Measuring the relationship between code quality metrics and community participation

Jeremy Boyd
The University of Texas at Austin
April 30, 2015

Introduction

Users and developers of open source software have long gathered in various places. For example, each project may have its own wiki where documentation and recipes are shared, issue tracker and mailing list where bugs and feature requests are discussed, or Internet Relay Chat (IRC) channel where assistance is requested and freely given. Many projects have all three of these. More recently, however, open source software developers and their users (many of whom are also software developers) have converged on Stack Overflow [13], a site for programming questions and answers. Stack Overflow is now considered a must-have tool for modern software developers; anecdotally, rarely does a day go by without visiting the site at least once.

But question-and-answer (Q&A) volume is a double-edged sword for maintainers of open source projects. On the one hand, it is gratifying to know that there are people using one's software in their own projects. On the other hand, increased question volume may indicate that the software is presenting problems to those developers. In this report, I attempt to address this second concern.

Below, I present a case study of five open source software projects written in the Python programming language. I approached this study with a simple hypothesis: that community engagement is *not* correlated with code quality metrics — e.g., lines of code (LOC), or cyclomatic complexity. I found, however, that for some projects there does exist a moderate, positive correlation between code quality and Stack Overflow activity, narrowly defined.

Data sources and collection

Stack Overflow activity

Questions on Stack Overflow may be "tagged" with meta-data. Notably, this meta-data generally includes a tag indicating the software package the question is about. The site provides a data explorer [14], which is effectively an interface to an instance of Microsoft's SQL Server product. Many tables are exposed, allowing researchers to investigate user activity, question/answer volume, etc. In previous work, I developed a series of parameterized queries [18][19] in the data explorer to answer questions about a particular tag. I was able to use these queries in the present report.

Due to implementation details of the database's underlying date-time representation, the queries resulted in quarterly breakdowns of the data in question. Thus, for each project, I gathered a list of all quarters for which relevant Q&A activity was present on Stack Overflow through the first quarter of 2015 (the last complete quarter for which data was available at time of writing). This provided a series of natural checkpoints to use when evaluating the projects' code quality.

Projects under evaluation

To determine which projects to study, I visited the search page [7] of a popular open source code-sharing site, GitHub, to determine the most popular projects written in Python. I picked the top five developer-facing projects, reasoning that user-facing projects would necessarily have low traction within the online developer community.

This process yielded the following selection of projects:

* **Django:** A web framework initially built by newsroom employees and therefore geared toward rapid development [8]. It is only a coincidence[1] that this project was the topic of previous study in this class.
* **Flask:** Another web framework, targeted toward simplicity [9]. This project fills a niche in the Python web ecosystem where Django is viewed as too large or unwieldy for the application being developed.
* **Requests:** A small library dedicated to making HTTP requests [10]. Its focus is on API simplicity, leading to its slogan, "HTTP for Humans." Requests has rapidly become the *de facto* standard for HTTP requests, supplanting the Python standard library's offerings.
* **Ansible:** A deployment and automation framework [10] that allows systems maintainers to deploy and update applications without installing agents on the systems under control.
* **Tornado:** Yet another web framework, originally built by engineers at Facebook [11]. Because of its event-loop based architecture, Tornado is often used in I/O-bound web applications that must maintain many open connections — e.g., for serving real-time data via Web sockets or as an API backend for mobile applications.

For most projects, I found the last commit on the first day of each quarter for which Stack Overflow data was available. Below is an example command for finding such a commit:

```
git checkout `git rev-list -1 --until="2008-07-01 00:00:00" master`
```

For two projects, their earliest code was committed *after* the start of the first quarter of Stack Overflow activity. For these, I picked the first commit relevant for code quality analysis. The list of evaluated commits for each project is available in the open source code accompanying this report [3].

Code quality metrics

In order to measure code quality, I used a code quality measurement tool built for Python projects called **radon** [1]. Although previous studies have used **PyMetrics** [2], it is badly out of date; its last release was in 2008. Radon leverages modern installation tools such as **pip** and has an easy-to-use command-line interface. It also easily provides output in structured formats such as JSON and XML.

---

[1] And not a very helpful coincidence at that. Because of methodological changes, I was unable to re-use any code quality measurements gathered during the initial project focused on Django.

Radon provides three types of code quality metrics:

* **Raw metrics:** Basic measurements of a codebase's size. These include lines of code (LOC), source lines of code (SLOC), logical lines of code (LLOC), comments, and multi-line strings (an idiosyncratic Python feature often used to comment code), and blank lines.
* **Cyclomatic complexity (CC):** A common measure of code quality first introduced by McCabe [4]. It is unbounded, but radon's authors indicate that a score greater than 20 for any block of code may be cause for concern. In my analysis, I will generally refer to mean file complexity (MFC) vs. block complexity.
* **Maintainability index (MI):** A metric first introduced in 1991 [5] that has seen some traction, most notably due to its inclusion in Microsoft's Visual Studio products and adaptation by the Software Engineering Institute. The metric's original formulation was as a function of cyclomatic complexity, Halstead Effort [6], and LOC. Radon uses a formula derived from both the Visual Studio and SEI variants. Values range from 0 to 100, with anything above 20 deemed acceptable.

Using radon, I gathered all three types of metrics for each quarterly checkout of the evaluated projects. Python's mechanism for denoting grouped modules of code into *packages* requires the presence of a file called __init__.py in a directory. Very often, these are empty. In order to minimize the effect of these empty files on the characterization of each codebase, I excluded all empty files from consideration when executing radon. The following snippet is a representative command used to execute radon's CC metrics, exclude any files, and route the resulting JSON output to a file:

```
radon cc -s -j --total-average -e "`find ./oss/django -type f -empty | tr '\n' ','`" ./oss/django > a75de3f.json
```

I prototyped my process using similar commands and developed a set of scripts available at [3] to automate the collection and summarization of this data.

Analyzing the data

To explore and analyze the dataset, I used iPython Notebook [15] in conjunction with pandas [16]. Graphs were drawn using matplotlib [17].

Characterizing the dataset

Overall codebase characteristics

In total, 97 code commits were collected and evaluated for this report. Over those 97 commits, 47170 files containing 6052837 LOC were measured for complexity. The mean number of files per commit is 486.29. The maximum cyclomatic complexity for any file was 679, and the mean file complexity (MFC) overall was 39.32. As we will see below, four of the projects hover in this range, with one outlier. Table 1 shows the overall summary statistics:

	# of files	LOC	MFC	Mean MI
Max	2031	269864	73.64	89.28
Min	1	48	1.0	59.19
Mean	486.29	62400.38	39.32	76.26
Median	83	18377	33.09	80.00

Table 1: Summary statistics over set of all commits evaluated, *n = 97*

Project codebase characteristics

Looking at each project individually, we can see some large differences between them. Table 2 shows some select summary statistics:

	Ansible	Django	Flask	Requests	Tornado
# of commits (=n)	10	27	20	17	23
Mean # of files	126.30	1552.74	61.25	62.82	73.48
Mean LOC	21742.7	181771.22	9730.35	12720.06	22466.91
MFC	38.83	29.71	32.48	32.44	61.83
Mean MI	72.11	84.51	77.12	81.40	63.84

Table 2: Summary statistics over all commits by project

Here we can see that the projects vary widely. Django is by far the largest project, with an order of magnitude more files and lines of code than any other project. It's also the oldest, having been around since the initial release of Stack Overflow. Yet, by both the mean file complexity or mean maintainability index measures, it is also the highest quality. Tornado, which has been present on Stack Overflow for nearly as long, ranks last in both MFC and mean MI. The combination of a relatively low mean number of files and a relatively high mean LOC may serve to explain this fact.

Figures 1 and 2 visualize the projects' quality metrics.

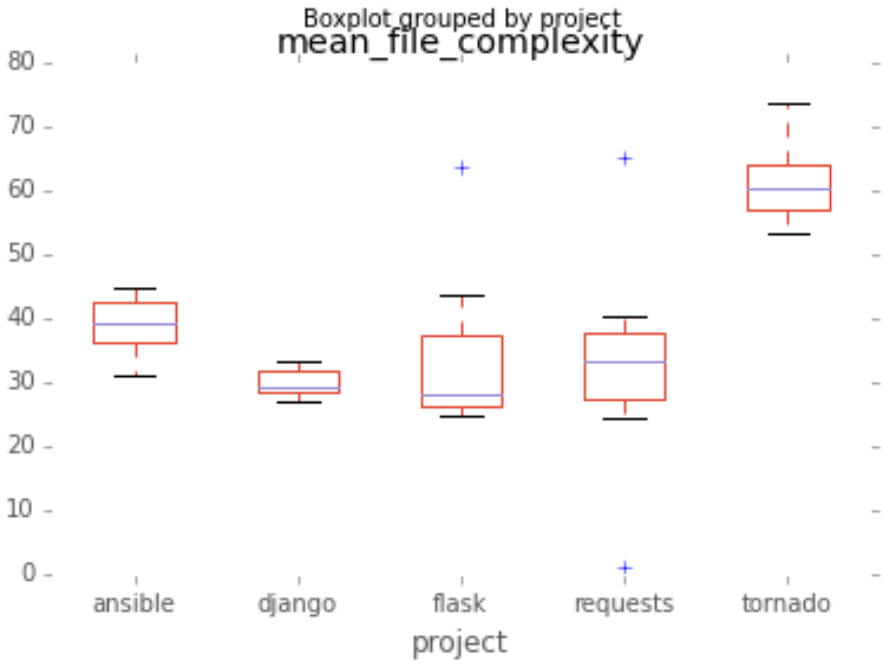

Figure 1: Mean file complexity by project

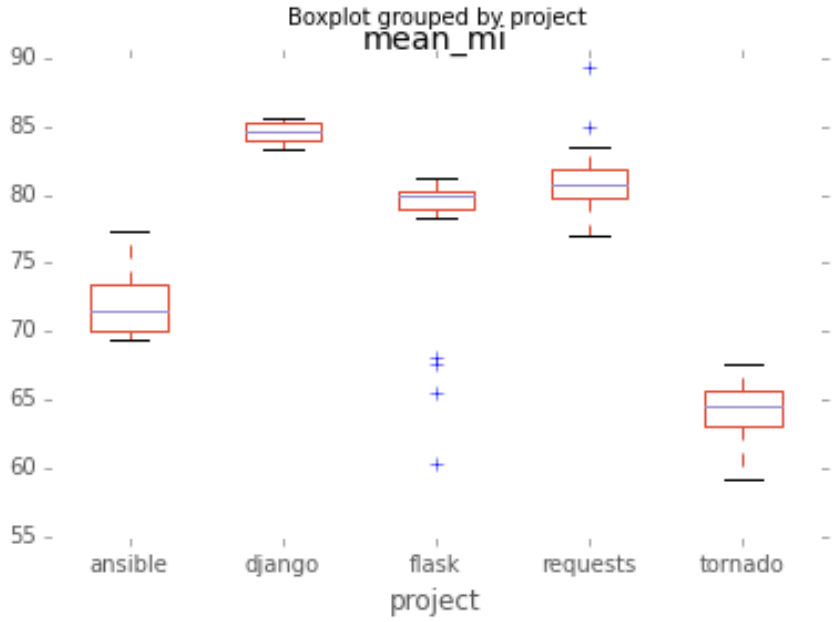

Figure 2: Mean MI by project

These figures provide a stark visualization of the relative lack of quality in the Ansible and Tornado projects.

Stack Overflow activity

As Figure 3 shows, Stack Overflow activity for each project is generally on the rise ('total_posts' is the series of all questions asked).

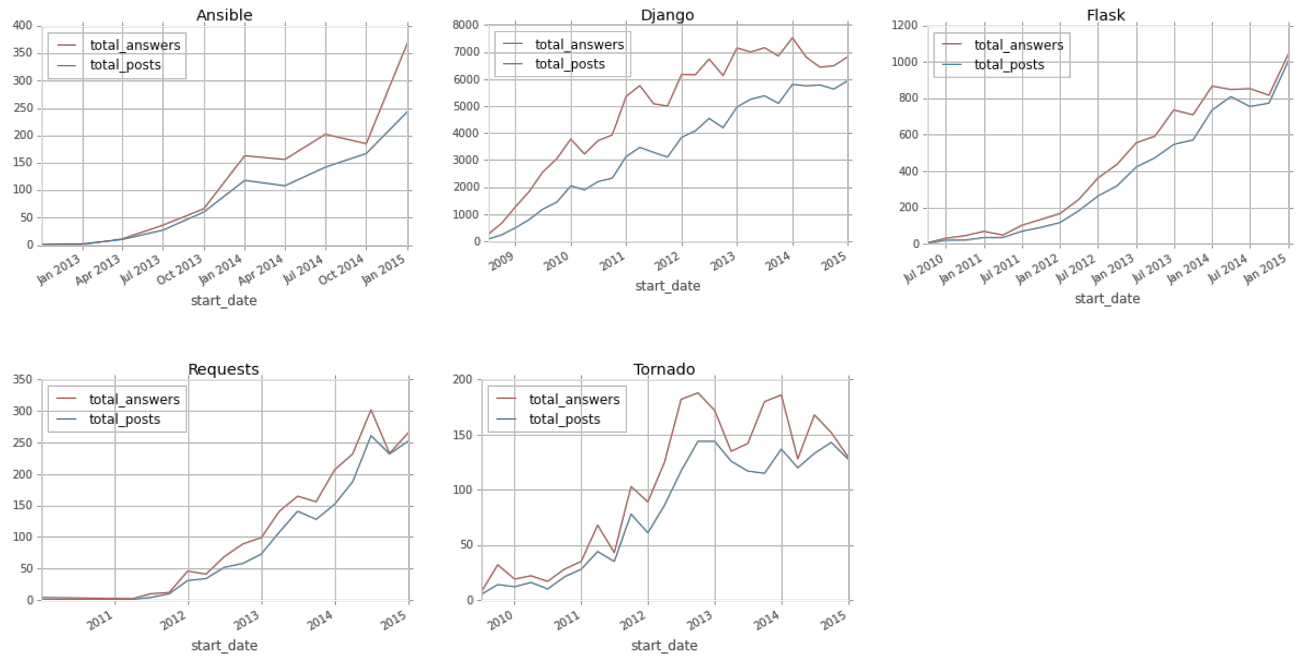

Figure 3: Stack Overflow questions and answers by project

For each project, the volume of answers generally outweighs the volume of questions, which is to be expected and may be a sign of quality engagement within the community. However, for our analysis this data is insufficient: the increase in question/answer volume may simply coincide with a larger set of users for either the project or for Stack Overflow itself. To try to mitigate the effect of date and therefore user base growth, I evaluated another piece of data.

Stack Overflow allows users to mark an answer to their question as "Accepted," meaning generally that the selected answer solves the questioner's problem. When we look at the percentage of overall questions with accepted answers, the picture is quite different. Figure 4[2] shows the trend for each project.

For most projects, there is a clear downward trend. The cause for this is unclear. One hypothesis, which I favor, is that all the "good" or "easy" questions have already been

[2] In all graphs that follow, 'answered_ratio' is the series representing the percentage of questions with accepted answers, which I also call the percentage of answered questions or PAQ. 'answered_ratio' is simply an artifact of the column names used within pandas for my analysis and graphing.

answered. Under this hypothesis, many of the most common questions developers have were asked early in a project's time on Stack Overflow. These were readily answered by the Stack Overflow community. As time progresses, these already-answered questions need not be asked by successive developers, since the answers were already available. Thus the difficulty of new questions became greater. For example, they might center around edge cases, new features, or defects not present in other environments. Testing this hypothesis is not easy, however, and beyond the scope of this work.

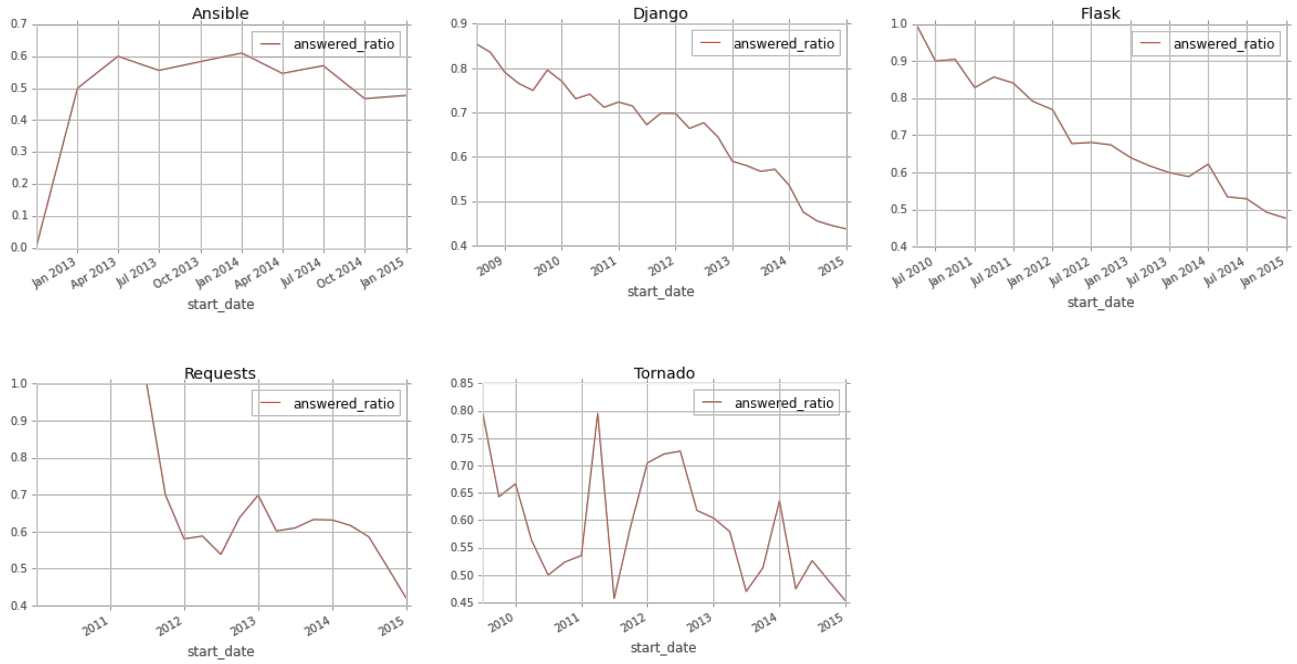

Figure 4: Percentage of questions with an accepted answer over time

In addition to this trend, some projects demonstrate other similarities with respect to their percentage of answered questions (PAQ), as can be seen in Table 3, which summarizes this metric for each project, and Figure 5, which visualizes PAA for each project as a boxplot.

	Ansible	Django	Flask	Requests	Tornado
# of commits (=n)	10	27	20	17	23
Min PAQ	0	43.78	47.60	42.06	45.31
Max PAQ	61.01	85.50	100.00	100.00	80.00
Mean PAQ	49.10	66.30	70.11	66.75	59.09
Median PAQ	55.09	69.81	67.58	61.70	57.93

Table 3: Summary of PAQ by project

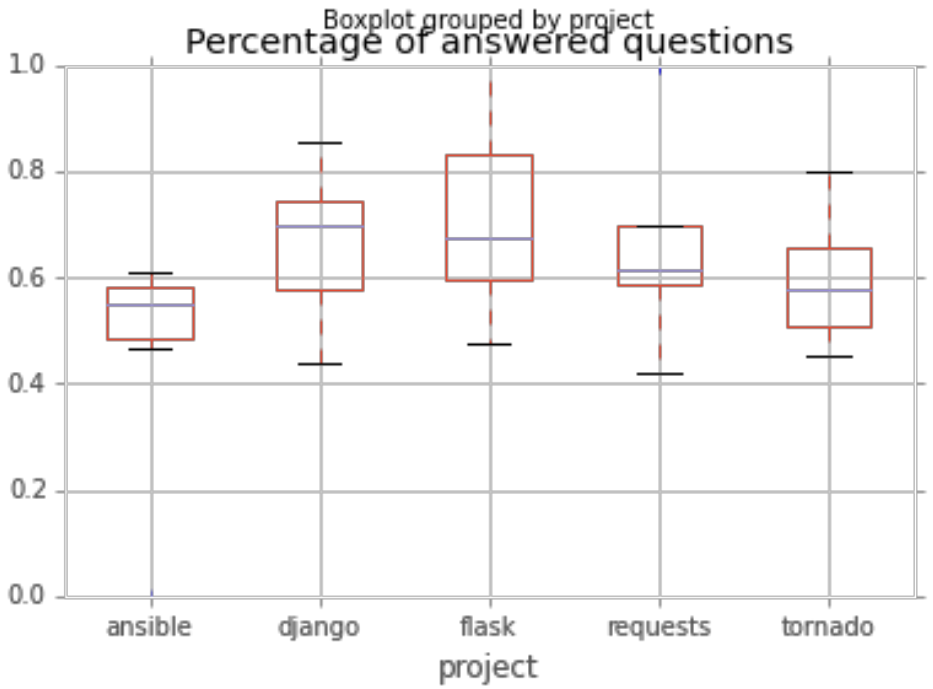

Figure 5: Boxplot of PAQ by project

Django, Flask, and Requests all have a mean PAQ in the high 60s, reflecting perhaps their popularity in their respective spheres. The relatively low mean and median of Ansible may be due to low sample size. It is also possible that users of Ansible congregate on Server Fault, Stack Overflow's sister-site dedicated to server administration questions. Tornado does not fare as badly as Ansible, but its median PAQ is only marginally better than Ansible's, and its mean PAQ is significantly lower than that of any of the top three projects.

Measuring the relationship between quality and PAQ

We turn now to our original question: what, if any, relationship exists between code quality metrics and Stack Overflow activity for these projects? The answer, unfortunately, varies. Figure 6 shows measurements of MFC, mean MI, and PAQ for each project over time.

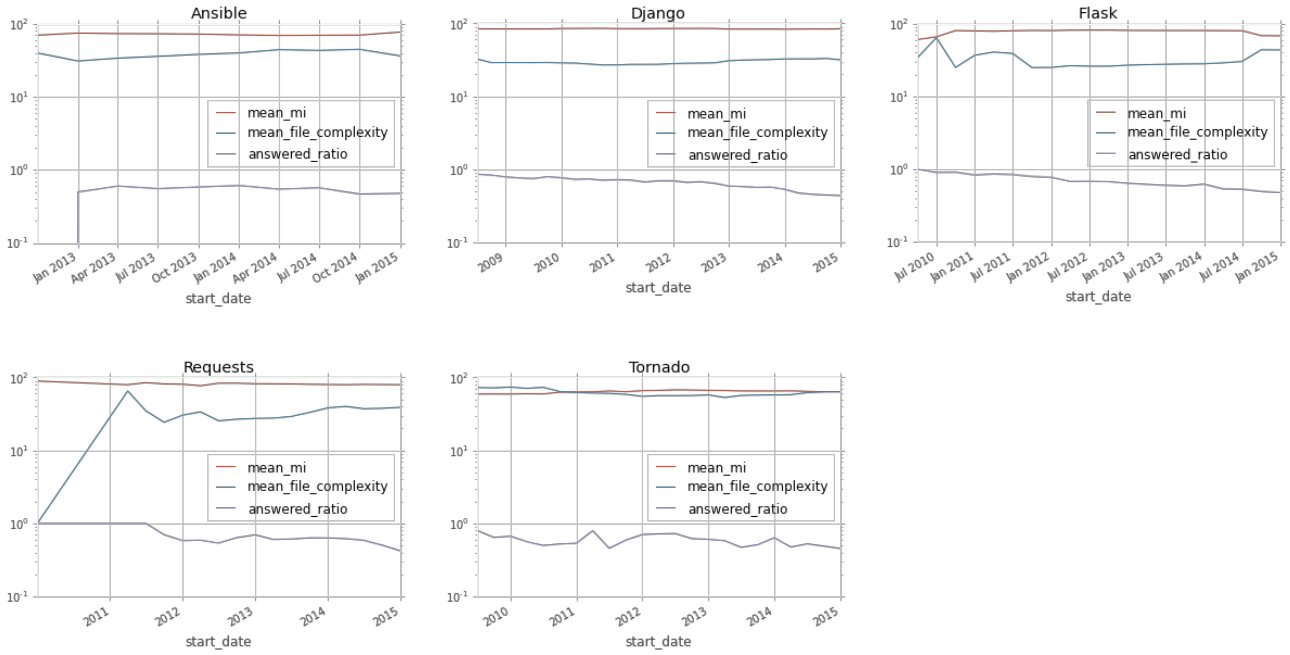

Figure 6: Code quality metrics vs. PAQ ('answered_ratio') over time

The downward trends in PAQ discussed in the previous section are clearly visible in every project except Ansible, which nevertheless shows a slight downward trend. Still, as we can see there is little visible relationship between the quality metrics and PAQ. This is borne out by the correlation coefficients of each project.

In order to interpret those coefficients correctly, the relationship between mean file complexity (MFC) and mean maintainability index (MI) must be kept in mind. As Table 4 shows, these have a strong negative correlation:

	Cor(MFC, mean MI)
Ansible	-0.7975
Django	-0.7631
Flask	-0.6897
Requests	-0.7107
Tornado	-0.9592
Overall	**-0.8867**

Table 4: Relationship between MFC and mean MI

Because of the construction of the maintainability index, this is an intuitive result. However, it may be useful to bear in mind as we consider Tables 5 and 6, which show a decidedly mixed the relationship between these metrics and the percentage of answered questions (PAQ).

	Cor(MFC, PAQ)
Ansible	-0.0991
Django	-0.6490
Flask	0.1949
Requests	-0.0302
Tornado	0.0487
Overall	**-0.1936**

Table 5: Relationship between MFC and PAQ

	Cor(mean MI, PAQ)
Ansible	0.1795
Django	0.4361
Flask	-0.2238
Requests	0.5584
Tornado	-0.0728
Overall	**0.2418**

Table 6: Relationship between mean MI and PAQ

The results are, admittedly, confusing. For Ansible and Django, as MFC increases (and quality ostensibly decreases), PAQ decreases. For Flask, the opposite is true: lower quality is correlated with higher PAQ. The results for Requests are wildly different, and the coefficients for Tornado are so small as to be ignored.

Django represents the most interesting case here, as for it code quality has a moderate, positive correlation with PAQ under both metrics. Figure 7 shows the corresponding scatter plots, which seem to validate this interpretation.

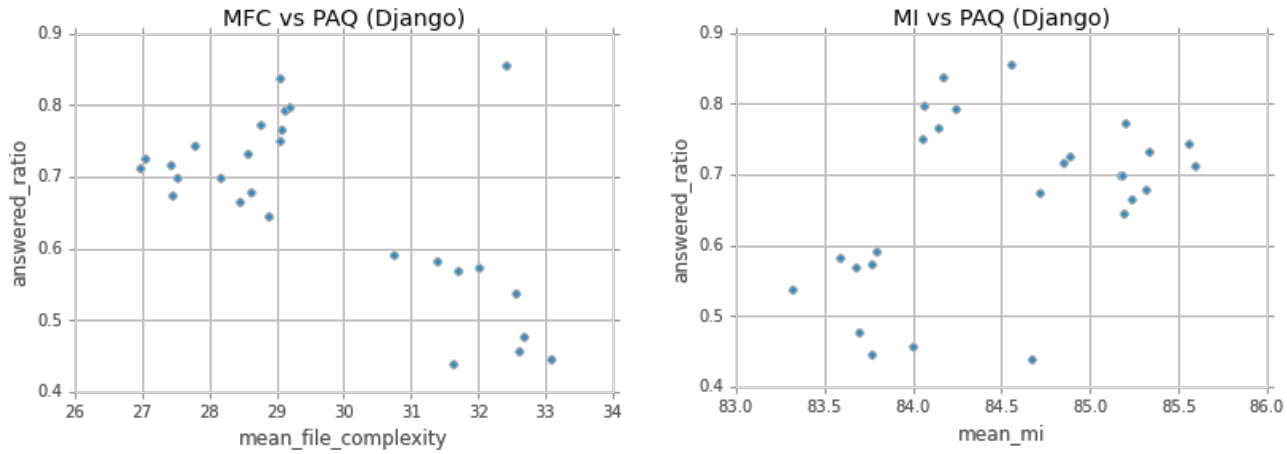

Figure 7: Scatterplots of MFC and MI vs PAQ (Django)

The confusion in these results is reflected in the overall coefficients. If any relationship exists between code quality and Stack Overflow, it is a weak, positive one: increased MFC is negatively correlated with PAQ, while increased mean MI is positively correlated with PAQ. Figure 8 visualizes the weakness of this relationship.

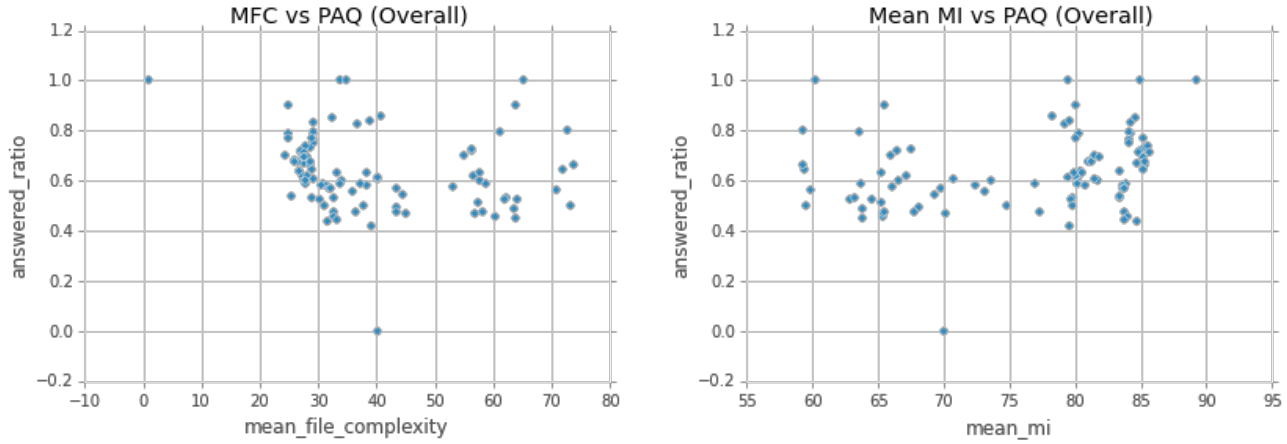

Figure 8: Scatterplots of MFC and MI vs. PAQ (Overall)

Unfortunately, excluding Django from these calculations does little to clarify the picture.

Conclusion

In this report I have evaluated the code quality of five popular open source software projects written in the Python language. I found that four of them -- Ansible, Django, Flask, and Requests -- were of decent quality, while the fifth, Tornado, scored far more poorly on the chosen metrics. I discussed the relative volume of Stack Overflow activity for each project, and noted a downward trend in the percentage of questions asked on the site with accepted answers. Finally, I evaluated what, if any, relationship exists between code quality metrics

and the percentage of questions with accepted answers. I found mixed results, with only one project exhibiting a consistent, moderate correlation between these two properties. I also found that these mixed results were reflected in measurements of the overall relationship between quality and Stack Overflow activity.

References

[1] http://radon.readthedocs.org/en/latest/index.html
[2] https://github.com/ipmb/PyMetrics
[3] https://github.com/boydjj/code_quality_and_stack_overflow
[4] McCabe, Thomas J. "A complexity measure." Software Engineering, IEEE Transactions on 4 (1976): 308-320.
[5] Oman, P., J. Hagemeister, and D. Ash. "A definition and taxonomy for software maintainability." *Moscow, ID, USA, Tech. Rep* (1992): 91-08.
[6] Halstead, Maurice H. "Elements of software science." (1977).
[7] https://github.com/search?l=python&q=stars%3A%3E1&s=stars&type=Repositories
[8] https://www.djangoproject.com/
[9] http://flask.pocoo.org/
[10] http://docs.python-requests.org/en/latest/
[11] http://www.ansible.com/home
[12] http://www.tornadoweb.org/en/stable/
[13] http://stackoverflow.com/
[14] http://data.stackexchange.com/
[15] http://ipython.org/notebook.html
[16] http://pandas.pydata.org/
[17] http://matplotlib.org/
[18] http://data.stackexchange.com/stackoverflow/query/296434/get-quarterly-breakdown-of-answered-unanswered-questions-for-a-tag
[19] http://data.stackexchange.com/stackoverflow/query/296447/quarterly-number-of-answers-for-a-tag