

CSCD01 - Team Name:

'Scrum Till You Waterfall'

Deliverable 1

Team 13

## Table of Contents

Software Development Process .....	3
Development Processes that were considered .....	3
Waterfall .....	3
Xtreme Programming .....	3
Test-driven Development .....	4
Scrum .....	4
Kanban .....	4
How we will implement the Waterfall Process .....	5
Conclusion .....	5
 Architecture of the System .....	6
Architecture Overview .....	6
Layer Structure Details .....	7
Auto Generated UML .....	8
Overall Structure of the Architecture UML .....	9

## **Software Development Process**

What our group has chosen to follow the **waterfall** process of development for the next deliverable of our project. We came to this decision after considering all the different kinds of processes we could have potentially chosen and their pros and cons, which we illustrate as follows.

### **Development Processes that were considered:**

#### **Waterfall**

##### **The Pros and Cons of using Waterfall**

###### Pros:

- With how short our sprints are, requirements don't change, so we can just plan in the beginning
- We see ourselves planning before the development and not necessarily using the iterative nature of agile development
- We see ourselves completing tasks within a sequence (planning, then testing,.....)

###### Cons:

- If there are any changes in requirements, (i.e. matplotlib decides to restructure their code base), then we have to redo our entire process

#### **Xtreme programming**

##### **The Pros and Cons of using Xtreme programming**

###### Pros:

- Sense of whole team goals - We are in this together and we support each other's work and goals. This will improve each other on the team.
- Encourages sitting together or having more face time leading to higher productivity.
- Good for small teams - we have 5.
- Encourages pair programming to take turns coding and brainstorming every 30 mins.

Cons:

- Requires an informative workspace with stories on a wall. Where do we find this wall so that everyone can see it during meetings?
- Slack - encourages the dropping of stories if getting behind. This should not be done as all the bugs need to be fixed.
- Requires a business side to interact with which we lack in the project.

## **Test-driven Development**

### The Pros and Cons of using Testdriven

Pros:

- For deliverables, we are already required to provide a suite of test cases, so it would be getting part of the deliverable out of the way earlier
- The test suite you prepare provides a specification as to what you are building
- Easy to flush out any misunderstandings before actually touching the code

Cons:

- Tedious to do, especially when the software is complex, and you are unsure of exactly how all the components interact
- Could cost us a lot of initial time with all the mocking and stubbing we would need to perform compared to directly making the functional calls

## **Scrum**

Not considered since we did it in CSCC01

## **Kanban**

### The Pros and Cons of using Kanban

Pros:

- Like the idea of developers pulling tasks from the board as they free up, rather than just being assigned all the tasks at the beginning
- With the board, it is easy to see exactly who is working on what at any given moment (as long as members update the board as they go)
- There's no real pressure to finish a certain task within a certain deadline (other than the overall looming deliverable deadline)

Cons:

- With tickets being pulled as they go, it is hard to forecast when all the tasks will be accomplished, and this could be a significant downfall as all the tasks will probably start being worked on near the deliverable deadline
- There may not be a sense of urgency to complete a task currently being worked on if there is no definitive timeframe
- Having to constantly manage a Kanban board could prove to be a chore especially for a team with few development tasks like this one

### **How We Will Implement The Waterfall Process:**

- We will be using Trello to keep track of our tasks during the planning process
  - Trello Board link: <https://trello.com/b/ldlh1z3l>
- Use the Big-Picture feature of Trello to drag and drop tasks and create dependencies between the tasks
- Gantt chart will be used for scheduling tasks and providing deadlines for each stage of the waterfall process
- We will have planning meetings before the development where each task will be created and assigned to a team member
- All activities will be done in sequence with planning before implementation, and implementation before verification and validation
- Upon completion of a bug/fix or a feature, the entire team will be notified to review the code and get feedback

### **Conclusion:**

In the end, our group carefully considered multiple options and we came to a consensus to use the **waterfall** development process for our CSCD01 project. Due to the upfront requirements being relatively strict, and the short deliverable timelines, we are not expecting too much change throughout a deliverable. Hence, we would like to use waterfall to gather all the requirements initially, design the bug fixes or features, implement, test, and then hopefully release. Waterfall especially makes sense for our team since we are not planning to meet daily or have a daily stand up. We are all understanding of the fact that we all have busy schedules and that the best approach is to split up tasks in the beginning and then set a firm deadline as to when tasks are due. Another reason we chose validation is that especially for our bug fixes, we know that the consumer validation criteria is pretty explicit in the bug tickets themselves, so we are at less risk of implementing something the consumer did not ask for. Also, as we will be working on multiple different bugs for the upcoming deliverable, we will not have to

integrate too much code with other developers in the team and tasks can be worked on more independently. Additionally, since the documentation and architecture of matplotlib are not likely to change anytime soon, the waterfall process would allow our development team to be as efficient as possible in planning what we want to do, how we are going to do it, and validate and test what we developed all within strict workflow to accomplish what we want to do by the required deadline.

## **Architecture of the System**

### **Architecture Overview**

*(refer to “Overall Structure of the Architecture” for the following description)*

The architecture of Matplotlib focuses on creating, updating, and rendering Figure objects. To achieve this, there are 3 different layers, the Backend layer, the artist layer, and the scripting layer.

The **Backend layer** is an abstraction layer that provides logic to the components rendering the Figures. It includes base classes that are in charge of managing the canvas that the figures are drawn on (FigureCanvasBase), handling drawing operations (RendererBase), and handling events.

The **Artist Layer** is responsible for generating plots to display, manipulate, and showcasing data visualization in a user-friendly and accurate way, mainly using the (Artist) class. This layer utilizes different containers and primitive type objects that extend from (Artist) and collection classes that can be used to graphically represent different things within the (FigureCanvasBase) when instantiated.

The **Scripting layer** provides a user interface that allows users to easily interact with the Artist and Backend layer. It provides a syntax that users can use to generate Figures and display the graph on the screen.

## **Layer Structure Details**

The following are notes of the dependencies and class' coupling for each of the three layers.

Based on notes from "McGregor, D. M. (2015). *Mastering matplotlib*. Birmingham: Packt Publishing."

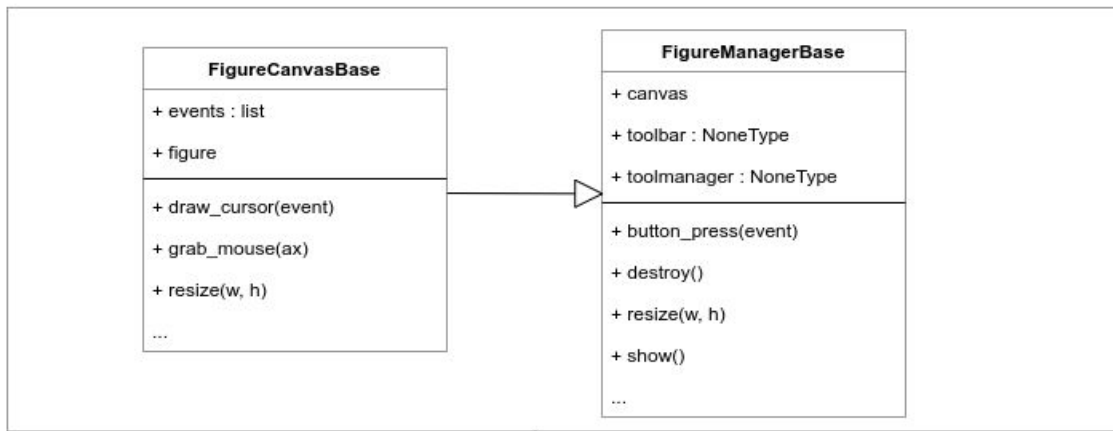
- Backend Layer
  - An abstraction layer over various components capable of rendering Figures
  - Has 2 functional categories
    - User interface backends (interactive)
    - Hardcopy backends (non-interactive)
  - Base classes for the backend include
    - FigureCanvasBase and FigureManagerBase
      - Represents the canvas which Figures are rendered
      - Holds a reference to figure
      - Update figure with respect to the canvas
      - Defining draw methods to render Figure
      - Defines event methods translating native GUI Toolkit events
    - RendererBase and GraphicsContextBase
      - Handles Drawing operations
    - Event, ShowBase, and TimerBase
      - Manages events, event loops, and timing
      - Methods instantiate Event classes and are connected to the callbacks stored in callbackRegistry which is stored as an attribute of FigureBaseCanvas
      - With every GUI Toolkit, there is an event loop. Each event loop utilizes a Timer which fires an event for callbacks to be made
  -
- Artist Layer
  - Generate plots for display, manipulation, and publication, which logic is mainly run through the *Artist* base class
  - In the *Artist* class handles the visual effects of what the user sees such as:
    - Visibility (what is hidden and what is not)
    - Labels (set what will be displayed in the legend)
    - Handles user interaction will callbacks
  - Container classes (Figure, XAxis, YAxis, Subplot, etc.) and Primitive classes (Rectangle, Polygon, Circle, AxesImage, etc) are subclasses of the *Artist* class
    - Since they are subclasses, they can render into the implementation of FigureCanvasBase
  - Collection classes (LineCollection, EventCollection, etc.) help to efficiently display a large number of Primitive type classes
- Scripting layer
  - User interface that simplifies the task of working with other layers
  - Known as pyplot
  - Pyplot uses Factory to create a figure manager when imported

### Auto Generated UML:

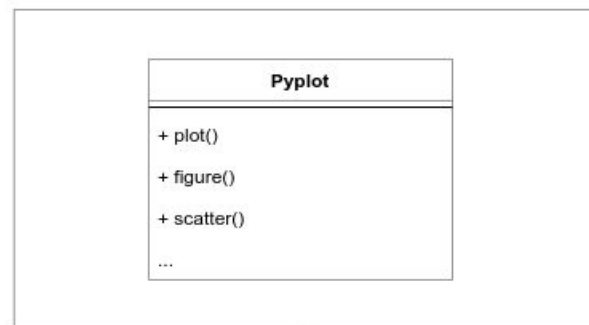


## Overall Structure of the Architecture UML:

### Backend



### Scripting



### Front End

