Contents

1. Define Organizational Structure

- **Dependency**: None
- **Action**: Define the roles and reporting relationships based on the Standard Workplace Definition.

2. Implement Role-Based Access Control (RBAC)

- **Dependency**: Defined Organizational Structure
- **Action**: Map permissions to each role according to the organizational hierarchy.

3. Build Task Management System

- **Dependency**: Implemented RBAC
- **Action**: Develop the task management system with interfaces for task creation, assignment, and tracking, ensuring access control based on roles.

4. Integrate Communication and Collaboration Tools

- **Dependency**: Built Task Management System
- **Action**: Integrate messaging systems, email notifications, and calendar tools, aligning communication channels with task management workflows.

5. Establish Resource Management Processes

- **Dependency**: Built Task Management System
- **Action**: Set up the document repository with version control, implement budget allocation mechanisms, and integrate resource management features into the task management system.

6. Design Performance Evaluation Framework

- **Dependency**: Built Task Management System
- **Action**: Define KPIs, develop performance monitoring processes, and integrate performance evaluation features into the task management system.

7. Implement Change Management Procedures

- **Dependency**: Built Task Management System
- **Action**: Establish change control processes, define roles and responsibilities for change management, and integrate change management features into the task management system.

8. Foster a Culture of Continuous Improvement

- **Dependency**: Built Task Management System
- **Action**: Encourage employee participation in process improvement initiatives, develop mechanisms for evaluating and implementing improvement ideas, and integrate continuous improvement features into the task management system.

9. Test and Iterate

- **Dependency**: Completed all Previous Steps
- **Action**: Conduct comprehensive testing of the workplace model, gather feedback from users, and iterate on the model based on testing results and user feedback.

10. Documentation and Training

- **Dependency**: Completed all Previous Steps
- **Action**: Document the workplace model, including processes, procedures, and best practices. Provide training and support to users to ensure effective adoption of the model.

1.Final Organizational Structure:

Roles:

1. **Administrator**
2. **Manager**
3. **Design Team**
4. **Development Team**
5. **Auxiliary Employee**

Reporting Relationships:

- **Administrator**: Oversees the entire organization.
- **Manager**: Reports to the Administrator; responsible for specific teams or departments.
- **Design and Development Teams**: Report to their respective Managers for task assignments and project oversight.
- **Auxiliary Employees**: Report directly to Managers or are assigned to specific teams as needed.

Hierarchical Levels:

- **Executive Level**: Administrator
- **Operational Level**: Managers, Design Team, Development Team
- **Support Level**: Auxiliary Employees

Pseudocode Implementation Plan:

Classes and Interfaces:

1. **Employee**: Base class representing an employee with common attributes and methods.
2. **Manager**: Subclass of Employee representing a manager with additional methods for team management.
3. **Team**: Interface representing a team, implemented by DesignTeam and DevelopmentTeam classes.
4. **DesignTeam**: Class representing the design team with methods for design-related tasks.
5. **DevelopmentTeam**: Class representing the development team with methods for development-related tasks.
6. **AuxiliaryEmployee**: Class representing an auxiliary employee with methods for supporting other teams.

1.1 Employee

Attributes:

1. **Employee ID**: A unique identifier for each employee within the organization.
2. **Name**: The full name of the employee.
3. **Role**: The role or position of the employee within the organization.
4. **Department**: The department to which the employee belongs.
5. **Email Address**: The email address of the employee for communication purposes.
6. **Phone Number**: The phone number of the employee for contact purposes.
7. **Address**: The physical address of the employee.
8. **Supervisor ID**: The Employee ID of the immediate supervisor or manager of the employee.
9. **Joining Date**: The date when the employee joined the organization.
10. **Salary**: The salary or compensation package of the employee.
11. **Employment Status**: Indicates whether the employee is active, on leave, terminated, etc.
12. **Skills**: A list of skills or competencies possessed by the employee.
13. **Tasks Assigned**: A list of tasks currently assigned to the employee.
14. **Performance Ratings**: Ratings or evaluations of the employee's performance.
15. **Training History**: Record of training sessions attended by the employee.

Methods:

1. **Constructor**: A constructor method to initialize the attributes of the Employee object.
2. **Getters and Setters**: Getter methods to retrieve the values of attributes and setter methods to set or update the values.
3. **Calculate Experience**: A method to calculate the experience of the employee based on the joining date.
4. **Update Contact Information**: A method to update the contact information of the employee.
5. **Assign Task**: A method to assign tasks to the employee.
6. **Update Employment Status**: A method to update the employment status of the employee.
7. **Provide Feedback**: A method to provide feedback on the performance of the employee.
8. **Request Leave**: A method to request leave or time off from work.
9. **View Performance Ratings**: A method to view performance ratings or evaluations.
10. **Attend Training**: A method to mark attendance in training sessions.
11. **Calculate Salary**: A method to calculate the salary of the employee based on the compensation package.

1.2 Manager

Attributes:

1. **Employee ID**: A unique identifier for each manager within the organization.
2. **Name**: The full name of the manager.
3. **Role**: The role or position of the manager within the organization.
4. **Department**: The department to which the manager belongs or oversees.
5. **Email Address**: The email address of the manager for communication purposes.
6. **Phone Number**: The phone number of the manager for contact purposes.
7. **Address**: The physical address of the manager.
8. **Supervisor ID**: The Employee ID of the manager's immediate supervisor or higher-level manager.
9. **Team**: The team or department managed by the manager.
10. **Direct Reports**: A list of Employee IDs for employees directly reporting to the manager.
11. **Tasks Assigned**: A list of tasks assigned by the manager to employees or teams.
12. **Performance Ratings**: Ratings or evaluations of the manager's performance.
13. **Meeting Schedule**: Schedule of meetings chaired or attended by the manager.
14. **Budget Allocation**: Information about budget allocations managed by the manager for projects or departments.
15. **Project Management Tools**: Tools or software used by the manager for project management and task tracking.

Methods:

1. **Constructor**: A constructor method to initialize the attributes of the Manager object.
2. **Getters and Setters**: Getter methods to retrieve the values of attributes and setter methods to set or update the values.
3. **Assign Task**: A method to assign tasks to employees or teams managed by the manager.
4. **Update Contact Information**: A method to update the contact information of the manager.
5. **Provide Feedback**: A method to provide feedback on the performance of employees or teams managed by the manager.
6. **Schedule Meeting**: A method to schedule meetings chaired or attended by the manager.
7. **Allocate Budget**: A method to allocate budget resources for projects or departments managed by the manager.
8. **View Direct Reports**: A method to view a list of employees directly reporting to the manager.
9. **View Performance Ratings**: A method to view performance ratings or evaluations received by the manager.
10. **View Task Progress**: A method to view the progress of tasks assigned by the manager.
11. **Generate Reports**: A method to generate reports on team performance, project status, and budget utilization.

1.3 Team interface

Attributes:

1. **Team Name**: A unique identifier for the team within the organization.
2. **Team Members**: A list of employees who are part of the team.
3. **Team Leader**: The employee designated as the leader or manager of the team.
4. **Tasks Assigned**: A list of tasks assigned to the team.
5. **Communication Channels**: Channels or tools used for communication within the team (e.g., messaging platform, email, video conferencing).
6. **Collaboration Tools**: Tools or platforms used for collaborative work within the team (e.g., document sharing, version control).
7. **Resource Allocation**: Information about resources allocated to the team, such as budget, equipment, and software.

Methods:

1. **Create Task**: Method to create a new task within the team and assign it to team members.
2. **Assign Task**: Method to assign an existing task to a specific team member.
3. **Update Task Status**: Method to update the status of a task (e.g., in progress, completed, delayed).
4. **Communicate**: Method to facilitate communication among team members using designated channels.
5. **Collaborate**: Method to support collaborative work within the team using designated tools.
6. **Allocate Resources**: Method to allocate resources to the team for project execution.
7. **Track Progress**: Method to track the progress of tasks and projects within the team.
8. **Evaluate Performance**: Method to evaluate the performance of the team based on predefined metrics or KPIs.
9. **Make Decisions**: Method to support decision-making processes within the team, such as voting or consensus building.
10. **Document Activities**: Method to document team activities, decisions, and outcomes for future reference.

1.4 DesignTeam

Attributes:

1. **Team Name**: A unique identifier for the design team within the organization.
2. **Team Members**: A list of employees who are part of the design team.
3. **Team Leader**: The employee designated as the leader or manager of the design team.
4. **Design Projects**: A list of design projects assigned to the team.
5. **Design Tools**: Tools or software used by the design team for creating and editing designs (e.g., Adobe Creative Suite, Sketch).
6. **Project Deadlines**: Deadlines or milestones associated with design projects assigned to the team.
7. **Client Requirements**: Requirements or specifications provided by clients for design projects.
8. **Feedback History**: History of feedback received from clients or stakeholders on completed design projects.

Methods:

1. **Create Design Project**: Method to create a new design project within the team and assign it to team members.
2. **Assign Task**: Method to assign specific design tasks (e.g., graphic design, UI/UX design) to team members.
3. **Update Project Status**: Method to update the status of a design project (e.g., in progress, completed, pending client approval).
4. **Collaborate**: Method to facilitate collaboration among team members during the design process.
5. **Review and Revise**: Method to review design drafts or prototypes, incorporate feedback, and revise designs accordingly.
6. **Communicate with Clients**: Method to communicate with clients or stakeholders regarding project requirements, updates, and feedback.
7. **Track Project Progress**: Method to track the progress of design projects, including task completion and adherence to deadlines.
8. **Evaluate Design Quality**: Method to evaluate the quality of design work based on predefined criteria (e.g., aesthetics, usability).
9. **Generate Reports**: Method to generate reports on design project status, resource utilization, and client satisfaction.

1.5 DevelopmentTeam

Attributes:

1. **Team Name**: A unique identifier for the development team within the organization.
2. **Team Members**: A list of employees who are part of the development team.
3. **Team Leader**: The employee designated as the leader or manager of the development team.
4. **Development Projects**: A list of development projects assigned to the team.
5. **Programming Languages**: Programming languages and technologies used by the development team for software development.
6. **Project Deadlines**: Deadlines or milestones associated with development projects assigned to the team.
7. **Client Requirements**: Requirements or specifications provided by clients or stakeholders for development projects.
8. **Code Repository**: Repository or version control system used by the team for managing source code (e.g., Git, SVN).

Methods:

1. **Create Development Project**: Method to create a new development project within the team and assign it to team members.
2. **Assign Task**: Method to assign specific development tasks (e.g., feature implementation, bug fixing) to team members.
3. **Update Project Status**: Method to update the status of a development project (e.g., in progress, completed, testing phase).
4. **Collaborate**: Method to facilitate collaboration among team members during the development process.
5. **Code Review and Merge**: Method to review code changes, provide feedback, and merge changes into the main codebase.
6. **Communicate with Clients/Stakeholders**: Method to communicate with clients or stakeholders regarding project requirements, updates, and feedback.
7. **Track Project Progress**: Method to track the progress of development projects, including task completion and adherence to deadlines.
8. **Testing and Quality Assurance**: Method to conduct testing and quality assurance activities to ensure the reliability and performance of software.
9. **Deploy and Release**: Method to deploy software releases to production environments and manage release cycles.
10. **Generate Reports**: Method to generate reports on development project status, resource utilization, and client satisfaction.

1.6 auxiliaryEmployee

Attributes:

1. **Employee ID**: A unique identifier for the auxiliary employee within the organization.
2. **Name**: The full name of the auxiliary employee.
3. **Role**: The role or position of the auxiliary employee within the organization.
4. **Department**: The department to which the auxiliary employee is assigned or supports.
5. **Email Address**: The email address of the auxiliary employee for communication purposes.
6. **Phone Number**: The phone number of the auxiliary employee for contact purposes.
7. **Address**: The physical address of the auxiliary employee.
8. **Supervisor ID**: The Employee ID of the supervisor or manager overseeing the auxiliary employee's work.
9. **Tasks Assigned**: A list of tasks assigned to the auxiliary employee.
10. **Task Status**: The status of tasks assigned to the auxiliary employee (e.g., pending, in progress, completed).
11. **Availability**: Information about the auxiliary employee's availability for task assignments (e.g., working hours, schedule).

Methods:

1. **Constructor**: A constructor method to initialize the attributes of the AuxiliaryEmployee object.
2. **Getters and Setters**: Getter methods to retrieve the values of attributes and setter methods to set or update the values.
3. **Assign Task**: A method to assign tasks to the auxiliary employee based on their skills and availability.
4. **Update Task Status**: A method to update the status of tasks assigned to the auxiliary employee.
5. **Communicate with Supervisor**: A method to communicate with the supervisor regarding task assignments, updates, and any issues.
6. **Request Assistance**: A method for the auxiliary employee to request assistance or guidance from their supervisor or other team members.
7. **Report Progress**: A method to report progress on assigned tasks to the supervisor or manager.
8. **Provide Support**: A method to provide support to other team members or departments as needed.
9. **Attend Meetings**: A method to attend meetings or discussions related to tasks or projects they are involved in.
10. **Update Availability**: A method to update the availability status of the auxiliary employee based on their schedule or workload.

2.Role-Based Access Control (RBAC)

Step 1: Define Roles

In this step, we'll identify the different roles within the system and determine the specific permissions associated with each role.

Roles:

1. Admin
2. Manager
3. Employee
4. Design Team Lead
5. Development Team Lead

Permissions:

Now, let's define the permissions associated with each role:

1. Admin:
   - createTask
   - assignTask
   - updateTaskStatus
   - viewEmployeeDetails
   - manageRoles (ability to manage roles and permissions)
2. Manager:
   - assignTask
   - updateTaskStatus
   - viewEmployeeDetails
3. Employee:
   - viewTaskDetails (ability to view task details assigned to them)
4. Design Team Lead:
   - assignTask
   - updateTaskStatus
   - viewEmployeeDetails
   - viewDesignProjects (ability to view design projects)
5. Development Team Lead:
   - assignTask
   - updateTaskStatus
   - viewEmployeeDetails
   - viewDevelopmentProjects (ability to view development projects)

Step 2: Assign Permissions to Users

In this step, we'll create users and assign roles to them. Each user will inherit the permissions associated with their assigned role.

Users:

Let's create some example users and assign roles to them:

1. AdminUser: Assigned Admin role
2. ManagerUser: Assigned Manager role
3. EmployeeUser: Assigned Employee role
4. DesignLeadUser: Assigned Design Team Lead role
5. DevLeadUser: Assigned Development Team Lead role

Step 3: Implement Access Control

Implementation:

1. **Define a Function to Check Permissions**:
    - Input: User object, permission string
    - Output: Boolean (True if user has permission, False otherwise)
    - Iterate through the roles assigned to the user.
    - Check if any of the user's roles have the given permission.
    - Return True if permission is found, False otherwise.
2. **Test Access Control**:
    - Create user objects with different roles.
    - Call the function to check permissions for each user and permission combination.
    - Print the result indicating whether the user has permission or not.

Step 4: Role Assignment

Implementation:

1. **Define a User Class**:
    - The User class represents users in the system and stores their name and assigned role.
2. **Assign Roles to Users**:

    - Create instances of the User class for each user and assign roles to them.
3. **Test Role Assignment**:
    - Print the roles assigned to each user to verify that role assignment is successful.

Step 5: Dynamic Role Assignment

Implementation:

1. **Define Functions for Role Assignment and Revocation**:
    - Implement functions or methods to dynamically assign and revoke roles for users based on changes in their roles and responsibilities within the organization.
2. **Update User Roles**:
    - Call the role assignment and revocation functions as needed when users' roles change.
3. **Test Dynamic Role Assignment**:
    - Verify that users' roles are updated correctly by printing their roles before and after role assignment or revocation.

Step 6: Testing and Validation

Implementation:

1. **Define Test Scenarios**:
    - Identify various scenarios to test the role-based access control (RBAC) implementation, including:
        - Users with different roles attempting to perform different actions.
        - Users' permissions being dynamically updated.
        - Users' access being restricted based on their roles and permissions.
2. **Implement Test Cases**:
    - Write test cases to cover each identified scenario.
    - Each test case should specify the expected outcome based on the RBAC rules.
3. **Execute Test Cases**:
    - Execute the test cases to validate the RBAC implementation.
    - Ensure that users can only perform actions and access resources that they have permission to do so.
4. **Analyze Test Results**:
    - Analyze the test results to identify any discrepancies or issues with the RBAC implementation.
    - Address any failures or unexpected outcomes by refining the implementation as necessary.

3. Build Task Management System

Step 3.1: Define Task Class

Implementation:

1. **Define the Task Class**:
    - Create a class named `Task` to represent tasks within the system.
    - Define the attributes such as task ID, description, status, deadline, etc.

```
CLASS Task:
  ATTRIBUTES:
    - taskId: string
    - description: string
    - status: string
    - deadline: date
    - assignee: string
    - progress: int

  CONSTRUCTOR(taskId, description, deadline):
    SET self.taskId = taskId
    SET self.description = description
    SET self.status = "Pending"
    SET self.deadline = deadline
    SET self.assignee = None
    SET self.progress = 0
```

```
METHOD updateStatus(newStatus):
    SET self.status = newStatus

METHOD setDeadline(deadline):
    SET self.deadline = deadline

METHOD assignTo(assignee):
    SET self.assignee = assignee

METHOD updateProgress(progress):
    SET self.progress = progress
```

Step 3.2: Implement Task CRUD Operations

Implementation:

1. **Create Task**:
   - Implement a function to create a new task.
2. **Read Task**:
   - Implement a function to read or retrieve a task by its ID.
3. **Update Task**:
   - Implement a function to update the status of a task.
4. **Delete Task**:
   - Implement a function to delete a task by its ID.

```
# Task CRUD operations


# Create a new task

FUNCTION createTask(taskId, description, deadline):

    CREATE newTask AS Task(taskId, description, deadline)

    ADD newTask TO TASK_LIST

    RETURN newTask


# Read or retrieve a task by its ID

FUNCTION readTask(taskId):

    FOR each task IN TASK_LIST:

        IF task.taskId == taskId:

            RETURN task

    RETURN None
```

```
# Update the status of a task
FUNCTION updateTaskStatus(taskId, newStatus):
    task = readTask(taskId)
    IF task IS NOT None:
        task.updateStatus(newStatus)


# Delete a task by its ID
FUNCTION deleteTask(taskId):
    task = readTask(taskId)
    IF task IS NOT None:
        REMOVE task FROM TASK_LIST
```

Step 3.3: Task Assignment and Tracking

Implementation:

1. **Assign Task**:
   - Implement a function to assign a task to a user or team.
2. **Track Task Progress**:
   - Implement a function to update and track the progress of a task.

```
# Task assignment and tracking


# Assign a task to a user or team

FUNCTION assignTask(taskId, assignee):

    task = readTask(taskId)

    IF task IS NOT None:

        task.assignTo(assignee)


# Update and track the progress of a task

FUNCTION trackTaskProgress(taskId, progress):

    task = readTask(taskId)

    IF task IS NOT None:

        task.updateProgress(progress)
```

Step 3.4: Deadline Management

Implementation:

1. **Set Task Deadline**:
    - Implement a function to set or update the deadline for a task.
2. **Notify Users of Approaching Deadlines**:
    - Implement functionality to notify users when deadlines are approaching or tasks are overdue.

```
# Deadline management


# Set or update the deadline for a task

FUNCTION setTaskDeadline(taskId, deadline):

    task = readTask(taskId)

    IF task IS NOT None:

        task.setDeadline(deadline)


# Notify users of approaching deadlines

FUNCTION notifyApproachingDeadlines():

    CURRENT_DATE = getCurrentDate()

    FOR each task IN TASK_LIST:

        IF task.deadline - CURRENT_DATE <= NOTIFICATION_THRESHOLD:

            SEND_NOTIFICATION(task.assignee, "Deadline approaching for task " +
task.taskId)


# Notify users of overdue tasks

FUNCTION notifyOverdueTasks():

    CURRENT_DATE = getCurrentDate()

    FOR each task IN TASK_LIST:
        IF CURRENT_DATE > task.deadline:
            SEND_NOTIFICATION(task.assignee, "Task " + task.taskId + " is overdue")
```

Step 3.5: Integrate Task Management with RBAC

Implementation:

1. **Check Permissions for Task Operations**:
   - Implement functions to check if a user has the necessary permissions to perform task operations.
2. **Enforce Permissions in Task Management**:
   - Integrate permission checks into task CRUD operations, task assignment, tracking, and deadline management.

```
# Integration with RBAC

# Check if a user has the necessary permissions
FUNCTION checkPermission(user, permission):
    RETURN user.hasPermission(permission)

# Integrate permission checks into task management operations
FUNCTION createTask(user, taskId, description, deadline):
    IF checkPermission(user, "CREATE_TASK"):
        CREATE newTask AS Task(taskId, description, deadline)
        ADD newTask TO TASK_LIST
        RETURN newTask
    ELSE:
        RETURN "Permission Denied"
```

```
FUNCTION readTask(user, taskId):
    IF checkPermission(user, "READ_TASK"):
        FOR each task IN TASK_LIST:
            IF task.taskId == taskId:
                RETURN task
        RETURN None
    ELSE:
        RETURN "Permission Denied"

FUNCTION updateTaskStatus(user, taskId, newStatus):
    IF checkPermission(user, "UPDATE_TASK"):
        task = readTask(user, taskId)
        IF task IS NOT None:
            task.updateStatus(newStatus)
    ELSE:
        RETURN "Permission Denied"

FUNCTION deleteTask(user, taskId):
    IF checkPermission(user, "DELETE_TASK"):
        task = readTask(user, taskId)
        IF task IS NOT None:
            REMOVE task FROM TASK_LIST
    ELSE:
        RETURN "Permission Denied"
```

```
FUNCTION assignTask(user, taskId, assignee):
   IF checkPermission(user, "ASSIGN_TASK"):
      task = readTask(user, taskId)
      IF task IS NOT None:
         task.assignTo(assignee)
   ELSE:
      RETURN "Permission Denied"

FUNCTION trackTaskProgress(user, taskId, progress):
   IF checkPermission(user, "TRACK_TASK"):
      task = readTask(user, taskId)
      IF task IS NOT None:
         task.updateProgress(progress)
   ELSE:
      RETURN "Permission Denied"

FUNCTION setTaskDeadline(user, taskId, deadline):
   IF checkPermission(user, "SET_DEADLINE"):
      task = readTask(user, taskId)
      IF task IS NOT None:
         task.setDeadline(deadline)
   ELSE:
      RETURN "Permission Denied"
```

4. Establish Resource Management Processes

Step 4.1: Define Resources

Implementation:

1. **Define the Resource Class**:
    - Create a class to represent resources with attributes like resourceId, resourceName, resourceType, and availabilityStatus.

```
# Resource class definition

CLASS Resource:
    ATTRIBUTE resourceId
    ATTRIBUTE resourceName
    ATTRIBUTE resourceType
    ATTRIBUTE availabilityStatus

    FUNCTION initialize(resourceId, resourceName, resourceType, availabilityStatus):
        SET self.resourceId = resourceId
        SET self.resourceName = resourceName
        SET self.resourceType = resourceType
        SET self.availabilityStatus = availabilityStatus

    FUNCTION setAvailabilityStatus(status):
        SET self.availabilityStatus = status

    FUNCTION getAvailabilityStatus():
        RETURN self.availabilityStatus
```

Step 4.2: Resource Allocation

Implementation:

1. **Resource Allocation and Deallocation**:
   - Implement functions to allocate and deallocate resources to tasks.

```
# Resource allocation and deallocation

# Allocate a resource to a task
FUNCTION allocateResourceToTask(resourceId, taskId):
    resource = findResource(resourceId)
    task = readTask(taskId)
    IF resource IS NOT None AND task IS NOT None AND resource.getAvailabilityStatus()
== True:
        task.addResource(resource)
        resource.setAvailabilityStatus(False)
        RETURN "Resource allocated"
    ELSE:
        RETURN "Allocation failed"

# Deallocate a resource from a task
FUNCTION deallocateResourceFromTask(resourceId, taskId):
    resource = findResource(resourceId)
    task = readTask(taskId)
    IF resource IS NOT None AND task IS NOT None:
        task.removeResource(resource)
        resource.setAvailabilityStatus(True)
        RETURN "Resource deallocated"
    ELSE:
        RETURN "Deallocation failed"
```

Step 4.3: Track Resource Usage

Implementation:

1. **Track Resource Usage**:
    - Implement functions to track the usage of resources

```
# Track resource usage

# Get all resources allocated to a task
FUNCTION getResourcesAllocatedToTask(taskId):
    task = readTask(taskId)
    IF task IS NOT None:
        RETURN task.getAllocatedResources()
    ELSE:
        RETURN None

# Get all tasks using a specific resource
FUNCTION getTasksUsingResource(resourceId):
    resource = findResource(resourceId)
    allocatedTasks = []
    IF resource IS NOT None:
        FOR each task IN TASK_LIST:
            IF resource IN task.getAllocatedResources():
```

```
        ADD task TO allocatedTasks
    RETURN allocatedTasks
ELSE:
    RETURN None
```

7. Implement Change Management Procedures

Key Components:

1. **Change Request**:
    - Define how changes can be requested.
    - Track details of the change request.
2. **Change Approval**:
    - Set up an approval workflow.
    - Define roles and responsibilities for approving changes.
3. **Change Implementation**:
    - Develop procedures for implementing approved changes.
    - Ensure proper communication and documentation.
4. **Change Review**:
    - Review and assess the impact of changes.
    - Adjust processes based on feedback and lessons learned.

1. Change Request

```
# ChangeRequest class definition


CLASS ChangeRequest:

    ATTRIBUTE requestId

    ATTRIBUTE requesterId

    ATTRIBUTE requestDate

    ATTRIBUTE description

    ATTRIBUTE status  # e.g., "Pending", "Approved", "Rejected"

    ATTRIBUTE priority

    ATTRIBUTE changeType


    FUNCTION initialize(requestId, requesterId, requestDate, description, status, priority,
changeType):

        SET self.requestId = requestId

        SET self.requesterId = requesterId
    SET self.requestDate = requestDate

    SET self.description = description

    SET self.status = status

    SET self.priority = priority

    SET self.changeType = changeType


FUNCTION setStatus(status):

    SET self.status = status


FUNCTION getStatus():

    RETURN self.status
```

2. Change Approval

```
# Change approval workflow


# Submit a change request
FUNCTION submitChangeRequest(changeRequest):
    ADD changeRequest TO ChangeRequestList
    RETURN "Change request submitted"


# Approve or reject a change request
FUNCTION approveChangeRequest(requestId, approverId, decision):
    changeRequest = findChangeRequest(requestId)
    IF changeRequest IS NOT None AND decision IN ["Approved", "Rejected"]:
        changeRequest.setStatus(decision)
        RETURN "Change request " + decision
    ELSE:
        RETURN "Invalid request or decision"
```

3. Change Implementation

```
# Change implementation procedures


# Implement a change
FUNCTION implementChange(requestId):
    changeRequest = findChangeRequest(requestId)
    IF changeRequest IS NOT None AND changeRequest.getStatus() == "Approved":
        PERFORM necessary actions TO implement change
        changeRequest.setStatus("Implemented")
        RETURN "Change implemented"
    ELSE:
        RETURN "Change cannot be implemented"
```

4. Change Review

```
# Change review procedures


# Review a change
FUNCTION reviewChange(requestId):
    changeRequest = findChangeRequest(requestId)
    IF changeRequest IS NOT None AND changeRequest.getStatus() == "Implemented":
        ASSESS impact of change
        RECORD feedback
        RETURN "Change reviewed"
    ELSE:
        RETURN "Change cannot be reviewed"
```