D01LAR BILLS/Team 30

Deliverable 1

CSCD01 WINTER 2020

Matplotlib Code Base & Planning

Submitted To:
Dr. Anya Tafliovich

Submitted By :
Joseph Sokolon
Wesley Ma
Raya Farhadi
Edgar Sarkisian

Contents

1 Architecture of Matplotlib **2**
1.1 Backend Layer . . . 2
1.2 Artist Layer . . . 2
1.3 Scripting Layer . . . 2
1.4 High-level classes UML diagram . . . 4
1.5 Full UML diagram . . . 4

2 Chosen Software Development Process **5**
2.1 Kanban . . . 5
2.2 Columns . . . 5
2.2.1 Backlog . . . 5
2.2.2 Planned . . . 5
2.2.3 In Progress . . . 6
2.2.4 In Review . . . 6
2.2.5 Done . . . 7
2.2.6 Released . . . 7
2.3 Advantages of Kanban . . . 7
2.4 Disadvantages of Kanban . . . 7
2.5 Other Software Development Methodologies Considered . . . 8
2.5.1 XP . . . 8
2.5.2 Waterfall . . . 8

1 Architecture of Matplotlib

Based off: *https://www.aosabook.org/en/matplotlib.html*

MPL's architecture is a three-layer stack, where a lower layer is not aware of the layer above it. From bottom to top, the layers are: backend, artist, scripting.

1.1 Backend Layer

The lowest level layer responsible for encapsulating interaction with the various supported UI toolkits. It has implementations of three abstract interfaces for each UI toolkit.

Figure Canvas provides the concept of a 'canvas' to draw on.
Renderer provides functions to draw on instances of Figure Canvas.
Event provides user input handling to interact with figures that were drawn.

Class diagram of the above and other core interfaces (PyCharm): https://imgur.com/Q7KfSWM
Class diagram of the QT5Cairo implementation (PyCharm): https://imgur.com/K6KtEnL

1.2 Artist Layer

A main abstract object, Artist, which uses a Renderer to draw various graphical elements on a Figure Canvas. This is done through the draw() method which each Artist implements. All elements in an MPL figure are some type of Artist.
There are two types of Artists: **Primitive Artists** such as Line2D, Rectangle, and Text, and **Composite Objects** such as Axes (Figure 1), Tick, and Figure, which are composed of other composite and primitive artists.

Class diagram that includes a variety of Artist types (PyCharm): https://imgur.com/z7KorfF

1.3 Scripting Layer

Essentially the pyplot interface, a developer-friendly API that is meant to be used by the general users of MPL.
Uses a config file to detect the user's Ui toolkit, and provides commands to automatically generate histograms, titles, etc using the various Artists, and the correct backend interfaces.

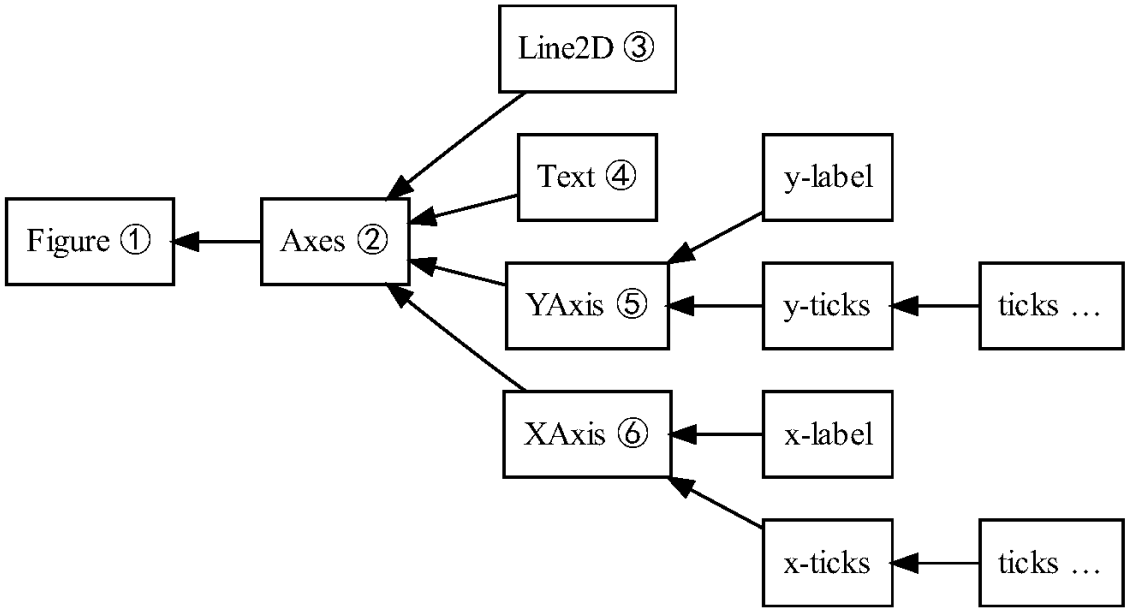

Figure 1: Axes is the most important Artist, containing many elements that make up the background of a plot (ticks, lines, grid, etc)
Taken from http://aosabook.org/images/matplotlib/artists_tree.png

1.4 High-level classes UML diagram

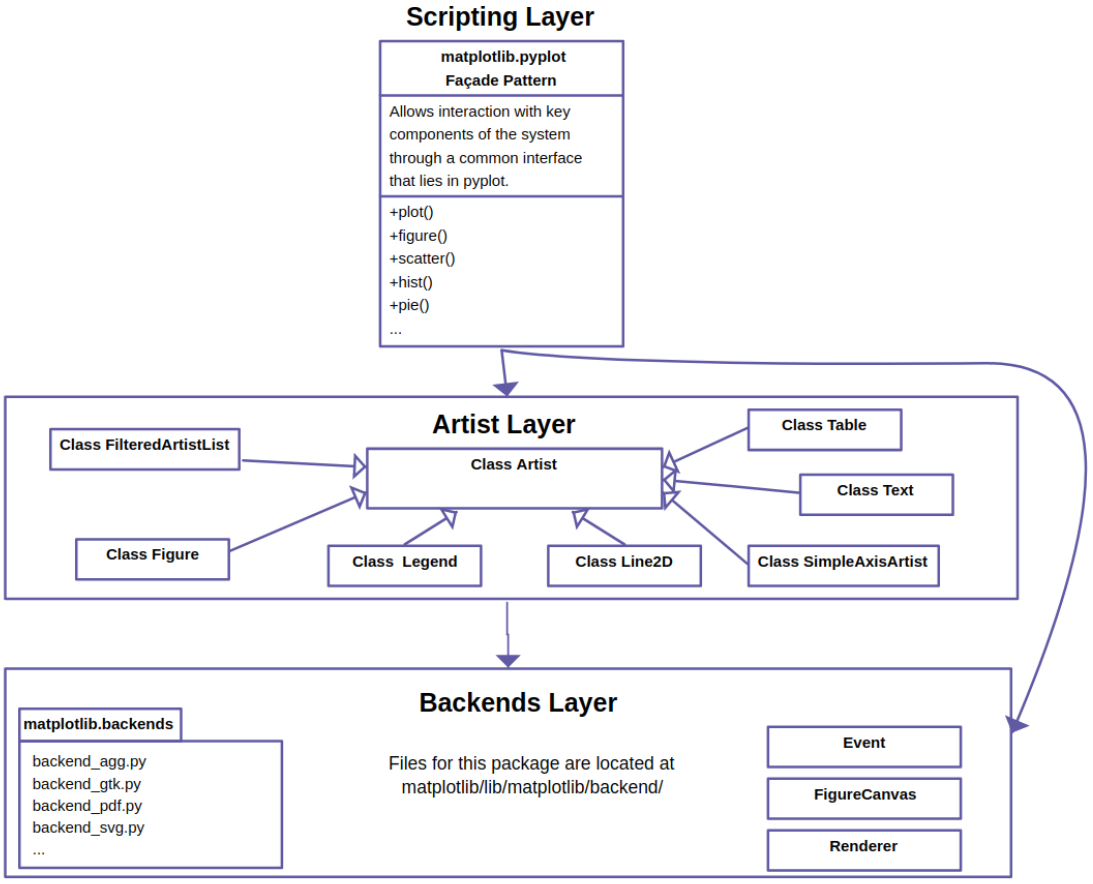

Figure 2: MPL 3-tiered architecture. Taken from
https://creately.com/diagram/example/hqz3w6j12/matplotlib%203-tiered%20architecture

1.5 Full UML diagram

A full version of the matplotlib UML diagram generated using pyreverse can be found here.

2 Chosen Software Development Process

2.1 Kanban

The software development process our team has decided to use is Kanban. We've decided to use Kanban because it's the process that best fits with the abilities and availability of the team.

We will be using the Github's project boards to implement our Kanban board. The columns "In Progress" and "In Review" will both have a max amount of 4 tasks at a time, to keep the team focused. We've chosen a max limit of 4 for both columns to reduce bottlenecks by keeping our work output even. If either of the columns contain an amount of tasks exceeding our limit, the team will prioritize completing the extra task. During the next stand-up we will also discuss if the extra task represents a bottleneck in our system and if we need to improve upon it. A task will either be a new feature or a bug-fix. Our board contains 5 columns. The column names, along with a description of the purpose of each column is listed in the section below.

To keep our team aligned and communicating we have two in person stand-ups, one on Monday and one on Thursday. We will also have online stand-ups every other day. During stand-ups team members will give a brief update on what they've been working on. During the in-person stand-ups we will also discuss tasks in the backlog when needed.

2.2 Columns

2.2.1 Backlog

A task is added to the backlog either when a team member finds a new feature that could be added or a bug that needs to be fixed. Tasks in the backlog are to be discussed at the next stand up meeting and will either be refined or split up into smaller tasks and added to "Planned", left in the backlog as something that is not urgent and can be discussed at a later meeting, or discarded as something the group doesn't feel is a necessary addition to the project.

2.2.2 Planned

A task is "Planned" when it's been identified by the group and discussed during an in-person stand-up. For a task to be planned the team must have identified the requirements and agreed upon a rough implementation plan for the task. In the planning phase, a task may be broken up into several more manageable tasks. These tasks will all have the same tag to keep them grouped.

2.2.3 In Progress

A task is "In Progress" when a team member has begun actively working on it. First, the team member should assign themselves to the task, and move the task into the "In Progress" column. If this is the first task of a particular feature tag being started then the team member should create the feature branch by fetching the latest changes to master from git on our fork. Then they should create a new feature branch off of master, where the changes will be implemented. If the task is not the first task for a given tag, instead the developer should branch off of an existing task for that tag. The team member should make sure to follow matplotlib's coding guidelines when developing the feature.

Once the feature and all of the tests have been implemented and tested the developer should create a pull request to the master branch on our fork. The code won't be merged to our master, this step just serves as a good method for identifying how the code on the feature branch differs from the code on master. It also serves as a good way to know if the code is able to be merged to master.

2.2.4 In Review

A task is "In Review" when all of the steps from "In Progress" have been completed and a pending pull request to the master branch (or in the case of a sub-task on the feature branch) has been created. The reviewer should first assign themselves to the task. While reviewing the reviewer should follow matplotlib's reviewers guidelines. The reviewer should pull the feature branch and functionally, test the changes on their computer and verify that the feature satisfies the requirements of the task. They should then read through the code diff on the pull request and identify any inconsistencies, errors or potential bugs in the code. The reviewer should run matplotlibs automated test suites and ensure the Travis, Appvyor, circle, and codecov tests are passing. If there are any issues with the review, feedback should be given in the pull request and the task should be moved back to the "In Progress" column.

Once all issues are resolved the reviewer should indicate that the code is ready. The pull request should **not** be merged into the master branch of our fork, the purpose of this pull request is simply to identify changes in the code. If all of the tasks for a given feature are complete, Then a developer should create a pull request from the feature branch to the trunk of matplotlibs repository and archive the existing request. These steps are taken to better align our workflow with the development workflow of matplotlib.

2.2.5 Done

A task is done when it has finished review and it's been merged into the main feature branch in our fork. The final task for a feature is considered done when a pull request from the feature branch on our fork, to the matplotlib trunk, has been created. If further changes are required the task is moved back to "In Progress".

2.2.6 Released

A task is released when it is merged into the main matplotlib git repository. Yay!!

2.3 Advantages of Kanban

Kanban doesn't impose strict deadlines or commitments on the members. All four of our members have full course loads of upper-year courses so we all have variable amounts of availability to work on the project. Kanban's deadlines are those defined by the business (in our case, course deliverables), and we consider this a major advantage of this process.

Kanban keeps all team members well informed on the progress of the team. Kanban is a visually-oriented process, utilizing the Kanban board as a real-time informational display of the team's progress. With the consistent context switching our members will be doing the board will serve as a solid foundation for our communication.

Kanban is a process that has some similarities to scrum, scrum is a process that we've all had experience working within both professional and academic settings so we feel that adjusting to Kanban would enable the team to reach our potential the fastest.

2.4 Disadvantages of Kanban

Kanban relies heavily on the project board for effective information dispersal. If the board is not consistently updated it can lead to communication issues where multiple people are working on the same task, or a task is blocked for some artificial reason.

Our implementation of Kanban isn't using estimates, and we are working continuously with our only deadlines being those determined by the course deliverables. For this reason, it will be challenging to predict the future progress and output of our team.

2.5 Other Software Development Methodologies Considered

2.5.1 XP

We did not chose Extreme Programming as our methodology because it requires a highly dedicated team of developers, who are all very proficient in the language and the product being developed. Since all of our team members are taking full course loads, we cannot devote more than a few hours a week to working on the project. Furthermore, none of us have contributed to matplotlib before, so omitting specification documents in favor of acceptance tests and coming up with a system metaphor to communicate instead of a formal architecture would be unreasonably complicated.

Additionally, XP relies heavily on user input from the client throughout the development process. Since there is no real client and we have no guarantee the matplotlib community will respond promptly, we would also be adding unnecessary external dependencies into our workflow.

2.5.2 Waterfall

We considered waterfall as our methodology because of its emphasis on spending a long time on planning and designing the solution. We are all new to working with matplotlib, so we agreed it would be a good idea to spend time understanding its architecture well before starting any new features. We also liked the idea of having clear and thorough requirements documents for our features. However, there were also aspects that we didn't find helpful like the fact that it wasn't iterative, so if we find out during the development phase that something isn't technically possible, we'd have to break the practice and make a redesign. Also, we did not want to leave testing until the very end since we all strongly believe that each developer should write full unit and integration tests before pushing their code.

Therefore in our chosen methodology we kept some key parts of waterfall like using thorough design and specification documents, but added iterative design and development, all of which we ultimately incorporated into Kanban.