# Bonus Buddy: Gamifying Workplace Productivity PM3

Kenneth Lao
Department of Computer Science, Virginia Tech
Blacksburg, VA, USA
kennethlao120@vt.edu

Zachary Beritt
Department of Computer Science, Virginia Tech
Blacksburg, VA, USA
zberritt@vt.edu

Albert Essiaw
Department of Electrical Engineering, Virginia Tech
Blacksburg, VA, USA
alberte@vt.edu

Qitao Yang
Department of Computer Science, Virginia Tech
Blacksburg, VA, USA
yqitao@vt.edu

## 1 PROCESS DELIVERABLE II

*Agile Software Development Process.*

### 1.1 Success:

- **Effective Team Collaboration:** Our team worked effectively to define the core requirements for Bonus Buddy. Through structured discussions and regular communication, we aligned on key functionalities, ensuring everyone contributed to shaping the project's direction. This collaboration was instrumental in establishing a shared understanding of goals and expectations.
- **Core Functionalities:** The implementation of task tracking and the reward system prototypes successfully demonstrated the foundational mechanics of Bonus Buddy. These components are functional and provide a solid basis for further feature enhancements, such as leaderboards and privacy options.

### 1.2 Challenges:

- **Balancing Rewards:** Balancing Rewards: The current reward distribution system doesn't have a comprehensive approach to balancing frequency and value. Frequent rewards might risk lowering their importance, and rare rewards might not maintain motivation among developers. This is an important area that might require more attention from the team.
- **User Engagement Metrics:** It has been difficult to find useful and practical measures of user engagement. We have set up basic measures such as task completion rates and reward points, but more research and proof are needed to discover better signs of lasting engagement and productivity. It is important to ensure that Bonus Buddy continues motivating users over time.

### 1.3 Solutions:

- **Better Reward Algorithm:** We decided to introduce a more dynamic reward mechanism, that will take into account task complexity, completion time, and engagement metrics. So that the reward mechanisms are impactful and do not lose their motivational value.
- **Better User Engagement Metrics:** We design and test more advanced metrics to measure user engagement beyond the completion of tasks, such as tracking streaks, peer comparisons, and productivity trends, in order to maintain long-term user motivation.

### 1.4 Prioritized Tasks for Next Milestone:

- **Leader board Development:** Next, we wanted to implement a public leaderboard feature to promote healthy competition among Software Developers. This includes customizable visibility settings to allow modification of their username and decide if they want their achievement to be visible to everyone else in the company.
- **Seasonal Reward System:** We also wanted to design a seasonal reward system that keeps users engaged through time-bound challenges and incentives. This system will introduce special goals and tasks related to seasonal themes like many video games, company milestones, or cultural events. For instance, employees may be encouraged to complete certain high-impact tasks during the end-of-year season to unlock exclusive rewards. The new seasonal reward system will also incorporate team-based challenges to encourage collaboration among colleagues. We aim to refresh these challenges and rewards periodically, thus managing to keep long-term interest in the feature alive and a sense of achievement that is associated with specific time frames.

## 2 HIGH-LEVEL DESIGN

### 2.1 Design

For the Bonus Buddy, we decided to use an **Event-Based Architecture**. This will allow for real-time communicate between our components. We will use a system with event producers and event consumers. The producers will generate events when tasks are complete or milestones are achieved. These will be transferred to a consumer which will consume these events and process them. These consumers can calculate points, update leaderboards, and distribute rewards as so.

### 2.2 Justification

An event-based design facilitates real-time communication between services, ensuring that Bonus Buddy can deliver immediate recognition for task completion. This architecture processes and acts on events instantly, enabling timely feedback for users. Its decoupled nature allows individual components, such as the Reward Engine and Leaderboard Manager, to function independently, simplifying maintenance and making it easier to scale as the user base expands.

Moreover, the architecture's high throughput capacity ensures the system can handle a large volume of employees and tasks without compromising performance. Finally, the design seamlessly supports integration with external services, allowing these systems to interact effortlessly by signaling events.

## 3 LOW-LEVEL DESIGN

### 3.1 Design Pattern Family

The **Observer Pattern** from the Behavioral Design Patterns family is particularly helpful for implementing the event-driven architecture of this project. This pattern ensures that different components, such as the *Reward Engine*, *Leaderboard Manager*, and *Notification Service*, can dynamically subscribe to and react to events generated by the *Task Manager*. This decouples the components, making the system easier to maintain and extend.

### 3.2 Justification

The **Observer Pattern** is ideal for this project for the following reasons:

- **Event-Driven Architecture:** It aligns perfectly with the project's requirement to handle task completions and milestone achievements as discrete events.
- **Loose Coupling:** Observers (e.g., *Reward Engine*, *Leaderboard Manager*) are decoupled from the event producer (*Task Manager*), allowing independent development and maintenance of each component.
- **Scalability:** Adding new observers (e.g., a future *Analytics Service*) is straightforward, requiring minimal changes to the existing system.

### 3.3 Pseudocode

Below is the pseudocode implementation of the Observer Pattern in the context of the *Bonus Buddy* system:

```
# Define EventManager (Subject)
class EventManager:
    observers = []  # List to store observers

    def add_observer(observer):
        observers.append(observer)

    def remove_observer(observer):
        observers.remove(observer)

    def notify(event):
        for observer in observers:
            observer.update(event)

# Define Observer Interface
class Observer:
    def update(event):
        # To be implemented by concrete observers

# Define RewardEngine (Concrete Observer)
class RewardEngine(Observer):
    def update(event):
        if event.type == "TaskCompleted":
            reward_points = calculate_points(event)
        update_user_rewards(event.user, reward_points)

    def calculate_points(event):
        # Logic to calculate reward points

# Define LeaderboardManager (Concrete Observer)
class LeaderboardManager(Observer):
    def update(event):
      if event.type == "TaskCompleted" or event.type == "MilestoneAch
            update_leaderboard(event.user)

    def update_leaderboard(user):
        # Logic to update leaderboard rankings

# Define TaskManager (Event Producer)
class TaskManager:
    event_manager = EventManager()

    def complete_task(user, task):
      event = create_event("TaskCompleted", user, task)
        event_manager.notify(event)

    def create_event(type, user, task):
        return {
            "type": type,
            "user": user,
            "task": task
        }

# Main Execution
event_manager = EventManager()
reward_engine = RewardEngine()
leaderboard_manager = LeaderboardManager()

# Register observers
event_manager.add_observer(reward_engine)
event_manager.add_observer(leaderboard_manager)

# Simulate task completion
task_manager = TaskManager()
task_manager.complete_task("User1", "TaskA")
```

### 3.4 Informal Class Diagram

The informal class diagram below illustrates the relationships between the components in the Observer Pattern:

```
+-------------------+
|   EventManager    |<-----------------+
+-------------------+                  |
| + add_observer() |                   |
| + remove_observer()                  |
| + notify()       |------------------>|
+-------------------+                  |
                                       |
+-------------------+                  |
|    Observer      |<----------------+
+------------------+
| + update(event)  |
+------------------+

+------------------------+
|    RewardEngine        | (inherits Observer)
+------------------------+
| + update(event)        |
| + calculate_points()   |
| + update_user_rewards()|
+------------------------+

+------------------------+
| LeaderboardManager     | (inherits Observer)
+------------------------+
| + update(event)        |
| + update_leaderboard() |
+------------------------+

+------------------------+
|      TaskManager       |
+------------------------+
| + complete_task()      |
| + create_event()       |
+------------------------+
```

## 3.5 Conclusion

Using the **Observer Pattern** allows the system to handle task completion and milestone events effectively. The design promotes scalability, maintainability, and clear separation of responsibilities among components, making it a robust choice for the low-level implementation of the *Bonus Buddy* system.

## 4 DESIGN SKETCH

## 4.1 Wireframe Mockup

Below is a wireframe mockup of the primary user interface for the *Bonus Buddy* system. The design highlights the key components and functionality available to the user:

- **Task Dashboard**: A clean interface displaying a list of tasks, categorized by priority (e.g., High, Medium, Low). Each task shows relevant details like due dates and current status.
- **Leaderboard**: A prominently displayed leaderboard that ranks employees based on their cumulative reward points, with options to filter visibility by team, department, or organization-wide.
- **Reward Hub**: A section that showcases rewards available to redeem, personalized recommendations based on user engagement, and information on seasonal rewards or special challenges.
- **Activity Summary**: A graphical summary of the user's performance, including streaks, completed tasks, and unlocked achievements.
- **Notifications**: A collapsible side panel for real-time updates on task completions, leaderboard movements, and upcoming challenges.

**Wireframe Overview**:

```
+---------------------------------------+
| Task Dashboard     | Notifications    |
|--------------------|------------------|
| - Task A (High)    | - Task Update    |
| - Task B (Medium)  | - New Reward     |
| - Task C (Low)     | - Leaderboard    |
+---------------------------------------+
|         Leaderboard (Top 5)           |
| 1. Alice: 120 pts                     |
| 2. Bob: 110 pts                       |
| 3. Carol: 100 pts                     |
+---------------------------------------+
|          Reward Hub                   |
| - $10 Gift Card (50 pts)              |
| - Extra Day Off (200 pts)             |
+---------------------------------------+
|      Activity Summary (Graph)         |
| - Streaks: 5 Days                     |
| - Task Completed: 10                  |
| - Seasonal Challenge: 2/5             |
+---------------------------------------+
```

## 4.2 Storyboard Illustration

The storyboard below illustrates a common use case:

(1) **Task Assignment**: A user logs in and sees a new task ("Prepare Quarterly Report") in their Task Dashboard.
(2) **Task Completion**: Upon completing the task, the user marks it as done, triggering an event.
(3) **Reward Notification**: The system awards the user 30 points, displayed in a pop-up notification.
(4) **Leaderboard Update**: The user moves up the leaderboard immediately in the interface.
(5) **Reward Redemption**: The user visits the Reward Hub to redeem a $10 gift card using their accumulated points.

## 4.3 Rationale

The design focuses on **user engagement** and **intuitive navigation**. The prominent placement of the Task Dashboard and Leaderboard ensures users can quickly access their tasks and track their progress. The Reward Hub encourages consistent participation by highlighting achievable rewards. Incorporating graphical summaries in the Activity Summary promotes a sense of accomplishment and visual clarity. These decisions align with the core principles of usability and feedback loops discussed in class, ensuring users stay motivated and invested in the platform.