Weekly Homework 6

Yevgeniy Terentyev, Miriam Gaetano
Softwaretechnik

June 1, 2025

1

We engaged with the tasks.

2

We engaged with the tasks.

3

3.1 Logical view

The logical view identifies essential classes that represent the core functionalities of the Pomodoro app. Key abstractions include:

- **Timer**: A class responsible for managing the countdown for work sessions and breaks. It has attributes for customizable durations (e.g., work session and short break) and methods to start, pause, and reset the timer.
- **WorkTimer, ShortBreak, LongBreak**: A series of subclasses or related classes that extends the Timer functionality to handle working timer, short break and longer breaks after a specified number of work sessions, allowing users to set durations for each.

The logical view includes a **Task** class that encapsulates the properties and behaviors associated with tasks in the Pomodoro app. This class should have:

- Attributes for task details (e.g., title, description, status).
- Methods for creating, editing, and deleting tasks, facilitating better organization and management of tasks associated with each Pomodoro session.

Relationships between the Timer and Task classes and their interactions:

- **Association**: The Timer class is associated with the Task class, indicating that each Pomodoro session can be linked to a specific task.
- **Composition**: The Timer may contain instances of the Task class, representing the tasks that are active during a work session. This relationship ensures that the timer functionality is directly tied to the tasks being managed.

3.2 Process View

The process architecture takes into account some non-functional requirements, such as performance and availability.

- **Concurrency and Distribution**: The process view addresses the concurrent execution of timers and task management functionalities. For example the main GUI process might use another processes to start a timer/break cycle and one for task management that would consent to application to run concurrently without displayed output getting stuck or feeling unresponsive. This processes can be lightweight threads in this case as there's no need for a completely independent process facilitating the communication.
- **Major/Minor tasks** In addition to the main major task that can be seen as series on minor tasks previously mentioned we could have another major task in the background periodically synchronizing the state of tasks tracking and new tasks created in the cloud storage. The communication can be achieved through well-defined inter-task communication mechanisms, such as message passing or event notifications.
- **Fault Tolerance and System Integrity** For if a timer process fails the app should be able to recover gracefully (fault tolerance) without losing the state of ongoing tasks or cycles and possibly even timers (integrity).

3.3 Development view

Focuses on the actual software module organization on the software development environment. The Pomodoro timer app can be organized into several **subsystems** or modules, each focusing on a specific functionality. This modular approach allows for easier development, testing, and maintenance. The proposed subsystems include:

- **Timer Subsystem**: Manages all timer functionalities, including work sessions, short breaks, and long breaks.
- **Task Management Subsystem**: Handles the creation, editing, and deletion of tasks associated with Pomodoro sessions.
- **User Interface Subsystem**: Manages the presentation layer, including user interactions and visual feedback.
- **Notification Subsystem**: Responsible for sending alerts and notifications to users when timers start, end, or require user action.

The development architecture can follow a **layered style**, with each layer having a well-defined responsibility. Some layers in our case might have blurred boundaries as the usually layers include multiple subsystems in complex systems which is not our case being a simple app. We can define the following layers:

- **Presentation Layer**: Contains the User Interface Subsystem, responsible for displaying the app and handling user input.
- **Application Layer**: Includes the Timer and Task Management Subsystems, which implement the core functionalities of the app.
- **Data Layer**: Manages data storage and retrieval, potentially including local storage for tasks and session history.
- **Notification Layer**: Handles all notification-related functionalities, ensuring timely alerts for users.

The following rules govern the development architecture of the Pomodoro timer app:

- **Partitioning**: Each subsystem should be developed independently, allowing for parallel development by different teams or developers.
- **Grouping**: Related functionalities should be grouped within the same subsystem to enhance cohesion and reduce complexity.
- **Visibility**: Subsystems should expose only necessary interfaces to other layers, minimizing dependencies and promoting encapsulation.

3.4 Physical View Considerations

Being a small system there's no need for a particular physical blueprint. The Pomodoro app can be deployed across various platforms:

- **Desktop**: Standalone applications for Windows, macOS, and Linux.
- **Mobile**: Applications for iOS and Android devices.
- **Web**: Accessible via web browsers for cross-device compatibility.

The app requires minimal hardware resources:

- **Client-Side**: Basic specifications such as 1 GB RAM and a dual-core processor.
- **Server-Side**: A simple server setup (e.g., 4 GB RAM) Can be reconsidered for scalability reasons. (AWS, Azure etc.)

The app can function in different network settings:

- **Local Network**: Operates offline with local data storage for high availability.
- **Internet Connectivity**: Synchronizes data with a cloud server when online, allowing access across devices.

3.5 Scenarios

As per peper: This view is redundant with the other ones (hence the "+1"), but it serves two main purposes:

- as a driver to discover the architectural elements during the architecture design as we will describe later
- as a validation and illustration role after this architecture design is complete, both on paper and as the starting point for the tests of an architectural prototype.

Notation for the Scenarios The notation is very similar to the Logical view for the components, but uses the connectors of the Process view for interactions between objects Note that object instances are denoted with solid lines. As for the logical blueprint, we capture and manage object scenario diagrams using Rational Rose.