Behavior-driven development (BDD) & Behavior-driven testing (BDT)

- Pavel Tišnovský
    - `ptisnovs@redhat.com`

- Slides:
    - `https://github.com/tisnik/presentations/tree/maste`
- Examples:
    - `https://github.com/tisnik/python-behave-demos`

TOC

- Why BDD?
    - BDD: behavior-driven development
    - Simple DSL
- Gherkin language
    - Feature/Scenario/Scenario Outline
    - Given/when/then
    - Variable parameters in tests
    - Tables as data source
    - Tables for multiple test runs with variable parameters
- Practical part
    - Python & Behave library
    - Project structure

    - Python modules testing
    - Native functions/libraries testing
    - REST API testing

Why BDD?

- Today IT systems are usually very complex
    - quite hard to get the overall picture
    - overall function vs implementation details
- Basic/typical problems
    - bug fixes are very expensive in the later stages of the development
    - customers don’t like features^W bugs
    - it is vital to verify and validate application behaviour during development
- Other problems
    - large development teams
    - more programming languages (back end, front end, microservices…)
    - customer might like to be part of development process
    - not well-defined roles in some cases (devel? QA? devops?)

BDD focuses

- Where to start in the process
- What to test and what not to test

- How much to test in one go
- What to call the tests
- How to understand why a test fails

BDD: behavior-driven development

- On the boundary between
    - integration tests
    - API tests
    - acceptance tests
- Should be part of verification and validation processes!
- Describe the behavior of the system, usually from customer perspective
    - from the BDD point of view, the system is handled as a black box
- Can be used for front end and for back end as well
- “Weird test scenario” → probably the system has improper behavior
    - (tram ticket machines in Brno)
    - button instead of switch
    - …

Who should be author of BDD tests?

- There does not exist a role like “BDD test writer”
    - it is different from unit tests, for example
- BDD is based on cooperation

    - customer
    - architect
    - devel
    - QA
- Sometimes the term “multiple-stakeholder” is used

Gherkin language

- Based on natural language + a few keywords
    - usually English is used
    - translated into other languages as well
- It is quite similar to Python, AsciiDoc etc.
    - indentation
    - based on keywords, not on special characters
    - tables “drawn” in ASCII
- It is not tightly bound with any real programming language
- → it can be used by non-developers
    - (but don’t tell user that he/she is writing source code… :-)

Given-When-Then

- Semi-structured way to write down test cases
- Three clauses
    - Given
    - When
    - Then

- The same clause on more consecutive lines?
    - And

Gherkin language - an example

```
Given the customer has logged into their current account
  And the balance is shown to be 100 euros
 When the customer transfers 75 euros to their savings acc
ount
 Then the new current account balance should be 25 euros
```

Test scenario parts:

- Keywords/clauses
    - Given, And, When, Then
- The rest is written in “plain English”
- Contains variable parts as well
    - 100, 75, 25

Tables

- Tables has two purposes in Gherkin language
    - specify list of values used later in tests
    - specify multiple tests with the same sentences, but with different parameters/variables.

Tables

```
Feature: Sum function test 1

  Scenario: Check the function sum()
    Given a list of integers
      |value |
      | 1    |
      | 10   |
      | 100  |
      | 1000 |
    When I summarize all those integers
    Then I should get 1111 as a result
```

Tables (second example)

```
Feature: Interface to bank backend

  Scenario: Check the exchange rate calculation
    Given the following exchange rate table
      | currency |  rate  |
      | CZK      |  1.000 |
      | CAD      | 16.172 |
      | HRK      |  3.407 |
      | USD      | 20.655 |
    When I sell 10 CAD
```

```
    Then I should receive 161.72 CZK
```

Tables for specifying multiple test runs

```
  Scenario Outline: Check the user search feature, perform
 the search for more users
    Given GitHub is accessible
    When I search for user with nick <nick>
    Then I should receive 200 status code
     And I should receive proper JSON response
     And I should find the user with full name <fullname>
     And I should find that the user works for company <co
mpany>

     Examples: users
     |nick|fullname|company|
     |torvalds|Linus Torvalds|Linux Foundation|
     |brammool|Bram Moolenaar|Zimbu Labs|
     |tisnik|Pavel Tišnovský|Red Hat, Inc.|
```

Combination of two tables with different purposes

```
  Scenario Outline: Check the exchange rate calculation
    Given the following exchange rate table
      | currency |  rate  |
```

```
      | CZK      |  1.000 |
      | CAD      | 16.172 |
      | HRK      |  3.407 |
      | USD      | 20.655 |
    When I sell <sold> <currency>
    Then I should receive <amount> CZK

    Examples: sold
        | sold | currency | amount |
        | 1    |   CZK    |    1.000 |
        | 10   |   CZK    |   10.000 |
        | 1    |   CAD    |   16.172 |
        | 100  |   CAD    | 1617.200 |
        | 2    |   HRK    |    6.814 |
```

Practical part

- Behave library
- Structure of project with BDD tests
    - tested module
    - the scenario
    - scenario implementation
    - test environment specification

Repository with examples

- https://github.com/tisnik/python-behave-demos
    - (well I still trust GitHub a bit :-)

```
git clone https://github.com/tisnik/python-behave-demos
```

Behave library

- Behave library
    - for Python 2.x and Python 3.x as well
    - most of Gherkin language is implemented
    - binding: test description <-> test implementation
    - decorators
    - test parameters are “deduced” from decorators

Context

- An object
    - automatically created/recreated
    - manipulated in test steps and in tests environment

```
@capture
def before_all(context):
    """Perform setup before the first event."""

@capture
def before_scenario(context, scenario):

```

```
    """Perform setup before each scenario is run."""

@capture
def after_scenario(context, scenario):
    """Perform cleanup after each scenario is run."""

@capture
def after_all(context):
    """Perform cleanup after the last event."""
```

Structure of project with BDD tests

```
├── feature_list.txt
├── features
│   ├── adder.feature
│   └── steps
│       └── common.py
├── requirements.in
├── requirements.txt
├── run_tests.sh
└── src
    └── adder.py
```

Structure of project with BDD tests

```
src/adder.py                        tested module
```

```
requirements.in/requirements.txt used by PIP
feature_list.txt                  list of test scenarios
features/                         test scenario(s) + implem
entation of test steps
run_tests.sh                      helper script to run Beha
ve
```

REST API testing

- *requests* library
- *Beautiful soup* for working with HTMLs, XMLs etc.

Test scenario

```
  @smoketest
  Scenario: Check the GitHub API entry point
    Given GitHub is accessible
    When I access the API endpoint /
    Then I should receive 200 status code

  Scenario: Check the user search feature
    Given GitHub is accessible
    When I search for user with nick torvalds
    Then I should receive 200 status code
     And I should receive proper JSON response
     And I should find the user with full name Linus Torva
lds
```

```
    And I should find that the user works for company Lin
ux Foundation
```

Test environment

```
import json
import os.path

from behave.log_capture import capture
import requests


def _is_accessible(context, accepted_codes=None):
    accepted_codes = accepted_codes or {200, 401}
    url = context.api_url
    try:
        res = requests.get(url)
        return res.status_code in accepted_codes
    except requests.exceptions.ConnectionError as e:
        print("Connection error: {e}".format(e=e))
    return False


def before_all(context):
    """Perform setup before the first event."""
    context.is_accessible = _is_accessible
    context.api_url = "https://api.github.com"
```

Test environment

```
@capture
def before_scenario(context, scenario):
    """Perform setup before each scenario is run."""
    pass


@capture
def after_scenario(context, scenario):
    """Perform cleanup after each scenario is run."""
    pass


@capture
def after_all(context):
    """Perform cleanup after the last event."""
    pass
```

Implementation of test steps

```
import json

from behave import given, then, when
from urllib.parse import urljoin
import requests
```

```
@given('GitHub is accessible')
def initial_state(context):
    assert context.is_accessible(context)


@given('System is running')
def running_system(context):
    """Ensure that the system is accessible."""
    assert is_accessible(context)


@when('I access the API endpoint {url}')
def access_endpoint(context, url):
    context.response = requests.get(context.api_url + url)


@when('I search for user with nick {nick}')
def search_for_user(context, nick):
    url = urljoin(urljoin(context.api_url, "users/"), nick
)
    context.response = requests.get(url)
```

Implementation of test steps (cont.)

```
@then('I should receive {status:d} status code')
def check_status_code(context, status):
```

```
    """Check the HTTP status code returned by the REST API
."""
    assert context.response.status_code == status


@then('I should receive proper JSON response')
def check_json_response(context):
    content_type = context.response.headers.get('content-t
ype')
    assert content_type.startswith('application/json')
    context.data = context.response.json()


@then('I should find the user with full name {fullname}')
def check_user_full_name(context, fullname):
    assert context.data is not None
    assert 'name' in context.data
    value = context.data.get('name')
    assert value == fullname, "{e} != {v}".format(e=fullna
me, v=value)


@then('I should find that the user works for company {comp
any}')
def check_company(context, company):
    assert context.data is not None
    assert 'company' in context.data
    value = context.data.get('company')
```

```
    assert value == company, "{e} != {v}".format(e=company
, v=value)
```

Useful links

- Why is software testing necessary? http://tryqa.com/why-is-testing-necessary/
- Why do we Test? What is the Purpose of Software Testing?: https://www.testingexcellence.com/why-do-we-test-what-is-the-purpose-of-software-testing/
- behave 1.2.6: https://pypi.org/project/behave/
- Welcome to behave!: https://behave.readthedocs.io/en/latest/
- Getting Started with Behavior Testing in Python with Behave https://semaphoreci.com/community/tutorials/getting-started-with-behavior-testing-in-python-with-behave
- What is Gherkin – BDD Language? http://toolsqa.com/cucumber/gherkin/
- Python Quick Reference: http://rgruet.free.fr/#QuickRef
- Python docs: http://www.python.org/doc/
- PEP 8: http://www.python.org/dev/peps/pep-0008/
- pep8.py: http://pypi.python.org/pypi/pep8/
- pylint: http://www.logilab.org/project/pylint
- Epydoc: http://epydoc.sourceforge.net/

Useful links (cont.)

- Sphinx: http://sphinx-doc.org/
- Python in Python: http://pypy.org/
- The key differences between Python 2.7.x and Python 3.x with examples: http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.htr
- Language differences and workarounds: http://python3porting.com/differences.html
- Everything you did not want to know about Unicode in Python 3: http://lucumr.pocoo.org/2014/5/12/everything-about-unicode/
- Unicode (Wikipedia): https://en.wikipedia.org/wiki/Unicode
- Dive Into Python: http://www.diveintopython.net/
- Dive into Python 3: http://www.diveintopython3.net/
- Given-When-Then: https://en.wikipedia.org/wiki/Given-When-Then

---

Thank you so much!