NeuraViz: A Web Application For Visualizing Artificial Neural Network Structures

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin–La Crosse

La Crosse, Wisconsin

by

Bennett Wendorf

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

May, 2024

NeuraViz: A Web Application For Visualizing Artificial Neural Network Structures

By Bennett Wendorf

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Prof. Albert Einstein Examination Committee Chairperson	Date
Prof. Isaac Newton Examination Committee Member	Date
Prof. Marie Curie Examination Committee Member	Date

Abstract

Wendorf, Bennett, "NeuraViz: A Web Application For Visualizing Artificial Neural Network Structures," Master of Software Engineering, May 2024, (Jason Sauppe, Ph.D.).

This manuscript describes the software engineering processes and principles adhered to during the development of Neuraviz, a web application for visualizing artificial neural network structures. Users upload pre-trained machine learning models from popular frameworks including Pytorch and Keras, and Neuraviz generates a visual representation of the model's architecture. The following manuscript focuses on the design, implementation, testing, and deployment of NeuraViz in an effort to comprehensively encapsulate the entire development process.

Acknowledgements

I would like to extend my sincerest thanks to my project advisor, Dr. Jason Sauppe, for his guidance and support throughout the development of NeuraViz. His feedback was always crucial in pointing me in the right direction, especially when I was overwhelmed with possibilities.

Thank you also to the entire computer science department at the University of Wisconsin-La Crosse for tirelessly helping me through all my coursework and projects throughout my tenure at the university. My ability to complete this monumental task would not have been possible without them.

I would also like to thank the open source community for providing the tools and resources used in this project. Open source software is an integral part of the modern software space and none of our lives would be the same without it.

Finally, I would like to thank my parents, family, friends, and all the teachers, mentors, and colleagues who have helped, supported, and encouraged me throughout my life. I am more grateful thank I can possibly express in words.

Table of Contents

Abstract i
Acknowledgments ii
List of Tables v
List of Figures vi
Glossary vii
1. Introduction 1
    1.1. Overview 1
    1.2. Background 1
    1.3. Goals 1
2. Software Development Process 2
    2.1. Overview 2
    2.2. Life Cycle Model 2
        2.2.1. Waterfall Model 2
        2.2.2. Iterative Model 3
        2.2.3. Agile (Scrum) Model 4
    2.3. Development Process Technologies 4
    2.4. Functional Requirements 5
    2.5. Non-Functional Requirements 6
3. Design 8
    3.1. Overview 8
    3.2. UML Class Diagram 8
    3.3. Database 8
    3.4. User Interface 8
4. Implementation 9
    4.1. Overview 9
    4.2. Technologies Used 9
        4.2.1. Client 9
        4.2.2. Server 9
        4.2.3. Data Layer 9
    4.3. Development 9
    4.4. Deployment 9
5. Testing 10
    5.1. Overview 10
    5.2. Verification 10
    5.3. Validation 10
6. Security 11
    6.1. Overview 11
    6.2. Threat Model 11
    6.3. Session Management 11
    6.4. Web Application Security 11
7. Conclusion 12
    7.1. Overview 12

7.2. Challenges 12
7.3. Future Work 12
8. References 13
9. Appendices 14

List of Tables

List of Figures

1 Waterfall Model Diagram 3
2 Iterative Model Diagram 4
3 Scrum Model Diagram 5
4 Jira Scrum Board 6

Glossary

1. Introduction

1.1. Overview

1.2. Background

1.3. Goals

2. Software Development Process

2.1. Overview

Developing software is an extensive and complex process that requires a lot of planning, both in relation to the methodologies used during the development process and the requirements, both functional and non-functional of the software. This section details life cycle models considered for NeuraViz's development, the model that was eventually chosen, and modifications to the model necessary for the development of this particular system. It also outlines functional and non-functional requirements for NeuraViz.

2.2. Life Cycle Model

Prior to beginning development of NeuraViz, a number of software life cycle models were considered to govern the pace and structure of development. In all, the waterfall model, iterative model, and agile model were considered. More specifically with agile, a variation of scrum, modified for a single developer, was considered. Ultimately, the modified scrum model was chosen for its flexibility and ability to adapt quickly to changing requirements.

2.2.1. Waterfall Model

The waterfall model is one of the oldest software development lifecycle (SDLC) models, originally proposed by Winston Royce in 1970 [1]. The model is a linear, sequential approach to software development, with each phase of the development process directly following the previous phase. Each subsequent phase relies on the previous phase, and as such the model does not allow going back to previous phases once they are completed. The first phase of the waterfall model is the requirement analysis phase, in which project requirements, both functional and non-functional, are gathered and documented. At this phase, requirements are also often analyzed for traits like consistency and feasibility. The second phase is the system design phase, in which the architecture of the software system is designed in full. All details of what needs to be done and how it will be completed are considered and documented during this phase. Third, the implementation phase is where the code for the software is written and the design from the previous step is implemented in full, exactly as specified during the design phase. During the fourth phase, the software is tested for bugs and errors, and issues are resolved as needed. Fifth, the software is deployed to the client in its entirety. In the waterfall model, this is the first time the client has seen the software. Finally, the software is maintained and updated as needed for as long as the client needs it. Figure 1 shows a diagram of the waterfall model and its constituent parts.

Because of its rigid structure, the waterfall model excels at being very easy to understand and pick up quickly for new developers, which was initially intriguing during model selection for NeuraViz. In addition, it is easy to manage with relatively little overhead in management. When project requirements are well understood up front and unlikely to change, the waterfall model also serves the benefit of ensuring design is completed before implementation begins, which leads to fewer mistakes and less necessity to change the code once it has been written. However, for projects where requirements are less well understood or are likely to change,

such as in this project, the waterfall model struggles to adapt and may lead to a design that was flawed in the first place with no way to fix it. In addition, the waterfall model does not allow for client feedback until the software is fully completed, which can lead to a lot of wasted time and effort if the client is not satisfied with the final product. In the case of this project where adaptability was crucial, the waterfall model would have been a poor choice.

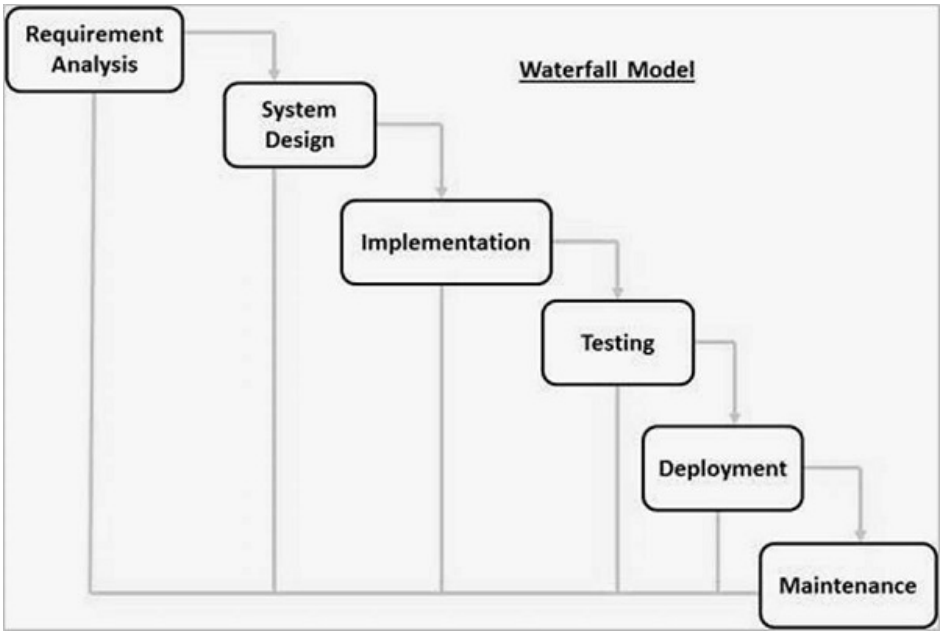

Figure 1. Waterfall Model Diagram [4]

2.2.2. Iterative Model

Like the waterfall model, the iterative model is mostly linear and sequential with a relatively rigid structure where each step directly follows the previous step. The phases in the iterative model, as shown in Figure 2, roughly match those in the waterfall model, including a requirements analysis phase, a design phase, and implementation phase, a testing phase, and a deployment phase. Unlike the waterfall model, however, the iterative model runs these phases, with the exception of requirements analysis, multiple times, restarting the sequence of phases after each deployment. This serves the major benefit of allowing the ability for the client to give feedback on the project sooner and more often.

Due to its similarity to the waterfall model, the iterative model exhibits many of the same benefits of waterfall in that it is easy to understand with a relatively linear structure and minimal overhead. Like the waterfall model, when requirements are understood at the project outset, the design is likely to be almost fully complete before development begins, so the code is also less likely to require changes later in the process. While the iterative model does improve on the waterfall model's lack of ability for client feedback, it still struggles with changing requirements as each iteration of the project is still long and expected to be a relatively complete implementation of the software. The lack of adaptability made the iterative model a poor choice for NeuraViz's development where changing requirements were expected from the beginning.

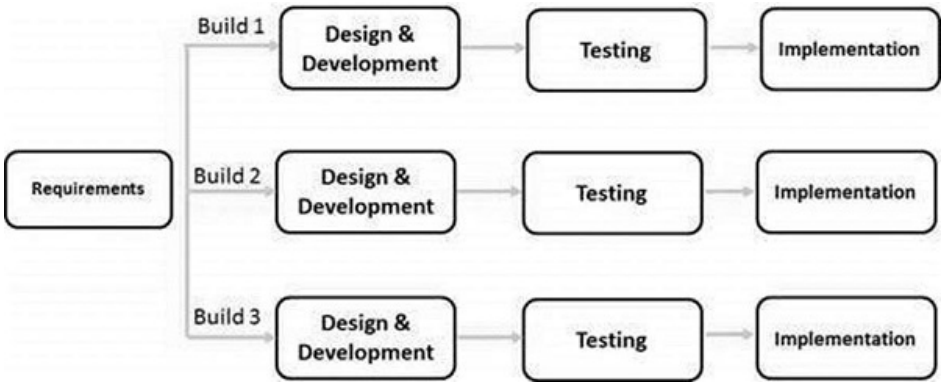

Figure 2. Iterative Model Diagram [3]

2.2.3. Agile (Scrum) Model

In contrast to the other models, agile models, and specifically scrum, was designed with the express intent to adapt to changing requirements quickly. Since the exact requirements and design for NeuraViz were not known well ahead of time, scrum was chosen for development due to its ability to pivot quickly when new design details were discovered. While agile is a category of software development models that focus on adaptability, scrum is a specific type of agile model that places emphasis on small, self-organized teams working in short, iterative cycles called sprints. Each sprint is typically two to four weeks long and ends with a review of the work completed during the sprint and a planning session for the next sprint. The scrum model is shown in Figure 3. While the diagram shows a two to three month timeline per iteration, scrum more typically follows a shorter sprint length.

Since NeuraViz only has one developer, the scrum model doesn't fit perfectly. However, many aspects of the model do fit relatively well, with some slight modifications. Scrum typically emphasizes daily stand-up meetings with each team of developers. Due to the longer timeline of NeuraViz's development and the single developer, these meetings were partially dealt with through daily review of the project board to help ensure that projects stay on track. In addition, the developer met every week with the project advisor, Dr. Jason Sauppe, who in some sense served as a stakeholder on the project and a product owner. While these meetings don't match exactly with any part of the scrum model, they serve as a combination of stand-up meetings and sprint retrospectives. For the purposes of this project, sprints were completed every week. This allowed for very quick turnaround on features, and enabled constant feedback and reflection on project requirements.

2.3. Development Process Technologies

An important part of the scrum model is managing small, independent projects that can be completed in a short amount of time. To help manage these projects, Jira's scrum template was used. This template allows for a main project backlog where user stories can be converted to sprint tasks. Each sprint can then be created and tasks can be scheduled into one. Once each sprint begins, projects/tasks move through columns including backlog, programming, testing required, and done. This allows for a clear view of exactly what is being worked on

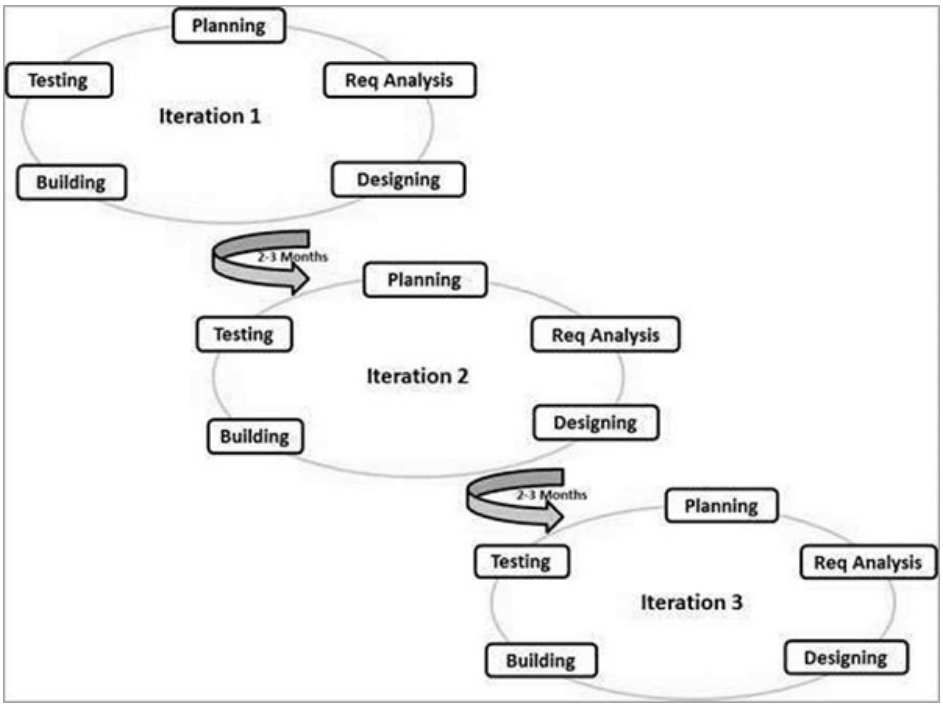

Figure 3. Scrum Model Diagram [2]

at any given time. The scrum board for sprint 21 (March 5th, 2024 - March 19, 2024) can be seen in Figure 4. Jira provides exceptional features for managing projects in a scrum format, allowing the tracking of not only sprint tasks through the stages of development on the project board, but additional features such as time tracking on projects as well.

In addition to Jira, Github and specifically branches were used to help keep track of individual sprint tasks. A new branch was created for each task, with the name of the branch including the task key from Jira and a brief description. Jira's integration with GitHub provided a link to see what phase the code was actually in directly from the Jira task, including whether a pull request had been created or merged.

2.4. Functional Requirements

Since agile methodologies were chosen for the development of this project, a set of functional requirements in the form of user stories were collected prior to the start of development. These user stories served as a guide for features to implement and helped ensure that no major functionality was missed during development.

NeuraViz is a relatively simple application with only one type of user. As such, the functional requirements are relatively straightforward, including the ability to upload a pre-trained machine learning model trained in either Pytorch or Keras. In addition, users should be able to see either the full graph of the uploaded model, or a collapsed version depending on the scale of the uploaded model. Additional functionality is also documented such as the ability to navigate the page via pan and zoom functionality on the graph, and clicking or hovering on various network components.

In addition to the functionality of viewing the network itself, user stories were also docu-

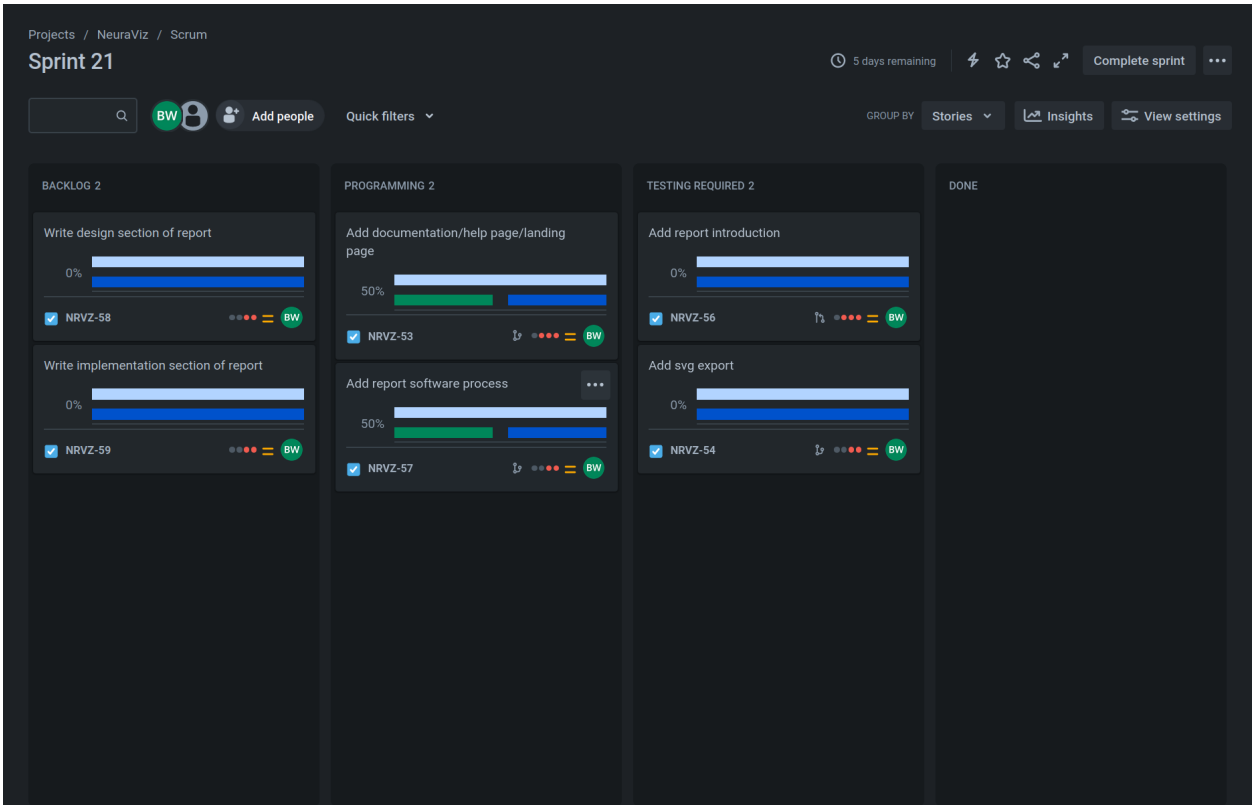

Figure 4. Jira Scrum Board

mented for export functionality, allowing the graph of the model to be exported both as an svg and as a LaTeX document in the format of the tikz library.

The full set of user stories is documented in the user stories document. This document also includes user stories for functionality that was not implemented in the current version of NeuraViz, but may be implemented in future versions.

2.5. Non-Functional Requirements

In addition to the functionality documented as user stories, non-functional requirements were also documented to ensure that the user experience of NeuraViz was as smooth as possible. Identified non-functional requirements are as follows:

- Large network layers are collapsed if they are too big to reasonably render.
- If I am unable to view the model visually, labels exist for screen readers as much as possible.
- If the site takes a long time to load, skeletonized components are shown to indicate that the site is still loading.
- As a user, my data is reasonably secure, both during transmission and processing.
- Themes are sufficiently differentiable for colorblind users.

- Invalid models are rejected and not stored unnecessarily.

3. Design

3.1. Overview

3.2. UML Class Diagram

3.3. Database

3.4. User Interface

4. Implementation

4.1. Overview

4.2. Technologies Used

4.2.1. Client

4.2.2. Server

4.2.3. Data Layer

4.3. Development

4.4. Deployment

5. Testing

5.1. Overview

5.2. Verification

5.3. Validation

6. Security

6.1. Overview

6.2. Threat Model

6.3. Session Management

6.4. Web Application Security

7. Conclusion

7.1. Overview

7.2. Challenges

7.3. Future Work

8. References

[1] Gagan Gurung, Rahul Shah, and Dhiraj Jaiswal. Software development life cycle models-a comparative study. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pages 30–37, 07 2020.

[2] TutorialsPoint. Sdlc - agile model. `https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm`. accessed: 03.14.2024.

[3] TutorialsPoint. Sdlc - iterative model. `https://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm`. accessed: 03.05.2024.

[4] TutorialsPoint. Sdlc - waterfall model. `https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm`. accessed: 03.05.2024.

9. Appendices