COSC 6386
Program Analysis and Testing
Project Report

Project: Numpy

Git Link: https://github.com/esshariprasad/numpy-private

Team Members:

Embar, Sai Shiva Hari Prasad (ID: 2088713)

Kola, Venkata Seshadri (ID: 2093941)

Nuthalapati, Sampath Babu (ID: 2090339)

1. Introduction:

NumPy is an open-source project aiming to enable numerical computing with Python. It was created in 2005, building on the early work of the Numeric and Numarray libraries. NumPy will always be 100% open source software, free for all to use and released under the liberal terms of the modified BSD license. NumPy is developed in the open on GitHub, through the consensus of the NumPy and wider scientific Python community. For more information on our governance approach, please see our Governance Document.

Testing plays an important role in software development process. Testing ensures that the code is delivered or pushed to the production is working as intended without any unexpected outcomes. Testing at various levels ensures the code quality and desired result of each module is maintained and updated in a successive manner by usage of version controls like git. To test any large project the documentation plays a major role in the testing process as this describes the setting up testing environment and the required prerequisites for the project. Without documentation it is hard to test and debug the project as it takes a lot of time to figure what each module does inside the project and what each module does inside the project.

As in our project we are working on NumPy which is a popular open-source library. Over the years hundreds of contributors contributed their code to the repository in line with the NumPy code standard guidelines.

NumPy maintains a great documentation and testing guidelines [2] for running the tests in the NumPy project.

Developer mindset Vs Tester Mindset:

A developer mindset is quite different from tester mindset where the develop thinks to get his code to be integrated into the production as quickly as possible. Whereas tester thinks to test the developer code thoroughly before approving it to production. Developer and Testing Phase in the SDLC process involves a huge portion of business revenue so careful consideration needs to take in optimizing this phase. Good quality code followed by good quality of tests cases helps in increasing the quality of code which can be co-related by reduction of developer and testers work hours in finding, reporting and fixes bugs inside the developed code.

Developer mindset has different expectations and actions throughout the Software development process as compared when the testers mindset. Coding phase in SDLC is dominated by the developer work hours and testing phase is dominated by tester's work hours both working hard to push new upgrades or features to the project they are working on. Tester might want to create or modify CI/CD pipelines to ensure the code that passes through the pipeline is of production quality level without any undesired outcomes. Whereas developer might want his/her code to be included into the production as early as possible without major tests case failures.

Developers develop software's which usually have huge number of files divided across different modules. These modules are needed to be tested at each new version release of the software product. Tester tests each software product before its release this includes different types of tests ranging from unit tests, regression, integration tests to find various bugs that might be present in code after being unrecognized in the coding phase by the developers.

A successful software release needs great collaboration between the developers and testers.

2. Project Setup:

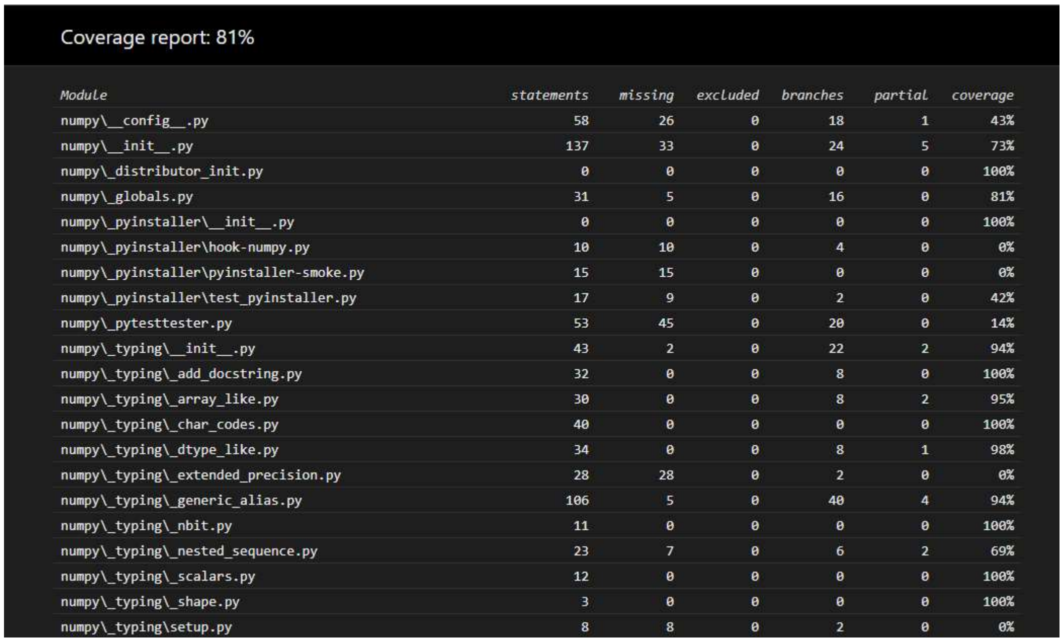
For the complete setup and execution of the project, please check the README.md present in the project's GitHub repository.

3. Testing Coverage:

As mentioned in the NumPy documentation [2]. We can build and run coverage for the coverage using the command *python runtests.py*. To run tests which are present in the project, we use the command *python runtests.py –coverage.*

Above statement builds the project and takes the coverage of each file and stores the html report in this location "*./coverage/ index.html*". After building and getting the coverage we found out that the overall coverage is at 81%. The coverage is present for each file at each file level and as whole. The coverage consists of all the *.py* files as these are in turn responsible for the calling the c program and *f2py* programs using highly exposed C-API used by numpy. NumPy exposes a rich C-API. These are tested using c-extension modules written "as-if" they know nothing about the internals of NumPy, rather using the official C-API interfaces only. Examples of such modules are tests for a user-defined rational dtype in _rational_tests or the ufunc machinery tests in _umath_tests which are part of the binary distribution. Starting from version 1.21, you can also write snippets of C code in tests that will be compiled locally into c-extension modules and loaded into python.

3.1 Report:



Coverage report: 81%

Module	statements	missing	excluded	branches	partial	coverage
numpy__config__.py	58	26	0	18	1	43%
numpy__init__.py	137	33	0	24	5	73%
numpy_distributor_init.py	0	0	0	0	0	100%
numpy_globals.py	31	5	0	16	0	81%
numpy_pyinstaller__init__.py	0	0	0	0	0	100%
numpy_pyinstaller\hook-numpy.py	10	10	0	4	0	0%
numpy_pyinstaller\pyinstaller-smoke.py	15	15	0	0	0	0%
numpy_pyinstaller\test_pyinstaller.py	17	9	0	2	0	42%
numpy_pytesttester.py	53	45	0	20	0	14%
numpy_typing__init__.py	43	2	0	22	2	94%
numpy_typing_add_docstring.py	32	0	0	8	0	100%
numpy_typing_array_like.py	30	0	0	8	2	95%
numpy_typing_char_codes.py	40	0	0	0	0	100%
numpy_typing_dtype_like.py	34	0	0	8	1	98%
numpy_typing_extended_precision.py	28	28	0	2	0	0%
numpy_typing_generic_alias.py	106	5	0	40	4	94%
numpy_typing_nbit.py	11	0	0	0	0	100%
numpy_typing_nested_sequence.py	23	7	0	6	2	69%
numpy_typing_scalars.py	12	0	0	0	0	100%
numpy_typing_shape.py	3	0	0	0	0	100%
numpy_typing\setup.py	8	8	0	2	0	0%

3.2 Table:

	statements	missing	excluded	branches	Partial	coverage
Total	105847	16719	0	29001	2282	81%

4. Effectiveness of the tests:

After building and generating the coverage report for the whole project we found that the coverage stands at 81% which is a good indicator that the developers involved in the NumPy are putting ample amount of time in generating enough test cases to pass the threshold of above > 80% of coverage which indicates that the project is in good shape with enough quality test cases. The project overall average for coverage is gets down by the interfacing modules which are responsible for fortan2python and Python and C using exposed C-API.

If we have a closer look at the coverage report for each file, we observe that the files(scimath.py,) which involve math operations which in turn use exposed C-API for their functionality are giving less coverage compared with other production files. This can be attributed with the memory limitation to assign large memory for few data types like numpy_array based on the memory.

5. Test Files with Assert Statements

NumPy tests are present in each module having its own tests by named folder tests in each module. The testing files in the NumPy project are named under test, testing, tests folders in the numpy directory. NumPy follows a format for test files where each test file inside the numpy project contains the test keyword inside them. This test files can easily be identified with regex expression "*test*.*" is where it matches all the files named test in the numpy project which finds each type of the assert statements in numpy project.

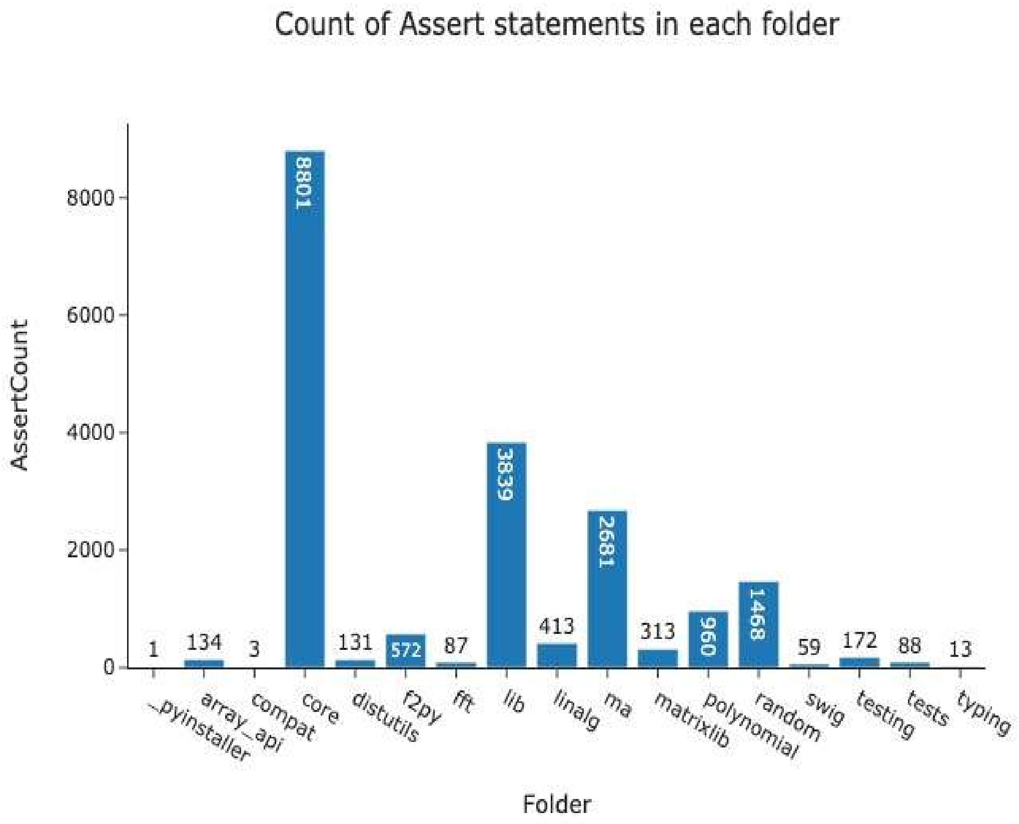
After having closer look at the numpy project we found different types of assert statements used by the developers inside the code that help in different types of assertions inside the code to ensure the module or function works as intended in the project.

Below is the list of different type of assertion found in the code with its regex matching.

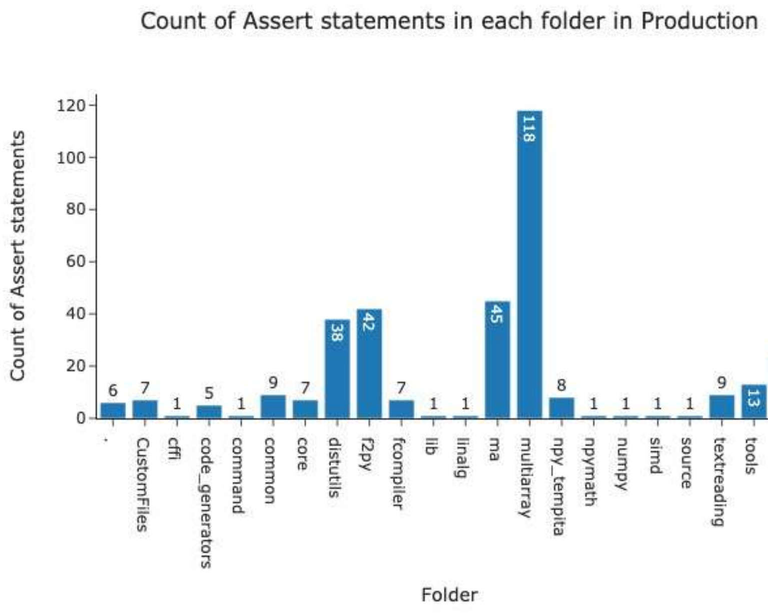
Assert statement type	Regex matches
Assert is instance	assert.* is .*
Assert method()=method()	assert .*\(\)== assert .*\(\)
Logical assert statements: assert a!= b |assert a==b | assert a >= b …	(assert.*==)|(assert.*!=)|(assert.*[><][A-Za-z0-9])
Assert(param, param, param);	assert\(.*;

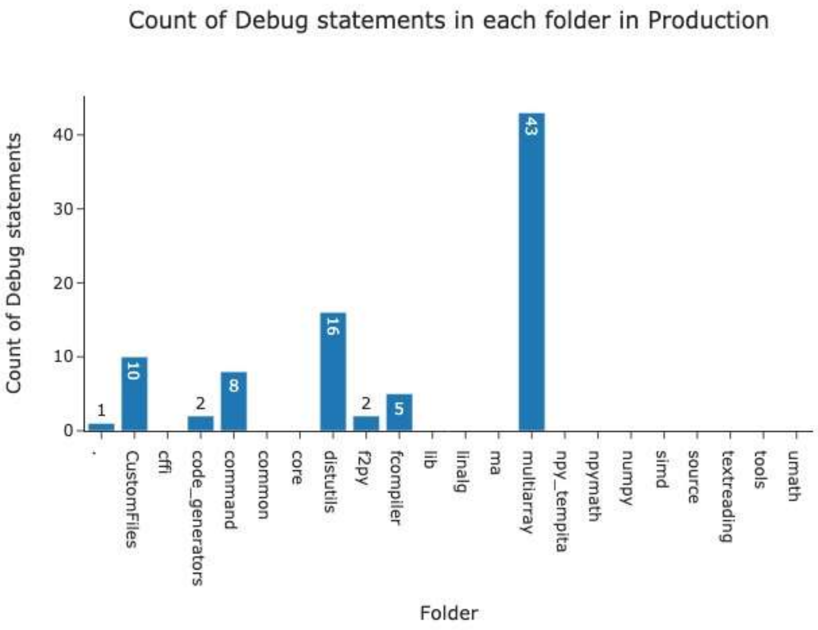
Custom assert: Assert_assertp1_() assert_assertnamep1_assertnamep2	({3,}assert_\w*\(.*\))|({3,}assert_\w{1,}_\w{1,}\(.*\))|(
Remaining Regexes used to match the asserts That are less in number	{3,}assert_\w{1,}_\w{1,}\(.*)|(.{3,}assert [A-Za-z]*.*\()|({3,}assert * [A-Za-z]+\.[A-Za-z (assert\(.*;

Asserts statements play a major part of testing process as this help us understand the expected input and output of each functionality present in different modules of the project. We were keen to capture this data in form of csv as this helps us visualizing the key information about the project.



The above is the plot for all the test files with assert statements in each folder. We can see that the numpy core folder has the highest number of assert statements which is 8801 and is as expected as it contains all the code files responsible for the actual run time of the numpy project. This includes different modules written in different languages like Fortran, C, and python. Then the lib module stands highest next to the core module with 3839 assert statements. The ma module (masked array module) stands third in the above bar chart with a total number of 2681 assert statements. And then the random module with 1468 assert statements. The core, lib, ma, random and polynomial modules comprise of most of the assert statements as seen above.





Numpy project has custom debug statements called as debug_asserts, debug, log.debug inside the numpy project which are captured by using regular expressions.

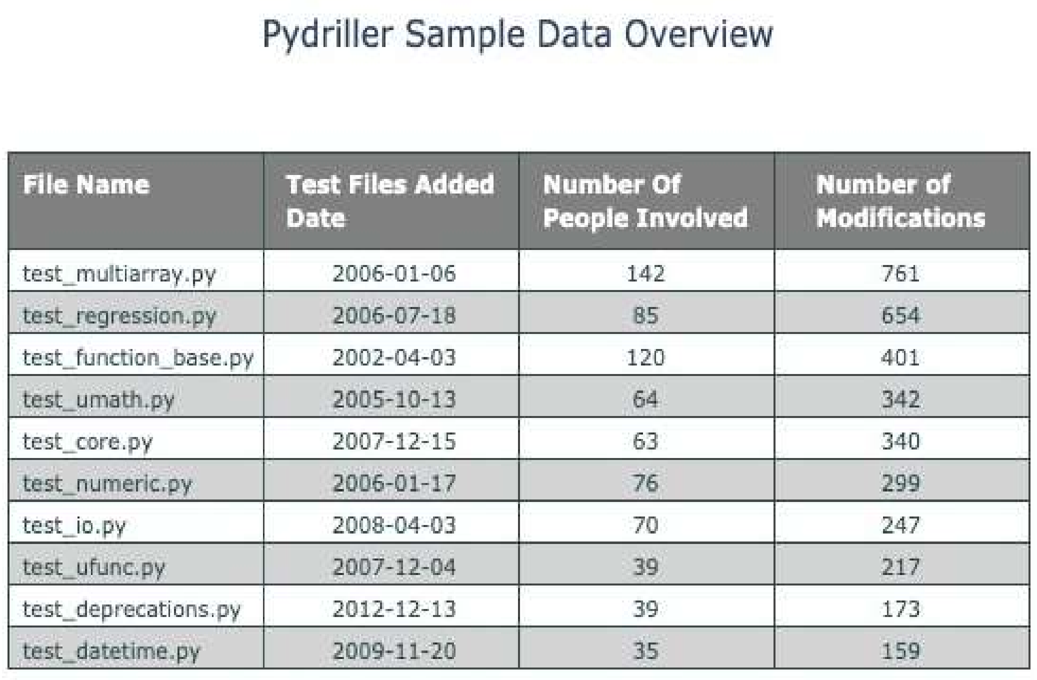
The above two graphs represent the assert and debug statements in production files. In the figure *Assert Statements in Production,* we can see that the *multiarray* has highest number of assert statements which is 118. And similarly, in the figure *Debug Statements in Production,* the *multiarray* again has the highest number of debug statements which is 43, and as the *multiarray* helps in *numpyarray* initialization, and plays a major role in this numpy project. We can see that the two graphs represent the same significance of the *multiarray*.

6. Analysis and Overview of Test Files:

PyDriller [3] is a Python framework that helps developers on mining software repositories. With PyDriller you can easily extract information from any Git repository, such as commits, developers, modifications, diffs, and source codes, and quickly export CSV files.

Plotly [4] allows users to import, copy and paste, or stream data to be analyzed and visualized. For analysis and styling graphs, Plotly offers a Python sandbox (NumPy supported), data grid, and GUI. Python scripts can be saved, shared, and collaboratively edited in Plotly.

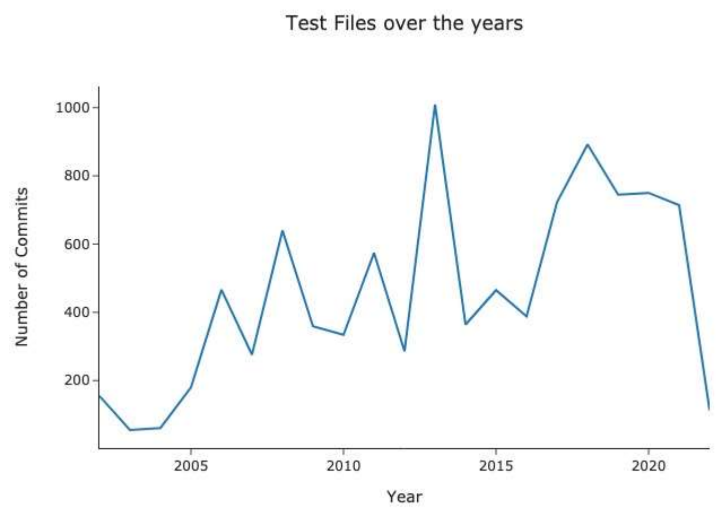
The below table and plots are generated using PyDriller and Plotly which is a Python graphing library which makes interactive, publication-quality graphs and are generated automatically using the datasets which can be found in the csv_data folder of the project directory (or on GitHub).



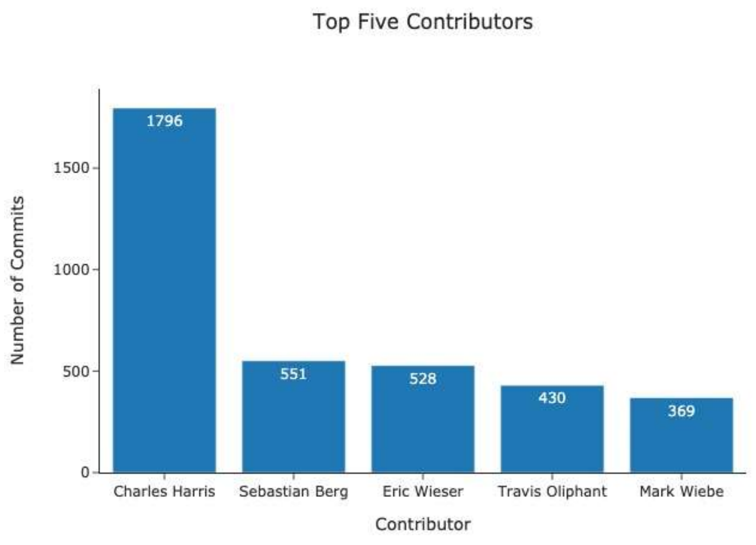
Pydriller Sample Data Overview

File Name	Test Files Added Date	Number Of People Involved	Number of Modifications
test_multiarray.py	2006-01-06	142	761
test_regression.py	2006-07-18	85	654
test_function_base.py	2002-04-03	120	401
test_umath.py	2005-10-13	64	342
test_core.py	2007-12-15	63	340
test_numeric.py	2006-01-17	76	299
test_io.py	2008-04-03	70	247
test_ufunc.py	2007-12-04	39	217
test_deprecations.py	2012-12-13	39	173
test_datetime.py	2009-11-20	35	159

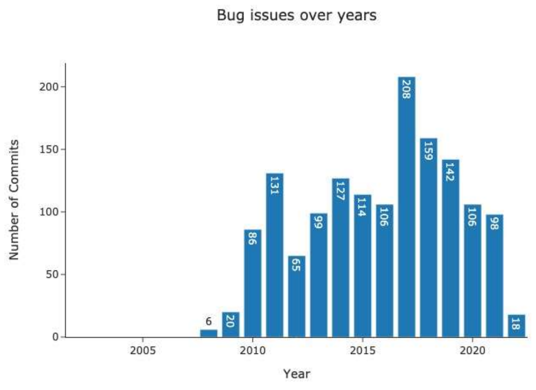
The above table is a sample collection of test files (top 10 ordered by number of modifications) which shows the name of the test files, when the test files have been added to the project, how many people were involved, and how often they are modified. The test file *test_multiarray.py* tops the list with a total number of 761 modifications. The full dataset *overview.csv* can be found in the *csv_data* folder of the project directory (or on GitHub). There's also a detailed dataset which has the path of the test files and along with their modification dates and names of the people who were involved.

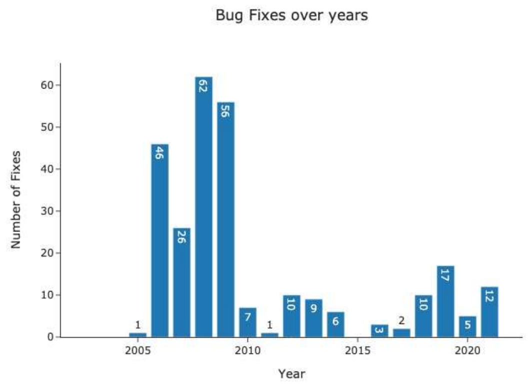


The above line graph shows the total number of test files that were created and modified each year. We can infer that the testers have put in their effort and created a lot of test files over the years which is why the coverage is more than 80%. More, number of test files need to be created so that the coverage can be increased. From the graph we can see that the test files in Numpy were created in the year 2002. There were 155 commits in the first year which is 2002. The tests added in this year was one of the least tests created by year. Since the year 2006, there was a significant increase in the number of test files that were added. 2013 was the year with highest number of commits related to the test files with a total of 1009 commits. Then in the year 2016 there was a significant decrease in the total number of commits with a total of just 388 which is way too low when compared to the year 2013. Then the year 2018 the total number of tests that were added were 892 which stands second with the total number of commits by year next to 2013.



The above bar chart shows the top five contributors according to the number of commits made by the contributors over the course of the Numpy project. We can see that Charles Harris tops the list with a total number of 1796 changes to the test files over the years. Sebastian Berg has a total number of 551 commits related to the testing which is the second maximum number of tests committed making him the top contributor next to Charles Harris. And Sebastian Berg's total number of commits is less than one-third of the Charles Harris' commits. Which mean majority of the tests were written by Charles Harris. Along with Charles Harris and Sebastian Berg, Eric Wieser are the members of the *NumPy Steering Council* and this council's role is to ensure, through working with and serving the broader NumPy community, the long-term well-being of the project, both technically and as a community. And finally, the top 5 contributors by number of commits in decreasing order are Charles Harris, Sebastian Berg, Eric Wieser, Travis Oliphant and finally Mark Wiebe. Travis Oliphant, the creator of Numpy, stood fourth in this list of total number of commits which really is impressive being creator as well as an active contributor to this project.





From the Issues Chart (left), we can see that the year 2017 tops the list with 208 issues raised. We can see that the frequency of bugs raised increased until 2017 and from then on, the bugs raised have been decreasing. We can infer that the developer's code quality until 2017 was poor (test cases prepared by the testers were successful in finding the issues) and we can also see that the developer's code quality has improved from 2018, hence, decrease in the number of bugs raised. Numpy has improved a lot in setting guidelines and there are various continuous integration (CI) services that are triggered after each commit which builds the code, run unit tests, measure code coverage, and check coding style of your branch. The CI tests must pass before the developers commit can be merged. If CI fails, contributors can find out why by clicking on the "failed" icon (red cross) and inspecting the build and test log.

From the Fixes Chart (right), we can see that the year 2008 tops the list with number fixes for that year being 62. There are still some open issues and thus there are less fixes shown above. We can see that there are a lot of fixes in between 2005 and 2010 as those were the initial days when Numpy was moved to the GitHub (and there might also be a case where a lot of bugs where fixed with few fixed). And there is this case where the version control system has really helped them to track the issues in a better way and fix them early, hence, less issues and fixes over the following years.

7. Conclusion:

As NumPy is an open-source project, there will be a greater number of contributors which means more risk as the community grows there are more developers contributing code to the project. As more developers contribute code and their solutions to problems might not be the best solution hence, leaving room for errors or bugs. Testers play an important role in the open-source projects like NumPy as they make sure that the code that is being developed needs to be working for its intended purpose and doesn't result in a problematic situation instead of providing a solution. NumPy has been improving its guidelines in terms of developer's contribution and increasing the number of tests for every code that is being committed. More number of tests should be written to improve the test coverage of the project (currently over 80%). These could be unit, integration, user interface, automation test. Sometimes also, there could be enough code coverage, but some edge cases for a feature might not be covered in those. Testers play a vital role in open-source projects and there needs to be more contribution from the testers would help to mitigate the risk of problematic code.

8. References:

[1] "Numpy documentation" - https://numpy.org/doc/stable/

[2] "Generating coverage report" - https://numpy.org/doc/stable/reference/testing.html

[3] "Mining software repositories" - https://pydriller.readthedocs.io/en/latest/

[4] "Creating tables, line and bar plots" - https://plotly.com/python/