Behavior Driven Development

Isaac Griffith

CS 2263
Department of Informatics and Computer Science
Idaho State University

Outcomes

After today's lecture you will be able to:

- Define behavior-driven development
- Understand the basic principles of BDD
- Convert between User Stories and BDD Specifications
- Understand some of the uses of BDD

Inspiration

“If the system’s not behaving, what is it doing?”

Behavior-Driven Development

Concept

- Behavior-Driven Development (BDD) is a term used to classify a method to build software by describing the application behavior from the perspective of an what is of value to the stakeholders.
- Other terms associated with the same concept –
  - Agile Acceptance Testing
  - Acceptance Test-Driven Development
  - Example-Driven Development
  - Code By Example
  - Story Testing
  - Story Test-Driven Development
  - Specification by Example

Definition

”Behavior-Driven Development (BDD) is a second-generation, outside-in, pull-based, multi-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” – Dan North (Creator of BDD)

BDD Principles

- Just Enough
- Deliver stakeholder value
- Behavior only

Idaho State University
Computer Science

User Story Template

Typically Seen Template

As a [role]
I want [feature]
So that [benefit]

A Modified Template

Title [title of the story]
In order to [benefit]
A [role]
Wants to [feature]

ROAR

User Story Example

- **Title**: Register customers for VIP program
  In order to be able to do direct marketing of products to existing customers,
  a marketing manager
  wants customers to register personal details by joining a VIP program.
- **Title**: Free delivery for VIP customers
  In order to entice existing customers to register for the VIP program,
  a marketing manager
  wants the system to offer free delivery on certain items to VIP customers

Scenario Template

Title [title of the scenario]

Given [some initial context / system State]

And [more context / system State]

When [an event occurs / user Action occurs]

Then [ensure outcome / system Reaction]

And [some outcomes / system Reactions]

User Story & Scenario Example

- **Title**: Customer withdraws cash
  In order to not want to wait in line at the bank,
  a customer
  wants to withdraw cash from the bank ATM
- **Title**: Account is overdrawn past the overdraft limit
  Given the account is overdrawn
  And the card is valid
  When the customer request for cash withdrawal
  Then ensure a rejection message is displayed
  And ensure the cash is not dispensed
  And ensure the card is returned

BDD Characteristics

- Ubiquitous Language
- Iterative Decomposition Process
- Plain text description using Story and Scenarios templates
- Automated Acceptance Testing with Mapping Rules
- Readable Behavior Oriented Specification & code
- Behavior Driven at different levels

BDD Tools

- JBehave, NBehave
- RSpec, MSpec, Spock
- StoryQ, Cucumber, SpecFlow

BDD Anti-Patterns

- BDD is a framework with keywords & flavors
- BDD is for defining system behavior and TDD for lower level components
- BDD is for business applications and TDD is for API libraries
- BDD is too explicit, verbose.
- BDD is for higher level, TDD is for lower level
- BDD is for integration testing, TDD is for unit testing
- BDD is outside-in, TDD is inside-out

Acceptance testing

- Acceptance tests are roughly tests corresponding to use-cases: one per use-case.
- They should test the customer-facing side of the app: is it “acceptable” to customers?

Behavior-Driven Development

BDD is a relatively new approach to acceptance testing

- BDD-style acceptance tests start off in a format similar to a use-case scenario (a story): a linear sequence of steps in English.
- A precise mapping of that English on to code is defined.
- So, its “just acceptance tests” but you don’t have to stare at a pile of code to decipher the test.
- BDD slowly is gaining acceptance and may be mainstream eventually

Integration Testing

- Integration tests are similar to acceptance tests - they also test the whole system
- But, integration tests are low-level things about whole system, not customer-facing
- If we were testing the front-end by automatically clicking on buttons that would be an acceptance test: “is the whole app performing acceptably?”

CI Services for testing

- One challenge of CI is individual developers have different build setups on their laptops
- A relatively recent solution is to use a CI service to run your build on a blank box and then run all your tests.
- The CI service defines the “gold standard” of success or fail of tests.
- The CI service requires a fully automated build and test process as a prerequisite
  - you should have this anyway; putting a CI service in the loop forces this

CI Services for testing

- Why Github and Travis together? For every push to master, Travis gets notified and builds project and runs your test suite!
- Jenkins is a service similar to Travis which you must run yourself on your own computer.
- CI servers can also deploy for you, e.g. to Heroku.
- Its harder to test front-ends on CI servers and we don’t require that for your projects.

Testing UIs

- Its hard to automate the input aspects of forms, scrollbars, etc.
- But things keep improving in terms of tools, and eventually it should be commonplace.
- Android – see Android UI testing best practices for more details.
- JavaScript Web front-ends – see e.g. TestCafe for programmatic testing of JavaScript web front-ends.
- Unit tests are for one component only; how to deal with missing components you may be interacting with? Mock them up; see e.g. mockito for a framework to help with Java mocking, etc.

Are there any questions?