TESTINGMIND

COMBINING DDD AND BDD TO BOOST ACCEPTANCE TESTING AND TESTING AUTOMATION

LEONARDO SOUZA MATTOS
#TAS19

Philadelphia
April 5th, 2019

About me

Technical Coach

IT Services
Principal Consultant

Developer

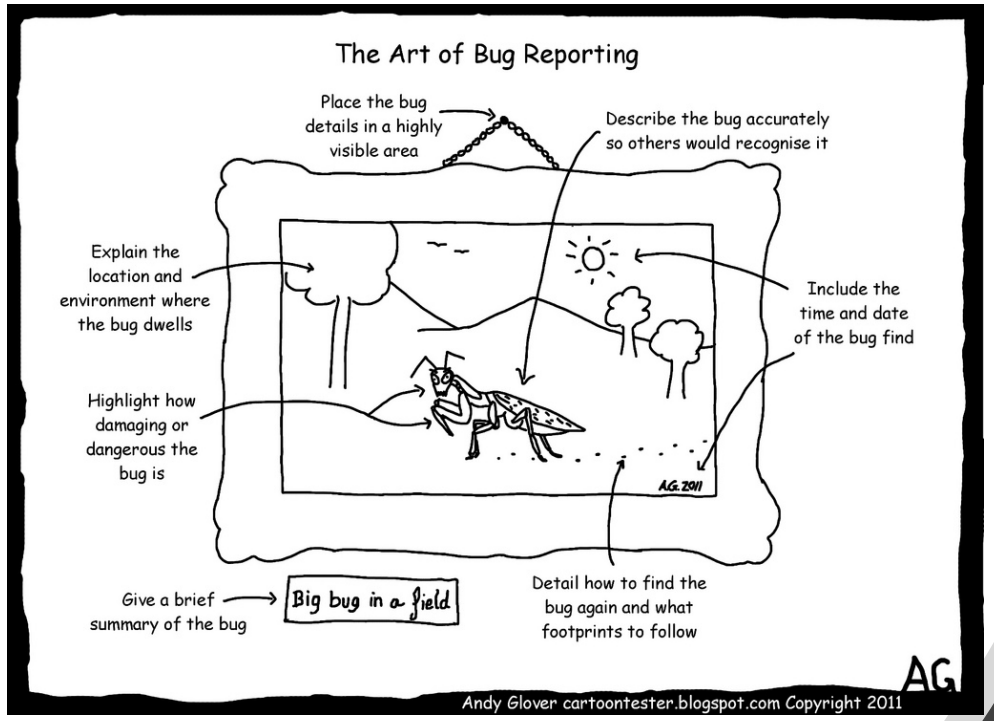
20+ years
In the industry

Soccer coach

Licensed for kids
U10 - U16

Adventurer

16 personalities test



Have you heard this before?

"I tested it how you told me to"

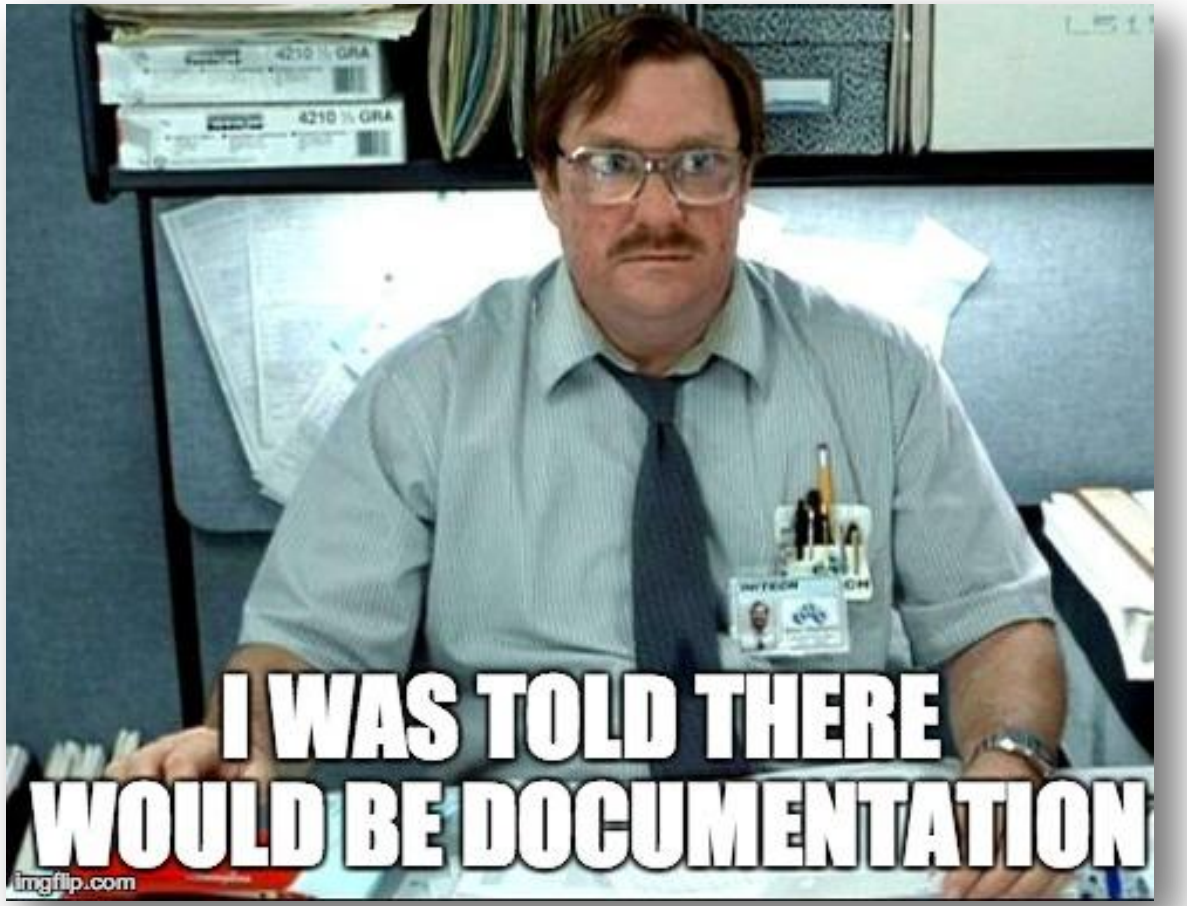
"Acceptance criteria said this is expected"

"I assumed it was fine"

"When you say 'account', you want to mean 'client', right?"

"I have not tested with this input. I didn't know it could happen"

" I didn't know what to test"


I WAS TOLD THERE
WOULD BE DOCUMENTATION
imgflip.com

Have you seeing this before?

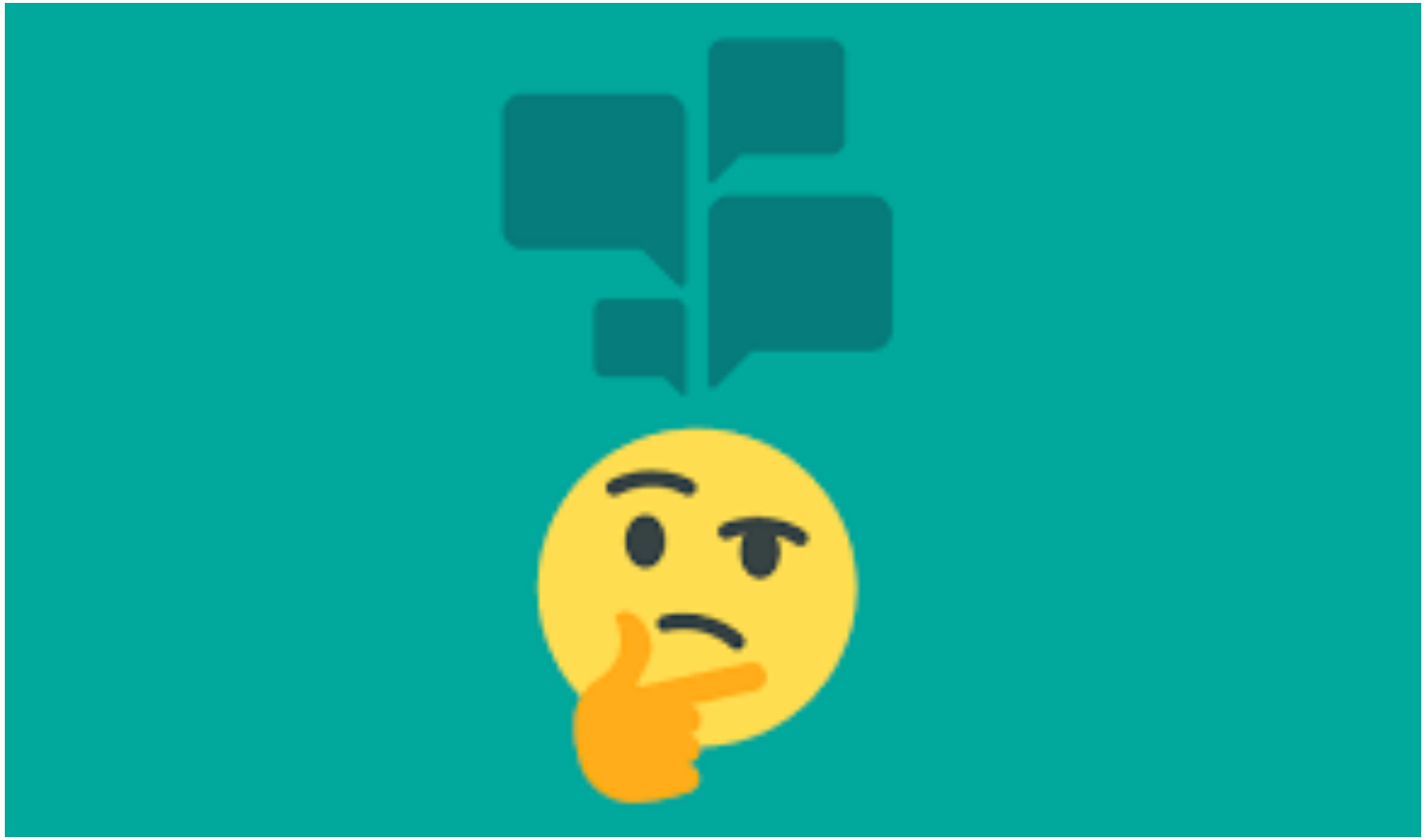
```
        return invTrans;
    }

    public void testBuyTransaction() {
        InvestmentAccount invAccount = (InvestmentAccount) engine.getAccount( accountCode: "11311",  currencySymbol: "EUR");
        AssetAccount assetAccount = (AssetAccount) engine.getAccount( accountCode: "12111",  currencySymbol: "EUR");
        InvestmentTransaction invTrans = buildBuyTransaction(invAccount, assetAccount);

        // test total amount of transaction
        assertEquals(new BigDecimal( v: 1310.50).setScale(2), invTrans.getTotal().setScale(2));

        // test market value of transaction from 3 January up to 8 January
        assertEquals(new BigDecimal( i: 0).setScale(2), invTrans.getMarketValue(TD_minus_1).setScale(2));
        assertEquals(new BigDecimal( v: 1030.50).setScale(2), invTrans.getMarketValue(TD).setScale(2));
        assertEquals(new BigDecimal( v: 1029.50).setScale(2), invTrans.getMarketValue(TD_plus_1).setScale(2));
        assertEquals(new BigDecimal( v: 1027.50).setScale(2), invTrans.getMarketValue(TD_plus_2).setScale(2));
        assertEquals(new BigDecimal( v: 1311.50).setScale(2), invTrans.getMarketValue(SD).setScale(2));
        assertEquals(new BigDecimal( v: 1332.50).setScale(2), invTrans.getMarketValue(SD_plus_1).setScale(2));

        // test cash balance of transaction from 3 January up to 8 January
        assertEquals(new BigDecimal( i: 0).setScale(2), invTrans.getAmount(assetAccount, TD_minus_1).setScale(2));
        assertEquals(new BigDecimal( v: -1030.50).setScale(2), invTrans.getAmount(assetAccount, TD).setScale(2));
        assertEquals(new BigDecimal( v: -1030.50).setScale(2), invTrans.getAmount(assetAccount, TD_plus_1).setScale(2));
        assertEquals(new BigDecimal( v: -1030.50).setScale(2), invTrans.getAmount(assetAccount, TD_plus_2).setScale(2));
        assertEquals(new BigDecimal( v: -1310.50).setScale(2), invTrans.getAmount(assetAccount, SD).setScale(2));
        assertEquals(new BigDecimal( v: -1310.50).setScale(2), invTrans.getAmount(assetAccount, SD_plus_1).setScale(2));
    }

    public void testSellTransaction() {
        InvestmentAccount invAccount = (InvestmentAccount) engine.getAccount( accountCode: "11311",  currencySymbol: "EUR");
        AssetAccount assetAccount = (AssetAccount) engine.getAccount( accountCode: "12111",  currencySymbol: "EUR");
        InvestmentTransaction invTrans = buildSellTransaction(invAccount, assetAccount);

        // test total amount of transaction
        assertEquals(new BigDecimal( v: 1310.50).setScale(2), invTrans.getTotal().setScale(2));
```

What was wrong with that?

Tests weren’t saying what the application does

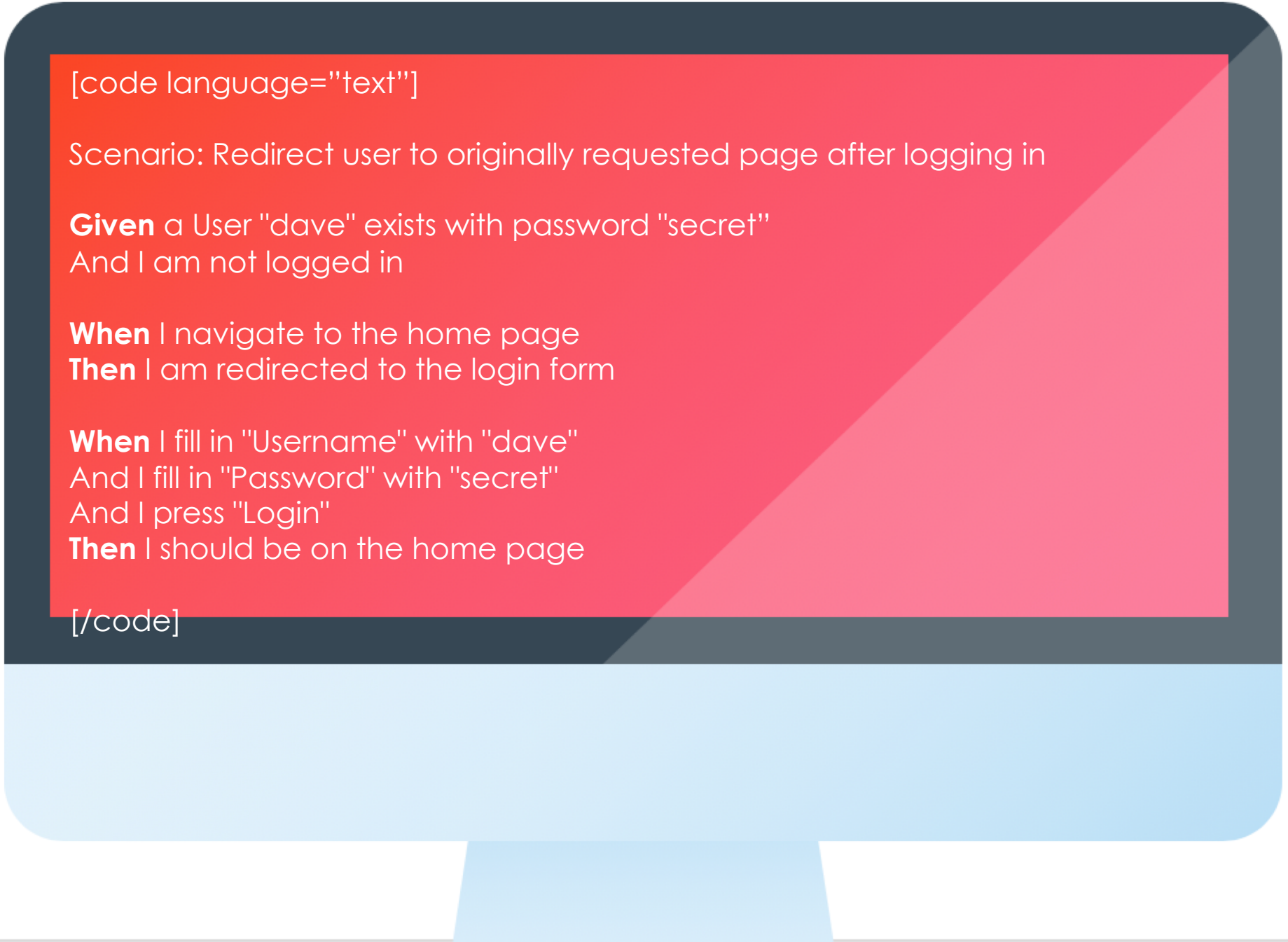
Created by developers, potentially after the code was done

If fails doesn’t tell me nothing. Need to go through the code to figure out something.



Is this TDD?

What is the domain?


[code language="text"]
Scenario: Redirect user to originally requested page after logging in
Given a User "dave" exists with password "secret"
And I am not logged in
When I navigate to the home page
Then I am redirected to the login form
When I fill in "Username" with "dave"
And I fill in "Password" with "secret"
And I press "Login"
Then I should be on the home page
[/code]

What was wrong with that?

Misunderstood BDD : just because its Gherkin does not mean its BDD

Design driven by the tool

Use the “new thing” without knowing how it really works.

What is the story?

What is the expected behavior?

What is the solution?

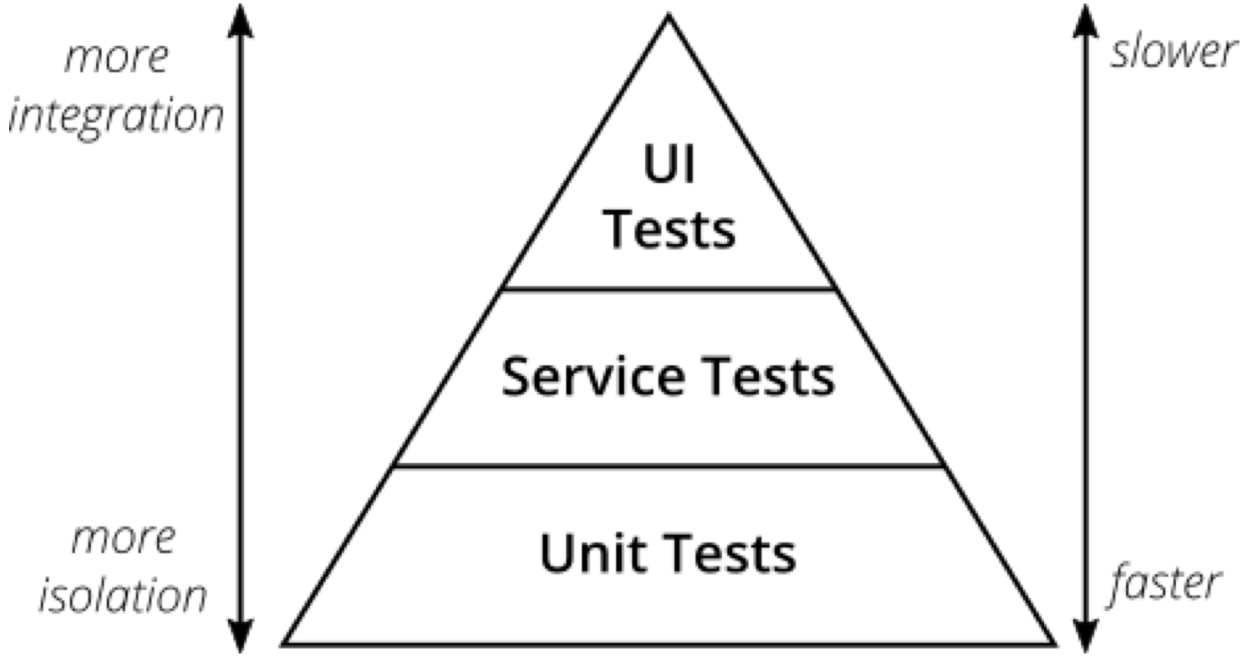
Domain language throughout tests, combining DDD and BDD

Its all about story:

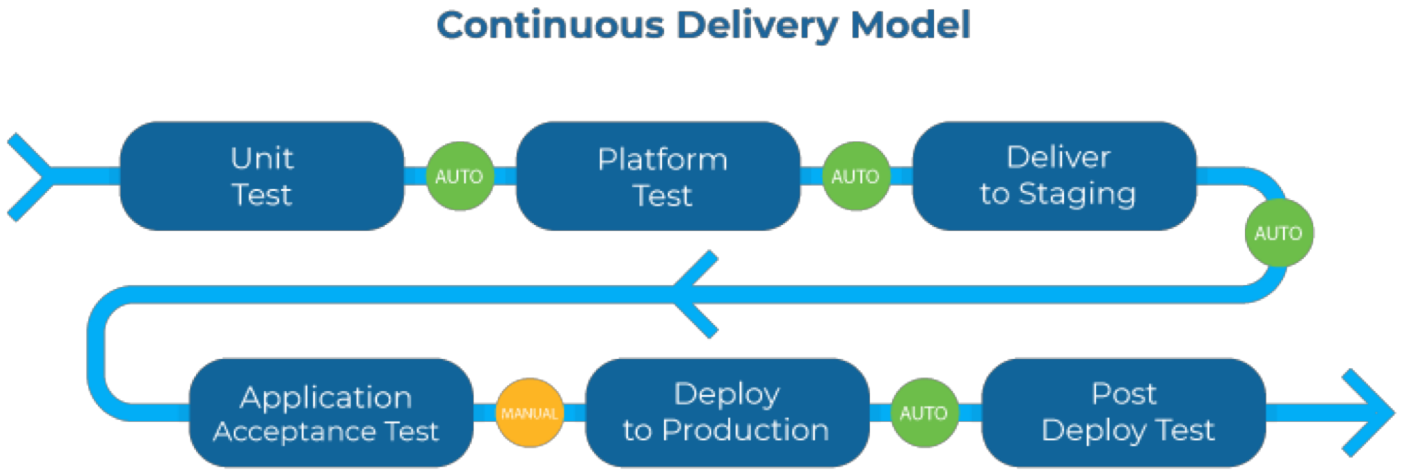
DDD is about empathy. Its about knowing each other , knowing the model.

BDD is all about story and what is the story of what we are building.

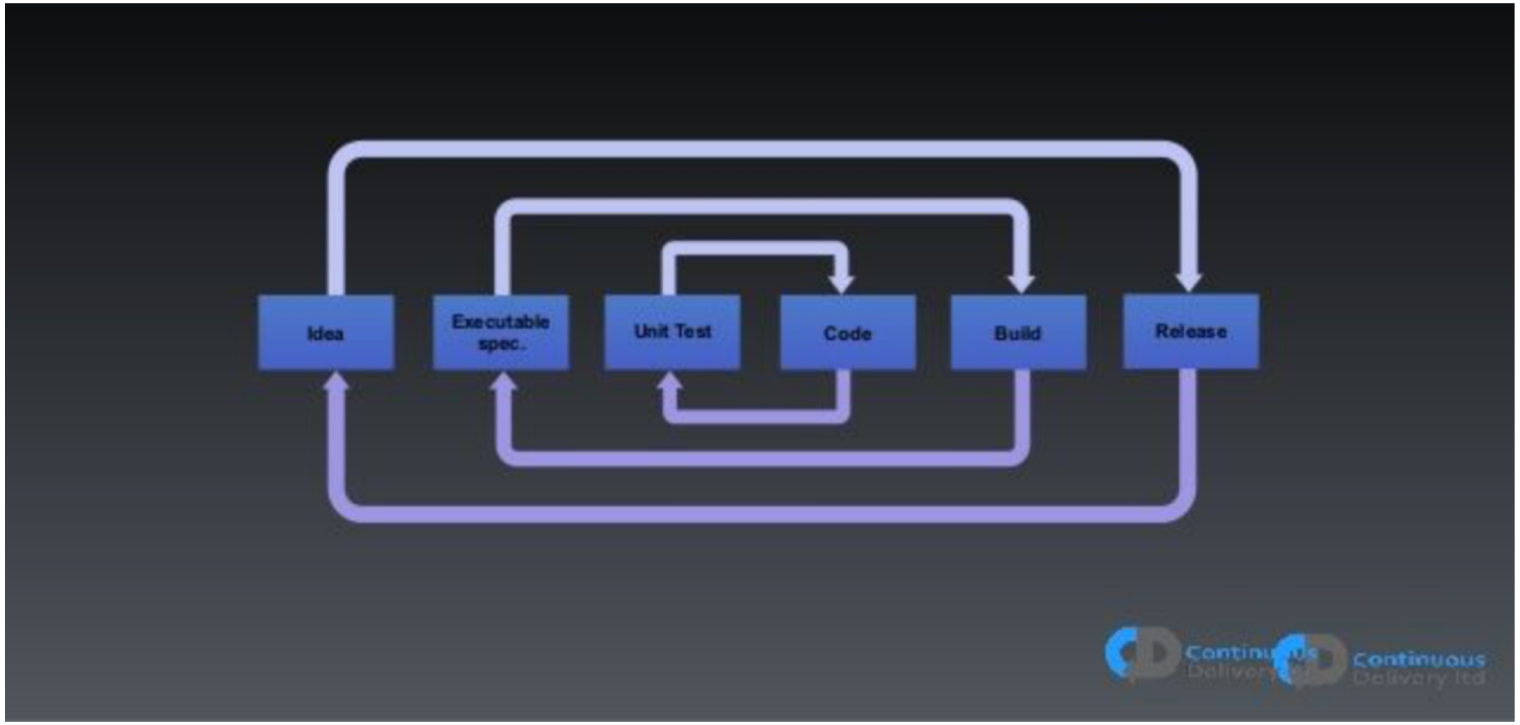
There are many testing pyramids



Testing pyramids are not a good map



Feedback loops



Navigation by public transport

As a commuter

I want to know the fastest route between two places,

In order to get to my destination as quickly as possible

navigation_by_public_transport.spec.ts

```
const expect = global['chai'].expect;
const EC = protractor.ExpectedConditions;

const NAV_BUTTON_CSS = "#searchbox-directions";
const TRAIN_BUTTON_CSS = ".directions-travel-mode-icon.directions-transit-icon";
const FROM_INPUT_CSS = "#sb_ifc51 .tactile-searchbox-input";
const TO_INPUT_CSS = "#sb_ifc52 .tactile-searchbox-input";

const TRAIN_LIST_CSS = ".section-listbox";

describe('Navigation by public transport', () => {

    before( callback: () => {
        // Setup environment data and stuff........
    })

    it('Navigate by public transport', () => {
        browser.get( destination: '/maps');

        let navButtonElement = element(by.css(NAV_BUTTON_CSS));
        browser.wait(EC.visibilityOf(navButtonElement), 5000);
        navButtonElement.click();

        let trainButtonElement = element(by.css(TRAIN_BUTTON_CSS));
        browser.wait(EC.visibilityOf(trainButtonElement), 5000);
        trainButtonElement.click();

        let fromInputElement = element(by.css(FROM_INPUT_CSS));
        browser.wait(EC.visibilityOf(fromInputElement), 5000);
        fromInputElement.sendKeys('Amsterdam Centraal');

        let toInputElement = element(by.css(TO_INPUT_CSS));
        browser.wait(EC.visibilityOf(toInputElement), 5000);
        toInputElement.sendKeys('Utrecht Centraal');
        toInputElement.sendKeys(protractor.Key.RETURN);

        let trainListElement = element(by.css(TRAIN_LIST_CSS));
        expect(trainListElement.isPresent()).to.eventually.be.true;


    })
});
```

navigation_by_public_transport.feature

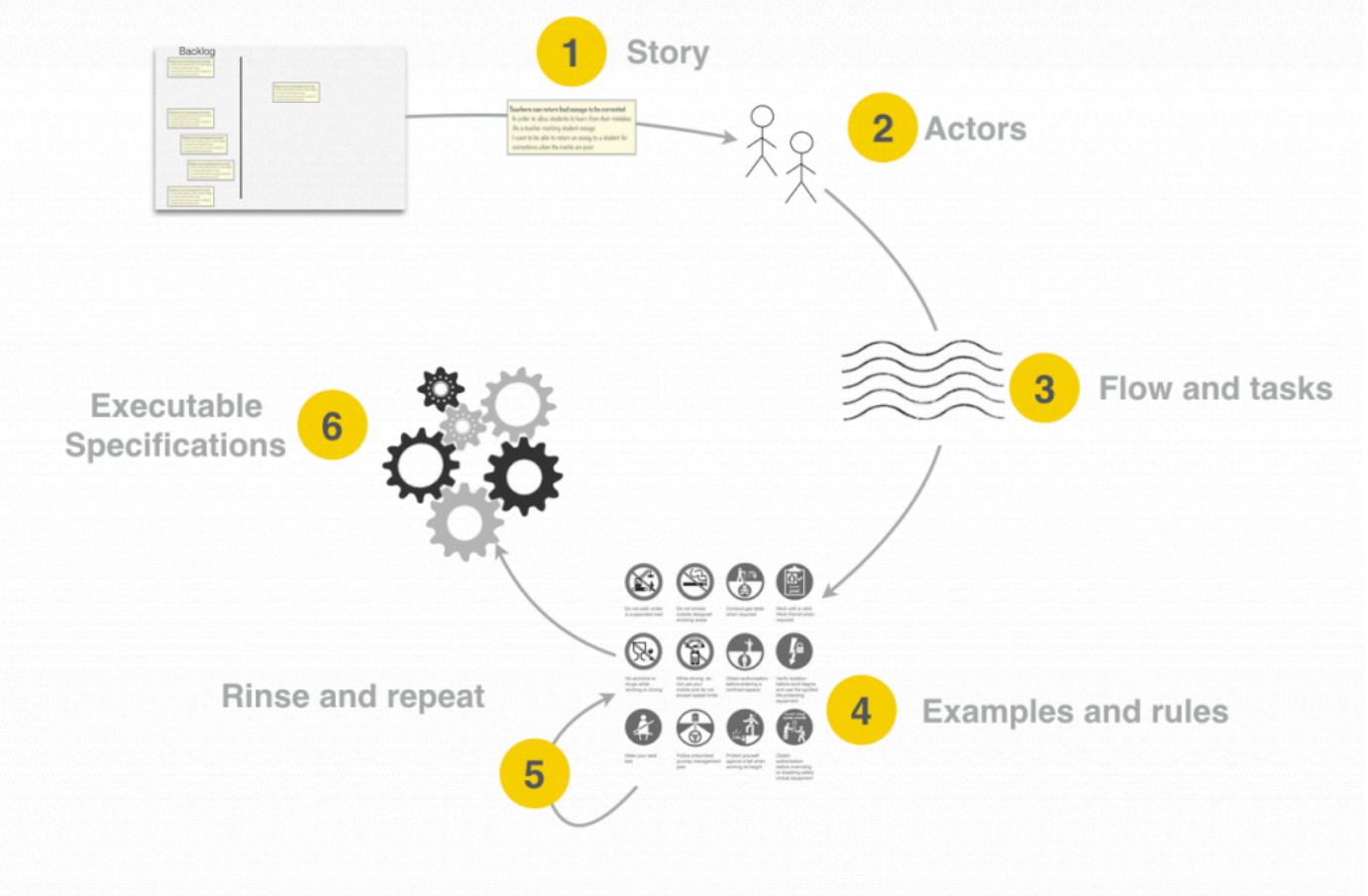
```
Feature: Navigation by public transport

  As a commuter,
  I'd like to know the fastest route between two places,
  So that I get to my destination as quickly as possible

  Scenario: Navigate by public transport

    Given Intercity trains from Utrecht CS leave at 16:58, 17:08, 17:18
    When Connie wants to travel from Utrecht CS to Amsterdam CS at 17:00
    Then she should be told about the trains departing at 17:08, 17:18

```

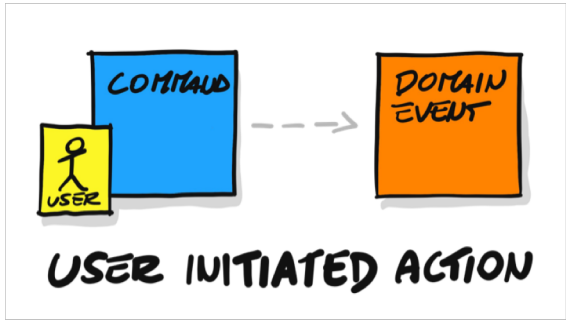
navigation_by_public_transport.ts

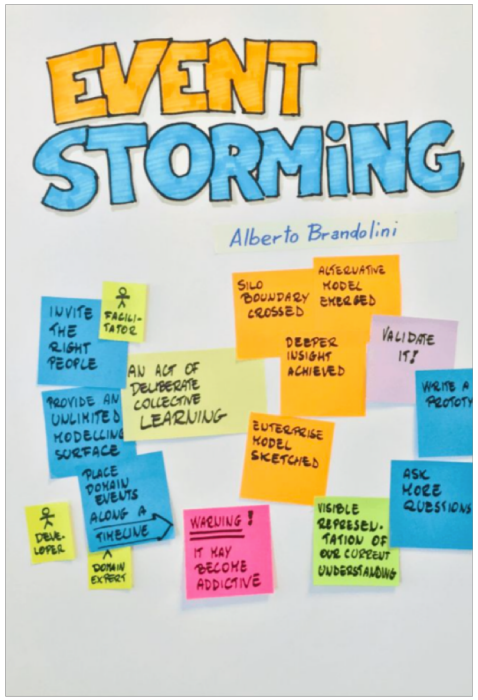
```typescript
import {browser, by, element, protractor} from 'protractor';

const expect = global['chai'].expect;
const EC = protractor.ExpectedConditions;

const NAV_BUTTON_CSS = "#searchbox-directions";
const TRAIN_BUTTON_CSS = ".directions-travel-mode-icon.directions-transit-icon";
const FROM_INPUT_CSS = "#sb_ifc51 .tactile-searchbox-input";
const TO_INPUT_CSS = "#sb_ifc52 .tactile-searchbox-input";

const TRAIN_LIST_CSS = ".section-listbox";

export = function myStepDefinitionsWrapper() {

    this.Given(/^Intercity trains from Utrecht CS leave at 16:58, 17:08, 17:18$/, function (callback) {
        // Setup environment data and stuff........
        callback();
    });
    this.When(/^Connie wants to travel from Utrecht CS to Amsterdam CS at 17:00$/, function () {
        browser.get( destination: '/maps');

        let navButtonElement = element(by.css(NAV_BUTTON_CSS));
        browser.wait(EC.visibilityOf(navButtonElement), 5000);
        navButtonElement.click();

        let trainButtonElement = element(by.css(TRAIN_BUTTON_CSS));
        browser.wait(EC.visibilityOf(trainButtonElement), 5000);
        trainButtonElement.click();

        let fromInputElement = element(by.css(FROM_INPUT_CSS));
        browser.wait(EC.visibilityOf(fromInputElement), 5000);
        fromInputElement.sendKeys('Amsterdam Centraal');

        let toInputElement = element(by.css(TO_INPUT_CSS));
        browser.wait(EC.visibilityOf(toInputElement), 5000);
        toInputElement.sendKeys('Utrecht Centraal');
        return toInputElement.sendKeys(protractor.Key.RETURN);

    });
    this.Then(/^she should be told about the trains departing at 17:08, 17:18$/, function () {
        let trainListElement = element(by.css(TRAIN_LIST_CSS));
        return expect(trainListElement.isPresent()).to.eventually.be.true;
    });
};
```

DDD + BDD in practice:

Feature Mapping!!



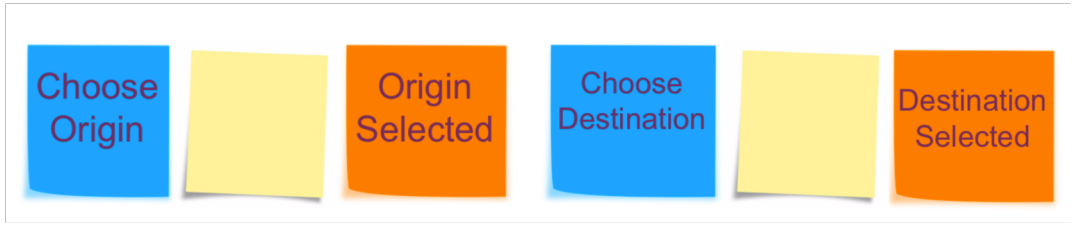




How feature mapping helps on acceptance testing

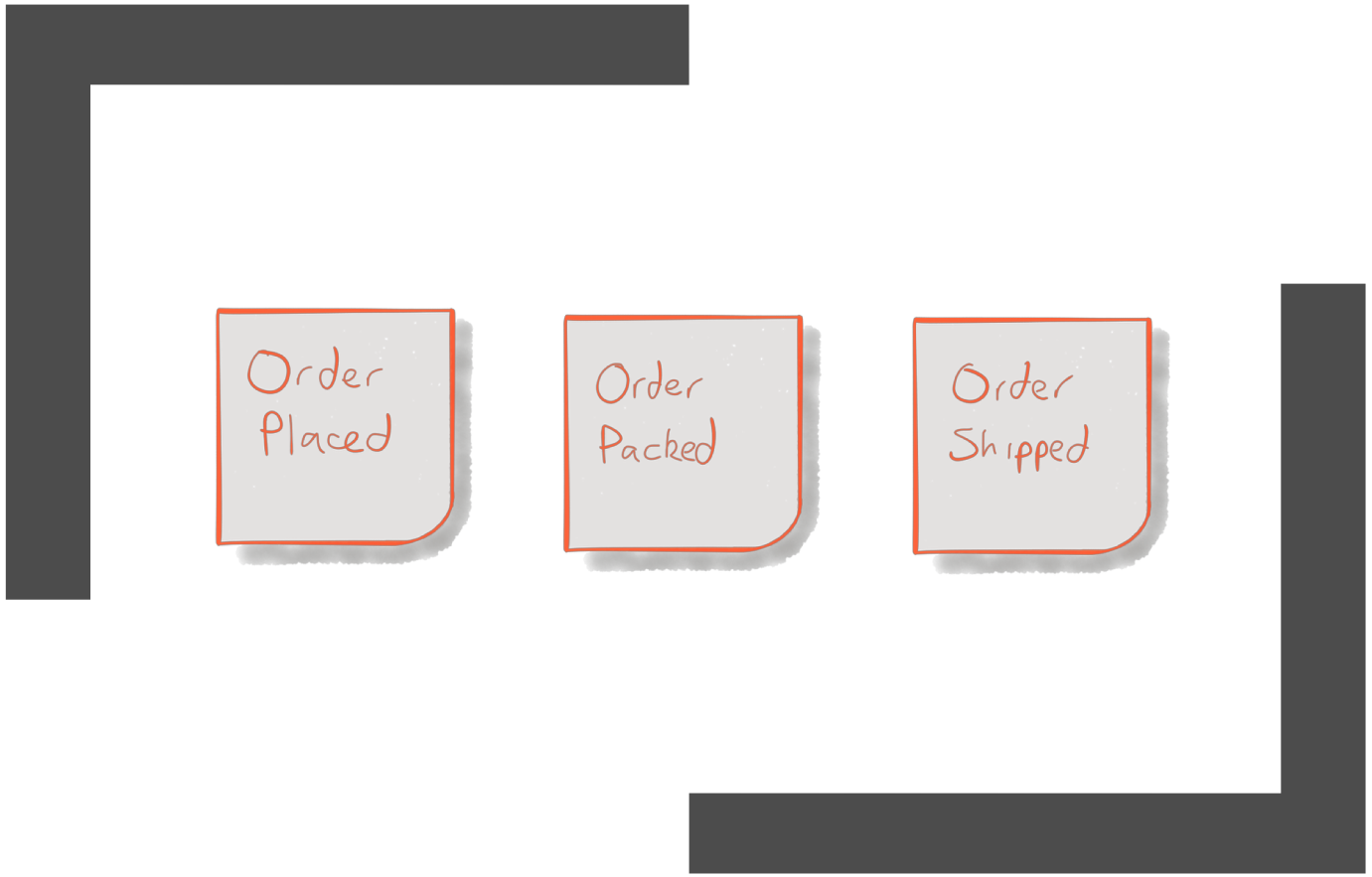
- This technique is all about creating specification based on examples, communicating human language what the software does. It’s a map on how is the flow of feature implementation
- Event storming and Examples and Rules, through making the ”Spec by Example” embedded on your testing code

Event storming



Event Storming is a form of organizational anthropology; it's all about discussing the ***flow*** of events in your organization and modelling that flow in an easy to understand way.

Event flow



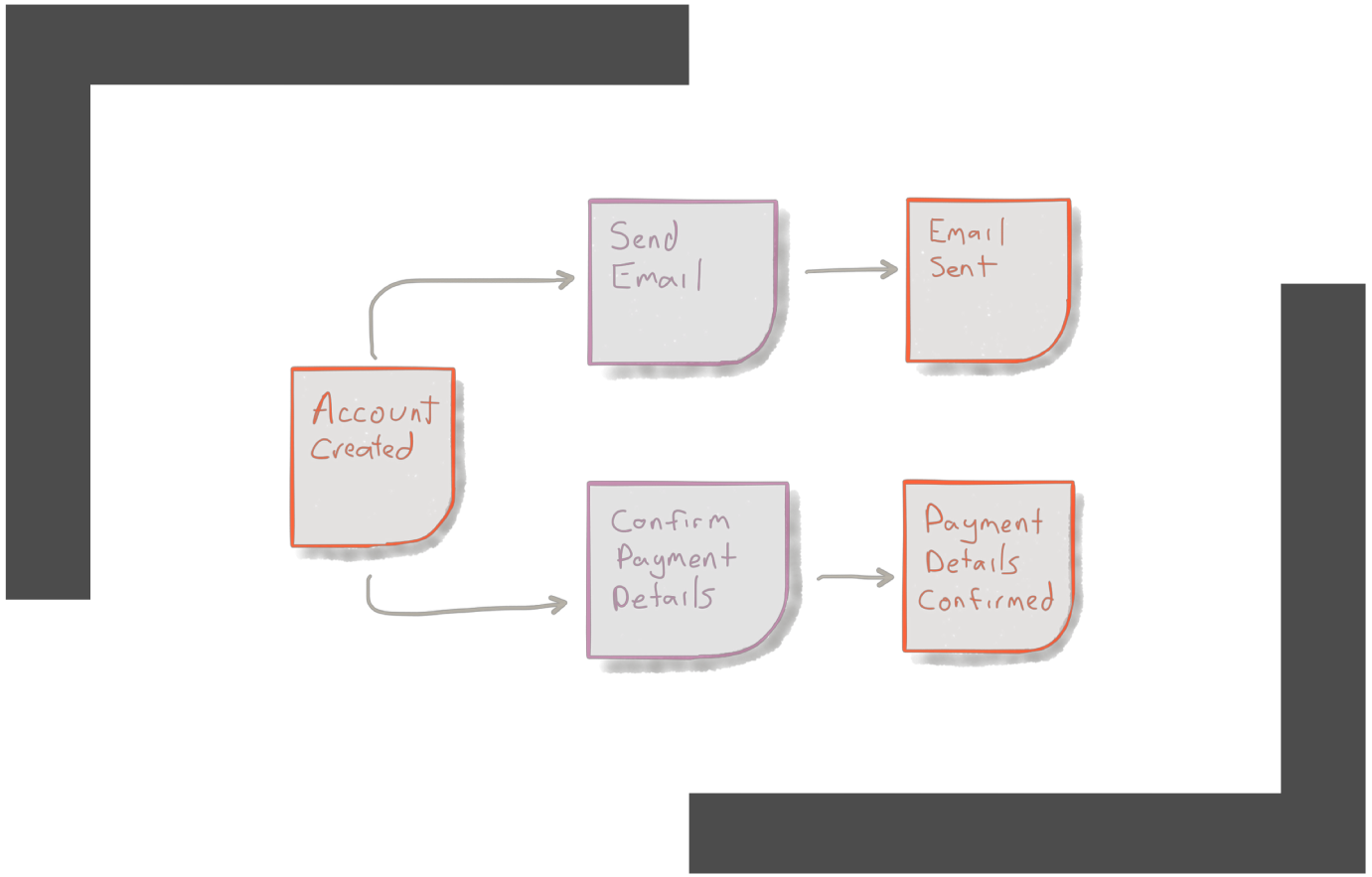
A single event isn't very exciting, so we need to consider events in term of flow — events over time. Orange stickies represent events. To model a business process we simply arrange events from left-to-right in time sequence. That's it! Sounds simple? It is. That's the point. There's no unnecessary ceremony between you, your team, and an interesting discussion that yields results.

Reaction

A *reaction* is something that needs to happen after something else happens. Reactions are best captured in the following statement,

"**This** happens **whenever that** happens". *This* is the reaction. *That* is the event. *Whenever* is the word that associates an event to a reaction to create a *policy*.

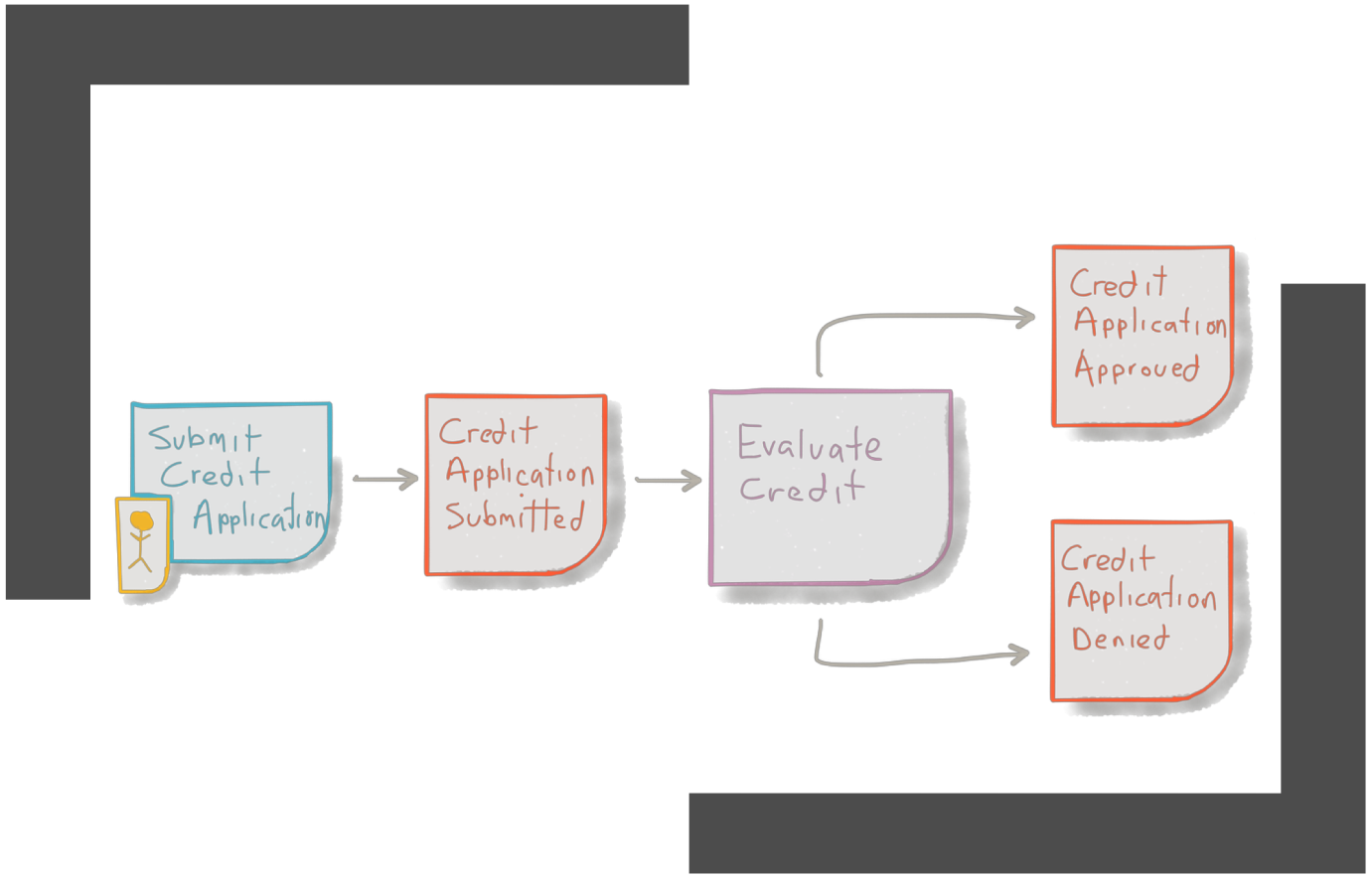
Policy



The flow of events and reactions together is called a *policy* — we call this flow a policy because the flow captures core business rules, such as *"we need to alert a customer whenever someone logs into their account"*.

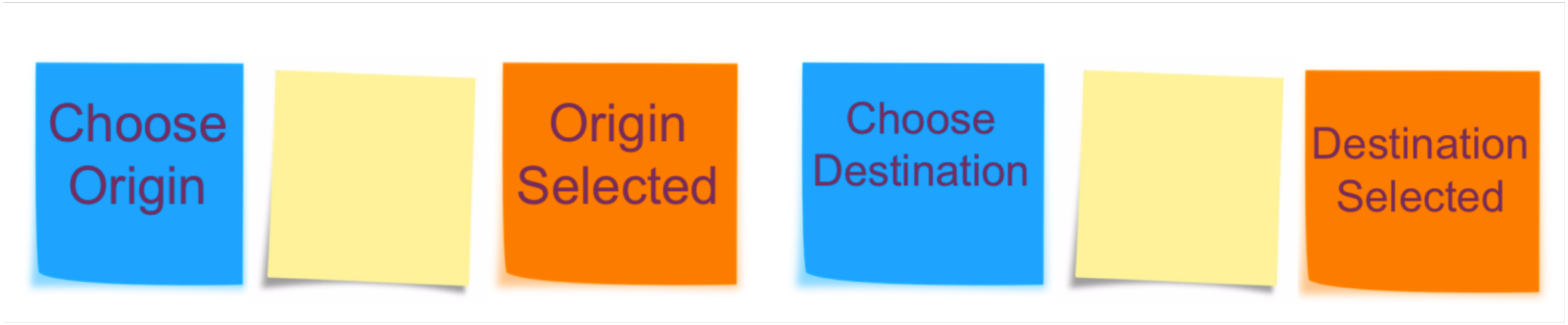
It's also important to note that a reaction — or an entire policy — doesn't need to represent a piece of software. For instance, *"confirm payment details"* may be a manual process that involves an administrator manually verifying wire transfer details.

Command



Eventually we need to dive deeper and map out exactly how users interact with our system; *commands* are critical because they often represent user interactions. Adding user interactions to our event flow brings us towards modelling the complete cause-and-effect of our system.

In our example



Example Mapping

This comes from Cucumber, from BDD

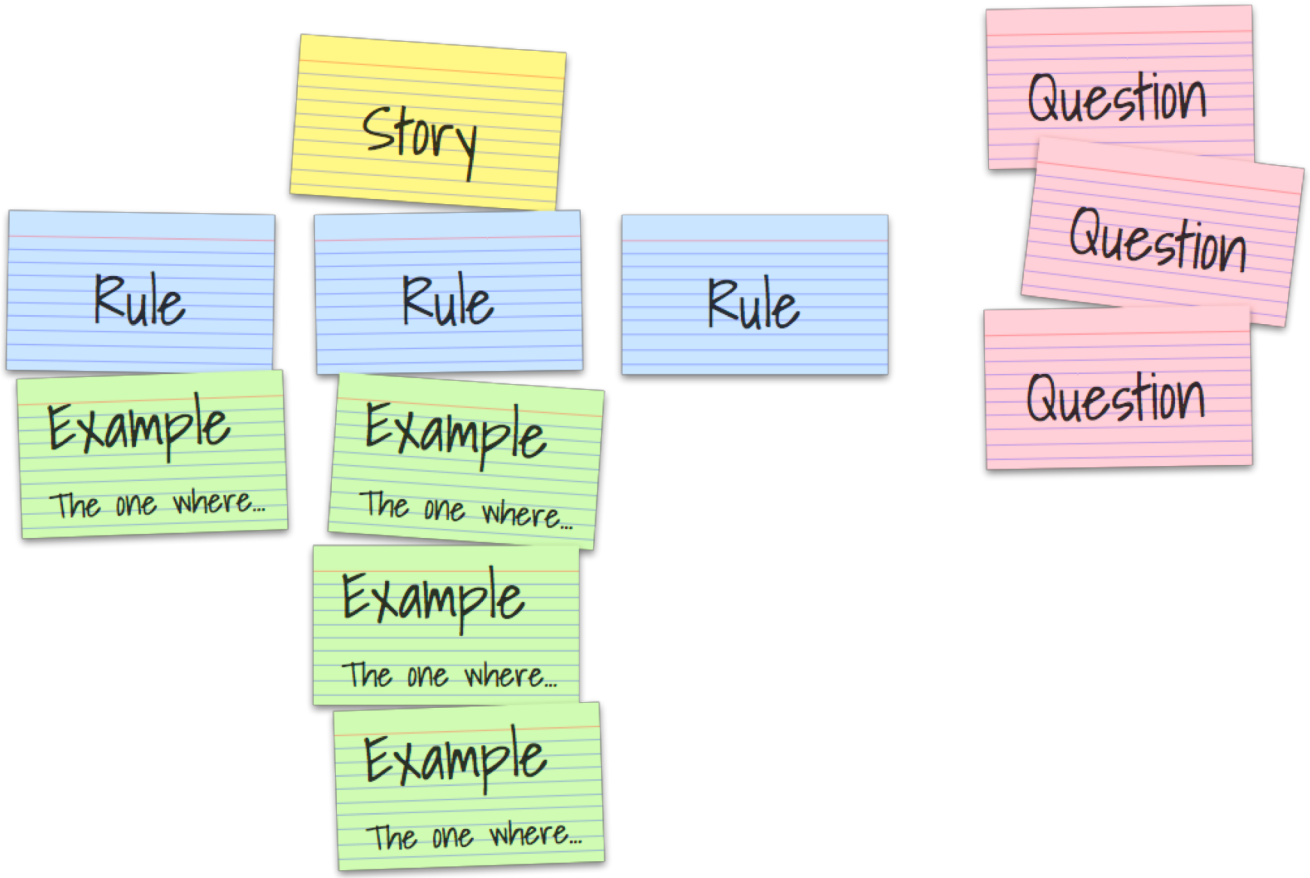
Discovery phase. Flipping out examples

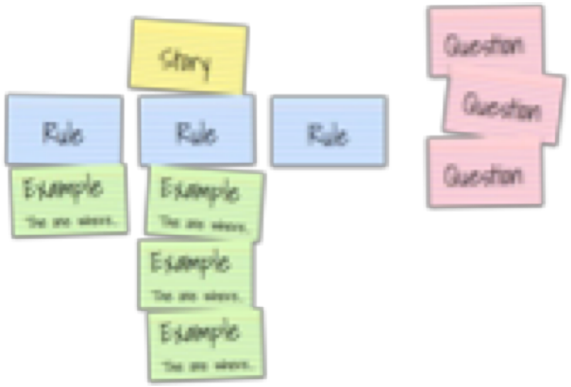
Domain experts needs to be in the room

Get 4 coloured cards

Don’t think initially on Gerkhin. Goes like “friends episode names” : “The one when”

If there is discussion or questions, you park on pink cards if cannot be solved in 15 min or no experts






```
import {ChooseDestination, ChooseOrigin, PlanAJourney} from '../../cucumber/support/saving_time/plan_a_journey';
import {JourneyList} from '../support/saving_time/journey_list';

const expect = global['chai'].expect;

const TRAIN_LIST_CSS = ".section-listbox";

describe('Navigation by public transport', () => {

    before( callback: () => {
        // Setup environment data and stuff........
    })

    it('Navigate by public transport', () => {

        PlanAJourney();
        ChooseOrigin( origin: 'Utrecht Centraal');
        ChooseDestination( destination: 'Amsterdam Centraal');

        expect(JourneyList.Train_List.isPresent()).to.eventually.be.true;

    })
});
```

```
const FROM_INPUT_CSS = "#sb_ifc51 .tactile-searchbox-input";
const TO_INPUT_CSS = "#sb_ifc52 .tactile-searchbox-input";


export function PlanAJourney() {
    browser.get( destination: '/maps');

    let navButtonElement = element(by.css(NAV_BUTTON_CSS));
    browser.wait(EC.visibilityOf(navButtonElement), 5000);
    navButtonElement.click();

    let trainButtonElement = element(by.css(TRAIN_BUTTON_CSS));
    browser.wait(EC.visibilityOf(trainButtonElement), 5000);
    trainButtonElement.click();
}

export function ChooseOrigin(origin: string) {
    let fromInputElement = element(by.css(FROM_INPUT_CSS));
    browser.wait(EC.visibilityOf(fromInputElement), 5000);
    return fromInputElement.sendKeys(origin);
}

export function ChooseDestination(destination: string) {
    let toInputElement = element(by.css(TO_INPUT_CSS));
    browser.wait(EC.visibilityOf(toInputElement), 5000);
    toInputElement.sendKeys(destination);
    return toInputElement.sendKeys(protractor.Key.RETURN);
}
```

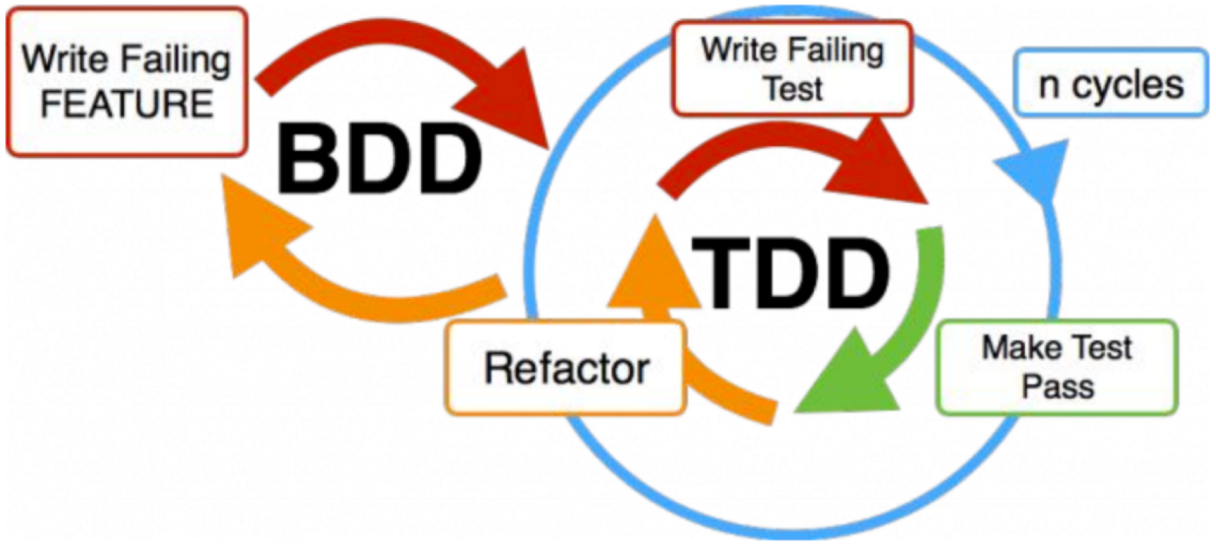
WebDriver

```
import ...

const expect = global['chai'].expect;

const TRAIN_LIST_CSS = ".section-listbox";

export = function myStepDefinitionsWrapper() {

    this.Given(/^Intercity trains from Utrecht CS leave at 16:58, 17:08, 17:18$/,
        // Setup environment data and stuff........
        callback();
    });
    this.When(/^Connie wants to travel from Utrecht CS to Amsterdam CS at 17:00$/,
        PlanAJourney();
        ChooseOrigin( origin: 'Utrecht Centraal');
        return ChooseDestination( destination: 'Amsterdam Centraal');

    });
    this.Then(/^she should be told about the trains departing at 17:08, 17:18$/,
        return expect(JourneyList.Train_List.isPresent()).to.eventually.be.true;
    });
};
```

Cucumber

Conclusion

- Enhance your acceptance testing with executable spec by example (they can be written in Excel, Cucumber, Fitnesse, JSON files, etc…
- Empathy from DDD + Story from BDD + Science = Better testing via quicker feedback (Your CD thanks you)
- It can and should be done from all test levels



Questions?