Lab 2 – Software Inspections

1. Software Inspections of user requirements defined in Lab session 1 – Railway ticketing portal:
    a. Create the team for Software Inspections:

Team Lead (Moderator):

- Responsible for coordinating and facilitating the inspection process.
- Ensures that the inspection team adheres to the defined process and schedule.
- May also be responsible for summarizing findings and tracking action items.

Business Analyst (Inspector):

- Brings domain expertise and a deep understanding of the railway ticketing industry.
- Ensures that the requirements align with business objectives and industry standards.
- Focuses on the clarity and accuracy of the requirements.

Software Architect (Author):

- Examines the requirements from a technical perspective.
- Ensures that the requirements can be implemented and integrated into the existing system architecture.
- Evaluates the feasibility of implementing dynamic pricing rules and integrating with external systems.

Quality Assurance (QA) Analyst (Reader):

- Focuses on the testability and verifiability of the requirements.
- Identifies potential test cases and test scenarios based on the requirements.
- Ensures that the requirements support robust testing.

User Experience (UX) Designer (Reader):

- Assesses the requirements from a user-centric perspective.
- Ensures that the user interface design supports a seamless user experience, especially with multi-language support.
- Looks for potential usability issues and suggests improvements.

Database Administrator (DBA) (Reader):

- Evaluates the database-related requirements, especially those related to customer profiles, transactions, and integration with external systems.
- Ensures that the database design is in alignment with the requirements.

Security Expert (Reader):

- Focuses on ensuring that the requirements address security concerns.
- Checks for user authentication and authorization, data protection, and compliance with data privacy regulations.
- Identifies potential security vulnerabilities.

Project Manager(Recorder):

- Provides oversight and ensures that the inspection process remains on schedule.
- Tracks issues and action items.
- Ensures that the inspection results are communicated and addressed in the project plan.

b. Follow the Software Inspections procedure:

1. **Preparation**: Assemble the inspection team; Distribute the document or artifact to be inspected.
2. **Overview Meeting**: Kickoff meeting to introduce the inspection process; Define roles and responsibilities; Set a schedule for the inspection.
3. **Individual Review:** Team members independently review the document; Identify issues, ambiguities, inconsistencies, and questions;
4. **Inspection Meeting:** Conduct a structured meeting led by a moderator; Present the document to the team; Discuss the document, findings, and concerns; Identify issues and make suggestions for improvements.
5. **Recording Findings:** Document all identified issues and suggestions; Categorize issues (e.g., defects, ambiguities); Note severity and impact, if relevant.
6. **Author/ Moderator Response:** The author reviews and responds to identified issues; Provides clarifications or corrections.
7. **Follow-Up Meeting (Optional**): Additional meeting for clarification
8. **Final Report:** Compile findings and responses into a final inspection report; Include a summary of the meeting, issues, and resolutions.
9. **Closure:** Consider the inspection closed once all issues are addressed.
10. **Action Items/Tickets/Tasks:** Assign action items for implementation or further investigation.
11. **Documentation and Reporting:** Document the process and outcomes for reference and audit purposes.

c. Use a written checklist of what should be considered (Requirements Checklist)

Requirements Checks

RQ1	Does each requirement relate back? • To the problem or opportunity statement • To a scenario or business process This is to ensure each requirement is needed
RQ2	Is each requirement tagged with a Unique Identifier reference?
RQ3	Is the requirement implementation-neutral? • Defines what functionality is required but not how • Allows the architect or system developer to decide what technology/method is best suited to achieve the functionality
RQ4	Is the requirement stated clearly and concisely? • Consists of a single requirement • No more than 30-50 words in length • Easily read and understood by non-technical people • Unambiguous and not susceptible to multiple interpretations • Does not contain definitions, descriptions of its use or reasons for its need • Avoids subjective or open-ended terms
RQ5	Is each requirement stated in precise, verifiable and in measurable terms? • Stated in such a way that is can be tested by inspection, analysis or demonstration • Make it possible to evaluate whether the solution met the requirement • Is it free of weak words (like the following): o Appropriate o Efficient o Powerful o Fast o Easy o Effective o Reliable o Compatible o Normal o User friendly o Before o After o Quickly o Timely o Strengthen o Enhance o Minimize o Maximize o Rapid o Sufficient o Adequate o Support o But not limited to
RQ6	Has the requirement been stated in active voice? • Has passive voice (shall be) been avoided?

RQ7	Does the requirement state what the system shall do, rather than what it shall not do? • If "shall not" has been used, is the use of the negative justified (for safety, etc.), and have double negatives been avoided? • If there are requirements that are not currently required but for a possible future phase these must be added to an appendix instead and not be part of the main requirements section
RQ8	Is each requirement complete? • Contains all the information that is needed to define the requirement • Leaves no one guessing (For how long?, 50% of what?) • Includes measurement units if necessary
RQ9	Can an objective test be written for the requirement? • Are both a test method and a test case evident in the wording of the requirement? • Are all necessary reaction windows or other tolerances stated in the requirement?
RQ10	Are the requirements consistent? • Does not conflict with other requirements in the requirement specification • Uses the same terminology throughout the requirement specification • Does not duplicate other requirements
RQ11	Is each requirement viable? • For example: • Can be met using existing technology • Can be achieved within budget • Can be met within schedule • Is something the organisation has the necessary skills to utilise • Will be used by end users • Helpful to build the solution
RQ12	Are the requirement's preconditions and triggers clearly defined within the requirement?
RQ13	Have exception scenarios been explored in the requirements?
RQ14	Has the rationale for the requirement been clearly stated? • Are there any associated requirements that might affect interpretation of this requirement and should therefore be referenced in the rationale statement? • If no rationale statement has been included, is the rationale obvious in the requirement statement or from associated directives or references?
RQ15	Has a Business value, source and rationale populated for each requirement?
RQ16	There is a mixture of requirements priorities with no more than 60% of them being ranked high.

Another example for Requirements Checklist:

- ✓ Are the requirements unambiguous and free from conflicting statements?
- ✓ Is each requirement specific, clear, and understandable to stakeholders?
- ✓ Have the requirements been prioritized to reflect their importance?
- ✓ Is there a defined scope for the project or system covered by the requirements?

- ✓ Do the requirements address the needs and objectives of the project stakeholders?
- ✓ Have all relevant stakeholders provided input and approved the requirements?
- ✓ Are the requirements traceable, with unique identifiers and links to source documents?
- ✓ Are the requirements testable, with clear and measurable criteria for validation?
- ✓ Have non-functional requirements (e.g., performance, security) been clearly specified?
- ✓ Are the requirements consistent with industry standards and best practices?
- ✓ Do the requirements account for external interfaces and dependencies?
- ✓ Are there clear constraints, assumptions, and limitations documented for the requirements?
- ✓ Have any risks or potential issues associated with the requirements been identified?
- ✓ Do the requirements include both functional and non-functional aspects of the system?
- ✓ Is there a mechanism to handle changes or updates to the requirements?
- ✓ Are the requirements free from unnecessary technical jargon or acronyms?
- ✓ Have the requirements been reviewed and validated by relevant experts or stakeholders?
- ✓ Is there clarity on the priority of each requirement (e.g., must-have, should-have, nice-to-have)?
- ✓ Are the requirements consistent with the project's overall goals and objectives?
- ✓ Are there clear acceptance criteria for each requirement, defining what success looks like?
- ✓ Do the requirements align with the project's budget, timeline, and resource constraints?
- ✓ Is the language used in the requirements document free from ambiguity and open to interpretation?
- ✓ Are there clear dependencies and relationships between requirements documented?
- ✓ Have potential conflicts between requirements been resolved or addressed?
- ✓ Do the requirements consider future scalability and extensibility needs?

2. Self-Code Review:
    a. Implement the calculation of the price according to dynamic pricing rules: IntelliJ Project
    b. Create your own code review checklist:

Code Review Checklist:

1. Readability and Maintainability:
   - ✓ Code is well-organized and follows a consistent coding style and naming conventions.
   - ✓ Comments are clear and explain complex or non-obvious code.
   - ✓ Code is self-explanatory and does not rely heavily on comments.
   - ✓ Magic numbers and strings are avoided or properly explained.
2. Code Structure:
   - ✓ Code is modular and follows the Single Responsibility Principle (SRP).
   - ✓ Functions and methods are appropriately sized and do one specific task.
   - ✓ Code is organized into logical modules, classes, or packages.
   - ✓ There are no large monolithic functions or classes.
3. Testing:
   - Code is accompanied by unit tests covering critical functionality.
   - Test coverage is adequate, and tests are well-structured.
   - Tests include positive and negative scenarios, boundary cases, and edge cases.
4. Performance:
   - ✓ Code is optimized for performance where it matters.
   - ✓ Resource-intensive operations are optimized.
   - ✓ No unnecessary database queries, loops, or data transfers.
5. Documentation:
   - ✓ Code is well-documented, including inline comments and explanations.
   - ✓ External dependencies, libraries, and APIs are properly documented.
   - ✓ API endpoints are documented for external use.
6. Version Control:
   - ✓ The code follows the version control guidelines and includes appropriate version control comments.
   - ✓ Commit messages are clear, concise, and meaningful.
7. Dependencies and Third-party Libraries:
   - ✓ Dependencies are managed using package managers or dependency management tools.
   - ✓ Licensing and usage of third-party libraries are compliant with project requirements.
8. Code Smells:
   - ✓ No code smells such as duplicated code, excessive nesting, or long methods.
   - ✓ Code adheres to best practices and design patterns.

c. Execute self-code review: