Software Process Models

Names and IDs:

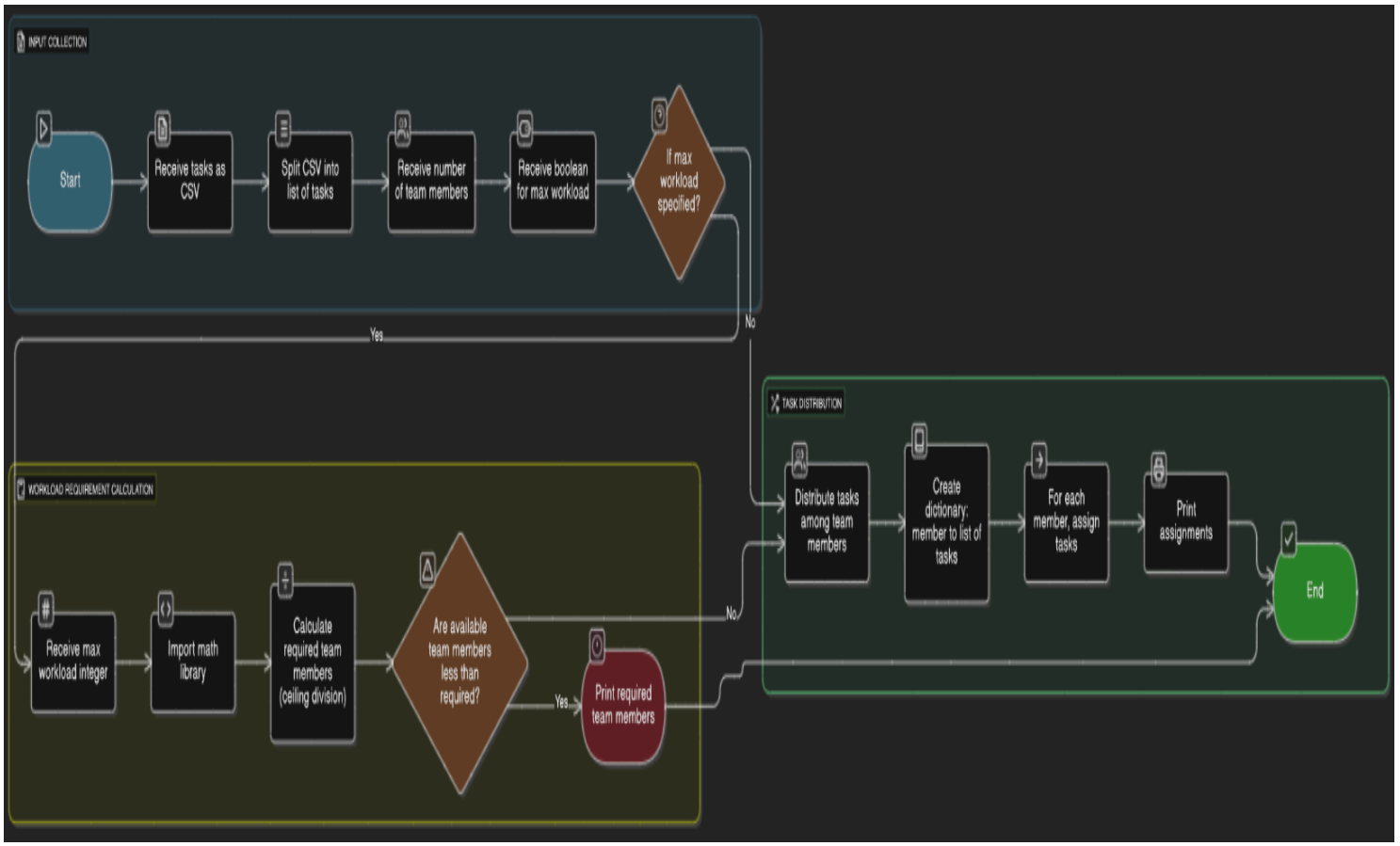
- Abdulaziz Binadwan (**202434500**)
- Taha Sindi (**202353110**)
- Saleh Al Rashdi (**202366550**)

The target in this project is to use the Waterfall strategy in developing a program that equally assigns work to team members (a project managing software).

Application Requirements:

- Being able to input the list of tasks
- Being able to input the overall number of team members
- Being able to specify the maximum number of tasks per team member
- After the maximum load per capita is specified, the app should present the minimum number of team members required for the tasks to be done

Design:



Implementation:

```
import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Get tasks from user
        System.out.print("Enter tasks (comma-separated): ");
        String tasksRaw = scanner.nextLine();
        List<String> tasks = new ArrayList<>();
        for (String t : tasksRaw.split(",")) {
            t = t.trim();
            if (!t.isEmpty()) {
                tasks.add(t);
            }
        }

        if (tasks.isEmpty()) {
            System.out.println("No tasks.");
            return;
        }

        // Get number of team members
        System.out.print("Number of team members: ");
        int members;
        try {
            members = Integer.parseInt(scanner.nextLine().trim());
        } catch (NumberFormatException e) {
            System.out.println("Invalid team size.");
            return;
        }

        if (members <= 0) {
            System.out.println("Invalid team size.");
            return;
        }

        // Optional max tasks per member
        System.out.print("Max tasks per member (optional, press Enter to skip): ");
        String maxStr = scanner.nextLine().trim();
        Integer maxPer = null;

        if (!maxStr.isEmpty()) {
            try {
                int val = Integer.parseInt(maxStr);
                if (val > 0) {
                    maxPer = val;
                }
            } catch (NumberFormatException ignored) {}
        }

        if (maxPer != null) {
            int totalTasks = tasks.size();
            int requiredMembers = (int) Math.ceil((double) totalTasks / maxPer);
            if (members < requiredMembers) {
                System.out.println("Increasing team size from " + members + " to " + requiredMembers + " to respect max limit.");
                members = requiredMembers;
            }
        }

        // Distribute tasks
        List<List<String>> distribution = new ArrayList<>();
        for (int i = 0; i < members; i++) {
            distribution.add(new ArrayList<>());
        }

        for (int i = 0; i < tasks.size(); i++) {
            distribution.get(i % members).add(tasks.get(i));
        }

        // Output
        System.out.println("\nTotal tasks: " + tasks.size());
        System.out.println("Team members: " + members);
        System.out.println("=".repeat(30));

        for (int i = 0; i < distribution.size(); i++) {
            List<String> bucket = distribution.get(i);
            System.out.println("Member " + (i + 1) + " (" + bucket.size() + " tasks): " + bucket);
        }
    }
}
```

Testing:

```
Enter tasks (comma-separated): task1,task2,task3,task4,task5,task6,task7,task8
Number of team members: 2
Max tasks per member (optional, press Enter to skip): 2
Increasing team size from 2 to 4 to respect max limit.

Total tasks: 8
Team members: 4
==============================
Member 1 (2 tasks): [task1, task5]
Member 2 (2 tasks): [task2, task6]
Member 3 (2 tasks): [task3, task7]
Member 4 (2 tasks): [task4, task8]

=== Code Execution Successful ===
```

```
Enter tasks (comma-separated): task1,task2,task3,task4,task5,task6,task7,task8
Number of team members: 3
Max tasks per member (optional, press Enter to skip): 3

Total tasks: 8
Team members: 3
==============================
Member 1 (3 tasks): [task1, task4, task7]
Member 2 (3 tasks): [task2, task5, task8]
Member 3 (2 tasks): [task3, task6]

=== Code Execution Successful ===
```

```
Enter tasks (comma-separated): task1,task2,task3,task4,task5,task6,task7,task8
Number of team members: 11
Max tasks per member (optional, press Enter to skip):

Total tasks: 8
Team members: 11
==============================
Member 1 (1 tasks): [task1]
Member 2 (1 tasks): [task2]
Member 3 (1 tasks): [task3]
Member 4 (1 tasks): [task4]
Member 5 (1 tasks): [task5]
Member 6 (1 tasks): [task6]
Member 7 (1 tasks): [task7]
Member 8 (1 tasks): [task8]
Member 9 (0 tasks): []
Member 10 (0 tasks): []
Member 11 (0 tasks): []

=== Code Execution Successful ===
```

```
Enter tasks (comma-separated):
No tasks.

=== Code Execution Successful ===
```

```
Enter tasks (comma-separated): task1,task2,task3,task4,task5,task6,task7,task8
Number of team members:
Invalid team size.

=== Code Execution Successful ===
```

Conclusion:

a) Throughout this project we chose to work using the Waterfall strategy mainly because the customer knew what he wanted in terms of requirements. The strategy we used gave us better and clearer managing output and comfort in having a clear and fixed timeline.

b) Knowing what to choose (waterfall or agile) in our next scenario is dependent on the next customer. If the customer has a vague or unfixed requirement list, agile is the better approach. It involves more interaction with the customer making us able to implement more requirements on the go as the code progresses. The customer, in addition, would have the ability to look at early versions of the code. Waterfall, on the other hand, require a customer with clear and unchangeable requirements that is ready to wait until the final version of the application is finished without seeing or knowing about progress.