National College of Ireland

Object Oriented Software Engineering

Academic Year 2024/2025

Adrián Tarín Martínez

X23388978

X23388978@student.ncirl.ie

Law Firm Management System – Part 2

Content

Task 9: Agile Development Methodology: Scrum

Agile development methodologies prioritize flexibility, iterative progress, and collaboration. Among them, Scrum stands out as a widely-used framework that divides work into incremental phases called sprints. Each sprint delivers a potentially shippable product increment, fostering continuous improvement and adaptability. Here, we propose using the Scrum methodology to develop the Law Firm Management System, ensuring a structured yet flexible approach to meet evolving requirements and stakeholder expectations.

Key Features of Scrum Methodology

1. Iterative Development: Work is divided into time-boxed sprints, typically lasting 1-4 weeks, with each sprint focusing on delivering a specific set of features.

2. Roles and Responsibilities:

- **Product Owner**: Represents the stakeholders, prioritizes the product backlog, and ensures alignment with business objectives.
- **Scrum Master**: Facilitates the Scrum process, removes obstacles, and ensures the team adheres to Scrum principles.
- **Development Team**: Cross-functional members responsible for delivering the product increment.

3. Artifacts:

- **Product Backlog:** A dynamic list of all desired features and functionalities, prioritized by the Product Owner.
- **Sprint Backlog:** A subset of the product backlog chosen for implementation in the current sprint.
- **Increment:** The potentially shippable output of each sprint.

4. Ceremonies:

- **Sprint Planning:** Defines the goals and scope for the upcoming sprint.
- **Daily Scrum (Stand-ups):** Short, daily meetings to review progress, identify obstacles, and align team efforts.
- **Sprint Review:** Demonstrates the sprint increment to stakeholders, gathering feedback.
- **Sprint Retrospective:** Reflects on the sprint process, identifying improvements for future iterations.

Application of Scrum to the Law Firm Management System

Using Scrum, the development of the Law Firm Management System would involve iterative sprints where the team collaborates with stakeholders to deliver prioritized features. For example:

- **Sprint 1:** Development of core security features like user authentication and role-based access.
- **Sprint 2**: Implementation of client management functionalities such as creating, modifying, and viewing clients.
- **Sprint 3:** Addition of case management features, including case creation, deletion, and modification.
- **Subsequent Sprints:** Focus on advanced functionalities like notifications, detailed procedure management, and user-friendly UI enhancements.

Advantages of Scrum

1. **Flexibility and Adaptability:** Scrum accommodates changing requirements, a critical aspect for projects where user feedback or legal regulations might evolve.

2. **Early and Frequent Deliveries:** Delivering working increments every sprint ensures stakeholders see tangible progress and can provide timely feedback.
3. **Enhanced Collaboration:** Daily Scrum meetings and collaborative ceremonies foster clear communication among team members and stakeholders.
4. **Risk Mitigation:** Regular feedback and iterative progress reduce the risk of building features that don't meet user needs.
5. **Continuous Improvement:** The sprint retrospective encourages ongoing process refinement, enhancing team efficiency and product quality over time.

Difference Between Scrum and Waterfall Approach

The Waterfall methodology follows a linear, sequential process divided into distinct phases: Requirements, Design, Implementation, Testing, Deployment, and Maintenance. Each phase must be completed before the next begins, with minimal flexibility for changes once development starts.

Key Differences:

1. Process Flow:
- Waterfall: Linear and rigid, with progress flowing in one direction.
- Scrum: Iterative and flexible, allowing for revisiting and refining features in successive sprints.

2. Requirement Changes:
- Waterfall: Accommodates little to no changes once the requirements phase is complete.
- Scrum: Welcomes changing requirements, even late in development.

3. Delivery:
- Waterfall: Delivers the complete product at the end of the development cycle.
- Scrum: Delivers usable increments after every sprint.

4. Feedback:
- Waterfall: Feedback is typically incorporated after testing or deployment.
- Scrum: Continuous feedback is integrated into each sprint.

5. Stakeholder Involvement:
- Waterfall: Limited to the requirements phase and final delivery.
- Scrum: Involves stakeholders throughout the development process via sprint reviews and backlog prioritization.

Task 10: Agile Artifacts Documentation

To document the Agile methodology used for the Law Firm Management System, we created user stories, backlogs, and burndown charts using Trello and Excel. These artifacts provided clarity in tracking progress, defining objectives, and aligning with the Scrum framework.

1. User Stories

User stories were written in the format: *"As a [user], I want to [action], so that [goal]."* They reflected stakeholder requirements and were grouped under four themes: **Security**, **Client Management**, **Case Management**, and **Procedures Management**.

Security Features

1. **Log in**
    - *As a user, I want to log into the system using my credentials, so that I can access my designated section securely.*
2. **Log out**
    - *As a user, I want to log out of the system, so that my session ends and unauthorized*

access is prevented.

3. **Change User Password**
    - *As a user, I want to change my password, so that I can ensure my account remains secure.*
4. **Create User**
    - *As an administrator, I want to create new user accounts, so that I can grant access to the system based on roles.*
5. **Delete User**
    - *As an administrator, I want to delete user accounts, so that I can revoke access for users who no longer need it.*
6. **Modify User**
    - *As an administrator, I want to update user information, so that user records remain accurate and current.*

Client Management Features

7. **Create Client**
    - *As a lawyer, I want to create a client profile, so that I can maintain accurate records of my clients.*
8. **Delete Client**
    - *As a lawyer, I want to delete client profiles, so that I can remove records that are no longer relevant.*
9. **Modify Client**
    - *As a lawyer, I want to edit client details, so that I can ensure client records remain up to date.*
10. **Consult Client**
    - *As a lawyer, I want to view client information, so that I can quickly access details when needed.*

Case Management Features

11. **Create Case**
    - *As a lawyer, I want to create a new case, so that I can manage legal work efficiently.*
12. **Delete Case**
    - *As a lawyer, I want to delete a case, so that I can remove cases that are no longer required.*
13. **Modify Case**
    - *As a lawyer, I want to edit case details, so that I can keep case information accurate.*
14. **Consult Case**
    - *As a lawyer, I want to view case information, so that I can review case details as needed.*

Procedures Management Features

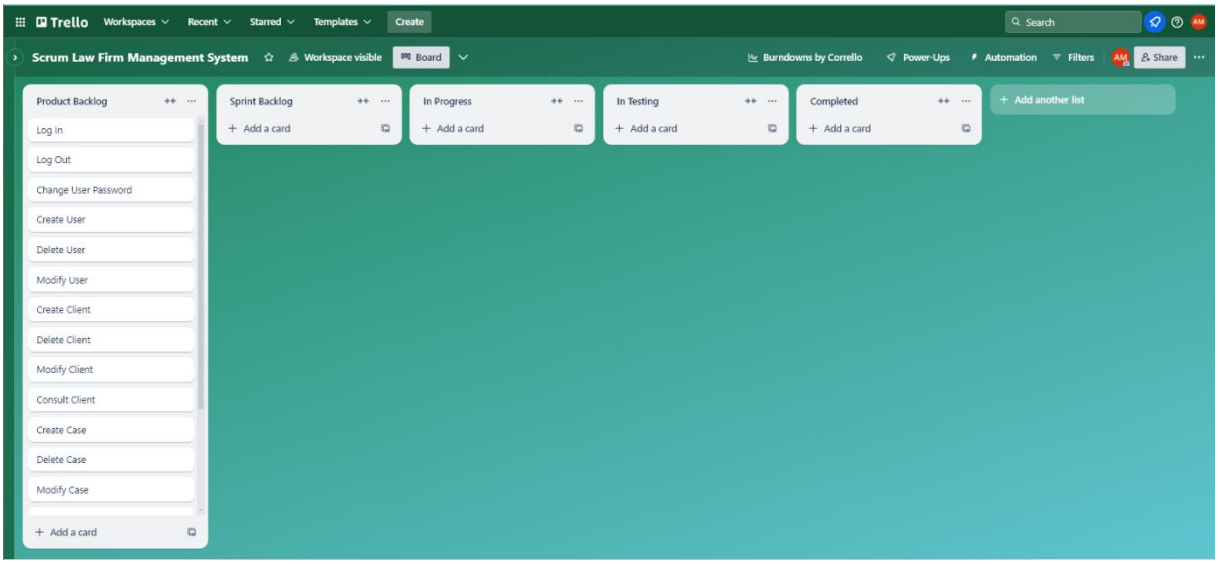
15. **Create Procedure**
    - *As a lawyer, I want to create a procedure within a case, so that I can document actions related to the case.*
16. **Delete Procedure**
    - *As a lawyer, I want to delete a procedure, so that I can remove unnecessary or redundant actions.*
17. **Modify Procedure**
    - *As a lawyer, I want to update procedure details, so that I can ensure the information is correct and complete.*
18. **Consult Procedure**
    - *As a lawyer, I want to view procedures associated with a case, so that I can understand the case's progression*

2. Backlogs

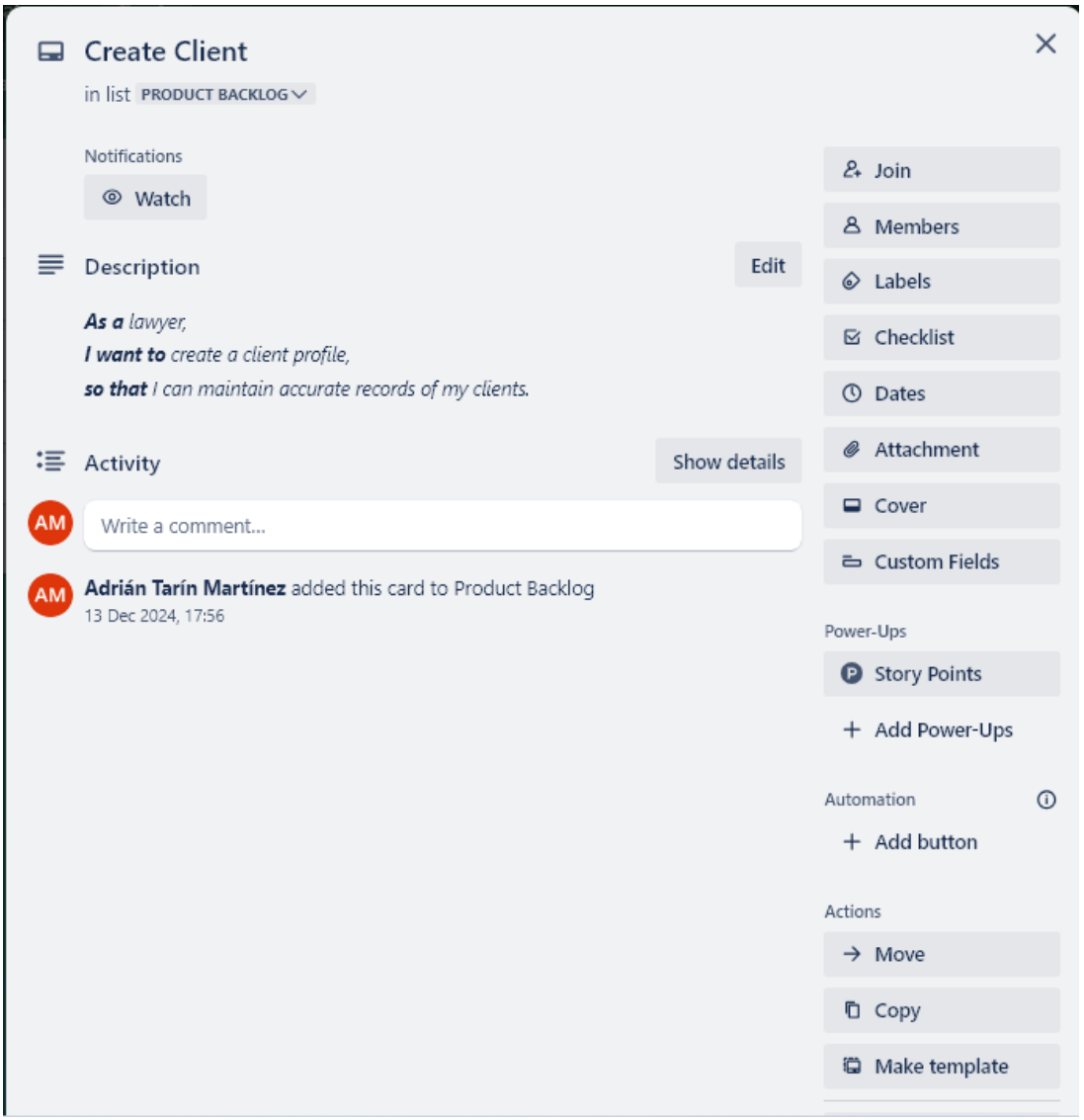
The product backlog in Trello prioritized user stories based on business value and feasibility. Sprint backlogs were created for critical features, ensuring focus and effective time management.

Trello Structure:

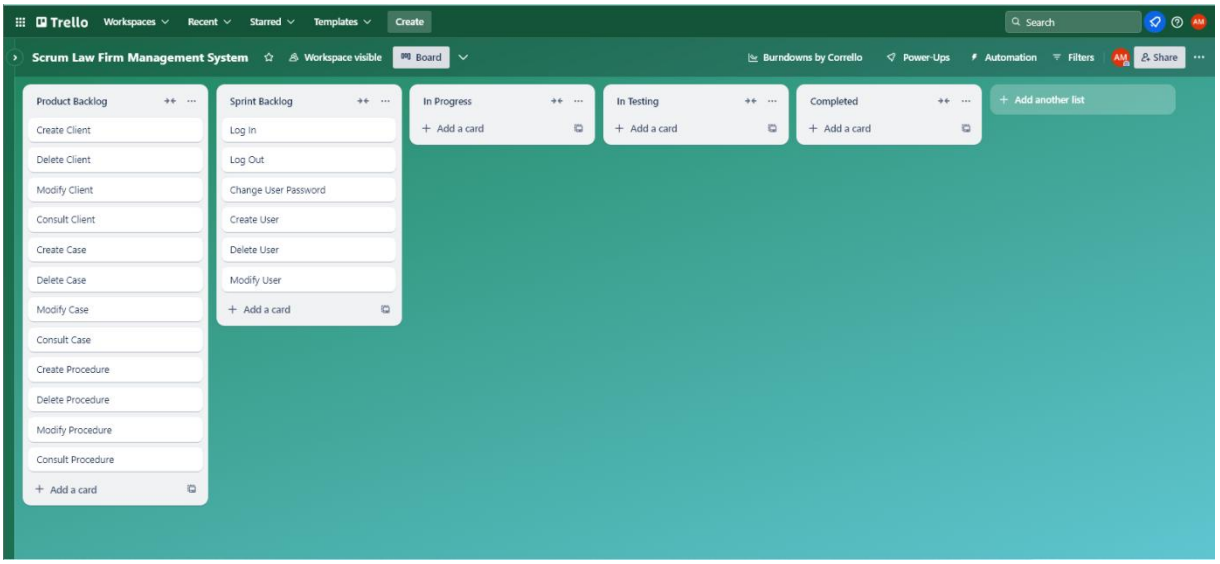
- **Product Backlog:** At the start, all tasks are listed in the **"Product Backlog"** column. This column holds every user story and task to be implemented, prioritized by business value and feasibility.



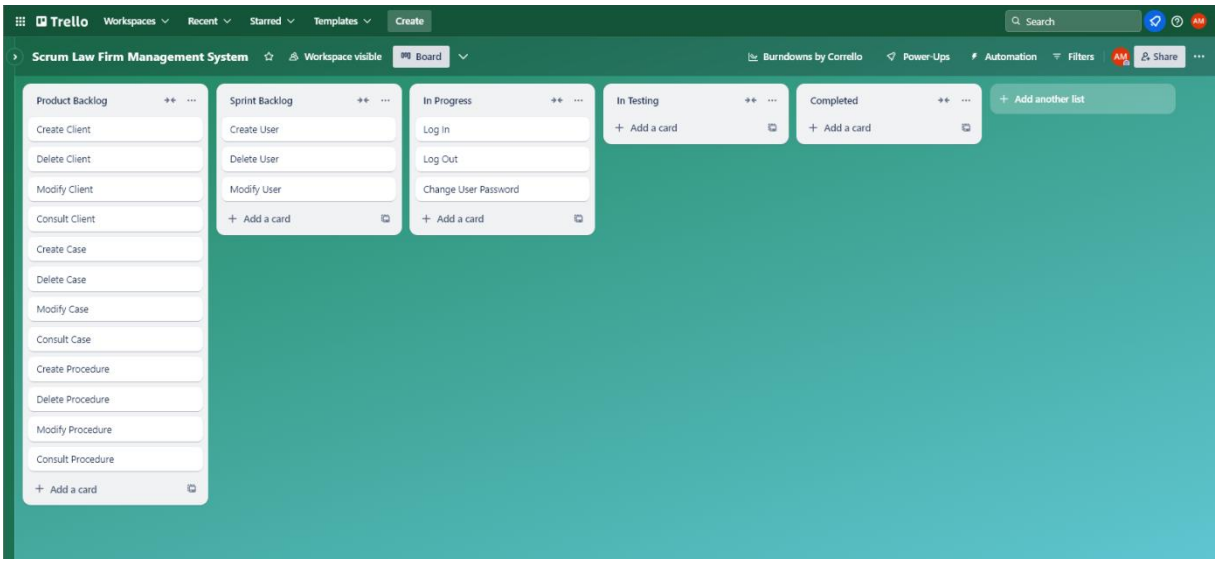
Each Trello card contains detailed information, including the user story previously mentioned. Within the card, we can add a description, assign team members, apply labels for categorization, create checklists for subtasks, set due dates, and upload attachments to enhance clarity and collaboration.



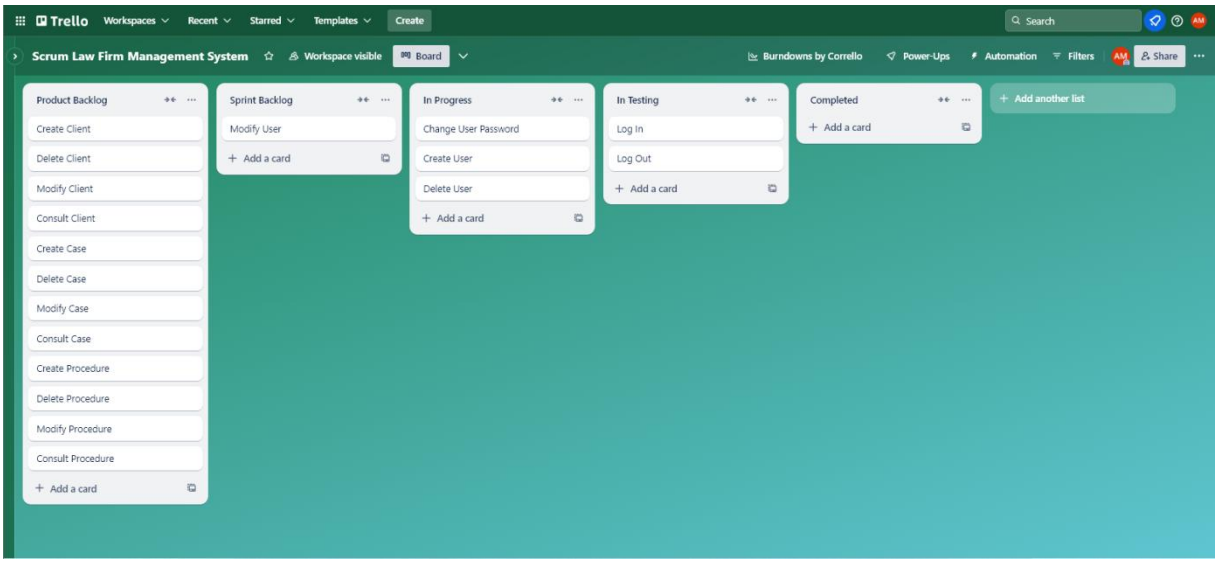
- **Sprint Backlog:** During sprint planning, selected tasks are moved to the **"Sprint Backlog"** column. For this sprint, we focused on tasks related to the *Security Features* outlined earlier. This ensures that the most critical elements are addressed in the current sprint.



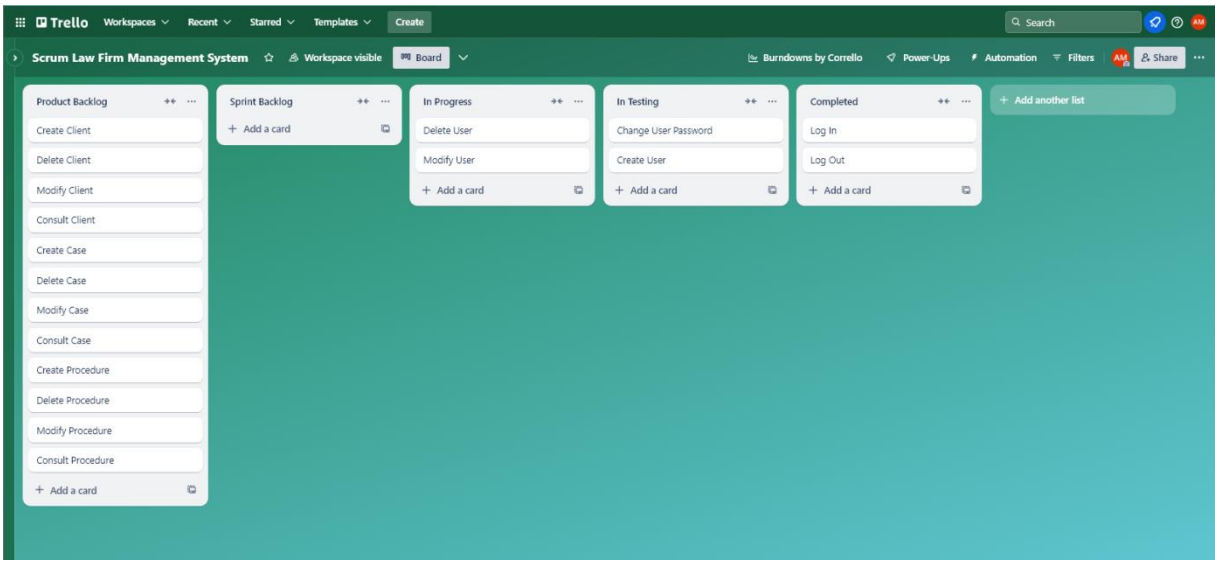
- **In Progress:** Once tasks are assigned to team members, they are moved to the **“In Progress”** column. This indicates that development work has started and team members are actively working on these tasks.



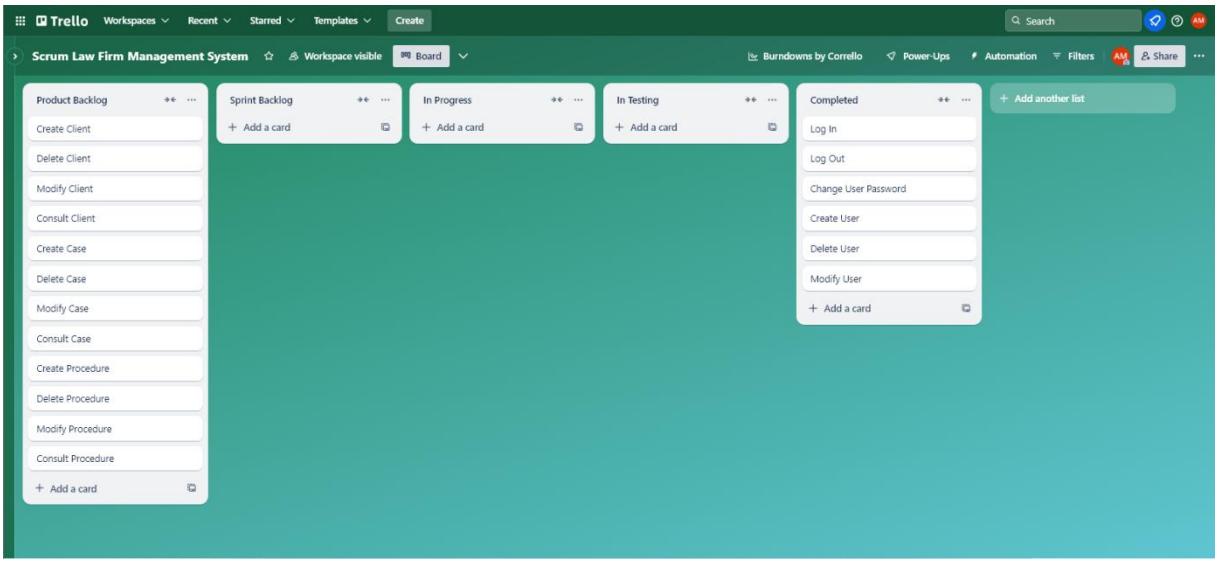
- **Testing:** After the development phase, tasks are shifted to the **“In Testing”** column for quality assurance. Here, unit testing and other verification processes are performed to ensure that the functionality meets the requirements and is free from defects.



- **Completed:** Once testing is successfully completed, the task is marked as done and moved to the **“Completed”** column. This reflects the successful delivery of the task, ready for deployment or review.



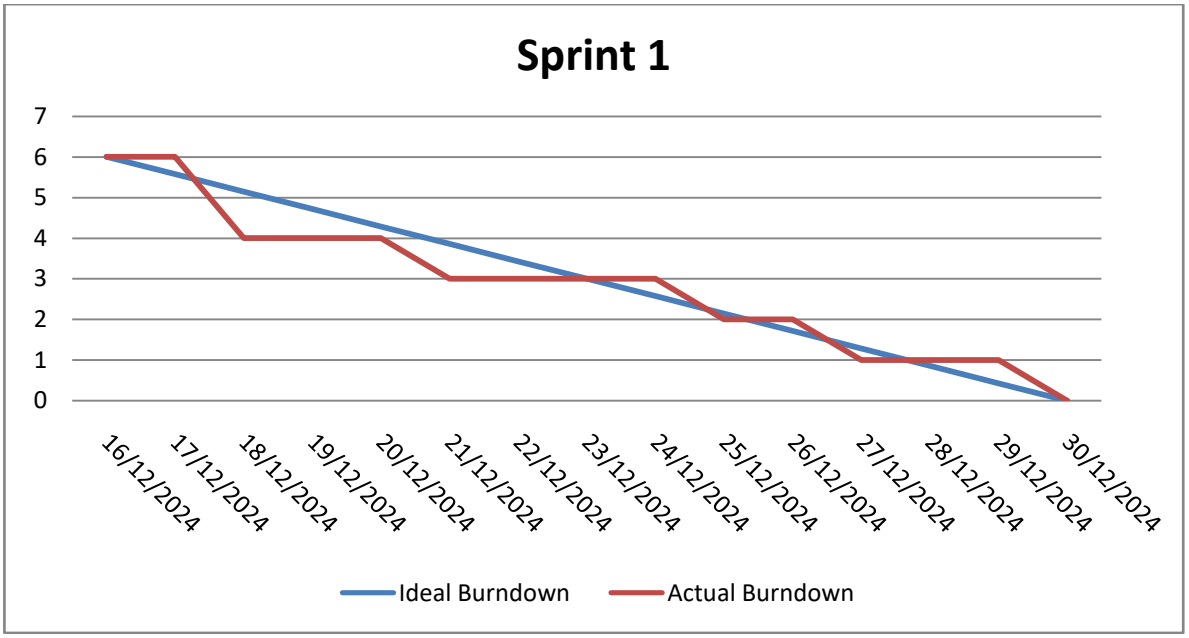
- **End of Sprint:** By the end of the sprint, all tasks from the sprint backlog should ideally move to the **"Completed"** column. This demonstrates that the sprint goals were achieved, and the deliverables are ready for the next phase or release.



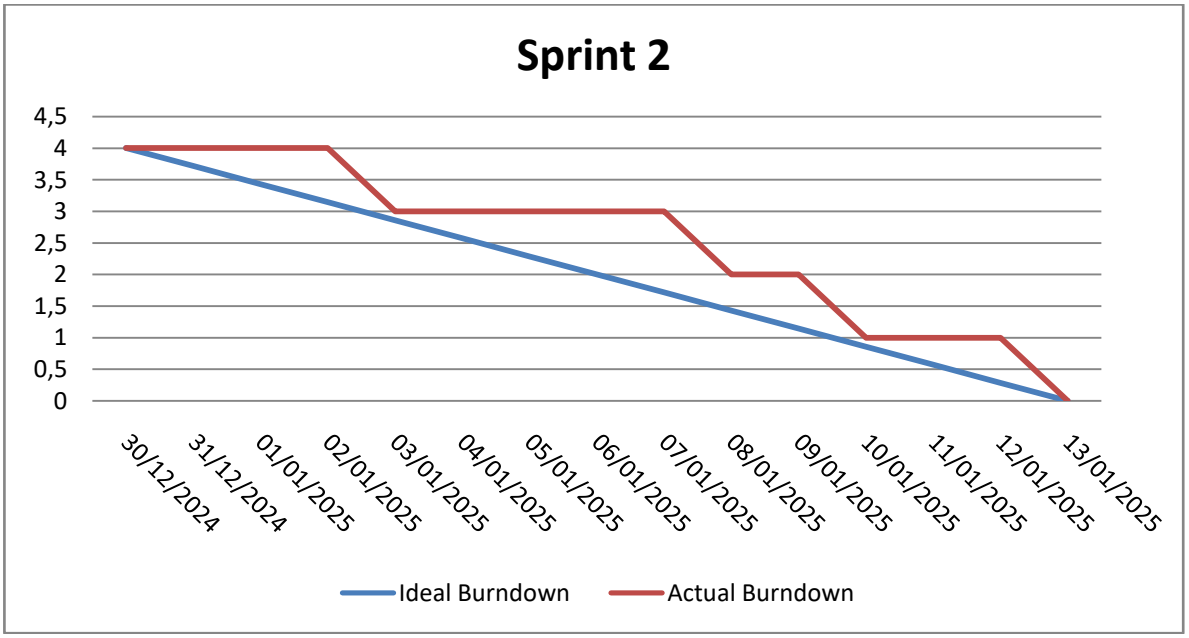
3. Burndown Charts

Burndown charts visualized progress by comparing remaining work against sprint duration. Using Excel sheets, we can update the chart daily to track task completion rates.

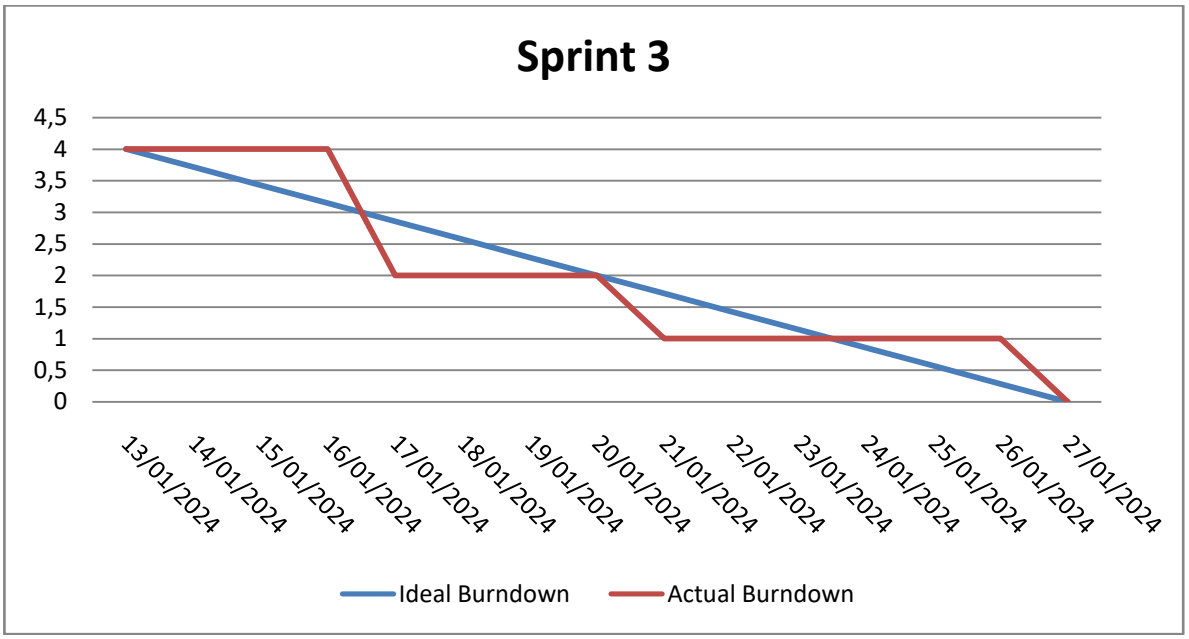
Sprint 1: Sprint 1: Security Features



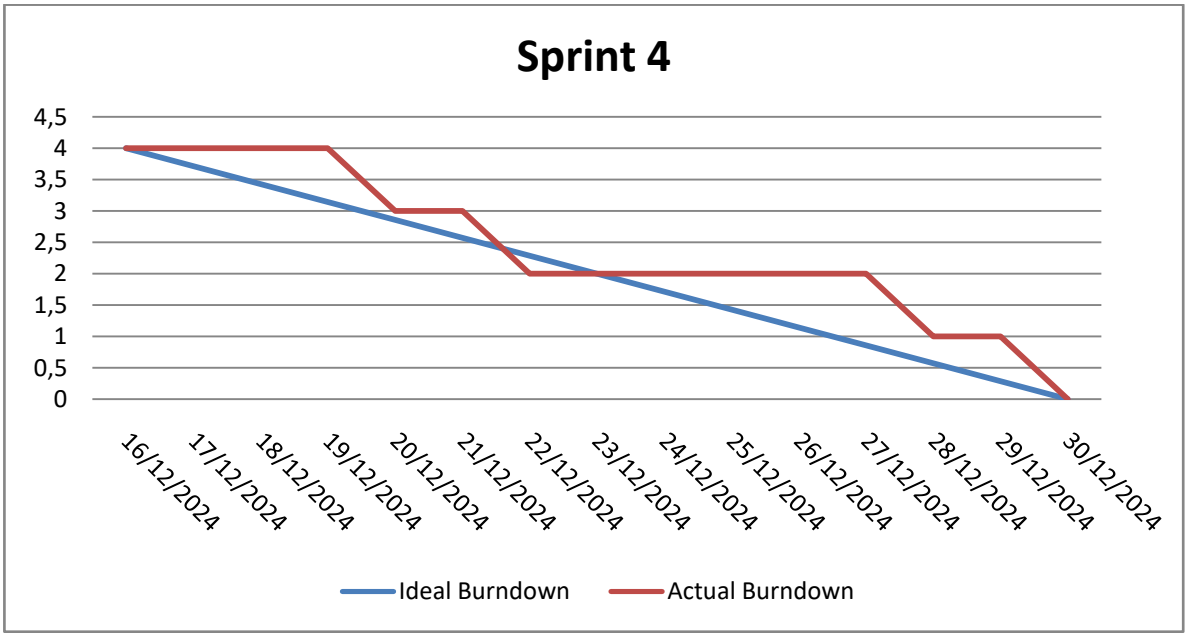
Sprint 2: Client Management Features



Sprint 3: Case Management Features



Sprint 4: Procedures Management Features



Task 11: Risk Assessment, Quality Management, and Communication Strategy

Risk Register

The risk register identifies potential project risks, evaluates their impact and likelihood, and outlines strategies for mitigation.

Risk ID	Description	Likelihood	Impact	Severity	Owner	Mitigation Actions	Status
R1	Changing stakeholder requirements	High	High	Medium	Product Owner	Regular sprint reviews with stakeholders to gather feedback early; maintain a prioritized product backlog.	Open
R2	Team member unavailability	Medium	High	High	Scrum Master	Cross-train team members; maintain documentation for knowledge transfer.	Open
R3	Technical issues with integration	Medium	Medium	Low	Development Team	Perform thorough testing in each sprint; allocate time for debugging and integration tasks.	Open
R4	Delays in approvals from stakeholders	Medium	High	High	Stakeholders	Set clear deadlines for feedback and approvals; maintain open communication channels.	Open
R5	Insufficient testing	Low	High	Low	QA Team	Implement test-driven development (TDD); conduct unit, integration, and acceptance testing in each sprint.	Open
R6	Miscommunication among team members	Medium	Medium	Low	Scrum Master	Use collaboration tools (e.g., Trello); hold daily standups to align team efforts.	Open
R7	Non-compliance with legal	Low	High	High	Legal Advisor	Conduct reviews with legal advisors and	Open

	protocols					ensure compliance with laws and regulations in requirements.	

Quality Management Strategy

Quality will be managed throughout the project lifecycle to ensure deliverables meet the desired standards and stakeholder expectations.

1. **Requirement Gathering:**
    - Conduct stakeholder interviews and surveys.
    - Document requirements clearly and validate them with stakeholders before proceeding.
2. **UML Diagrams:**
    - Review diagrams with both technical and non-technical stakeholders to ensure accuracy and clarity.
    - Follow best practices in UML diagram creation and version control for traceability.
3. **User Stories:**
    - Validate each user story with stakeholders for completeness and relevance.
4. **Product Backlogs:**
    - Prioritize the backlog items based on business value and urgency.
    - Conduct regular backlog refinement sessions to adjust priorities and include stakeholder feedback.
5. **Task Assignment:**
    - Use collaboration tools like Trello to assign tasks with clear deadlines and responsibilities.
    - Maintain a transparent view of task progress for all team members.
6. **Testing and Verification:**
    - Implement test-driven development (TDD) and continuous integration practices.
    - Regularly conduct sprint reviews and retrospectives to identify quality improvement opportunities.

Communication Management Strategy

Effective communication is critical to managing diverse stakeholders and ensuring smooth project progress.

1. **Internal Team Communication:**
    - **Daily Standups:** Conduct daily Scrum meetings to discuss progress, blockers, and next steps.
    - **Collaboration Tools:** Use tools like Slack or Microsoft Teams for instant messaging and file sharing.
    - **Documentation:** Maintain detailed and accessible project documentation in a shared repository.
2. **Stakeholder Communication:**
    - **Sprint Reviews:** Hold regular sprint reviews to demonstrate progress and gather feedback.
    - **Reports:** Provide stakeholders with progress updates, including burndown charts and completed tasks, at the end of each sprint.
    - **Feedback Loops:** Establish dedicated feedback sessions with stakeholders to ensure alignment on project objectives.
3. **Conflict Resolution:**

- Escalate conflicts to the Scrum Master or Project Manager for resolution.
- Use structured decision-making processes to address disagreements objectively.

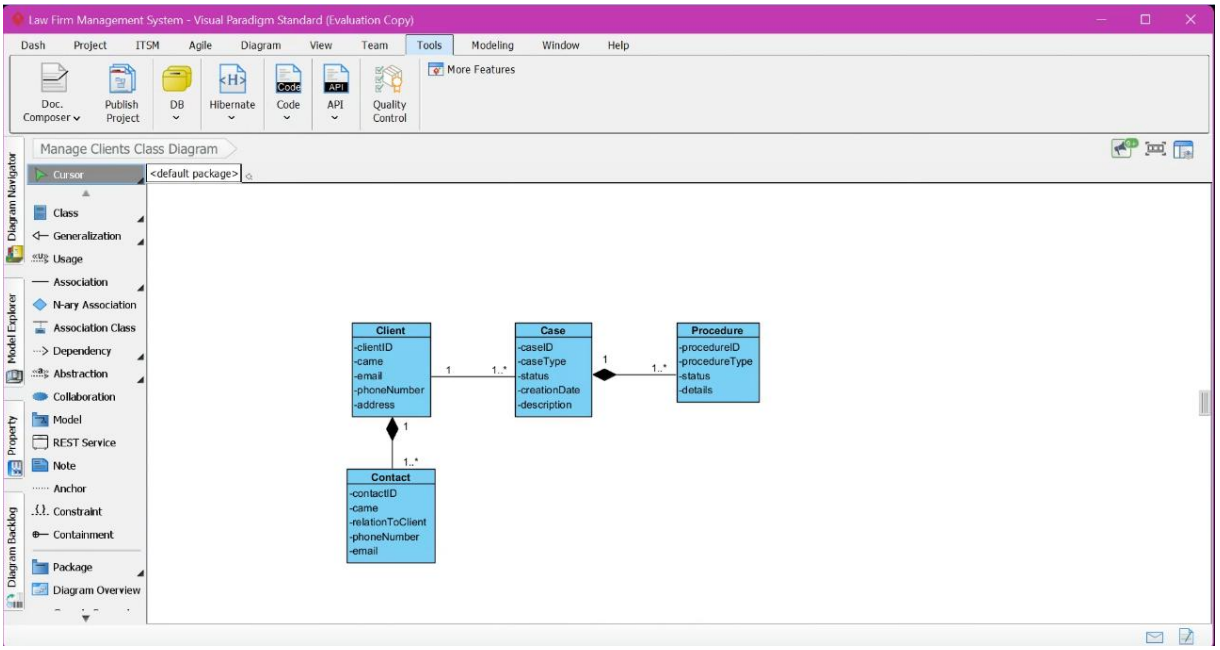
By ensuring active participation, open communication, and a clear focus on quality, the project can efficiently deliver a high-quality Law Firm Management System aligned with stakeholder expectations.

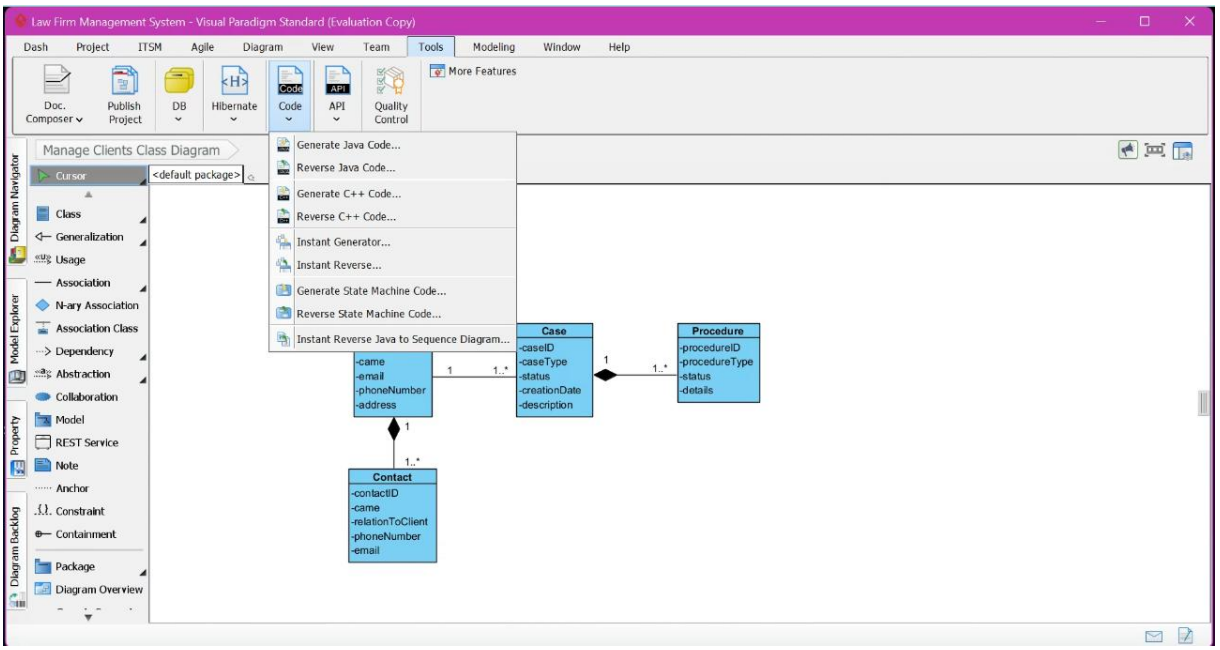
Task 12: Developing classes in Java

To implement the use case described in Part 1, Java was chosen as the programming language. The classes were developed using Visual Paradigm software, which enables the generation of Java code directly from the UML class diagram.

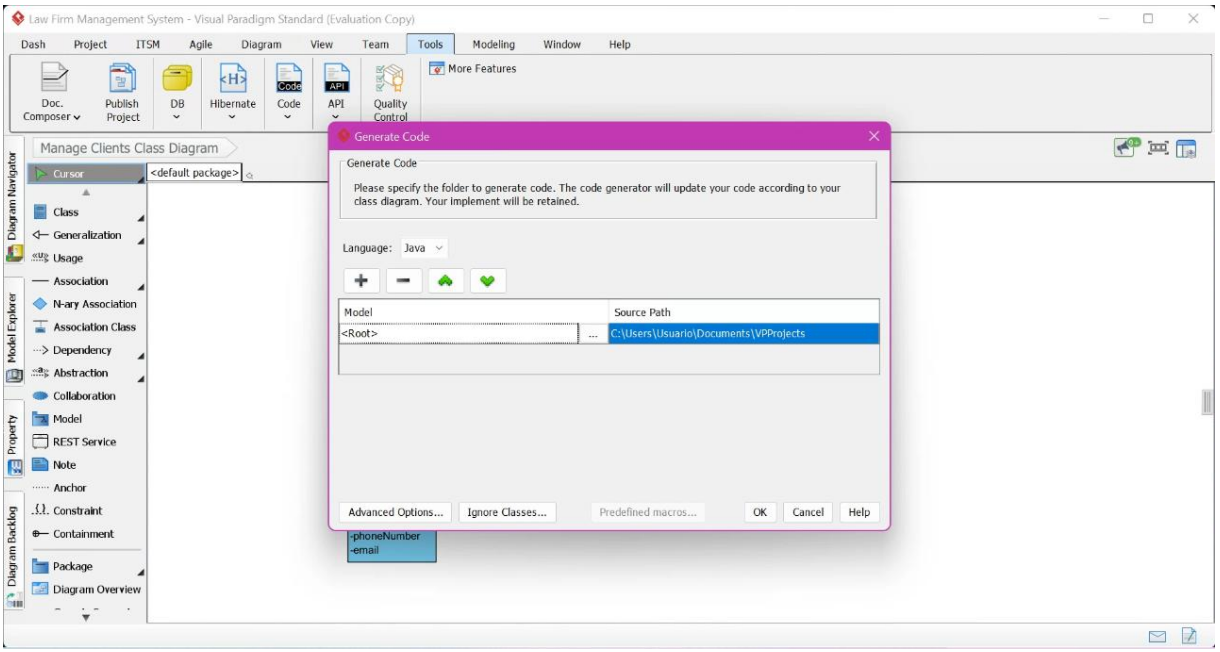
Steps to Generate Java Code in Visual Paradigm:

1. Navigate to: **Visual Paradigm > Tool > Code > Generate Java Code.**

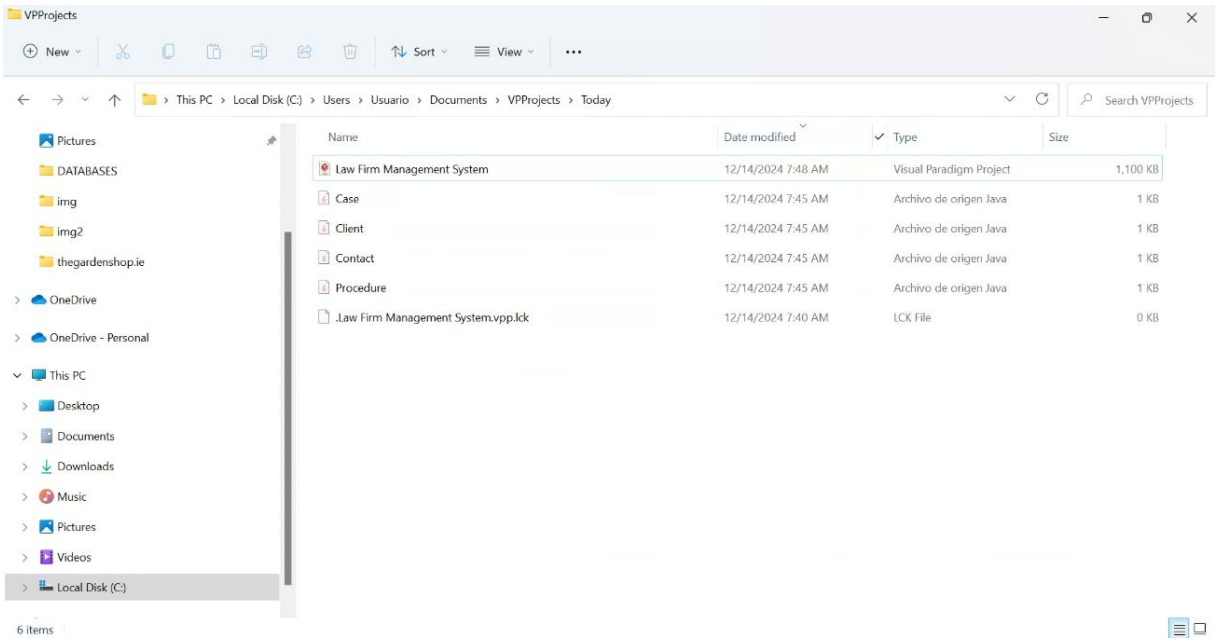




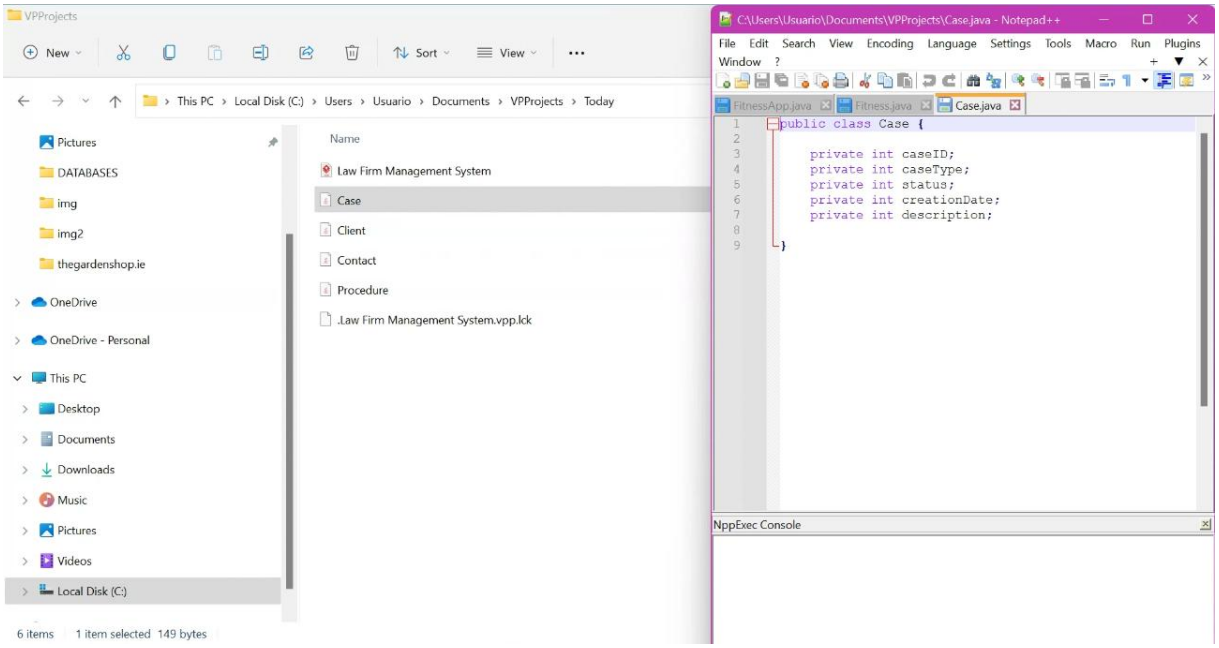
2. Specify the source path for the generated files and click **OK.**



Output:
The generated Java classes will be located in the specified folder.



Example of Generated Class:



Code of Generated Classes:

Case Class:

```
public class Case {

    private int caseID;
    private int caseType;
    private int status;
    private int creationDate;
    private int description;
```

```
}
```

Client Class:

```
public class Client {

    private int clientID;
    private int came;
    private int email;
    private int phoneNumber;
    private int address;

}
```

Contact Class:

```
public class Contact {

    private int contactID;
    private int came;
    private int relationToClient;
    private int phoneNumber;
    private int email;

}
```

Procedure Class:

```
public class Procedure {

    private int procedureID;
    private int procedureType;
    private int status;
    private int details;

}
```

Task 13: Testing

In this task, we developed and ran unit tests for the classes created in Task 12 to validate their functionality. Initially, the tests failed because the classes were incomplete or incorrectly implemented. After modifying the Client and Case classes, the tests passed successfully. Screenshots are provided to show the failed tests before modification and the passed tests after correction.

1. Test Methodology: Unit Testing

We follow the unit testing methodology to verify the functionality of individual classes and their interactions. Unit testing ensures that each class works as intended in isolation, allowing for early detection of errors.

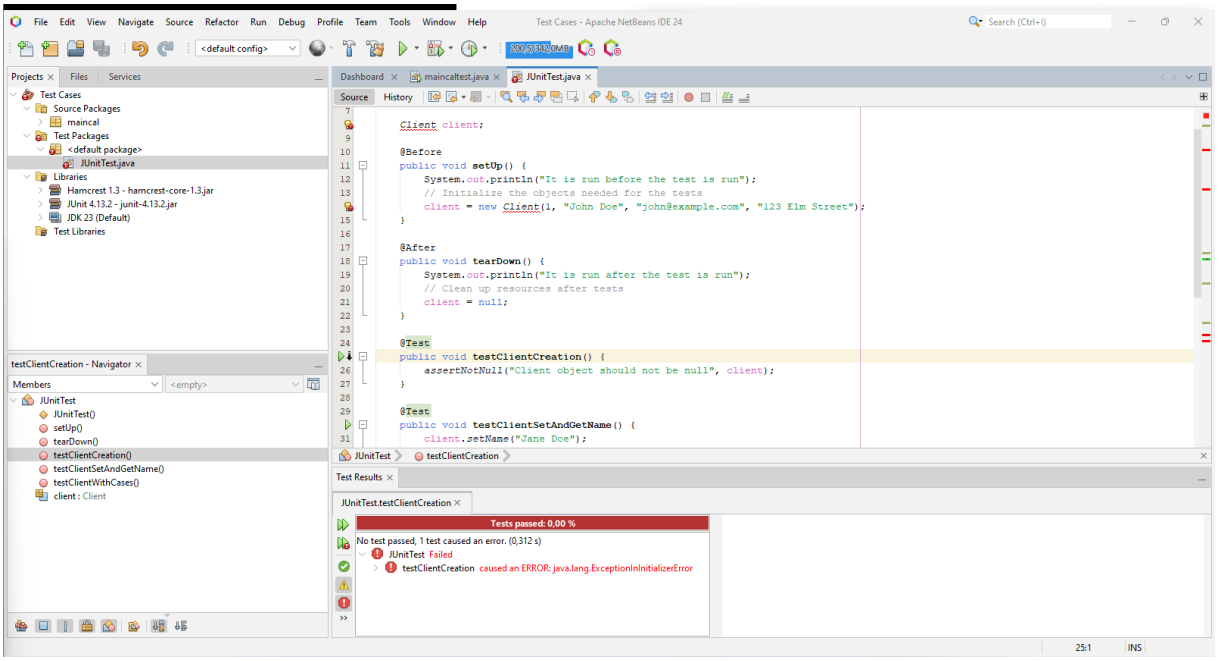
Justification for Unit Testing

1. **Focus on Individual Units**: Tests the smallest testable parts of the application (classes and their attributes/relationships).
2. **Error Isolation**: Identifies issues early in development by isolating problems to specific classes.

3. **Data Integrity**: Ensures relationships like one-to-many associations (Client to Case) work as intended.
4. **Repeatability**: Allows retesting during changes without re-executing the entire system.

2. Initial Test Failures

We wrote three test cases to verify the Client and Case classes:



1. **Test Client Object Creation:** Ensures the client object is created without being null.

```
@Test
public void testClientCreation() {
    assertNotNull("Client object should not be null", client);
}
```

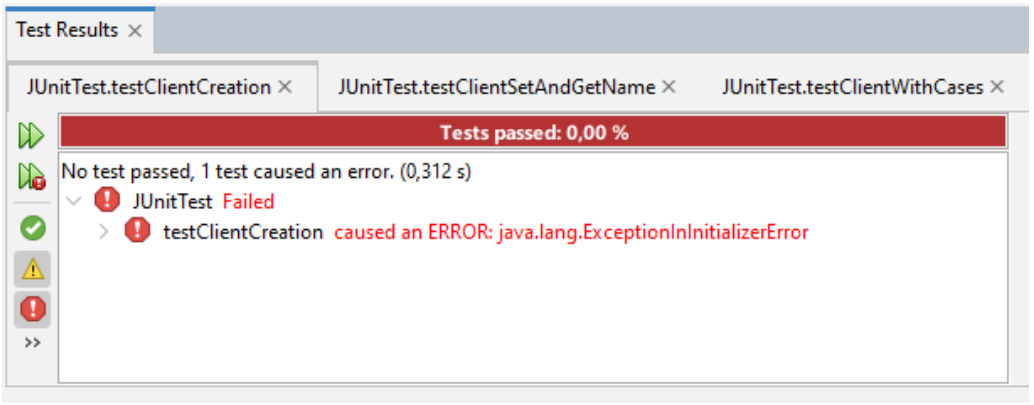
2. **Test Client Set and Get Name:** Verifies the setName and getName methods work correctly.

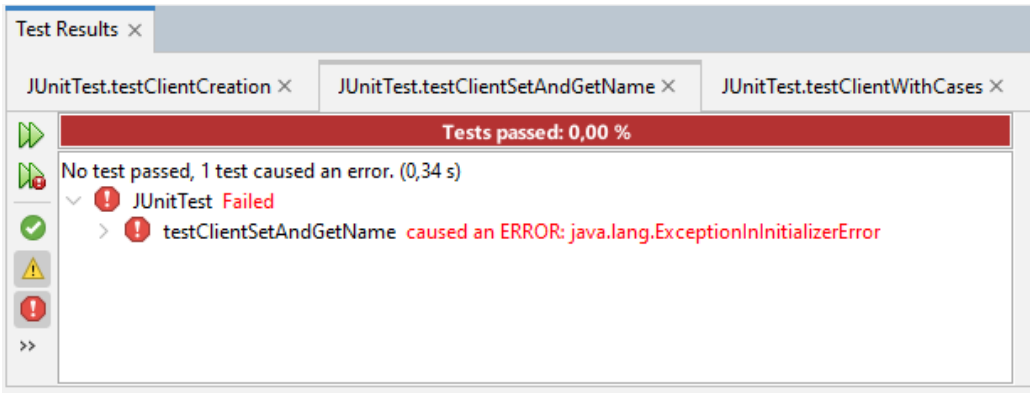
```
@Test
public void testClientSetAndGetName() {
    client.setName("Jane Doe");
    assertEquals("Name should be Jane Doe", "Jane Doe",
  client.getName());
}
```

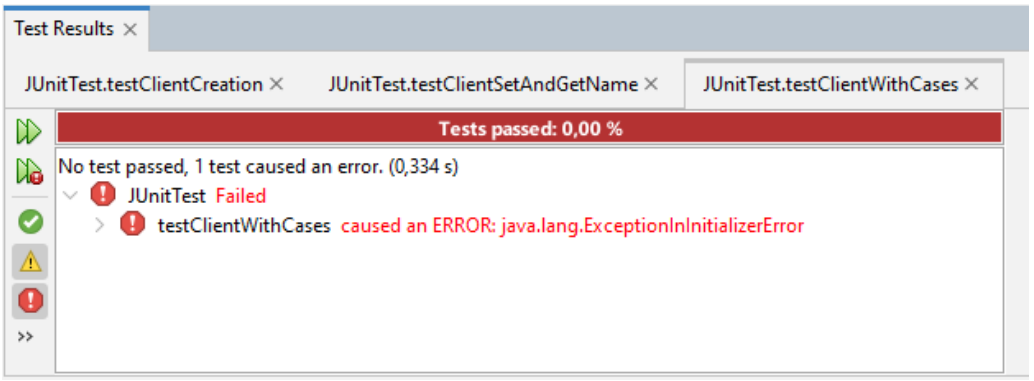
3. **Test Managing Client Cases:** Validates that cases can be added to a client and retrieved properly.

```
@Test
public void testClientWithCases() {
    Case case1 = new Case(101, "Civil", "Open");
    Case case2 = new Case(102, "Criminal", "Closed");
    client.addCase(case1);
    client.addCase(case2);
    assertEquals("Client should have 2 cases", 2,
  client.getCases().size());
}
```

These tests failed due to issues in the class implementations:







Initial State of Classes:

- The Case Class:

```
public class Case {
    private int caseID;
    private int caseType;
    private int status;
    private int creationDate;
    private int description;
}
```

-The Client Class:

```
public class Client {

    private int clientID;
    private int came;
    private int email;
    private int phoneNumber;
    private int address;

}
```

Result of Initial Tests:

- **Outcome:** All three tests failed.
- **Reason:** Missing or incorrect method implementations and data type issues.

3. Modifications to Classes

To address the issues, we refined the Client and Case classes as follows:

Updated Case Class:

1. Changed data types for caseType and status to String.
2. Simplified the class to focus on caseID, caseType, and status.

```
package Classes;

public class Case {

    private int caseID;
    private String caseType;
    private String status;

    // Constructor
    public Case(int caseID, String caseType, String status) {
        this.caseID = caseID;
        this.caseType = caseType;
        this.status = status;
    }

    // Getters and Setters (if needed)
    public int getCaseID() {
        return caseID;
    }

    public String getCaseType() {
        return caseType;
    }

    public String getStatus() {
        return status;
    }

}
```

Updated Client Class:

1. Added an ArrayList to store associated cases.
2. Implemented addCase and getCases methods for managing cases.
3. Ensured proper getters and setters for the client's attributes.

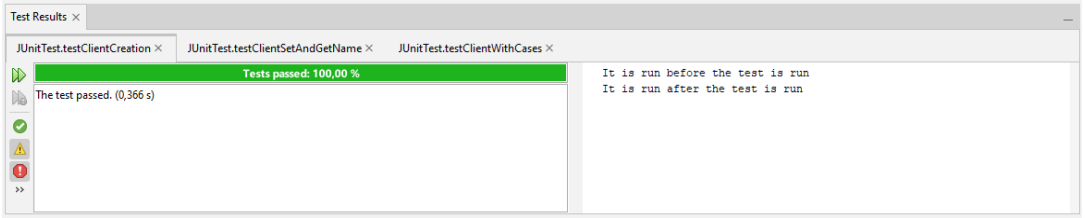
```
package Classes;
```

```
import java.util.ArrayList;
import java.util.List;

public class Client {

    private int clientID;
    private String name;
    private String email;
    private String address;
    private List<Case> cases;

    // Constructor
    public Client(int clientID, String name, String email, String address)
{
        this.clientID = clientID;
        this.name = name;
        this.email = email;
        this.address = address;
        this.cases = new ArrayList<>(); // Initialize the cases list.
    }

    // Getter and Setter for Name
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    // Methods to Manage Cases
    public void addCase(Case newCase) {
        cases.add(newCase);
    }

    public List<Case> getCases() {
        return cases;
    }

}
```

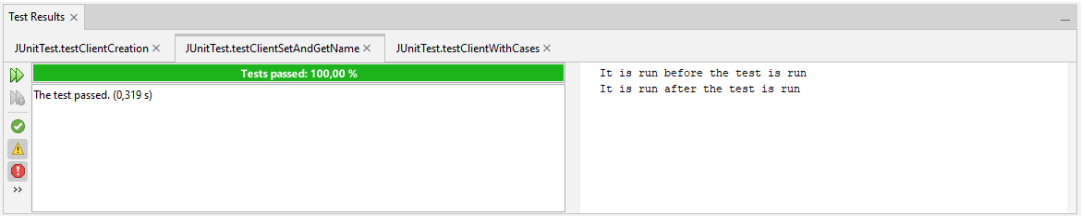
4. Passing Tests After Modifications

After updating the classes, the tests were re-run, and all tests passed successfully:

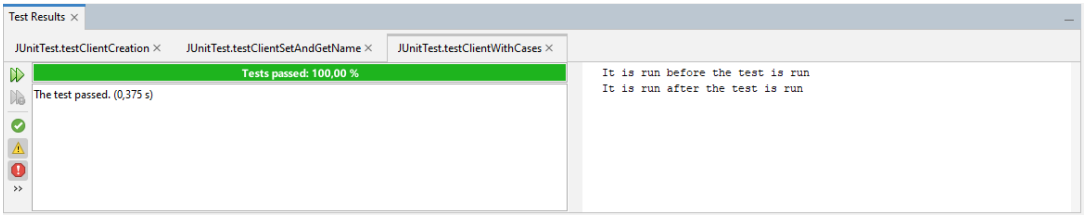
1. **Client Object Creation Test:** Confirmed the client object was created without issues.



2. **Set and Get Name Test:** Verified the name could be set and retrieved accurately.



3. **Managing Cases Test:** Validated that cases could be added to the client and retrieved as a list.



Example of Passing Test for Managing Cases:

```
@Test
public void testClientWithCases() {
    Case case1 = new Case(101, "Civil", "Open");
    Case case2 = new Case(102, "Criminal", "Closed");
    client.addCase(case1);
    client.addCase(case2);
    assertEquals("Client should have 2 cases", 2,
client.getCases().size());
}
```

Result of Final Tests:

- **Outcome:** All tests passed successfully.
- **Reason:** Correct implementation of methods and data types in the Client and Case classes.

5. Test Case Summary

Test ID	Test Case	Description	Expected Outcome	Initial Result	Final Result
T01	Client Object Creation	Verify that a Client object is correctly instantiated.	Client object should not be null.	Failed	Passed
T02	Setting Client Name	Validate the setName and getName methods.	Name should be updated and retrieved.	Failed	Passed
T03	Managing Client Cases	Verify addCase and getCases methods.	Client should correctly manage cases list.	Failed	Passed

6. Conclusion

The testing process highlighted the importance of iterative development and verification. Initial failures identified issues in the Client and Case classes, which were resolved through careful debugging and implementation. Screenshots document both failed and successful test runs, demonstrating the progress and correctness of the final implementation.