BDD

Sujata Batra

BDD

"BDD is using examples at multiple levels to create a shared understanding and surface uncertainty to deliver software that matter." –According to Dan North(responsible for evolution of BDD)

Sujata Batra

BDD Key Features

- Providing a shared process and shared tools promoting communication to the software developers, business analysts and stakeholders to collaborate on software development, with the aim of delivering product with business value.
- What a system should do and not on how it should be implemented.
- Providing better readability and visibility.
- Verifying not only the working of the software but also that it meets the customer's expectations.

BDD Practice

- The two main practices of BDD are –
  - Specification by Example (SbE)
  - Test Driven Development (TDD)

Specification by Example

- Specification by Example (SbE) uses examples in conversations to illustrate the business rules and the behavior of the software to be built.

- Specification by Example enables the product owners, business analysts, testers and the developers to eliminate common misunderstandings about the business requirements.

Test Driven Development

- Test Driven Development, in the context of BDD, turns examples into human readable, executable specifications.

- The developers use these specifications as a guide to implement increments of new functionality. This, results in a lean codebase and a suite of automated regression tests that keep the maintenance costs low throughout the lifetime of the software.

Agile BDD

In Agile software development, BDD method is used to come to a common understanding on the pending specifications.

The following steps are executed in Agile BDD −

- The developers and the product owner collaboratively write pending specifications in a plain text editor.
- The product owner specifies the behaviors they expect from the system.
- The developers
  - Fill the specifications with these behavior details.
  - Ask questions based on their understanding of the system.
- The current system behaviors are considered to see if the new feature will break any of the existing features.

Agile Manifesto and BDD

BDD aligns itself to the Agile manifesto as follows –

Agile Manifesto	BDD Alignment
Individuals and interactions over processes and tools.	BDD is about having conversations.
Working software over comprehensive documentation.	BDD focuses on making it easy to create software that is of business value.
Customer collaboration over contract negotiation.	BDD focuses on scenarios based on ideas with continuous communication with the customer as the development progresses. It is not based on any promises.
Responding to change over following a plan.	BDD focuses on continuous communication and collaboration that facilitates absorption of changes.

TDD Vs BDD

- TDD describes how the software works.
- On the other hand, BDD –
  - Describes how the end user uses the software.
  - Fosters collaboration and communication.
  - Emphasizes on examples of behavior of the System.
  - Aims at the executable specifications derived from the examples

TDD in a BDD way

TDD focuses on how something will work, BDD focuses on why we build it at all.

BDD answers the following questions often faced by the developers –

Question	Answer
Where to start?	outside-in
What to test?	User Stories
What not to test?	anything else

These answers result in the story framework as follows –

Story Framework

As a **[Role]**

I want **[Feature]**

so that **[Benefit]**

- This means, 'When a **Feature** is executed, the resulting **Benefit** is to the Person playing the **Role.**'

TDD in a BDD way

These answers result in the Example framework as follows –

Example Framework

Given some initial context,

When an event occurs,

Then ensure some outcomes.

This means, ‘Starting with the initial context, when a particular event happens, we know what the outcomes should be.’

Story and Scenarios

Story

As a customer,

I want to withdraw cash from an ATM,

so that I do not have to wait in line at the bank.

Sujata Batra

Story and Scenarios

There are two possible scenarios for this story.

Scenario 1 – Account is in credit

Given the account is in credit

And the card is valid

And the dispenser contains cash

When the customer requests cash

Then ensure the account is debited

And ensure cash is dispensed

And ensure the card is returned

Story and Scenarios

Scenario 2 – Account is overdrawn past the overdraft limit

Given the account is overdrawn

And the card is valid

When the customer requests cash

Then ensure a rejection message is displayed

And ensure cash is not dispensed

And ensure the card is returned

Sujata Batra

Development Cycle

- The Development Cycle for BDD is an **outside-in** approach.
- **Step 1** – Write a high-level (outside) business value example (using Cucumber or RSpec/Capybara) that goes red. (RSpec produces a BDD framework in the Ruby language)
- **Step 2** – Write a lower-level (inside) RSpec example for the first step of implementation that goes red.
- **Step 3** – Implement the minimum code to pass that lower-level example, see it go green.
- **Step 4** – Write the next lower-level RSpec example pushing towards passing Step 1 that goes red.
- **Step 5** – Repeat steps Step 3 and Step 4 until the high-level example in Step 1 goes green.

Specification by Example

The objective of Specification by Example is to focus on development and delivery of prioritized, verifiable, business requirements.

It supports a very specific, concise vocabulary known as ubiquitous language that −

- Enables executable requirements.
- Is used by everyone in the team.
- Is created by a cross-functional team.
- Captures everyone's understanding.

Purpose of Specification by Example

The primary aim of Specification by Example is to build the right product.

It focuses on shared understanding, thus establishing a single source of truth.

It enables automation of acceptance criteria so that focus is on defect prevention rather than defect detection.

It also promotes test early to find the defects early.

Use of SbE

Specification by Example is used to illustrate the expected system behavior that describes business value.

The illustration is by means of concrete and real life examples.

These examples are used to create executable requirements that are −

- Testable without translation.
- Captured in live documentation.

Reasons why we use examples to describe particular specifications −

- They are easier to understand.
- They are harder to misinterpret.

Advantages of SbE

- Increased quality
- Reduced waste
- Reduced risk of production defects
- Focused effort
- Changes can be made more safely
- Improved business involvement

SbE- A Set of Process Pattern

Specification by Example is defined as a set of process patterns that facilitate change in software products to ensure that the right product is delivered efficiently.

The process patterns are −

- Collaborative specification
- Illustrating specifications using examples
- Refining the specification
- Automating examples
- Validating frequently
- Living documentation

Collaborative Specification

The objectives of collaborative specification are to –

- Get the various roles in a team to have a common understanding and a shared vocabulary.
- Get everyone involved in the project so that they can contribute their different perspectives about a feature.
- Ensure shared communication and ownership of the features.

- These objectives are met in a specification workshop also known as the Three Amigos meeting.
- The Three Amigos are BA, QA and the developer. Though there are other roles in the project, these three would be responsible and accountable from definition to the delivery of the features.

During the meeting –

- The Business Analyst (BA) presents the requirements and tests for a new feature.
- The three Amigos (BA, Developer, and QA) discuss the new feature and review the specifications.
- The QA and developer also identify the missing requirements.
- The three Amigos
  - Utilize a shared model using a ubiquitous language.
  - Use domain vocabulary (A glossary is maintained if required).
  - Look for differences and conflicts.
- Do not jump to implementation details at this point.
- Reach a consensus about whether a feature was specified sufficiently.
- A shared sense of requirements and test ownership facilitates quality specifications
- The requirements are presented as scenarios, which provide explicit, unambiguous requirements. A scenario is an example of the system's behavior from the users' perspective.

Illustrating Specification using Examples

- Scenarios are specified using the Given-When-Then structure to create a testable specification –

Given <some precondition>

And <additional preconditions> **Optional**

When <an action/trigger occurs>

Then <some post condition>

And <additional post conditions> **Optional**

- This specification is an example of a behavior of the system. It also represents an Acceptance criterion of the system.
- The team discusses the examples and the feedback is incorporated until there is agreement that the examples cover the feature's expected behavior. This ensures good test coverage.

Refining the Specification

- Be precise in writing the examples. If an example turns to be complex, split it into simpler examples.
- Focus on business perspective and avoid technical details.
- Consider both positive and negative conditions.
- Adhere to the domain specific vocabulary.
- Discuss the examples with the customer.
  - Choose conversations to accomplish this.
  - Consider only those examples that the customer is interested in. This enables production of the required code only and avoids covering every possible combination, that may not be required
- To ensure that the scenario passes, all the test cases for that scenario must pass. Hence, enhance the specifications to make them testable. The test cases can include various ranges and data values (boundary and corner cases) as well as different business rules resulting in changes in data.
- Specify additional business rules such as complex calculations, data manipulation / transformation, etc.
- Include non-functional scenarios (e.g. performance, load, usability, etc.) as Specification by Example

Automating Examples

- The automation layer needs to be kept very simple – just wiring of the specification to the system under test. You can use a tool for the same.

- Perform testing automation using Domain Specific Language (DSL) and show a clear connection between inputs and outputs. Focus on specification, and not script. Ensure that the tests are precise, easy to understand and testable.

Validating Frequently

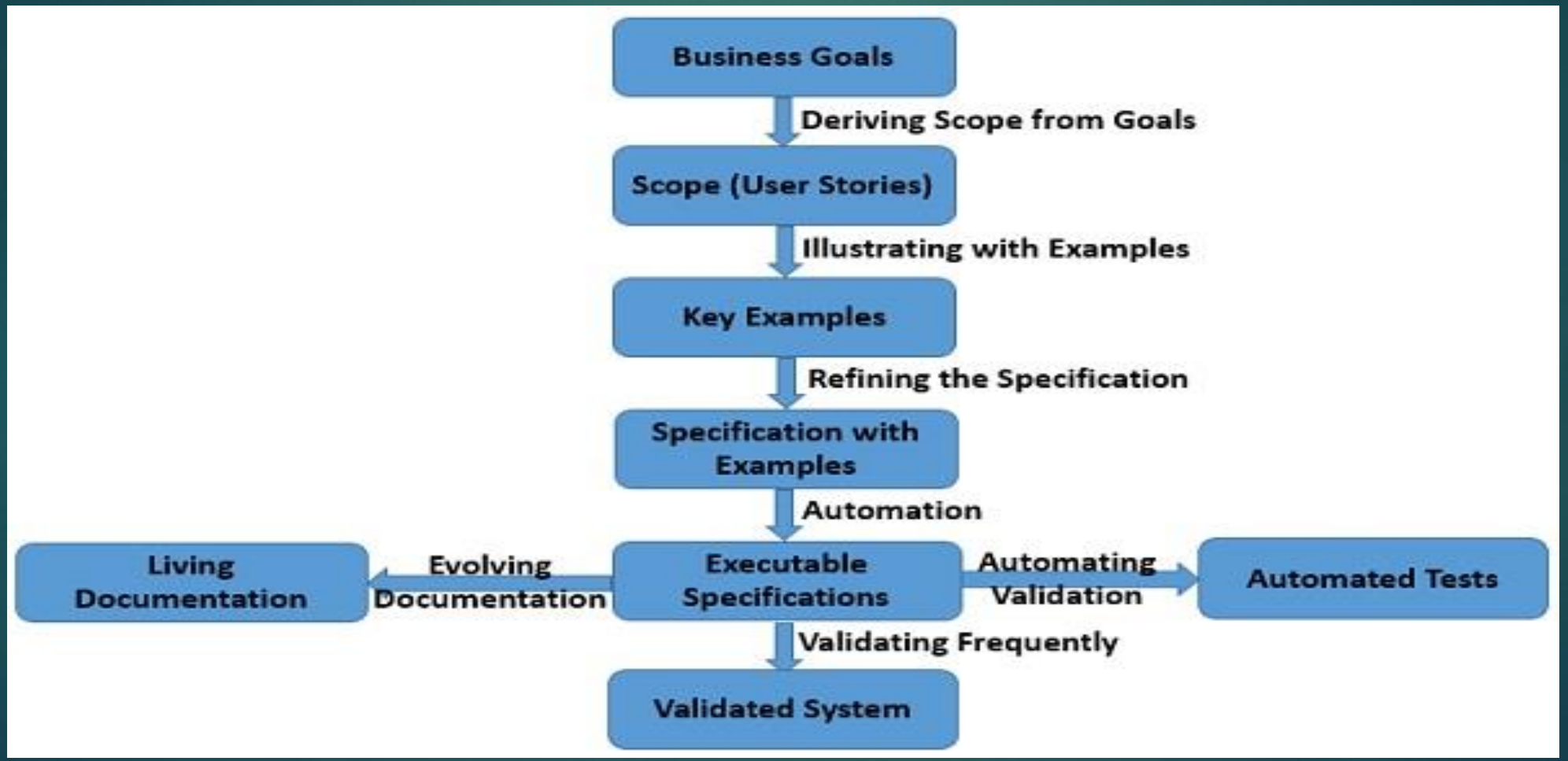
- Include example validation in your development pipeline with every change (addition/modification). There are many techniques and tools that can (and should) be adopted to help ensure the quality of a product. They revolve around three key principles- Test Early, Test Well and Test Often.

- Execute the tests frequently so that you can identify the weak links. The examples representing the behaviors help track the progress and a behavior is said to be complete only after the corresponding test passes.

Living Documentation

- Keep the specifications as simple and short as possible. Organize the specifications and evolve them as work progresses. Make the documentation accessible for all in the team.

SbE Process Steps



Tools

- Cucumber (Ruby framework)
- SpecFlow (.NET framework)
- Behave (Python framework)
- JBehave (Java framework)
- JBehave Web (Java framework with Selenium integration)
- Lettuce (Python framework)
- Concordion (Java framework)
- Behat (PHP framework)
- Kahlan (PHP framework)
- DaSpec (JavaScript framework)
- Jasmine (JavaScript framework)
- Cucumber-js (JavaScript framework)
- Squish GUI Tester (BDD GUI Testing Tool for JavaScript, Python, Perl, Ruby and Tcl)
- Spock (Groovy framework)
- Yadda (Gherkin language support for frameworks such as Jasmine (JavaScript framework))