Project Milestone 3

This project milestone will focus on the design of your system.

Process Deliverable II (3%)

Our team chose to use the incremental model, allowing us to develop the system in increments where each delivers functionality for a specific prioritized requirement. This approach allows us to deliver features earlier, and provides opportunities for continuous feedback.

Increment 1: Centralized Access and Permissions Management

Description:

This increment addressed the centralized access and permissions management feature, which was noted as a high-priority requirement. This feature enables new developers to request and automatically gain access to essential project resources.

Design Overview:

1. Login and Access Request Interface: The system will provide a user-friendly interface for new developers to login and request access.
2. Automated Permission Logic: A back end process will automate the granting of access based on pre-registered profiles created by the project manager.
3. Notification System: Developers and managers will be notified once access has successfully been granted.

Increment 2: Automated Development Environment Setup

Description:

This increment automates the setup of a development environment for new software engineers, which was noted as a high-priority requirement.

Design Overview:

1. Environment Configuration Interface. The system allows the project manager to define requirements for the new developer's setup.
2. Automation Scripts: Scripts will be created to automate setup tasks such as cloning repositories, installing dependencies, and configuring environment variables.

3. Status Dashboard: Progress of the system setup will be displayed to the new developer and project manager.

Increment 3: Automated Codebase Documentation Overview

Description:

This increment builds a feature in our system which generates a high-level overview of the new developer's project codebase, helping them understand its structure in a quicker manner. This was noted as a medium-priority requirement.

Design Overview:

1. Document Parsing Engine: Automatically extracting information about the codebase structure and functionality.
2. Visualization Tool: Provides a graphical representation of the codebase structure.
3. Search and Query Feature: Allows users to search and filter for specific components or relationships within the codebase.

Increment 4: Task Progress Tracking System

Description:

This increment builds a feature in our system which provides real-time tracking during the onboarding process for new hires. This was noted as a medium-priority requirement.

Design Overview:

1. Progress Dashboard: Displays and categorizes current tasks as completed, in-progress, or pending with sufficient descriptions.
2. Notifications: Sends new hires reminders regarding upcoming or overdue tasks.
3. Task Integration: Links process tracking with the onboarding system for seamless integration.

High-level Design (4%)

Our group decided that we would use the architectural pattern of event-driven programming. This high level design pattern focuses on the production, detection, consumption of, and response to different events and would be most appropriate for the structure of the system because the system only needs to change or update when the user requests it. Additionally, this

pattern would promote the use of a graphical user interface in which the user can interact with to cause/trigger the events. The different tools being added to the development environment do not necessarily need to have communication with each other so a hierarchical design isn't required. Similarly, since the system only needs to be updated when the user needs it, there is no need for a pipeline or pipe and filter structure that updates the system on regular intervals.

Using an event-driven programming structure for the system also promotes adaptability because the system can add or remove different development tools in response to different actions. Additionally, this architectural pattern facilitates adding our ability to add new features to the Onboarding Manager which is important for our incremental model as we would plan to continually add new tools. Different events such as request for access to different parts of the application such as the documentation database, or a user trying to add or remove a development tool from the work environment would elicit a response from the system that would change the user's view or work environment configuration. Overall, the architectural pattern of event-driven programming best suits our idea for the Onboarding Manager.

Low-level Design (4%)

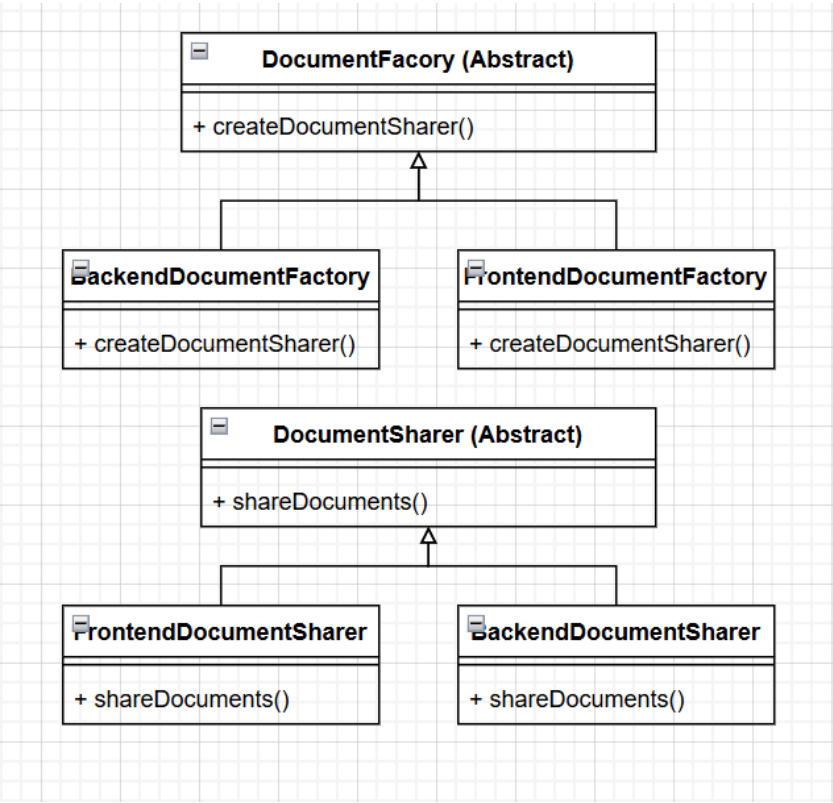
For the low-level design we decided to use the Creational Design Pattern Family, particularly the Abstract Factory pattern, for implementing the automation of document sharing in the onboarding manager. This pattern is well suited for scenarios where multiple variations of a task, such as sharing different sets of documents based on a developer's role, are required. The Abstract Factory allows us to create families of related objects without specifying the exact class of object that will be created. By using this pattern, we can dynamically generate the appropriate document-sharing task (frontend, backend, or general documents) based on the developer's needs, ensuring a streamlined and flexible onboarding experience.

Pseudocode for Document Sharing using Abstract Factory

1. Abstract Factory
    - Define an abstract class or interface DocumentFacotry with a method createDocumentSharer(). This method will return a DocumentSharer object, but the exact implementation of DocumentSharer depends on the specific factory.
2. Concrete Factories
    - Create a FrontendDocumentFactory class that implements DocumentFactory. This factory will create a FrontendDocumentSharer.

    - Create a BackendDocumentFactory class that implements DocumentFactory. This factory will create a BackendDocumentSharer.
3. Abstract Product
    - Define an abstract class or interface DocumentSharer with a method shareDocments(). This method will be implemented differently by each specific type of document sharer.
4. Concrete Product
    - Create a FrontEndDocumentSharer class that implements DocumentSharer. This class will handle sharing frontend-specific documents.
    - Create a BackEndDocumentSharer class that implements DocumentSharer. This class will handle sharing backend-specific documents.
5. Client Code
    - The client (onboarding manager) will decide which type of document sharing is needed.
    - The client will instantiate the appropriate DocumentFactory.
    - The client will call createDocumentSharer() on the factory to the the appropriated DocumentSharer object.
    - The client will call shareDocuments() on the DocumentSharer to share the correct set of documents with the developer.

Informal Class Design



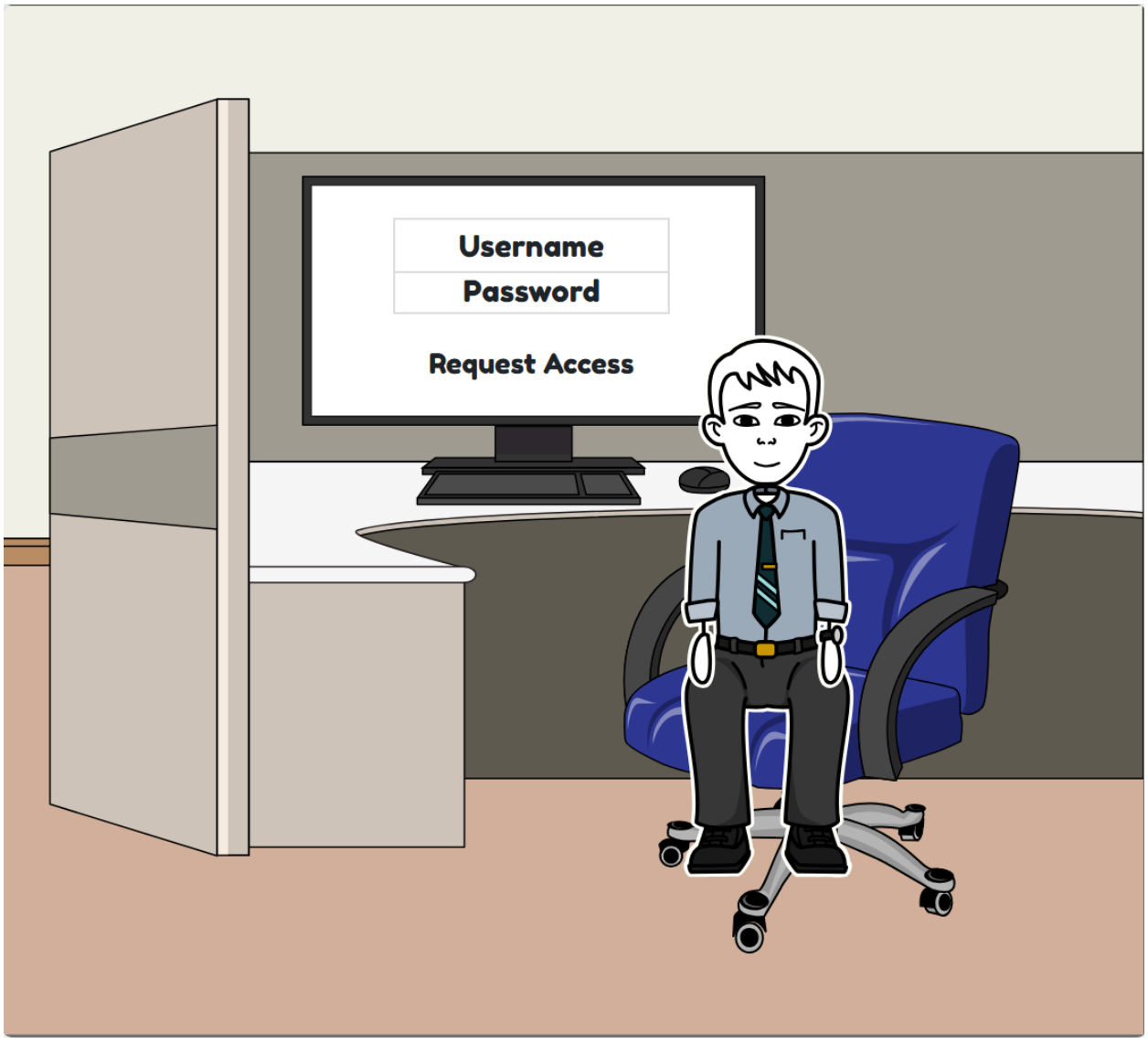
Design Sketch (4%)

Design sketches provide a visual overview of the look and feel of your project. This may include but is not limited to one of the following:

- Creating a wireframe mockup of your project user interface in action.
- Creating a storyboard that illustrates a primary task that a user would complete with your project.

Storyboard:

Scene 1: Login and Access Request

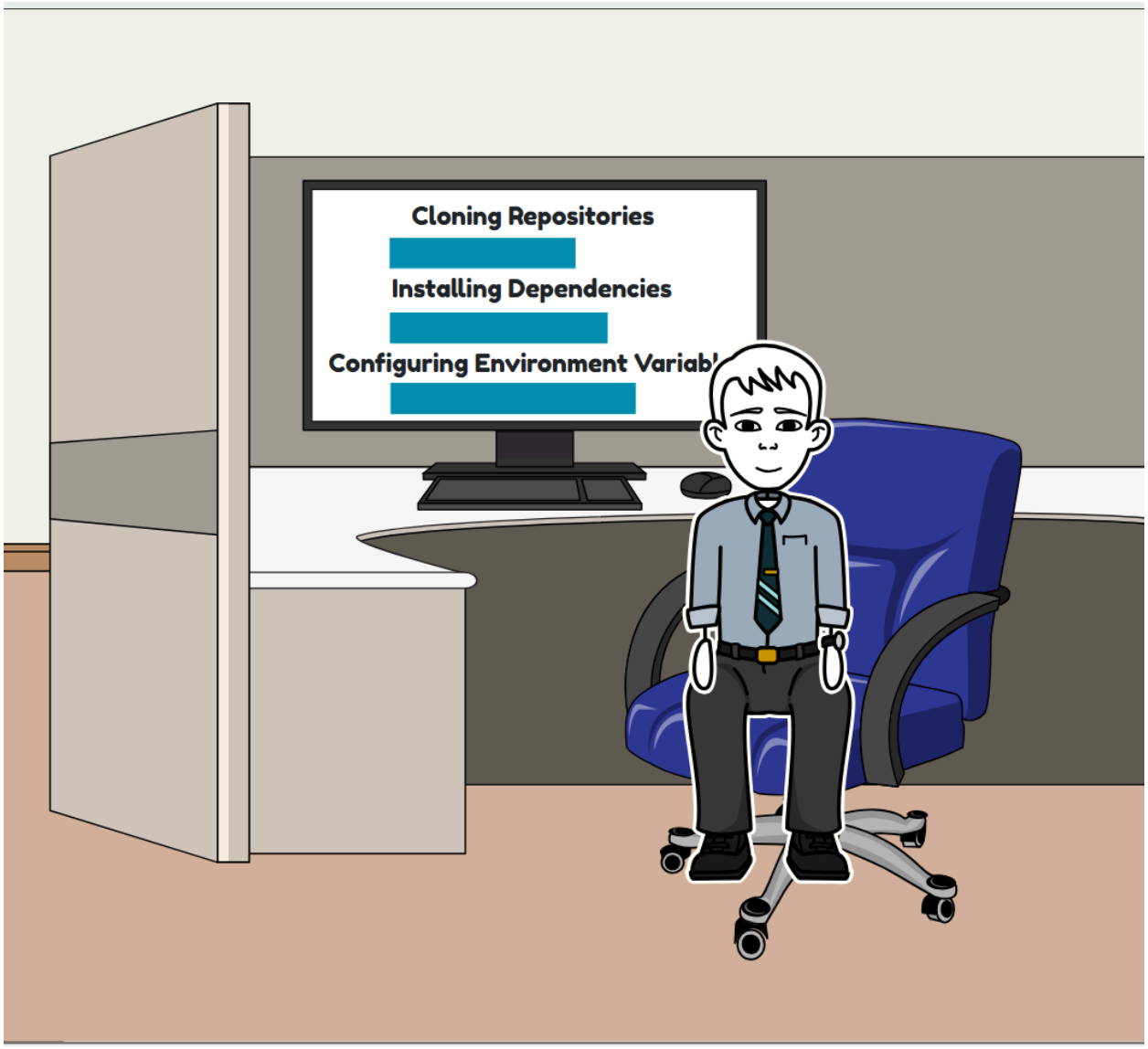


The developer logs into the Onboarding Manager and requests access to project resources.

Scene 2: Automated Permissions Management

The system verifies the developer's profile and grants them permissions automatically. Notifications are sent to both the developer and the project manager.

Scene 3: Automated Development Environment Setup



The system begins setting up the development environment. A progress dashboard keeps the developer informed.

Scene 4: Developer Ready to Work

With all tasks completed, the developer is fully onboarded and ready to contribute to the project.

Provide a brief rationale (no more than 1 paragraph) explaining some of the design decisions for your program based on the provided sketch. Use concepts discussed in class to justify your design.