“The fuck” application

V. P. van Wijhe(s2426145) H. Le(s2391740) W.H. Nijhuis(s2099624)

June 26, 2021

1 Introduction

This report describes and evaluates the pipeline development that is built to improve an application using the Agile methodology. We chosen “the fuck” code base, an open source project to practice our pipeline construction. To give a brief description, “the fuck” helps to correct mistake of the previous commands with the built-in rules. We inspected that the chosen project already has an adequate pipeline and a well organized structure, so we are aiming for improvements by introducing new features specifically new rules, increasing test coverage and evolving the CI/CD workflow.

This document was written in two sections, the first section covers code base information, elaborate pipeline, motivation and evaluation on chosen tools. The second section will answer the question relates to the reflection of agile practice and theory, with respect to the project and with respect to an industrial software development project that has been described in the literature.

2 Project

2.1 Codebase

“the fuck”[1] - an open source application written in python. The code base was contributed voluntarily by the developers community, there are 153 contributors at the moment. Vladimir Iakovlev[2] the starter and owner of the app, is responsible for approving pull requests and integration. The application is used by people who often work in the command line, and are willing to use a handy tool for correcting their errors. Below is the statistic of the source base:

- Latest release 3.31
- Test Coverage: 94%
- Number of matched rules: 161
- Line of functional code: 4352
- Line of test code: 5635

2.2 Infrastructure

Github is used to store and manage revision of the project, we chose Github because it is well known and also has features for project management and continuous integration and deployment. These features are free so it fits perfectly for a non-profit project. It is also preferable to be centralized using an all in one provider. In a nut shell, we use Github Actions for CI/CD, and Github project management board as a management platform.
An alternative solution is TravisCI for CI and Pivitol tracker for management. Both options are good candidates for the automation of the workflow, however requiring more efforts to hook up and integrate with the Github repository.
The fuck already had a CI workflow implemented with Github actions consisting of a code style check using flake8, unit tests and functional tests. Because flake8 was already used in the project, it could not be used to generate more code style errors. Pylint was therefore chosen to do a code style check as well. It is made by the same people who created flake8

and also checks for PEP-8[3] like code, but it is more difficult to please. Pylint is also implemented into the CI workflow so with every push/pull request it is checked. The CI workflow is also further extended with Bandit which is a program that scans the code base for vulnerabilities. The final step of the is to publish the current test statistics to an coveralls.io server. So the CI workflow includes:

1. Setting up Python.
2. Caching dependencies to speed up future workflow executions.
3. Install the dependencies.
4. Flake8 code style check.
5. Pylint code style check.
6. Bandit Security issues check.
7. Unit tests
8. Functional tests
9. Send the test data to coveralls.io

While the continuous integration was already present, the continuous deployment still had to be developed. When the code-base is drafted for release and published in Github, the CD workflow (Github action) is activated and it:

1. Sets up Python
2. Installs the necessary dependencies.
3. Builds the package using the Setup.py file.
4. Publishes the package to the Test.PyPi repository.

2.3 Code Inspection and Refactoring

Flake8 did not generate any errors/warning for the entire code-base, Pylint however generated errors and warnings for almost every file. To keep the CI workflow result in a pass without having to change the entire project, Pylint was limited to checking rules/apt_get.py and ag_literal.py. Pylint finished successfully after refactoring the two files. This was done by adding docstrings to each function and changing some names of variables to their correct typographical convention. Each rule and test consist of a separate python file this creates an clear project structure, the rule also needs to be added to Readme.md file together with a description.

The vulnerability check runs using Bandit resulted in 897 low, 10 medium and 1 high priority issue. At first it was determined to fix the high priority vulnerability, however after attempting to fix the problem it was determined that it would take to much time and resources to fix the problem and was therefore skipped. As to not keep getting failed Bandit executions and stopping the workflow, it was chosen to comment it out. In the future it could be easily be put in again to start checking the code for vulnerabilities again.

A big part of the fuck are the multitude of different command line mistakes that it can fix. This is possible because volunteers are able to write and submit their own rules which correct a mistake. We decided that we wanted to contribute to these rules and write our own including its accompanying test. *cd_prev.py* This rule adds a space in the "cd -" command when it is written together. *gcloud_auth.py* When a user forgets to login with its gcloud account before executing a command of gcloud the command can't be executed. The gcloud rule first logs the user in afterwhich it runs the command the user tried before.

To increase the test coverage of the already existing code-base a test is written to test the *cargo.py* rule. This rule existed in the code-base before it was cloned,lllllllll but did not contain a test. After writing a test for this rule, every rule had one.

2.4 Testing

The fuck source already included unit test and functional test which were written using the Pytest framework. The TDD cycle is applied when developing the new rules. Unit tests where created first, then the functional code was written to make the test pass.

The first TDD cycle was done creating the gcloud_auth.py rule, the second TDD cycle was done creating the cd_prev.py rule. Each test file covers 3 test cases:

- 'test_match' to test the match function, test case: error message matches the rule
- 'test_not_match' to test the match function, test case: error message does not match the rule
- 'test_get_new_command' to test 'get new command' function, test case: the error command relates to the new created rule, the return command correct the error

We also increase the coverage slightly by 0.32% by adding some simple tests to the file missing coverages

Using an already written unit tests it was discovered that when a new rule is added, it also needed be added to the Readme.md file. If the test had not given an error this would not have been done, and the code-base would have been worse because of it. This gave a clear example why unit tests are necessary.

2.5 Project Management

The project management tool is used is Project boards on GitHub[4]. It is made up of issues, pull requests, and notes that are categorized as cards in 5 columns: To do, In progress, Review In Progress, Reviewer Approve, and Done. The board also configured actions that automatically move the card between columns. Below is the rules for card to travel to the columns, it basically relates either on issue or pull request(PR):

- To do: new Issued added
- In progress: closed Issued/PR reopened, new PR
- Review In Progress: PR is pending approval by reviewer
- Reviewer Approve: PR is approved by reviewer
- Done: Issued closed, PR merged, PR closed without merge

Issues in open source project is a prominent tool to track the tasks, enhancements, bugs which are created by both contributors and users. So we are in favor of the management tool that facilitate the collaboration and reference to the issues. We found that project boards not only help to organize our works but it also has the automatic workflow to keep the status of the project in sync with the associated issues and pull requests.

3 Methodology

3.1 Agile practice: Your project

The following practices does our system support, *Control Code Versions and Development Branches, Automate Tests, Use Continuous Integration and Automate Deployment.*

3.1.1 Which practices does your project support?

For control code version we used the given GitHub classroom repository. For every new class a new branch is used to prevent conflicts with the main branch. After a class is finished an tested it is merged to the main branch. Using different branches allows for a greater number of people working on the same code branch without interfering with each other.
After committing and pushing the code to the repository, a GitHub action is run which contains unit and functional tests. The automated testing makes sure that everyone understands the current state of the repository/branch, it also makes a person more likely to push code that is tested on the local machine before pushing. Having the automated tests does require the upkeep of writing new tests for new classes. The project supports Continuous integration using Github actions, this is a CI tool provided by Github which makes it easy to integrate in a repository. Code style checks and unit tests are included in the workflow which improves the metrics of the tests by executing them more frequently.

A continuous integration workflow was also implemented using Github actions. The user is able to publish the current main branch after which the workflow build and exports the package to Pypi.

3.1.2 What is missing in the project?

A workflow could be created that monitors and updates the state of issues. When it's been a long time after the issue has been posted, a label could be attached to it. This would make it more clear to the user which issue needs attention. Adding this feature to the workflow is optional.

3.1.3 What would you need outside of technical knowledge?

Maintaining a big project would require good communication skills. Having an extensive CI/CD system removes a lot of the burdens of managing it.

3.2 Agile Philosophy: Industrial Project

3.2.1 Question a

There are not many projects which use one method, it can be a combination of for example *Joint Application Development* and the *Waterfall* method. In figure 5 is this classified as a hybrid method and thus so not a traditional method. Apparently more then half of the projects which uses the *Waterfall* method, also use another method.

3.2.2 Question b

There are some points which can actually benefit the progress in multi-million projects. The point of *Deprecation of upfront tasks* says that trying out smaller applications before implementing it in the bigger project is no serious requirement but can actually help the process. Implementing all the code in one run which can lead to lots of mistakes. This way, it can be hard to find out which piece of code is the mistake. Thus so, in bigger multimillion it can be necessary to develop code in smaller pieces.
The point of *User stories as a basis of requirements* can also be necessary to fulfill a multi-million project. Every project can be really specific and based on a user story, the big customer. The justified argument which is given to set a user story into requirements for the project.
The point of *Rejection of traditional manager tasks* is of course justified, a bigger project and especially a multi-million project should be coordinated by a manager, one man who has an overview of what is going on and if goes into the good direction. With this method is there one person who is responsible for the deadlines and contact with for example the customer.

3.2.3 Question c

The most likely methodology which is used for the project is the Agile methodology. The final project is done with one small team which fits the requirement. The organization behind the final project is also small with a small number of employees. The budget of the project is also low, there is no budget as it is considered an educational project.

References

[1] URL: `https://github.com/nvbn/thefuck/tree/master/thefuck`.

[2] URL: `https://nvbn.github.io`.

[3] URL: `https://www.python.org/dev/peps/pep-0008/`.

[4] URL: `https://docs.github.com/en/issues/organizing-your-work-with-project-boards/managing-project-boards/about-project-boards`.