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

List of Tables

List of Figures

Glossary

1. Introduction

1.1. Overview

1.2. Background

1.3. Goals

2. Software Development Process

2.1. Overview

The process of 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 lays out 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. 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.

Because of its rigid structure, the waterfall model excels at being very easy to understand and pick up quickly for new developers. 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, 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.

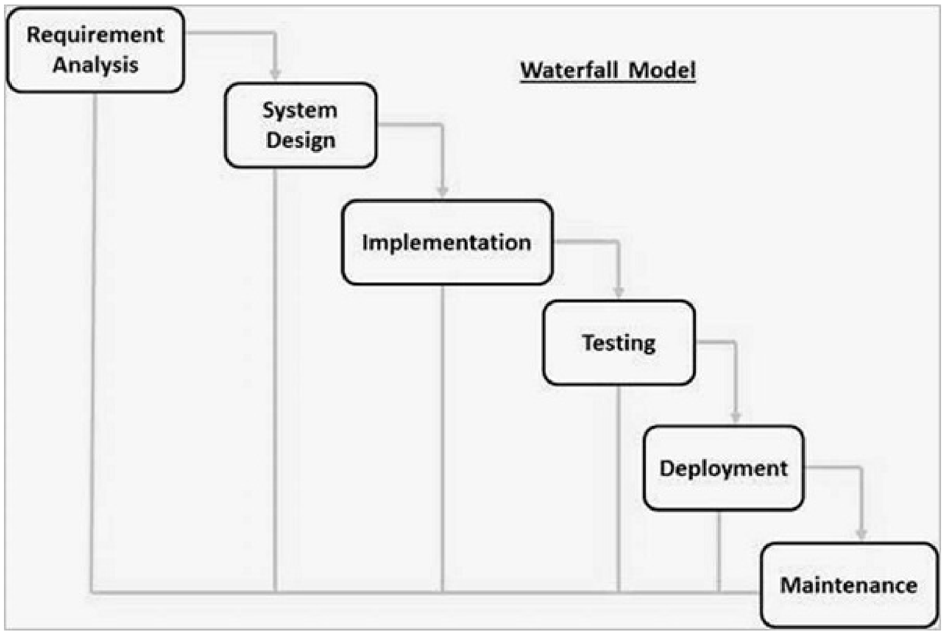

Figure 1. Waterfall Model Diagram

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 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.

2.2.3. Agile (Scrum) Model

2.3. Requirements

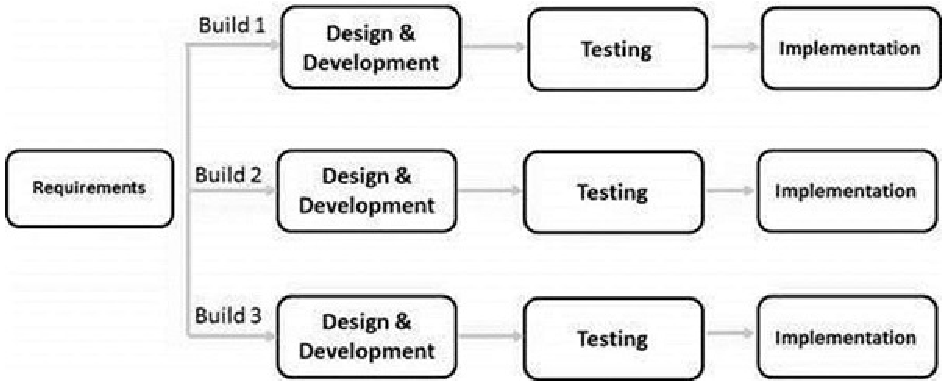

Figure 2. Iterative Model Diagram

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. Bibliography

9. Appendices