Quality Audit Evidence Document (Refactored Version)

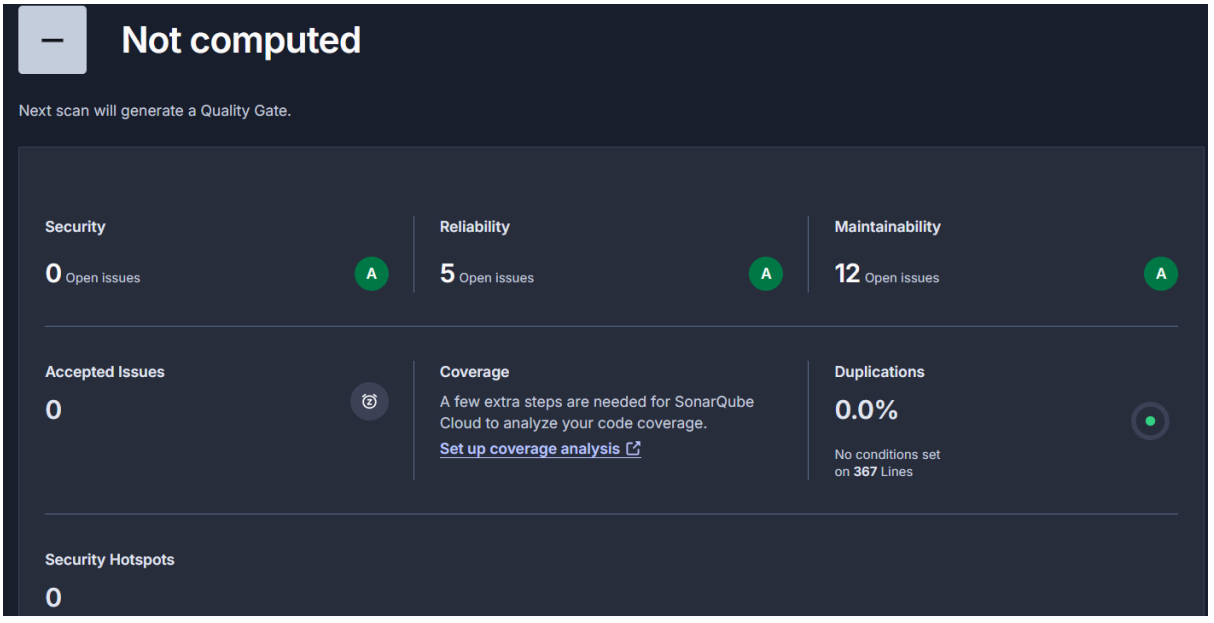
Subject and Topic

- Subject: Software Architecture and Design
- Melanny Guate
- Date: November 11, 2025
- Topic: Post-Refactoring Quality Audit using GitHub and SonarCloud.
- Class Objective: Document the quality audit of the optimized (Refactored) version and the remaining outstanding issues.

Step 1: Audit Connection and Execution Evidence (Dashboard)

- Instruction: Paste the screenshot of the main Dashboard (Summary) for the Refactored version, confirming that the SonarCloud analysis ran correctly.

1. Refactored Version



Step 2: Quality Analysis of the Refactored Version

- Instruction: Document the Quality Gate status, classifications (A, B, C, D), the number of Issues for key metrics, and the duplication percentage, using the information from the SonarCloud Summary section.

1. Metrics Audit

- Note: The Refactored version presents 17 open issues in total, mainly related to Maintainability.

Métrica / Atributo	Versión Refactorizada
Quality Gate	[Registrar el estado: Pasó/Falló]
Reliability (Fiabilidad)	Clasificación: **A** / Issues Abiertos: **5**
Maintainability (Mantenibilidad)	Clasificación: **A** / Issues Abiertos: **12**
Security (Seguridad)	Clasificación: **A** / Issues Abiertos: **0**
Duplicación	Porcentaje: **0.0%**

- 2. Summary of Issues (Issues and Security Hotspots)
- Instruction: Document the total number of issues found in the Issues section and classify them by severity.

Refactored Version

- Total Number of Issues: 17 (Estimated: 5 Reliability + 12 Maintainability)
- Bugs (Estimated by Reliability): Total Reliability: 5
- Vulnerabilities (Estimated by Security): 0
- Code Smells (Estimated by Maintainability): 12
- Security Hotspots: 0

Step 3: Documentation of Outstanding Priority Findings

- Instruction: Document the 5 most significant issues that still remain open in the Refactored version (those contributing to the 17 total issues) to plan their future correction.

Priority Finding 1: Unexpected Lexical Declaration in `switch`

- Issue Version: Refactored
- Type of Issue: Code Smell (Major - Maintainability)

- Screenshot of the Issue (SonarCloud):
  [Include a screenshot of the Code Smell `Unexpected lexical declaration in case block.`]
- Issue Description and Impact: A variable declaration using `let` or `const` was found directly inside a `case` block of a `switch` statement. This can lead to variable scope errors, as the declaration is not contained within its own code block (`{}`). The impact is on Maintainability and Reliability, as code behavior may be unexpected or lead to difficult-to-debug logic errors.
- Future Correction Plan: [Briefly describe how this issue is planned to be corrected in the next development iteration. (Solution: Enclose the variable declarations within a block of curly braces `{}` in the corresponding `case`).]

Priority Finding 2: Minimal Text Contrast (Accessibility)

- Issue Version: Refactored
- Type of Issue: Code Smell (Major - Maintainability)
- Screenshot of the Issue (SonarCloud):
  [Include a screenshot of the Code Smell `Text does not meet the minimal contrast requirement with its background.`]
- Issue Description and Impact: The CSS code uses text and background color combinations that do not meet the minimum accessibility contrast requirements (WCAG 2.1). This directly affects Usability for users with visual impairments or for any user in bright conditions. Although it is a Maintainability *Code Smell*, its real impact is on Accessibility and user experience.
- Future Correction Plan: [Briefly describe how this issue is planned to be corrected in the next development iteration. (Solution: Adjust the hexadecimal color values of the text or background to ensure the contrast ratio is at least 4.5:1).]

Priority Finding 3: Use of Obsolete Function `parseInt()`

- Issue Version: Refactored
- Type of Issue: Code Smell (Minor - Reliability and Maintainability)
- Screenshot of the Issue (SonarCloud):
  [Include a screenshot of the Code Smell `Prefer 'Number.parseInt()' over 'parseInt()'.`]
- Issue Description and Impact: The code is using the global `parseInt()` function. Although still functional, the modern ECMAScript standard (ES2015) promotes the use of `Number.parseInt()`. Using the global version can cause context confusion and, in certain optimizations, may have slightly inferior performance. The impact is on Maintainability and compliance with modern coding conventions.
- Future Correction Plan: [Briefly describe how this issue is planned to be corrected in the next development iteration. (Solution: Replace all instances of `parseInt()` with the static function `Number.parseInt()`).]

Priority Finding 4: Use of Obsolete Function `parseFloat()`

- Issue Version: Refactored
- Type of Issue: Code Smell (Minor - Reliability and Maintainability)
- Screenshot of the Issue (SonarCloud):
  [Include a screenshot of the Code Smell `Prefer 'Number.parseFloat()' over 'parseFloat()'.`]
- Issue Description and Impact: Similar to Finding 3, the global `parseFloat()` function is being used, which is considered obsolete by the ES2015 convention in favor of `Number.parseFloat()`. The impact is on Maintainability and code uniformity, as using global functions without their object prefix can be less clear about their origin.
- Future Correction Plan: [Briefly describe how this issue is planned to be corrected in the next development iteration. (Solution: Replace all instances of `parseFloat()` with the static function `Number.parseFloat()`).]

Priority Finding 5: Outstanding Reliability Bug (Estimation)

- Issue Version: Refactored
- Type of Issue: Bug (Medium - Reliability)
- Screenshot of the Issue (SonarCloud):
  [Include a screenshot of one of the remaining 5 Reliability Bugs (e.g., a potentially null assignment or a loop error).]
- Issue Description and Impact: Since the Dashboard indicates 5 Reliability Issues, this finding represents one of the remaining Bugs that could cause unexpected behavior. For instance, it could be a missing validation in a loop (`while` or `for`) or an incorrect type comparison. The main impact is on Reliability, as, even though the code was refactored, this specific Bug still has the potential to fail in production.
- Future Correction Plan: [Briefly describe how this issue is planned to be corrected in the next development iteration. (Solution: Review the logic of the reported line and apply the *fail-fast principle* or ensure correct variable initialization to eliminate the possibility of runtime failure).]