Software Build Automation Tools

Laboratory Exercises 5

Subject
Automating software testing in Python using pytest, with integration into Git, GitHub, and CI/CD workflows

Author:
PhD Hubert Zarzycki

Introduction

Automated testing is an essential component of modern software development. It ensures code reliability, detects bugs early, and speeds up the development cycle. Among the many testing frameworks available for Python, *pytest* stands out for its simplicity, flexibility, and ease of integration with Continuous Integration/Continuous Deployment (CI/CD) tools.

This laboratory session focuses on setting up a Python project in *PyCharm*, integrating it with *GitHub*, and writing automated tests using pytest. The exercises also include running tests in a CI/CD pipeline, simulating real-world software development workflows.

This laboratory session provided hands-on experience in *test automation using pytest*. It shows how to:

- Set up a test automation project in *PyCharm*.
- Write and execute *automated tests*.
- Use *fixtures* and *parameterized tests*.
- Integrate testing into a *CI/CD pipeline* using *GitHub Actions*.

Mastering these concepts will help you in software development by ensuring code quality and reliability in professional environments.

1. Setting Up the Development Environment

1.1. Installing PyCharm

PyCharm is a popular IDE for Python development with built-in tools for version control and testing.

- Download PyCharm (Community Edition) from **https://www.jetbrains.com/pycharm/**.
- Install PyCharm by following the on-screen instructions.
- Launch PyCharm and create a new project.

1.2. Creating a Python Project

- Open PyCharm and select *New Project*.
- Choose a project location (e.g., `C:\Users\Student\pytest_project`).
- Ensure "Create a new virtual environment" is selected.
- Click *Create*.

1.3. Setting Up Git in PyCharm

To track changes and collaborate, the project should be connected to a *Git repository*.

- Open PyCharm and navigate to *VCS* > *Enable Version Control Integration*.
- Select *Git* and click *OK*.
- Open the terminal in PyCharm and initialize a Git repository:

```
git init
```

- Create a `.gitignore` file to exclude unnecessary files:

```
echo "venv/" > .gitignore
```

- Add and commit the project:

```
git add .
git commit -m "Initial commit"
```

1.4. Connecting to GitHub

- Create a repository on *GitHub* at https://github.com.
- Copy the repository URL.
- In PyCharm, open the terminal and push the project to GitHub:

```
git remote add origin <repository_url>
git branch -M main
git push -u origin main
```

2. Writing and Running Tests with pytest

2.1. Installing pytest

To install pytest, open the terminal in PyCharm and run:

```
pip install pytest
```

2.2. Writing a Simple Test

Create a new directory named `tests/` and inside it, create a file called `test_sample.py` with the following content:

```
def add(x, y):
    return x + y
```

```
def test_add():
    assert add(2, 3) == 5
    assert add(-1, 1) == 0
```

2.3. Running the Tests

Run the test suite using:

```
pytest
```

Pytest will discover test files that start with `test_` and execute all functions that start with `test`.

3. Advanced Testing Features

3.1. Using Fixtures

Fixtures in pytest help set up test data before running tests.

Modify `test_sample.py` to include a fixture:

```
import pytest

@pytest.fixture
def sample_data():
    return {"name": "Alice", "age": 30}

def test_sample_data(sample_data):
    assert sample_data["name"] == "Alice"
    assert sample_data["age"] == 30
```

Run the tests again:

```
pytest
```

3.2. Parameterized Tests

To test multiple inputs efficiently, use parameterization:

```
@pytest.mark.parametrize("a, b, expected", [(2, 3, 5), (-1, 1, 0), (0, 0, 0)])
def test_add_param(a, b, expected):
    assert add(a, b) == expected
```

4. Integrating pytest with CI/CD (GitHub Actions)

4.1. Creating a GitHub Actions Workflow

Automating tests on every push ensures code quality.

- Create a `.github/workflows/pytest.yml` file in the project root.
- Add the following configuration:

```
name: Python Test Automation

on:
  push:
    branches:
      - main
  pull_request:

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: "3.10"

      - name: Install dependencies
        run: pip install pytest

      - name: Run tests
        run: pytest
```

4.2. Committing and Pushing to GitHub

After adding the workflow file, commit and push:

```
git add .
git commit -m "Added GitHub Actions workflow for pytest"
git push origin main
```

GitHub will automatically run the tests upon each push.

Laboratory Tasks

The aim is to practice version control in PyCharm while exploring CI/CD integration.

1. For the tasks create an archived project file as well as a report including:
   - Authors' names
   - Repository link
   - Steps performed during the lab and observations.
   - Screenshots and code snippets of key operations (Git setup, test execution, GitHub Actions)
   - Description of any challenges faced and solutions applied.
   - Final state of the repository.
   - A brief analysis of the test results.

The report should be submitted via the e-learning platform as a DOC or PDF document and archived project files.

2. Setting Up the Project
    - Create a new Python project in PyCharm
    - Initialize a Git repository and connect it to GitHub.
    - Set up a virtual environment and install pytest.
3. Writing and Running Tests
    - Create a simple Python function.
    - Write a test case using pytest.
    - Run the test suite and verify the results.
4. Implementing Advanced Testing Features
    - Modify the test suite to use pytest fixtures.
    - Implement parameterized tests for multiple inputs.
    - Rerun the tests and analyze the results.
5. Automating Tests with CI/CD
    - Create a GitHub Actions workflow file.
    - Implement a CI pipeline that installs dependencies and runs tests.
    - Push the changes to GitHub and observe the automated test execution.
    - Fix any failing tests and verify successful execution.

6. Please submit to the e-learning platform an archived project file as well as a report summarizing the work, including challenges faced and solutions applied. Include a link to the repository.

Literature

Loeliger, J., & McCullough, M. (2012). *Version Control with Git: Powerful tools and techniques for collaborative software development.* O'Reilly Media.
Chacon, S., & Straub, B. (2014). *Pro Git.* Apress.
Rubalcaba, C. (2021). Git for Programmers*: Master Git for Effective Development and Deployment.* Apress.
Okken, B. (2017). Python Testing with pytest. Pragmatic Bookshelf.
Hambleton, A. (2020). Test-Driven Development with Python. O'Reilly Media.
Beck, K. (2003). Test-Driven Development: By Example. Addison-Wesley.GIT Documentation - *https://git-scm.com/doc*
GitHub Learning Lab - *https://lab.github.com/*
Atlassian GIT Tutorials - *https://www.atlassian.com/git/tutorials*
GitHub Actions Documentation - *https://docs.github.com/en/actions*
PyCharm Guide: *https://www.jetbrains.com/pycharm/guide/*
pytest Documentation: https://docs.pytest.org/