MPHYG001 Assignment 1: Packaging Greengraph

Danial Dervovic
University College London
danial.dervovic.11@ucl.ac.uk

January 5, 2016

Introduction/Usage

For this assignment, the task was to take the Greengraph code[1], packaging it up into a form that can be `pip` installed and run from the command line. This code may be found at the GitHub repository `ddervs/GreenGraph`. The usage of the packaged function is shown below (the output of `greengraph --help`).

```
usage: greengraph [-h] [--steps STEPS] [--out OUT] start end

Plots amount of green space between two locations.

positional arguments:
  start          starting location, string
  end            final location, string

optional arguments:
  -h, --help     show this help message and exit
  --steps STEPS  number of calculated points, defaults to 20, integer
  --out OUT      output file, "*.png" or "*.pdf"
```

Problems

The main problems encountered when completing the work were creating the automated tests and using the `mock` library.

Knowing which cases should be tested for in each method was tricky, requiring thought about which few input data were representative of what a given method would encounter in its usage. Naturally, this wasn't completely exhaustive, and extended usage of the package would help to make the testing suite more watertight.

For the methods such as `Map.count_green`, where it was fairly clear what representative cases were needed, these inputs and outputs were coded directly into each `test_*` method, either testing if a certain output was returned for a given input, or if functions residing within the method had been called with the correct arguments. With two methods, `Greengraph.location_sequence` and `Greengraph.green_between`, fixtures were used in their tests. In the former, the method was hard coded into a generator script, with test inputs. Then, the input and associated output were dumped into a YAML file, that the test script then used to test the method against. While seemingly circular in that the method effectively tests against itself, were the master method to change, it would still be being tested against the correct YAML fixtures file. The hard-coding of the method in the generator script means that if more test input and output is needed, the fixtures will still be reliable, even if the method itself had changed.

In the test for `Greengraph.green_between`, the same procedure is used to generate the output for given input, which is dumped into a fixtures file, along with some intermediate data, which is used by mocks within the test. The method is then tested using the mock data, for equality with the fixtures file output. With the fixtures generation script for this method, internet resources are used, but once the YAML file is generated, the unit test itself *does not* use internet resources.

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

The main issue with using the `mock` library was getting used to the concept of a mock itself and knowing which methods correspond to the various functionalities. Once this was understood, there were no further problems.

Preparing Work for Release

Preparing work for release has advantages and disadvantages. Some advantages are:

- Open source code which is useful to others is improved and built-upon by the community. Having more people look at code can only improve its quality.
- Code which is released may have a life beyond the time it's useful to its author.
- Releasing code forces the author to improve its readability and quality pre-release, if they want to see the benefits of releasing it.

There are some costs however:

- Extra time is required to prepare work for release, which could be better spent. The value of a particular code set to the wider community must be weighed up before taking the time to release the code.
- If the code has potential commercial uses, making the code open-source takes away the potential for directly earning money from it. That said, it is possible to earn money from open-source code, such as paid additional functionality or learning materials.

On balance, it appears that releasing research code as open-source is in general a good idea.

Research software would generally be released via some online code repository such as GitHub, or a package index such as PyPI. With PyPI, submitting a package is somewhat complicated but there are easily accessible instructions on how to do so[2]. By submitting software to PyPI, it can be easily installed by anyone with an internet connection using `pip`. The benefit of this is that all dependencies are (usually) installed automatically, software conflicts are safely handled and the software is usable after a one line command. A disadvantage to this is that this procedure isn't compatible with all systems, so sometimes having code that a user compiles from source can be the better option.

Building a Community of Users

For a software project to have any kind of longevity, it needs a community of people both using the software and maintaining the software.

The first and possibly most important thing needed, to encourage the growth of such a community, is for the project to solve a unique problem and be worth learning/adding to. If the same thing has been done before there is no need for users to migrate from using the existing software.

Provided this is the case, it is important for the software to be under open version control (on GitHub for instance). The software needs to be promoted at conferences, online, or by word-of-mouth, to whomever it is relevant to. Usage of an issue tracker and prompt response to bug reports is also a necessity, especially before a community of contributors has been established.

[2] `http://peterdowns.com/posts/first-time-with-pypi.html`