Aditya Kelekar

Towards Better Open-Source Development:

Improving PyQtGraph’s Feature-Development Process

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme

Thesis

3 April 2018

Author Title Number of Pages	Aditya Kelekar Towards Better Open-Source Development: Improving PyQGraph's Feature-Development Process 27 pages + 2 appendices
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Kimmo Saurén. Senior Lecturer

PyQtGraph library is an open-source, pure-python graphics library. The popularity of open-source software, such as PyQtGraph, depends to a large extent on the utility of its features. The objective of this thesis is to present guidelines for the feature development process of the PyQtGraph library.

The PyQtGraph library's software development process consists of a user group that proposes and develops features through forum interactions and by writing new code.

As the developer community, including the maintainer of the PyQtGraph library, works on a voluntary basis, users might not have the time and commitment to work on a feature for an extended period of time. In order to overcome this constraint, the code development process of the PyQtGraph project should be optimized to facilitate easy collaboration between users.

For the purpose of studying the code development process of the PyQtGraph library, the library's forum posts were analysed. Different types of forum discussions were studied: those leading to new features as well as those that did not generate any new feature.

Based on an analysis of the project's forum posts and the corresponding changes to the Github repository, guidelines for the development of a tool that would aid collaboration between users during the feature development process were created.

This thesis could be useful for a developer contributing to the PyQtGraph project, as also to the maintainer of the PyQtGraph project to review the code development process.

Keywords	Feature development, open-source, PyQtGraph, Qt

Contents

List of Abbreviations

GUI	Graphical User Interface
API	Application Programming Interface

1. Introduction

The popularity of most open-source software depends on the utility of the features that a software provides to its users. So too is the case with PyQtGraph library, a pure-python graphics and GUI library built on PyQt4 / PySide and NumPy. Thus, in the dynamic world of software applications, the developers of any popular library should be able to recognize the need for new features and improving existing ones.

Of course, besides features, there are other factors that a prospective user of a software seeks as well, such as the stability of the code and compatibility of the software with the other usage goals of the user. But finding the right features is often the first criteria for most users.

The PyQtGraph library's software development process consists of a user group that proposes and develops features through interactions on the Library's forum pages and by modifying the existing code on the library's Github page. The software development process is visible for all users and also open for everyone to contribute.

The software development process at PyQtGraph library has one main problem commonly seen in open-source projects: lack of resources. As the entire developer community, including the maintainer of the library, works on a voluntary basis, users might not have the time and commitment to work on a feature for an extended period of time. This means that the interaction channels between the users of the PyQtGraph library software development project should be optimized for a software development process that supports easy collaboration between users.

The objective of this thesis is to present some guidelines for the software development process of the PyQtGraph library, in particular, guidelines related to interaction methods between the developers of the library. An additional objective of this thesis is to research the main challenges the developer community of PyQtGraph faces.

PyQtGraph’s strengths and weaknesses are discussed in Chapter 2: PyQtGraph: A GUI Tool for Real-Time Applications. Discussions related to improvements for features are the subject of Chapter 3: New Feature Development Process. Based on the outcomes of the different individual cases, guidelines for the feature development process have been created, which are seen in Chapter 4: Improving the New Feature Development Process.

A discussion on whether or not the objectives of this thesis were realized forms the basis of Chapter 5: Conclusions. The chapter also looks at the possibility of extending the results of this study to open-source software development in general. Another matter for discussion pertains to the kind of further studies that could profit from this current study and what could be a probable topic for another thesis that would take the findings of the current one in use.

2. PyQtGraph: A GUI Tool for Real-Time Applications

This chapter examines in brief the PyQtGraph library tool, its main strengths and weaknesses, and how it compares with the other important tools in the domain in which it is used.

This comparison with other tools is done on the basis of a collection of user reviews from the Library's forum pages. The objective is to provide a balanced view and to that end those user comments are discussed where the users appear to have some degree of experience using PyQtGraph. If counter-arguments have been made, these too are presented.

By thus presenting an overview of PyQtGraph's utility, it is hoped that the discussion on the Library's code development process is provided with an appropriate context. The reasons for feature development should then make more sense. The reader would also have an idea as to what features of PyQtGraph have been well regarded within the user community and what kinds of feature developments have been expected.

2.1 PyQtGraph's Dependencies

According to the PyQtGraph's home page http://www.pyqtgraph.org/, the PyQtGraph library has been built on PyQt4/PySide and NumPy. [1] This fact is also obvious when one follows the instructions for installing PyQtGraph as per PyQtGraph's official documentation; one of the steps listed mentions downloading and installing the following libraries [2]:

- Python 2.7 or Python 3.x
- A Qt library such as PyQt4.8+, PyQt5, or PySide
- NumPy

Python-OpenGL bindings are required for running 3D graphics applications.

Luke Campagnola, PyQtGraph's maintainer, states that the library derives its high speed of compilation by harnessing the power of NumPy for computations and using Qt's GraphicsView framework for the display part [1].

The Qt components are available to PyQtGraph through PyQt or PySide, both of which provide similar functionalities [3].

According to Riverbank Computing, the software company that owns and develops PyQt, "PyQt is a binding, that is, a set of libraries that make the C++/Qt application development framework's libraries accessible to Python programmers" [4, p6]. PySide is another open source software project, with a similar objective to that of PyQt: that is, to provide Python bindings for the Qt framework [5].

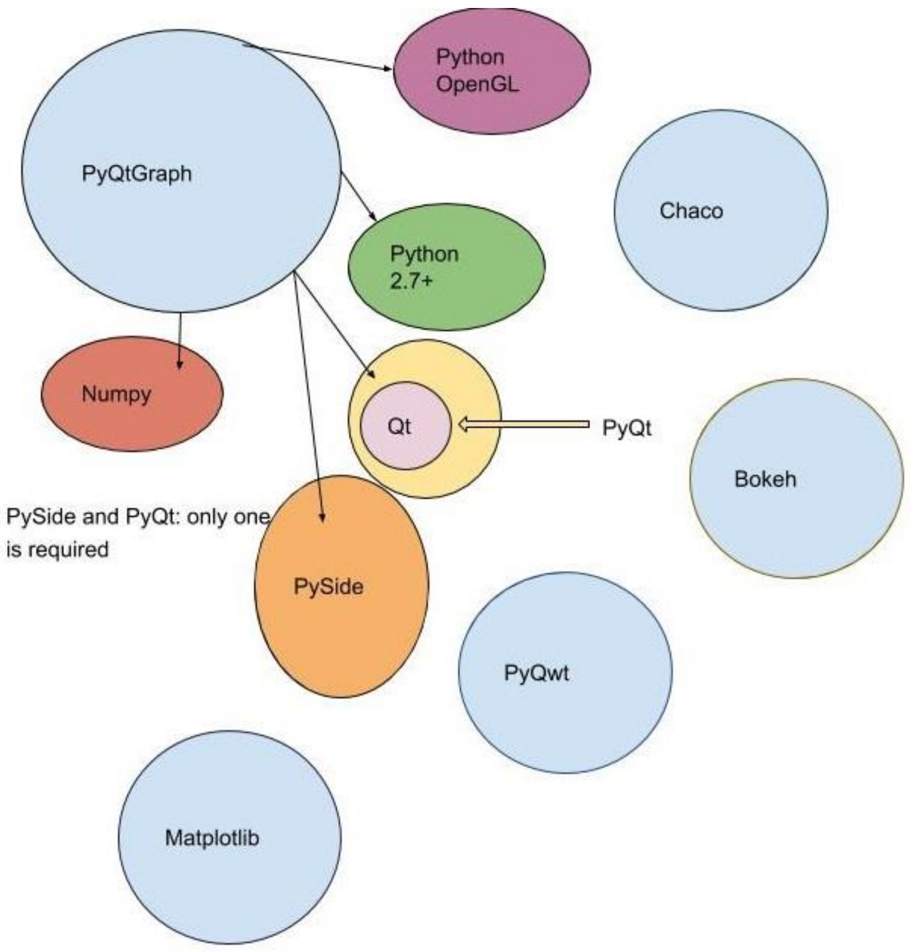


Figure 1: Diagram showing PyQtGraph's dependencies and the other major graphics libraries based on Python. The dependencies of the other libraries are not shown.
NOTE: The relative sizes of the figures used are not an indicator of any metric.

Qt itself is a cross-platform software framework with ready-made UI elements, C++ libraries, and an integrated development environment. It is currently being developed both by The Qt Company, a publicly listed company, and the Qt Project, under open-source governance, involving individual developers and firms working to advance Qt. [6.]

Qt comes with a collection of widgets such as calendars, tables, tree views, and also web browser widgets. Besides, Qt offers support for creating custom widgets [4, p5]. The widgets commonly used in PyQtGraph applications are borrowed from the Qt framework. Figure 1 illustrates the other software parts needed for running PyQtGraph applications; these are also called “dependencies”. The figure also shows the other major graphics libraries based on Python.

Though Qt is known to be a mature platform [4, p5] and has been a key component of PyQtGraph, there have been other changes in the graphics landscape. One of them is the rise in importance of the OpenGL specification. This specification, which is considered the graphics hardware industry's foundation for high performance graphics, is a 2D and 3D graphics API that is operating-system independent [7].

The OpenGL development is likely to affect PyQtGraph's future growth plans too. PyQtGraph's maintainer Campagnola has declared that he plans to use "VisPy", a visualization library based on OpenGL, to replace Qt as the rendering engine for PyQtGraph's 2D graphics. VisPy, which is currently under development, will focus on visualization but it will not have the GUI toolkit features that are synonymous with PyQtGraph. The library is currently being developed by a set of authors, which includes Campagnola, most of whom have previously worked on other GUI software. [1.]

2.2 PyQtGraph’s Real-time Strength Trumps

The PyQtGraph library is primarily seen as a tool for data plotting. According to Luke Luke Campagnola, maintainer of the PyQtGraph project, Matplotlib is generally regarded as the de-facto Python tool for plotting [1]. Campagnola also mentions that, however, the PyQtGraph library has some distinct advantages in some areas such as real-time plotting. The library also comes with a set of examples that provide an idea about the library’s capabilities.

Campagnola chooses to recommend Matplotlib for most plotting applications. “If you are starting a new project and do not need any of the features specifically provided by PyQtGraph, you should start with Matplotlib,” he says on the PyQtGraph project page. In his opinion, Matplotlib is much more mature, has an enormous user community, and produces useful publication-quality graphics. [1.]

Discussions on user forums show that there are some applications where PyQtGraph performs better than Matplotlib, and one of them is indeed real-time data plotting. In a discussion on Stackoverflow, an online community for developers to share their programming knowledge, a user “gorgoth” with a Stackoverflow reputation of 73 (the reputation score is seen at the side of the user’s name on the provided reference number, 2), who had extensively used both Matplotlib and PyQtGraph claimed that for any sort of fast or 'real-time' plotting, he would “strongly recommend PyQtGraph.” [8.]

The preference for PyQtGraph over Matplotlib in real-time plotting has also been mentioned in the results of experiments conducted by Scott Harden, an electronic hobbyist. In his online journal, Harden reports that while plotting a sine wave from a sound card he observed that the speed (frame rate) using PyQtGraph’s PlotWidget was 20 times more than that obtained by using Matplotlib’s MatplotlibWidget. [9.]

Programmers who have used Matplotlib for similar real-time applications have cribbed on the slowness of Matplotlib’s performance. In a Stackoverflow discussion titled “Why is plotting with Matplotlib so slow?”, the general consensus seems to be that Matplotlib is not optimized for real-time use. [10.]

Campagnola mentions two other attributes of PyQtGraph, which, in his opinion, make it advantageous to use PyQtGraph in specific applications: one is portability and the other is the benefits to be had from the science/engineering features of PyQtGraph. [1.]

According to Campagnola, PyQtGraph has been developed as a pure-python package, which means that it runs on virtually every platform supported by NumPy and PyQt.

Besides, being pure-python, no compiling is required. “If you require portability in your application, this can make your life a lot easier,” Campagnola says. [1.]

The other aspect of PyQtGraph that Campagnola points out is that PyQtGraph has many other features besides plotting. Many scientific and engineering applications have specialized plotting requirements. PyQtGraph’s advanced features like its ImageView and ScatterPlotWidget analysis tools address these plotting requirements. Other PyQtGraph-specific plotting accessories include ROI-based data slicing, parameter trees, flowcharts, and multiprocessing. [1.]

2.3 Ease of Installation and Deployment

Users contemplating PyQtGraph for plotting some graphics may have a few other factors in mind: for instance, ease of deployment of a software library, ease of use and availability of support and documentation.

The same user, gorgoth, mentioned earlier, also supported other comments that installation of PyQtGraph is trivial. In his experience, the tool displays and performs on both Windows and Linux roughly equivalently (minus Window manager differences). [8.]

However, PyQtGraph is not the tool of choice for Web-based applications. In the context of development of a graphical tool for a multi-user, disparate-systems scenario, the deployment of a graphics application on many systems is simplified by using a Web-based graphics library. The PyQtGraph application is not tailor-made for Web applications and so Web implementations are not straight-forward. One such Stackoverflow discussion on displaying PyQtGraph and PyQt widgets on web remarks on the unsuitability of embedding PyQtGraph in Web applications [11.]

Thus, applications requiring collaboration, especially by new and remote users, PyQtGraph's non-Web interface may prove to be a deterrent.

One of the Web-based graphic libraries based on Python is Bokeh. Its goals are to make it provide developers with a tool to plot data from remote, possibly large data sets and streaming data. Bokeh applications publish to the Web, so this makes it easier for other users to explore and interact. [12.]

2.4 PyQtGraph Documentation

PyQtGraph's limited official documentation has been criticized by some reviewers.

The user mentioned earlier (gorgoth) says that while Web documentation for PyQtGraph is admittedly less than desirable, the source code is well commented and easy to read. These factors, in his opinion, outweigh the lack of documentation. “Couple that [well-commented source code] with well-documented and diverse set of demo code and in my experience it far surpasses Matplotlib in both ease of use,” he says. [8.]

However, the official documentation is not the only recourse available while trying to build applications that use PyQtGraph. The user discussion boards supplement the documentation in many cases. The popular discussion boards are:

- PyQtGraph GoogleGroups https://groups.google.com/forum/#!forum/pyqtgraph,
- PyQtGraph GitHub issue tracker, https://github.com/pyqtgraph/pyqtgraph/issues,
- StackOverflow discussion pages with the appropriate (PyQtGraph) tag, https://stackoverflow.com/questions/tagged/pyqtgraph

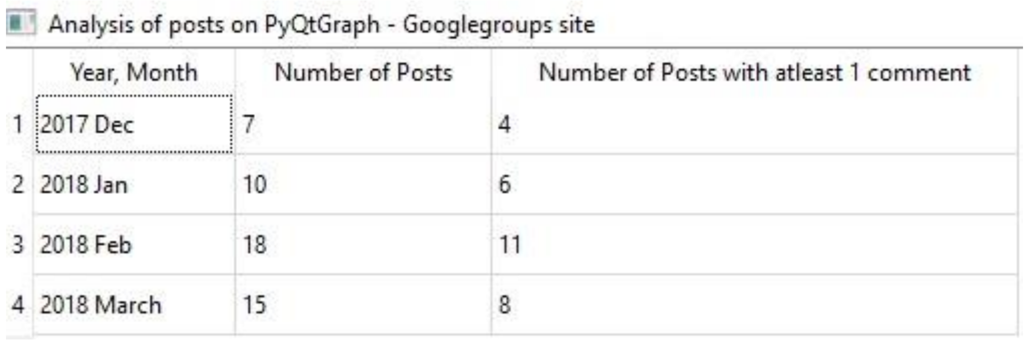
Reading forums posts on the above-mentioned forums, it is seen that PyQtGraph has found traction with various kinds of users, often even with those who have little experience to Python. These users have found assistance to varying degrees from the PyQtGraph developer and user community.

An analysis of posts in the past four months on the PyQtGraph Google Groups site shows that 29 (58%) of the 50 forum posts by users received comments from other users [13]. Comparing a similar number of forum posts on Bokeh’s Google Groups site (50 posts, corresponding to the time period from March 20 – April 11 2018), it was seen that 80 % of the posts received comments from other users [14]. Thus, the PyQtGraph Google

Groups site user queries have a poorer response rate than those on Bokeh’s Google Groups site.

As the Google Groups site does not indicate whether or not the author was satisfied with the support received, no inference can be made regarding the percentage of cases in which the issue was resolved.

Figure 2 is a screenshot of a Python program, written for the purpose of this thesis, that creates a table to show the results of analyzing posts on PyQtGraph’s Google Groups site. The number of posts for each month and the number of posts that received at least one comment were counted manually from the Google Groups site. The program uses the PyQtGraph library to create the table.



Analysis of posts on PyQtGraph - Googlegroups site

	Year, Month	Number of Posts	Number of Posts with atleast 1 comment
1	2017 Dec	7	4
2	2018 Jan	10	6
3	2018 Feb	18	11
4	2018 March	15	8

Figure 2: Analysis of posts on PyQtGraph Google Groups site

3. New Feature Development Process

In this chapter, the new feature development process for PyQtGraph is discussed. The feature development and bug-tracking process as observed in open-source software development are also discussed in general. This serves to compare and contrast the development process peculiarities of the PyQtGraph case with that of the general open-source software model.

To lay the foundation for the discussion on the feature development process, the importance of feature development in open-source software is first discussed.

3.1 Linking Open Source Success to Feature Development

The success of a paid software product is often measured in terms of its market share, which in turn, is determined by the sales revenue of the product. However, open-source software is characterized by the absence of revenue, and, as such, using the market share yardstick to measure success is not possible.

Gaby Fachler, open-source community manager for the CloudSlang project, makes a case for using the so-called "Pirate Metrics" for measuring the success of open-source software projects [15].

The "Pirate Metrics" developed by Dave McClure in 2007 were originally intended to refer to the five most crucial areas for a startup to focus on [16]. These five areas were:

- Acquisition: Optimize Your Acquisition
- Activation: Speed Up Your Activation
- Retention: Retention Is King
- Referral: Build A Referral Machine
- Revenue: Create More Revenue

According to Fachler, these metrics when applied to open-source software projects, can help serve as an effective framework to rate a project's success at each significant level of user engagement. In the model that he constructs, an attractive blog post's ability to generate unique site visits to a particular open-source project's page is regarded as the "acquisition" step, and the potential for that to lead to product downloads to "activation". If a user opens new issues, meaning creates a post about a possible improvement for an existing feature or suggests a new feature, that initiative can be the "retention" step, while starring the product's Github page is a type of "referral". Finally, the last metric of "revenue" in this context is not money, but the action of a user to contribute code back to the project.

Fachler's use of Pirate Metrics for measuring the success of open-source project PyQt-Graph is depicted in figure 3.

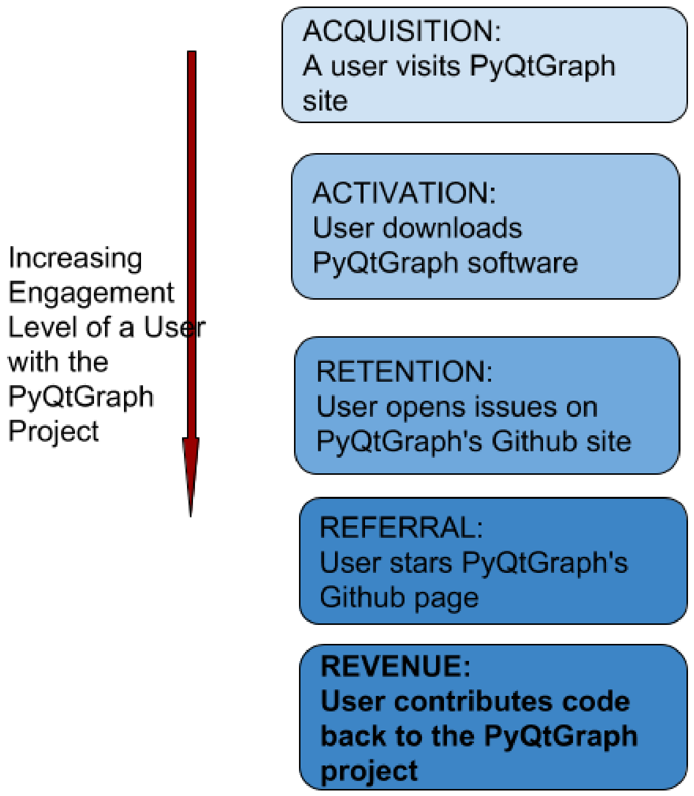

ACQUISITION:
A user visits PyQtGraph site
ACTIVATION:
User downloads PyQtGraph software
RETENTION:
User opens issues on PyQtGraph's Github site
REFERRAL:
User stars PyQtGraph's Github page
REVENUE:
User contributes code back to the PyQtGraph project
Increasing Engagement Level of a User with the PyQtGraph Project

Figure 3 Measuring success of PyQtGraph project using Pirate Metrics [15]

3.2 The Feature Development Process

As in other open-source software projects, addition of new features to the existing PyQt-Graph code happens through a process that requires collaboration across players and is iterative by nature. In the whitepaper *Understanding the Open-source Development Model*, Ibrahim Haddad and Brian Warner note that it is generally a subset of developers -- ones that have taken ownership for delivering a feature -- that take the feature-development process to completion [17].

The open-source development process emphasizes peer review throughout the entire development life-cycle. Developers are expected to submit their code to project mailing lists for periodic public peer review. The code sharing is particularly crucial when a feature achieves a development milestone. Users outside of the development team are thus aware of the changes, and can influence the outcome of the discussion on whether or not changes should be incorporated into the existing code repository.

Feature requests in open-source projects may begin with a proposal to a mailing list, a discussion on IRC, or as a feature request in a project's bugzilla. The request is typically made by whoever is likely to lead the implementation work.

As PyQtGraph is a smaller project compared to other open-source projects such as Mozzilla and Libre Office, the team of developers is smaller and the development process is simpler. The current maintainer of the library, Luke Campagnola, who is also the founder of this open-source project, has the administrative rights to accept pull requests to the source code, which is stored on Github and distributed under the MIT open-source license [1].

There are two distinct procedure possibilities in the PyQtGraph development process for a feature request or a (potential) feature enhancement to see the light of the day.

In the first case, a user initiates the process by posting a request or complaining about a specific problem on the group's forum pages; these pages being the PyQtGraph's Google Group page, posts on StackOverflow or on PyQtGraph's Github page. In this case, the maintainer of the PyQtGraph library and other users comment on the request, and some developer (hopefully) with the required skills volunteers to patch the issue.

The other possibility for new code development is for a user to take up one of the issues documented in the “to-do" list and offer a solution. This list, which is easy to read and comment on, is maintained by Luke Campagnola as a google document on the group’s Github page under the Wiki Section [18].

3.3 To Accept or Not to Accept

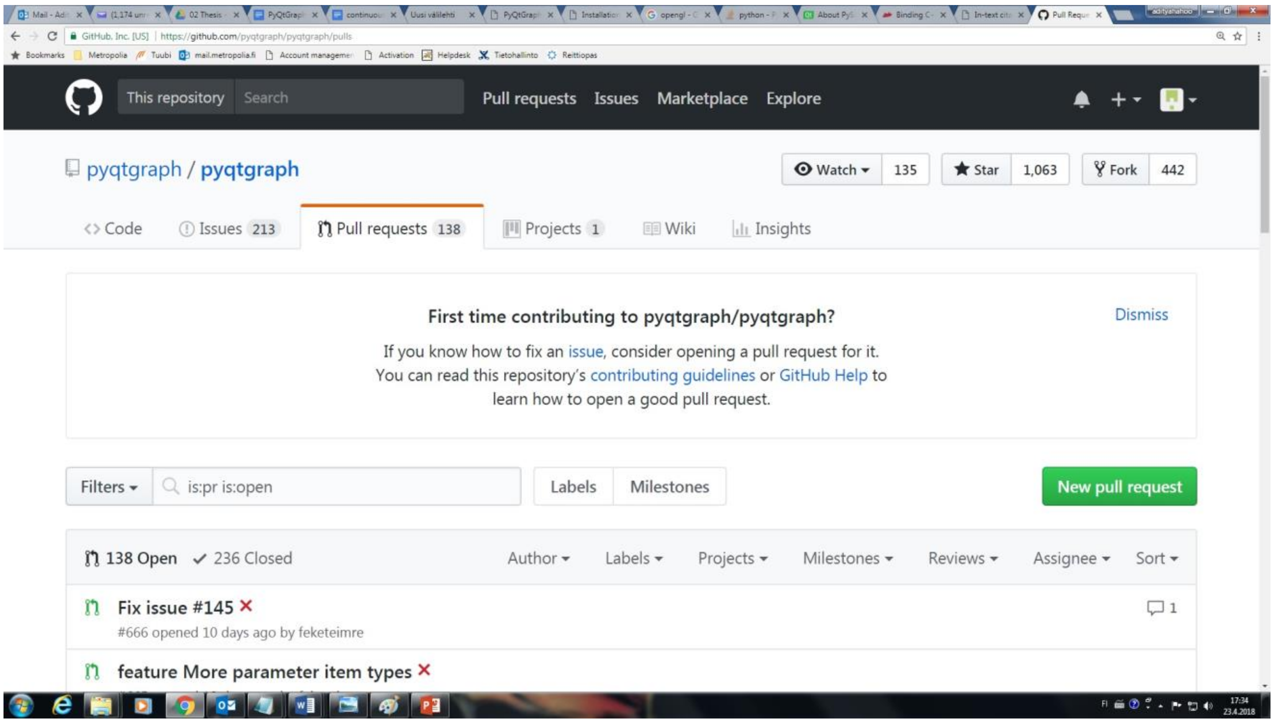
How do open-source code projects maintainers decide what feature requests in terms of pull requests to accept and what to reject? Felix-Krause, creator of fastlane, an open-source tool for publishing apps on the App Store, discusses in his article *Scaling Open-source Communities* his observations on his project's progress over a period of time as his project base grew from a handful to over a thousand and he began receiving many feature requests from users [19].

Krause had to spell out his vision for the project and accept requests that were in agreement with this vision, while denying requests that did not comply. By doing so, the senders of the requests also had clarity about the decision on their pull request.

As a warning, Krause [19] also mentions that in the case that a code maintainer decides to merge a pull request with the master code (in other words, accept the pull request), then it is the responsibility of the maintainer to attend to any consequences arising from the new code. For instance, if the feature breaks in the future, it is the maintainer who will have to fix it and that take may take as much time or more as reviewing the code. Therein lies the importance of reviewing a feature pull request thoroughly before accepting it.

Before proceeding to the feature development process in PyQtGraph, a quick review of the pull request process for the PyQtGraph project is in order. The PyQtGraph project is hosted on Github. For a person to develop a feature or suggest a new one, the usual procedure is that a user first downloads a copy of the project code onto his/her machine, modifies the local copy of the code and then places a “pull” request on the Github project page. Other users can then review the code and comment on its benefits but it is up to the maintainer to accept the pull request or not.

Figure 4 is a snapshot of the PyQtGraph project's Github page showing the status of current pull requests. The button for a new pull request is on the same page.


https://github.com/pyqtgraph/pyqtgraph/pulls

This repository | Search | Pull requests | Issues | Marketplace | Explore

pyqtgraph / **pyqtgraph**

Watch 135 | Star 1,063 | Fork 442

Code | Issues 213 | Pull requests 138 | Projects 1 | Wiki | Insights

First time contributing to pyqtgraph/pyqtgraph?

Dismiss

If you know how to fix an issue, consider opening a pull request for it.
You can read this repository's contributing guidelines or GitHub Help to learn how to open a good pull request.

Filters | is:pr is:open | Labels | Milestones | New pull request

138 Open ✓ 236 Closed | Author | Labels | Projects | Milestones | Reviews | Assignee | Sort

Fix issue #145 ✗ — 1
#666 opened 10 days ago by feketeimre

feature More parameter item types ✗

Figure 4. Screenshot of PyQtGraph project's Github's "pull requests" page, showing the status of current requests [20].

3.4 Three Cases of Feature Development

For the purpose of studying the code development process in the PyQtGraph Library, the project's forum posts were browsed. From the hundreds of discussions available, a few discussions relating to desired (new) features or improvements for existing features were chosen for a more complete analysis.

The discussions chosen represented different types of user forum interaction outcomes: in some cases, new feature requests went through a complete cycle that led to successful feature development; in others, either the necessary code development efforts were missing or a submitted code patch was not accepted. Each case offers some insight into the characteristics of the PyQtGraph project's code development process.

Each of the discussions chosen either had the maintainer of PyQtGraph Library commenting on it or the discussion had led to a pull request on PyQtGraph's Github page being accepted. Acceptance of the pull request suggests that the maintainer had agreed to incorporate the changes in the code repository. The discussions selected were chosen from among those where the maintainer had participated because it provided some kind of legitimacy to the discussion.

Besides, the primary objective of this study requires the observation of the new feature development process and an understanding of the bug resolution process in PyQtGraph Library. For a feature to be added to the library or a bug to be resolved, the maintainer of PyQtGraph Library must participate in the process.

However, that does not mean that discussions where the maintainer of PyQtGraph Library has not commented on are frivolous. A discussion relevant to PyQtGraph Library's maintenance could have been overlooked by the maintainer or it could be a repeat of a previous discussion.

3.4.1 Case A: A Proposal for “Brushing” up a Plot Feature

A user, Bobby Henley, reported an issue on PyQtGraph's Google Groups forum about the limitation of PyQtGraph's scatterplot feature [21]. Henley wanted to construct a scatterplot containing a large amount of points, and with the possibility of changing the colors of points at a later stage.

The user had already browsed through the documentation for setBrush, and as per his understanding, the documentation indicated that the “setBrush” feature should be able to take a list of values and set the brush colors accordingly.

However, in the user's experience, the command with a serious handicap: it accepted only a single brush value for all points on a scatterplot. When the user tried to input a list, PyQtGraph reported an error.

Here is the user's example code of what worked (code A) and what doesn't work (code B), though it should have worked

(code A)

Using only one input value, the following code works well:

```
example=pg.ScatterPlotItem()
example.setBrush('r')
```

(code B)

When the input is a list:

```
example=pg.ScatterPlotItem()
example.setBrush(['r']*numberofscatterpoints)
```

Using the above code returns the following error:

```
 File      "E:\Robert\WinPython-64bit-2.7.6.4\python-2.7.6.amd64\lib\site-pack-
ages\pyqtgraph\graphicsItems\ScatterPlotItem.py", line 446, in setBrush
    if kargs['mask'] is not None:
          KeyError: 'mask'
```

This post triggered a discussion [21] where another user, Nicholas Tan Jerome, opined that he too had noticed the problem and that Henley (the user who posted the anomaly) was right in suggesting that setBrush should also accept a list. Besides, in Jerome’s opinion, a user should be able to use a mask to determine how to fill the list.

Jerome also presented a solution [21]: a “hack” to bypass the problem. The hack was to pass the "mask" keyword to the “setBrush” method.

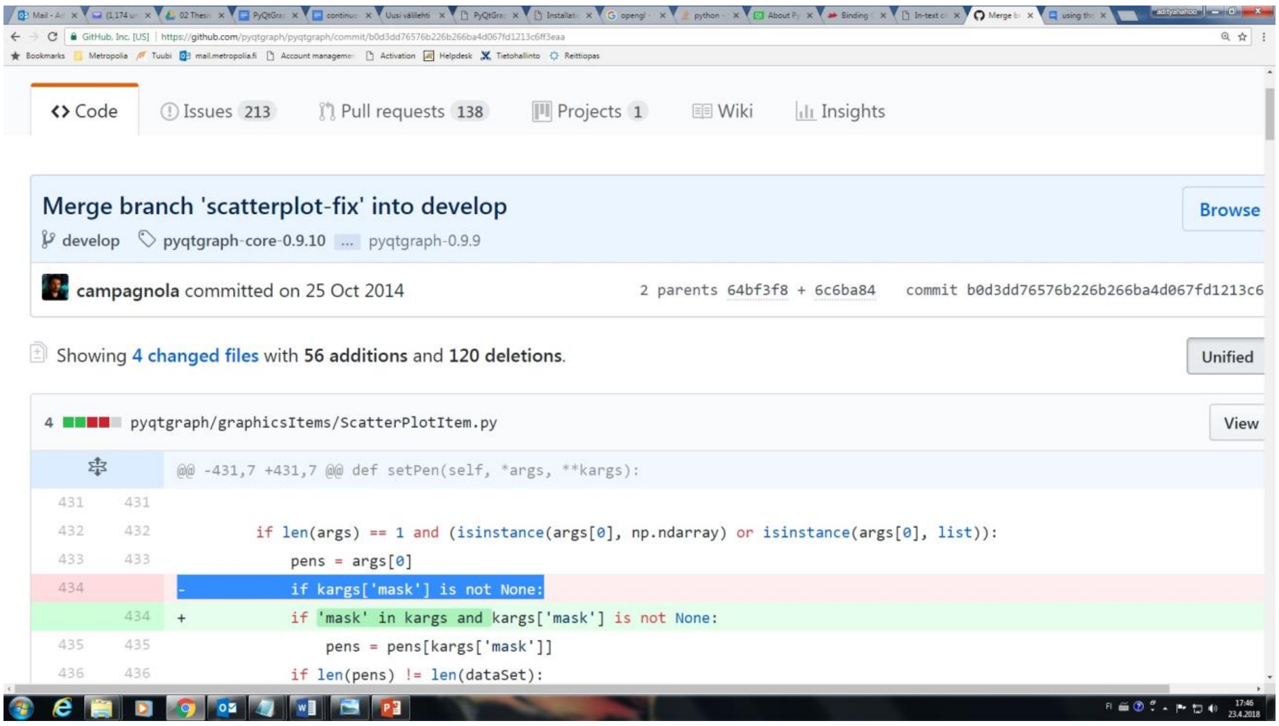
The solution presented by Jerome was to change the code to the following in the file ScatterPlotItem.py :

```
example.setBrush(['r']*numberofscatterpoints, mask=None)
```

In Jerome’s analysis, the keyword was not checked properly. “If one can just check the mask keyword before, then we will not need to always pass mask=None to make this work,” he opined.

Based on his findings, Jerome had presented the code for review to the PyQtGraph community in a pull request. This request had been accepted.

All in all, this example seemed to be a good case of a bug fix by a community review that benefits users at large and requires minimum intervention by the moderator. The change in code is, incidentally, just one line, as seen in figure 5.


GitHub, Inc. [US] | https://github.com/pyqtgraph/pyqtgraph/commit/b0d3dd76576b226b266ba4d067fd1213c6ff3eaa
Code
Issues 213
Pull requests 138
Projects 1
Wiki
Insights
Browse
Unified
View

Merge branch 'scatterplot-fix' into develop

develop pyqtgraph-core-0.9.10 … pyqtgraph-0.9.9

campagnola committed on 25 Oct 2014   2 parents 64bf3f8 + 6c6ba84   commit b0d3dd76576b226b266ba4d067fd1213c6

Showing 4 changed files with 56 additions and 120 deletions.

4 pyqtgraph/graphicsItems/ScatterPlotItem.py

```
@@ -431,7 +431,7 @@ def setPen(self, *args, **kargs):

        if len(args) == 1 and (isinstance(args[0], np.ndarray) or isinstance(args[0], list)):
            pens = args[0]
-           if kargs['mask'] is not None:
+           if 'mask' in kargs and kargs['mask'] is not None:
                pens = pens[kargs['mask']]
            if len(pens) != len(dataSet):
```

Figure 5. Screenshot of the changed program file showing one code line added and one code line deleted [22].

The code development process for this particular code in file ScatterPlotItem.py is illustrated in the process-diagram in figure 6:

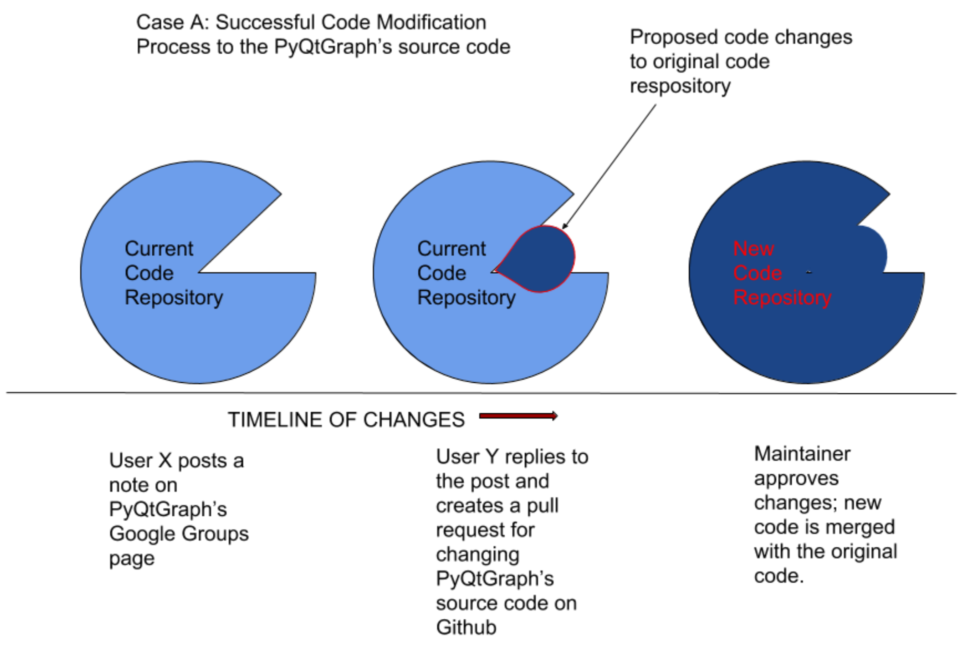


Figure 6. Timeline of events for a typical successful feature-addition process.

3.4.2 Case B: Trouble with Data Overload

The nature of PyQtGraph library -- stable, well-documented basic applications that however tend to work slower as data-sets get very large -- means that significant feature-addition requests happen largely in areas where “performance issues” appear. These issues manifest themselves in slower rendering of the results, or a complete inability to process the data. Such discussion can be seen on the PyQtGraph user forums, on Google Groups and sometimes on Stackoverflow.

A user, Louie D on PyQtGraph’s Google Groups forums, had a requirement of plotting thousands of lines (100,000 lines) and asked for guidance on how to proceed [23]. He had noticed that for the first 10,000 lines, the plot rendering works as expected, but as he added more lines to the plot there was an exponential performance decrease “that brings the program to its knees before 100,000 lines”. He added a plot of 100,000 lines, as seen in figure 7, to clarify his question.

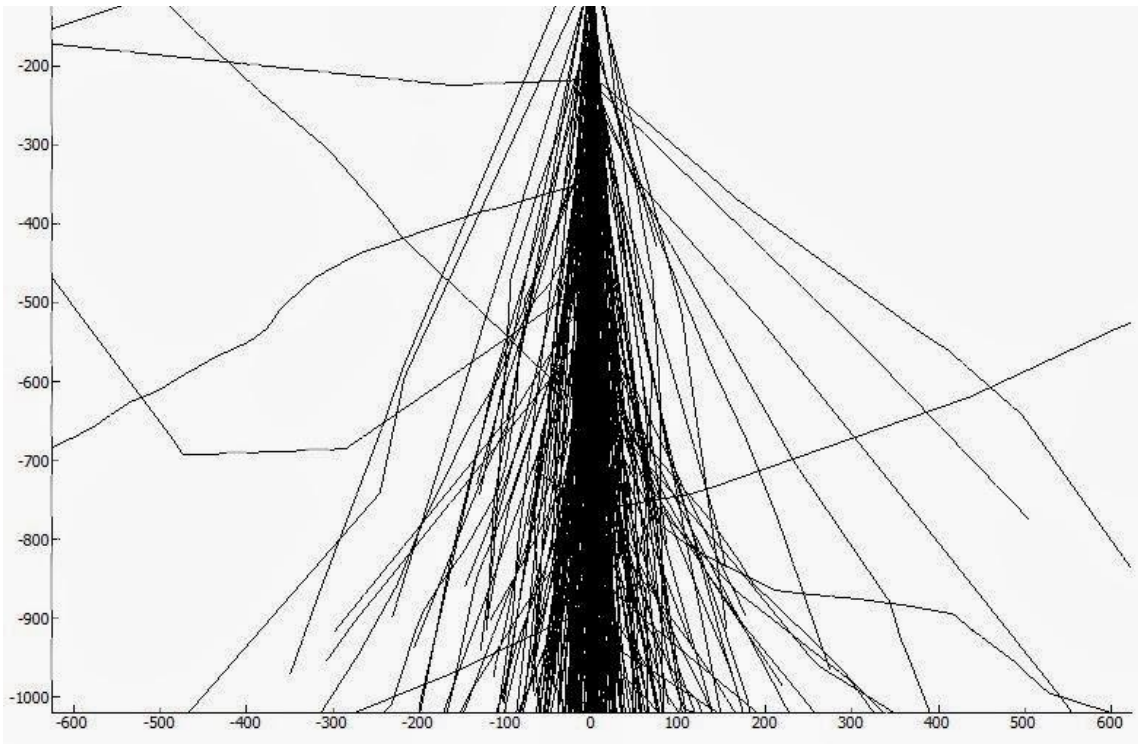


Figure 7. Snapshot of PyQtGraph’s plotting tool by user Louie [23]

The user admitted to being relatively new to python. However, he stated what he felt must be the cause for the observed slow output: that some sort of append/pushback function required a complete data copy every time he added a new line.

The user’s code reproduced what he had been using in his code thus:
“Currently I'm creating a plot & window and then using:

```
curve = pg.PlotCurveItem(x,y)

  plt.addItem(curve)
```

for each different line I want to add to the plot.”

Luke Campagnola, the maintainer of PyQtGraph library, commented in detail [23], with both suggestions and enquiries. He also asked for clarification on whether the user wanted to improve performance in the initial plotting, or in interactive pan/zooming, or both.

The second thing Campagnola was interested in knowing was whether the lines being plotted were largely overlapping or non-overlapping. This knowledge could be put to use in deciding on the best strategy for improving performance. If a simple example were to be provided, it would help, Campagnola noted. [23.]

Campagnola also provided a possible solution: putting all lines into a single item. He noted, however, that this was probably not a good solution as he ran into a similar problem as the one the user was already facing: when Campagnola experimented with 100,000, line, his machine wasn't performing normally and eventually Qt crashed after a memory allocation issue.

What was the reason behind this abnormal memory usage? Campagnola [23] suspected that this memory is being used directly by Qt since most of the memory usage appears immediately after the item is added to the plot.

If indeed Qt was responsible for the memory hoarding, then finding a solution would require some clever thinking as the Qt libraries themselves were not a part of PyQtGraph. The installation guide for PyQtGraph notes that the Qt libraries, which are a dependency for PyQtGraph, are required for PyQtGraph's functioning [2].

The user wrote back clarifying that he was only interested in plotting the lines and not in other features. He also provided a sample plot of the data he was working with.

Next up in the discussion, another user Jochen Schröder [23] wrote in to talk about the work he had done under similar data conditions. Schröder had developed 2d opengl plotting additions to PyQtGraph.

With these additions, Schröder claimed that he could easily plot more than 100, 000 spots and lines. Schröder's specific instructions were as below:

“The github directory is here:

https://github.com/cycomanic/PyQtGraph/

use the glcamera_attribute branch.”

Campagnola also replied [23], this time offering a second solution: he felt that in the current case, one of high overlap and no interactivity, probably stacking many lines into a single QGraphicsPathItem is the best option.

Campagnola had run the code against the user’s data and found out that although he couldn’t efficiently get 10M points into a single path item, it was possible to do 100 paths with 100k points each. He admitted that it was slow (about 10 seconds per update), but nevertheless it worked.

Clearly, the user Louie D who had first asked the question, had at least two alternatives between which to decide: one being to continue to use PyQtGraph, by adopting the code trick Campagnola had suggested. The other option was to abandon attempts to use the PyQtGraph in its current form, and instead use the 2d opengl plotting additions to PyQt-Graph developed by the user Jochen Schröder (as discussed previously in this section). Here, it should be noted that these additions were not part of the PyQtGraph code re-pository, so they did not come with the same user-tested assurance as the PyQtGraph project.

Another point worth noting is that this discussion raised by Louie D served to inform other users about the existence of the 2d opengl plotting additions to PyQtGraph as developed by another user Jochen Schröder. These plotting additions had not been tied back to the original PyQtGraph project, so it is only by browsing discussions such as this forum post that the general reader can be made aware of the specific code extension available for PyQtGraph.

Unfortunately, it is not clear what recourse the user Louie D resorted to because there is no further discussion on the forum.

3.4.3 Case C: A New Time Axis for PyQtGraph

This third and final code case study looks at what happened when a user offered a set of code files by creating a Pull Request on PyQtGraph's GitHub page for a new feature that had been lacking in the PyQtGraph library.

By default, a PyQtGraph example does not automatically provide a date axis (there exists, however, a time axis). Lukas Heiniger, nicknamed "3rdCycle", a PyQtGraph user, wrote on PyQtGraph's GitHub "Pull Requests" page to announce that he had implemented a date axis with basic time zone support [24]. "At the moment it only covers minute to year scales but it should be quite easy to add new zoom levels as required," 3rdCycle had clarified.

Luke Campagnola, the maintainer of PyQtGraph library, had commented [24] that though the feature developed seemed useful and that the code block "looked good", there were some problems. There were four items in the list that needed mending, according to Campagnola.

Some of these correction items, such as adopting the correct code style, seemed relatively simple to ammend, but others were probably not that straightforward. For instance, Campagnola felt that at certain instances on the scale, with stages corresponding to greater than one year and less than one minute stages, the PyQtGraph application should probably revert to the default AxisItem behavior.

Another observation made by Campagnola was that the time scale was not indicating years that appeared before 1900. 3rdCycle had reasoned that this was due to strftime() method not supporting dates before 1900. He had conceded that there did not seem to be an easy way around this without introducing new dependencies.

This particular discussion [24] on the forum continued back and forth with some more suggestions by Campagnola and some rectifications by the developer 3rdCycle. This discussion had been started five years ago, however, the feature had not yet seen the light of the day at the time of writing this thesis. That is, the pull request by the developer had not been accepted by Campagnola, the maintainer of PyQtGraph library.

This particular proposed feature found support from other users as well. At least three other users had voiced their interest on the PyQtGraph's GitHub "Issues" page in seeing the feature developed [24]. 3rdCycle had cooperated with Campagnola in trying to resolve the pending issues by replying in detail on how he was approaching the problems but some issues had remained unresolved. Campagnola had admitted that this feature was a "tough one" and that is why it had not yet been implemented.

Figure 8 is a simplified version of the timeline of interactions between the developer, 3rdCycle and maintainer, Campagnola. Snapshots of the interaction are provided in the Appendix.

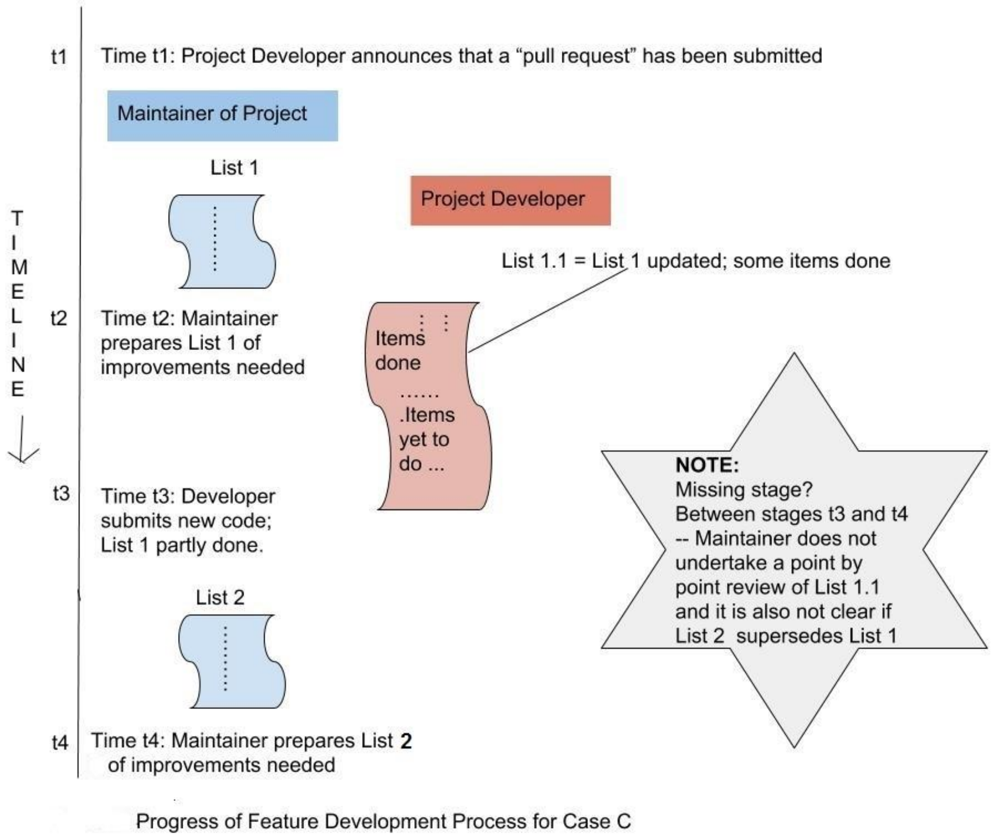

t1
Time t1: Project Developer announces that a "pull request" has been submitted
Maintainer of Project
List 1
Project Developer
List 1.1 = List 1 updated; some items done
TIMELINE
t2
Time t2: Maintainer prepares List 1 of improvements needed
Items done
......
.Items yet to do ...
NOTE:
Missing stage?
Between stages t3 and t4
-- Maintainer does not undertake a point by point review of List 1.1 and it is also not clear if List 2 supersedes List 1
t3
Time t3: Developer submits new code; List 1 partly done.
List 2
t4
Time t4: Maintainer prepares List 2 of improvements needed
Progress of Feature Development Process for Case C

Figure 8: Timeline of interactions for the "New Time Axis" proposed feature

Though the content matter of the discussions is quite clear, it is unclear as to whether the maintainer accepts the changes made by the developer at time, t4. It is also not very clear as to what are the minimal set of changes the maintainer expects to see for the feature to be accepted in the project code.

It is pertinent to note that pull requests that have taken many years to resolve present other problems that further delay the code's acceptance. An interesting observation surfaced in the forum while discussing the challenges in accepting the Time Axis feature: this issue concerned changes in the Python language during the periodic revision of the language that were now affecting the code in the "pull request" files. The offending code related to a dictionary iterkeys()method, used in one of the methods setZoomLevelForDensity method. The iterkeys()method is no longer available in Python 3. A user Götz Christ had noticed this and also indicated the solution: replacing the now-defunct *iterkeys()* with *keys()* [24].

4. Improving the New Feature Development Process

A study of the PyQtGraph library development process showed that different feature development attempts meet with varied levels of success. While it would do good for a potential contributor to keep the feature development guidelines found on the forum's Github page in mind (contents attached as Appendix 2), including such principles as following the style guide and testing the new code against different data sets, that alone does not guarantee that the code would be accepted.

The case studies showed that the concerns for the maintainer of the PyQtGraph library relate not just to the efficacy of some (new) code but also to whether a (new) patch of code will work in the future and whether the particular feature under development is aligned with the developer's vision. In those cases where the ensuing discussion has led to bug fixes without disrupting the direction for PyQtGraph's development, theses bug fixes have been accepted. The direction for PyQtGraph's development, in this case, is the one that is spelt out by PyQtGraph's maintainer, Luke Campagnola.

The maintainer's vision can be partly gleamed from the comments he has made on the user forums at different instances.

The current process of posting a Pull Request on Github is simple and straightforward. From the case studies presented, it appears that this process works very well in the case when code modifications are relatively minor. For instance, a user, having fixed a bug or created a new feature, sends a 'Pull Request' with a modified program file that, in the opinion of the maintainer, is fit to be accepted. Such was the case with case in section 3.1.1 Case A: A proposal for "brushing" up a plot feature.

4.1 Reviewing the Pirate Metrics Model for Open-Source Projects

However, more complex feature development may be too much work for a single user, keeping in mind that all work on the PyQtGraph Project, like other open-source projects, is voluntary.

An analysis of the feature development cases has shown that the use of Pirate Metrics for measuring success of an open-source project such as PyQtGraph could be modified to reflect the reality that code contribution itself by a user is often not adequate. The new code suggested undergoes a process of review and changes during which the contributor needs to remain active or, a team of substitute developer/s needs to be available who can complete the required code changes.

Figure 9 is an attempt to depict the need for including this interaction component in the PyQtGraph's Pirate Metrics success measurement diagram. An effective interaction process at this stage is essential for the acceptance of the new code in the repository, and thus important for the continued success of the project.

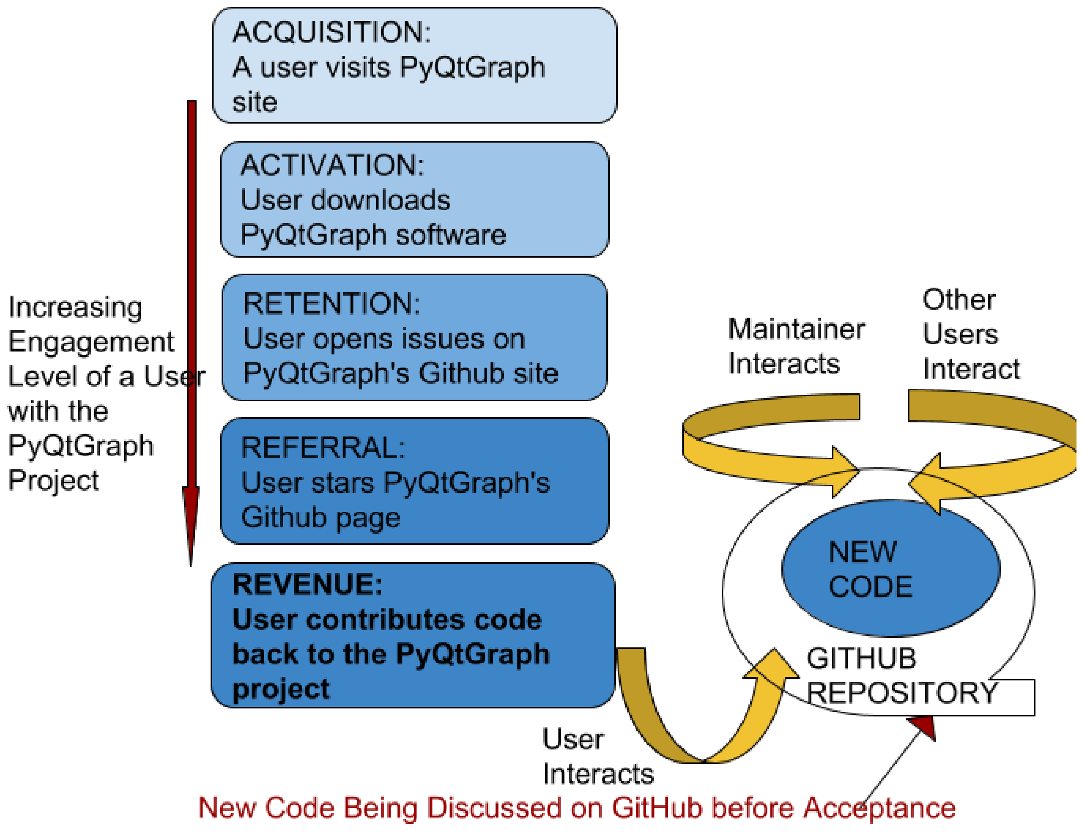

ACQUISITION:
A user visits PyQtGraph site
ACTIVATION:
User downloads PyQtGraph software
RETENTION:
User opens issues on PyQtGraph's Github site
REFERRAL:
User stars PyQtGraph's Github page
REVENUE:
User contributes code back to the PyQtGraph project
Increasing Engagement Level of a User with the PyQtGraph Project
Maintainer Interacts
Other Users Interact
NEW CODE
GITHUB REPOSITORY
User Interacts
New Code Being Discussed on GitHub before Acceptance

Figure 9: Measuring success of PyQtGraph project using Pirate Metrics + 'Interaction' Component

It is also important to note that it is advantageous for the project's code growth that this interaction between desirous code developers is conducted within a short span of time. One of the reasons being that if the feature development process takes too long the underlying dependencies may have undergone revision and so the (new) code may not compile. Another reason is that developers on the forum have no contract of commitment to develop some code. If the process stretches long, a developer may lose interest in contributing.

4.2 Tool for Aiding Collaboration Between Users

In the scenario when a developer issuing a Pull Request is likely to benefit from some kind of collaboration with other experienced users, how then does the developer reach out to other users? PyQtGraph Google Groups is one good candidate for an appropriate forum, however, the Pull Request process on Github is not automatically visible in PyQt-Graph Google Groups forum.

The Pull Request process initiator can, of course, post on the PyQtGraph Google Groups but it is possible that not every initiator does so. Besides, it is not always evident from the subject line of a PyQtGraph Google Groups whether it has an associate pull request on Github. In case study Case C: A New Time Axis for PyQtGraph, the corresponding discussion on PyQtGraph Google Groups was titled "Example with datetime axis", while the actual post about the Pull Request was written as a comment, several posts beneath the starting post. A user searching for new feature development discussions could easily have missed this post, or even if he had read this post, it would not be clear from the starting discussions whether there was a possibility to collaborate for contributing code.

These factors support the development of a tool that would aid collaboration between users during the feature development process.

Such a tool could be an extension to Github or be a new feature to PyQtGraph Google Group or come as a standalone third-party application that would do the following:

- synergize Pull Request posts on Github and discussions on PyQtGraph's Google Groups by an efficient tag process
- collect together all comments on posts where the maintainer/s provide idea about the direction for PyQtGraph's development

- during the course of interaction between the maintainer and developer about the Pull Request for a specific feature: the tool should provide a means for listing the improvements expected by the maintainer for the pull request in question to be accepted. The tool should also provide a means to measure the progress of further code corrections/development on these listed items.

5. Conclusions

PyQtGraph is an open-source software library used for data visualization and creating GUIs. This study has aimed at understanding the mechanism of PyQtGraph project code's maintenance and growth and providing suggestions to improve the same.

The alternative graphical tools available, their strengths and weaknesses with respect to PyQtGraph, have also been discussed in brief. While doing so, the context in which PyQtGraph is used has been clarified. Issues regarding features that don't work or those that are found lacking in PyQtGraph have also been taken up; the source for most of this being the user forums. The comparison discussions make it easier to comprehend the importance of the community effort to strengthen the PyQtGraph project code.

This study could aid a developer wishing to contribute to PyQtGraph project code. The study would also be useful to the maintainer of the PyQtGraph project to review the code development efforts in the recent past, so as to steer future discussion in a manner that would solicit developers' support.

The issue of quality of the code submitted during a Pull Request and what are the acceptable standards for the maintainer is a debatable one. During one of the feature development initiatives (the case of ' A New Time Axis for PyQtGraph'), though the feature in question was seen to have quite a few supporters, the maintainer had found the code lacking in some respects and had not accepted it in the code repository.

The above-mentioned case study, which unfortunately did not result in a new feature, also hinted that the collaboration between the maintainer and the developer was not adequate. Though it cannot be said with certainty as to what impeded the new feature from being developed, efforts can be taken to increase collaboration between users by developing a collaboration tool. A suggestion for improving collaboration between users at a specific stage of feature development has also been presented in the previous chapter. The development of such a tool and its experimental use could be the topic of another study.

The code development process studied was for one open-source library. Based on the interactions between different code contributors and the maintainer, the above conclusions have been drawn.

The dynamics between developers in another open-source project may be different from this one. However, by reading this thesis, the reader will be in a position to create some metrics for understanding another open-source project.

References

1. Luke Campagnola. PyQtGraph Project Home page: http://www.pyqtgraph.org/ [Internet] [cited 15 January 2018]

2. Luke Campagnola. PyQtGraph Project Official Documentation page: http://www.pyqtgraph.org/documentation/installation.html [Internet] [cited 15 January 2018]

3. Stackoverflow discussion: https://stackoverflow.com/questions/6888750/pyqt-or-pyside-which-one-to-use [Internet] [cited 01 March 2018]

4. PyQt Whitepaper. https://www.riverbankcomputing.com/static/Docs/PyQt4/pyqt-whitepaper-us.pdf [Internet] [cited 30 January 2018]

5. About PySide https://wiki.qt.io/About_PySide [Internet] [cited 15 March 2018]

6. Qt Wiki https://en.wikipedia.org/wiki/Qt_(software) [Internet] [cited 15 March 2018]

7. OpenGL https://www.khronos.org/opengl/ [Internet] [cited 15 March 2018]

8. Stackoverflow discussion: Using matplotlib or pyqtgraph to graph real time data https://stackoverflow.com/questions/13181118/using-matplotlib-or-pyqtgraph-to-graph-real-time-data [Internet] [cited 20 March 2018]

9. Live Data in PyQt4 with PlotWidget https://www.swharden.com/wp/2016-07-31-live-data-in-pyqt4-with-plotwidget/ [Internet] [cited 01 March 2018]

10. Stackoverflow discussion: Why is plotting with Matplotlib so slow? https://stackoverflow.com/questions/8955869/why-is-plotting-with-matplotlib-so-slow [Internet] [cited 01 March 2018]

11. Stackoverflow discussion: Displaying pyqtgraph and pyqt widgets on web https://stackoverflow.com/questions/29928485/displaying-pyqtgraph-and-pyqt-widgets-on-web [Internet] [cited 01 March 2018]

12. Bokeh Project Home page: http://bokeh.pydata.org/en/latest/ [Internet] [cited 01 March 2018]

13. PyQtGraph Google Groups page: https://groups.google.com/forum/#!forum/pyqtgraph [Internet] [cited 03 April 2018]

14. Bokeh Google Groups page: https://groups.google.com/a/continuum.io/forum/#!forum/bokeh [Internet] [cited 12 April 2018]

15. Gaby Fachler. Using Pirate Metrics to measure success of open-source projects https://opensource.com/business/16/6/pirate-metrics [Internet] [cited 15 April 2018]

16. Walter Chen. AARRR! Dave McClure's "Pirate Metrics" And The Only Five Numbers That Matter https://www.inc.com/walter-chen/aarrr-dave-mcclure-s-pirate-metrics-and-the-only-five-numbers-that-matter.html [Internet] [cited 15 April 2018]

17. Ibrahim Haddad (Ph.D.) and Brian Warner, The Linux Foundation. Understanding the Open-source Development Model https://www.linuxfoundation.org/events/understanding-the-open-source-development-model/ [Internet] [cited 01 March 2018]

18. Documentation TODO PyQtGraph Github Wiki page: https://github.com/pyqtgraph/pyqtgraph/wiki/Documentation-TODO [Internet] [cited 15 January 2018]

19. Felix Krause. Scaling Open-source Communities https://academy.realm.io/posts/tryswift-felix-krause-scaling-open-source-communities-github-management/ [Internet] [cited 01 March 2018]

20. PyQtGraph Github Pull Request page https://github.com/pyqtgraph/pyqtgraph/pulls [Internet] [cited 23 April 2018]

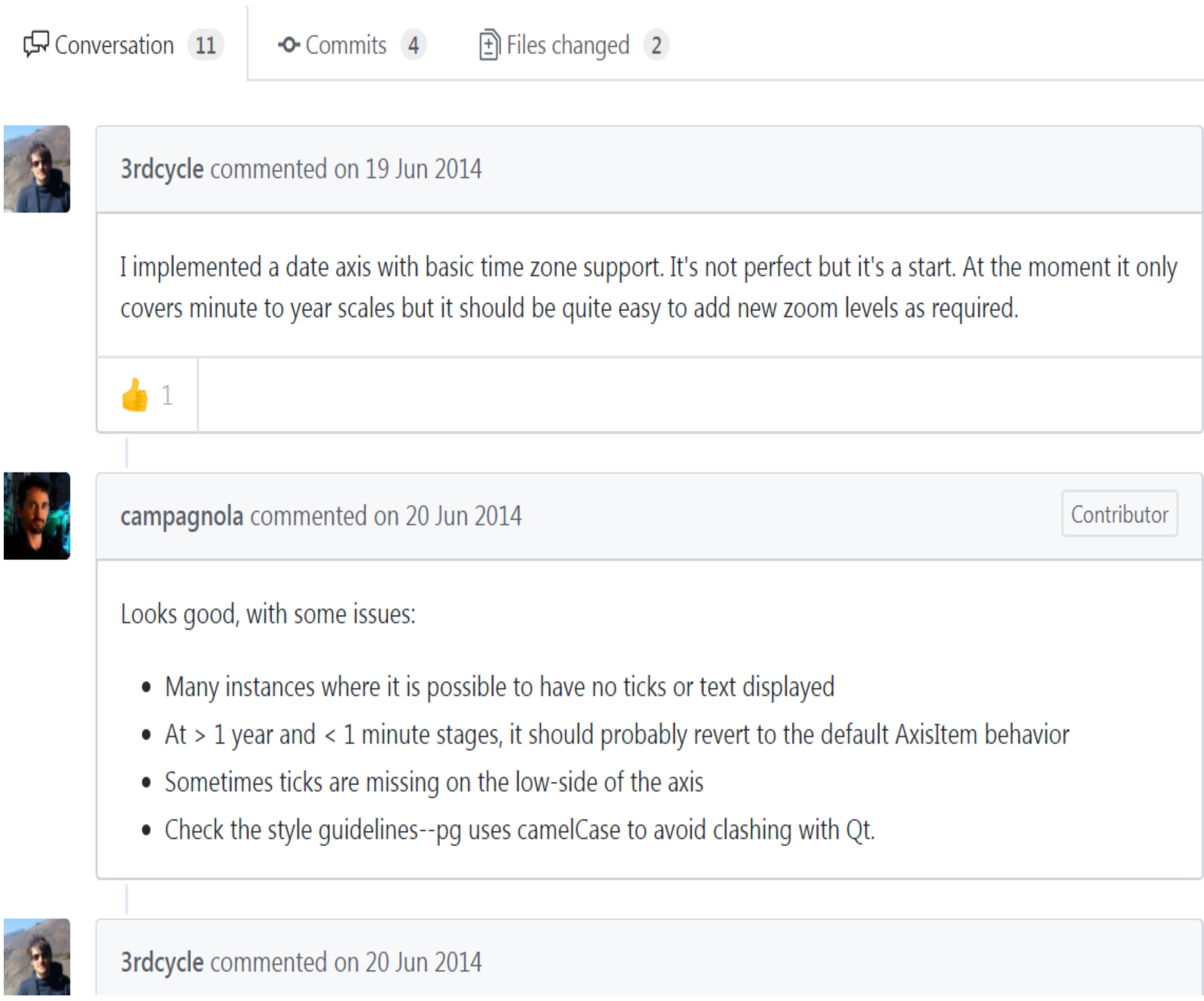
21. Using the setBrush command for a list https://groups.google.com/forum/#!topic/PyQtGraph/xVyCC2f7gVo [Internet] [cited 15 March 2018]

22. PyQtGraph Github Scatterplot-Fix page https://github.com/pyqtgraph/pyqtgraph/commit/b0d3dd76576b226b266ba4d067fd1213c6ff3eaa [Internet] [cited 15 March 2018]

23. Plotting many (thousands of) lines with PyQtGraph https://groups.google.com/forum/#!topic/pyqtgraph/kz4U6dswEKg [Internet] [cited 15 March 2018]

24. DateAxisItem #74 https://github.com/pyqtgraph/pyqtgraph/pull/74 [Internet] [cited 15 March 2018]

1. Appendix 1: PyQtGraph's GitHub "Time Axis" Feature Page Interactions



Conversation 11 Commits 4 Files changed 2

3rdcycle commented on 19 Jun 2014

I implemented a date axis with basic time zone support. It's not perfect but it's a start. At the moment it only covers minute to year scales but it should be quite easy to add new zoom levels as required.

👍 1

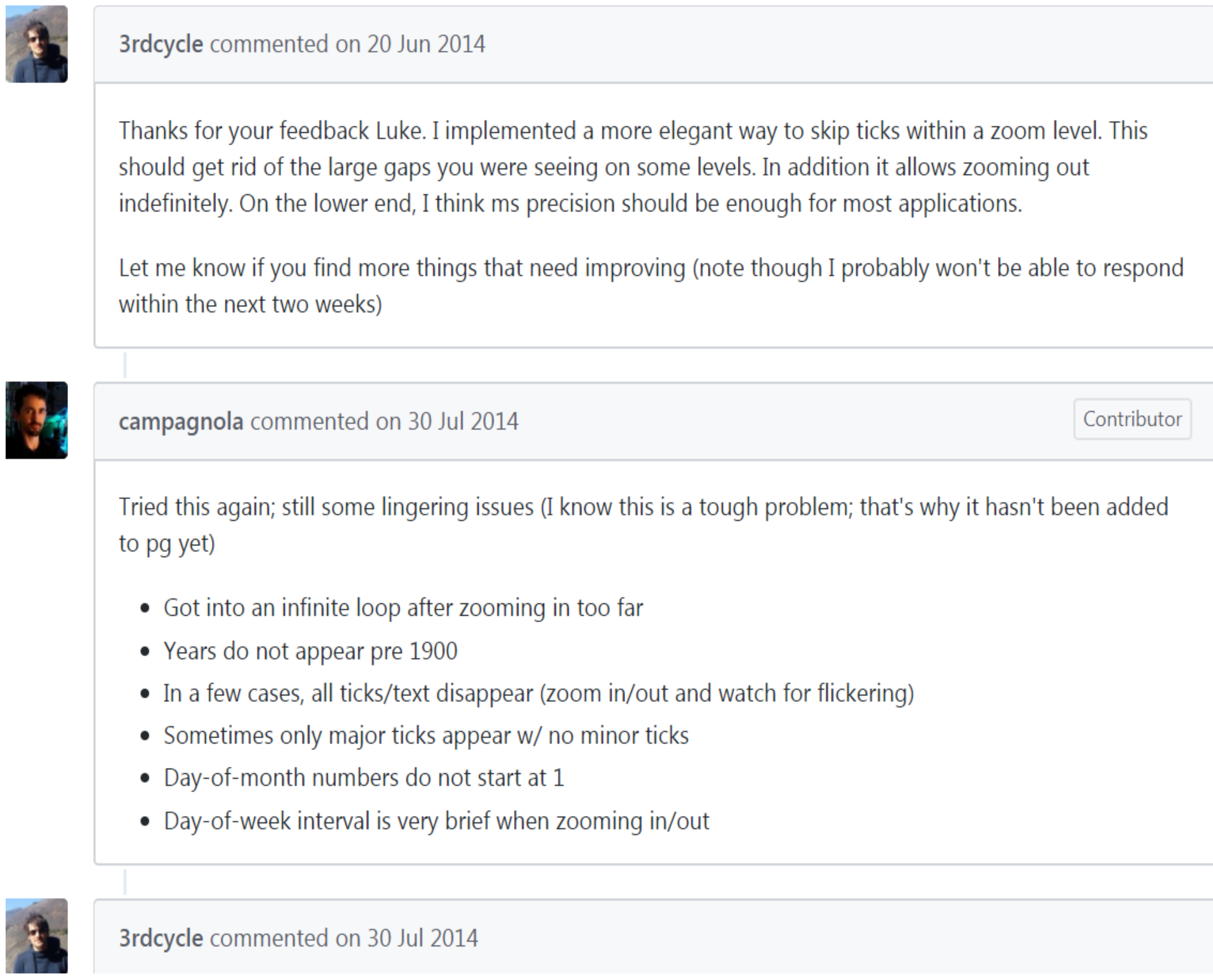
campagnola commented on 20 Jun 2014 Contributor

Looks good, with some issues:

- Many instances where it is possible to have no ticks or text displayed
- At > 1 year and < 1 minute stages, it should probably revert to the default AxisItem behavior
- Sometimes ticks are missing on the low-side of the axis
- Check the style guidelines--pg uses camelCase to avoid clashing with Qt.

3rdcycle commented on 20 Jun 2014

Figure 1: "Time Axis" Feature Page Interactions – page 1



3rdcycle commented on 20 Jun 2014

Thanks for your feedback Luke. I implemented a more elegant way to skip ticks within a zoom level. This should get rid of the large gaps you were seeing on some levels. In addition it allows zooming out indefinitely. On the lower end, I think ms precision should be enough for most applications.

Let me know if you find more things that need improving (note though I probably won't be able to respond within the next two weeks)

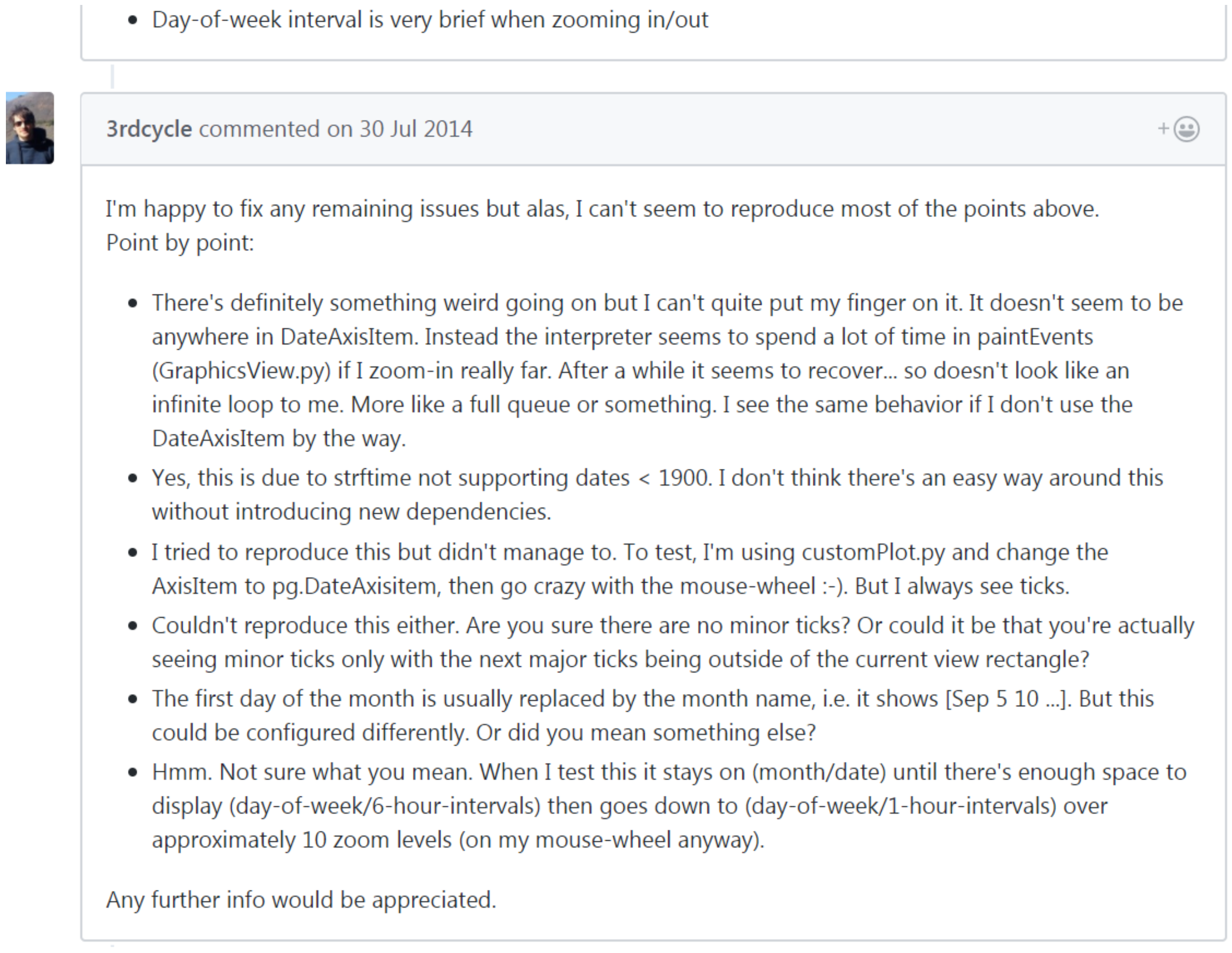
campagnola commented on 30 Jul 2014 Contributor

Tried this again; still some lingering issues (I know this is a tough problem; that's why it hasn't been added to pg yet)

- Got into an infinite loop after zooming in too far
- Years do not appear pre 1900
- In a few cases, all ticks/text disappear (zoom in/out and watch for flickering)
- Sometimes only major ticks appear w/ no minor ticks
- Day-of-month numbers do not start at 1
- Day-of-week interval is very brief when zooming in/out

3rdcycle commented on 30 Jul 2014

Figure 2: “Time Axis” Feature Page Interactions – page 2



- Day-of-week interval is very brief when zooming in/out

3rdcycle commented on 30 Jul 2014

I'm happy to fix any remaining issues but alas, I can't seem to reproduce most of the points above.
Point by point:

- There's definitely something weird going on but I can't quite put my finger on it. It doesn't seem to be anywhere in DateAxisItem. Instead the interpreter seems to spend a lot of time in paintEvents (GraphicsView.py) if I zoom-in really far. After a while it seems to recover... so doesn't look like an infinite loop to me. More like a full queue or something. I see the same behavior if I don't use the DateAxisItem by the way.
- Yes, this is due to strftime not supporting dates < 1900. I don't think there's an easy way around this without introducing new dependencies.
- I tried to reproduce this but didn't manage to. To test, I'm using customPlot.py and change the AxisItem to pg.DateAxisitem, then go crazy with the mouse-wheel :-). But I always see ticks.
- Couldn't reproduce this either. Are you sure there are no minor ticks? Or could it be that you're actually seeing minor ticks only with the next major ticks being outside of the current view rectangle?
- The first day of the month is usually replaced by the month name, i.e. it shows [Sep 5 10 ...]. But this could be configured differently. Or did you mean something else?
- Hmm. Not sure what you mean. When I test this it stays on (month/date) until there's enough space to display (day-of-week/6-hour-intervals) then goes down to (day-of-week/1-hour-intervals) over approximately 10 zoom levels (on my mouse-wheel anyway).

Any further info would be appreciated.

Figure 3: “Time Axis” Feature Page Interactions – page 3

Appendix 2: PyQtGraph’s GitHub Pull Request Guidelines

```
Contributions to pyqtgraph are welcome!


                Please use the following guidelines when preparing changes:


                * The preferred method for submitting changes is by github pull request
                  against the "develop" branch.


                * Pull requests should include only a focused and related set of changes.
                  Mixed features and unrelated changes may be rejected.


                * For major changes, it is recommended to discuss your plans on the mailing
                  list or in a github issue before putting in too much effort.


                * Along these lines, please note that pyqtgraph.opengl will be deprecated
                  soon and replaced with VisPy.


                * Writing proper documentation and unit tests is highly encouraged. PyQtGraph
                  uses nose / py.test style testing, so tests should usually be included in a
                  tests/ directory adjacent to the relevant code.


                * Documentation is generated with sphinx; please check that docstring changes
                  compile correctly.
```

* Style guidelines:

    * PyQtGraph prefers PEP8 for most style issues, but this is not enforced rigorously as long as the code is clean and readable.

    * Use `python setup.py style` to see whether your code follows the mandatory style guidelines checked by flake8.

    * Exception 1: All variable names should use camelCase rather than underscore_separation. This is done for consistency with Qt

    * Exception 2: Function docstrings use ReStructuredText tables for describing arguments:

      ```
      ============== ========================================================
      **Arguments:**
      argName1       (type) Description of argument
      argName2       (type) Description of argument. Longer descriptions must
                     be wrapped within the column guidelines defined by the
                     "====" header and footer.
      ============== ========================================================
      ```

      QObject subclasses that implement new signals should also describe these in a similar table.

```
* Setting up a test environment.


    Tests for a module should ideally cover all code in that module,
    i.e., statement coverage should be at 100%.


    To measure the test coverage, install py.test, pytest-cov and pytest-xdist.
    Then run 'py.test --cov -n 4' to run the test suite with coverage on
    4 cores.
```