Assignment 1: Packaging Greengraph

Rianna Kelly
Student ID: 16100406
Course: MPHYG001

December 18, 2016

Abstract

The assignment was to package the greengraph code written by Dr. James Hetherington (taken from the UCL research software engineering library[1]) into a git repository, suitable to be installed using pip and to be called from the command line. This report will document the entry point and usage of the greengraph package, as well as discuss problems encountered during this assignment. It will also briefly reflect on wider issues around preparing work for release, the use of package managers, and how to initially consider building a community of users for a project.

1 Introduction

The code has been broken up into appropriate files, and arranged into an appropriate folder structure. Git version control was used and relevant commit messages were submitted. Each method within the two classes (Greengraph and Map) has an automated test, and the build status and test coverage can be seen on the homepage of the following git repository: `https://github.com/RiannaK/Coursework1`. Appropriate fixtures are defined where necessary, and mocks are used to avoid the tests interacting with the internet. The packaged code is suitable to be installed either directly through python setup.py or via pip installation, with usage and install instructions clearly defined within the README.md file of the git repository. Further files are included in the git repository relating to the licence for further use and the citation requirements.

2 Entry Point and Usage

The package can be installed on the user's computer either by manually downloading from GitHub and installing, or by installing directly from GitHub using the standard pip command.

Once installed, the greengraph script can be called from the command line using the following syntax:

```
greengraph --from Bristol --to Cambridge --steps 50 --out my_image.png
```

There are four command line arguments:

Arg 1	Arg 2	Default
--from	-f	London
--to	-t	Oxford
--steps	-s	10
--out	-o	GreenGraph.py

If any inputs are omitted, the script will be run assuming the default arguments outlined in the table above were provided.

In the example above, the greengraph code will consider the beginning position to be Bristol and the final point to be Cambridge. Ten linearly spaced points (i.e. ten latitude and longitude pairs) between Bristol and Cambridge will then be calculated, and the number of green pixels in each satellite image will be counted (each satellite image is a 400x400 pixel image centred at the specified latitude and longitude point). A line graph is then produced plotting the green pixel count at each of these ten points, overall it will show the green pixel count between Bristol and Cambridge.

The output graph showing the green pixel count at each step will be generated automatically as a pop-up on the screen, and will also be saved in the current directory. Unless specified, the default file name for this image is 'GreenGraph.png'.

3 Problems Encountered

I encountered a range of problems during this assignment. It was not initially clear how to use 'from' as a command line parameter as python reserves this keyword. This was solved using the keyword argument 'dest' which parses the input to an alternative field. It took a while to get the continuous integration build server – Travis – properly configured as I was unfamiliar with the configuration of the travis.yml file before beginning this project.

By default, patch will return a MagicMock as its return value if no value is specified. I had trouble ensuring all methods would complete as it was not always obvious how to return sensible mock data from patched requests. For instance, I had to programmatically generate test data for Map's 'green' method. If a real .png file was used to test this method, it would not be clear how to assert whether the return value was correct. I also provided a mock byte array to the image reader, although I note that the image reader could have been mocked in its entirety.

Any method accessing an external resource should be patched to ensure that the tests will not fail as a result of external factors (failing internet connection/google maps API temporarily down). I had some trouble knowing which methods access the internet, and thus needed patching. This was resolved by disabling the internet on my machine and ensuring that all tests continued to pass.

Although it did not cause direct trouble, there is always an issue of knowing how much of the code has been covered by unit tests. I resolved this using Coveralls and pytest's coverage package. For example, the Travis script runs the following command (corresponding output also shown below):

```
py.test --cov-report term-missing --cov=greengraph

greengraph/test/test_command.py .
greengraph/test/test_graph.py ......
greengraph/test/test_map.py .......

----------- coverage: platform linux, python 3.5.2-final-0 -----------
Name                              Stmts   Miss  Cover   Missing
---------------------------------------------------------------
greengraph/__init__.py                0      0   100%
greengraph/command.py                19      0   100%
greengraph/graph.py                  27      0   100%
greengraph/map.py                    33      0   100%
greengraph/test/__init__.py           0      0   100%
greengraph/test/test_command.py      26      0   100%
greengraph/test/test_graph.py        62      0   100%
greengraph/test/test_map.py         118      0   100%
---------------------------------------------------------------
TOTAL                               285      0   100%
========================= 14 passed in 2.74 seconds ==========================
```

4 Further Considerations

The following section of the report will briefly reflect on wider issues around preparing work for release, the use of package managers, and how to initially consider building a community of users for a project.

4.1 Advantages and disadvantages of packaging

Below we will consider the advantages and disadvantages (costs) involved in preparing work for release and using package managers.

Advantages

There are a number of advantages to preparing work for release and using package managers. Of all of these, perhaps the most important reason is to ensure that code which will likely prove useful to a sizeable audience is freely and easily available for use. This ensures that programmers can spend more time developing application specific logic and less time 'reinventing the wheel' by programming generic code that has been written many times before.

Of course, once code is available online it has the advantage of being open to peer-review and (one hopes, constructive) criticism from the open-source community. A community of programmers working on a code base will invariably produce code that is of a higher standard than a code base developed in isolation and so this feedback process is of considerable importance. It should be noted that people of all levels of technical expertise can contribute; developers can work directly on the source code and less technical users can raise issues/bugs to be addressed by the development team.

It is also important to consider the fact that preparing the work to be released forces the lead contributor to think of the final user; what is the aim of the library? Is it clear how it will be used? Is it user-friendly? Does it build? Do all tests pass?

Sharing code provides a strong incentive to developers to produce code to a high standard. Unit, component and integration tests will typically be written which provides end users with a certain level of confidence in the code's reliability. Moreover, it gives developers who wish to contribute the confidence to refactor and improve the code base without fear of inadvertently breaking existing functionality. Within this assignment, I chose the 'Arrange, Act, Assert' pattern for formatting and arranging the code within my unit tests[2]. I also included the 'system under test' (sut) pattern to clearly distinguish the tested method from the setup[3]. Finally, tests are self-documenting and so are one of the easiest ways to become familiar with the library's functionality.

Disadvantages

There is a fairly steep learning curve associated with packing the code, which can be a deterrent for beginner/intermediate programmers. Furthermore, packaging may not always be appropriate. For instance, ad hoc scripts with limited re-usability would not be sensible candidates for packaging.

Finally, a possible disadvantage of packaging code could be the introduction of duplicate code. Before any code is shared online, or better still, written, the Python Package Index (PyPI) should be checked.

4.2 Building a community of users

There are several steps to consider when building a community of users for a project. Initially it must be clear that the library is necessary. In many cases the package may already be part of an existing library, or could simply be an additional feature or fix to an existing library. It is always preferred that an existing library is upgraded as opposed to duplicated.

Once a project idea is established, a first step would be to use a version control system, such as Git. Many others are available, such as CVS and Subversion. Git is complemented by GitHub, an online repository hosting service, and is well suited to community-based collaboration.

Git also provides a platform to implement a branching strategy. For this assignment, I used a simplified version of the referenced branching model[4]. In this model, there are two versions of the code with the develop branch existing in parallel to the master branch. The bulk of the work was completed on the develop branch and, once the code was in a good working state, was merged into the master branch. A more sophisticated branching strategy would utilise more branches to isolate individual development features. Finally, to moderate code and ensure quality, a subset of core developers should be responsible for merging code into the two main branches via pull requests.

The aim is to build a community of users; therefore, it is imperative to attract and retain users. This requires the code to be regularly maintained and queries from users and contributors to be responded to within an appropriate time frame. A project with contributors that regularly share work and discuss will create an effective environment that will ultimately improve the project for all users and contributors.

GitHub offers a very useful issue section that allows contributors to report errors in the code. This should be widely encouraged, and the GitHub issues feature allows people to be assigned to each issue. Each contributor will bring a unique skill and motivation to the project, and it is helpful to assign each issue based on relevant experience. I found that issues can be closed via a git commit using keywords such as 'closes' and 'fixes'. For example, I closed issue three on my assignment using the git commit message 'Fixed issue of blank graph. This closes #3'.

Adding a detailed README.md file can be an incredibly useful tool for presenting the package to others. At a minimum, it should include the aim of the package, installation instructions and example usage. Further supplementary information is advisable – for instance, I included Travis and Coveralls badges to show that the package builds and all tests pass with 100% coverage.

It is also vital to choose the correct software license and ensure that the relevant license and citation files are visible.

References

[1] 110Capstone
`http://development.rc.ucl.ac.uk/training/engineering/ch01data/110Capstone.html`

[2] Unit Test Basics
`https://msdn.microsoft.com/en-GB/library/hh694602.aspx`

[3] SUT at XUnitPatterns.com
`http://xunitpatterns.com/SUT.html`

[4] A successful git branching model
`http://nvie.com/posts/a-successful-git-branching-model/`