**Topic:behat**

**They are many flavors in bdd I will mention some those:**

Ruby has cucumber

Php has behat

Java has jebhave

Phython has lettuce

These tools are similar in business facing but it differs in technology facing.

Your project, features, scenarios and steps will come under business facing.

The business-facing parts of a behat test suite, stored in feature files, must be written according to syntax rules known as Gherkin so that behat can read them.

Step definitions, support code, automation library will come under technology facing.

-behat is a open source and command line tool.

- When you run it, it reads in your specifications from plain-language text files called *features*, examines them for *scenarios* to test, and runs the scenarios against your system.

-Each scenario is a list of *steps* for behat to work through. So that behat can understand these feature files, they must follow some basic syntax rules.

-The name for this set of rules is *Gherkin*.

What makes Behat stand out from the crowd of other testing tools is that it has been designed specifically to ensure the acceptance tests can easily be read—and written—by anyone on the team.

This reveals the true value of acceptance tests: as a communication and collaboration tool. The easy readability of behat tests draws business stakeholders into the process,

helping you really explore and understand their requirements.

**Why we call features has automated acceptance tests:**

**-Instead of a business stakeholder passing requirements to the development**

**team without much opportunity for feedback.**

**-the developer and stakeholder collaborate to write automated tests that express the outcome that the stakeholder wants.**

**-We call them acceptance tests because they express what the software needs to do in order for the stakeholder to find it *acceptable*.**

**-The test fails at the time of writing, because no code has been written yet, but it captures what the stakeholder cares about and gives everyone a clear signal as to what it will take to be *done*.**

Topic: gherkin language

-Many software projects suffer from low-quality communication between the domain experts and programmers on the team.

- When the team uses this gherkin language consistently in their conversations, documentation, and code, the chances of misunderstandings are greatly reduced.

-Lines starting with the keyword Feature.

-it just gives you a convenient place to put some summary documentation about the group of tests

Scenario:

-A feature usually contains a list of scenarios.

In valid Gherkin, a Feature must be followed by one of the following:

• Scenario

• Background

• Scenario Outline

-Every scenario consists of a list of [steps](http://docs.behat.org/en/v2.5/guides/1.gherkin.html#steps), which must start with one of the keywords Given, When, Then, But or And.

-In addition to basic [scenarios](http://docs.behat.org/en/v2.5/guides/1.gherkin.html#scenarios), feature may contain [scenario outlines](http://docs.behat.org/en/v2.5/guides/1.gherkin.html#scenario-outlines) and [backgrounds](http://docs.behat.org/en/v2.5/guides/1.gherkin.html#backgrounds).

-Scenario is one of the core Gherkin structures.

-Every scenario starts with the Scenario: keyword , followed by an optional scenario title.

When behat runs a scenario, if the system behaves as described in the scenario, then the scenario will pass; if not, it will fail.

Background:

A *background* section in a feature file allows you to specify a set of steps that are common to every scenario in the file.

Scenario outline:

We can use a scenario outline to specify the steps once

and then play multiple sets of values through them.

We indicate *placeholders* within the scenario outline using angle brackets (<..>) where we want real values to be substituted.

The scenario outline itself

is useless without a table of Examples, which lists rows of values to be substituted for each placeholder.

You can have any number of Scenario Outline elements in a feature and any number of Examples tables under each scenario outline.

Behind the scenes, behat converts each row in the Examples table into a scenario before executing it. You can prove this to yourself by using the –expand

grouping:

You can use [tags](http://docs.behat.org/en/v2.5/guides/1.gherkin.html#tags) to group features and scenarios together, independent of your file and directory structure.

Step definitions:

When you’re executing the feature, the trailing portion of each step is matched to a regular expression, which executes a PHP callback function. That we call as step definitions.

Which automation library we used in behat?

Mink is an open source browser controller/emulator for webapplications written php 5.3

So in order to test our web application, we need a way to stimulate this interaction between the browser and the web application in our tests. We need mink.

**Controlling the browser:**

In mink the entry point to the browser is called session.

Interacting with page:

Now that your session is started, you want to open the a page with visit function.

Using the

//get the current page url

Echo $session->getcurrenturl();

//use history controls

$session->reload();

$session->back();

$session->forward();

Resetting the session:

//soft –reset

$session->reset();

//hardreset

$session->stop();