ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTY OF ELECTRICAL AND COMPUTER ENGINEERING
SOFTWARE ENGINEERING II
ACCEPTANCE TEST WORKSHOP - II TERM 2023

Objectives:

- Validate the correct operation of a system through acceptance tests.

Requirements

- Python
- Behave for Python

Introduction

Behavior Driven Development (BDD) is a way for software teams to work that closes the gap between businesspeople and technical people by:

- Encouraging collaboration across roles to build shared understanding of the problem to be solved.
- Working in rapid, small iterations to increase feedback and the flow of value.
- Producing system documentation that is automatically checked against the system's behavior [1].

ACCEPTANCE TESTING is a level of software testing where a system is tested for acceptability. It is formal testing with respect to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system, and if it is acceptable for delivery [1].

The framework to write the acceptance tests is Cucumber, what use "Gherkin" language. The 10 key words of Gherkin are: [2]:

- Feature
- Scenario
- Given
- When
- Then
- And
- But
- Scenario Outline
- Examples
- Background

Let us review the 10 key words.

- **Given:** This puts the system in a known state. It's a set of key pre-conditions for a scenario (e.g. user has logged in, user has money in their account etc.)
- **When:** This is the key action, a user will take. It's the action that leads to an outcome
- **Then:** This is the observable outcome. It's what happens after the user makes that action
- **Scenario:** This is used to describe the scenario & give it a title. The reason we do this is because a feature or user story will likely have multiple scenarios.
- **And:** This is used when a scenario is more complicated. It can be used in association with Given, When, or Then.

- **But:** Can be used in association with Then. It's used to say something shouldn't happen as an outcome.
- **Feature:** is used to give a title for the feature/piece of functionality. A feature contains lots of scenarios. For example, "Sign in" might be a feature … or "push alerts" …. it's the title of a piece of functionality.
- **Scenario Outline / Examples:** These are used together. They are used to combine a set of similar scenarios.
- **Background:** This sets the context for all scenarios below it. If you find that scenarios have common Given/Ands, Background can be used to eliminate the repetition. Background is run before each of your scenarios.

For More information about the key words check [3].

Workshop activities

This workshop is divided into three main segments: 1) Project Setup. 2) Test Development. 3) Iterative Process.

To develop these activities, we will use a simple To-Do List Manager project.

The To-Do List Manager is a command-line application that allows users to manage their tasks by adding, listing, and marking them as complete. Requirements and more information will be found at the end of this document.

Part 1: Prepare the project

1. Create a repository for the project.
2. Create a new directory for the project and navigate into it.
3. Install the Behave library with this command in the command prompt or terminal.

```
pip install behave (o pip3)
```

4. Fulfill the requirements given in the last part of this document.
5. In the requirements section, there are **4 main suggested features**. Add **two more** and specify them in the document.
6. Verify that there are no errors in the code. Run your main file.

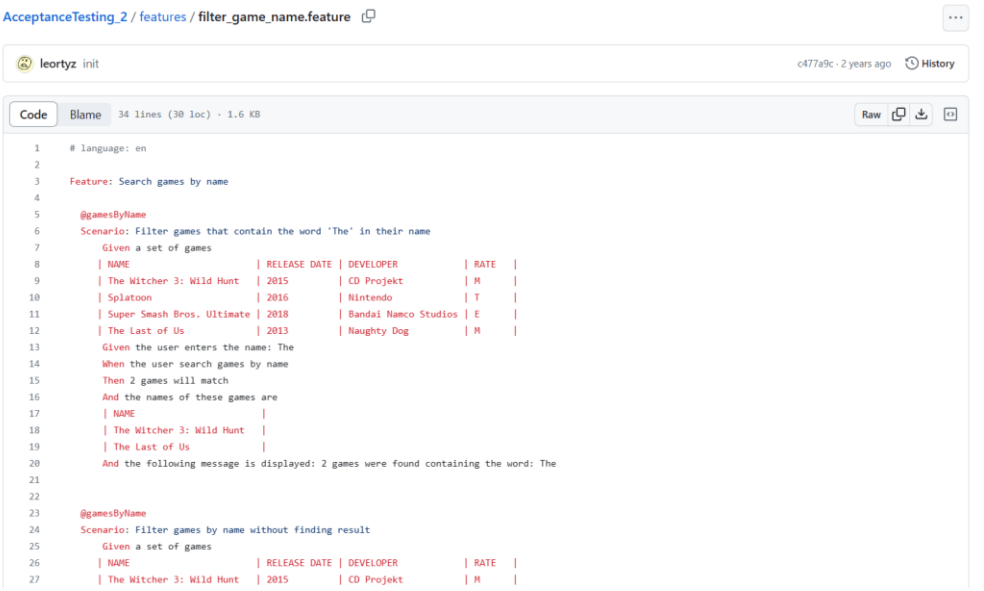
```
python main.py
```

Part 2: Test Development.

1. After creating the project, you will start with the tests. Review the requirements in the last part of this document. There are 4 main **Suggested Behave Scenarios.** You will need to think **in 2 more**, which may involve either failed interactions or the newly added requirements from earlier sections of this document.
2. If not done in previous steps, create a features folder and inside it a Feature File [4] and a steps folder. The steps folder will contain your acceptance tests. You will end with a structure like this:

```
features/
features/everything.feature
features/steps/
features/steps/steps.py
```

3. You will need to add those scenarios of step 1 to the .feature file inside the features folder. At the end, the structure of the file should look like this:



4. Then, you start creating the acceptance tests, called steps in Behave. Steps used in the scenarios are implemented in Python files in the “steps” directory.

   For example:
   Given a Scenario (that you defined in the .feature file):

```
Scenario: Adding a task
   Given the To-Do list is empty
    When the user adds a task "Buy groceries"
    Then the to-do list should contain "Buy groceries"
```

   Step code implementing the three steps here might look like:

```python
# Step 1: Given the to-do list is empty
@given('the to-do list is empty')
def step_impl(context):
    # Set the to-do list as an empty list
    global to_do_list
    to_do_list = []

# Step 2: When the user adds a task "Buy groceries"
@when('the user adds a task "{task}"')
def step_impl(context, task):
    # Add the task to the to-do list
    global to_do_list
    to_do_list.append(task)

# Step 3: Then the to-do list should contain "Buy groceries"
@then('the to-do list should contain "{task}"')
def step_impl(context, task):
    # Check if the task is in the to-do list
    assert task in to_do_list, f'Task "{task}" not found in the to-do
list'
```

If you need, you can read the documentation for more detailed functions. [5]

Note: You can call these whatever you like as long as they use the python *.py file extension.

5. Fulfil the project with all the necessary steps.
6. Run the acceptance test and analyze the result.

```
behave --format=json --outfile=result.json
```

Change the name for your group name or individual.

Part 3. Iterative Process

For this activity, you must review your previous results and check errors that may have been found. Correct those errors and run the tests again.

If there are none, specify it in the document.

Deliverables

1) The repository link to the To Do List Project

2) The tests results to json using the given command. Name the report with the name of each student or group.

3) Practice report with at least: cover, introduction, development, conclusions and recommendations, and references.

Rubric

Description	Weight
Part 1 - Tool configuration, evidence	10
Part 1 – Code creation, evidence, quality	25
Part 2 – Feature file creation	15
Part 2 – Acceptance tests creation	30
Part 2 – Test run evidence	10
Part 3 – Tests/code correction evidence	10
Total	**100**
Report – each missing section	-3
Individual penalty per missing evidence in taking part of the project.	-30
Penalty per missing repository link	-100
Penalty per each requirement not fulfilled	-5
Penalty for not uploading required deliverables as specified	-30

Late Submission Policy

Delay (§)	Penalty (Ω)
1 hour or less	loss of 10%
1 to 6 hours	loss of 20%
6 to 24 hours	loss of 30%
Over 24 hours:	loss of 100%

References

[1] Software Testing Fundamentals, "Acceptance Testing," [Online]. Available: http://softwaretestingfundamentals.com/acceptance-testing/.

[2] R. Hewitt, "Gherkin for Business Analysts," Modern analyst, 2017. [Online]. Available: https://www.modernanalyst.com/Resources/Articles/tabid/115/ID/3810/Gherkin-for-Business-Analysts.aspx. [Accessed 14 Marzo 2020].

[3] "Gherkin Reference," [Online]. Available: https://cucumber.io/docs/gherkin/reference/. [Accessed Marzo 2020].

[4] Behave, "Read the Docs," [Online]. Available: https://behave.readthedocs.io/en/latest/tutorial/#feature-files.

[5] B. A. Reference. [Online]. Available: https://behave.readthedocs.io/en/latest/api/.

[6] S. Vergara, "¿Qué es BDD (Behavior Driven Development)?," ITDO, 18 Julio 2019. [Online]. Available: https://www.itdo.com/blog/que-es-bdd-behavior-driven-development/. [Accessed 14 Marzo 2020].

To Do List Manager

Description: The To-Do List Manager is a command-line application that allows users to manage their tasks by adding, listing, and marking them as complete.

A task can have the attributes you think will fit the most, like name, status or description.

Functionality:

- Add a new task to the to-do list.
- List all the tasks in the to-do list.
- Mark a task as completed.
- Clear the entire to-do list.

Requirements:

- Python 3.x installed on the system.
- Behave and pytest libraries installed in the virtual environment.

Example Project Structure:

```
- todo_list.py  #Main application logic. You may use more than one files.
- features/     # Behave feature files and step definitions
  - todo_list.feature
  - steps/
      - todo_list_steps.py
```

Note: These are suggested names, you may use the ones you think fit the most.

Suggested Features:

- Add a task to the to-do list.
- List all tasks in the to-do list.
- Mark a task as completed.
- Clear the entire to-do list.
  Note: Add two more features, so it becomes 6 in total.

Suggested Behave Scenarios:

Scenario: Add a task to the to-do list

- Given the to-do list is empty
- When the user adds a task "Buy groceries"
- Then the to-do list should contain "Buy groceries"

Scenario: List all tasks in the to-do list

- Given the to-do list contains tasks:
  | Task |
  | Buy groceries |
  | Pay bills |
- When the user lists all tasks
- Then the output should contain:

```
Tasks:
 -  Buy groceries
 - Pay bills
```

Scenario: Mark a task as completed

- Given the to-do list contains tasks:
  | Task | Status |
  | Buy groceries | Pending |
- When the user marks task "Buy groceries" as completed
- Then the to-do list should show task "Buy groceries" as completed

Scenario: Clear the entire to-do list

- Given the to-do list contains tasks:
  | Task |
  | Buy groceries |
  | Pay bills |
- When the user clears the to-do list
- Then the to-do list should be empty

Note: Add two more scenarios, so it becomes 6 in total.