FeedService.java

Path: reader-core\src\main\java\com\sismics\reader\core\service\FeedService.java

---

Problems in the Code

1. **Insufficient Modularization**

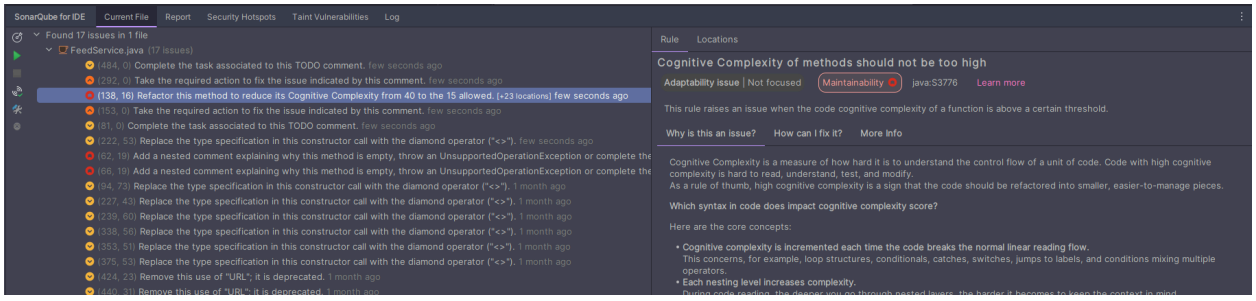
    - High **Cognitive Complexity** due to deeply nested conditionals, loops, and mixed operators in conditions.
    - Large functions like `synchronize()` handle multiple responsibilities, making them difficult to read and maintain.
    - Hard to test and modify due to the complexity of flow control and deep nesting.

Impact

1. Insufficient Modularization

- Large methods with deep nesting make the code harder to understand and debug.
- Difficult to test and maintain as changes in one part of the method may break other functionalities.
- Harder to extend as the code is tightly coupled within complex functions.

Analysis by Sonarqube



SubscriptionImportAsyncListener.java

Path:
`reader-core/src/main/java/com/sismics/reader/core/listener/async/SubscriptionImportAsyncListener.java`

Problems in the Code

1. **Feature Envy**
    - **Issue**:
      Several methods in this class directly manipulate the internal data or logic of other classes like `Feed`, `Article`, `UserArticle`, and `JobEventDao`. These methods are overly concerned with the behavior and structure of external entities, indicating misplaced responsibilities.
    - **Example**:
        - `importFeedFromStarred` performs tasks like synchronizing feeds (`FeedService`), checking or creating articles (`ArticleDao`), and subscribing users to articles (`UserArticleDao`).
        - `processImportFile` combines logic for file handling, feed extraction, and user article subscription.
    - **Impact**:
        - High coupling with external classes (e.g., `FeedService`, `ArticleDao`).
        - Violates the Single Responsibility Principle (SRP) by bundling responsibilities that should belong to other classes.
        - Reduces maintainability, as changes in external class logic may require updates to this class.

---

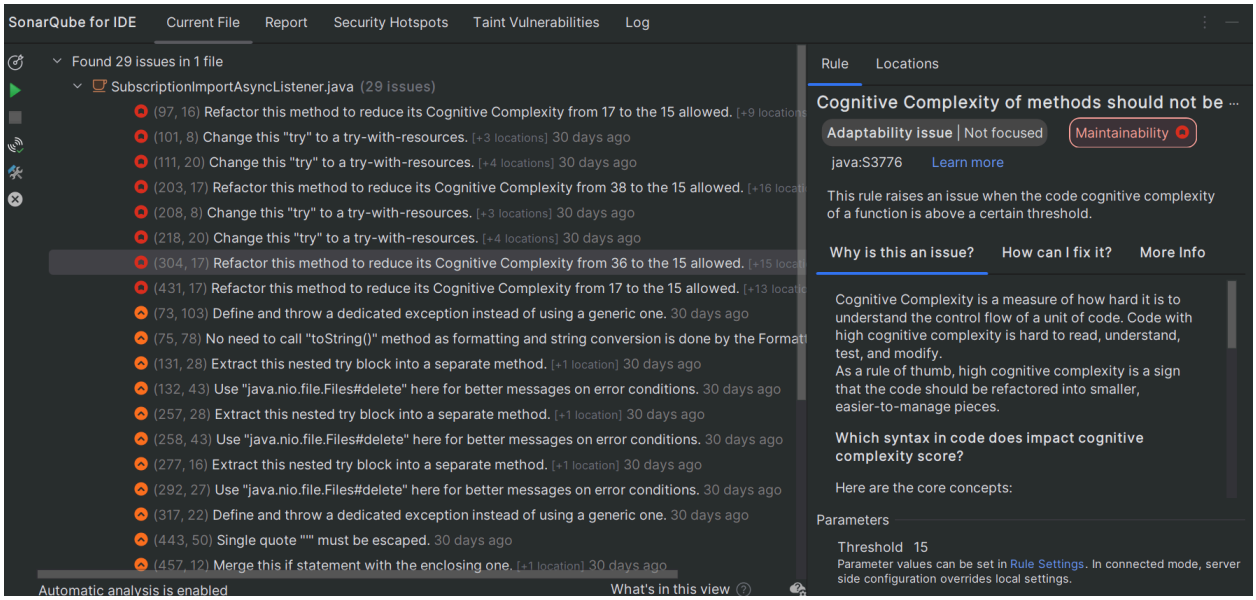
2. **Hub-like Modularization**
    - **Issue**:
      This class serves as a central hub for multiple unrelated responsibilities such as file handling, job creation, feed synchronization, and article subscriptions. It directly interacts with several classes and manages their operations.
    - **Example**:
        - `processImportFile` handles ZIP extraction, OPML parsing, starred article imports, and feed subscriptions all within the same method.
        - `createJob` combines logic for analyzing file contents, determining feed/article counts, and managing database interactions for job creation.
    - **Impact**:

        - High complexity due to tightly coupled and centralized logic.
        - Difficult to test or extend individual functionalities without affecting the entire class.
        - Increases the risk of breaking unrelated features when making changes.

---

3. **Multifaceted Abstraction**
    - **Issue**:
      The class handles multiple unrelated concerns, such as processing ZIP and OPML files, managing database transactions, and logging job events, leading to a violation of cohesive design principles.
    - **Example**:
        - `createJob` is responsible for:
            1. Guessing file types.
            2. Extracting outlines from OPML files.
            3. Creating database job entries and events.
        - `importOutline` combines responsibilities like managing categories, checking feed subscriptions, and synchronizing feeds.
    - **Impact**:
        - Decreases code readability and maintainability.
        - Makes the class harder to understand, as it lacks a single, well-defined responsibility.
        - Changes to one part of the functionality (e.g., file handling) may inadvertently affect unrelated logic (e.g., database operations).

Analysis by SonarQube


SonarQube for IDE
Current File
Report
Security Hotspots
Taint Vulnerabilities
Log
Found 29 issues in 1 file
SubscriptionImportAsyncListener.java (29 issues)
(97, 16) Refactor this method to reduce its Cognitive Complexity from 17 to the 15 allowed. [+9 locations]
(101, 8) Change this "try" to a try-with-resources. [+3 locations] 30 days ago
(111, 20) Change this "try" to a try-with-resources. [+4 locations] 30 days ago
(203, 17) Refactor this method to reduce its Cognitive Complexity from 38 to the 15 allowed. [+16 locati
(208, 8) Change this "try" to a try-with-resources. [+3 locations] 30 days ago
(218, 20) Change this "try" to a try-with-resources. [+4 locations] 30 days ago
(304, 17) Refactor this method to reduce its Cognitive Complexity from 36 to the 15 allowed. [+15 locati
(431, 17) Refactor this method to reduce its Cognitive Complexity from 17 to the 15 allowed. [+13 locati
(73, 103) Define and throw a dedicated exception instead of using a generic one. 30 days ago
(75, 78) No need to call "toString()" method as formatting and string conversion is done by the Formatt
(131, 28) Extract this nested try block into a separate method. [+1 location] 30 days ago
(132, 43) Use "java.nio.file.Files#delete" here for better messages on error conditions. 30 days ago
(257, 28) Extract this nested try block into a separate method. [+1 location] 30 days ago
(258, 43) Use "java.nio.file.Files#delete" here for better messages on error conditions. 30 days ago
(277, 16) Extract this nested try block into a separate method. [+1 location] 30 days ago
(292, 27) Use "java.nio.file.Files#delete" here for better messages on error conditions. 30 days ago
(317, 22) Define and throw a dedicated exception instead of using a generic one. 30 days ago
(443, 50) Single quote "'" must be escaped. 30 days ago
(457, 12) Merge this if statement with the enclosing one. [+1 location] 30 days ago
Automatic analysis is enabled
What's in this view
Rule
Locations
Cognitive Complexity of methods should not be ...
Adaptability issue | Not focused
Maintainability
java:S3776
Learn more
This rule raises an issue when the code cognitive complexity of a function is above a certain threshold.
Why is this an issue?
How can I fix it?
More Info
Cognitive Complexity is a measure of how hard it is to understand the control flow of a unit of code. Code with high cognitive complexity is hard to read, understand, test, and modify.
As a rule of thumb, high cognitive complexity is a sign that the code should be refactored into smaller, easier-to-manage pieces.
Which syntax in code does impact cognitive complexity score?
Here are the core concepts:
Parameters
Threshold 15
Parameter values can be set in Rule Settings. In connected mode, server side configuration overrides local settings.

UserResource.java

Path-> reader-web/src/main/java/com/sismics/reader/rest/resource/UserResource.java

Problems in the Code

1. Cyclically Dependent Modularization

The `UserResource` class suffers from **cyclic dependencies** because it directly interacts with multiple components, creating **tightly coupled modules**:

- **Direct dependency on `UserDao`, `CategoryDao`, and `AuthenticationTokenDao`**
  - The class not only invokes methods but also instantiates DAOs directly.
  - This **violates dependency inversion** as the REST layer should not be responsible for persistence.
- **Event Handling (`UserCreatedEvent`, `PasswordChangedEvent`)**
  - Business logic and event publishing are mixed, making it difficult to isolate concerns.
  - The class raises events **without a dedicated event-handling layer**, leading to **spaghetti code**.
- **Validation using `ValidationUtil` in multiple methods**
  - Instead of delegating to a **separate validation layer**, validation logic is scattered throughout.

Consequences

- **Hard to test:** The class depends on multiple modules, making unit testing difficult.
- **Rigid design:** Changing one module (e.g., DAO implementation) **requires modifying this class**.
- **Code duplication:** Validation logic and database interactions are repeated across methods.

---

2. Broken Modularization

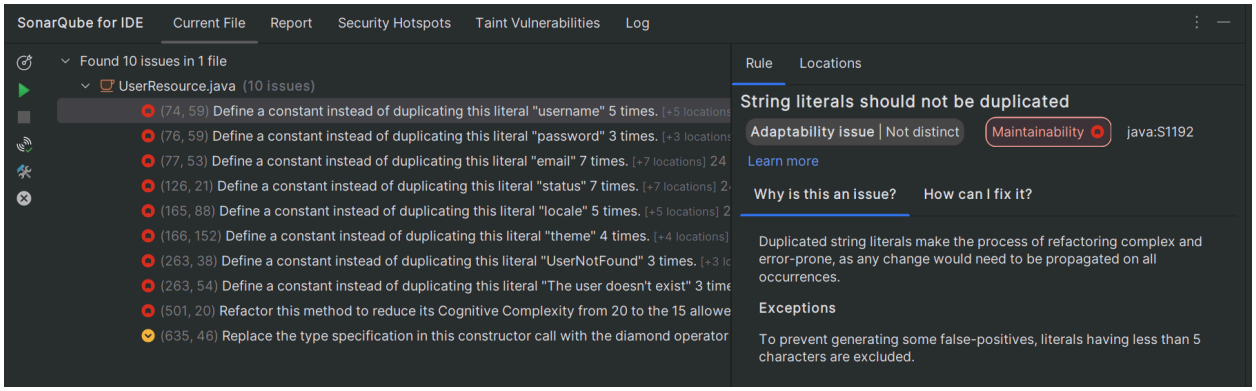
The `UserResource` class violates **Single Responsibility Principle (SRP)** by handling multiple concerns:

- **REST API Handling:** Defines endpoints and processes HTTP requests/responses.
- **Validation:** Performs input checks using `ValidationUtil`.
- **Business Logic:** Updates user attributes and sets default values.
- **Persistence:** Interacts directly with the DAO layer to create, update, and delete users.
- **Event Publishing:** Fires `UserCreatedEvent` and `PasswordChangedEvent` within the same methods.

Impact

- **Low maintainability:** Code modifications become riskier due to interdependent functionalities.
- **Difficult scalability:** Extending features (e.g., adding a user notification system) **requires modifying this monolithic class**.
- **Poor separation of concerns:** Business logic should be in **services**, not in the resource class.

Analysis By Sonarqube



SubscriptionResource.java

Path-> reader-web/src/main/java/com/sismics/reader/rest/resource/SubscriptionResource.java

Problems in the Code

1. **Insufficient Modularization**

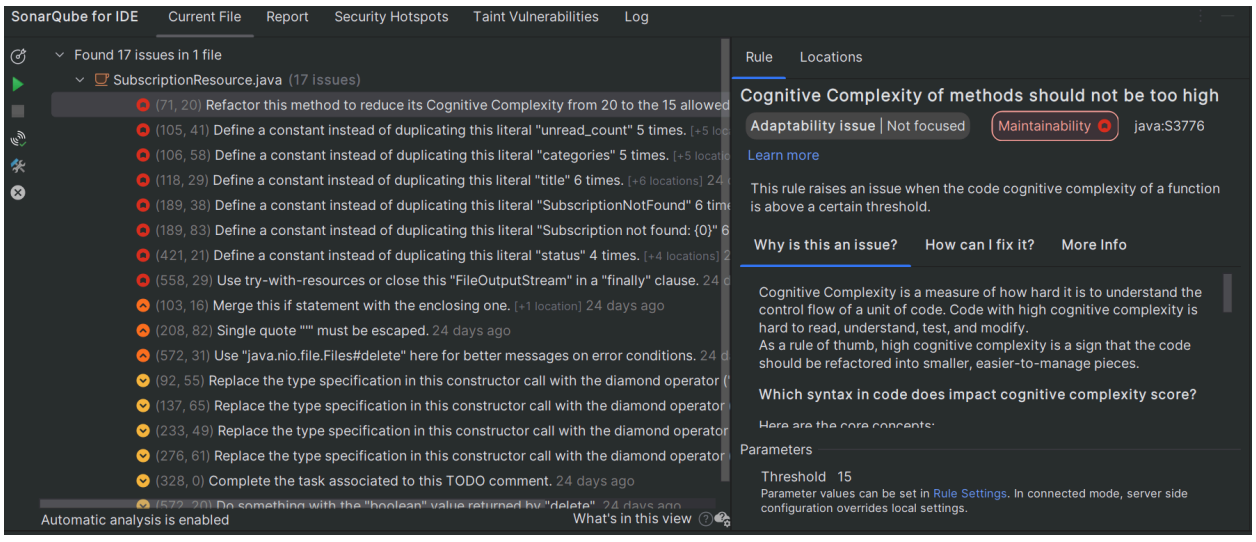
    - High **Cognitive Complexity** due to deeply nested conditionals, loops, and mixed operators in conditions.
    - Large functions like `list` and `get` handle multiple responsibilities, making them difficult to read and maintain.
    - Hard to test and modify due to the complexity of flow control and deep nesting.

Impact

1. Insufficient Modularization

- Large methods with deep nesting make the code harder to understand and debug.
- Difficult to test and maintain as changes in one part of the method may break other functionalities.
- Harder to extend as the code is tightly coupled within complex functions.

Analysis by Sonarqube



ArticleDao.java

Path->reader-core/src/main/java/com/sismics/reader/core/dao/jpa/ArticleDao.java

Problems in the Code

1. **Duplicate Abstraction**
    - **Repeated query strings** are used across multiple methods:
        - `select a from Article a where a.deleteDate is null` in `findAll`.
        - `update T_ARTICLE set ...` in `update`.
        - `update T_ARTICLE set ART_DELETEDATE_D = :deleteDate` in `delete`.
    - This redundancy increases maintenance overhead and makes the code harder to refactor.
    - No centralized management of query logic to enforce consistency.

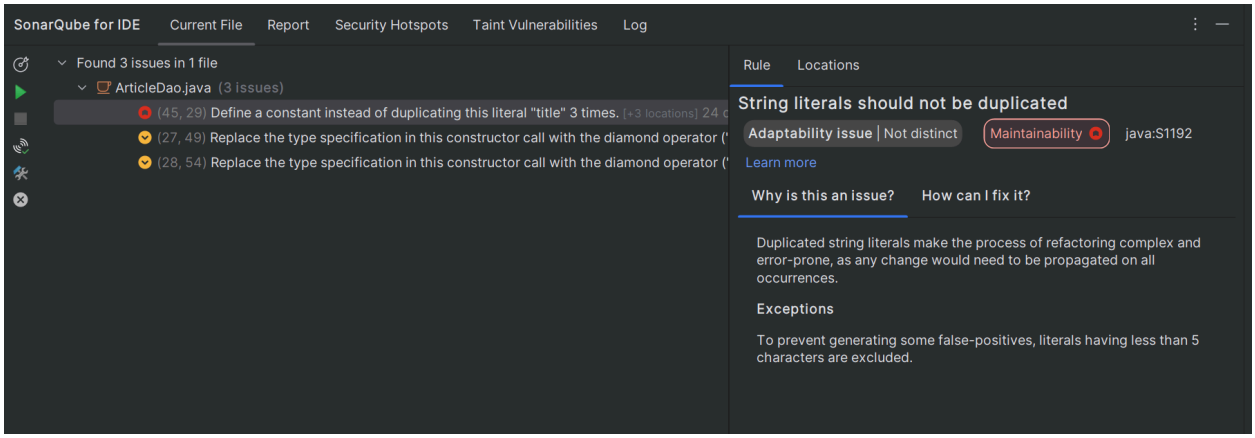
Impact

-

2. Duplicate Abstraction

- Maintaining and refactoring the code becomes difficult due to repetitive query strings.
- A single change in query logic requires multiple updates, increasing the risk of bugs.
- Reduces code reusability and violates the **DRY (Don't Repeat Yourself) principle**.

Analysis by Sonarqube


SonarQube for IDE
Current File
Report
Security Hotspots
Taint Vulnerabilities
Log
Found 3 issues in 1 file
ArticleDao.java (3 issues)
(45, 29) Define a constant instead of duplicating this literal "title" 3 times. [+3 locations] 24
(27, 49) Replace the type specification in this constructor call with the diamond operator
(28, 54) Replace the type specification in this constructor call with the diamond operator
Rule
Locations
String literals should not be duplicated
Adaptability issue | Not distinct
Maintainability
java:S1192
Learn more
Why is this an issue?
How can I fix it?
Duplicated string literals make the process of refactoring complex and error-prone, as any change would need to be propagated on all occurrences.
Exceptions
To prevent generating some false-positives, literals having less than 5 characters are excluded.

FeedSubscriptionDao.java

Path-> reader-core/src/main/java/com/sismics/reader/core/dao/jpa/FeedSubscriptionDao.java

Problems in the Code

1. Duplicate Abstraction

- Dealing with repeated string constants like "categoryId" and "userId".
- Also using variables while creating query improves readability and maintainability.

Impact

- Difficult to maintain and refactor.
- High risk of bugs when changing query logic.
- Redundant code increases complexity and reduces reusability.

Analysis by Sonarqube