**JINNAH UNIVERSITY FOR WOMEN**



**COURSE:** Software Construction and development

**PROJECT TITLE:** Fruit Grading System

* **Group Members**

**1**-Umm e Ruman Asif

**2**-Anzila Alvi

**3**-Tehreem Altaf

**4**-Areeba Khan

* **Group leader:**

1. Anzila Alvi

* **Division of work among group members**

|  |  |  |  |
| --- | --- | --- | --- |
| S.NO | TASK | NAME | DURATION |
| 1 | Algorithm implementation and code optimizing | **Umm-e-ruman** | **4 days** |
| 2 | Unit testing and code profiling | **Areeba khan** | **3 days** |
| 3 | Testing and quality assurance | **Tahrim Altaf** | **2 days** |
| 4 | Version control(github) and code profiling | **Anzila Alvi** | **2 days** |

* **Introduction**
* **Background**

In the contemporary food industry, the need for efficient and reliable fruit grading systems is paramount. The Text-Based Fruit Grading System emerges as a solution to address the challenges associated with manual grading processes. By leveraging technology, this system aims to automate and streamline the evaluation of fruit quality based on key parameters such as weight and sweetness.

Manual fruit grading can be a time-consuming and subjective process, leading to inconsistencies in quality assessment. Automation through software not only accelerates this process but also introduces a level of precision and accuracy that is challenging to achieve manually. The integration of technology in fruit grading aligns with the broader trend of leveraging digital solutions to enhance efficiency in agricultural practices.

* **Objectives**

The primary objectives of the Text-Based Fruit Grading System are as follows:

Automation of Grading Process:

Develop a software system capable of automatically grading fruits based on specified criteria.

User-Friendly Interface:

Implement a user-friendly console interface that allows users to easily select and grade fruits.

Algorithmic Efficiency:

Optimize the grading algorithm to ensure efficient and accurate assessments.

Performance Profiling:

Utilize code profiling techniques to identify and address potential performance bottlenecks.

As the project progresses, the system aims to contribute to the improvement of fruit quality assessment, providing a valuable tool for farmers, distributors, and others involved in the fruit supply chain. The introduction of automation not only enhances productivity but also facilitates more consistent and objective grading practices.

The subsequent sections of this report will delve into the methodologies, technologies used, results obtained, and discussions surrounding the development and optimization of the Text-Based Fruit Grading System.

* **Scope**

The scope of the project includes creating a sample set of fruits with associated attributes such as weight and sweetness. The system allows users to input their preferred fruit and sweetness level and employs a KNN algorithm to predict the sweetness grade.

* **Techniques used**
* **Programming Language**

**C#:** Chosen for its suitability in developing console applications, C# provided a robust and type-safe programming environment.

* **Integrated Development Environment (IDE)**

**Visual Studio:** Utilized as the primary integrated development environment, providing a comprehensive set of tools for coding, debugging, and testing.

* **Testing Framework**

**NUnit:** Selected as the testing framework for unit testing the components of the Text-Based Fruit Grading System. NUnit facilitated the creation and execution of test cases to ensure the reliability of the code.

* **Code Profiling**

**Visual Studio Profiler:** Employed for code profiling, this tool allowed for the identification of performance bottlenecks within the application, facilitating targeted optimizations.

These technologies collectively formed a cohesive and efficient development stack, enabling the team to implement, test, and optimize the Text-Based Fruit Grading System effectively. The choice of tools aligned with the project's goals of simplicity, reliability, and maintainability.

* **Code optimization and profiling**
* **Code Optimization:**

The code optimization phase focused on enhancing the efficiency of the grading algorithm and improving overall performance. The primary areas of optimization included reducing redundant computations and streamlining the grading logic.

* **Redundant Computation Reduction:**

In the Grade Fruit method, the weight and sweetness values of the fruit were repeatedly accessed. To eliminate redundancy, these values were stored in local variables, resulting in a more streamlined and efficient implementation:

static int GradeFruit(Fruit fruit)

{

double weight = fruit.Weight;

double sweetness = fruit.Sweetness;

// Grading logic based on weight and sweetness

if (weight >= 150 && sweetness >= 4.0)

{

return 1; // Excellent

}

else if (weight >= 100 && sweetness >= 3.0)

{

return 2; // Good

}

else

{

return 3; // Poor

}

}

This optimization aimed to enhance the readability of the code while minimizing redundant computations during the grading process.

* **Code Profiling**

Code profiling was conducted using Visual Studio Profiler to identify and address potential performance bottlenecks. The profiling process involved the execution of the application while monitoring resource usage and identifying areas where optimizations could be applied.

* **Profiling Results:**

The Visual Studio Profiler revealed insights into the application's resource utilization, including CPU usage, memory consumption, and execution times of specific methods. While the provided codebase is relatively small and focused, the profiling results indicated that the grading algorithm performed efficiently.

No critical performance issues were identified during profiling. However, the profiling process remains a valuable practice as the application scales or additional features are introduced. It provides a proactive approach to maintaining optimal performance by identifying potential bottlenecks early in the development lifecycle.

* **Optimized Results:**

The optimization of the grading algorithm resulted in a more concise and readable implementation. Although the specific impact on performance may be modest in this small-scale application, these optimizations lay the foundation for maintaining efficiency as the project evolves.

Continued vigilance in code optimization and profiling is recommended as the system grows in complexity or handles larger datasets in future iterations.

The subsequent sections will delve into the overall results, including the implementation of features, testing outcomes, and discussions surrounding the development of the Text-Based Fruit Grading System.

**Results**

* **Implemented Features**

The Text-Based Fruit Grading System successfully implemented several key features to provide a comprehensive and user-friendly fruit grading experience.

* **User Interface**

A console-based user interface allows users to view available fruits and input their selection for grading.

* **Display of Available Fruits**

The system displays a list of available fruits, presenting their names, weights, and sweetness levels.

* **Grading Algorithm**

The grading algorithm assesses fruits based on predefined criteria, categorizing them into Excellent, Good, or Poor grades.

5.2 Code Optimization Results

Code optimization efforts focused on improving the efficiency of the grading algorithm and eliminating redundant computations. The following results were achieved:

* **Redundant Computation Reduction**

Redundant computations were reduced by storing the weight and sweetness values of the fruit in local variables within the GradeFruit method.

* **Impact**

While the impact on performance in this small-scale application may be modest, the optimization enhances code readability and lays the groundwork for maintaining efficiency in future iterations.

* **Test Results**

Unit testing using the NUnit framework was employed to ensure the correctness and reliability of the grading algorithm. The following outcomes were observed:

* **Test Coverage**

Comprehensive test cases were designed to cover various scenarios, including fruits with different weights and sweetness levels.

* **Test Success**

All unit tests passed successfully, validating the accuracy of the grading algorithm and confirming that the system behaves as expected.

* **Regression Prevention**

Unit tests serve as a valuable tool for preventing regressions, ensuring that code changes do not introduce new issues or compromise existing functionality.

* **Conclusion**

The implemented features, code optimization results, and successful test outcomes collectively contribute to the overall success of the Text-Based Fruit Grading System. The following sections will delve into discussions on challenges faced during development, lessons learned, and potential future enhancements for the project.

**Techniques used**

1. **Object-Oriented Programming (OOP):**

class Fruit

{

public string Name { get; }

public double Weight { get; }

public double Sweetness { get; }

public Fruit(string name, double weight, double sweetness)

{

Name = name;

Weight = weight;

Sweetness = sweetness;

}

}

1. **User Interface Design:**

Console.WriteLine($"Available Fruits:");

foreach (var fruit in fruits)

{

Console.WriteLine($"{fruit.Name} - Weight: {fruit.Weight}g, Sweetness: {fruit.Sweetness}");

}

Console.Write("Enter the index of the fruit you want to grade: ");

**3.Code Optimization:**

static int GradeFruit(Fruit fruit)

{

double weight = fruit.Weight;

double sweetness = fruit.Sweetness;

// Grading logic based on weight and sweetness

if (weight >= 150 && sweetness >= 4.0)

{

return 1; // Excellent

}

else if (weight >= 100 && sweetness >= 3.0)

{

return 2; // Good

}

else

{

return 3; // Poor

}

}

**4.Unit Testing with NUnit:**

class FruitGradingTests

{

[Test]

public void GradeFruit\_Excellent()

{

Fruit excellentFruit = new Fruit("ExcellentFruit", 160, 4.5);

Assert.AreEqual(1, GradeFruit(excellentFruit));

}

[Test]

public void GradeFruit\_Good()

{

Fruit goodFruit = new Fruit("GoodFruit", 120, 3.5);

Assert.AreEqual(2, GradeFruit(goodFruit));

}

[Test]

public void GradeFruit\_Poor()

{

Fruit poorFruit = new Fruit("PoorFruit", 80, 2.5);

Assert.AreEqual(3, GradeFruit(poorFruit));

}

}

**5.Agile Development with Scrum**

This methodology is not directly reflected in code snippets but is applied through iterative development, regular sprint meetings, and continuous adaptation.

**6. Code Profiling with Visual Studio**

Profiling is not explicitly shown in code snippets. It is typically done through the Visual Studio Profiler tool during the development process.

These techniques collectively contribute to the project's success, promoting modularity, efficiency, user-friendliness, and reliability.

**7.Documentation Comments:**

/// Represents a text-based fruit grading system.

class Program

{

// ...

}

* **Project Summary**

The Text-Based Fruit Grading System has successfully achieved its primary objectives, providing a user-friendly platform for automating fruit grading based on weight and sweetness criteria. The project summary encompasses key accomplishments:

* **Achievements**

**Automation:** The system automates the fruit grading process, reducing the reliance on manual inspection.

**Efficiency:** Code optimization efforts enhance the efficiency of the grading algorithm, contributing to improved performance.

**User-Friendly Interface:** The console-based interface offers a simple yet effective means for users to interact with the system.