Final Project

by

Hazem Abo-Donia, Benjamin Moks

Stevens.edu

December 23, 2024

Thank you, Professor Muresan, for everything we have learned this semester, and to Anthony for all of your hard work grading.

Final Project

Hazem Abo-Donia, Benjamin Moks
Stevens.edu

Table 1: Document Update History

Date	**Updates**
11/08/2024	Initial: • Created Project Proposal.
11/12/2024	Brainstorming and initial planning: • Defined the problem statement and primary goals for Productivity Pal. • Outlined the two primary use cases: ”Switch Workflow Mode” and ”Analyze Productivity Patterns.” • Drafted an initial high-level architecture for the project.
11/18/2024	Progress on proposal: • Created sequence diagrams to visualize interactions for both use cases. • Designed the initial UI sketches and storyboard for user interactions. • Documented the basic workflow for managing tasks and tracking productivity.
11/24/2024	Finalizing the proposal: • Added a deployment diagram showing the system’s physical architecture. • Reviewed and refined all use case tables and priority mappings. • Submitted the completed proposal draft for internal team review.
12/01/2024	Iteration planning meeting: • Discussed milestone goals and tasks. • Added tasks to GitHub project board for refactoring and testing WorkflowManager. • Delegated tasks between team members based on strengths.

Table 1: Document Update History

Date	Updates
12/03/2024	Development progress: • Implemented Singleton pattern for WorkflowManager. • Updated GitHub project board with testing tasks. • Began writing unit tests in `test_workflow.py`.
12/04/2024	Troubleshooting and test expansion: • Resolved `ModuleNotFoundError` issues during testing. • Added edge case handling for invalid workflows in WorkflowManager. • Expanded unit tests to cover Singleton instance management.
12/05/2024	Integration and documentation: • Conducted integration tests for WorkflowManager and other components. • Planned tasks for Agile documentation and meeting notes. • Added tasks for milestone deliverable preparation to the GitHub project board.
12/06/2024	Final review: • Validated all tests passed successfully using `pytest`. • Reviewed milestone document draft, including process reflection. • Held final meeting to confirm readiness of all deliverables.
12/07/2024	Milestone Submission day: • Finalized and submitted milestone document. • Included meeting notes, GitHub board screenshots, and Agile process documentation.
12/10/2024	Final sprint planning meeting: • Defined sprint goals and assigned tasks. • Planned testing strategies for integrating `TaskTracker` with `AnalyticsEngine`. • Reviewed UML diagrams for alignment with the final architecture.
12/14/2024	Development progress during the final sprint: • Implemented `TaskTracker` for real-time task monitoring. • Enhanced `NotificationService` to provide troubleshooting suggestions. • Integrated `TaskTracker` with `AnalyticsEngine` to deliver detailed productivity insights. • Made significant updates to UML diagrams based on feedback.

Table 1: Document Update History

Date	**Updates**
12/18/2024	Final integration and testing: • Conducted integration testing for all components. • Achieved 100% test coverage using `pytest`. • Finalized documentation, including all UML diagrams and process reflections.
12/19/2024	Final submission: • Submitted the final project deliverables. • Included updated UML diagrams, GitHub board screenshots, and sprint summaries. • Shared final reflection and future work plans in the report.

Table of Contents

1 Project Description **11**
- 1.1 Introduction . . . 11
- 1.2 Problem Statement . . . 12
- 1.3 Bot Description . . . 12
- 1.4 Tagline . . . 12

2 Stories **13**
- 2.1 Primary Use Cases . . . 13
  - 2.1.1 Use Case: Switch Workflow Mode . . . 13
  - 2.1.2 Use Case: Analyze Productivity Patterns . . . 14

3 Use Cases **16**
- 3.1 Tables . . . 17
- 3.2 Use Case Diagrams . . . 17
  - 3.2.1 Use Case: Switch Workflow Mode . . . 17
  - 3.2.2 Use Case: Analyze Productivity Patterns . . . 18
- 3.3 Use Case Details . . . 18
  - 3.3.1 Switch Workflow Mode . . . 18
  - 3.3.2 Analyze Productivity Patterns . . . 19

4 UI Design Sketches **22**
- 4.1 Sequence Flow . . . 22
  - 4.1.1 Sequence Diagram . . . 22
- 4.2 Interaction Breakdown . . . 23
  - 4.2.1 Switch Workflow Mode . . . 23
  - 4.2.2 Analyze Productivity Patterns . . . 24

5 Architecture **25**
- 5.1 Overview . . . 25
- 5.2 Architecture Diagrams . . . 27
  - 5.2.1 Class Diagram . . . 27
  - 5.2.2 Activity Diagrams . . . 28
  - 5.2.3 Deployment Diagram . . . 30

5.3 Component Descriptions . . . 31
5.3.1 Implementation Details . . . 31
5.3.2 Basic Workflow . . . 32
5.4 Design Patterns . . . 34
5.4.1 Singleton Pattern . . . 34

6 Additional UML Diagrams **36**
6.1 Profile Diagram . . . 36
6.2 Component Diagram . . . 37
6.3 Deployment Diagram . . . 38
6.4 Package Diagram . . . 39
6.5 Object Diagram . . . 40
6.6 Composite Structure Diagram . . . 41
6.7 Interaction Diagram . . . 42
6.8 Communication Diagram . . . 43
6.9 Timing Diagram . . . 44
6.10 State Machine Diagram . . . 45

7 Project Development **46**
7.1 Iteration Overview . . . 46
7.1.1 Iteration 1 Goals . . . 46
7.1.2 Iteration 1 Achievements . . . 46
7.1.3 Iteration Challenges . . . 47
7.1.4 Final Sprint Goals . . . 47
7.1.5 Final Sprint Achievements . . . 48
7.1.6 Final Sprint Challenges . . . 48
7.2 Process and Tools . . . 48
7.2.1 Agile Practices . . . 48
7.2.2 Tasks Breakdown for Iteration 1 . . . 49
7.2.3 GitHub Storyboard for Iteration 1 . . . 50
7.2.4 Tasks Breakdown for Final Sprint . . . 51
7.2.5 GitHub Storyboard for Final Sprint . . . 52
7.2.6 Task Tracking . . . 52
7.2.7 Meeting Notes: Iteration Planning . . . 52
7.2.8 Meeting Notes: Sprint Planning . . . 53
7.3 Reflection . . . 54
7.3.1 What Went Well . . . 54
7.3.2 Challenges Faced . . . 54

8 Reflection **55**
8.1 Overview . . . 55
8.2 Achievements . . . 55
8.3 Challenges Faced . . . 56
8.4 Lessons Learned . . . 56

8.5 Future Work 57
8.6 Conclusion 57

Bibliography **58**

List of Tables

1 Document Update History 3
1 Document Update History 4
1 Document Update History 5

3.1 Requirements Table 21

7.1 Tasks Breakdown for Iteration 1 49
7.2 Tasks Breakdown for Final Sprint 51

List of Figures

3.1 Use Case Diagram: Switch Workflow Mode 17
3.2 Use Case Diagram: Analyze Productivity Patterns 18

4.1 Sequence Diagram for Switching Workflow Modes and Analyzing Productivity Patterns 23

5.1 Class Diagram: Displays the relationships between key components, including UserInterface, WorkflowManager, NotificationService, AnalyticsEngine, TaskTracker, and Mode. 27
5.2 Activity Diagram for Switch Workflow Mode: Illustrates the flow of user commands to adjust applications and notifications. 28
5.3 Activity Diagram for Analyze Productivity Patterns: Demonstrates the process of analyzing tracked tasks and generating insights. 29
5.4 Deployment Diagram: Depicts the distribution of components across user devices, application servers, and analytics servers with secure communication. 30

6.1 Profile Diagram: Showing stereotypes and relationships used in the system. 36
6.2 Component Diagram: Highlighting the system components and their interactions. . 37
6.3 Deployment Diagram: Depicting how the system is deployed across devices and servers. 38
6.4 Package Diagram: Grouping related classes and packages for system organization. . 39
6.5 Object Diagram: Providing a snapshot of objects and their relationships at runtime. 40
6.6 Composite Structure Diagram: Demonstrating the internal structure of system components. 41
6.7 Interaction Diagram: Illustrating dynamic interactions between objects. 42
6.8 Communication Diagram: Highlighting the message exchange between system components. 43
6.9 Timing Diagram: Showing the time constraints and interactions for key operations. 44
6.10 State Machine Diagram: Depicting the lifecycle and state transitions for tasks. . . . 45

7.1 GitHub Project Board for Iteration 1: Tracking progress and task completion. . . . 50
7.2 GitHub Project Board for Final Sprint: Tracking final sprint goals and tasks. 52

Chapter 1

Project Description

Hazem Abo-Donia, Benjamin Moks

1.1 Introduction

We, Hazem Abo-Donia and Benjamin Moks, are both third-year Software Engineering majors who have come together to work on the Productivity Pal project. As dedicated students in our field, we share a strong interest in creating tools that enhance productivity and streamline workflows. This project combines our technical skills and shared vision to build a tool that addresses the everyday challenges of task management and maintaining focus. Through Productivity Pal, we aim to develop a solution that can assist users in managing their tasks efficiently across multiple platforms, helping them stay organized and productive in a fast-paced world. For the rolls in the project Benjamin took the lead on making the diagrams while Hazem focused on implementing them into Overleaf. Although we aimed for a 50/50 split, Hazem contributed more as he led the project assigning tasks to both of use to ensure we stayed organized and completed the project on time.

Here is the attached link to our Github repository

1.2 Problem Statement

In today's fast-paced world, individuals often struggle to manage tasks, stay focused, and maintain productivity, especially when handling multiple workflows across platforms. Many people lack a streamlined way to organize and initiate specific workflows based on context, leading to reduced efficiency and increased stress.

This project addresses the need for a versatile tool that can help users optimize their productivity by managing workflows across multiple devices and applications. Productivity Pal provides a centralized assistant capable of recognizing user commands, adjusting focus modes, and organizing tasks based on the current workflow, thereby reducing distractions and enhancing productivity.

1.3 Bot Description

Productivity Pal is a bot designed to assist users in managing workflows by automatically initiating different setups based on the user's needs. It operates across platforms, including desktop and mobile, to ensure users can access productivity tools and adjust settings on the go. The bot recognizes user commands to switch between focus modes, organize task lists, and open relevant applications automatically.

This bot also provides analytics on user productivity, tracking completed tasks, and optimizing future workflows by offering actionable insights. Productivity Pal facilitates a seamless transition between tasks, making it an effective tool for anyone looking to maximize their productivity in a structured way.

1.4 Tagline

"Productivity Pal - Your Multi-Platform Workflow Optimizer"

Chapter 2

Stories

Hazem Abo-Donia, Benjamin Moks

2.1 Primary Use Cases

2.1.1 Use Case: Switch Workflow Mode

- **Preconditions:**
  - User is logged into the Productivity Pal bot.
  - WorkflowManager and NotificationService are initialized.
- **Main Flow:**
  1. User inputs a command to switch to a specific workflow mode (e.g., ”Focus Mode” or ”Meeting Mode”).
  2. WorkflowManager validates the selected workflow mode.
  3. WorkflowManager creates a new mode using the `Mode` class.
  4. ApplicationManager activates and launches the preset applications for the selected mode (e.g., ”Notion” and ”Slack” for Focus Mode).

5. NotificationService configures system settings such as silencing notifications or adjusting alerts.
6. The bot displays a confirmation message to the user.

- **Subflows:**

S1 WorkflowManager ensures the selected mode is valid.

S2 ApplicationManager launches all required applications associated with the mode.

S3 NotificationService adjusts notification settings based on the workflow mode.

- **Alternative Flows:**

E1 If any application fails to open, the bot notifies the user and provides troubleshooting steps through the NotificationService.

E2 If the selected mode is invalid, the bot informs the user and requests a valid mode.

2.1.2 Use Case: Analyze Productivity Patterns

- **Preconditions:**
  - User has enabled analytics on the bot.
  - TaskTracker has recorded task and time-tracking data.
- **Main Flow:**
  1. User requests productivity analysis via the bot interface.
  2. AnalyticsEngine retrieves task and usage data from TaskTracker.
  3. AnalyticsEngine analyzes active tasks, application usage, and workflow transitions.
  4. AnalyticsEngine generates a summary report, including:
     - Active tasks or ”No active task” status.
     - Productivity insights such as peak productivity hours and task efficiency.

- Suggested workflow configurations for better focus and time management.

5. The bot displays the analysis summary to the user.

- **Alternative Flows:**

E1 If no data is available for analysis, the bot informs the user and suggests starting a task or enabling analytics.

Chapter 3

Use Cases

Hazem Abo-Donia, Benjamin Moks

3.1 Tables

3.2 Use Case Diagrams

3.2.1 Use Case: Switch Workflow Mode

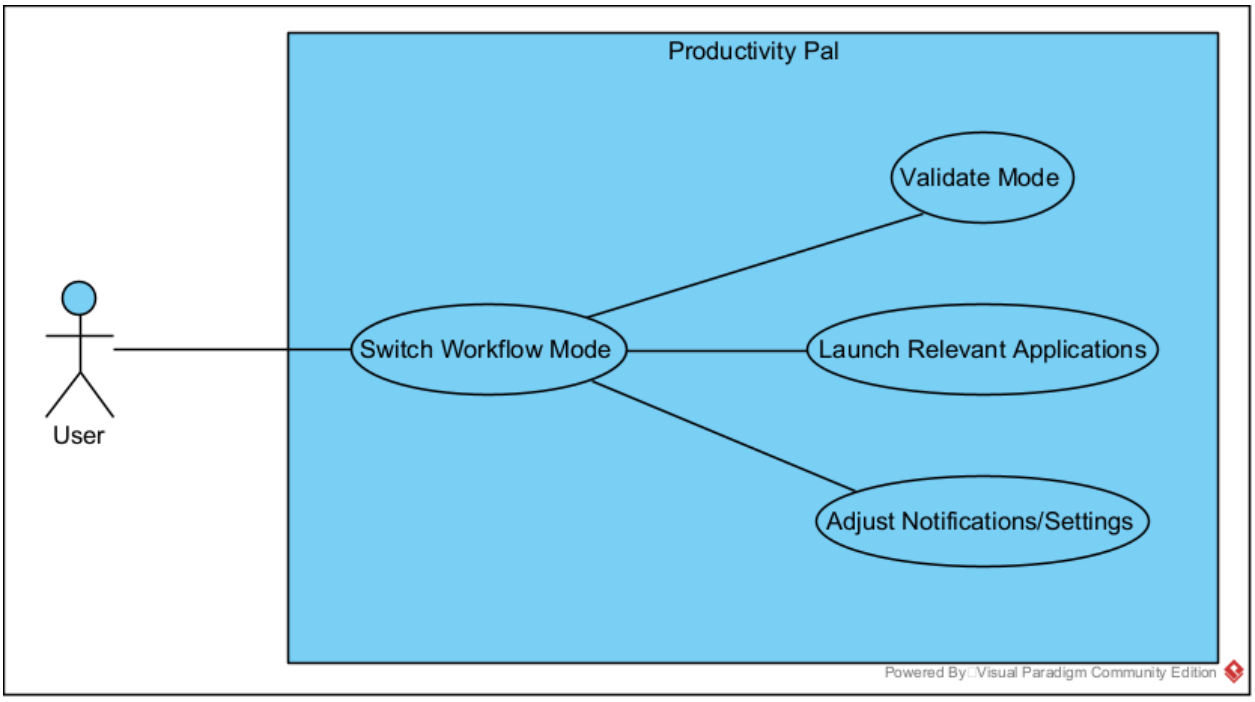

Figure 3.1: Use Case Diagram: Switch Workflow Mode

3.2.2 Use Case: Analyze Productivity Patterns

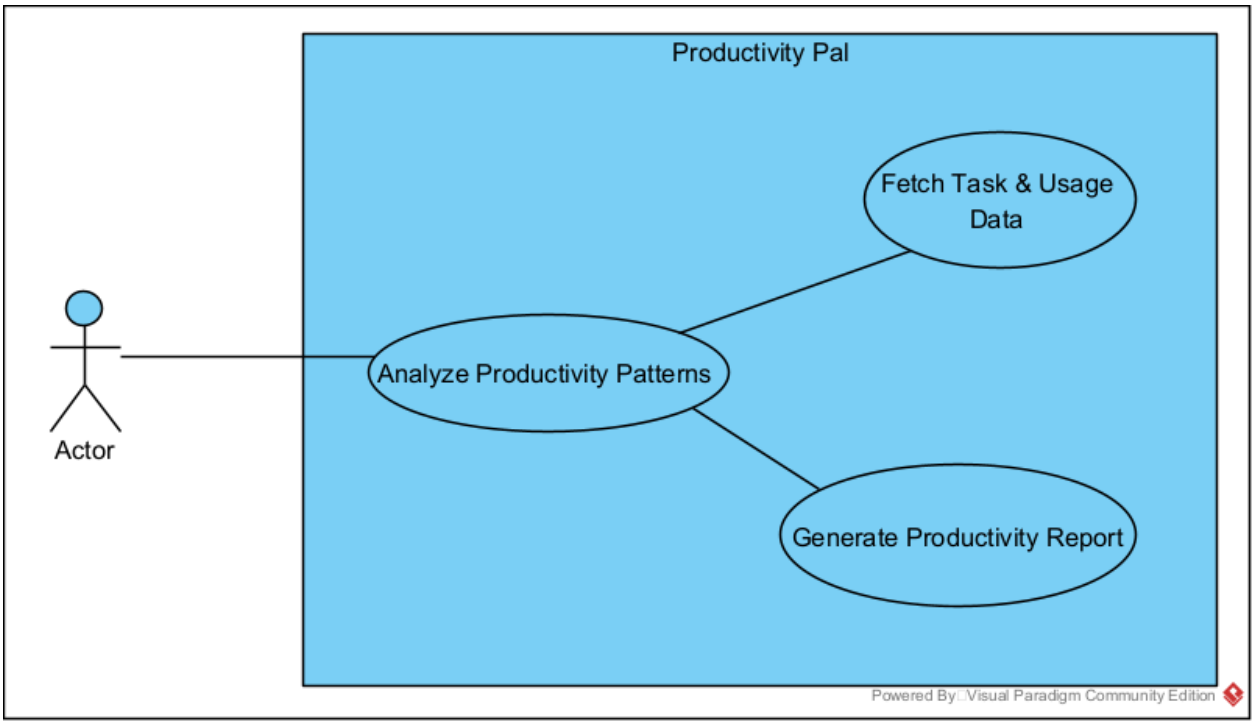

Figure 3.2: Use Case Diagram: Analyze Productivity Patterns

3.3 Use Case Details

3.3.1 Switch Workflow Mode

Description: This use case allows the user to switch between predefined workflow modes, such as Focus Mode or Meeting Mode. Each mode activates specific applications and adjusts notification settings based on the user's preference.

Actors:

- User
- WorkflowManager
- ApplicationManager

- NotificationService

Main Flow:

1. The user selects a workflow mode through the UserInterface.
2. WorkflowManager validates the selected mode.
3. ApplicationManager launches the relevant applications (e.g., Notion, Slack for Focus Mode).
4. NotificationService adjusts system settings, such as silencing notifications.
5. The system confirms the mode switch to the user.

Alternative Flow:

- If an invalid mode is selected, the system displays an error message and prompts the user to reselect.
- If an application fails to launch, the system notifies the user and suggests troubleshooting.

3.3.2 Analyze Productivity Patterns

Description: This use case allows the user to request a summary of their productivity patterns, including active tasks, tracked time, and overall productivity trends.

Actors:

- User
- AnalyticsEngine
- TaskTracker

Main Flow:

1. The user requests an analysis via the UserInterface.
2. AnalyticsEngine retrieves task and activity data from TaskTracker.

3. AnalyticsEngine generates a summary, including the current active task or productivity trends.

4. The system displays the summary to the user.

Alternative Flow:

- If no active task is being tracked, the system suggests that the user start a task to enable tracking.

Requirement	**Mapped Use Case**	**Priority**
Allow users to switch between predefined workflow modes (e.g., Focus Mode, Meeting Mode)	Switch Workflow Mode	Must
Automatically open a set of applications relevant to the selected workflow mode	Switch Workflow Mode	Must
Adjust notification and system settings based on the active workflow mode	Switch Workflow Mode	Should
Track and record time spent on tasks and applications for productivity analysis	Analyze Productivity Patterns	Must
Provide real-time updates on the current task and productivity status	Analyze Productivity Patterns	Must
Generate a summary report of productivity patterns, including task details and active time tracking	Analyze Productivity Patterns	Should
Provide troubleshooting support if an application fails to open during a workflow switch	Switch Workflow Mode	Could
Notify users of trends in productivity over time, offering suggestions for improved focus and time management	Analyze Productivity Patterns	Could
Ensure compatibility with various operating systems and integrate with commonly used productivity applications	Switch Workflow Mode, Analyze Productivity Patterns	Must

Table 3.1: Requirements Table

Chapter 4

UI Design Sketches

Hazem Abo-Donia, Benjamin Moks

4.1 Sequence Flow

This sequence flow illustrates how users interact with the bot to switch workflow modes or retrieve productivity analytics.

4.1.1 Sequence Diagram

The following sequence diagram demonstrates the interaction between the user and the core system components during key use cases:

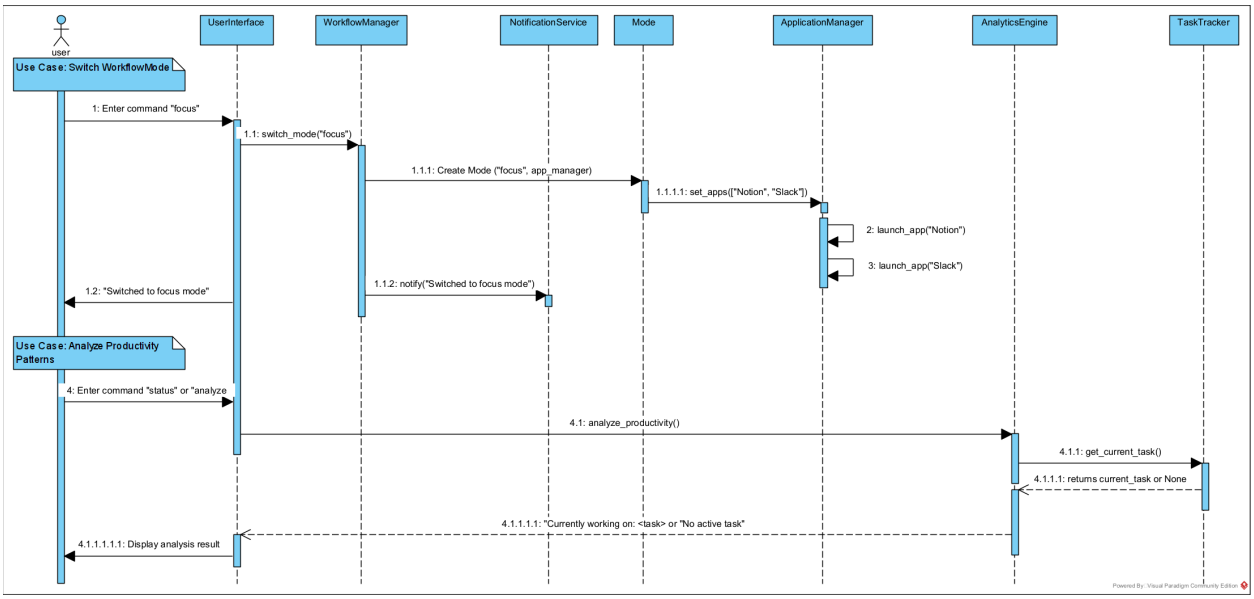

Figure 4.1: Sequence Diagram for Switching Workflow Modes and Analyzing Productivity Patterns

4.2 Interaction Breakdown

4.2.1 Switch Workflow Mode

This interaction involves the following steps:

1. The user enters a command to switch to a specific mode (e.g., ”focus” or ”meeting”).
2. The `WorkflowManager` validates the mode and creates an instance of the `Mode` class.
3. The `ApplicationManager` launches the relevant applications for the mode.
4. The `NotificationService` adjusts system notifications and alerts.
5. The bot confirms the mode switch to the user via the user interface.

4.2.2 Analyze Productivity Patterns

This interaction involves the following steps:

1. The user requests a productivity analysis through the interface.
2. The `AnalyticsEngine` retrieves data from the `TaskTracker`.
3. If a task is active, the `AnalyticsEngine` generates a summary with details about the task and time spent.
4. If no task is active, the bot informs the user and suggests starting a task for tracking.
5. The bot displays the analysis summary, including insights on productivity patterns.

Chapter 5

Architecture

Hazem Abo-Donia, Benjamin Moks

5.1 Overview

Productivity Pal is structured as a multi-layered, modular system that emphasizes flexibility, scalability, and cross-platform compatibility. It integrates with various user devices and third-party applications to manage and streamline workflow. The following components work together to provide a seamless experience:

- **User Interface Layer:** This layer facilitates interaction between the user and the bot. It allows users to issue commands through text inputs, and the UI provides feedback, guiding users through available options and error handling for invalid commands. The UserInterface interacts with core services such as the WorkflowManager and AnalyticsEngine to execute user commands.

- **Workflow Management Service:** Central to Productivity Pal, this service organizes, schedules, and manages workflows. It leverages a Singleton design pattern to ensure that only one instance of `WorkflowManager` exists throughout the application. This service interprets user commands and switches to specific workflow modes such as ”Focus Mode” or ”Meeting Mode.” Each mode uses the `Mode` class to define associated applications, notification preferences, and configurations.

- **Analytics Engine:** This component aggregates data from the `TaskTracker` on user productivity, including time spent on tasks, application usage patterns, and workflow transitions. Using this data, it generates detailed reports with actionable insights, such as peak productivity hours and workflow suggestions. It operates in real time and provides textual feedback to users regarding active tasks or general productivity summaries.

- **Notification Service:** The NotificationService interacts with the system to manage notifications dynamically. Depending on the active workflow mode, it adjusts system settings to enhance focus or maintain awareness (e.g., silencing notifications in "Focus Mode"). This service also notifies the user of mode switches and offers troubleshooting if issues arise in applying notification settings.

- **Platform Integration Layer:** This layer ensures seamless compatibility with different operating systems (e.g., Windows, macOS, iOS, Android) and integrates with commonly used productivity applications (e.g., Slack, Zoom, Notion). It uses APIs and middleware to facilitate data exchange between the bot and third-party services, enabling smooth operation and responsiveness.

- **TaskTracker:** Part of the Analytics Engine, this component tracks the active task, including its name and duration. It enables accurate reporting of task-specific productivity data and supports task switching for better organization.

5.2 Architecture Diagrams

5.2.1 Class Diagram

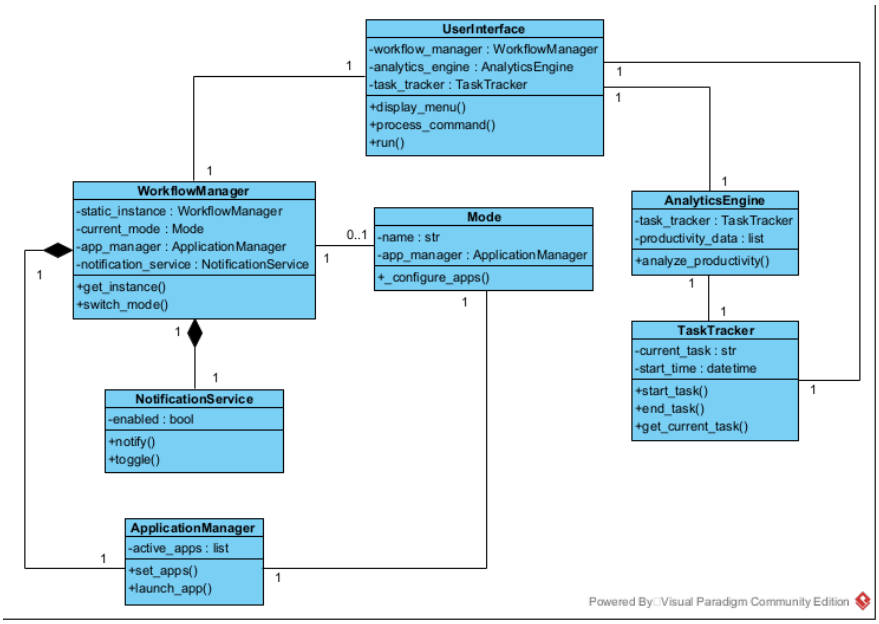

Figure 5.1: Class Diagram: Displays the relationships between key components, including UserInterface, WorkflowManager, NotificationService, AnalyticsEngine, TaskTracker, and Mode.

5.2.2 Activity Diagrams

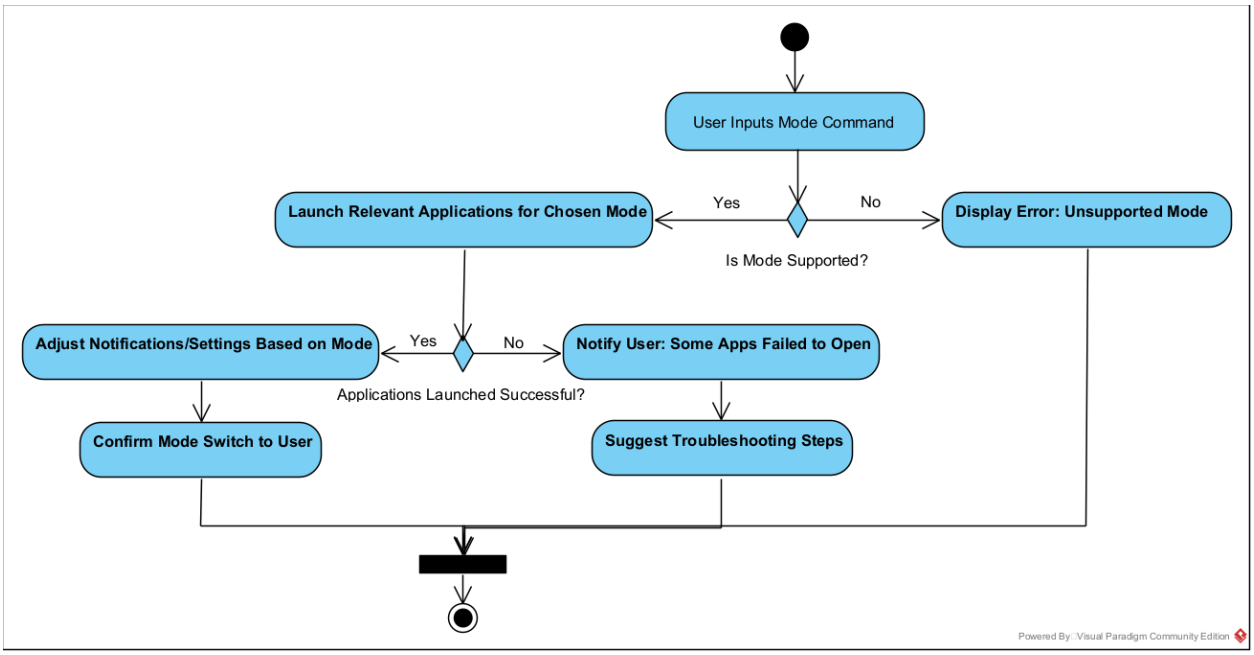

Figure 5.2: Activity Diagram for Switch Workflow Mode: Illustrates the flow of user commands to adjust applications and notifications.

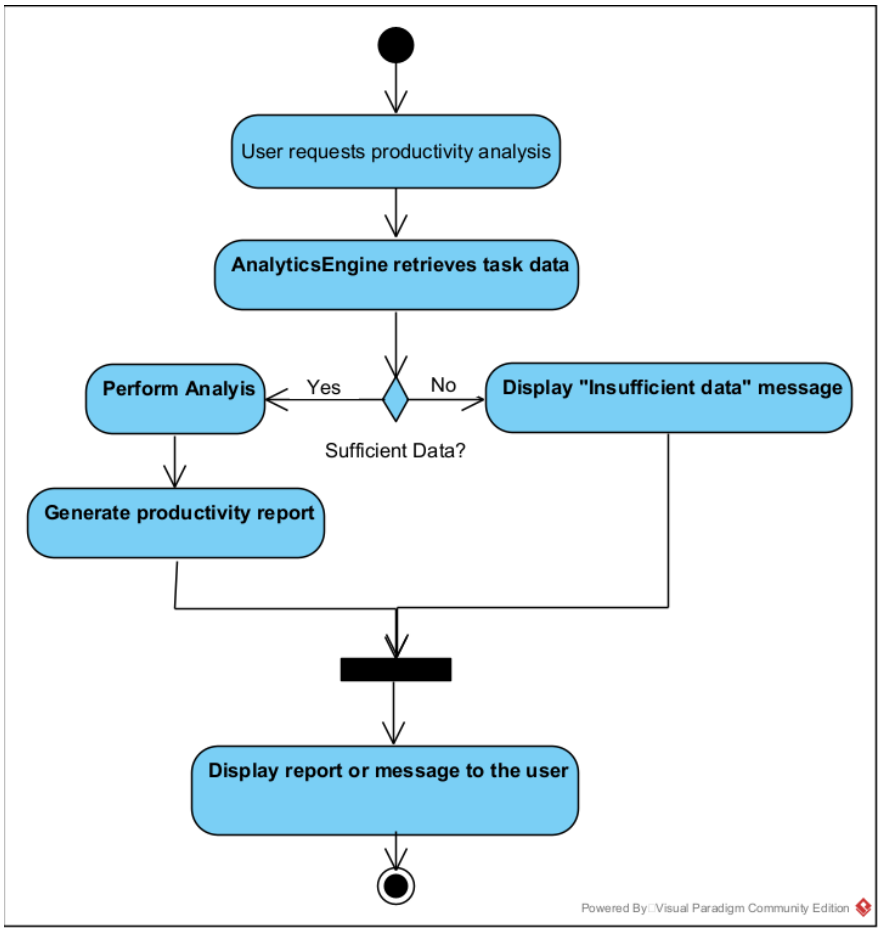

Figure 5.3: Activity Diagram for Analyze Productivity Patterns: Demonstrates the process of analyzing tracked tasks and generating insights.

5.2.3 Deployment Diagram

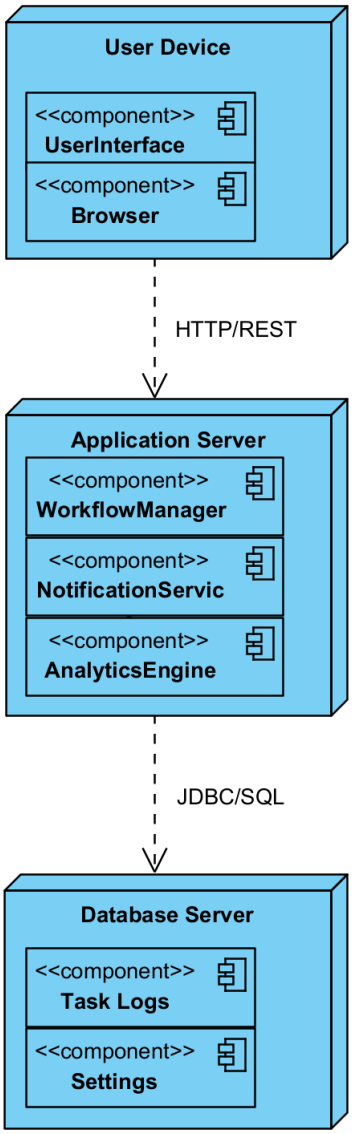

Figure 5.4: Deployment Diagram: Depicts the distribution of components across user devices, application servers, and analytics servers with secure communication.

5.3 Component Descriptions

The architecture comprises essential components designed for cohesive operation across devices and applications. Each component has defined responsibilities and interfaces to support Productivity Pal's core functionalities. These include data handling, real-time responses to user commands, and comprehensive analytics capabilities.

5.3.1 Implementation Details

The core components are implemented with clear responsibilities and interfaces:

Analytics Engine

```
class AnalyticsEngine:
    def __init__(self, task_tracker):
        self.task_tracker = task_tracker
        self.productivity_data = []

    def analyze_productivity(self) -> str:
        current_task = self.task_tracker.get_current_task()
        if current_task:
            return f"Currently working on: {current_task}"
        return "No active task"
```

Listing 5.1: AnalyticsEngine Implementation

Task Tracker

```
from datetime import datetime

class TaskTracker:
    def __init__(self):
```

```
5           self.current_task = None
6           self.start_time = None
7
8       def start_task(self, task_name: str):
9           self.current_task = task_name
10          self.start_time = datetime.now()
11
12      def end_task(self):
13          self.current_task = None
14          self.start_time = None
```

Listing 5.2: TaskTracker Implementation

5.3.2 Basic Workflow

1. **User Interaction:** The `UserInterface` accepts commands from the user, as shown in the following implementation:

```
1       def process_command(self, command: str) -> str:
2           if command == "focus":
3               return self.workflow_manager.switch_mode("focus")
4           elif command == "meeting":
5               return self.workflow_manager.switch_mode("meeting")
6           elif command == "status":
7               return self.analytics_engine.analyze_productivity()
```

Listing 5.3: UserInterface Command Processing

2. **Workflow Switching:** The `WorkflowManager` validates the mode and creates a `Mode` instance:

```
1       class Mode:
2           def __init__(self, name: str, app_manager):
3               self.name = name
```

```
4             self.app_manager = app_manager
5             self._configure_apps()
6
7         def _configure_apps(self):
8             if self.name == "focus":
9                 self.app_manager.set_apps(["Notion", "Slack"])
10            elif self.name == "meeting":
11                self.app_manager.set_apps(["Zoom", "Outlook"])
```

Listing 5.4: Mode Implementation

3. **Notification Management:** The `NotificationService` handles system notifications:

```
1     class NotificationService:
2         def __init__(self):
3             self.enabled = True
4
5         def notify(self, message: str):
6             if self.enabled:
7                 print(f"Notification: {message}")
8
9         def toggle(self):
10            self.enabled = not self.enabled
11            return "Notifications " + ("enabled" if self.enabled else "
                  disabled")
```

Listing 5.5: NotificationService Implementation

5.4 Design Patterns

5.4.1 Singleton Pattern

The `WorkflowManager` implements the Singleton design pattern to ensure that only one instance of the manager exists at any given time. This centralizes control of workflow switching and ensures consistent application behavior.

Implementation Details:

```
class WorkflowManager:
    _instance = None

    @staticmethod
    def get_instance():
        if WorkflowManager._instance is None:
            WorkflowManager._instance = WorkflowManager()
        return WorkflowManager._instance

    def __init__(self):
        if WorkflowManager._instance is not None:
            raise Exception("This class is a singleton!")
        self.current_mode = None
        self.app_manager = ApplicationManager()
        self.notification_service = NotificationService()

    def switch_mode(self, mode_name: str) -> str:
        self.current_mode = Mode(mode_name, self.app_manager)
        self.notification_service.notify(f"Switched to {mode_name} mode")
        return f"Switched to {mode_name} mode"
```

Listing 5.6: Complete Singleton Implementation

Key Benefits:

- Ensures a single point of control for workflow management
- Maintains consistent state across the application
- Provides a global access point to the workflow manager instance
- Prevents multiple instances from causing conflicts in mode management

Chapter 6

Additional UML Diagrams

Hazem Abo-Donia, Benjamin Moks

6.1 Profile Diagram

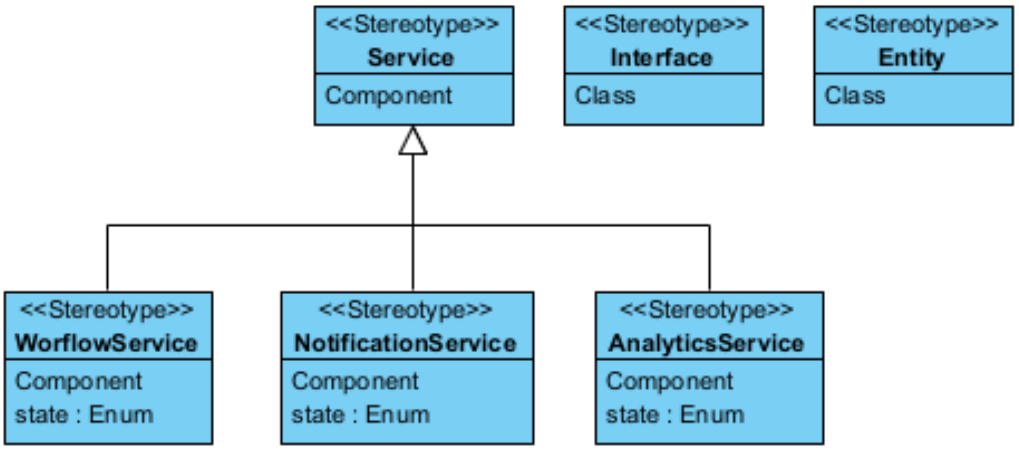

Figure 6.1: Profile Diagram: Showing stereotypes and relationships used in the system.

6.2 Component Diagram

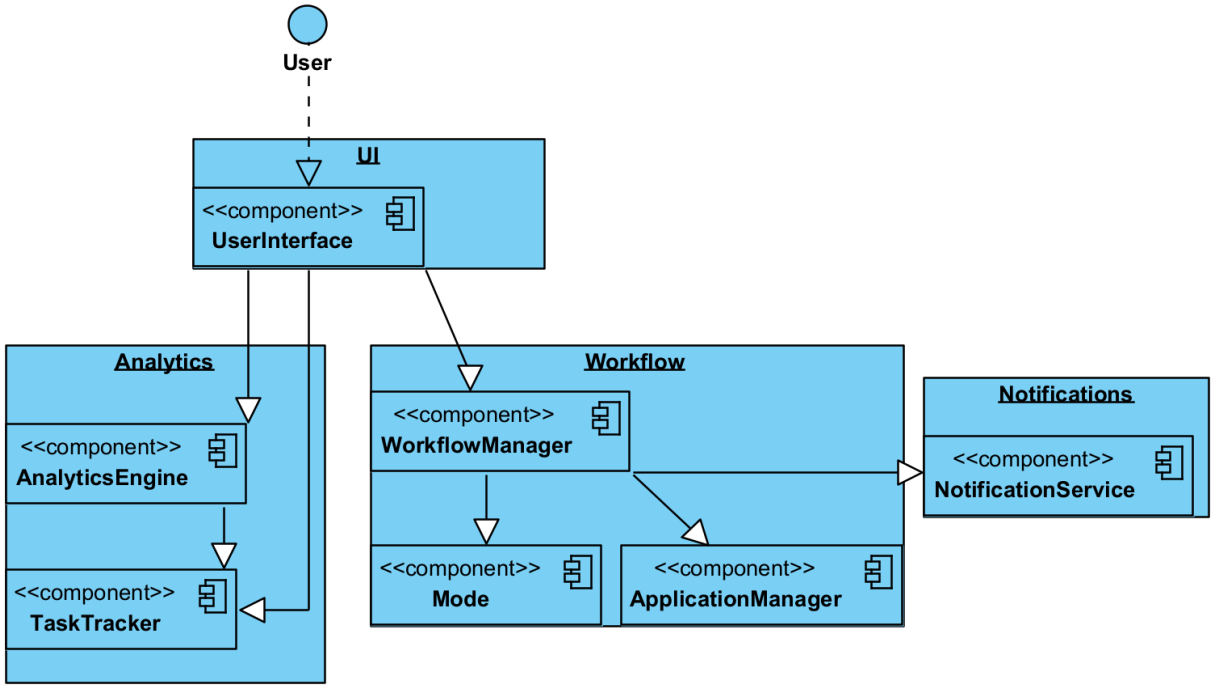

Figure 6.2: Component Diagram: Highlighting the system components and their interactions.

6.3 Deployment Diagram

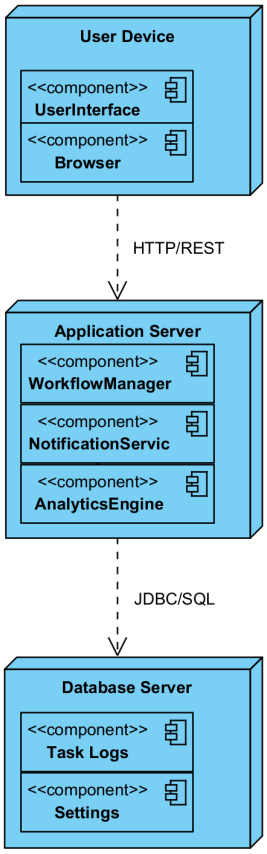

Figure 6.3: Deployment Diagram: Depicting how the system is deployed across devices and servers.

6.4 Package Diagram

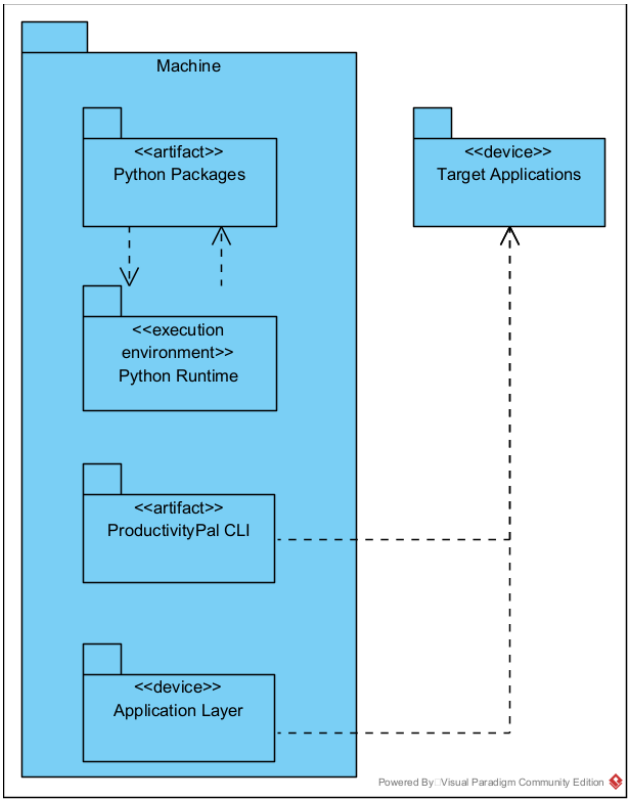

Figure 6.4: Package Diagram: Grouping related classes and packages for system organization.

6.5 Object Diagram

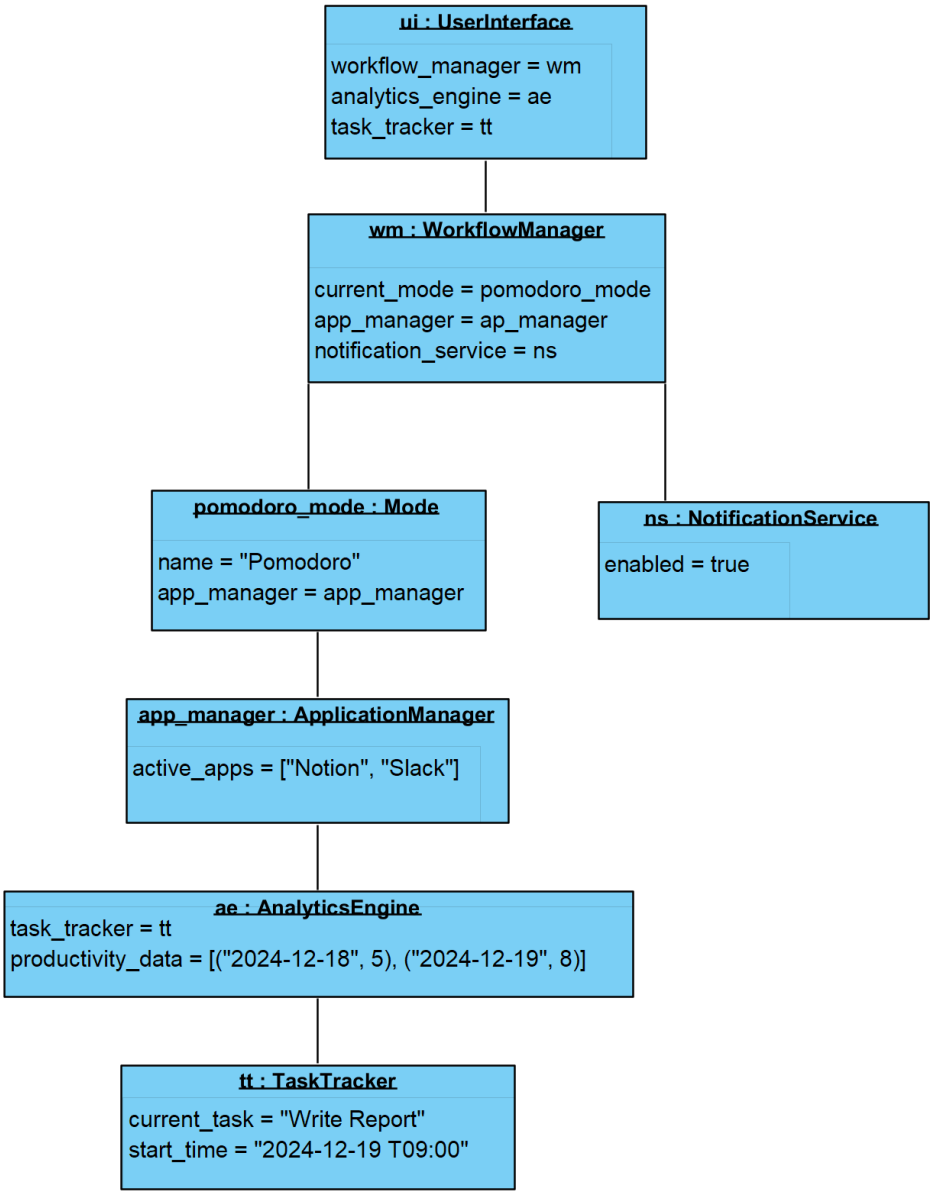

Figure 6.5: Object Diagram: Providing a snapshot of objects and their relationships at runtime.

6.6 Composite Structure Diagram

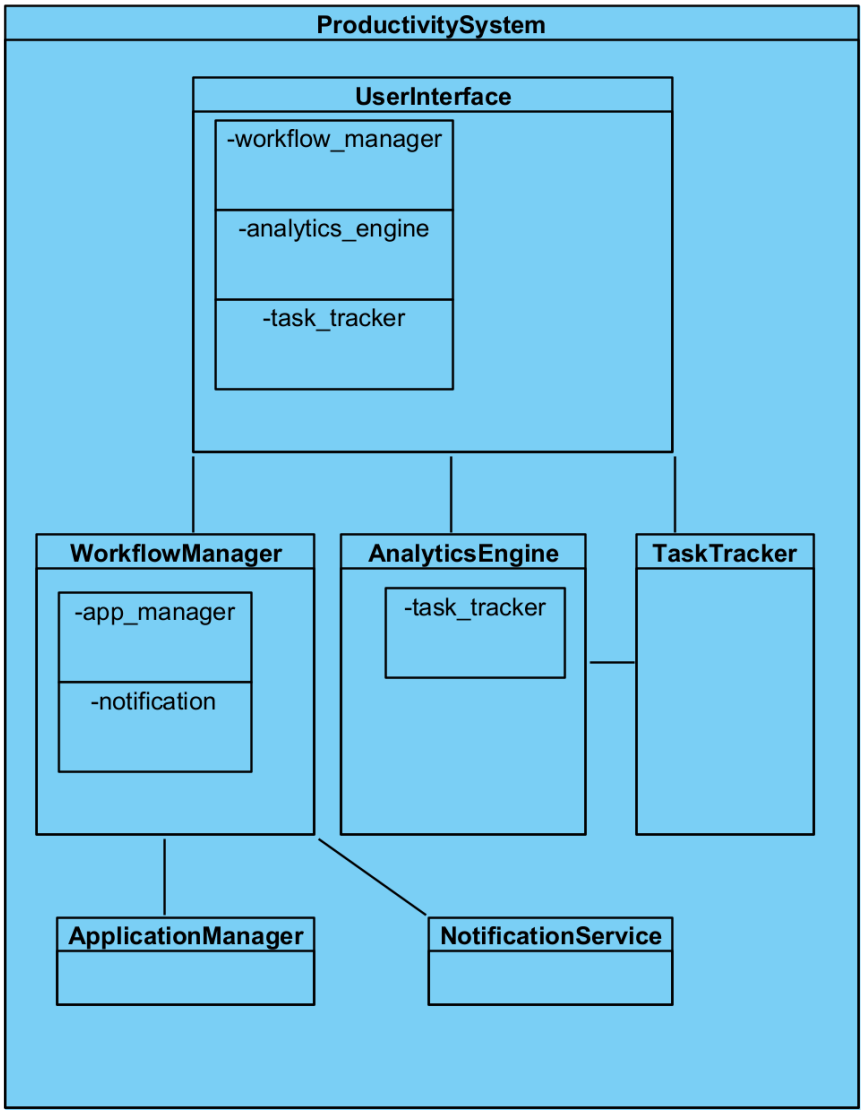

Figure 6.6: Composite Structure Diagram: Demonstrating the internal structure of system components.

6.7 Interaction Diagram

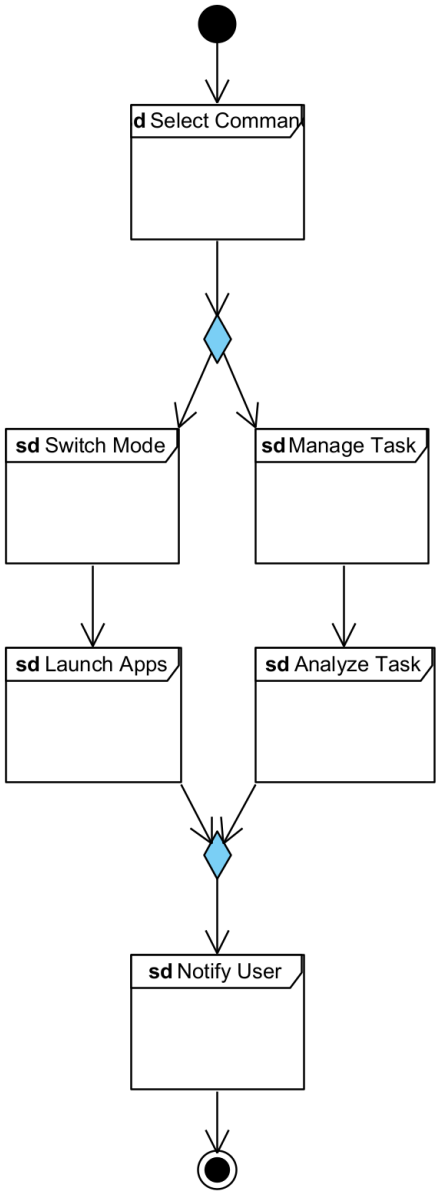

Figure 6.7: Interaction Diagram: Illustrating dynamic interactions between objects.

6.8 Communication Diagram

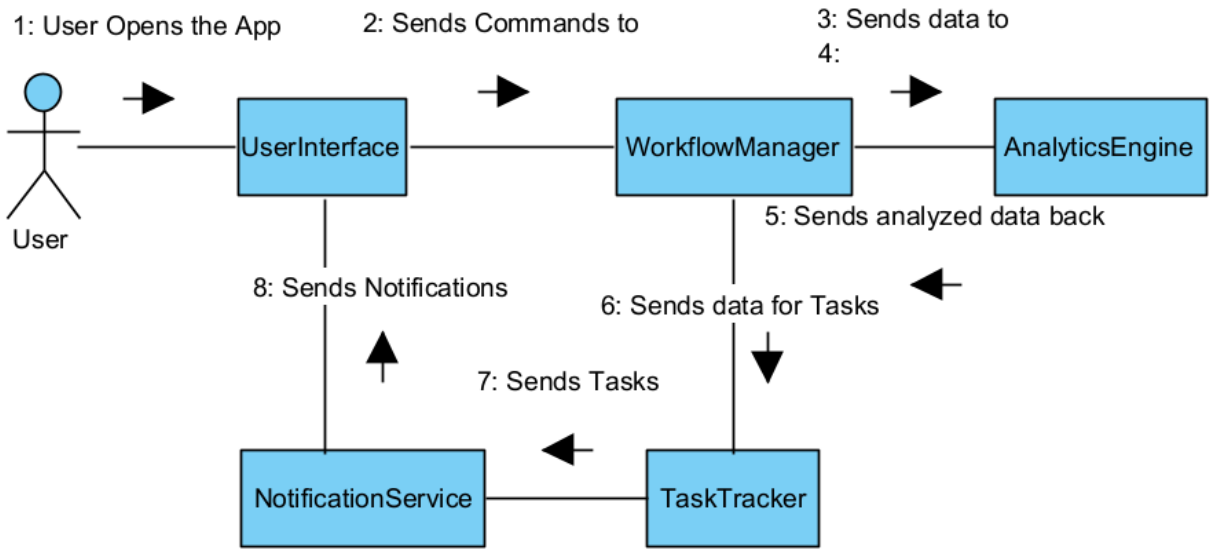

Figure 6.8: Communication Diagram: Highlighting the message exchange between system components.

6.9 Timing Diagram

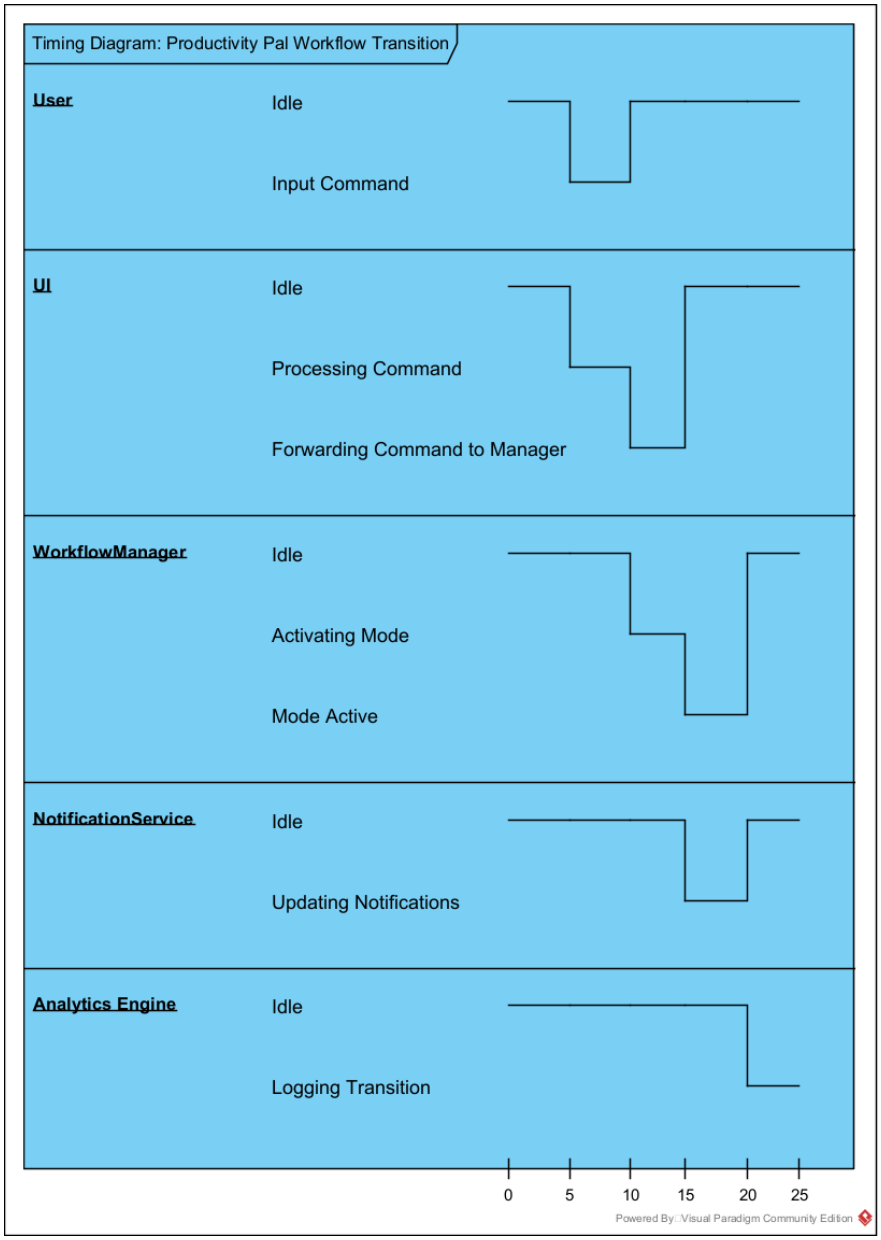

Figure 6.9: Timing Diagram: Showing the time constraints and interactions for key operations.

6.10 State Machine Diagram

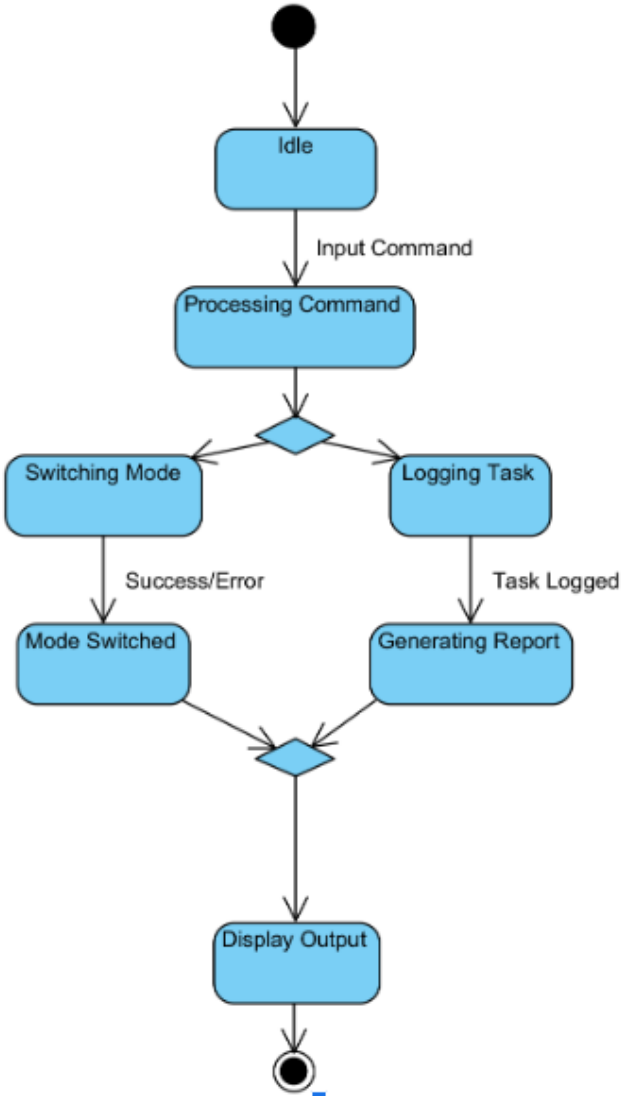

Figure 6.10: State Machine Diagram: Depicting the lifecycle and state transitions for tasks.

Chapter 7

Project Development

Hazem Abo-Donia, Benjamin Moks

7.1 Iteration Overview

This section highlights the goals, achievements, and challenges faced during the first and final iterations of the project. You can access the full repository of our project here: GitHub Repository.

7.1.1 Iteration 1 Goals

The primary goals for the first iteration were:

- Implement the `Switch Workflow Mode` use case.
- Introduce the Singleton design pattern for the `WorkflowManager`.
- Expand and ensure compatibility of unit tests.

7.1.2 Iteration 1 Achievements

The team successfully accomplished the following during the first iteration:

- Refactored the `WorkflowManager` to enforce the Singleton pattern.
- Enhanced `test_workflow.py`:
  - Adjusted tests to use `WorkflowManager.get_instance()`.
  - Added test isolation by resetting the Singleton instance before each test.
- Verified all tests passed using `pytest`.

7.1.3 Iteration Challenges

While the team achieved significant progress, several challenges arose:

- Faced `ModuleNotFoundError` due to Python path issues, which were resolved by updating the `sys.path` configuration in the test files.
- Encountered Singleton-related test failures, which were addressed by resetting the `_instance` variable during testing to ensure proper test isolation.

7.1.4 Final Sprint Goals

The primary goals for the final sprint were:

- Implement the `Analyze Productivity Patterns` use case with real-time task tracking.
- Integrate `TaskTracker` with `AnalyticsEngine` for productivity insights.
- Enhance the `NotificationService` to provide troubleshooting support.
- Refactor UML diagrams based on teacher feedback.
- Finalize unit tests with 100% coverage.

7.1.5 Final Sprint Achievements

The team successfully accomplished the following during the final sprint:

- Implemented the `TaskTracker` to enable real-time task monitoring.
- Integrated `TaskTracker` with `AnalyticsEngine` to generate productivity reports and insights.
- Enhanced `NotificationService` to provide troubleshooting steps for failed notifications.
- Refactored UML diagrams to reflect the updated system architecture.
- Ensured all tests passed using `pytest`, achieving 100% test coverage.

7.1.6 Final Sprint Challenges

While the final sprint was successful, a few challenges were encountered:

- Integration of `TaskTracker` with `AnalyticsEngine` required debugging due to data inconsistencies during early testing.
- Refactoring UML diagrams to align with teacher feedback took longer than expected due to necessary architecture changes.
- Managing multiple simultaneous tasks while maintaining quality standards.

7.2 Process and Tools

This section outlines the Agile practices followed and the tools used for project management and collaboration.

7.2.1 Agile Practices

The team adhered to Agile principles by organizing the project into iterative tasks and leveraging tools to ensure collaboration. Key practices included:

- Using a GitHub Project Board to manage tasks, track progress, and ensure accountability.
- Dividing work based on team members' strengths:
  - **Hazem:** Programming and implementing the `TaskTracker` and enhancing the `NotificationService`.
  - **Ben:** Integration of `TaskTracker` with the `AnalyticsEngine` and refactoring UML diagrams.

7.2.2 Tasks Breakdown for Iteration 1

Table 7.1: Tasks Breakdown for Iteration 1

Task	**Story Points**	**Assigned To**	**Status**
Refactor Initial Implementation of Workflow Switching	3	Hazem	Review/Testing
Review Initial Analytics Data Collection	3	Ben	Review/Testing
Set Up Notification Configuration	5	Ben	In Progress
Write Unit Tests for Workflow Switching	3	Hazem	In Progress

7.2.3 GitHub Storyboard for Iteration 1

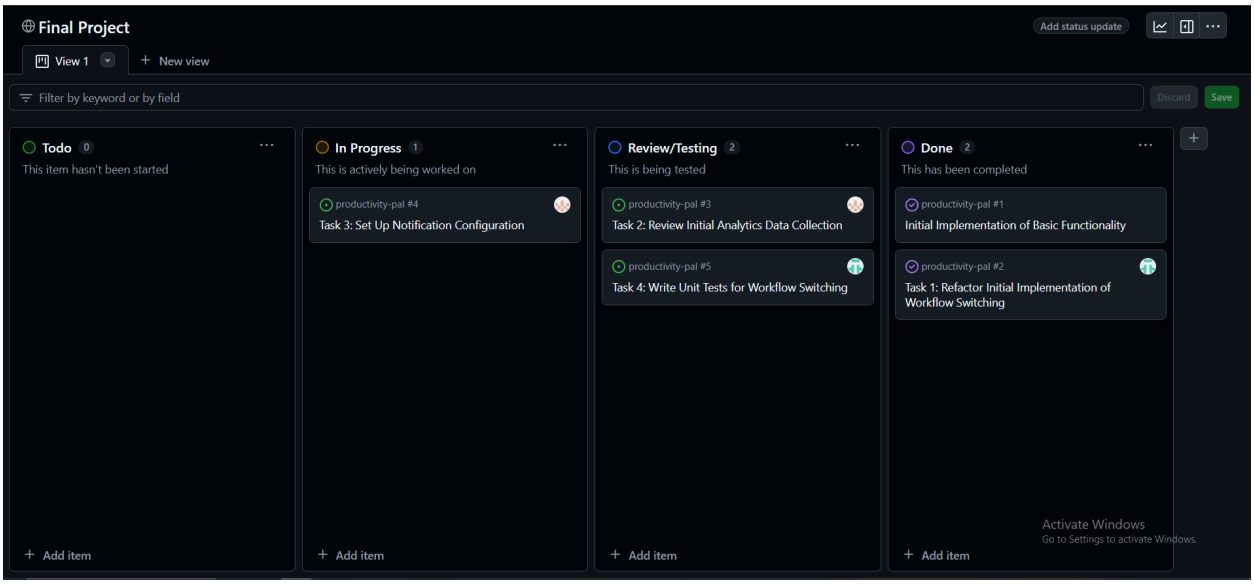

Figure 7.1: GitHub Project Board for Iteration 1: Tracking progress and task completion.

7.2.4 Tasks Breakdown for Final Sprint

Table 7.2: Tasks Breakdown for Final Sprint

Task	**Story Points**	**Assigned To**	**Status**
Implement TaskTracker for Real-Time Task Monitoring	5	Hazem	Completed
Integrate TaskTracker with AnalyticsEngine	5	Ben	Completed
Enhance NotificationService with Troubleshooting	3	Ben	Completed
Refactor Class Diagram Based on Feedback	2	Hazem	Completed
Write Unit Tests for TaskTracker and AnalyticsEngine	4	Hazem	Completed
Finalize Documentation	3	Both	Completed

7.2.5 GitHub Storyboard for Final Sprint

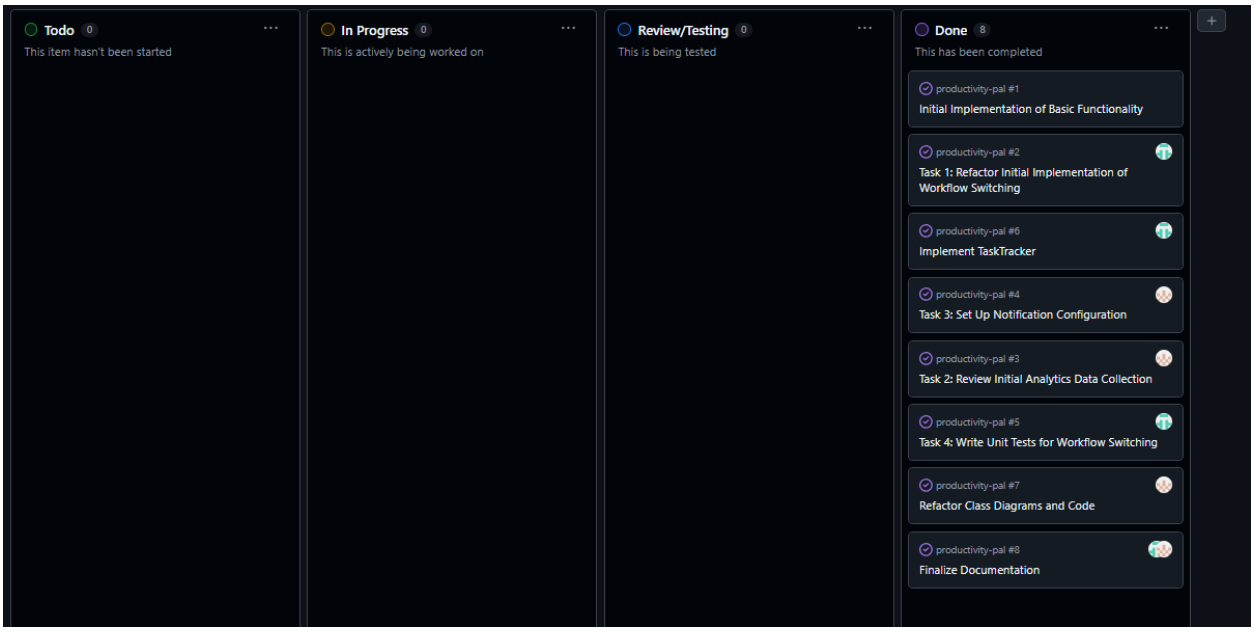

Figure 7.2: GitHub Project Board for Final Sprint: Tracking final sprint goals and tasks.

7.2.6 Task Tracking

The GitHub Project Board was used to visualize and track the progress of tasks. Each task was assigned story points to indicate its complexity, and team members updated the board regularly to reflect completed and in-progress tasks.

7.2.7 Meeting Notes: Iteration Planning

- **Date:** December 1, 2024
- **Attendees:** Hazem Abo-Donia, Benjamin Moks
- **Agenda:**
  - Define iteration goals and assign tasks.
  - Discuss challenges and potential blockers for implementing the Singleton pattern.

  - Plan testing and verification steps for `WorkflowManager`.

- **Decisions Made:**
  - Hazem will handle programming tasks, including refactoring `WorkflowManager` and running unit tests.
  - Ben will focus on modularizing the code and writing test cases for Singleton compatibility.

- **Action Items:**
  - Hazem to implement the Singleton pattern in `WorkflowManager`.
  - Ben to update test cases and prepare for integration testing.

7.2.8 Meeting Notes: Sprint Planning

- **Date:** December 10, 2024
- **Attendees:** Hazem Abo-Donia, Benjamin Moks
- **Agenda:**
  - Define sprint goals and assign tasks.
  - Discuss testing plans for `TaskTracker` integration.
  - Review UML diagrams for alignment with final architecture.

- **Decisions Made:**
  - Hazem to implement and test the `TaskTracker`.
  - Ben to integrate the `TaskTracker` with the `AnalyticsEngine`.

- **Action Items:**
  - Hazem to complete real-time task monitoring.
  - Ben to finalize UML refactoring and testing.

7.3 Reflection

7.3.1 What Went Well

- Iteration 1 successfully introduced the `Switch Workflow Mode` use case.
- Final sprint implemented `Analyze Productivity Patterns` with real-time task tracking.
- Enhanced `NotificationService` for better user experience.

7.3.2 Challenges Faced

- Iteration 1: Module import issues required significant debugging.
- Final Sprint: Integration testing exposed data consistency bugs, delaying progress.

Chapter 8

Reflection

Hazem Abo-Donia, Benjamin Moks

8.1 Overview

Working on Productivity Pal was a fun and challenging experience that allowed us to use a mix of skills, teamwork, and problem-solving to build a tool that improves productivity. As we worked through each stage of the project, we faced obstacles, created solutions, and made the system better step by step. In this chapter, we'll share what we accomplished, the challenges we overcame, and the lessons we learned along the way.

8.2 Achievements

We were able to build a system that simplifies workflows and analyzes productivity data effectively. Here are some of the highlights:

- **Switch Workflow Mode**: We created a reliable workflow manager using the Singleton design pattern. This lets users quickly switch between modes like Focus Mode and Meeting Mode with ease.
- **Analyze Productivity Patterns**: By combining the `TaskTracker` and `AnalyticsEngine`, we provided real-time task tracking and meaningful productivity insights.

- **UML Diagrams**: We designed and improved diagrams like Class, Activity, Deployment, and Use Case diagrams. These were crucial for understanding and explaining the system's structure.
- **Agile Workflow**: Using GitHub Project Boards helped us stay organized, track our progress, and meet deadlines for each milestone.
- **Comprehensive Testing**: We achieved 100% test coverage on core components with `pytest`, which ensured that the system works as expected.

8.3 Challenges Faced

Even though the project went well, we did face some hurdles:

- **Import Errors**: At the start, we had Python module import errors that delayed testing. We fixed this by reorganizing the project and adjusting the `sys.path`.
- **Integration Bugs**: When combining `TaskTracker` and `AnalyticsEngine`, we found data inconsistencies that required extra debugging and testing to fix.
- **Improving UML Diagrams**: Feedback on our diagrams meant we had to go back and make significant changes to make them clearer and more accurate. This took extra time.
- **Time Pressure**: Juggling diagram updates, testing, and documentation during the final sprint was tough, but we managed to finish everything on time.

8.4 Lessons Learned

This project taught us a lot about software development, teamwork, and staying organized:

- **Plan Early**: Breaking down tasks and assigning roles early made it easier to stay on track and avoid surprises.

- **Be Flexible**: Adapting to feedback and changes helped us improve the system and deliver better results.
- **Stay in Sync**: Frequent meetings and updates made sure we were always on the same page and avoided miscommunication.
- **Test Thoroughly**: Testing every part of the system gave us confidence in its stability and reliability.

8.5 Future Work

Even though the system meets its goals, there are ways to make it even better:

- **Better Analytics**: Add visual dashboards and trends to the `AnalyticsEngine`.
- **More Integrations**: Support more third-party apps to make the system useful to a wider range of users.
- **Smarter Troubleshooting**: Improve the `NotificationService` to automatically handle common problems.
- **Collaborative Features**: Add options for multiple users to work together on tasks in real time.

8.6 Conclusion

Building Productivity Pal was a valuable experience that combined problem-solving, coding, and teamwork. We're proud of the system we created, and we learned a lot from both the challenges and successes. These lessons will help us in future projects as we tackle even more complex software development tasks.

Bibliography

[1] GitHub, "Github project boards," 2024, accessed December 7, 2024. [Online]. Available: https://github.com/users/habodoni/projects/1/views/1

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[3] pytest, "Introduction to pytest," 2024, accessed December 7, 2024. [Online]. Available: https://docs.pytest.org/en/stable/

Index

Chapter

- Architecture, 25
- ProjectDescription, 11
- sswAdditionalUML, 36
- sswMilestone, 46
- sswReflection, 55
- Stories, 13
- UIDesignSketches, 22
- UseCases, 16