Double Loop

TDD & BDD Done Right!

@jessicamauerhan | #ATO2016

@jessicamauerhan

Senior Software Engineer

TDD/BDD Expert

jessicamauerhan@gmail.com

jmauerhan.wordpress.com

What is Test Driven Development?

Test Driven Development is a process in which you first write a failing unit test, execute that test, write just enough code to make the test pass, then refactor that code for improvements.

Benefits of Test Driven Development

Primary

- Guides you towards creating a well-abstracted and highly modular system.
- Makes the software easier to change, maintain and understand.
- Promotes using design patterns, refactoring, and helps prevent scope creep.

Secondary

- Complete test coverage
- Preventing and identifying regressions and mistakes with automated tests
- Executable documentation.

What is Behavior Driven Development?

Behaviour-driven development is an “outside-in” methodology. It starts at the outside by **identifying business outcomes**, and then drills down into the feature set that will achieve those outcomes. Each feature is captured as a “**story**”, which defines the scope of the feature along with its **acceptance criteria**.

Dan North, “What’s in a Story”
http://dannorth.net/whats-in-a-story

It’s the idea that you start by writing **human-readable sentences** that **describe a feature** of your application and how it should work, and only then implement this behavior in software.

Behat Documentation
http://docs.behat.org/en/v2.5

Why Practice BDD?

BDD Is Not About...

- Well Designed Code
- Automated Testing
- Implementation
- UI Testing

BDD Is About...

- Communication
- Collaboration
- Documentation
- Preventing Regressions

Why Practice Behavior Driven Development?

"You can turn an **idea** for a **requirement** into **implemented, tested, production-ready code** simply and effectively, as long as the requirement is specific enough that **everyone knows what's going on**."

Dan North, "What's in a Story"
http://dannorth.net/whats-in-a-story

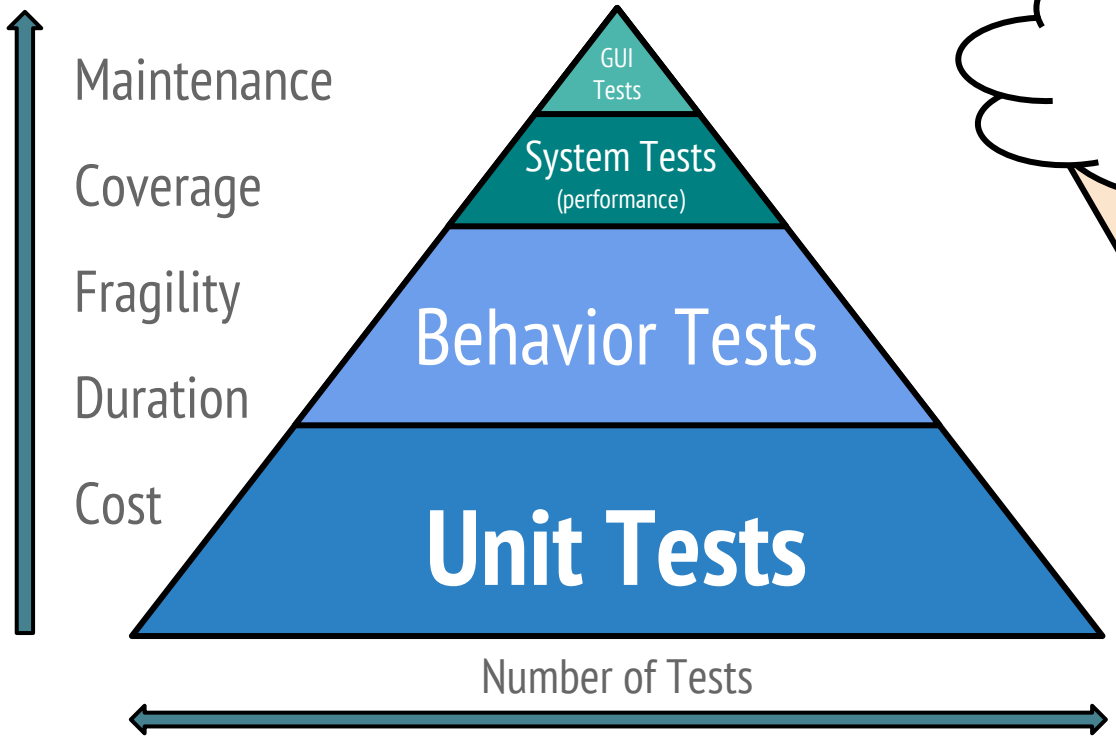
Accuracy

Unit Tests describe accuracy of code and prescribe maintainable code

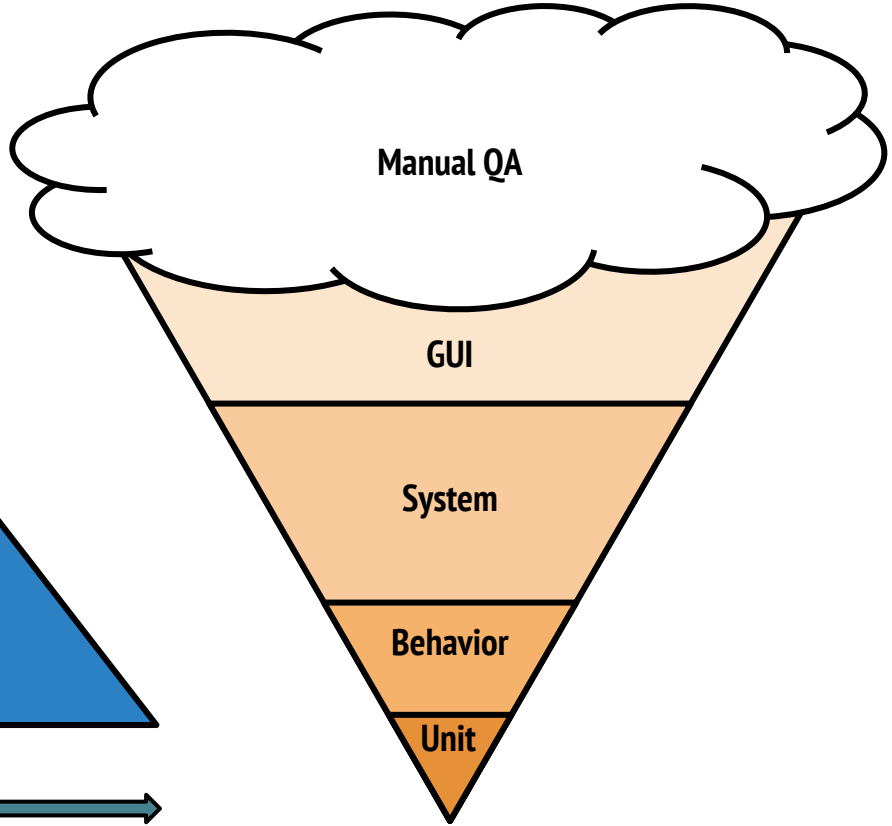
Suitability

Behavior Tests describe suitability of software's features for the end user

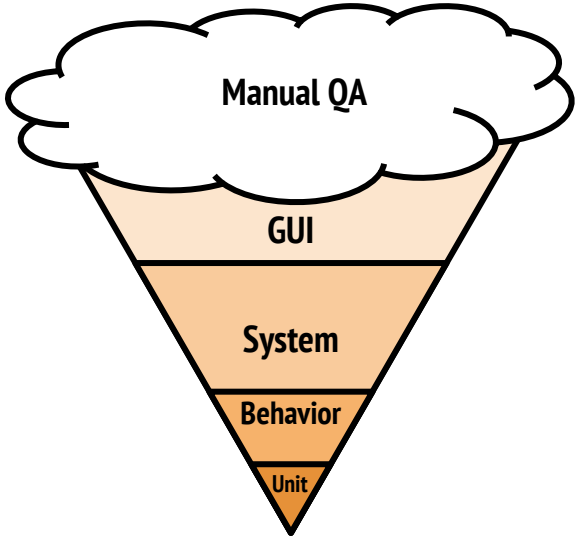
Testing Pyramid

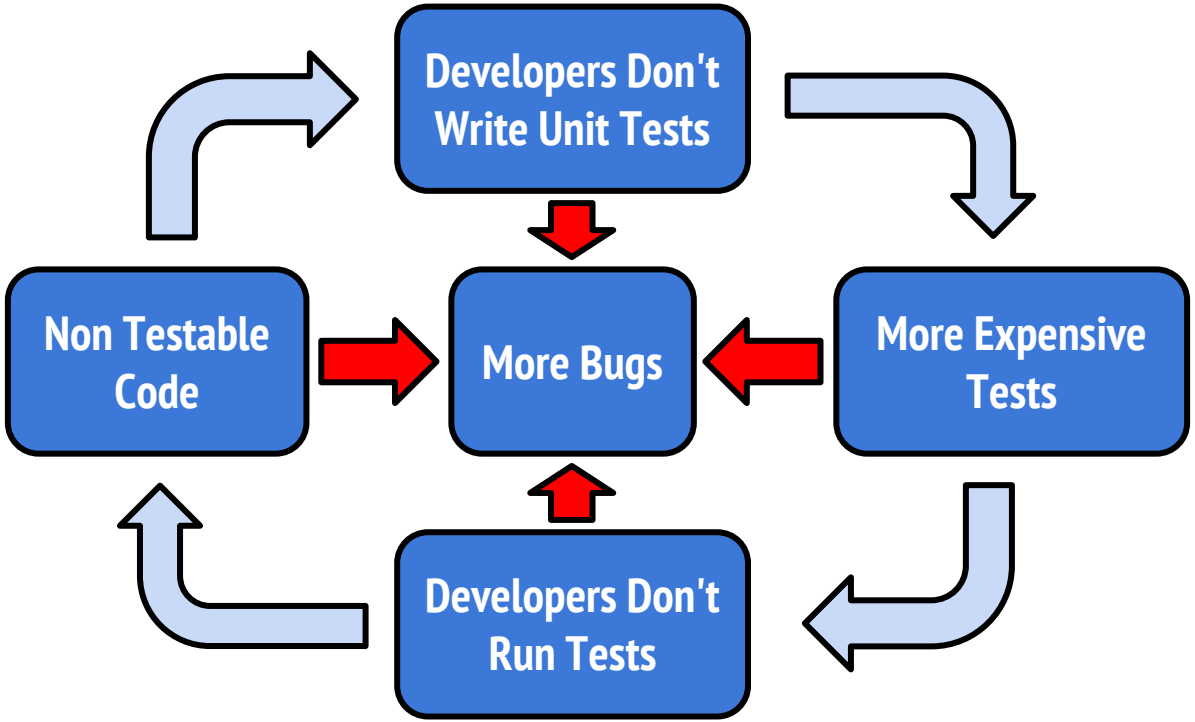


Testing Ice Cream Cone

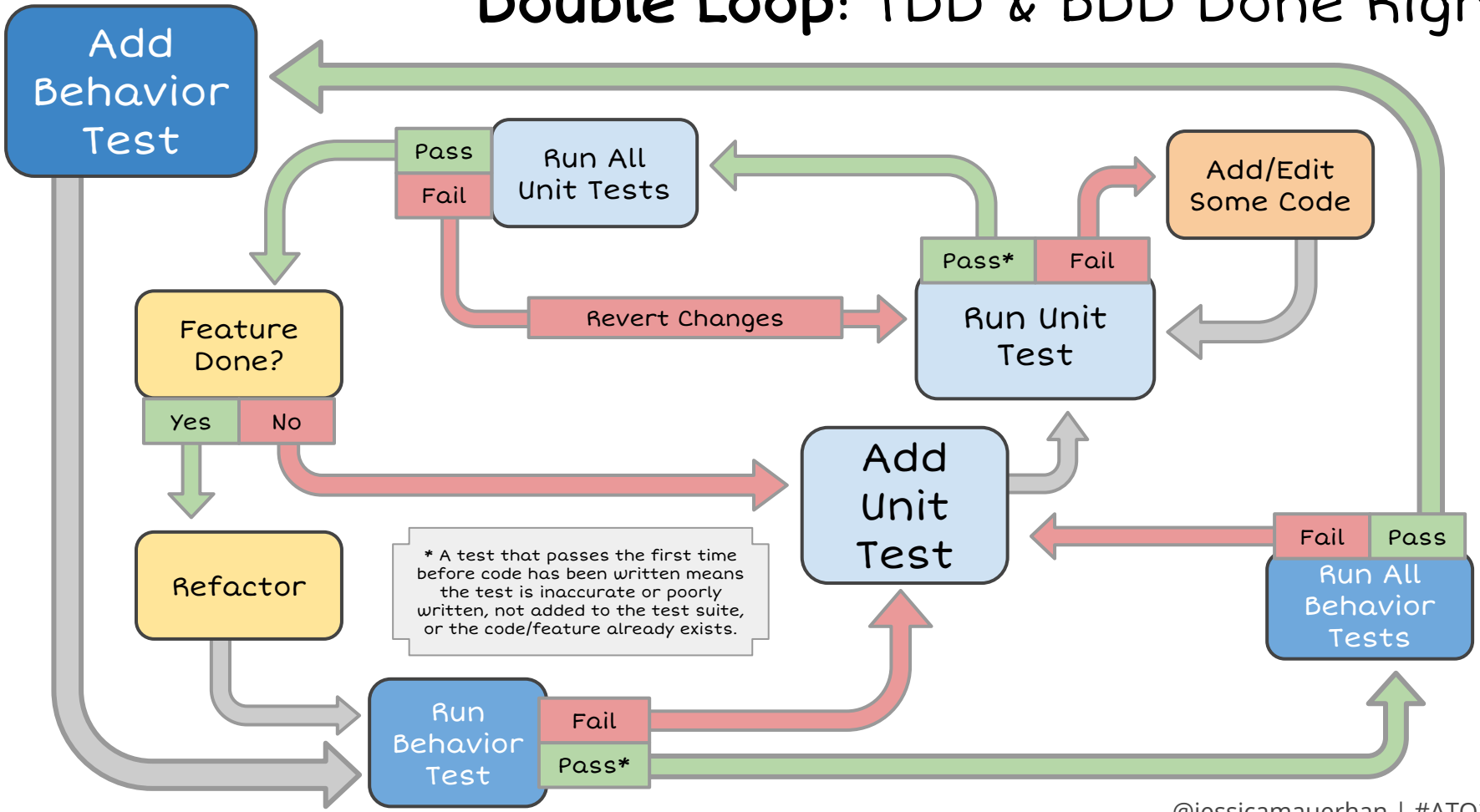


The Vicious Cycle of the Testing Ice Cream Cone





Double Loop: TDD & BDD Done Right!



Add Behavior Test

Parts of a Behavior Test

- **Feature: Gherkin**
- Test Code
- BDD Framework / Test Runner

Gherkin

- Business Readable, Domain Specific Language
- Living Documentation / User Manual
- User Story with Narrative
- Contains at least one Scenario: Acceptance Criteria

Feature: Account Holder Withdraws Cash

As an Account Holder
I want to withdraw cash from an ATM
So that I can get money when the bank is closed

Scenario: Account has sufficient funds
Given the account has a balance
And the card is valid
And the machine contains at least the amount of my balance
When I request an amount less than or equal to my balance
Then the ATM should dispense that amount
And the amount should be subtracted from my account balance
And the card should be returned

Scenario: Account has insufficient funds
Given the account has a balance
And the card is valid
And the machine contains at least the amount of my balance
When I request an amount greater than my balance
Then the ATM should not dispense any money
And the ATM should say there are insufficient funds
And my account balance should not change
And the card should be returned

- Feature: Short, Descriptive, Action
- Narrative
  - As a [role]
    I want [feature]
    So that [benefit / business reason]
  - Use “5 Whys” to determine narrative
- Acceptance Criteria: Scenarios
  - Given: **Exact** Context
  - When: Action/Event
  - Then: Outcomes
  - And/But: More of the same...
-

Feature: View Countdown before Broadcast

As a user viewing a broadcast page before the broadcast starts
I want to see a countdown timer
So that I can know how long until the broadcast actually starts

Scenario: View Countdown before Broadcast

Given I view the “Future Broadcast” broadcast
Then I should see "Future Broadcast" on the page
And I should see "Future Broadcast Author" on the page
And I should see "This broadcast begins at 6:00 pm EST" on the page
And I should see a countdown timer

Process

- Author describes Feature with implementation specific example
- Developer adds fixture data

Issues

- Test only works before 6pm
- What is the intent?
- Confusion for business users

Feature: View Countdown before Broadcast

As a user viewing a broadcast page before the broadcast starts
I want to see a countdown timer
So that I can know how long until the broadcast actually starts

Scenario: View Countdown before Broadcast

Given there is a broadcast scheduled for the future
When I view that broadcast's page
Then I should see the broadcast title
And I should see the broadcast author's name
And I should see "This broadcast begins at" followed by the start time in EST
And I should see a countdown timer

Changes

- Given now explains exact context
- Test is no longer time-dependent
- Intent - not implementation

Why?

- Overall understanding
- Communication value
- Help identify poorly written code

Feature: Customer Views Product and Services Catalog

As a customer
I want to view the product catalog
So that I can browse products and services

Scenario: Customer views Product Catalog

Given I view the catalog
When I select a state from the list of states
Then I should see the list of products for sale

Scenario: Display Local Services in Product Catalog

Given the company has a regional office in a state
When I view the catalog
And I select that state from the list of states
Then I should see the list of services offered

Scenario: Don't Display Local Services in Product Catalog For States With No Regional Office

Given the company does not have a regional office in a state
When I view the catalog
And I select that state from the list of states
Then I should not see a list of services offered

```
echo '<h1>Products</h1>';
foreach ($products AS $product) {
   echo '<p>' . $product->getName() . '</p>';
}
```

```
echo '<h1>Services</h1>';
foreach ($services AS $service) {
   echo '<p>' . $service->getName() . '</p>';
}
```

Feature: Customer Views Product and Services Catalog

As a customer
I want to view the product catalog
So that I can browse products and services

Scenario: Customer views Product Catalog

Given I view the catalog
When I select a state from the list of states
Then I should see the list of products for sale

Scenario: Display Local Services in Product Catalog

Given the company has a regional office in a state
When I view the catalog
And I select that state from the list of states
Then I should see the list of services offered

Scenario: Don't Display Local Services in Product Catalog For States With No Regional Office

Given the company does not have a regional office in a state
When I view the catalog
And I select that state from the list of states
Then I should not see a list of services offered

```
echo '<h1>Products</h1>';
foreach ($products AS $product) {
   echo '<p>' . $product->getName() . '</p>';
}


if(count($services) > 0) {
   echo '<h1>Services</h1>';
   foreach ($services AS $service) {
       echo '<p>' . $service->getName() . '</p>';
   }
}
```

Writing Great Features

- Exact Context
- Independent Scenarios & Features
- Intention, not Implementation
- Defined Narrative
- All Paths Explored

Feature "Smells"

- Time Dependency
- Interdependency
- Multi-Scenario Scenarios
- Missing Scenarios
- Overuse of Variables
- Examples of Behavior (Implementation, not Intention)

Parts of a Behavior Test

- Feature: Gherkin
- **Test Code**
- **BDD Framework** / Test Runner

BDD Frameworks / Test Runners

- Java - JBehave, Cucumber and more
- Ruby - Cucumber
- PHP - Behat
- JavaScript - cucumber.js, jasmine
- ...

Scenario: Customer views Product Catalog

Given I view the catalog

When I select a state from the list of states

Then I should see the list of products for sale

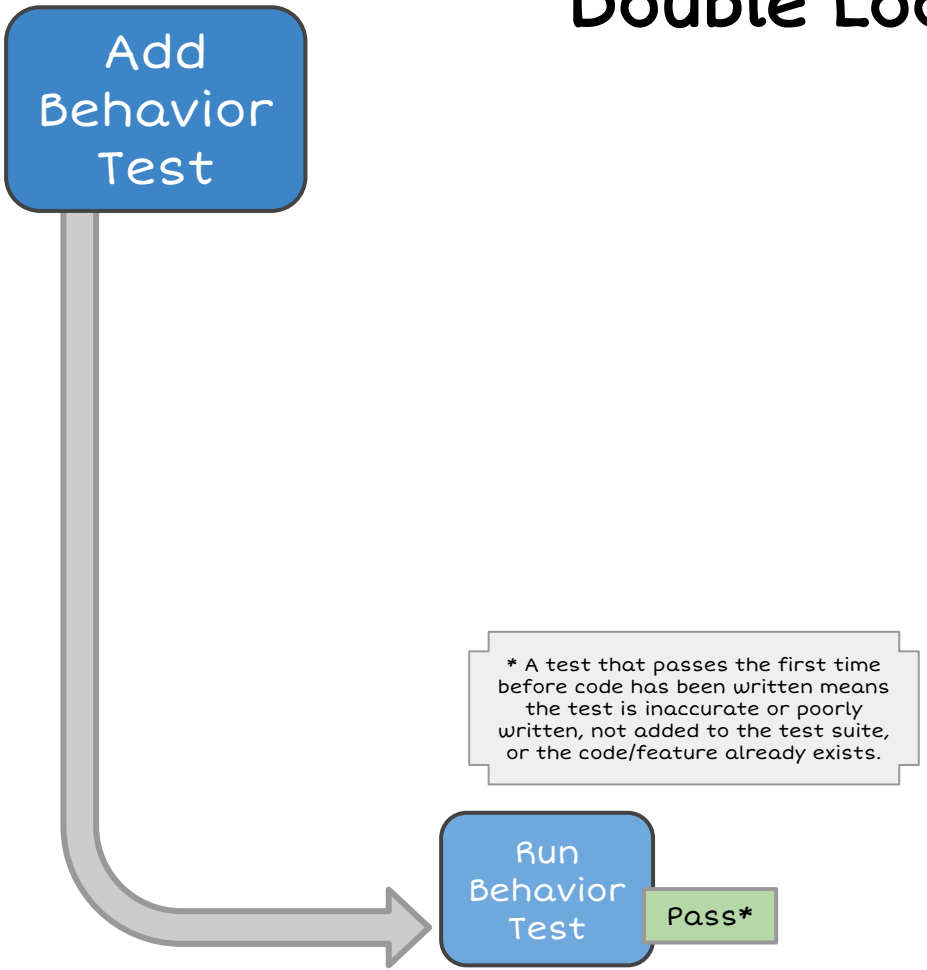
Behat Code

```
/**
* @Given I view the catalog
*/
public function iViewTheCatalog()
{
   $this->visit('catalog');
}

/**
* @When I select a state from the list of states
*/
public function iSelectAStateFromTheListOfStates()
{
   $this->selectOption('states', rand(1, 52));
}

/**
* @Then I should see the list of products for sale
*/
public function iShouldSeeTheListOfProductsForSale()
{
   $productsDiv = $this->getSession()
                       ->getPage()
                       ->find('css', '#products');
   Assertion::notNull($productsDiv);
}
```

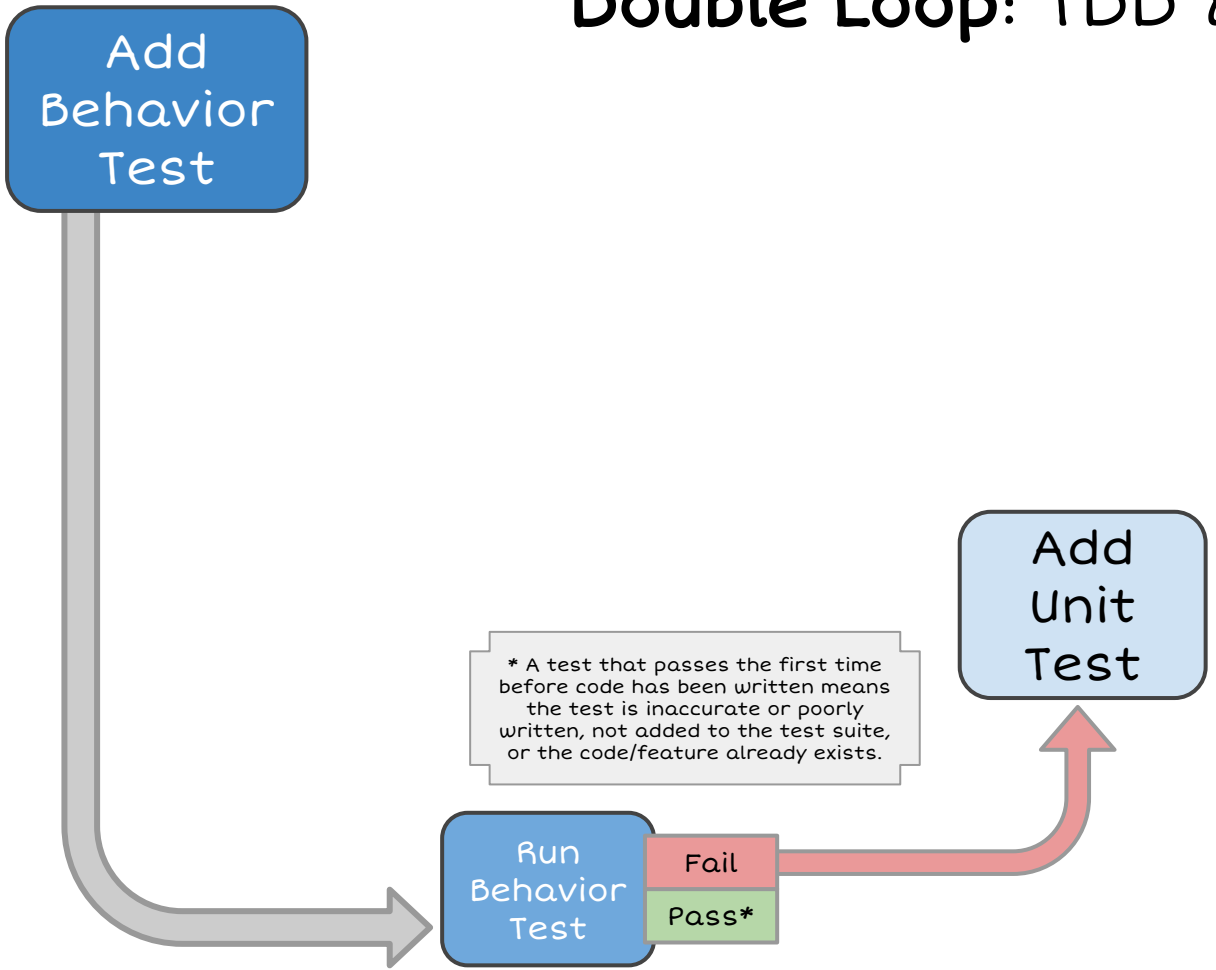
Double Loop: TDD & BDD Done Right!



Parts of a Behavior Test

- Feature: Gerkhin
- Test Code
- BDD Framework / **Test Runner**

Double Loop: TDD & BDD Done Right!

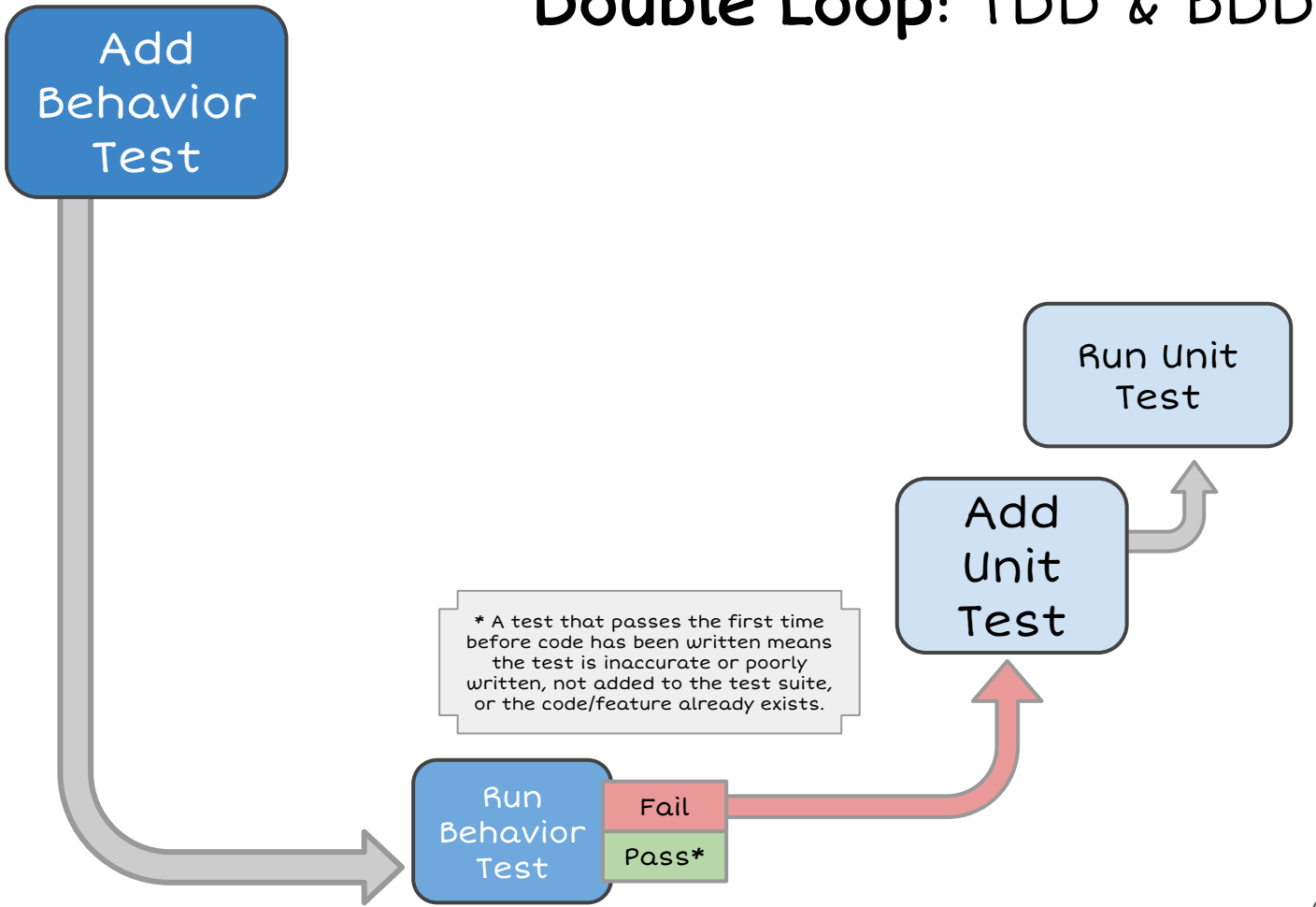


Writing Unit Tests

What is a unit?

- Unit: As small as possible
- Design and Describe behavior
  - Setup scenario for behavior
  - Cause behavior to occur
  - Assert result is as expected
- Verbose and Descriptive Names

Double Loop: TDD & BDD Done Right!

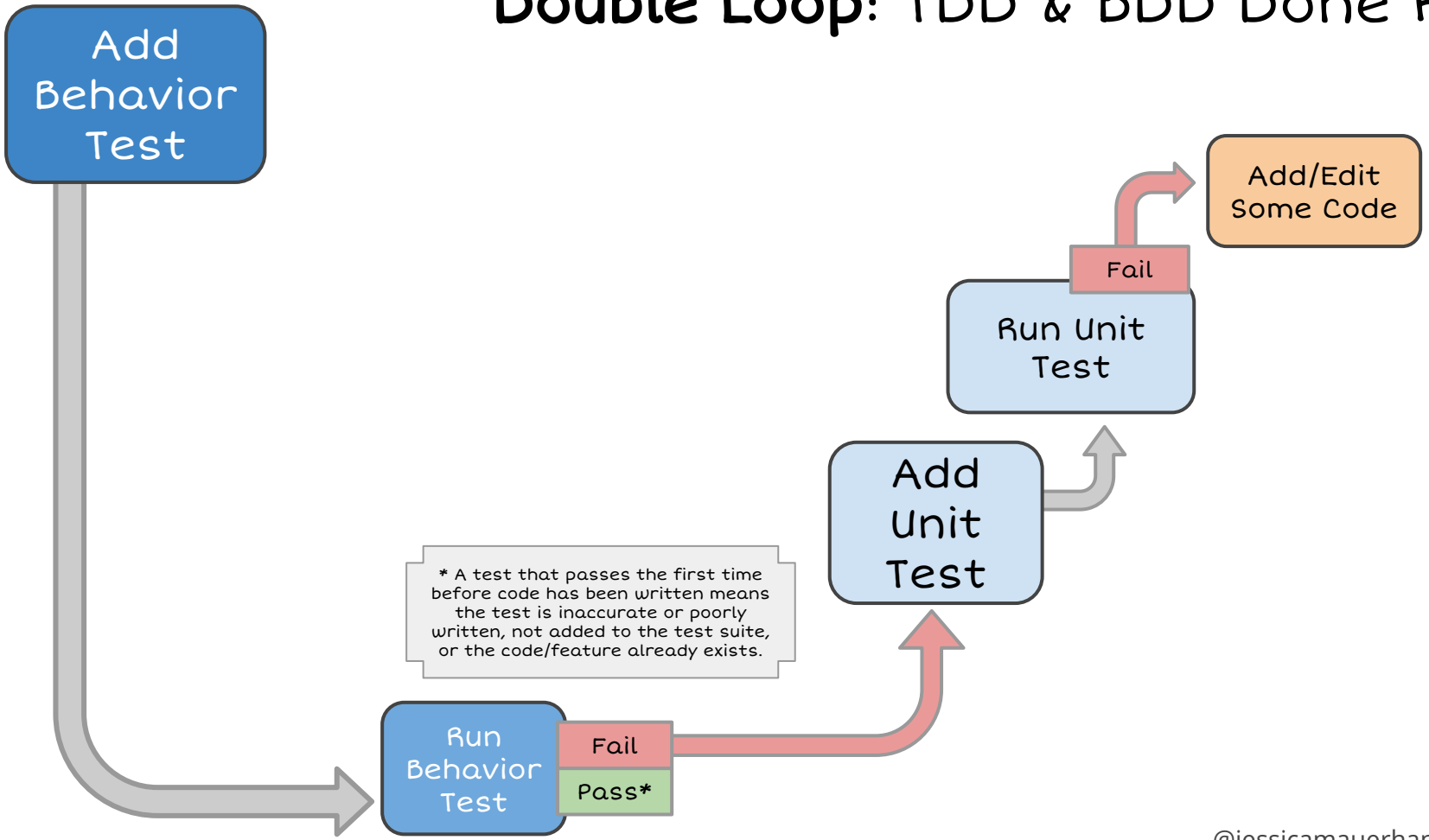


Executing Unit Tests

Always execute before writing production code

- Ensures test fails when code is broken
- Ensures test is added to suite
- Tells you exactly what to do next

Double Loop: TDD & BDD Done Right!



Writing Code

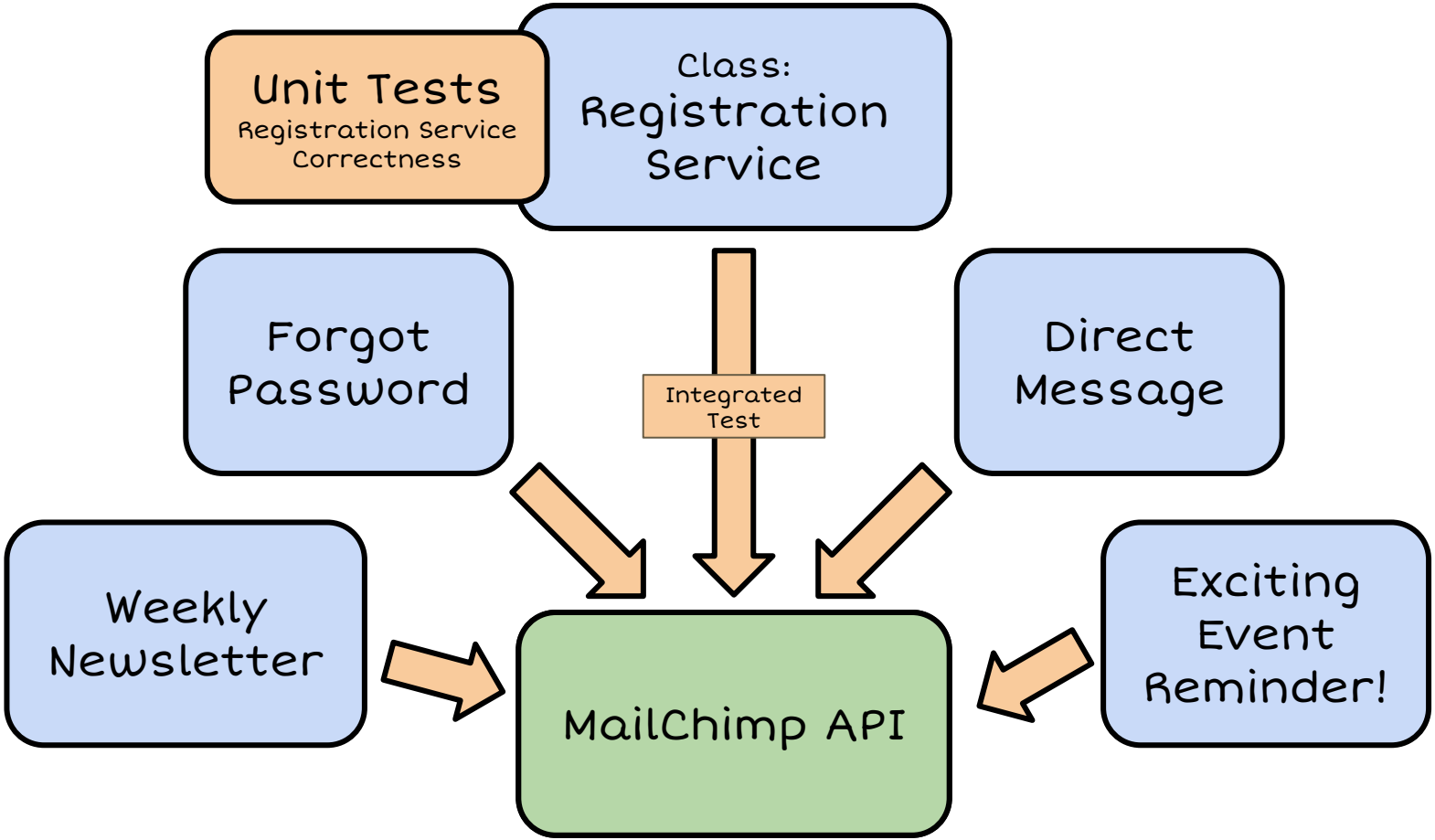
Just Enough

- Add just enough code to pass the test
- Avoid over abstracting or adding untested logic

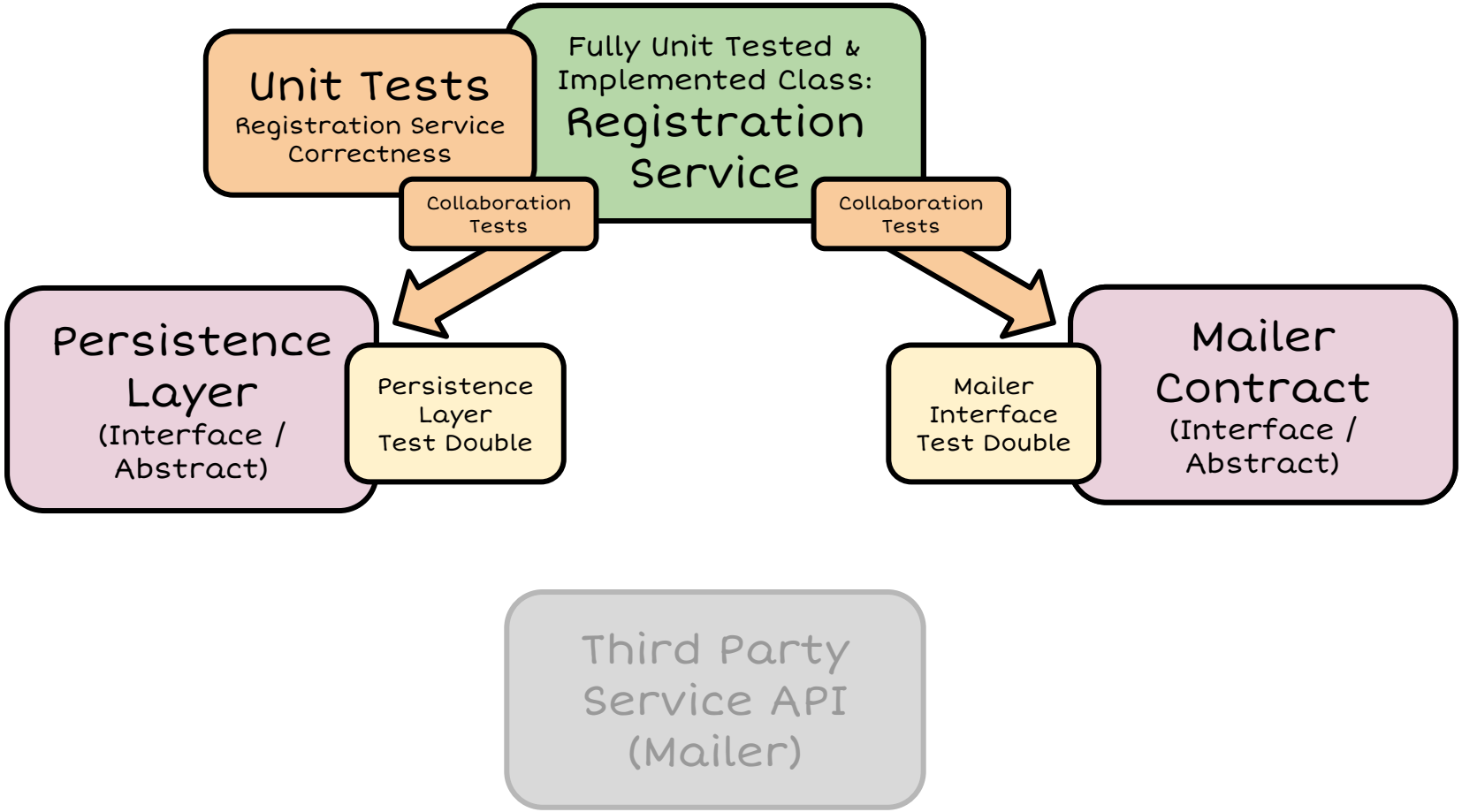
Three Types of Unit Tests

- Correctness
- Contract
- Collaboration

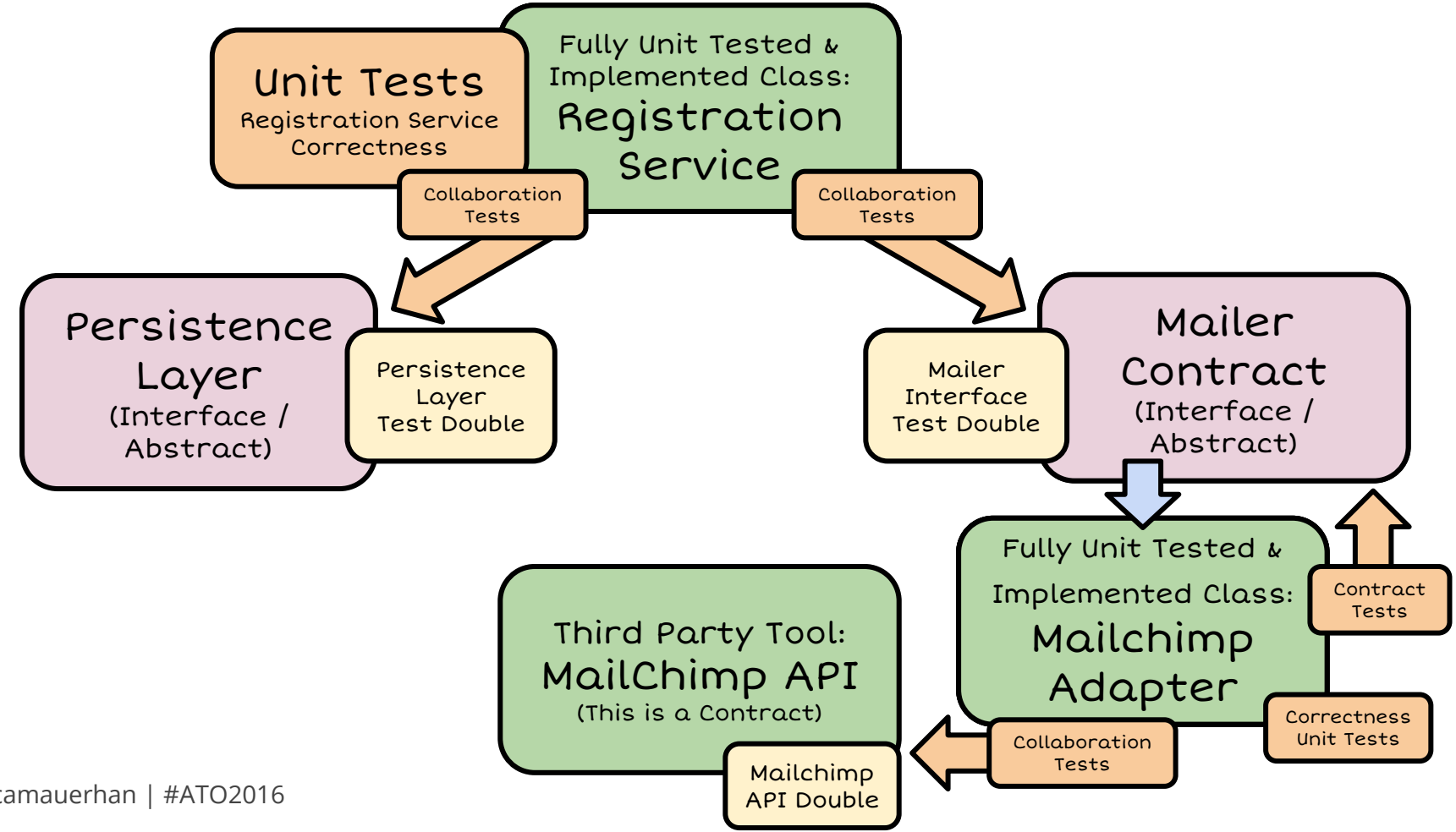
TDD: Integration Testing with Contract & Collaboration Tests



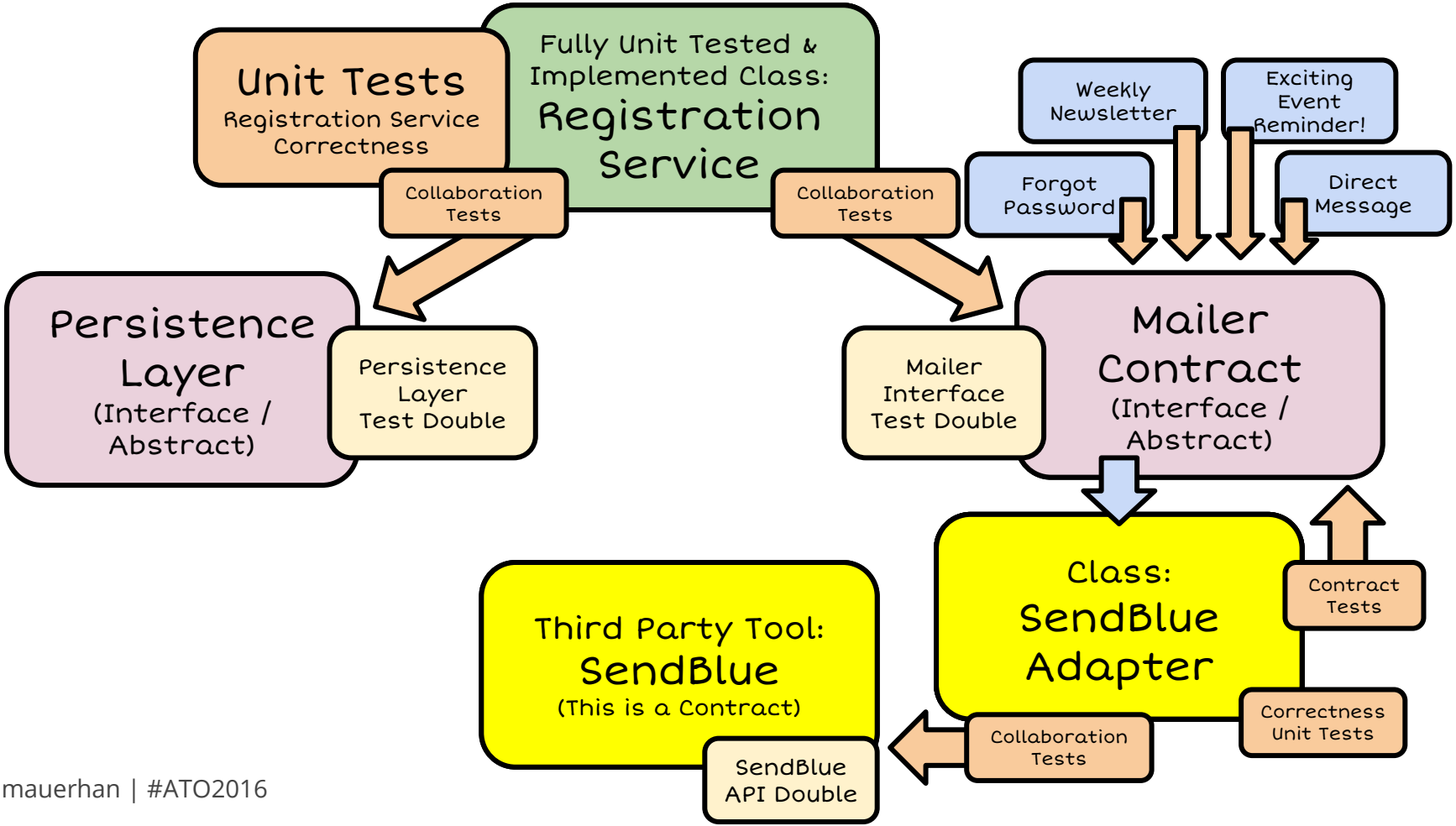
TDD: Integration Testing with Contract & Collaboration Tests

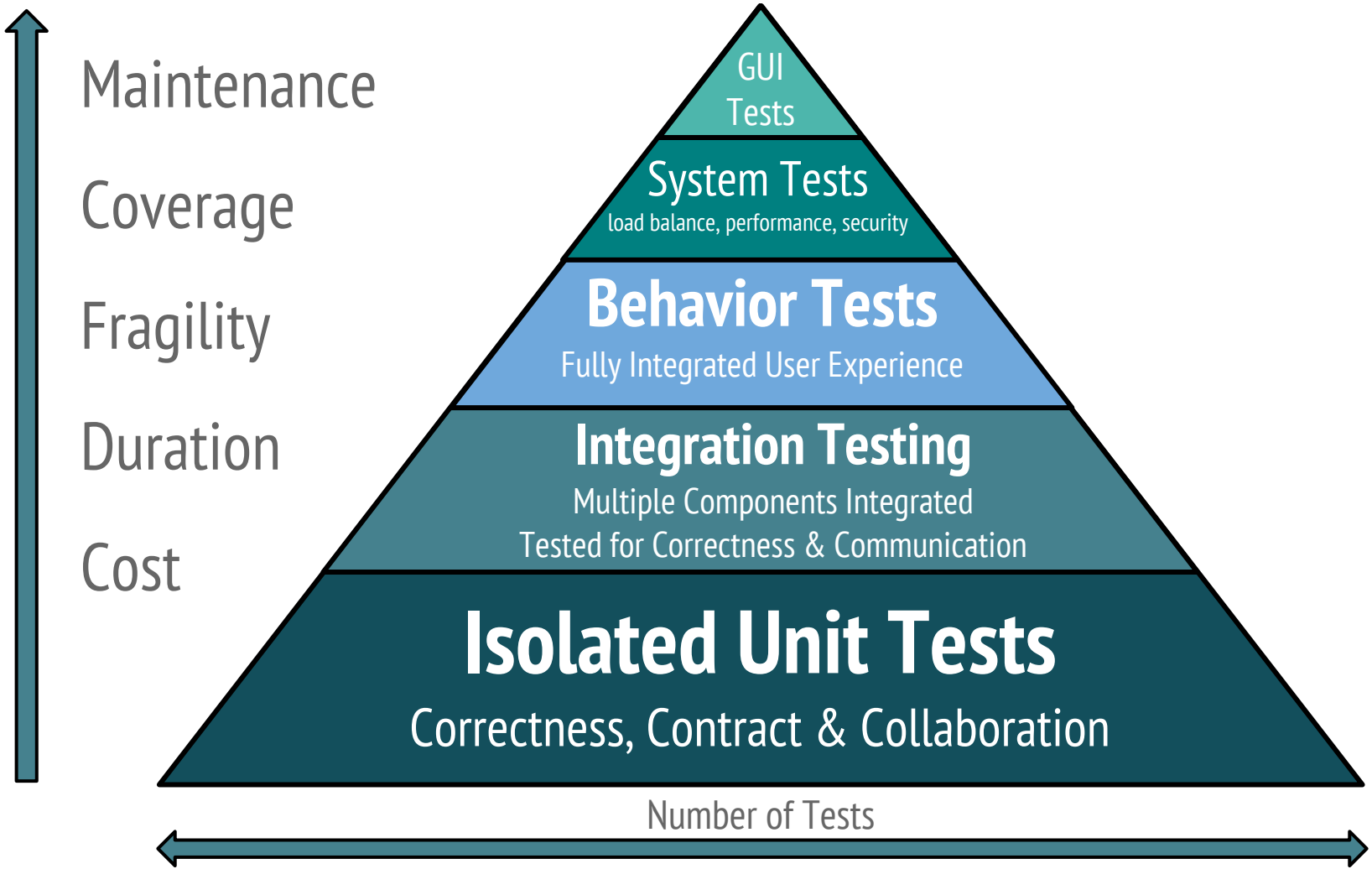


TDD: Integration Testing with Contract & Collaboration Tests

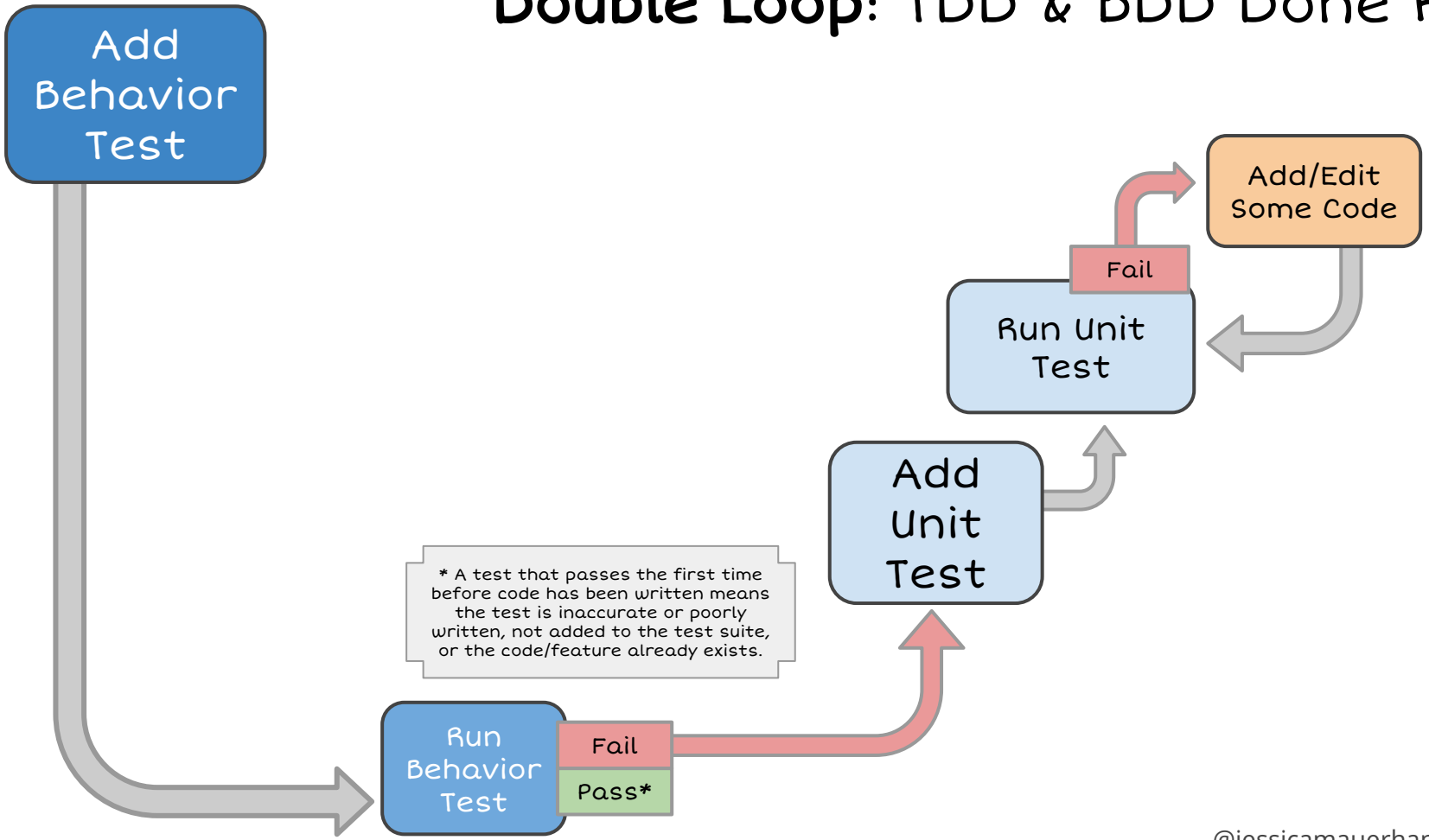


TDD: Integration Testing with Contract & Collaboration Tests




Maintenance
Coverage
Fragility
Duration
Cost
GUI Tests
System Tests
load balance, performance, security
Behavior Tests
Fully Integrated User Experience
Integration Testing
Multiple Components Integrated
Tested for Correctness & Communication
Isolated Unit Tests
Correctness, Contract & Collaboration
Number of Tests

Double Loop: TDD & BDD Done Right!

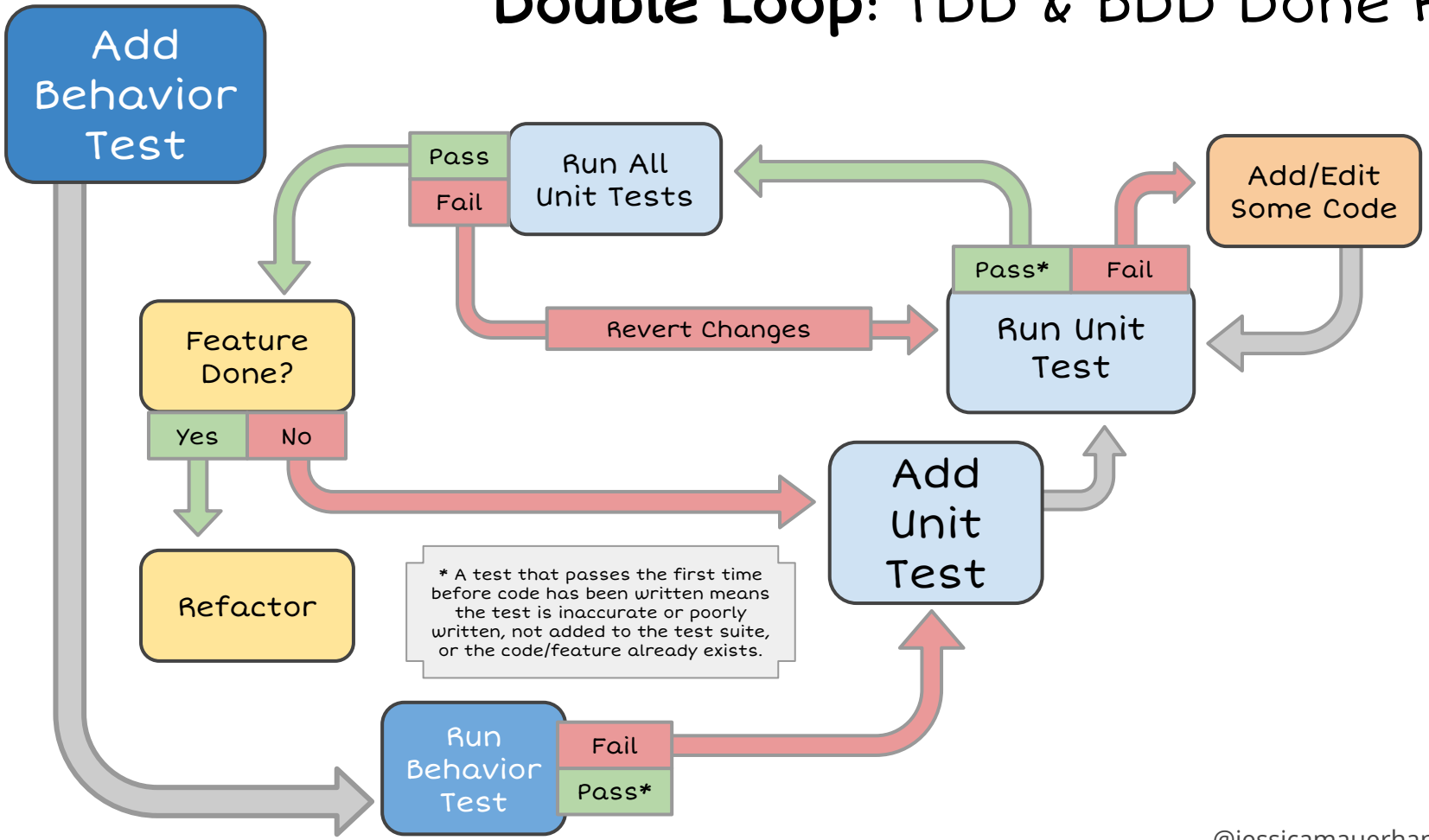


Failing Test or Test Suite?

Revert!

- If the test fails, revert your work
- Commit small changes, often!

Double Loop: TDD & BDD Done Right!



Refactoring

The process of restructuring code without changing its behavior

Refactoring

- Improving non-functional aspects
- Removing duplication
- Renaming
- Simplifying

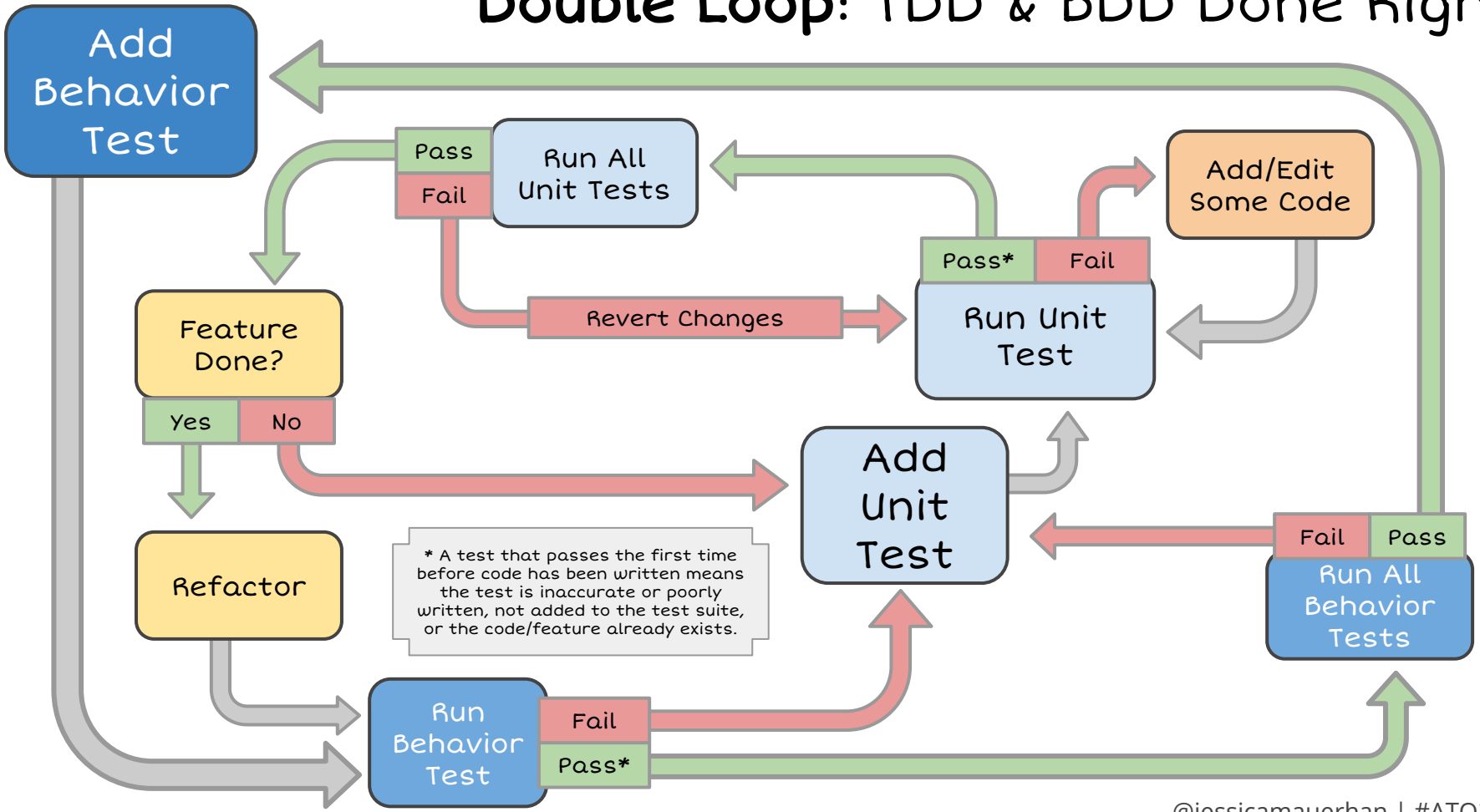
Unit Tests Always Passing!

Adding Functionality

- Changes that would break a unit test
- New functionality that is not tested

New Unit Tests Fail, Then Pass

Double Loop: TDD & BDD Done Right!



Thank You!

Double Loop:
TDD & BDD Done Right!

Feedback & Questions?
Welcome & Encouraged!

@jessicamauerhan

jessicamauerhan@gmail.com

jmauerhan.wordpress.com