# Methodology

The methodology section serves as a roadmap detailing the design, development, and evaluation of the VV Python library. The objective here is to offer comprehensive insights into the technical aspects of VV, elucidating the rationale behind various design and architectural choices, as well as the methods used for implementation and assessment. Given that this library aims to bridge a gap in healthcare data visualization, especially in handling big data and providing customizable solutions for automation, it is crucial to understand the techniques and technologies that make it both functional and scalable.

This section will start by explaining the basic ideas behind the VV project. Then, we'll get into the actual development aspects, including our use of Object-Oriented Programming (OOP) and Test-Driven Development (TDD). Lastly, we will explore how the library was evaluated, describing the metrics and methods used during the evaluation phase.

## Requirement Analysis

This section outlines the key functions and quality features expected of the VV system. It provides a set of clear requirements that will guide the design and implementation stages of the project. To enhance the system's effectiveness and ease of use, we present selected use cases that illustrate how VV will interact with other systems for better integration in the broader data visualization landscape. In short, this section sets the foundational requirements that will guide the development efforts.

### User Stories

User stories serve as a vehicle for capturing product functionality from the end user's perspective. These stories encapsulate discrete system features in a format that is easy to read and understand by both non-technical stakeholders and the development team [[37,38]](https://paperpile.com/c/wYkhtl/8m6D+1DNo). In the context of VV, a system designed to automate the rendering of graphical charts from clinical research data, the user stories described here are aimed to outline the essential features and functionalities that satisfy the needs of different roles involved in clinical research.

#### Scope

The following user stories are specifically tailored to the needs of clinical researchers, medical writers, data analysts, and system administrators who are the key stakeholders of the VV system. They focus on tasks related to data visualization, report generation, and system management within the context of clinical research.

#### Stakeholder Definitions

* **Clinical Researcher**: A professional conducting clinical studies.
* **Medical Writer**: A professional responsible for creating documents that describe research results, product use, and other scientific dissemination outlets.
* **Data Analyst**: A person responsible for interpreting complex clinical data sets.
* **Data Scientist**: A professional who uses scientific methods, processes, algorithms, and systems to extract insights and knowledge from structured and unstructured data.
* **System Administrator**: A person responsible for managing and maintaining the system infrastructure, including VV.

#### Story 1: Batch Rendering of Clinical Charts

* **As a** clinicalresearcher/data scientist,
* **I want** VV to batch render multiple charts from automatically generated clinical reports,
* **So that** a large volume of data can be visually represented quickly and efficiently.
* **Acceptance Criteria**:
  + System should be able to accept multiple clinical reports as input.
  + System should be able to render charts in batches without manual intervention.
  + Rendered charts should accurately represent the data from the clinical reports.
  + System should provide an option for selecting the types of charts to be rendered (bar, line, etc.).
  + Batch rendering process should complete within a reasonable time frame (e.g., under 5 minutes for 50 reports).

#### Story 2: Deployment to Miro for Triage

* **As a** clinical researcher/data scientist,
* **I want** VV to deploy rendered charts directly to specific boards in Miro,
* **So that** they can be quickly triaged alongside tabular reports.
* **Acceptance Criteria**:
  + System should integrate with Miro API.
  + System should be able to send rendered charts to specified Miro boards.
  + Rendered charts should appear on the Miro boards in a layout that facilitates triage.
  + Charts should be deployed to Miro boards without manual intervention.

#### Story 3: Export Charts for Research Documents

* **As a** medical writer,
* **I want** VV to export rendered charts in formats suitable for academic manuscripts, posters, and other research documents,
* **So that** the visual data complements the written content.
* **Acceptance Criteria**:
  + System should offer multiple export formats such as PNG, JPEG, SVG, etc.
  + Exported charts should maintain high resolution and quality.
  + System should allow for batch export of multiple charts.

#### Story 4: Inclusion of Supplementary Material

* **As a** clinical researcher/data scientist,
* **I want** VV to render charts that can be included as supplementary material when publishing,
* **So that** we can increase the transparency of our research.
* **Acceptance Criteria**:
  + System should allow rendering of charts that are suitable for supplementary material in terms of quality and resolution.
  + System should allow for easy categorization or labeling of such charts for supplementary material.
  + Charts should be exportable in a format accepted by major research publications.

#### Story 5: Automated Data Retrieval

* **As a** data analyst,
* **I want** VV to automatically retrieve data from predefined clinical report formats,
* **So that** I don't have to manually input data for chart rendering.
* **Acceptance Criteria**:
  + System should be able to identify and read predefined clinical report formats.
  + System should accurately extract relevant data fields from these reports.
  + Data retrieval should happen automatically through API calls.

#### Story 6: Customization of Chart Types

* **As a** clinical researcher/data scientist,
* **I want** to specify the type of chart (bar, line, scatter, etc.) VV should render,
* **So that** the chart is most appropriate for the data being represented.
* **Acceptance Criteria**:
  + System should offer a range of chart types (bar, forest plot, survival, etc.).
  + Users should be able to easily select the desired chart through configuration.
  + Rendered charts should accurately represent the selected chart type.

#### Story 7: Logging and Monitoring

* **As a** system administrator,
* **I want** VV to keep logs of all chart rendering activities,
* **So that** I can monitor system performance and troubleshoot issues.
* **Acceptance Criteria**:
  + System should maintain logs for each chart rendering activity.
  + Logs should include timestamps, types of charts rendered, and any errors or warnings.
  + Logs should be easily accessible for review and analysis.

#### Story 8: Re-run Chart Rendering with Updated Data

* **As a** clinical researcher/data scientist/medical writer,
* **I want** to re-run chart rendering when new data is available,
* **So that** my visual representations are always up-to-date.
* **Acceptance Criteria**:
  + System should allow for easy updating of data sources.
  + Users should be able to initiate re-rendering without having to redo the entire setup.

### Non-functional Requirements

The non-functional requirements for VV aim to outline the quality attributes the system should possess. These are essential aspects that define how well the system performs its functions rather than what functions it performs. They encompass characteristics like modularity, error handling, and auditability, among others. These requirements are especially critical in ensuring that VV is not only functional but also efficient, maintainable, and adaptable to various environments and use-cases. Below is a list of the non-functional requirements we deem essential for the system:

#### System Architecture

* **Modularity**: The system should be modular to allow for easier debugging and updating of individual components.
* **Extensibility**: Designed in a way to easily allow the addition of new functionalities.

#### Usability and User Experience

* **Configurability**: Users should be able to easily configure chart rendering options regardless of the environment (API, module, terminal).
* **Environment Agnosticism**: Should be usable as an importable Python module, accessible via web API, or through the terminal.

#### Reliability

* **Error Handling**: The system should be able to gracefully handle errors and exceptions, providing useful error messages.

#### Maintenance and Support

* **Documentation**: All code should be well-documented, and system documentation should be easily accessible for maintenance activities.
* **Auditability**: Should provide logging features to keep track of data processing and rendering activities.

## Technical Foundations and Development Paradigms

The objective of VV is the automation of data visualization, helping with the challenges in handling large and complex data sets common in healthcare. Concurrently, the project serves an educational purpose, offering the developer a framework to explore and learn fundamental software development paradigms. This educational aspect makes it crucial to ensure that the project adheres to established coding practices and methodologies, making it both a practical tool for data visualization and a case study in applying robust software development principles.

The following sections will delve into the specifics of these foundational principles, revealing how they guided the choices in architecture and functionalities in VV.

### Modularity

In VV, modularity is a fundamental element guiding our design approach. This ensures that each module is a self-contained unit with well-defined interfaces, enhancing both reusability and portability, attributes highly valued in specialized fields like healthcare informatics [[39]](https://paperpile.com/c/wYkhtl/acC6T).

### Object-Oriented Programming (OOP)

In the VV library, OOP serves as a pivotal architectural choice, both for the developer's educational enrichment and the system's overall functionality and extensibility. Employing OOP facilitates encapsulation, which allows for the bundling of data and methods that operate on that data within single units or classes.

OOP also uses inheritance, enabling code reusability and abstraction. For instance, different types of charts, be it a bar chart, a forest plot, or a survival plot, can be represented as individual classes. These classes can contain methods to set chart properties, draw axes, and render the data. Since each chart type may have common characteristics such as a title or axes labels, inheritance allows these shared features to be abstracted into a parent class. Specific chart types can then inherit from this parent class, enabling them to reuse common code while still allowing for their own specialized features. Furthermore, different deployment targets, like cloud storage or Miro boards, can also be abstracted into separate classes, encapsulating the methods required for deploying visualizations to these locations. This makes the system adaptable and easier to integrate with new deployment options as needs evolve.

### Test-Driven Development (TDD)

TDD serves as a rigorous verification mechanism that aligns with the project’s objective of delivering a reliable and high-quality tool. Furthermore, for the author, who is in the process of learning software development, employing TDD provides a valuable educational experience. This hands-on exposure is expected to be invaluable in future projects and particularly beneficial when collaborating within larger teams that also utilize TDD.

To implement TDD in this project, we selected pytest as the testing library for its feature-rich environment, ease of use, and compatibility with various Python frameworks. It provides detailed failure reports to streamline debugging, and its straightforward syntax is especially beneficial for those new to TDD [[40]](https://paperpile.com/c/wYkhtl/W7lT).

## Evaluation Metrics and Methods

The evaluation phase for the VV Python library was designed to assess both the functional capabilities of the library and its impact on workflow efficiency. The key performance indicators (KPIs) used for this evaluation were "Time to First Chart Draft" and "Time to Final Chart," designed to capture the time-efficiency gains enabled by the VV library.

### Time to First Chart Draft

This metric captures the time needed from receiving the initial dataset to generating the first draft of a chart. For the manual method, this involves gathering values for relevant measures, preparing a Vega-Lite JSON definition, populating the JSON with the data and adjusting necessary parameters.

### Time to Final Chart

This metric gauges the time from the receipt of the initial data to the point where the chart is exported in the appropriate format (e.g., SVG) and uploaded to a platform like Google Drive and included in a Miro board for further analysis and comparison. This encompasses the entire lifecycle of chart production and is intended to capture any efficiency gains that may be achieved through the VV library.

### Data Sources for Evaluation

The primary data source for these evaluations is time-tracking data from MTG Research and Development Lab activities. This data focuses on chart development for academic papers and is an integral part of our methodology. It has been recorded using a tracker within the Monday.com platform, which is the project management tool employed by the company for all R&D activities. This time-tracking data from past projects, where chart generation was performed manually, serves as a comparative baseline for evaluating the VV Python library's effectiveness.

### Simulation for Adjustment for Fatigue and Human Intervention

To provide a comprehensive evaluation of the VV Python library's efficiency in chart creation, we extended our analysis by including a simulation that includes considerations for task fatigue and additional human intervention for validation. For this exercise, we focused on the "Time-to-Final-Chart" metric, which captures the total time needed to finalize a chart, accounting for all adjustments and confirmations.

The analysis was conducted using R (version 4.2.3) [[41]](https://paperpile.com/c/wYkhtl/4IRV), and visualizations were generated using the ggplot2 package [[13]](https://paperpile.com/c/wYkhtl/1tUb).