BDD & CUCUMBER

컨텐츠검색플랫폼팀
조영일

목차

- BDD의 개념
- Behavioral specification
- 예제
- Framework implementations
- Cucumber practice

BDD is a second-generation, outside-in, pull-based, multiple-stakeholder, multiple-scale, high-automation, agile methodology. It describes a cycle of interactions with well-defined outputs, resulting in the delivery of working, tested software that matters.

– Dan North, 2009

BDD의 개념

- Behavior-Driven Development
  - TDD의 특화된 형태

  - 목표
    - 단위 테스트가 아니라 사업적인 이해관계/목적에 따라 테스트가 작성되어야 함
    - 수용 테스트는 사용자 이야기(user story)라는 표준 애자일 프레임웍을 이용해서 작성되어야 함
    - 수용 기준은 시나리오의 형태로 작성되어야 하고 조건, 사건, 결과의 클래스로 구현되어야 함

BDD의 문제의식

프로세스는 어디서 시작할까?

무엇을 테스트하고 무엇을 테스트하지 말까?

한 번에 얼마나 많이 테스트할까?

왜 테스트가 실패했는지를 어떻게 이해할까?

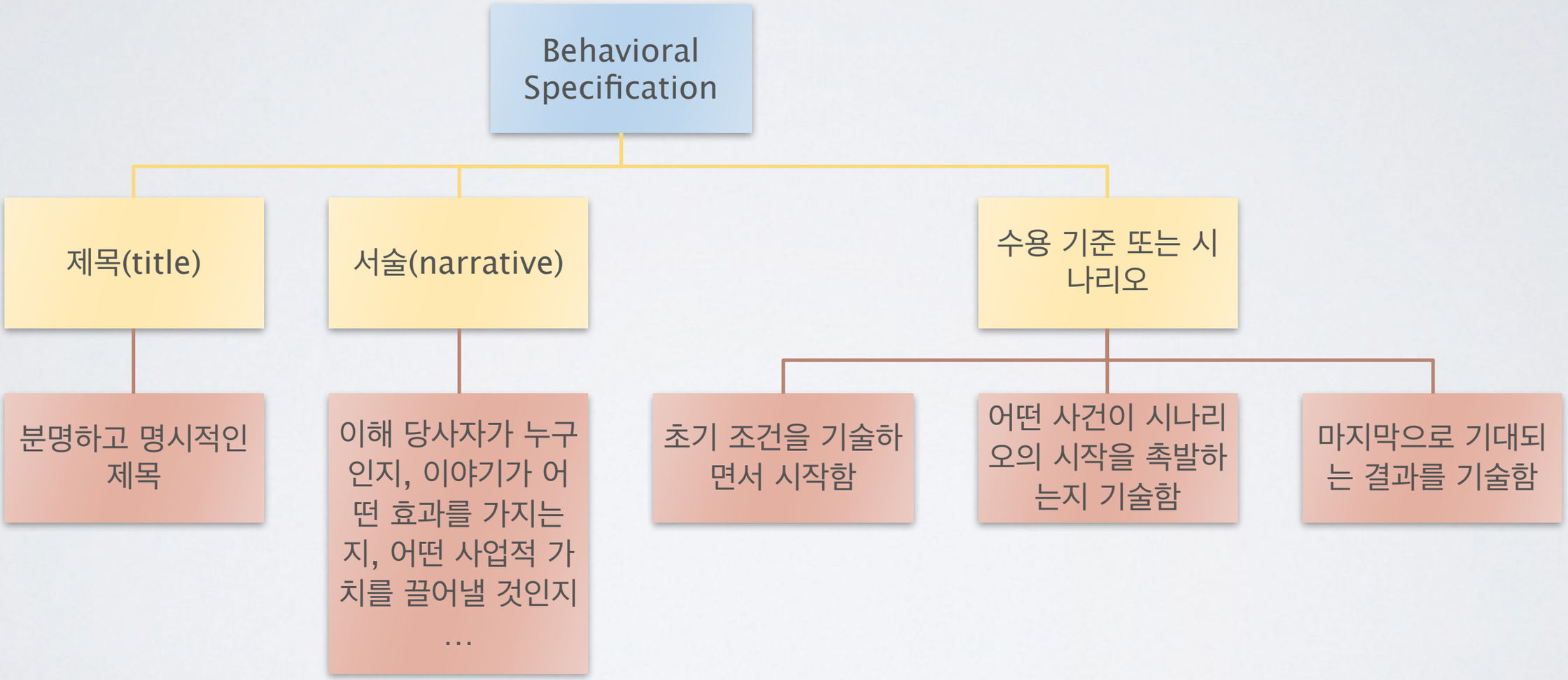
TDD와의 비교

TDD	BDD
기능	행동 with 사업적 가치
단위의 입도가 불명확함	사업적 가치에 의해 보다 명확해짐
inside-out	outside-in

BEHAVIORAL SPECIFICATION

- OO 분석/설계에서 나온 개념인 사용자 이야기에서 빌려온 개념
- 사업 분석가와 개발자가 문서화해서 협업해야 함

BEHAVIORAL SPECIFICATION 의 구조



예제

Story: Returns go to stock

In order to keep track of stock
As a store owner
I want to add items back to stock when they're returned

Scenario 1: Refunded items should be returned to stock
Given a customer previously bought a black sweater from me
And I currently have three black sweaters left in stock
When he returns the sweater for a refund
Then I should have four black sweaters in stock

Scenario 2: Replaced items should be returned to stock
Given that a customer buys a blue garment
And I have two blue garments in stock
And three black garments in stock.
When he returns the garment for a replacement in black,
Then I should have three blue garments in stock
And two black garments in stock

예제

이야기: 반품은 재고가 된다.

목적: 재고를 추적하기 위해
자격: 상점 주인으로서
기대: 상품이 반품되면 재고로 되돌려놓음

시나리오1: 환불된 상품은 재고로 돌려놔야 한다.
조건: 고객이 나한테서 전에 검정 스웨터를 샀었다.
그리고 나는 현재 3개 검정 스웨터가 재고로 남아 있다.
사건: 그가 그 스웨터를 환불하려고 한다.
결과: 나는 재고로 검정 스웨터 4개를 가져야 한다.

시나리오2: 교환된 상품은 재고로 돌려놔야 한다.
조건: 고객이 파란색 의상을 구입한다.
그리고 나는 2개의 파란색 의상을 재고로 가지고 있고
그리고 3개의 검정 의상을 재고로 가지고 있다.
사건: 그가 그 의상을 검정으로 교환해달라고 한다.
결과: 나는 3개의 파란색 의상을 재고로 가지게 되고
그리고 2개의 검정 의상을 재고로 가지게 된다.

FRAMEWORK IMPLEMENTATIONS

IBEHAVE

text

Scenario: A trader is alerted of status

Given a stock and a threshold of 15.0
When stock is traded at 5.0
Then the alert status should be OFF
When stock is traded at 16.0
Then the alert status should be ON

POJO

Mapping Stories

```
public class TraderSteps {
    private TradingService service;   // Injected
    private Stock stock; // Created

    @Given("a stock and a threshold of $threshold")
    public void aStock(double threshold) {
        stock = service.newStock("STK", threshold);
    }
    @When("the stock is traded at price $price")
    public void theStockIsTraded(double price) {
        stock.tradeAt(price);
    }
    @Then("the alert status is $status")
    public void theAlertStatusIs(String status) {
        assertThat(stock.getStatus().name(), equalTo(status));
    }
}
```

```
public class TraderStories extends JUnitStories {

    public Configuration configuration() {
        return new MostUsefulConfiguration()
            .useStoryLoader(new LoadFromClasspath(this.getClass()))
            .useStoryReporterBuilder(new StoryReporterBuilder()
                .withCodeLocation(codeLocationFromClass(this.getClass()))
                .withFormats(CONSOLE, TXT, HTML, XML));
    }

    public List<CandidateSteps> candidateSteps() {
        return new InstanceStepsFactory(configuration(),
             new TraderSteps(new TradingService())).createCandidateSteps();
    }

    protected List<String> storyPaths() {
        return new StoryFinder().findPaths(codeLocationFromClass(this.getClass()),
"**/*.story");
    }
}
```

RSPEC

```ruby
class Burger
  attr_reader :options

  def initialize(options={})
    @options = options
  end

  def apply_ketchup
    @ketchup = @options[:ketchup]
  end

  def has_ketchup_on_it?
    @ketchup
  end
end
```

```ruby
describe Burger do
  describe "#apply_ketchup" do
    subject { burger }
    before  { burger.apply_ketchup }

    context "with ketchup" do
      let(:burger) { Burger.new(:ketchup => true) }

      it { should have_ketchup_on_it }
    end

    context "without ketchup" do
      let(:burger) { Burger.new(:ketchup => false) }

      it { should_not have_ketchup_on_it }
    end
  end
end
```

CUCUMBER

- Behavioral Specification
    - Feature: Addition
        - In order to avoid silly mistakes
        - As a math idiot
        - I want to be told the sum of two numbers
        - Scenario: Add two numbers
            - Given I have entered 50 into the calculator
            - And I have entered 70 into the calculator
            - When I press add
            - Then the result should be 120 on the screen

- Step definition
    - Given /I have entered (.*) into the calculator/ do |n|
        - calculator ||= Calculator.new
          calculator.push(n.to_i)
    - end

CUCUMBER

- 최초 실행

```
Feature: Addition
    In order to avoid silly mistakes
    As a math idiot
    I want to be told the sum of two numbers

Scenario: Add two numbers                          # features/first.feature:6
  Given I have entered 50 into the calculator # features/step_definitons/calculator_steps.rb:14
    undefined method `push' for #<Calculator:0x007fefd093e3e8> (NoMethodError)
    ./features/step_definitons/calculator_steps.rb:15:in `/I have entered (\d+) into the calculator/'
    features/first.feature:7:in `Given I have entered 50 into the calculator'
  And I have entered 70 into the calculator   # features/step_definitons/calculator_steps.rb:14
  When I press add                            # features/step_definitons/calculator_steps.rb:18
  Then the result should be 120 on the screen # features/step_definitons/calculator_steps.rb:22
```

CUCUMBER

- Calculator 클래스를 정의, push method 작성

```
Feature: Addition
    In order to avoid silly mistakes
    As a math idiot
    I want to be told the sum of two numbers

Scenario: Add two numbers                          # features/first.feature:6
  Given I have entered 50 into the calculator      # features/step_definitons/calculator_steps.rb:14
  And I have entered 70 into the calculator        # features/step_definitons/calculator_steps.rb:14
  When I press add                                 # features/step_definitons/calculator_steps.rb:18
    undefined method `add' for #<Calculator:0x007fb4750b5690 @args=[50, 70]> (NoMethodError)
    ./features/step_definitons/calculator_steps.rb:19:in `/I press (\w+)/'
    features/first.feature:9:in `When I press add'
  Then the result should be 120 on the screen      # features/step_definitons/calculator_steps.rb:22
```

CUCUMBER

- add method 작성

```
Feature: Addition
     In order to avoid silly mistakes
     As a math idiot
     I want to be told the sum of two numbers

Scenario: Add two numbers                          # features/first.feature:6
  Given I have entered 50 into the calculator # features/step_definitons/calculator_steps.rb:14
  And I have entered 70 into the calculator   # features/step_definitons/calculator_steps.rb:14
  When I press add                                 # features/step_definitons/calculator_steps.rb:18
  Then the result should be 120 on the screen # features/step_definitons/calculator_steps.rb:22
    expected: 120.0
         got: 121 (using ==) (RSpec::Expectations::ExpectationNotMetError)
    ./features/step_definitons/calculator_steps.rb:23:in `/the result should be (.*) on the screen/'
    features/first.feature:10:in `Then the result should be 120 on the screen'
```

CUCUMBER

- add method 수정

```
Feature: Addition
    In order to avoid silly mistakes
    As a math idiot
    I want to be told the sum of two numbers

  Scenario: Add two numbers                          # features/first.feature:6
    Given I have entered 50 into the calculator      # features/step_definitons/calculator_steps.rb:14
    And I have entered 70 into the calculator        # features/step_definitons/calculator_steps.rb:14
    When I press add                                 # features/step_definitons/calculator_steps.rb:18
    Then the result should be 120 on the screen      # features/step_definitons/calculator_steps.rb:22

1 scenario (1 passed)
4 steps (4 passed)
```

국제화

```
# language: ko
기능: 나눗셈
  예기치못한 실수을 방지하기 위해
  분수를 계산 능력 요구한다

시나리오: 보통 숫자
  조건 계산기에 3을 입력했음
  그리고 계산기에 2을 입력했음
  만일 내가 divide를 누르면
  그러면 화면에 출력된 결과는 1.5이다
```

테이블 형태의 테스트 케이스 입력

```
# language: ko
기능: 덧셈
  예기치못한 실수을 방지하기 위해
  수학을 잘 못하는 사람으로써
  두숫자의 합을 알고 싶다

  시나리오 개요: 두 숫자를 더하기
    조건 계산기에 <입력1>을 입력했음
    그리고 계산기에 <입력2>을 입력했음
    그리고 계산기에 <입력3>을 입력했음
    만일 내가 <버튼>를 누르면
    그러면 화면에 출력된 결과는 <결과>이다

  예:
    | 입력1 | 입력2 | 입력3 | 버튼 | 결과 |
    | 20    | 30    | 40    | add  | 90   |
    | 2     | 5     | 7     | add  | 14   |
    | 0     | 40    | 20    | add  | 60   |
```

CUCUMBER IMPLEMENTATIONS

- Implementations
  - Cucumber 자체는 Ruby 기반이지만 다른 언어와 연동되는 구현들을 가지고 있음
  - Java, Flex, Python, Perl, Lua, Erlang, PHP, Javascript, C++, C#/F# .Net
  - Selenium
  - Ruby on Rails
- Cucumber-JVM
  - Pure-java implementation
  - Spring 등의 dependency injection container와 통합됨
  - Maven의 pom.xml을 통해 container나 junit와 연동됨

CUCUMBER-JVM

- pom.xml

```xml
<dependencies>
    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>cucumber-picocontainer</artifactId>
        <version>1.1.1</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>cucumber-junit</artifactId>
        <version>1.1.1</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.10</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```