

# Complexity Analyzer

Sprint 3  
10/26/2025

Name	Email Address
Ryerson Brower	ryerson.brower178@topper.wku .edu
Name2	WKU email address2

CS 396  
Fall 2024  
Project Technical Documentation

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Project Overview . . . . .	1
1.2	Project Scope . . . . .	1
1.3	Technical Requirements . . . . .	1
1.3.1	Functional Requirements . . . . .	1
1.3.2	Non-Functional Requirements . . . . .	2
<b>2</b>	<b>Project Modeling and Design</b>	<b>3</b>
2.1	System Architecture . . . . .	3
2.2	Defined Microservices . . . . .	3
2.3	Development Considerations . . . . .	3
<b>3</b>	<b>Implementation Approach</b>	<b>3</b>
3.1	Software Process Model . . . . .	3
3.2	Key Algorithms and Techniques Used . . . . .	3
3.3	Microservices and Docker Usage . . . . .	3
3.4	Container Network Communication Setup and Testing . . . . .	3
3.5	Infrastructure as Code approach . . . . .	3
<b>4</b>	<b>Software Testing and Results</b>	<b>3</b>
4.1	Software Testing with Visualized Results . . . . .	3
<b>5</b>	<b>Conclusion</b>	<b>3</b>
<b>6</b>	<b>Appendix</b>	<b>3</b>
6.1	Software Product Build Instructions . . . . .	3
6.2	Software Product User Guide . . . . .	3
6.3	Source Code with Comments . . . . .	4

## List of Figures

# 1 Introduction

## 1.1 Project Overview

The Algorithm Complexity Analyzer project aims to develop a microservice-based system that automates the evaluation of algorithmic efficiency through Big-O and recurrence relation analysis. The project provides a scalable platform that decomposes the complexity analysis process into independent services, each responsible for a specific function such as time complexity detection, recurrence modeling, and result presentation. By using Docker containers, the system ensures cross-platform compatibility, simple deployment, and reliable communication between services.

The primary purpose of the project is to simplify the process of analyzing algorithmic time complexity, which is often performed manually and can often times be done wrong. Developers and students frequently struggle to derive Big-O notation from algorithmic structures. This tool addresses that challenge by automating the process and identifying recurrence relations from code snippets and giving an explanation behind each result.

The project is designed to solve several practical and educational problems. It eliminates the need for manual complex mathematical analysis by providing automated, accurate, and explainable evaluations. It also bridges the gap between theoretical computer science problems and practical software engineering. Furthermore, by containerizing each microservice, the system supports execution, easy maintenance, and scalable updates without the risk of affecting other services.

The target audience for this system includes, software developers, computer science students, and educators who would either like to automate their own code, or learn how to evaluate the computational complexity of algorithms efficiently. Educational institutions could integrate the tool into programming or algorithm design courses. Overall, this project delivers a platform-independent, educational, and beginner-friendly solution for algorithmic complexity evaluation. By combining automation, explanation, and scalability within a microservice architecture, it deepens our understanding of computer science theory and software engineering practice.

## 1.2 Project Scope

The Algorithmic Complexity Analyzer project focuses on developing a microservice-based tool that automates the evaluation of algorithmic efficiency through Big-O and recurrence relation analysis. The project will deliver a containerized system highlighting the microservice architecture that allows us to work on separate services independently. This ensures scalability and modularity.

**Deliverables:** The key deliverables include: source code for each microservice, Dockerfiles and configuration files for deployment, a working prototype of the analyzer capable of accepting code files, automated orchestration scripts using Docker Compose, and documentation detailing installation, usage, and system architecture. The final deliverable will be a deployable system that can evaluate and explain algorithmic complexity within five seconds of receiving input.

**Work and Resources:** The project required about two weeks of development, testing, and documentation. Tasks include system design, microservice implementation, Docker setup, and testing. Required resources include Docker Desktop, a code editor such as Visual Studio Code, a GitHub repository for version control, and access to a command line interface for deployment testing.

**Benefits and Outcomes:** The primary benefit of this system is automating analyzing algorithmic efficiency, which is a complex and error-prone process. The outcome is user-friendly to those who have experience using a command line interface, educational, and easily scalable. It helps students understand computational complexity by giving explanations and can assist developers in optimizing algorithms during the design phase.

## 1.3 Technical Requirements

### 1.3.1 Functional Requirements

The functional requirements ensure that the test automation framework effectively supports quality assurance throughout all stages of development. Automated testing validates the functionality of individual modules, allowing early detection of errors before the final integration. Integration testing ensures that interconnected components communicate and function correctly before deployment. A centralized test case management system enables developers to easily create, modify, and organize tests. This improves accessibility and collaboration efforts.

<b>Mandatory Functional Requirements</b>
The system must support the creation and execution of automated unit tests for individual components of the disaster response coordination application to ensure that each unit functions correctly in isolation.
The framework must facilitate automated integration testing to verify that different components of the application work together as intended, ensuring data flows correctly between modules.
The system must provide a mechanism for managing test cases, allowing developers to create, modify, and organize test cases within the automation framework for easy access and execution.
The automation framework must seamlessly integrate with existing CI/CD pipelines to enable automatic triggering of test execution upon code commits or pull requests, ensuring immediate feedback on code quality.
The system must generate detailed reports on test results, including pass/fail rates, code coverage metrics, and historical trends, allowing teams to assess code quality and identify areas for improvement.
<b>Extended Functional Requirements</b>
Ext. Req 1
Ext. Req 2
Ext. Req 3

Integration with CI/CD pipelines provides continuous code validation, preventing bugs from progressing into larger problems. Finally, detailed reporting on test performance along with tracing the code through historical reports helps teams track quality trends in order to make data-driven improvement decisions. Together, these functions streamline the development process, enhancing reliability, and ensures all systems components perform as intended under real-world conditions.

### 1.3.2 Non-Functional Requirements

<b>Mandatory Non-Functional Requirements</b>
The framework must include robust error handling and logging mechanisms to capture and report any failures or exceptions during test execution, providing developers with actionable insights for debugging and resolution.
The test automation framework must execute all automated tests within an average time of under 5 minutes to ensure timely feedback during the development process, facilitating rapid iteration.
The system must be designed to accommodate an increasing number of test cases and applications without degradation in performance, allowing the addition of new tests as the project evolves.
The system must implement security best practices to protect test data and configurations, ensuring that sensitive information is not exposed during test execution or reporting processes.
The test automation framework must provide an intuitive and streamlined interface that allows developers to easily create, manage, and run test cases, ensuring a smooth onboarding process with helpful documentation and interactive tutorials to enhance user engagement and satisfaction.
<b>Extended Non-Functional Requirements</b>
Ext. Req 1
Ext. Req 2
Ext. Req 3

The non-functional requirements focus on making the test automation framework reliable, fast, and easy to use. Strong error handling and logging are needed so developers can quickly see what went wrong during testing and fix problems without wasting time or causing delays. The tests should run in under five minutes to give quick feedback, helping teams improve efficiency and make improvements right away. The framework also needs to handle test cases and applications without slowing down, so it can grow as the project expands. Security is important to make sure that any test data, results, or settings are kept safe and private at all times. Finally, the system should be simple to use with clear instructions, examples, and tutorials that help new developers easily

use the application. Altogether, these requirements ensure the network is dependable, secure, and efficient for all development teams.

## 2 Project Modeling and Design

### 2.1 System Architecture

### 2.2 Defined Microservices

### 2.3 Development Considerations

## 3 Implementation Approach

### 3.1 Software Process Model

### 3.2 Key Algorithms and Techniques Used

### 3.3 Microservices and Docker Usage

### 3.4 Container Network Communication Setup and Testing

### 3.5 Infrastructure as Code approach

## 4 Software Testing and Results

### 4.1 Software Testing with Visualized Results

**Test Plan Identifier:**

**Introduction:**

**Test item:**

**Features to test/not to test:**

**Approach:**

**Test deliverables:**

**Item pass/fail criteria:**

**Environmental needs:**

**Responsibilities:**

**Staffing and training needs:**

**Schedule:**

**Risks and Mitigation:**

**Approvals:**

## 5 Conclusion

Overall, this was a very useful project to work on. We had very limited knowledge using Docker and microservices before this project. Although we are nowhere near experts in the space yet, we gained footing in containerization practices. The project itself is very useful and fortified our understanding on algorithm analysis as we developed the product. To make it easier on ourselves, we stuck to a command line interface. In past projects, we have tried to stay away from this, but for this one the most important part was making sure it was calculating things effectively and efficiently. Going forward, if we had more time to work on this project or were to continue to work on it after this sprint is done. I believe we could create a simple web-based or graphical user interface. Then we could make the system simpler to use and be able to insert more complex code. As of now, it is pretty basic but we really enjoyed the direction it was going as we went through the tough learning curve.

## **6 Appendix**

### **6.1 Software Product Build Instructions**

Text goes here.

### **6.2 Software Product User Guide**

Text goes here.

### **6.3 Source Code with Comments**

Text goes here.