

BDD ACROSS ENTERPRISE

Behavior Driven
Development



Each team saw value in the BDD process and is successfully using it today.

Summary

In April I helped a Fortune 50 company roll out Behavior Driven Development¹ across a multisite enterprise (2-3,000 developers). They'd been doing Scrum for three years and wanted to roll out an engineering practice that gives them a better system for tracing requirements from the source to the functionality via automated test and test results.

I worked with a team on each of their tracks, six Scrum teams. Each team saw value in the BDD process and is successfully using it today.

Background on BDD

Behavior Driven Development is a **requirements communication** strategy which as a side effect, prepares for system test automation similar to Acceptance Test Driven Development (ATDD). Requirements are expressed by the PO as sets of scenarios, commonly implemented using the keywords of Given, When, Then.

Given When Then:

(This app is an iTunes plugin that culls out invalid song entries in a playlist.)

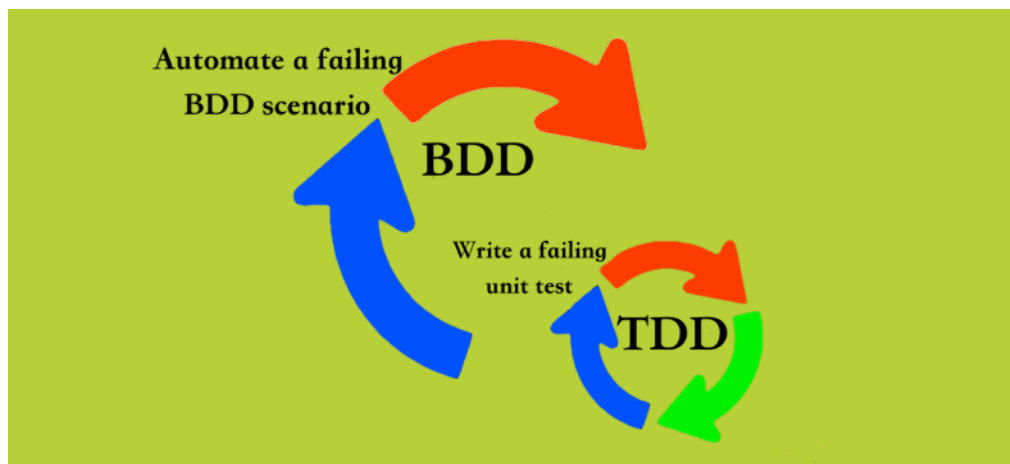
Given there exists bad entries in playlist
Never Played
When trying to play this playlist
Then remove invalid playlist entries

(Read more about this example at:

<http://confessionsofanagilecoach.blogspot.com/2013/09/well-written-behavior-driven.html>)

By supplying a Scrum team with a set of behaviors defined in the Given/When/Then format, the process:

- ➔ **Divides behaviors from implementation details**—The team's PO focuses on what behavior will serve the end user rather than get bogged down in implementation details such as what UI widget to use, system architecture, and class level design.
- ➔ **Adds structure**—Fuzzy user stories that the developers don't understand or the PO doesn't really understand means that during the sprint they are still trying to decide the behavior and our level of effort estimates will be of low quality.
- ➔ **Reduces over-engineering or wrong-engineering**—By creating an explicit and finite set of testing scenarios that describe the intended behavior, development knows what the business wants and is free to design/engineer the implementation that satisfies that behavior.



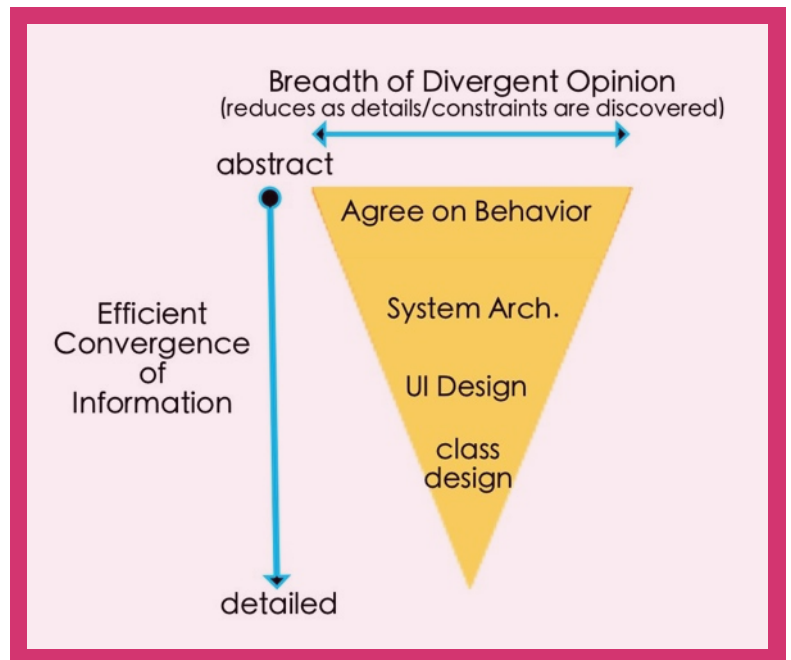
Improvements to Software Delivery

During the coaching engagement, teams, PO, and management noticed the following:

- ➔ **greater understanding of the work**—by having conversations about scenarios with the development team, more scenarios were generated to flesh out requirements which were missed

- ➔ **discussions started at a higher level**—conversations became about getting the behavior right before spending time on implementation details (UI implementation, system architecture, ...)

- ➔ **scenarios became a quality gate for User Stories**—when the PO worked on scenarios with her stakeholders, it became clear what behaviors they understood well enough to define and what other items weren't ready for Sprint Planning as they needed more followup with users or marketing



If the BDD scenarios were used to implement automated tests (rather than only for expressing requirements), there were further improvements to the development process:

- ➔ BDD's directly produced artifacts (scenarios) were also used to implement automated Acceptance Tests (i.e. System tests specifically used to decide when a given feature was finished moreover operate as automated regression tests.)

- ➔ because the scenarios are understandable by POs, release managers, end users, etc., and are automated, the collection of BDD scenarios became LIVE documentation about the behaviors of the system.

- ➔ at release time, it will be easier to communicate what features a product is shipping with as the scenarios are written in behavioral language that a release manager understands because it's devoid of technical implementation details

Program Operationalization

The client and I put together a fun and catchy BDD rewards program that was designed to at least get them to try it, and had further rewards for being fully cross functional, and finally, doing BDD as part of their Story Definition of Done. The deadline was set for 4 months after the first day of coaching.

I worked with an engineering team in each of their product lines for 8 days:

- ➔ a few hours of classroom training on what BDD is, how to write scenarios using Given/When/Then, and some technical slides on implementing the automation.
- ➔ a few hours to a day working with PO and QA professionals on adding BDD scenarios to their existing User Stories
- ➔ a two hour BDD coding dojo with the entire team
- ➔ daily: pair programmed with a team members on creating an automated BDD test for a feature (User Story) that was on their product backlog for that sprint. I paired with a given team member until I felt that they “got it” and then I’d switch to another team member. The stretch goal was to pair with everyone on the team.

Results

Five out of six of the teams I worked with reached Mini Me during their first eight-nine days of coaching. The remaining team reached Mini Me in 9 days. (Another team donated a day to them as they didn’t want to go beyond Mini Me for various reasons.) Three of the six teams were more than halfway to Mr. Bigglesworth during the first eight days.

Because I’m a consultant, the product I’m delivering is people and processes they can operate with when I leave. To see how I did, I checked in four months later, after the end of the rewards program. Three teams had reached

Level UP your BDD Mastery

1. An automated BDD Scenario tested and implemented within a sprint. (Team drinks on Co.)
2. Everyone on the team has implemented a BDD Scenario test. (Team cafeteria lunch on Co.)
3. September deadline: Within one sprint, the team delivers BDD tests & functionality for entire Sprint Backlog. (Chic* team lunch on Co.) *up to \$25 per person

Team*	Mastery	# of tests
team S	Mini Me	1061
team VC**	Mini Me	1
team I	Dr. Evil	60
team F	Dr. Evil	***
team T	Dr. Evil	***
team NH	Mini Me	4

* Listed by order of coaching. Team NH, for example, had only six weeks before the end of the program.
 ** This team’s management wanted only one test implemented to learn the value of BDD. Later, they merged with team NH and asked a NH team member to be a BDD specialist.
 *** Not known at press time

the Dr. Evil level to the surprise and delight of company management. The remaining teams were still at Mini Me, meaning they were unable to get every team member involved with BDD, although were adding BDD tests to their test suite (but not adding enough to keep up with the rate of producing functionality, which Dr. Evil requires).

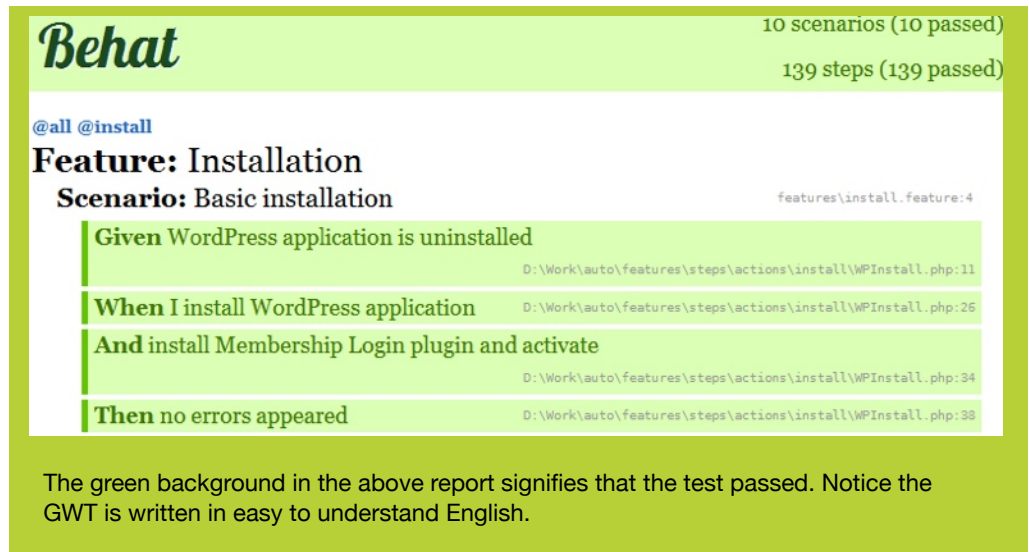
By adding a little more structure (Given/When/Then) and focusing on behavior, BDD takes the value of regression testing and testing “done” that Acceptance Test Driven Development gave us, and adds: reduction in the churn POs have with their teams due to coupling UI design with

new behavior, focused stakeholder discussions around behaviors, and a “live” document that describes the behaviors the application actually can do, (refreshed daily, hourly, or continuously). No document written on paper, email, or in a spreadsheet has the value of a BDD scenario automating a test.

About Lance Kind

He started his first software development project on a Vic 20 at the age of 10 and his first “consulting gig” was advising his high school math teacher how to get the school’s modems working.

Several years and a BS and MS degree later, he worked for Hewlett Packard, Microsoft, and then SolutionsIQ. Software engineering was beginning to feel dull until 1999 when he started doing eXtreme Programing under the tutelage of Kent Beck, and then later, started doing Scrum + XP. After delivering several successful projects, he started consulting and training across many sectors, from internet startup and social media to financial, energy, insurance, telecom, and medical devices.



The screenshot shows a Behat test report. At the top, it says "Behat" in a stylized font. To the right, it indicates "10 scenarios (10 passed)" and "139 steps (139 passed)". Below this, there are tags "@all @install". The main section is titled "Feature: Installation" and "Scenario: Basic installation". The scenario is broken down into four steps, each with a green background indicating it passed:

- Given** WordPress application is uninstalled (D:\Work\auto\features\steps\actions\install\WPInstall.php:11)
- When I install** WordPress application (D:\Work\auto\features\steps\actions\install\WPInstall.php:25)
- And install** Membership Login plugin and activate (D:\Work\auto\features\steps\actions\install\WPInstall.php:34)
- Then** no errors appeared (D:\Work\auto\features\steps\actions\install\WPInstall.php:38)

At the bottom of the screenshot, a note states: "The green background in the above report signifies that the test passed. Notice the GWT is written in easy to understand English."



I had my doubts
Any new process, especially one that lauds better requirements sounds like reverting to the “old days of BIG upfront requirements.” But it was similar enough to ATDD to be worth a try. I was happy to see I was getting a greater understanding of how my software should **behave**.