Code Book - AXIS2 Architecture Issues

Preface

This enhanced codebook provides a detailed framework for categorizing and analyzing issues based on their architectural significance, existence, property, executive nature, and associated quality attributes. The codebook includes expanded definitions, coding rules, and visual aids to ensure consistency and clarity in issue evaluation and classification.

Variable Definitions

1. Issue Key: A unique identifier for each issue (e.g., AXIS2-881).
2. Title: A short description summarizing the issue.
3. IsArchitecture: A binary indicator (1 = Architectural, 0 = Not Architectural) indicating if the issue impacts system architecture significantly.
4. Existence: A binary indicator (1 = Existence, 0 = Not Existence) to denote if the issue involves substantial changes, such as adding/modifying critical system components.
5. Property: A binary indicator (1 = Property, 0 = Not Property) indicating if the issue focuses on quality attributes like performance, maintainability, or scalability.
6. Executive: A binary indicator (1 = Executive, 0 = Not Executive) for issues related to dependencies, external tools, or utility development.
7. Quality Attributes: Lists associated attributes such as Performance, Scalability, etc., that define the quality focus of the issue.
8. Justification: A detailed explanation supporting the classification of the issue, based on the criteria above.

Coding Guidelines

Code Book - AXIS2 Architecture Issues

1. **IsArchitecture**: Issues are classified as architectural if they impact the structure or design of the system, influencing significant components or interactions. Use the decision flowchart (Appendix A) to determine if an issue qualifies as architectural.

2. **Existence**: Mark this attribute if the issue involves major structural changes such as adding/modifying system components. Examples include new feature implementations or significant refactoring efforts.

3. **Property**: Issues marked as property focus on enhancing or addressing quality attributes such as:

   - Performance improvements, security enhancements, or scalability modifications.
   - Refer to the attribute dependency table (Appendix B) to cross-check if property is applicable.

4. **Executive**: An issue is marked as executive if it involves dependency management, external tool support, or utility tool development. For edge cases, consult the executive guidelines checklist (Appendix C).

Examples and Edge Cases

1. **Issue Key**: AXIS2-881
   - **Title**: JAXWS: Refactoring invocation APIs
   - **IsArchitecture**: 1 (Architectural)
   - **Existence**: 0 (Not an Existence issue)
   - **Property**: 1 (Addresses maintainability and modifiability)
   - **Executive**: 0 (No external technology involved)

Code Book - AXIS2 Architecture Issues

- **Justification**: The refactoring reorganizes core invocation APIs to enhance system maintainability without adding new components.

2. **Issue Key**: AXIS2-921
   - **Title**: Add system level initialization/cleanup hooks
   - **IsArchitecture**: 1 (Architectural)
   - **Existence**: 1 (Involves adding significant system-level mechanisms)
   - **Property**: 0 (No specific quality attribute enhancement)
   - **Executive**: 0 (No dependency on external technology)
   - **Justification**: Introduces a new system-level mechanism, impacting the system's architecture layer directly. It qualifies as an existence issue since it adds major components at a system level.

3. **Edge Case**: A minor refactoring task
   - This issue does not add or significantly modify components; therefore, it should not be marked as existence. However, if the refactoring task aims to enhance performance (a quality attribute), it may be considered under property.

Glossary of Terms

1. **Modifiability**: The ease with which a system can accommodate changes without significant rework.
2. **Scalability**: The ability of the system to handle increased loads by adding resources.
3. **Quality Attributes**: Characteristics that define the overall performance and behavior of the system, such as reliability, usability, and security.
4. **Refactoring**: Reorganizing code to improve its structure and readability without altering its

functionality.

5. **Dependencies**: External components, libraries, or tools that the system relies on for certain functionalities.