat the process)

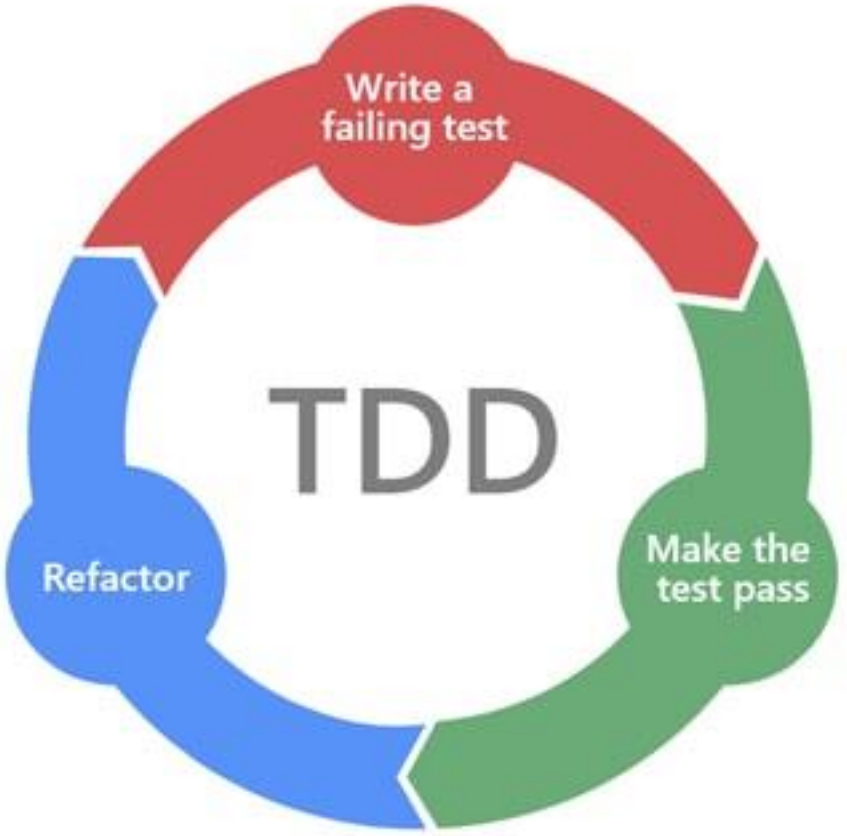
```
@Test
public void testSubtraction() {

    Calculator calculator = new Calculator();
    int result = calculator.subtract (7, 4);
    assertEquals(3, result);
}
```

Implement the functionality…

```
public class Calculator {

    public int add (int a, int b) {
        return a + b;
    }

    public int subtract (int a, int b) {
        return a - b;
    }

}
```

TDD Explained

RED –
GREEN –
REFACTOR
cycle



RED-GREEN-REFACTOR (Shipping Example)

RED: Write a Failing Test

```
// ShippingServiceTest.java
@Test
void shouldCalculateStandardShipping() {
    ShippingService service = new ShippingService();

    // 10km * 10 + 5kg * 5 = 125.0
    double cost = service.calculate ("STANDARD", 10, 5);
    assertEquals (125.0, cost);
}
```

Result: 🔴 **Fail**

RED-GREEN-REFACTOR (Shipping Example)

GREEN: Make it pass – quickly !

```
// ShippingService.java
public class ShippingService {
    public double calculate(String type, double dist, double weight) {
        if (type.equals("STANDARD")) {
            return (dist * 10) + (weight * 5);
        }
        return 0;
    }
}
```

Result:  **Pass**

Additional shipping services…

Imagine you add "Express" shipping, “overnight” shipping, international, etc.

```
if (type.equalsIgnoreCase("STANDARD")) {
        cost = (dist * 1.5) + (weight * 0.5);
    } else if (type.equalsIgnoreCase("EXPRESS")) {
        cost = (dist * 3.0) + (weight * 1.2);
        if (dist < 10) cost += 5.0;                    // Flat fee for short express
    } else if (type.equalsIgnoreCase(”OVERNIGHT")) {
        cost = (dist * 5.0) + (weight * 2.5) * 3;      // multiplication factor of 3
    } else if (type.equalsIgnoreCase(”INTERNATIONAL")) {
        cost = (dist * 5.0) + (weight * 2.5) + 1000;   // Customs fee of Rs. 1000
    } else {
        throw new IllegalArgumentException("Unknown shipping type");
    }
```

Technical Debt !!!

Refactor (clean the design)

Imagine you add "Express" shipping, “overnight” shipping, etc.

```
// 1. Move logic to a dedicated Strategy
interface ShippingRate {
    double compute(double dist, double weight);
}

// 2. Specific implementation – Easily Testable
class StandardRate implements ShippingRate {
    public double compute(double d, double w) { return (d * 10) + (w * 5); }
}

// 3. Clean Service
public class ShippingService {
    public double calculate(ShippingRate rate, double dist, double weight) {
        return rate.compute(dist, weight);
    }
}
```

Result: 🟢 **Still Pass** (The original test is updated to pass the StandardRate object)

Source code integration (Manual)

- **Fetch Source Code:** Source code (download from version control system)
- **Resolve Dependencies:** Ensure all required third-party libraries are present and compatible.
- **Compile:** Run the compiler manually
- **Manual Testing:** Manually run unit tests, check for code style issues (linting), and perform integration tests.
- **Package & Deploy:** Manually package the executable files into an artifact (like a .jar or .exe file) and then manually copy or install it onto a test or production server

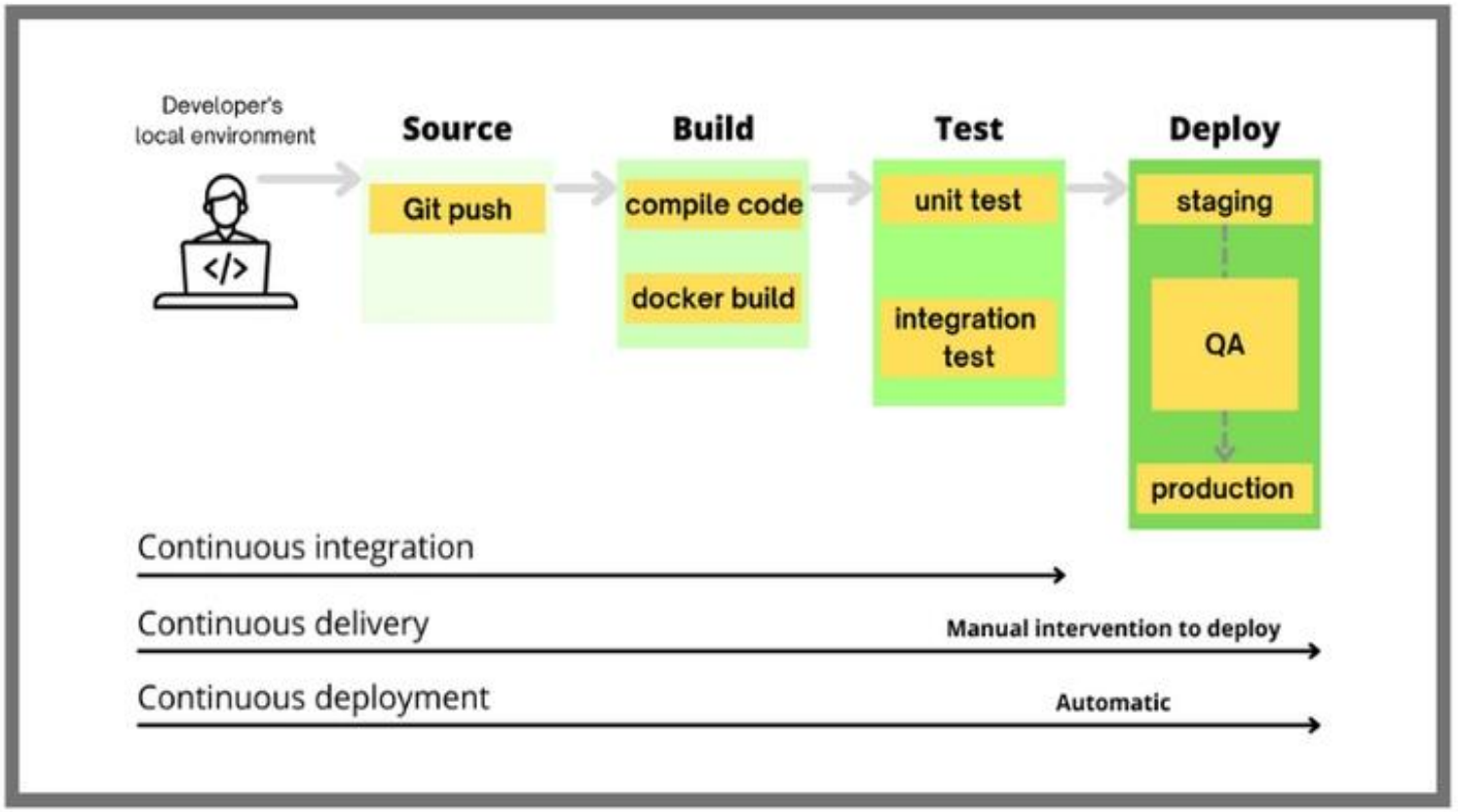
Continuous Integration (automated)

CI is a software development practice where developers merge their code changes into a shared repository frequently.

- **Commit:** A developer pushes code to the repository.
- **Build:** An automated server immediately detects the change and builds the project.
- **Test:** Automated tests (unit, integration) run to ensure the new code hasn't broken anything.
- **Feedback:** The team is instantly notified if the build or tests fail.

CI/CD pipeline

Deployment is Manual (Continuous Delivery) or automated (Continuous Deployment)



Development problems addressed – What about Release problems ?

- Database issues
- OS issues
- Too slow in real settings
- Infrastructure issues
- Source from many repositories
- Different versions (libraries, compilers, local utilities, etc)
- Missing dependencies
- …

Developers & Operations teams need to work together…

Developers

- Designing
- Coding
- Testing, bug tracking, reviews
- Continuous Integration
- …

Operations

Managing/Allocating hardware/OS updates/resources, database

Monitoring load spikes, performance, crashes hardware updates

Backups, Rollback releases

etc.

- ✓ Can there be better coordination between Developers and Operators?
- ✓ Reduce issues while moving changes from development to production
- ✓ Configurations as code
- ✓ Automation (Delivery and Monitoring)

DevOps – Common practices

- Continuous Integration
- Continuous Delivery
- Infrastructure as code, test and deploy in containers
- Monitoring and logging
- Microservice architecture
- Communicate and Collaborate

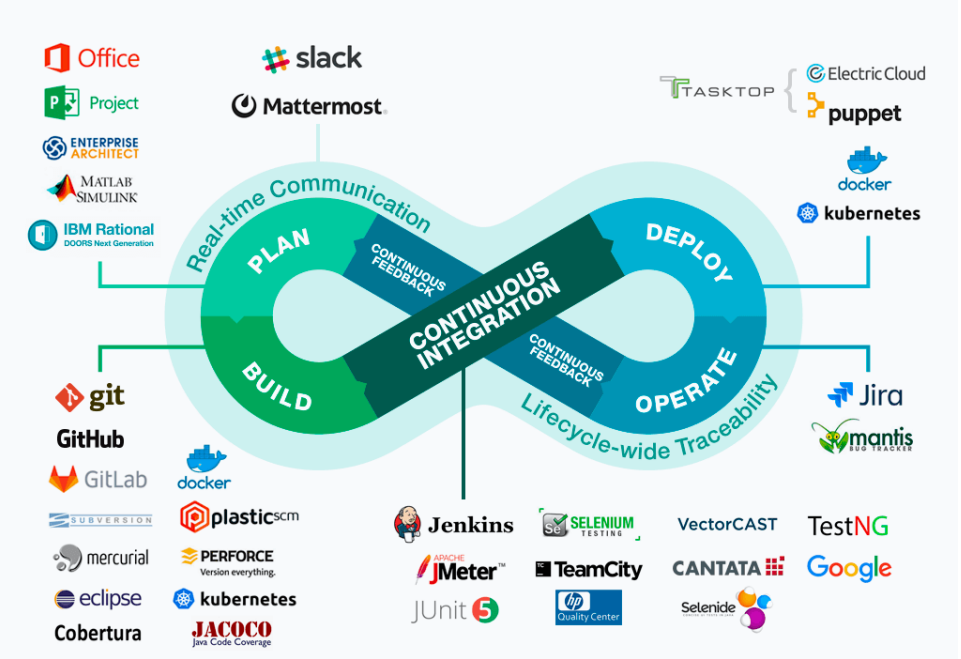
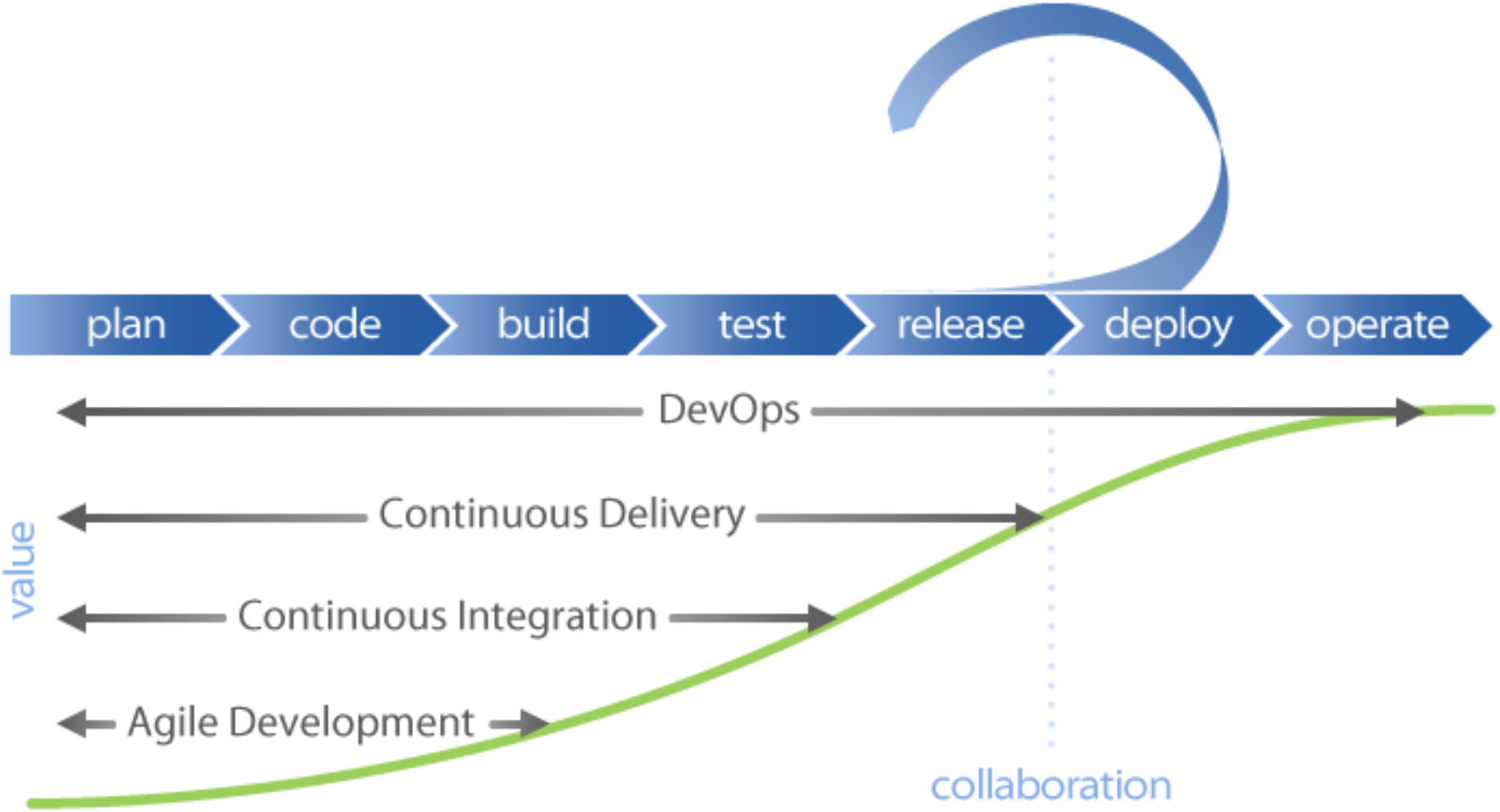


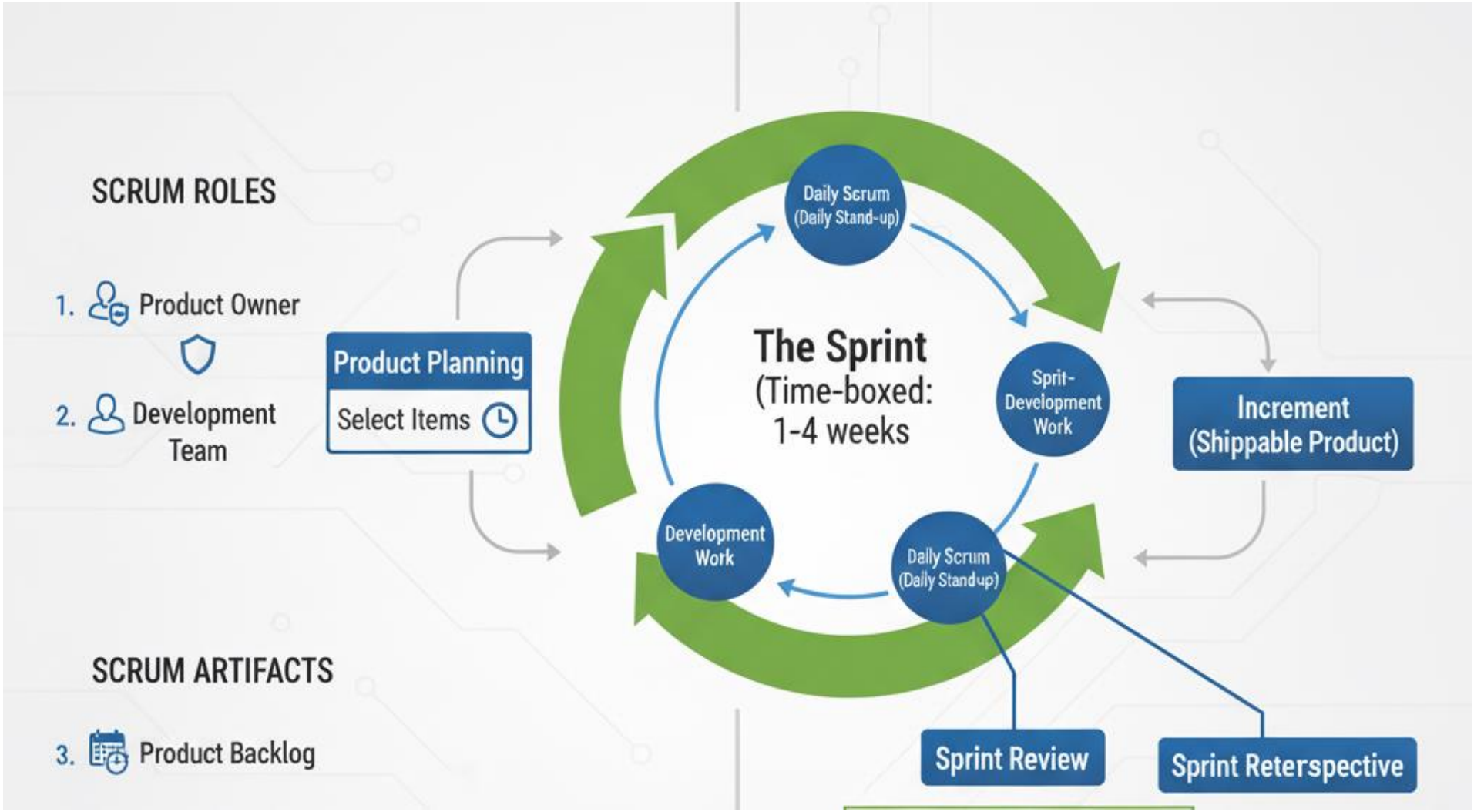
Image Source: https://intland.com/wp-content/uploads/2019/07/devops-infinity-1-1.png

Agile, CI, CD, DevOps…

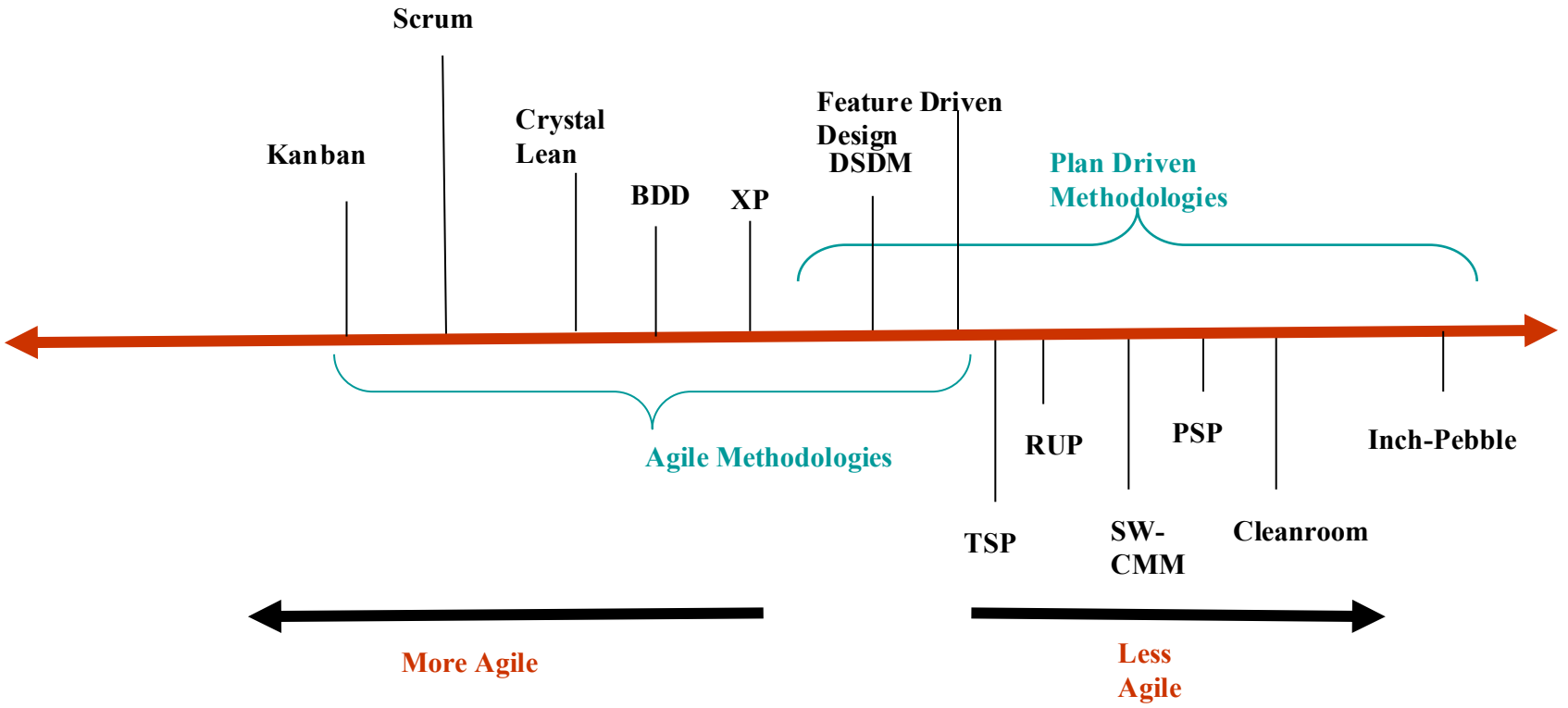


image@CollabNet

Scrum Process



The Process Methodology Spectrum



It's not that black and white. The process spectrum spans a range of grey !

Adapted from (Barry Boehm & Turner)

Questions ?