Maintaining your code and tests

Because you are never finished

Ellert van der Velden

ADACS Code Testing Workshop 2019

Why should you maintain your code (and tests)?

Why should you maintain your code (and tests)?

- Common responses against:

Why should you maintain your code (and tests)?

- Common responses against:
  - I am the only one using my code (also in response to documentation);

Why should you maintain your code (and tests)?

- Common responses against:
  - I am the only one using my code (also in response to documentation);
  - It works on my machine!;

Why should you maintain your code (and tests)?

- Common responses against:
  - I am the only one using my code (also in response to documentation);
  - It works on my machine!;
  - I know how to use my code;

Why should you maintain your code (and tests)?

- Common responses against:
  - I am the only one using my code (also in response to documentation);
  - It works on my machine!;
  - I know how to use my code;
  - My tests are sufficient enough;

Why should you maintain your code (and tests)?

- Common responses against:
  - I am the only one using my code (also in response to documentation);
  - It works on my machine!;
  - I know how to use my code;
  - My tests are sufficient enough;
  - Etc…

Why should you maintain your code (and tests)?

- My counter-responses:

Why should you maintain your code (and tests)?

- My counter-responses:
  - Python is a very rapidly evolving programming language:

Why should you maintain your code (and tests)?

- My counter-responses:
  - Python is a very rapidly evolving programming language:
    - No single minor version has survived for 2 years before being succeeded.

Why should you maintain your code (and tests)?

- My counter-responses:
  - Python is a very rapidly evolving programming language:
    - No single minor version has survived for 2 years before being succeeded.
  - Python is also increasingly becoming more popular:

Why should you maintain your code (and tests)?

- My counter-responses:
  - Python is a very rapidly evolving programming language:
    - No single minor version has survived for 2 years before being succeeded.
  - Python is also increasingly becoming more popular:
    - Currently the 3rd most popular programming language, after C and Java.

Why should you maintain your code (and tests)?

- My counter-responses:
  - Python is a very rapidly evolving programming language:
    - No single minor version has survived for 2 years before being succeeded.
  - Python is also increasingly becoming more popular:
    - Currently the 3rd most popular programming language, after C and Java.
  - Future you is a different person.

Why should you maintain your code (and tests)?

- My counter-responses:
  - Python is a very rapidly evolving programming language:
    - No single minor version has survived for 2 years before being succeeded.
  - Python is also increasingly becoming more popular:
    - Currently the 3rd most popular programming language, after C and Java.
  - Future you is a different person.
- Python code/packages become outdated incredibly quickly.

Python basics

Python basics

- Python is designed as an open-source programming language;

Python basics

- Python is designed as an open-source programming language;
- Code sharing and recycling is encouraged;

Python basics

- Python is designed as an open-source programming language;
- Code sharing and recycling is encouraged;
- Result: Your code probably relies on a few (or more) third-party packages;

Python basics

- Python is designed as an open-source programming language;
- Code sharing and recycling is encouraged;
- Result: Your code probably relies on a few (or more) third-party packages;
- Solution: Write these down in a *requirements.txt* file;

Python basics

- Python is designed as an open-source programming language;
- Code sharing and recycling is encouraged;
- Result: Your code probably relies on a few (or more) third-party packages;
- Solution: Write these down in a *requirements.txt* file;
  - Setup files can take these requirements into account automatically.

Code requirements

Code requirements

```
codecov
numpy>=1.12.0
pytest>=3.8.0
pytest-cov
pytest-pep8
scipy>=1.1.0
```

Code requirements

```
codecov
numpy>=1.12.0
pytest>=3.8.0
pytest-cov
pytest-pep8
scipy>=1.1.0
```

```
e13tools>=0.6.8
h5py>=2.8.0
hickle>=3.4.0
matplotlib>=2.2.4
mlxtend>=0.9.1
mpi4pyd>=0.2.4
numpy>=1.12.0
pyqt5>=5.9
pytest-mpl>=0.10.0
scikit-learn>=0.19.1
scipy>=1.0.0
sortedcontainers>=1.5.9
threadpoolctl>=1.0.0
tqdm>=4.7.6
```

Code requirements

- Two common mistakes:

```
codecov
numpy>=1.12.0
pytest>=3.8.0
pytest-cov
pytest-pep8
scipy>=1.1.0
```

```
e13tools>=0.6.8
h5py>=2.8.0
hickle>=3.4.0
matplotlib>=2.2.4
mlxtend>=0.9.1
mpi4pyd>=0.2.4
numpy>=1.12.0
pyqt5>=5.9
pytest-mpl>=0.10.0
scikit-learn>=0.19.1
scipy>=1.0.0
sortedcontainers>=1.5.9
threadpoolctl>=1.0.0
tqdm>=4.7.6
```

Code requirements

- Two common mistakes:
  - Not specifying all requirements;

```
codecov
numpy>=1.12.0
pytest>=3.8.0
pytest-cov
pytest-pep8
scipy>=1.1.0
```

```
e13tools>=0.6.8
h5py>=2.8.0
hickle>=3.4.0
matplotlib>=2.2.4
mlxtend>=0.9.1
mpi4pyd>=0.2.4
numpy>=1.12.0
pyqt5>=5.9
pytest-mpl>=0.10.0
scikit-learn>=0.19.1
scipy>=1.0.0
sortedcontainers>=1.5.9
threadpoolctl>=1.0.0
tqdm>=4.7.6
```

Code requirements

- Two common mistakes:
  - Not specifying all requirements;
  - Not specifying minimum required versions.

```
codecov
numpy>=1.12.0
pytest>=3.8.0
pytest-cov
pytest-pep8
scipy>=1.1.0
```

```
e13tools>=0.6.8
h5py>=2.8.0
hickle>=3.4.0
matplotlib>=2.2.4
mlxtend>=0.9.1
mpi4pyd>=0.2.4
numpy>=1.12.0
pyqt5>=5.9
pytest-mpl>=0.10.0
scikit-learn>=0.19.1
scipy>=1.0.0
sortedcontainers>=1.5.9
threadpoolctl>=1.0.0
tqdm>=4.7.6
```

Code requirements

- Two common mistakes:
  - Not specifying all requirements;
  - Not specifying minimum required versions.
- Both are annoying and tedious to deal with as a user, especially the latter.

```
codecov
numpy>=1.12.0
pytest>=3.8.0
pytest-cov
pytest-pep8
scipy>=1.1.0
```

```
e13tools>=0.6.8
h5py>=2.8.0
hickle>=3.4.0
matplotlib>=2.2.4
mlxtend>=0.9.1
mpi4pyd>=0.2.4
numpy>=1.12.0
pyqt5>=5.9
pytest-mpl>=0.10.0
scikit-learn>=0.19.1
scipy>=1.0.0
sortedcontainers>=1.5.9
threadpoolctl>=1.0.0
tqdm>=4.7.6
```

Not specifying all code requirements

Not specifying all code requirements

- Common reasons:

Not specifying all code requirements

- Common reasons:
  - The requirement in question is a very common package, like NumPy;

Not specifying all code requirements

- Common reasons:
  - The requirement in question is a very common package, like NumPy;
  - The requirement in question is satisfied by another requirement.

Not specifying all code requirements

- Common reasons:
  - The requirement in question is a very common package, like NumPy;
  - The requirement in question is satisfied by another requirement.
- This is wrong as you cannot guarantee these assumptions.

Not specifying all code requirements

- Common reasons:
  - The requirement in question is a very common package, like NumPy;
  - The requirement in question is satisfied by another requirement.
- This is wrong as you cannot guarantee these assumptions.

- TODO: Specify all YOUR imports as requirements, unless they are builtins.

Not specifying minimum required versions

Not specifying minimum required versions

- Specifying the minimum required version guarantees functionality;

Not specifying minimum required versions

- Specifying the minimum required version guarantees functionality;
- Failing to do so can lead to irritating and frustrating situations for the user.

Not specifying minimum required versions

- Specifying the minimum required version guarantees functionality;
- Failing to do so can lead to irritating and frustrating situations for the user.

- TODO: Use your current versions as the minimum versions.

Pytest plugins

Pytest plugins

- Many packages provide plugins for *pytest* to make it easier to test certain features of your code:

Pytest plugins

- Many packages provide plugins for *pytest* to make it easier to test certain features of your code:
  - *pytest-mpl*: Tools for testing and comparing Matplotlib figures;

Pytest plugins

- Many packages provide plugins for *pytest* to make it easier to test certain features of your code:
  - *pytest-mpl*: Tools for testing and comparing Matplotlib figures;
  - *pytest-pep8*: Tests if your code is PEP8-compliant;

Pytest plugins

- Many packages provide plugins for *pytest* to make it easier to test certain features of your code:
  - *pytest-mpl*: Tools for testing and comparing Matplotlib figures;
  - *pytest-pep8*: Tests if your code is PEP8-compliant;
  - *pytest-cov*: Check the code coverage of your tests.

Pytest plugins

- Many packages provide plugins for *pytest* to make it easier to test certain features of your code:
  - *pytest-mpl*: Tools for testing and comparing Matplotlib figures;
  - *pytest-pep8*: Tests if your code is PEP8-compliant;
  - *pytest-cov*: Check the code coverage of your tests.
- Can be easily installed using *pip* and enabled with *pytest --xxx*.

Code coverage: What?

Code coverage: What?

```
---------- coverage: platform win32, python 3.6.6-final-0 ----------
Name                                Stmts   Miss  Cover   Missing
-----------------------------------------------------------------
example_scripts\__init__.py             0      0   100%
example_scripts\downsampler.py         91     69    24%   39-103, 141-172, 215-217, 222-225, 228-231, 263-273
example_scripts\galaxy.py              58     21    64%   51-52, 149, 168-169, 206-219, 245-254
-----------------------------------------------------------------
TOTAL                                 149     90    40%
```

Code coverage: Why?

Code coverage: Why?

- Write near-exhaustive tests;

Code coverage: Why?

- Write near-exhaustive tests;
- Check for code redundancy;

Code coverage: Why?

- Write near-exhaustive tests;
- Check for code redundancy;
- Find non-covered code;

Code coverage: Why?

- Write near-exhaustive tests;
- Check for code redundancy;
- Find non-covered code;
- Special test-cases help in the future.

Code coverage: How?

Code coverage: How?

- Aim for 100% coverage, including branch coverage;

Code coverage: How?

- Aim for 100% coverage, including branch coverage;
- If that is not possible, ask yourself why (maybe use *pragma: no cover*);

Code coverage: How?

- Aim for 100% coverage, including branch coverage;
- If that is not possible, ask yourself why (maybe use *pragma: no cover*);

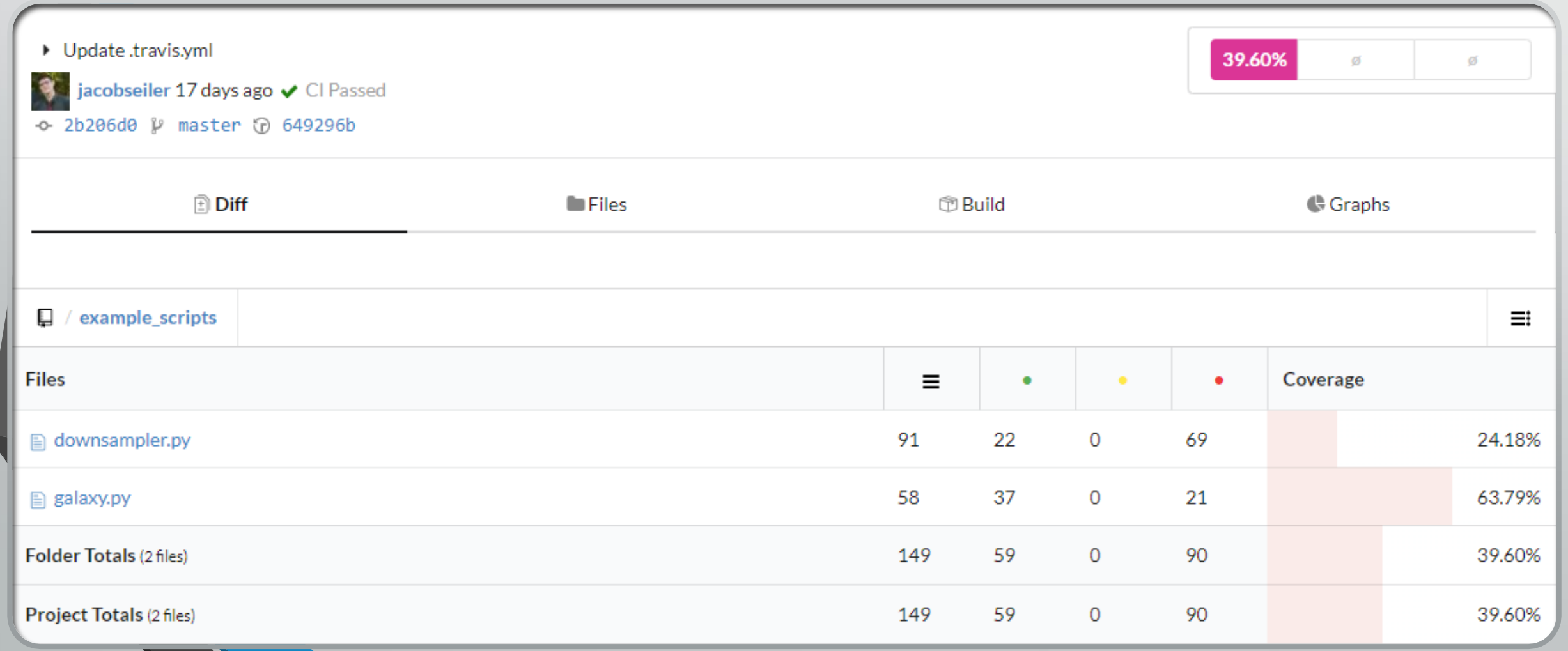
```
main_code()
if flag:  # pragma: no cover
    do_action()
    do_another_action()
main_code_continued()
```

Code coverage: How?

- Aim for 100% coverage, including branch coverage;
- If that is not possible, ask yourself why (maybe use *pragma: no cover*);
- Make sure to write a single test for a single coverage case (e.g., do not cover multiple exception cases in the same test).

Code coverage: Where?

Code coverage: Where?



Update .travis.yml

jacobseiler 17 days ago ✔ CI Passed

2b206d0 master 649296b

39.60%

Diff | Files | Build | Graphs

/ example_scripts

Files					Coverage
downsampler.py	91	22	0	69	24.18%
galaxy.py	58	37	0	21	63.79%
Folder Totals (2 files)	149	59	0	90	39.60%
Project Totals (2 files)	149	59	0	90	39.60%

Code coverage: Where?

Code coverage: Where?

- Once you have your coverage reports, you can upload them to CodeCov;

Code coverage: Where?

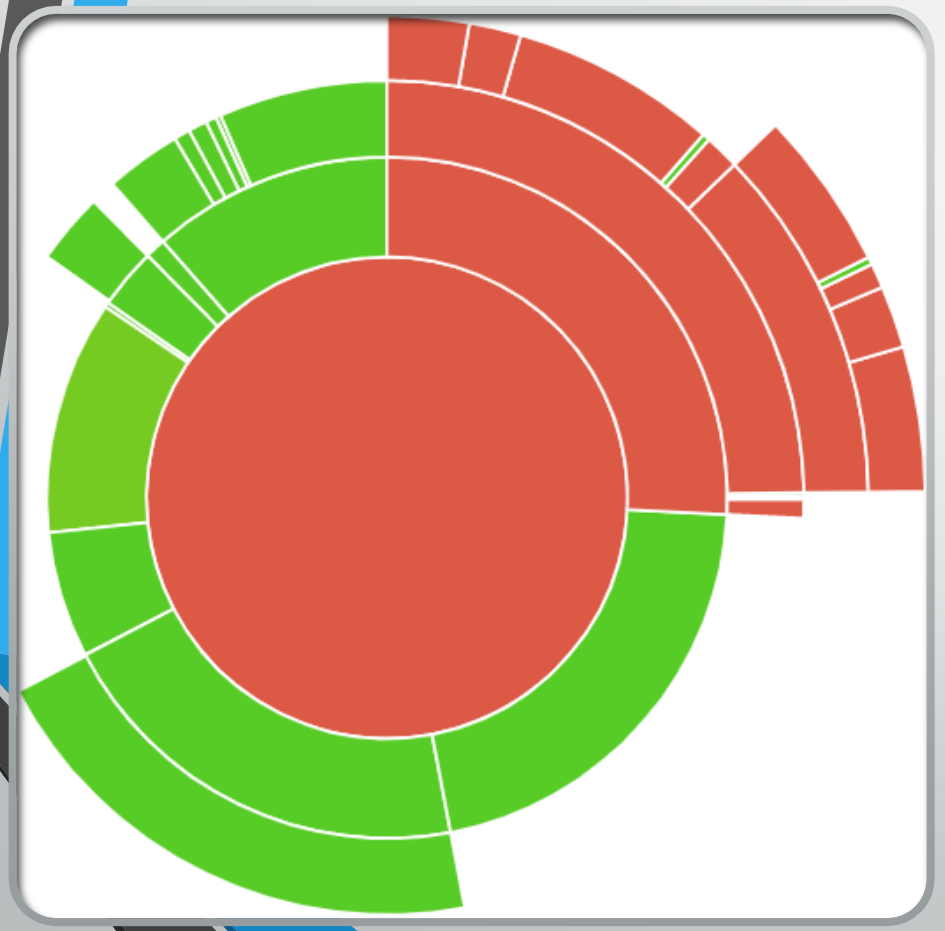
- Once you have your coverage reports, you can upload them to CodeCov;
- CodeCov keeps track of your code coverage;

Code coverage: Where?

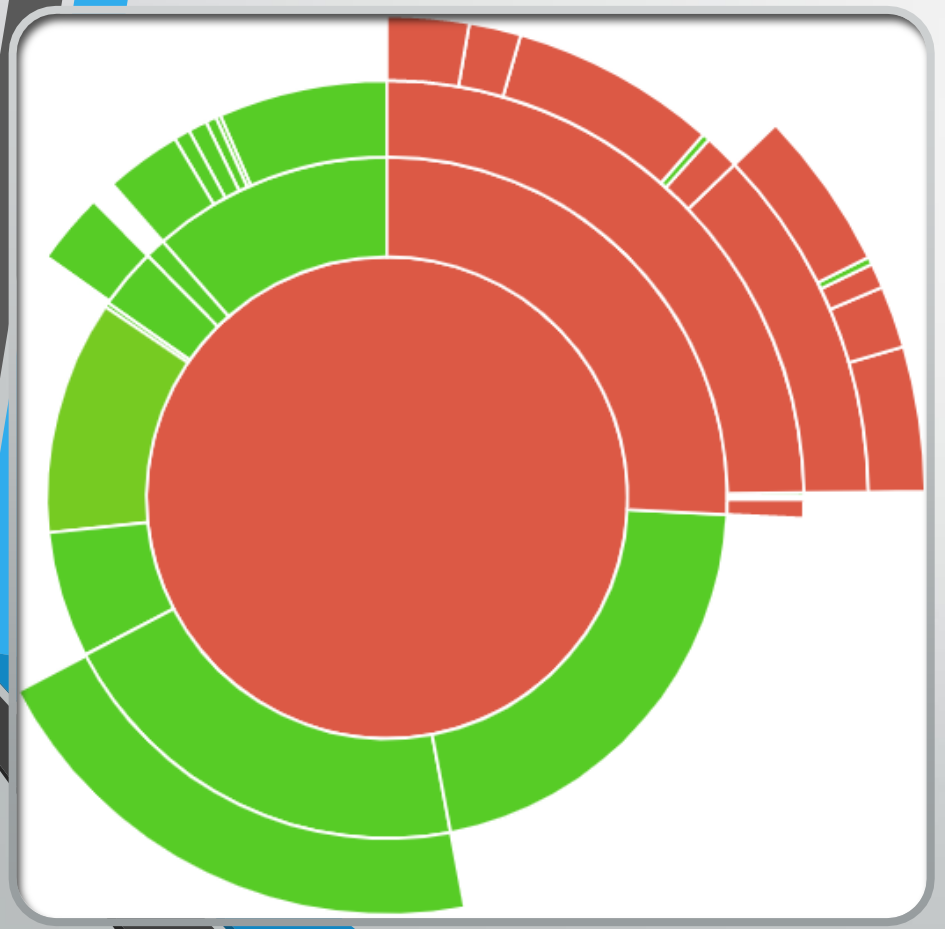
- Once you have your coverage reports, you can upload them to CodeCov;
- CodeCov keeps track of your code coverage;
- It can also provide commit status messages.

Code coverage: Where?

Code coverage: Where?



Code coverage: Where?



+115 +24 +91

[WIP #27] Added a details option that gives the details overview of an emulator iteration. 85.16% 20.93% -2.55%

1313e 12 days ago dev a9c4a4d CI Passed

+119 +8 +111

[WIP #27] Fixed that the options menu sometimes did not open in the center of the GUI. 87.40% 9.68% ø

1313e 13 days ago dev 04f1ac8

+16 +1 +15

[WIP #27] Made some modifications to the way figures are saved. 87.71% 6.78% -0.51%

1313e 14 days ago dev 60d677c CI Passed

+25 +1 +24

[WIP #27] More improvements to the Projection GUI. 88.21% 20.15% -2.90%

1313e 14 days ago dev 46ad99c CI Passed

+159 +25 +134

[WIP #27] Changed the two main parts of the GUI to be dock widgets, allowing them to be moved by the user. 91.12% 4.00% ø

1313e 14 days ago dev 5e9f1a3

+42 +2 +40

Enforce that PyQt5 is used. 92.04% 25.00% ø

Improving your maintenance

Improving your maintenance

- Maintaining your code makes everybody happier;

Improving your maintenance

- Maintaining your code makes everybody happier;
- Specify all your requirements with minimum versions;

Improving your maintenance

- Maintaining your code makes everybody happier;
- Specify all your requirements with minimum versions;
- Aim for 100% code coverage.

Improving your maintenance

- Maintaining your code makes everybody happier;
- Specify all your requirements with minimum versions;
- Aim for 100% code coverage.

- Use a CI service to automate most of the process…

Coverage TL;DR

1. "*pip install pytest-cov*";
2. Add arguments "*--cov --cov-report=term-missing*" to pytest (or to "*addopts*");
3. Optionally add "*--cov-branch*" for branch coverage;
4. Activate repo on CodeCov to start monitoring coverage.