CSCD01: Deliverable 1

Team Name: //TODO

Table of Contents:

Commentary on Architecture:

matplotlib code architecture consists of three layers: Backend; Artist; and Scripting. The layers stack on top of one another so that as you move up, each layer cannot see the implementation of the layers beneath it but can contribute to the implementation of higher layers.

The Backend Layer:
The Backend layer is the first and lowest layer. It establishes abstract classes and their various implementations. The goal of this layer is to provide the core of matplotlib with an implementation that allows it to work in interactive (eg. Qt5) and non-interactive (eg. PDF) formats. It primarily contains the FigureCanvas, Renderer, and Event, with their related methods and functions. The FigureCanvas acts as the 'canvas' that the images and graphs are 'drawn' onto. The Renderer is used by the Artist layer to draw the necessary shapes and images on the canvas. Lastly, the Event listens for user events such as a mouse click and sends the event information upstream so that it can be handled.

The Artist Layer:
The Artist layer is the second layer. It sits on top of the Backend layer and uses renderer instances to draw the graphs and images. Everything drawn on the FigureCanvas is an instance of the Artist object. There are two groups of Artists: primitives and containers. Where primitives are the objects that can be seen in a graph (such as Line2D and Rectangle), containers are collections of Artists (such as Figure and Axes). For each Figure there can be multiple Axes, but each Axes can only have one Figure. This relationship is founded on the fact that a Figure represents the image while the Axes are individual plots within the image.

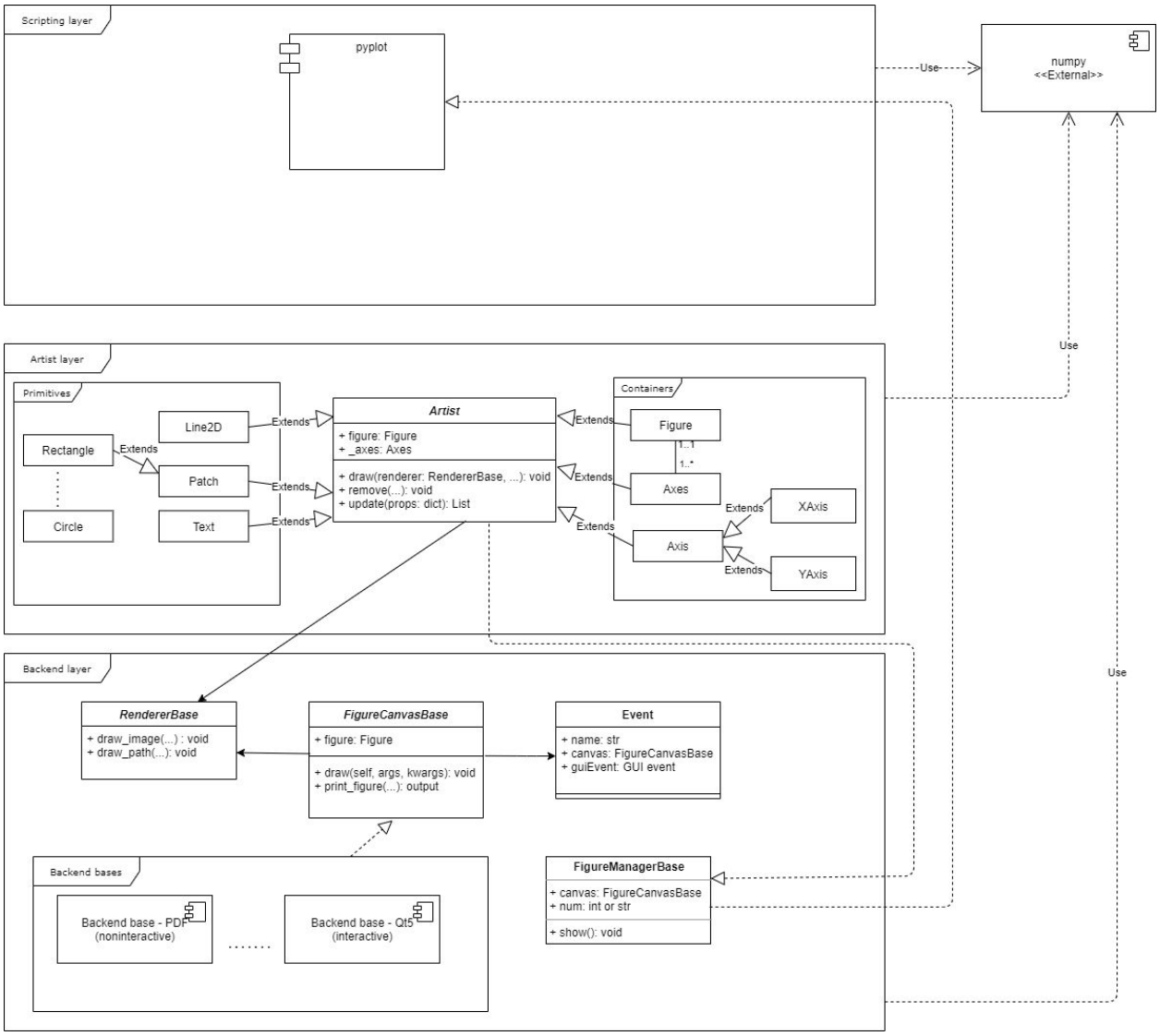
The Scripting Layer:
The Scripting layer is the third and final layer. It sits on top of the Artist layer and provides the various elements of the Artist layer to the user through a pyplot state-based interface. The user can avoid using the pyplot interface (and therefore the scripting layer) by using the object-oriented API instead. Which API the user uses depends on the complexity of their plots. pyplot is primarily used for simple graphs, while more complex graphs will require the object-oriented interface. If a user is more familiar with the object-oriented interface they can avoid using pyplot altogether - anything pyplot can do can also be done through the object oriented interface. However, even when using the object-oriented interface pyplot is often still used to create a figure, for simplicity. In addition, pyplot is considered easier and simpler for new users; however, it is more rigid and limited than the object-oriented approach. Two primary methods provided by pyplot are *plot* and *show*. *plot* and *show* are both used to render the created graph. While *plot* is used for general graph plotting, *show* is used to force the graph to render in non-interactive mode.

Overview:

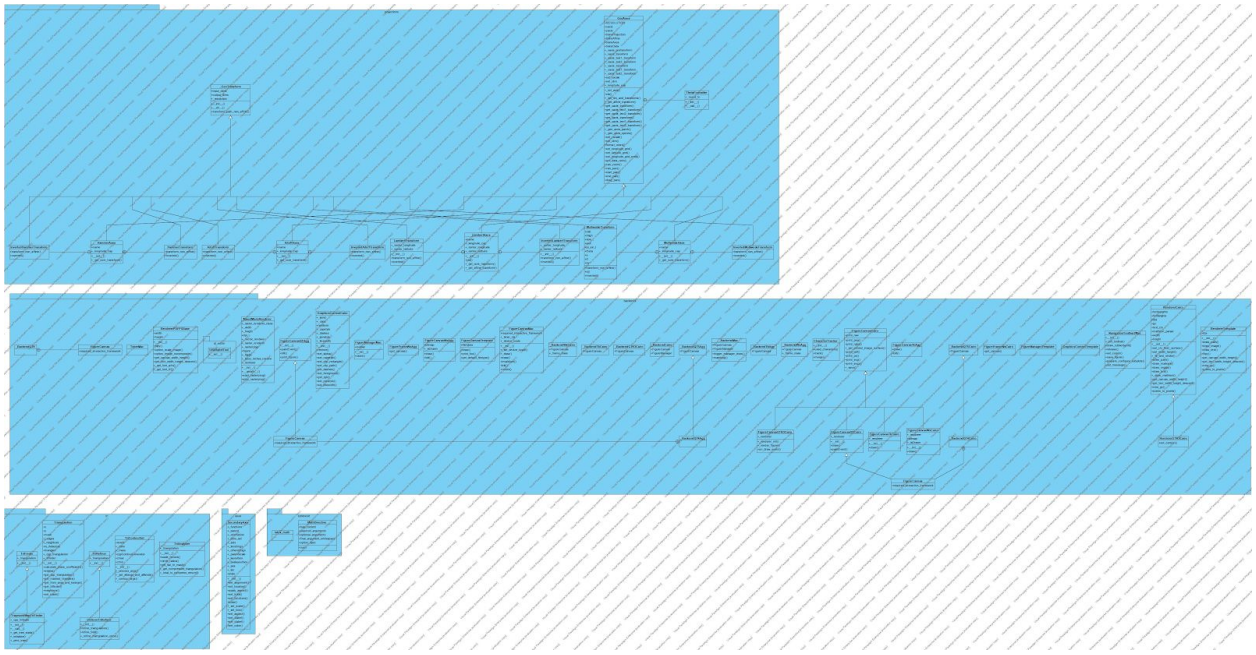
The three-layer encapsulation architecture applied by matplotlib separates the code by function. The modular design creates high-cohesion and low-coupling; making the system easier to understand, maintain, and update. The low-coupling also makes it easier to test individual parts of the system in isolation, since the interdependence between modules is low. This is especially beneficial for open source projects since multiple contributors need to be able to work on the code without conflicting with one another. High cohesion improves modular re-usability decreasing future work by allowing contributors to reuse pre-existing code. In addition, the high-cohesion reduces complexity by reducing the operations in each module - another important feature for open source projects, where new contributors need to spend time understanding the existing code. Our team finds the three-layer architecture used by matplotlib to be a good choice for an open source project of its size. The modular design provides multiple benefits that make contribution to and maintenance of the project easier.

UML models

High-level UML:



Reverse engineering generated UML:



Software Development Process:

Our team chose the Kanban development process for our next deliverable. We will use Trello as our Kanban board and trade out daily meetings for daily asynchronous updates via the group chat on WhatsApp. This will allow us to keep team members up-to-date without having to set aside the time for daily meetings. Our Kanban board will have seven columns: 'To Do', 'In-progress', 'Ready for Review 1', 'In Review 1', 'Ready for Review 2', 'In Review 2', and 'Done'. The 'In-progress' column will have a limit of five tasks at any given time - one per team member. Alongside our daily updates in the chat we will have two weekly planning meetings to adjust our schedule and task priorities for the deliverable on an as-need basis. In order to keep our documentation up-to-date we will add the requirement that documentation must be updated at the end of each task before the task can be moved to the 'Ready for Review 1' column. Furthermore, all testing must be completed by the developer prior to moving a task to the 'Ready for Review 1' column. The first reviewer will move the task to the 'In Review 1' column. When completed, they will move the task to the 'Ready for Review 2' column. In the event that a second reviewer wants to simultaneously review the task, they will move the task directly to the 'In Review 2' column. When a reviewer completes a review, they will add their initials to the task title so that it is visually clear that the review is complete. The changes, in code, testing, and documentation, must be checked off by two other team members before the task can be moved from the 'In Review 2' column to the 'Done' column. Tasks cannot be moved back to the 'In-progress' column if a reviewer fails the task on their review. Instead the task will be moved back to the 'Ready for Review 1' column with the phrase 'Re-do' added to the task title. When the necessary changes have been made to the task, the developer will remove the 'Re-do' tag from the task title, letting team members know that it is available for review. Following traditional Kanban guidelines we will not be assigning any special roles to team members for the next deliverable, but will continue working together in our pre-existing team structure.

Our team chose the Kanban development process for multiple reasons. Kanban is easy to implement and requires no special training or practice from the team. This is essential since we do not have a lot of time to spend on learning a new process. As a small team working on a short-term project, we need a flexible development process that will allow us to make short-notice changes to schedule on an as-need basis. Additionally, Kanban limits work in-progress and provides a visual way to track team progress on each task. Both of these features are important to our team since we do not want to spend too long on any one task. Kanban will help us to quickly identify bottlenecks, allowing us to address them earlier. Limiting the number of in-progress tasks will help our team stay organized and make the most of the time we have by ensuring that we complete tasks in order of assigned priority.

We made two main modifications to the traditional Kanban process. The ‘Review’ columns lets us place greater emphasis on the importance of good documentation and organized code. Matplotlib has strict coding, testing, and documentation guidelines that we need to follow. When an item is in one of the ‘Review’ columns it provides team members a chance to double check that all of our code is up to the Matplotlib standard improving our chances of having our code accepted. The last change we made was to switch out daily meetings for asynchronous daily updates - this was done primarily to save time. Additionally, as a team of students who are not working 9-5 on the project we are unlikely to all have updates every single day.

A downside of using Kanban is the flexibility inherent in the process will mean that our team will have to spend a significant amount of time on planning. Where more rigid development processes would allow us to follow the specified steps, Kanban will require us to take initiative in planning and re-planning our next steps. Team members will have to keep the board up-to-date in order to make use of many of Kanban’s advantages. This is another potential time sink, since team members must always remember to refer back to the board with any updates.

Waterfall is a good process to use when the requirements are clearly defined in advance. It has the benefit of being easy to implement and understand since it follows a clear flow. Unfortunately, it is not well designed for repeating steps - phases are not easy to revisit. This makes it unwieldy for projects with poorly-defined requirements. Since this is our first open-source contribution we did not feel that the rigid structure of Waterfall’s phases would be best suited for our next deliverable.

Incremental Delivery is good for pushing frequent updates. This has the benefit of frequent feedback from users and is good for dynamic markets. The regular user feedback creates a system of continuous testing, allowing bugs to be found early on. It can be very difficult to manage however and requires a lot of work. Since we are not dealing with the end-users directly (and cannot deploy to them) and do not expect frequent feedback from the Matplotlib developers we did not feel that Incremental Delivery fits the project. In addition, we are fixing bugs in the next deliverable which does not leave us with a lot of room to deliver in increments - a bug fix should be delivered all at once, not in pieces.

Boehm’s Spiral Model is good for analysing and managing risks. It has the benefit of identifying the largest risks early on, reducing the overall risk as the project progresses. The Spiral Model is well-suited to critical systems or new markets. Given the low-risk nature of fixing bugs we did not feel that Boehm’s Spiral Model is the best process for the next deliverable.

Rational Unified Process is good for use with the Rational tool designed by IBM. It has the benefit of having a well-documented tool specifically designed for this process. The iterative nature of the process provides a great deal of flexibility while the tool helps to keep everything organized. However, the Rational tool is difficult to learn and we did not feel that our time would be well-spent learning to use it. Rational is also designed with a business model in mind. Since the next deliverable is focused on bug-fixing, this means we would need to heavily modify the Rational Unified Process to better suit our project.

Extreme Programming is good for keeping the business and development teams separated and ensuring they work together to meet the overall goals of the company. It has the benefit of rapidly reducing risk by moving the riskiest features forward. However, Extreme Programming is difficult to implement due to it's very rigid design. Given that we don't have a business team or clients we can't follow the rigid design. Furthermore, the daily integration and testing required by Extreme Programming is not feasible for our team.