SOEN 6471

ADVANCED SOFTWARE ARCHITECTURE

SUMMER 2023

Deliverable 2

Declaration

We, the members of the team, have read and understood the Fairness Protocol and the Communal Work Protocol, and agree to abide by the policies therein, without any exception, under any circumstances, whatsoever.

Group D

Jigar Maheshbhai Borad
Bhavye Budhiraja
Rancy Chadha
Payal Raj Chaudhary
Raviraj Bhaveshbhai Savaliya

Table of Contents

Problem 4

Matplotlib architecture aims to satisfy the following quality attributes:

4.1.1 Accessibility

Matplotlib aims to provide accessibility in terms of its documentation and ease of use. With Matplotlib, developers can create accurate plots based on amounts of data. When analyzing these data plots, good developers will make it easier to see patterns and trends in the data sets. Thus, Matplotlib simplifies data, making it more accessible [1].

4.1.2 Customizability

Matplotlib provides users with different levels of customizability. Users can use the scripting layer for easy visualization with default settings, or they can directly call objects in the artist layer for more control and creativity.

4.1.3 Modularity

Matplotlib follows a modular design by organizing objects into backend, artist, and scripting layers. This modular structure allows for easy addition and reusability of code, as modules within each layer can be developed independently.

4.1.4 Dependability

Matplotlib aims to be dependable by providing stable functionality and reliable performance. It has a large community of contributors who actively maintain and enhance the library, ensuring that issues are addressed promptly.

4.1.5 Maintainability

Matplotlib focuses on maintainability by following good software engineering practices, adhering to modularity, and providing extensive documentation. It has a well-defined architecture that separates concerns and allows for easy maintenance and updates.

4.1.6 Performability

Matplotlib aims to deliver efficient performance for typical visualization tasks. It includes optimizations to minimize rendering time and memory usage, enabling users to handle large datasets efficiently.

4.1.7 Portability

Matplotlib aims to be portable by ensuring compatibility across different platforms and operating systems. It is designed to work seamlessly in various environments, allowing users to develop and run their visualizations consistently.

Matplotlib should strive to address the following quality attributes, but they may not be explicitly satisfied by its current architecture:

4.2.1 Affectability

While Matplotlib provides customization options, it does not explicitly incorporate design choices aimed at evoking emotional responses from users. However, through careful customization of visual elements such as colors, styles, and annotations, developers can create visually appealing plots that may have an emotional impact on users[2].

4.2.2 Simplicity

The ability to use the scripting layer for quick and straightforward visualization contributes to the simplicity attribute. However, it's worth noting that Matplotlib's API[3] can be overwhelming for beginners due to its extensive functionality and the learning curve associated with mastering the library. While efforts have been made to improve documentation and provide comprehensive examples, further simplification of the API and enhancing user onboarding experiences could be areas of improvement.

4.2.3 Ethicality

Matplotlib itself does not enforce specific ethical guidelines. However, ethical considerations in data visualization, such as proper handling of sensitive data, accurate representation of information, and avoiding biased or misleading visualizations, are the responsibility of the users and developers utilizing Matplotlib.

4.2.4 Legality

Matplotlib's architecture does not explicitly address legality. Compliance with legal regulations, such as data protection laws, copyright, or licensing requirements, is the responsibility of the users and developers incorporating Matplotlib into their applications. It's important to ensure that data and content used with Matplotlib comply with applicable legal regulations.

4.2.5 Resilience

Matplotlib's architecture does not explicitly address system-level resilience. As a plotting library, its primary focus is on providing visualizations and not on system-level robustness or fault tolerance. The library's stability and robustness primarily rely on the underlying Python runtime environment and the integrity of the data provided by the users.

4.2.6 Safety

Matplotlib's architecture does not specifically address safety concerns. Since Matplotlib primarily deals with data visualization, it does not perform safety-critical operations or interact with physical systems where safety considerations are paramount. However, developers should ensure the safety of their systems by implementing appropriate data validation and sanitization practices when using Matplotlib in conjunction with other components.

4.2.7 Scalability

Matplotlib aims to handle a wide range of data sizes efficiently. However, the scalability of Matplotlib can be limited by hardware constraints, such as memory limitations or rendering capabilities. For extremely large datasets or high-performance requirements, specialized libraries or techniques, such as data sampling or distributed computing, may be necessary to achieve optimal scalability[4].

4.2.8 Privacy

Privacy is not directly addressed by Matplotlib's architecture. As a library, it does not interact with or store user data directly. However, developers using Matplotlib need to ensure the privacy of any data they utilize in their visualizations.

4.2.9 Usability

Matplotlib aims to be usable by providing a comprehensive set of functions and options for creating plots and visualizations. However, its flexibility and extensive configuration options can sometimes make it more challenging for beginners. Providing higher-level abstractions or simplified interfaces could improve usability for novice users.

Problem 5

Matplotlib's architecture [5] can be understood by looking at it from various viewpoints and perspectives that help us know more about its structure and its specification. Before we dive into a few views out of the several views, it is imperative to know what concerns are being addressed by representing this library from given views. One of the important concerns is that what motivated or what were the use cases that lead to Matplotlib's architecture, it's also worthwhile to explore what components are vital for providing visuals that Matplotlib is known for and of course how these components interact to finally produce a visual. Based on these concerns, we present below given views:

5.1 Logical View

The Logical View of Matplotlib, helps stakeholders and developers understand the composition and relationships between key components of Matplotlib's architecture. As mentioned in [6] this view is responsible for the conceptual organization of layers and high-level functionality of components in each layer. Matplotlib as we are aware follows layered architecture [5]. The three layers are Scripting Layer, Artist Layer and Backend layer. Each layer that sits above another layer knows how to talk to the layer below it, but the lower layer is not aware of the layers above it. The three layers from bottom to top are: backend, artist, and scripting. The details of these layers are given in Section 6A.1.1 Layered Architecture Pattern. Thus, the implementation view is best suited to get insight into how the different layers are implemented and how they interact with each other. The specific classes and modules within each layer may vary based on the implementation details of Matplotlib.

We will represent this view using package view as given below (Artist Layer shows only main packages however there are more packages that are not shown):

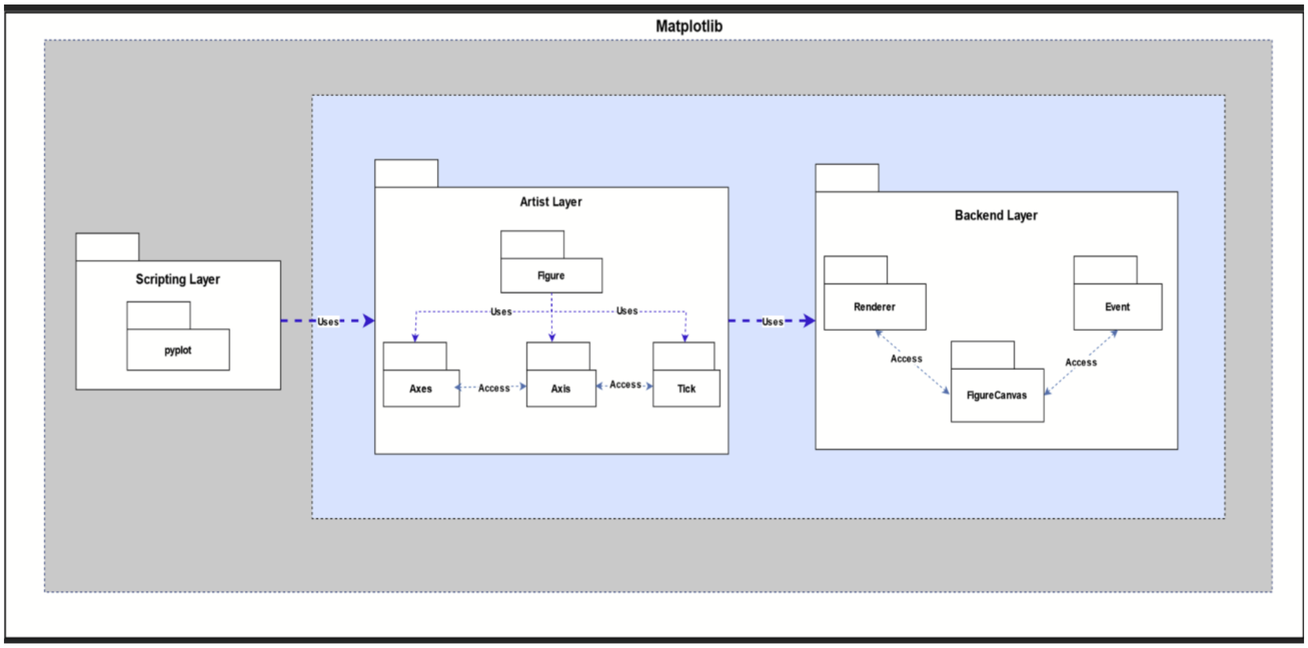


Fig 1: Package Diagram of Matplotlib

5.2 Implementation View

The Implementation View focuses on the actual source code, data files, and executables of Matplotlib. Matplotlib as we are aware follows layered architecture [5]. Thus, the implementation view is best suited to get insight into how the different layers are implemented and how they interact with each other. The specific classes and modules within each layer may vary based on the implementation details of Matplotlib.

However, before we look at the Implementation View, it is crucial to know about these key components of Matplotlib that work together to facilitate the creation, customization, and rendering of plots and visualizations in Matplotlib.

5.2.1 Pyplot

Pyplot[7] is a module in Matplotlib that provides a high-level interface for creating and managing figures, axes, and other plot elements. Pyplot allows users to create plots, customize plot elements, add labels, titles, legends, and annotations, and save or display the plot.

5.2.2 Backend

The backend[8] module in Matplotlib contains the implementations of different backends, which are responsible for rendering the visualizations. Matplotlib supports various backends, including Agg, TkAgg, QtAgg, GTKAgg, and more. Each backend provides a different mechanism for displaying the plots. It provides concrete implementations of the abstract interface classes FigureCanvas, Renderer and Event.

5.2.3 Artist

The artist[9] module in Matplotlib contains classes representing different graphical elements that can be rendered in plots. These elements include lines, text, patches (such as rectangles, circles, polygons), images etc. Each artist class provides methods and attributes for configuring and customizing the appearance of the graphical elements.

5.2.4 Axes

The axes[10] module allows users to control the visual aspects and layout of individual plots within a figure. The axes module in Matplotlib contains classes for managing the plot axes and their properties. Axes represent an individual plot within a figure and provide methods for creating and modifying plot elements, such as lines, markers, labels, grids, and more.

5.2.5 Figure

The figure[11] module in Matplotlib contains classes for managing the overall figure, which serves as the top-level container for all the plot elements. Figures can contain one or more axes, allowing users to create multiple subplots or arrange multiple plots in a grid-like structure. The figure module provides methods for creating, configuring, and saving figures. The figure module acts as the container that holds the entire plot and provides control over the global properties of the visualization. It talks to Renderer and Events class too. Now that we know about components, we can look at the implementation view of Matplotlib based on how these components relate to each other:

Pyplot serves as the interface for users to interact with Matplotlib's plotting capabilities, while the Backend handles the rendering process based on the selected backend. Artists, Axes, and Figure components provide the means to customize the appearance, structure, and organization of the plots. Together these components work to render the final visual/plot on the user's chosen backend. Given below is the class diagram of matplotlib's classes/components and their relationship to each other. Please note that the class diagram shows limited and key classes, since it is quite exhaustive list.

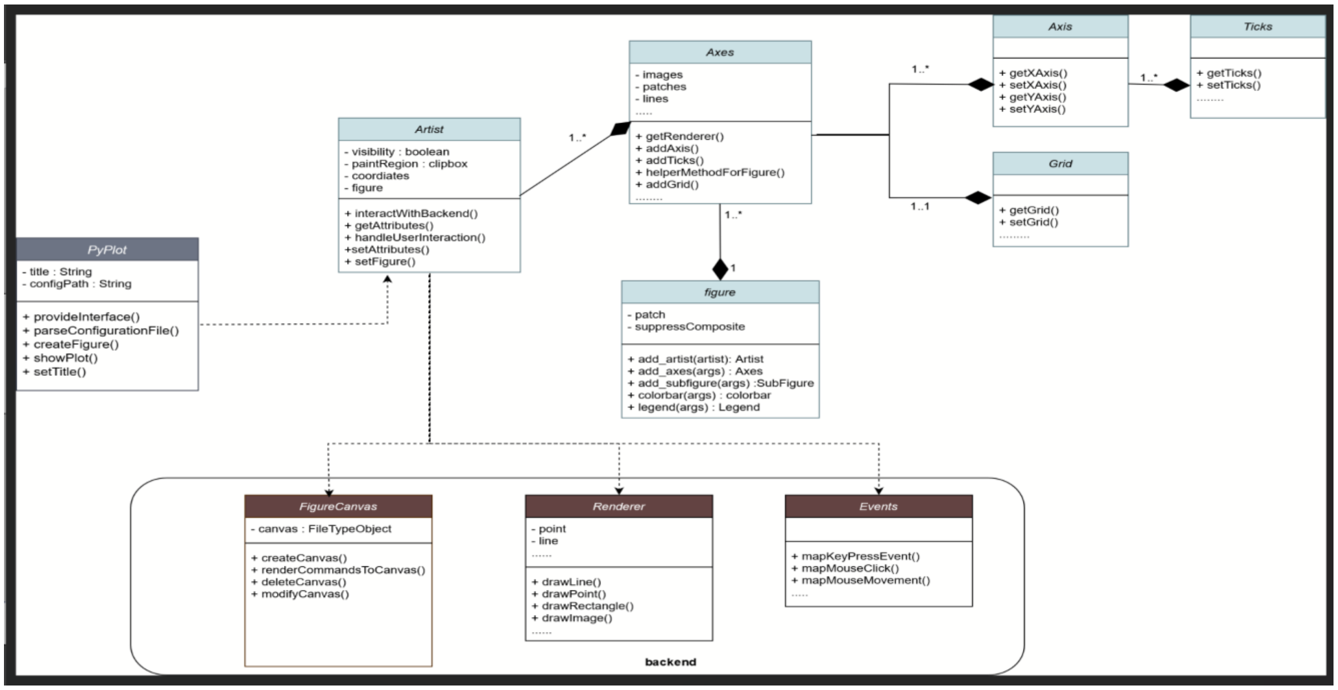


Fig 2: Class Diagram of Matplotlib

5.3 Process View

This view focuses on the dynamic aspects of Matplotlib, including the behavior, interactions, and communication among layers. It emphasizes on the runtime behavior of the system and showcases the interaction between layers from the point of the user providing data to matplotlib to display the plot. It helps visualize the flow of control and the sequence of messages exchanged between components during system execution. A sequence diagram is commonly used to represent these interactions:

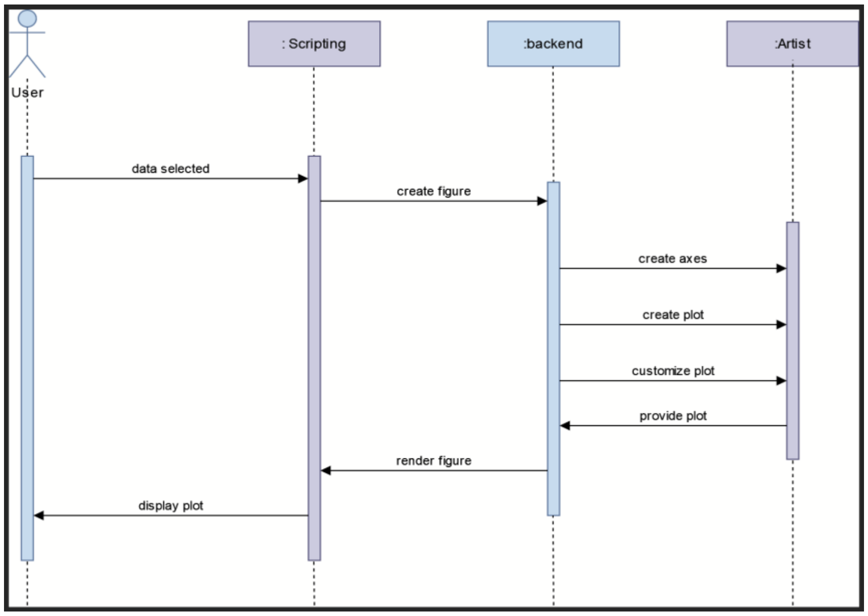


Fig 3: Sequence Diagram of Matplotlib

In this sequence diagram, the User component initiates the interaction by providing data to plot the visual(step 1). The scripting layer receives the data and invokes Backend layer to create figure(step 2). Then, the Backend invokes the Artist Layer (step 3) to create axes, create and customize and provide plot. The Artist Layer performs the required updates and renders figure(step 4). Finally, Matplotlib completes the interaction and returns the result to the User.

The '4+1' view model of software architecture [6] and this view model conforms with the ADF specifications defined in ISO/IEC 2022 [12]. This viewpoint model consists of 5 views: Logical View, Implementation View, Process View, Deployment View and Usage View. Above given views correspond to the 4+1 Architecture viewpoint model.

Problem 6A

6A.1 Patterns

Matplotlib, being a complex software library, utilizes various patterns, principles, styles, and tactics of software architecture to achieve its goals. Here are some architectural concepts that can be observed in Matplotlib and their applications:

6A.1.1 Layered Architecture Pattern

Layered architectures[13] are said to be the most common and widely used architectural framework in software development. It is also known as an n-tier architecture and describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software. Below are the layers in matplotlib architecture.

6A.1.1.1 Scripting layer

The scripting layer is the topmost layer on which most of our code will run. The methods in the scripting layer almost automatically take care of the other layers, and all we need to care about is the current state. Facade pattern is used by this layer. Scripting layer allows interaction with key components of a system through a common interface that lies in pyplot. Scripting layer is a procedural visualization library in that we tell the underlying software which drawing action we want in order to render our data.

6A.1.1.2 Artist layer

The artist layer is the second layer in the architecture. It is responsible for how data is arranged. It handles various plotting functions, like axis, which coordinates on how to use the renderer on the figure canvas.

6A.1.1.3 Backend layer

The backend layer is the bottom layer, which consists of the implementation of the various functions that are necessary for plotting. There are three essential classes from the backend layer **FigureCanvas**(The surface on which the figure will be drawn), **Renderer**(the class that handles rendering or drawing on the canvas), and **Event**(It handles the mouse and keyboard events).In this layer, actual drawing is done.

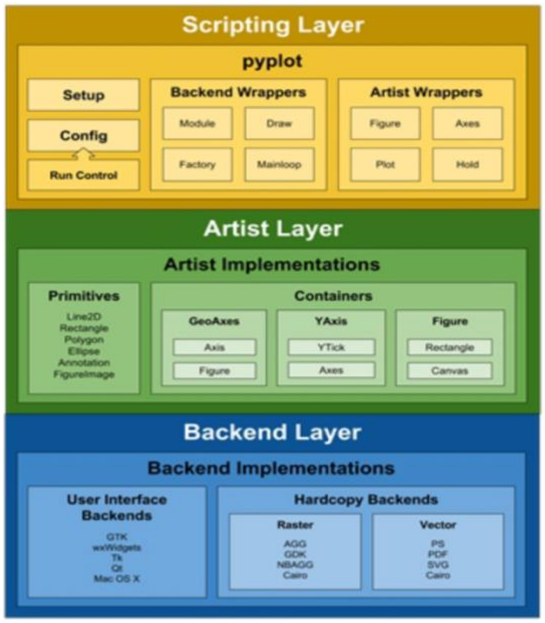


Fig 4: Layered Architecture of Matplotlib

6A.1.2 Event-Driven Architecture

Matplotlib employs an event-driven architecture[14], where user interactions (events) trigger actions and updates in the plot. The library provides event handlers and call backs to respond to events, enabling interactivity and real-time updates.

6A.1.3 Model-View-Controller (MVC) Pattern

Matplotlib adopts the MVC pattern[15], separating the plot data and objects (model) from the visual representation (view) and user interactions (controller). This promotes code modularity, flexibility, and easier maintenance.

6A.1.3.1 Model

The model[16] encapsulates some application data and the logic that manipulates that data, independently of the user interface. It responds to requests for data about its state (usually from a view). It responds to instructions to change state (usually from a controller). There is usually a single model.

6A.1.3.2 View

The view[16] renders to the user a specific portion of the data from the model into a suitable modality (say, visual). The 'user' may be a human or a machine. There can be multiple different views for different purposes.

6A.1.3.3 Controller

The controller[16] receives user input and translates it into a request to the model. (A ‘request’ could, for example, be making calls on appropriate model objects, or instructing the model to perform certain actions.) There can be multiple different controllers for different purposes.

6A.2 Principles

6A.2.1 Single Responsibility Principle

Single responsibility[17] principle is a relatively basic principle that is used in matplotlib to build code. It can be applied to classes, software components, and microservices. Utilizing this principle makes code easier to test and maintain, it makes matplotlib easier to implement, and it helps to avoid unanticipated side-effects of future changes.

6A.2.2 Open Closed Principle

The idea of open-closed principle[2] is that existing, well-tested classes will need to be modified when something needs to be added. Yet, changing classes can lead to problems or bugs. In matplotlib this is used so that instead of changing the class, you simply want to extend it. Following this principle is essential for writing code that is easy to maintain and revise. Class complies with this principle if it is:

- Open for extension, meaning that the class’s behavior can be extended; and
- Closed for modification, meaning that the source code is set and cannot be changed.

6A.2.3 The Liskov Substitution Principle

Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it [20]. Matplotlib uses this principle to avoid unexpected consequences of changes and avoids having to open a closed class in order to make changes.

6A.2.4 The Interface Segregation Principle

The general idea of interface segregation principle[19][18] is that it’s better to have a lot of smaller interfaces than a few bigger ones. Fine grained interfaces that are user specific. Smaller interfaces mean that while developing the software should prefer composition over inheritance and for decoupling over coupling.

We have observed that in matplotlib it starts with an existing interface and add new methods. Instead, start by building a new interface and then let the class implement multiple interfaces as needed.

6A.2.5 Dependency Inversion Principle

This principle offers a way to decouple software modules. Simply put, dependency inversion principle means “dependencies on abstractions, not on concretions.” [19] In matplotlib, “high level modules do not depend upon low level modules.

One popular way to comply with this principle is using a dependency inversion pattern, although this method is not the only way to do so. Whatever method is chosen to utilize, finding a way to utilize this principle will make your code more flexible, agile, and reusable.

6A.3 Styles

Matplotlib provides a variety of built-in styles that you can use to customize the appearance of your plots. These styles define the colors, line styles, fonts, gridlines, and other visual aspects of your plots. Here are some popular Matplotlib styles:

6A.3.1 Default Style

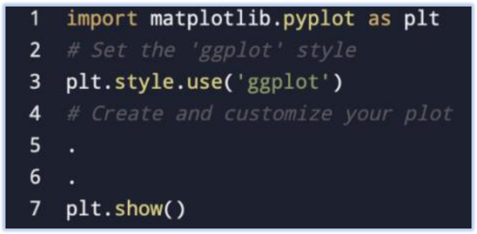
This is the default style[21] used by Matplotlib. It provides a clean and simple look for plots. It can be used by not specifying any style.

6A.3.2 Classic Style

This style emulates the classic Matplotlib[21] look from earlier versions. It has a slightly different colour palette and a more traditional plot appearance.

6A.3.3 Ggplot Style

This style mimics the aesthetics of the popular GGplot library[22] in R. It provides a clean, modern look with grey backgrounds and colored lines. To use a specific style, plt.style.use() function can be used. For example:



```python
import matplotlib.pyplot as plt
# Set the 'ggplot' style
plt.style.use('ggplot')
# Create and customize your plot
.
.
plt.show()
```

6A.3.4 Seaborn Style

This style is based on the Seaborn library[22], which provides enhanced statistical plotting capabilities. It offers a visually pleasing look with subtle colors and improved default settings.

6A.3.4 Fivethirtyeight Style

This style replicates the visual style of the plots seen on the FiveThirtyEight website.[22] It is characterized by bold colors, thick lines, and large fonts.

6A.3.5 Dark Background Style

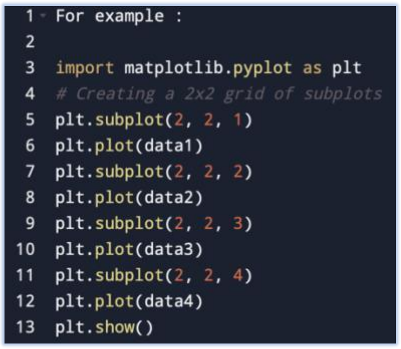
This style[21] sets a dark background with light-colored lines and markers. It is useful for creating plots that are visually appealing in dark or low-light environments.

Problem 6B

While Matplotlib is a widely used and respected plotting library, there are some potential undesirables, including anti-patterns and code smells, that can be observed in its software architecture. Here are a few examples:

6B.1 Anti Pattern[24]

I. **Overusing plt.subplot()**: One antipattern in matplotlib is the excessive use of plt.subplot() function to create multiple subplots within a figure. While subplots can be useful for organizing multiple plots, it becomes an antipattern when used excessively or unnecessarily. This can lead to cluttered and confusing visualizations, making it difficult to interpret the data.



```python
For example :

import matplotlib.pyplot as plt
# Creating a 2x2 grid of subplots
plt.subplot(2, 2, 1)
plt.plot(data1)
plt.subplot(2, 2, 2)
plt.plot(data2)
plt.subplot(2, 2, 3)
plt.plot(data3)
plt.subplot(2, 2, 4)
plt.plot(data4)
plt.show()
```

II. **Using Pylab**: The use of the pylab module, which imports functions from both Matplotlib and NumPy into the global namespace, is generally discouraged. It can lead to namespace pollution and conflicts. Instead, it is recommended to use the object-oriented interface or the pyplot interface directly.[23]

III. **Ignoring Figure and Axes Objects:** Matplotlib provides Figure and Axes objects that allow for precise control over plot elements. Ignoring these objects and directly manipulating the global state or relying solely on the stateful pyplot interface can lead to code that is difficult to maintain and understand. It's recommended to leverage the power of the object-oriented interface for more flexibility and modularity.

IV. **Neglecting labels and titles:** Omitting axis labels, plot titles, and legends can make the visualization less informative and harder to interpret. Always include relevant labels and titles to provide context and clarity. Use plt.xlabel(), plt.ylabel(), plt.title(), and plt.legend() functions to add these elements to your plot.

6B.2 Smells[17][25][28]

Category	**Description**
Organization and Structure Smells	• Mixing different plot creation styles (object-oriented and pyplot interfaces) inconsistently. • Lack of modularity, resulting in long and complex functions for creating and customizing plots. • Duplicated code segments for similar plot configurations or customizations. • Poor naming conventions for variables, functions, or classes related to plots.
Configuration and Customization Smells	• Excessive use of magic numbers or hardcoded values without proper documentation. • Overcomplicated plot configurations with unnecessary complexity. • Ignoring or not utilizing Matplotlib's APIs and functionalities for customization. • Not leveraging the power of styles and themes provided by Matplotlib.
Data Handling and Integration Smells	• Manual data conversions instead of using the capabilities of NumPy or Pandas integration with Matplotlib. • Insufficient error handling for incorrect or unsupported data formats. • Lack of data validation and checks before plotting.
Documentation and Comments Smells	• Inadequate or missing documentation for plot creation, customization, and usage. • Lack of inline comments or explanations for complex or non-obvious code segments.

	• Absence of explanations for the purpose or context of plot-related variables or configurations.

6B.3 Other undesirables[24][25]

I. **Steep learning curve:** Matplotlib has a relatively steep learning curve, especially for users who are new to data visualization or have limited experience with Python. Its syntax and object-oriented structure can be initially challenging to grasp, and users may need to invest time in understanding its intricacies and various customization options.

II. **3D plotting limitations:** While matplotlib does allow 3D plotting, there are certain performance and interactive constraints. Using specialized 3D visualization frameworks like Mayavi or Plotly can be more convenient and effective than creating sophisticated 3D visualisations or working with enormous datasets, which can be computationally expensive.

III. **Performance concerns:** While matplotlib is a versatile library, it may not always provide the best performance for certain types of visualizations or large datasets. Users working with big data or requiring highly efficient rendering may explore alternatives such as Plotly or Datashader.

Problem 7

Matplotlib does have a robust set of functionalities, but there are areas where the software architecture and software architecture description of matplotlib can be improved. Here are some reasoned-based[28] suggestions for enhancing its architecture and documentation:

7.1 ISSUE: Lack of Modularization and Separation of Concerns

The functionality[27] in Matplotlib is combined into a monolithic package. Lack of modularization makes it difficult to understand, maintain, and extend the library**.**

RESOLUTION:

Modularize Matplotlib by breaking down the functionality into smaller, independent modules. Each module should focus on specific plot types or features, promoting code reusability. Clearly define the responsibilities and interactions between modules. Enhance maintainability and extensibility by separating the core plotting engine from the user interface layer and data handling**.**

7.2 ISSUE: Absence of Clear Abstraction Layers

The software architecture lacks clear abstraction layers. Core plotting engine, user interface layer, and data handling are not properly separated.

RESOLUTION

Introduce clear abstraction layers to improve maintainability and extensibility. Separate the core plotting engine from the user interface layer and data handling. Clearly define the interfaces and interactions between different layers. Enable swapping out components and integrating with other libraries or frameworks. Facilitate testing and documentation of individual components.

7.3 ISSUE: Inadequate Documentation

The existing documentation of Matplotlib can be improved in terms of clarity, completeness, and organization. Comprehensive examples, use cases, and best practices are lacking. Insufficient context on the underlying architecture, design decisions, and module interactions.

RESOLUTION

Enhance the documentation to provide clear and comprehensive examples. Include a wide range of use cases and best practices to guide users. Provide more context on the underlying architecture and design decisions. Improve organization and structure of the documentation. Address the needs of both beginners and advanced users.

7.4 ISSUE: Inconsistent Api Design

The API design[27] in Matplotlib can be complex and inconsistent. Function names, parameters, and method signatures vary across different modules.

RESOLUTION

Refine the API design to ensure consistency in function names, parameters, and method signatures. Follow Python's style guidelines (PEP 8) for naming conventions and code formatting. Simplify and streamline the API to make it more intuitive and easier to use. Provide clear documentation and examples for API usage.

7.5 ISSUE: Performance Optimization

Certain operations in Matplotlib can be computationally expensive. Performance bottlenecks may affect efficiency, especially for large datasets or real-time visualizations.

RESOLUTION

Analyze and optimize critical performance bottlenecks in Matplotlib. Utilize efficient algorithms and data structures for plotting operations. Explore hardware acceleration options, such as GPU utilization, to improve performance. Provide optimizations for handling large datasets and real-time visualizations.

7.6 ISSUE: Lack of Community Engagement and Feedback

Matplotlib could benefit from more active community engagement and feedback channels. Users' suggestions, use cases, and pain points are not effectively incorporated into the library's development.

RESOLUTION

Establish clear channels for community engagement, such as forums, issue trackers, or mailing lists. Actively encourage and collect feedback from the user community.

Implementing these suggestions would require a concerted effort from the matplotlib development team. By addressing architectural shortcomings, refining the documentation, and prioritizing user feedback, matplotlib can evolve into an even more powerful and user-friendly.

Contributions on Deliverable 2

Group D	Contributions
Jigar Maheshbhai Borad	• Gathered information on undesirables in matplotlib. • Researched on smells and anti-patterns • Worked on Problem 6 B • Collaborated with Payal on problem 6 overall. • Reviewed problem 7 and problem 6A
Bhavye Budhiraja	• Research and investigated matplotlib's improvements and potential enhancements for Problem 7 • Worked and prepared content for Problem 7 • Reviewed problem 4 and 5 • Worked on project document formatting as per provided template
Rancy Chadha	• Researched on various viewpoint models and views for problem 5 and Matplotlib architecture • Designed and created UML diagrams for problem 5 • Reviewed problem 4 along with Ravi and collaborated with Payal on Matplotlib layers • Collaborated with Bhavye on Document formatting
Payal Raj Chaudhary	• Researched on patterns, principles and styles of matplotlib • Worked on Problem 6 part A. • Collaborated with Jigar on part B on patterns and anti-patterns • Collaborated with Rancy on matplotlib layers • Reviewed Problem 5 and 7
Raviraj Bhaveshbhai Savaliya	• Gathered information related to quality attributes of Matplotlib architecture • Classified satisfying and aiming to but not satisfying attributes. • Worked on Problem 4 • Collaborated with Bhavye on problem 7 • Reviewed Problem 5 and Problem 7

GitHub Repo Link: https://github.com/Bhavye27/SOEN-6471-Team-D-Matplotlib.git

References

1. [Pankaj Kamthan, 2023] Introduction to Software Product Quality, Available: https://users.encs.concordia.ca/~kamthan/courses/soen-6471/software_product_quality_introduction.pdf
2. [Katriel Ester Amanda, Mihai Anton, Thomas van Tussenbroek, Mick Koertshuis] matplotlib - From Vision to Architecture, Available: https://2021.desosa.nl/projects/matplotlib/posts/essay2-vision-to-architecture
3. Matplotlib API Reference. Retrieved from: https://matplotlib.org/stable/api/index.html
4. [Anand Balachandran Pillai] Architectural Quality Attributes. Retrieved from: https://subscription.packtpub.com/book/application-development/9781786468529/1/ch01lvl1sec11/architectural-quality-attributes
5. [K. Moir] The Architecture of Open Source Applications (Volume 2). Available: https://aosabook.org/en/v2/matplotlib.html
6. [Pankaj Kamthan, 2023] View and Viewpoints of Software Architectures, Available: http://users.encs.concordia.ca/~kamthan/courses/soen-6471/software_architecture_views_viewpoints.pdf
7. Matplotlib PyPlot Reference. Retrieved from: https://matplotlib.org/stable/api/pyplot_summary.html#module-matplotlib.pyplot
8. Matplotlib backends Reference. Retrieved from: https://matplotlib.org/stable/api/index_backend_api.html
9. Matplotlib artists Reference. Retrieved from https://matplotlib.org/stable/api/artist_api.html
10. Matplotlib Axes Reference. Retrieved from https://matplotlib.org/stable/api/axes_api.html
11. Matplotlib figure Reference. Retrieved from https://matplotlib.org/stable/api/figure_api.html
12. ISO/IEC/IEEE 42010, 2011; ISO/IEC/IEEE 42010, 2022
13. [Baeldung] Layered Architecture. Available: https://www.baeldung.com/cs/layered-architecture
14. Matplotlib Event Handling and Picking. Retrieved from https://matplotlib.org/stable/users/event_handling.html#listening-for-events
15. Matplotlib Users Guide. Retrieved from https://matplotlib.org/stable/users/index.html#user-architectures
16. [Pankaj Kamthan, 2023] Software Architecture Patterns, Available: http://users.encs.concordia.ca/~kamthan/courses/soen-6471/software_architecture_patterns.pdf
17. Matplotlib Documentation. Retrieved from: https://matplotlib.org/stable/contents.html
18. Matplotlib GitHub Repository. Retrieved from: https://github.com/matplotlib/matplotlib
19. [Hunter, J. D. (2007)] Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95. DOI: 10.1109/MCSE.2007.55
20. https://2021.desosa.nl/projects/matplotlib/posts/essay2-vision-to-architecture/
21. Matplotlib Style Sheets. Retrieved from: https://matplotlib.org/stable/gallery/style_sheets/style_sheets_reference.html

22. Matplotlib Style Examples. Retrieved from: https://matplotlib.org/stable/gallery/style_sheets/style_sheet_reference.html
23. Matplotlib PyLab. Retrieved from: https://matplotlib.org/stable/api/index.html#module-pylab
24. [Jake VanderPlas] Visualization with Matplotlib. Available: https://jakevdp.github.io/PythonDataScienceHandbook/04.00-introduction-to-matplotlib.html
25. Matplotlib Discussions. Retrieved from: https://github.com/matplotlib/matplotlib/discussions
26. https://link.springer.com/article/10.1007/s11219-008-9068-1
27. Matplotlib Third party dependencies. Retrieved from: https://matplotlib.org/stable/devel/MEP/MEP11.html
28. [Pankaj Kamthan, 2023] The 'Undesirables' Of Software Architecture, Available: https://users.encs.concordia.ca/~kamthan/courses/soen-6471/software_architecture_undesirables.pdf