Scorpion Mine-Sweeper

SDP

1. Overview

This section provides a summary of the Scorpion Mine-Sweeper project and the scope of this Software Development Plan (SDP/SPMP). It defines the purpose, system scope, main objectives, assumptions, constraints, deliverables, and high-level schedule.

1.1. Project Summary

1.1.1 Purpose, scope, and objectives

Purpose:

Develop an educational cooperative Minesweeper-based game with integrated

knowledge questions. Built using Agile and MVC to provide a stable, engaging, and maintainable software product.

Scope:

Two synchronized game boards, multiple difficulty levels, shared lives/score, question management via CSV, scoring & life-tracking, GUI interface, automated result saving, high-scores screen, and a complete MVC architecture.

The project is limited to desktop development using Java, Swing, and CSV file handling.

Objectives:

Deliver a fully playable game in three iterations, with correct rules, fast performance (<1s), intuitive GUI, CSV-based question management, and compliance with software engineering standards; full requirements are defined in the Scorpion Mine-Sweeper SRS.

1.1.2 Assumptions and constraints

Assumptions: Development in Java MVC with keyboard/mouse input; CSV storage; Agile-Scrum 3 iterations.

Constraints: Limited semester timeline; desktop only; open-source tools; lightweight GUI compatible with standard Java runtimes.

1.1.3 Project deliverables

Documents: SRS, SDP/SPMP, design diagrams, test reports.

Software: Game logic engine, GUI, question manager, score/lives system, difficulty module, end-game & result tracking.

Final output: Functional game, executable JAR & source repository, optional user manual, updated CSV files.

1.1.4 Schedule and budget summary

Schedule – 3 Iterations

- **Iteration 1:** Core game logic → Console prototype
- **Iteration 2:** GUI development → Functional visual interface
- **Iteration 3:** Full version → Complete playable game

Effort & Budget

Workload shared among four team members across planning, implementation, and testing.No financial budget. Resources: team time, availability, and free open-source tools.

2. References

The following references were used during the creation of the SDP and system design:

1- IEEE Std 1058 – Software Project Management Plans
2- IEEE Std 830 – SRS Specification Standard
3- Course guidelines and templates provided by the instructor
4- Java SE Documentation (Oracle)
5- Java Swing GUI toolkit documentation
6- GitHub documentation for version control workflow
7- Scorpion Mine-Sweeper SRS (internal document)
8- Discord – Team Communication Platform

3. Definitions

Term	Definition
MVC	Model–View–Controller architectural pattern
Iteration / Sprint	Agile development cycle delivering working software
CSV	File format for storing questions/results
User Story	Requirement from user perspective
PO (Product Owner)	Approves and verifies requirements
Difficulty Configuration	Parameters for scoring, board size, mines, etc.
Board Cell	A tile in the game grid (mine/bonus/question/empty)

QA	Quality assurance testing
Merge Request	Request to merge reviewed code into main branch

5. Managerial process plan

5.2 Work plan

The project is developed using the **Agile iterative model** across **three sprints**, delivering functional increments that gradually expand features and quality until a fully playable cooperative game is completed.

5.2.1 Work activities

Work Breakdown Structure (WBS) by Iteration

Iteration 1	**Implementing the basic logic of the game**
User Story 1	As a player, I want to start a new cooperative game with a selected difficulty (Easy/Medium/Hard), so that two boards are created with shared lives and score.
User Story 2	As a player, I want the system to randomly place mines, question cells, surprise cells and numbers, so that every board is fair and balanced.
User Story 3	As a player, I want revealing or flagging a cell to update the shared score and lives based on the rules, so that results are clear and consistent.
User Story 4	As a player, I want the game to detect win/lose when safe cells are done or lives reach zero, so that I know when the game ends.
User Story 5	As a player, I want used question/surprise cells to be skipped when already used, so they cannot be reused.
User Story 6	As a player, I want empty areas to open automatically, so I don’t need to click each cell manually.
Iteration 2	**Implementing the GUI of the game**

User Story 1	As a player, I want a main game screen with two boards (different colors), shared hearts/score, remaining mines, and turn indicator, so that I understand the game state.
User Story 2	As a player, I want to reveal cells and place/remove flags using mouse clicks, so interaction is simple.
User Story 3	As a player, I want pop-up windows for question/surprise cells with multiple-choice answers and feedback, so I immediately see score/life changes.
User Story 4	As a player, I want a clear team-turn indication, so we can coordinate play.
User Story 5	As a player, I want to see clear messages and visual feedback when I hit a mine, answer correctly, or lose a life, so that I can easily understand the outcome of every action.
User Story 6	As a player, I want an end-game summary showing score, lives, and result, so I know the final outcome.
User Story 7	As a new player, I want a short help/instructions screen, so I can learn quickly.
Iteration 3	**Building a more full featured version**
User Story 1	As a player, I want results (names, difficulty, score, lives, correct answers, duration) saved automatically, so I can track performance.
User Story 2	As a player, I want to view a history/high-scores screen from CSV, so I can compare previous results.
User Story 3	As a user, I want to add/edit/delete questions, so the question bank can stay updated.
User Story 4	As a player, I want safe loading and error handling for CSV/config files, so the system remains reliable.
User Story 5	As a player, I want fast gameplay (<1s response), so the experience feels smooth.

Detailed Plan for Iteration 1

Iteration	User story	Task	Executor

Iteration 1	As a player, I want to start a new cooperative game with a selected difficulty (Easy/Medium/Hard), so that two boards are created with shared lives and score.	- Design cooperative start-game flow (two boards, shared score/lives) - Implement board creation and difficulty mapping (board size, mines, special cells, starting lives) - Test start/restart behavior end-to-end	Developer 1
Iteration 1	As a player, I want the system to randomly place mines, question cells, surprise cells and numbers, so that every board is fair and balanced.	- Implement random placement algorithm - Validate distribution according to difficulty rules (statistical check)	Developer 2
Iteration 1	As a player, I want revealing or flagging a cell to update the shared score and lives based on the rules, so that results are clear and consistent.	- Implement reveal/flag logic and scoring rules - Integrate life changes from difficulty table - Unit tests for scoring, life update, and flag states	Developer 3
Iteration 1	As a player, I want the game to detect win/lose when safe cells are done or lives reach zero, so that I know when the game ends.	- Implement win/lose handling and status functions - Display final status (console prototype)	Developer 3
Iteration 1	As a player, I want used question/surprise cells to be skipped when already used, so they cannot be reused.	- Implement and store “used” state on activation - Logic validation to prevent re-opening	Developer 2

Iteration 1	As a player, I want empty areas to open automatically, so I don’t need to click each cell manually.	- Implement recursive reveal for empty chain expansion - Test behavior on sample boards	Developer 4
Iteration Acceptance Test		- Full test of game logic (start, reveal, scoring, win/lose, auto-reveal, used cells)	Developer 4

5.2.2 Schedule allocation

Task Number	Description	Execution Week
1	Team formation, role distribution, full requirements definition, SRS completion, user story list	**Week 1–2**
2	**Iteration 1 – Core Logic:** game initialization, board creation, difficulty setup, random placement, scoring/lives rules, win/lose detection, recursive reveal	**Week 3–4**
3	**Iteration 2 – GUI:** full interface, two colored boards, mouse interactions, pop-ups, turn indicator, visual feedback, end-game summary, help screen	**Week 5–8**
4	**Iteration 3 – Advanced Features:** saving to CSV, history & high-scores screens, question CRUD, safe file handling	**Week 9–10**
5	Integration, performance optimization, full testing, bug fixing, update SDP, prepare user manual	**Week 11–12**
6	Final delivery, presentation, exported JAR, multi-device validation	**Week 13**

5.3 Control plan

The control plan defines how requirements, quality, and progress are monitored throughout development.

Its goal is to ensure alignment with objectives, maintain quality, detect issues early, and confirm that each iteration delivers a stable working version. Control focuses on:

Requirements control, Quality control and Progress and iteration tracking

Control is maintained through **Agile iterations**, **GitHub version management**, **team reviews**, and **testing at the end of each iteration**.

5.3.1 Requirements control plan

Requirements Traceability

All requirements (User Stories) are traced to development tasks, implementation components, and test cases, ensuring full coverage and consistency through all development stages.

Change Control Procedure

Requirement changes are submitted as Change Requests in GitHub with full description and impact analysis on game logic, schedule, architecture (MVC), and risks. Changes require Product Owner approval, update the SDP/backlog/WBS, and are followed by a new Git version tag.

Requirement Freeze Policy

Requirements are locked at the start of each iteration; only bug fixes, minor UI adjustments, and low-impact updates are allowed to maintain stability.

Reviews and Prototypes

Iteration 1 uses a console prototype, Iteration 2 a GUI prototype, and Iteration 3 includes usability testing and full flow review.

Version and Documentation Control

All significant changes are logged, versioned in Git, and reviewed before merging to ensure controlled evolution and documentation accuracy.

5.3.4 Quality control plan

The quality plan ensures that the system meets functional, performance, and usability expectations. It defines the testing strategy, review process, and acceptance criteria for all deliverables.

Code Quality Assurance

All code changes undergo peer review through Merge Requests, follow Java coding standards, maintain strict MVC separation, and use small maintainable classes with clear naming conventions.

Verification (Technical Testing)

During Iteration 1, unit tests (JUnit) validate core logic, and integration tests verify interactions between the Board, Cell, Difficulty, and Score modules. Both standard and edge-case scenarios are tested, including boundary behaviors and used-cell handling.

Validation (Functional Testing)

Full gameplay is tested in console mode in Iteration 1 and in the GUI in Iteration 2, with usability and user experience validation in Iteration 3. CSV files for questions and results are also validated.

Acceptance Testing

At the end of each iteration, the Product Owner verifies correct game initialization, mine/question distribution, scoring and life updates, end-game logic, and proper handling of used cells.

Responsibilities

Developers perform unit and integration testing, peer reviewers ensure code quality and compliance, and the Product Owner conducts acceptance testing.

5.4 Risk management plan

Risk 1 – Unbalanced Mine/Question Distribution

Problem: Distribution algorithm may not match difficulty settings.

Impact: Unfair gameplay and inconsistent player experience.

Mitigation: Statistical testing on 100–200 boards, validate configuration tables, check consistency.

Owner: Rema Naamneh

Risk 2 – CSV File Corruption (Questions / Results / Configuration)
Problem: CSV files may be missing or unreadable.
Impact: Game cannot load questions or save results.
Mitigation: File validation, default CSV creation, exception handling and logging.
Owner:

Risk 3 – Poor MVC Layer Separation
Problem: Mixing Model/View/Controller responsibilities.
Impact: Harder maintenance, more bugs, reduced code clarity.
Mitigation: Architecture review each iteration, strict separation, peer-reviewed merges.
Owner: Maria Abergel

Risk 4 – Incorrect Difficulty Configuration
Problem: Difficulty settings may not differentiate gameplay meaningfully.
Impact: Unbalanced challenge and reduced engagement.
Mitigation: Maintain configuration table, large-scale testing, cross-validation of scoring and lives rules.
Owner: (Assign)

6. Technical process plan

This section specifies the technical approach used to develop the Scorpion Mine-Sweeper system, including the lifecycle model, workflows, tools, testing strategy, and acceptance milestones.

6.1 Process model

The project follows an **Agile Iterative & Incremental model** across **three iterations**, each providing a functional product increment with continuous feedback, testing, and risk control.
Development begins with **Project Initiation** (team roles, requirements analysis, SRS completion, user story prioritization, work distribution, and setup of tools/repository).
During **Iteration Planning**, user stories are selected and decomposed into tasks (WBS) and responsibilities.

In the **Design & Architecture phase**, system interfaces, MVC structure, game logic, GUI layout, and object model are defined and refined throughout the three iterations. The **Implementation & Integration phase** develops features through continuous integration via GitHub with incremental builds.

Verification & Testing: unit tests in Iteration 1, integration testing in Iteration 2, and full system validation (performance, reliability, usability) and acceptance review in Iteration 3.

Review & Retrospective sessions evaluate progress, update risks, and refine planning.

The process concludes with **Project Closure**, including delivery of the executable JAR, documentation, and final presentation.

Milestones:

- Iteration 1 – Console logic prototype (functional review)
- Iteration 2 – Playable GUI prototype (interface review)
- Iteration 3 – Fully integrated final system (acceptance review)

Tailoring: Agile is adapted to academic constraints using fixed schedule milestones and parallel work distribution due to limited time.

6.2 Methods, tools and techniques

Development is based on **Java 19+** using the **MVC architecture** and **Java Swing** for GUI. **CSV files** provide persistent storage for questions, scores, and configuration. Tools include Eclipse/IntelliJ for development, Git & GitHub for version control and collaboration, JUnit for automated unit testing, Google Sheets/Excel for CSV editing, and Microsoft Word for documentation. Engineering techniques include object-oriented design, event-driven programming, recursive reveal logic, randomized placement of special cells, continuous integration, and iterative testing. Documentation follows **IEEE-830 (SRS)** and **IEEE-1058 (SPMP)** standards, and coding follows the **Java Style Guidelines**.

7. Supporting process plan

This plan defines the supporting processes throughout project development.

Configuration management is performed using GitHub with feature branches, peer-reviewed merges, and version tags at each iteration.

Quality assurance includes merge-request reviews, compliance with Java standards and MVC structure, and verification of acceptance criteria defined per user story.
Documentation (SRS, SDP, test reports, user manuals) is maintained in Google Drive following IEEE-830 and IEEE-1058 standards.
Reviews and audits occur at the end of each iteration, including peer review of code and documents.
Problem resolution is managed via GitHub Issues, with prioritization by severity, assignment to team members, and validation before closure.
Subcontractor management is not required because the project is developed entirely by the internal team.

7.2 Verification and validation plan

This plan defines the **scope, tools, techniques, responsibilities**, and **independence** of V&V activities. Verification and validation apply to all **functional and non-functional requirements**, including **game logic**, **board creation**, **reveal/flag actions**, **random placement**, **win/lose detection**, **GUI interaction**, **CSV saving/loading**, **performance (response < 1s)**, **usability**, and **accessibility**.

Verification ensures the product is **built correctly** through **traceability mapping (User Stories → Tasks → Test Cases)**, **milestone reviews**, **peer code/document reviews**, **prototypes** (console in Iteration 1, GUI in Iteration 2), **unit testing with JUnit**, and **integration testing** between GUI and game logic.

Validation ensures the **right product is built** using **system-level testing**, **end-to-end gameplay scenarios**, **acceptance tests**, **scoring and file-handling validation**, and **manual usability testing**.
Automated tools include **JUnit** (testing) and **GitHub Issues** (bug tracking, prioritization, progress). Responsibilities: **Developers** perform implementation and initial testing, **Peer Reviewers** verify correctness independently, and the **Team** performs system validation and acceptance tests. **Independence** is ensured by separating implementation from review.

Success criteria: all user-story test cases pass, no high-severity issues remain, iteration milestones are approved, and the final integrated version is accepted. This V&V process ensures **continuous quality control** and confirms that the system meets all requirements and expectations.

7.3 Documentation plan

The **Documentation Plan** defines how all project documents will be created, reviewed, maintained, and delivered throughout the development of the *Scorpion Mine-Sweeper* system. Its purpose is to ensure **clarity, consistency, and traceability** across all phases of the project.

The documentation includes both **non-deliverable internal artifacts** and **deliverable materials for final submission**. Non-deliverable documentation includes the **SRS**, **SDP/SPMP**, **design documentation (including MVC architecture diagrams)**, **traceability records**, **test plans and test reports**, **meeting summaries**, and **review comments**. Deliverable documentation includes the **final working game (.JAR executable)**, **source code (GitHub repository)**, the **User Manual**, the **final presentation**, and supporting **CSV datasets**.

Roles and responsibilities are distributed among team members: the **Project Manager** supervises documentation progress and approves baseline versions, the **development team** prepares technical design and code-related documentation, and the **QA team** produces validation and testing documentation, including test results. All documentation follows the **IEEE-830 SRS standard** and **IEEE-1058 SPMP standard**, and complies with **Java coding style** requirements.

Documentation is developed **iteratively along with project milestones**. Drafts are shared via **Google Drive and GitHub**, reviewed at the end of each iteration, and only approved documents are stored as **baseline versions**. All materials remain accessible to team members and stakeholders to support accurate verification, validation, and final delivery at project closure.