ENSE 375 – Software Testing and Validation

Project Title

Team member name (UofR ID)

Team member name (UofR ID)

Table of Contents

- Proof read the text for typing and grammar mistakes.
- Follow the IEEE Bibliography style for the references by selecting "References/ Citations & Bibliography/ Style".

List of Figures

List of Tables

1 Introduction

- Give a brief description of the design and a summary of the relevant background information related to the topic. Give a rationale about what is needed and why.
- Give the reader an overview of what is in the next sections.
- Do not put any detailed results of your work here.

2 Design Problem

This section has the following two subsections:

2.1 Problem Definition

Proposed Project:	Smart Study Session Planner
Date Produced:	05/16/2025
Background	As university students ourselves, we find it hard to always stick to a planned study session since scheduling conflicts may arise unexpectedly due to other commitments that we may have forgotten about. Much like us, many students seem to have issues with planning and sticking to schedules, so we decided to do something about it. **The Smart Study Session Planner (SSSP),** will automatically create balanced weekly study calendars that make laying out a student's schedule a whole lot easier. The primary goal is to convert a student's deadlines, availability and study preferences into an editable planner that boosts productivity.
Key Features	• Timer • Calendar • Statistics Dashboard • In-app notification
Primary Goal	Convert a student's **deadlines, time availability, and study preferences** into an **optimized, editable weekly plan** that maximizes preparedness while minimizing stress.
Stakeholders	*Primary*: Students (end users). *Secondary*: Academic advisors, wellness centers (interested in stress metrics)

2.2 Design Requirements

This section has the following three subsections:

2.2.1 Functions -

To turn scattered deadlines and personal constraints into an easy-to-follow study plan, the Smart Study Session Planner runs a multi-stage pipeline. It first imports external calendar events, then records user-entered tasks and preferences, applies a scoring model to rank them in terms of priority, and arranges the tasks into free time blocks,and finally updates the confirmed schedule.

It will also be capable of running an integrated Pomodoro timer to help with focus on studies. The concept of taking short breaks to sustain a state of concentration will be exercised, and people can tweak the duration of their breaks up to a certain amount.

- Core scheduling loop: import → prioritize → allocate → notify.
- Lifecycle utilities: Pomodoro timer, .ics export, progress dashboard.

.

2.2.2 Objective

Our primary objective is to give students an intelligent, low-friction tool that converts all their course deadlines and personal time preferences into a balanced weekly study timetable. By automatically ranking tasks on urgency and weight, then fitting them into the student's available time blocks, the planner seeks to reduce last-minute cramming and academic stress while preserving healthy breaks and sleep windows.

The project aims to:

(1) implement a reliable scheduling algorithm that produces conflict-free plans in seconds,

(2) integrate seamless calendar import/export so no data is re-typed,

(3) provide a built-in Pomodoro timer that logs progress, and

(4) deliver these features through a clean, cross-platform Java desktop interface backed by automated tests and open-source libraries.

2.2.3 Constraints — Project‑Management, Java Tooling & System‑Testing/Validation

The constraints for this project are more centered around testing and validation quotas, but not limited to them. The development tools are also limited to Java and Java-based toolkits, which the team is a little new to.

Project‑management limits All milestones must fit the 12‑week ENSE 375 schedule and be completed by a three‑person team using only free/open‑source resources (GitHub Student tier, Figma EDU, etc.). Deliverables—code, documents, and test artifacts—must live in a shared Github repo with issues and pull‑request reviews enforced.

Java‑only tooling limits The entire system must compile and run with Java 17 + JavaFX. The packaged JAR file must execute offline on a standard student laptop without external services.

● Tooling gate: Java17 / JavaFX; no proprietary or paid dependencies.

System testing & validation limits

● Usability validation requires a System Usability Scale (SUS) ≥ 70 from ≥ 3 student participants.

● Test gate: ≥ 80 % coverage, SUS ≥ 70.

3 Solution

In this section, you will provide an account of some solutions your team brainstormed to implement and test the project. Some solutions might not have all the desired features, some might not satisfy the constraints, or both. These solutions come up in your mind while you brainstorm ways of implementing all the features while meeting the constraints. Towards, the end you select a solution that you think has all the features, testable and satisfies all the constraints. Remember that an engineering design is iterative in nature!

3.1 Solution 1

Concept: A lightweight, command-line Java application that reads deadlines and preferences from a CSV file and outputs an optimized weekly study schedule as plain text.

Testing Strengths:

● Path and Data-flow testing: Easily achievable due to the simple, linear structure of the scheduling algorithm. High coverage (≥90%) can be quickly obtained.

● Control Flow Graph (CFG): Clear and simple, making it ideal to demonstrate structural testing concepts taught in class (node, edge, and edge-pair criteria).

Testing Weaknesses and Reasons for Not Selecting:

● No GUI: Unable to implement interactive tests (state-transition tests, use-case tests).

● Lack of validation features: Boundary value, equivalence class, and decision table testing are minimal due to the simplistic text-only interaction.

● Usability concerns: Low usability as measured by the System Usability Scale (SUS); likely to fall below the acceptable SUS ≥70 benchmark.

3.2 Solution 2

Concept: A moderate-scope Swing-based desktop application guiding users through three steps: (1) import deadlines from CSV, (2) set weekly availability, and (3) review a read-only, auto-generated weekly study schedule.

Testing Strengths:

● Integration Testing: Clear separation between scheduling engine and UI allows effective bottom-up integration testing.

● Coupling Data-flow Criteria: Structured data exchange via Data Transfer Objects (DTOs) between the GUI and the scheduling engine enables robust coupling data-flow validation.

● Decision Table, Boundary, and Equivalence Testing: Supports comprehensive validation of input constraints and conflict-handling logic through targeted test cases.

● Usability (SUS Validation): GUI provides enough interactivity to potentially achieve SUS ≥70, though limited by the read-only final schedule view.

Remaining Testing Concerns (To Address in Next Steps):

● GUI limitations restrict the extent of state-transition and model-based testing (MBT) scenarios.

● Swing's event-dispatch threading model can introduce flaky tests in continuous integration environments.

● Adding interactive features such as timers later could significantly expand complexity, potentially exceeding the approximately six-week timeline.

3.3 Final Solution

The final solution selected by the team is **Solution 3: A JavaFX-based interactive study planner with integrated database support**. Unlike Solution 2, which relied on a static read-only schedule, Solution 3 offers dynamic user interaction, real-time updates, and persistent data storage to create a personalized and optimized weekly study plan.

This project allows users to:

1. **Adjust weekly availability** and instantly regenerate schedules based on updates.
2. **Store and retrieve schedules and tasks** from a database for long-term use.
3. **Visualize tasks** in an intuitive calendar interface with filtering and task-priority indicators.

Why Solution 3 is Better than Other Solutions:

- **Enhanced Usability:** The JavaFX GUI offers a highly interactive and visually clear scheduling interface, outperforming Solution 1 (command-line) and Solution 2 (limited Swing GUI) in user satisfaction. Usability tests indicate that Solution 3 can exceed the SUS ≥70 benchmark due to its drag-and-drop task adjustments and immediate feedback.

- **Comprehensive Testing:** Its modular structure (GUI layer, scheduling engine, database module) supports rigorous testing:
    - **Integration Testing** between the scheduling algorithm, UI, and database is straightforward due to well-defined APIs and data-flow boundaries.
    - **Model-based Testing** can be implemented thanks to the interactive GUI and underlying state model.
    - **Validation Testing** (boundary, equivalence class, and decision tables) ensures correct handling of various input constraints and error scenarios.
- **Scalability:** Solution 3 supports future feature enhancements, such as reminders, recurring tasks, and real-time synchronization, without overhauling the architecture.
- **Persistence & Reliability:** Database support eliminates the limitations of CSV-only storage in Solution 2, ensuring data consistency and recovery.

3.3.1 Components

The main components of Solution 3 are:

1. **Database Module (SQLite/MySQL)**
    - **Purpose:** Store tasks, deadlines, preferences, and generated schedules for persistent access.
    - **Testing Method:** Data integrity and CRUD operations tested through boundary values and SQL injection checks.
2. **Scheduling Engine**
    - **Purpose:** Compute optimal study plans using user-defined deadlines, availability, and task priorities.
    - **Testing Method:** Path and data-flow testing (≥90% coverage) and equivalence class analysis for varying input sizes.

3. **JavaFX GUI Layer**
    - **Purpose:** Provide an intuitive interface for adding tasks, adjusting availability, and visualizing schedules in real time.
    - **Testing Method:** Model-based testing (MBT) for state transitions, usability testing (SUS validation), and exploratory UI tests.
4. **Data Transfer and Validation Layer**
    - **Purpose:** Convert user input into validated data structures before scheduling.
    - **Testing Method:** Decision table and boundary value testing for input validation and error handling.

3.3.2 Environmental, Societal, Safety, and Economic Considerations

- **Environmental Impact:** The application's lightweight design minimizes CPU and memory usage, enabling energy-efficient operation.
- **Societal Impact:** By improving time management and reducing academic stress, the planner fosters better study habits and mental well-being.
- **Safety Considerations:** Built-in data validation prevents invalid schedules (e.g., overlapping tasks or negative durations), ensuring reliable outputs.
- **Economic Considerations:** The system is built using free, open-source technologies (Java, JavaFX, and SQLite), making it cost-effective and easily maintainable.

Test Cases and Results

To validate the functionality and reliability of the study planner, the following test suites were designed and executed:

1. **Functional Test Cases**
    - **Task Creation and Editing:** Verified that users can add, edit, and delete tasks with valid inputs.
    - **Schedule Generation:** Confirmed that tasks are distributed optimally based on deadlines, priorities, and weekly availability.
    - **Conflict Handling:** Tested overlapping tasks and invalid date ranges to ensure error prompts and correct resolution.
2. **Boundary Value Testing**
    - Tested scenarios with no tasks, one task, and a large number of tasks to ensure stability and correct scheduling.
3. **Integration Testing**
    - Verified the interaction between the GUI and scheduling engine, ensuring that changes to tasks are immediately reflected in the schedule.

The current solution has some limitations:

- **No CSV Import or Export:** All tasks must be entered manually, which may be time-consuming for users with a large number of tasks.
- **Statistics Dashboard Removed:** While initially planned, the statistics dashboard was removed to avoid over-complicating the project and to ensure stability of core features such as scheduling, notifications, and the timer.
- **Scalability of GUI:** When too many tasks are added, the interface may become visually cluttered.
- **Platform Dependency:** The solution is desktop-based and does not currently support mobile or web platforms.

4 Team Work

Since this is a group project, you must have a fair distribution of tasks among yourselves. To this end, you must hold meetings to discuss the distribution of tasks and to keep a track of the project progress.

4.1 Meeting 1

Time: Month Date, Year, hour: minutes am/pm to hour: minutes am/pm
Agenda: Distribution of Project Tasks

Team Member	Previous Task	Completion State	Next Task
Team member 1	N/A	N/A	Task 1
Team member 2	N/A	N/A	Task 2
Team member 3	N/A	N/A	Task 3

4.2 Meeting 2

Time: Month Date, Year, hour: minutes am/pm to hour: minutes am/pm
Agenda: Review of Individual Progress

Team Member	Previous Task	Completion State	Next Task
Team member 1	Task 1	80%	Task 1, Task 5
Team member 2	Task 2	50%	Task 2
Team member 3	Task 3	100%	Task 6

4.3 Meeting 3

Provide a similar description here.

4.4 Meeting 4

Provide a similar description here.

5 Project Management

Provide a Gantt chart showing the progress of your work here. Mention all the tasks along with their predecessors. Provide the slack time of each task and identify the critical path.

6 Conclusion and Future Work

- A summary of what you achieved. Mention all the design functions and objectives that you achieved while satisfying testing requirements?
- While keeping the limitations of your solution, provide recommendations for future design improvements.

7 References

- Use the IEEE reference style.
- Do not put any reference if it is not cited in the text.

8 Appendix

If you want to provide an additional information, use this appendix.