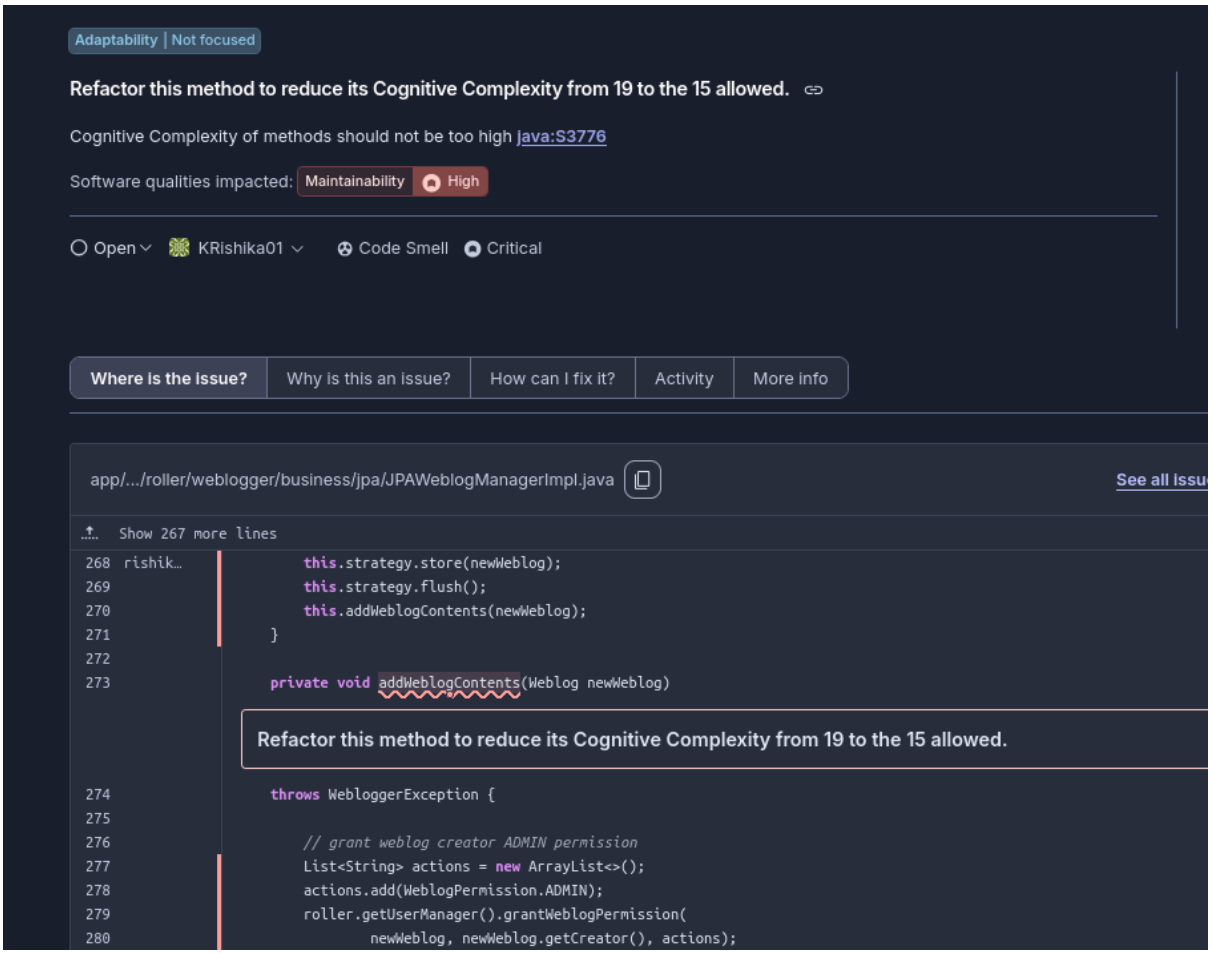
1. **JPAWeblogManagerImpl**
   Design Smell - **Feature Envy**

   A method has **Feature Envy** when it uses another class's data and methods more than its own class's data.

   SonarQube Reference:



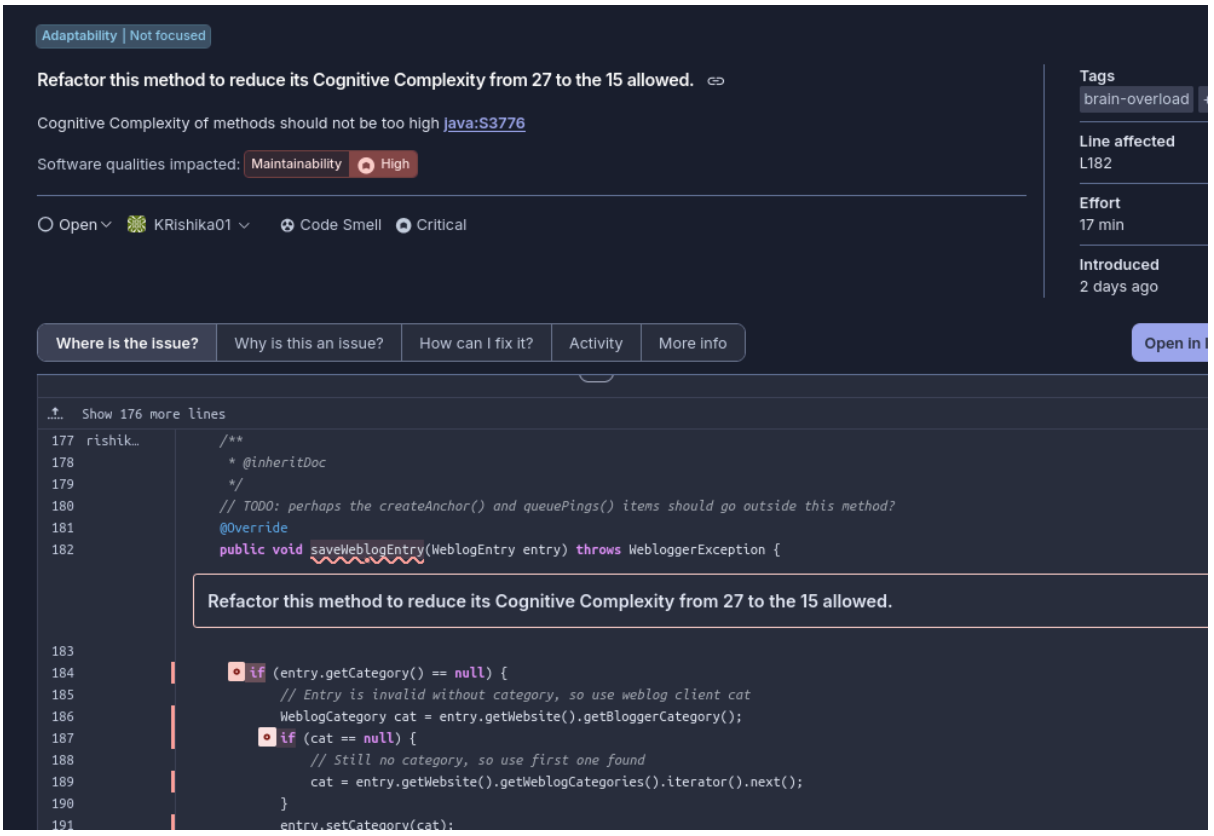
   High Cognitive Complexity means

- Handles too many cases
- Coordinates many external objects
- Knows too much about another class

   Designite Java Reference:
   The tool detected an instance of this smell because addWeblogContents is more interested in members of the type: Weblogger.

2. **JPAWeblogEntryManagerImpl**
   Design Smell - **Insufficient Modularization**

   SonarQube Reference



   - The method has too much logic in one place
   - Many branches, loops, or nested conditions
   - The method is handling multiple responsibilities

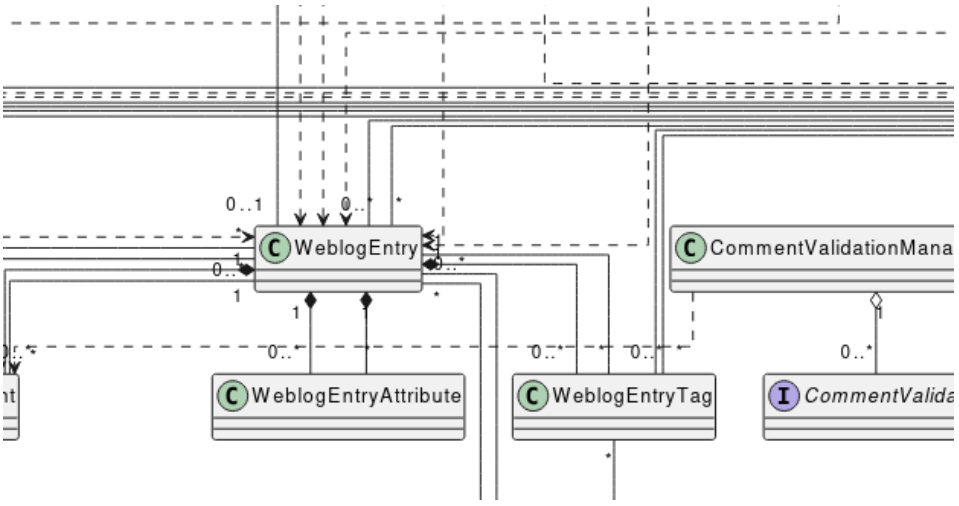
   Designite Java Reference:
   The tool detected the smell in this class because the class has a bloated interface (a large number of public methods). Total public methods in the class: 44 public methods

3. **WeblogEntry**
   Design smell - hub-like modularisation

   Designite reference:
   The tool detected the smell in this class because this class has high number of incoming as well as outgoing dependencies. Incoming dependecies: EntryEdit; Entries; EntryRemove; Comments; EntryBean; WeblogEntriesPermalinkPager; WeblogEntriesListPager;

WeblogEntriesPreviewPager; WeblogEntriesLatestPager; WeblogEntriesMonthPager; WeblogEntriesDayPager; WeblogTrackbackRequest; WeblogCommentRequest; WeblogPreviewRequest; WeblogPageRequest; SiteWideCache; URLModel; TrackbackServlet; PageServlet; CommentServlet; WeblogCalendarModel; BigWeblogCalendarModel; WeblogEntryManager; PluginManagerImpl; PluginManager; EncodePreTagsPlugin; WeblogEntryPlugin; ObfuscateEmailPlugin; ConvertLineBreaksPlugin; SmileysPlugin; IndexOperation; AddEntryOperation; LuceneIndexManager; RemoveEntryOperation; ReIndexEntryOperation; IndexManager; AutoPingManager; JPAWeblogEntryManagerImpl; JPAAutoPingManagerImpl; Weblog; WeblogEntryComment; WeblogEntryTag; CommentSearchCriteria; WeblogEntryAttribute; WeblogEntryWrapper; CacheManager; CacheHandler; MailUtil; Trackback; RollerAtomHandler; EntryCollection; BloggerAPIHandler; MetaWeblogAPIHandler. Outgoing dependencies: WeblogEntry.PubStatus; Weblog; WeblogCategory; User; WebloggerFactory; WeblogEntryAttribute; WebloggerException; WeblogEntryTag; Utilities; WebloggerRuntimeConfig; WeblogEntryManager; CommentSearchCriteria; DateUtil; GlobalPermission; WeblogPermission; UserManager; WeblogEntryPlugin; HTMLSanitizer; WeblogEntryComment; RollerConstants

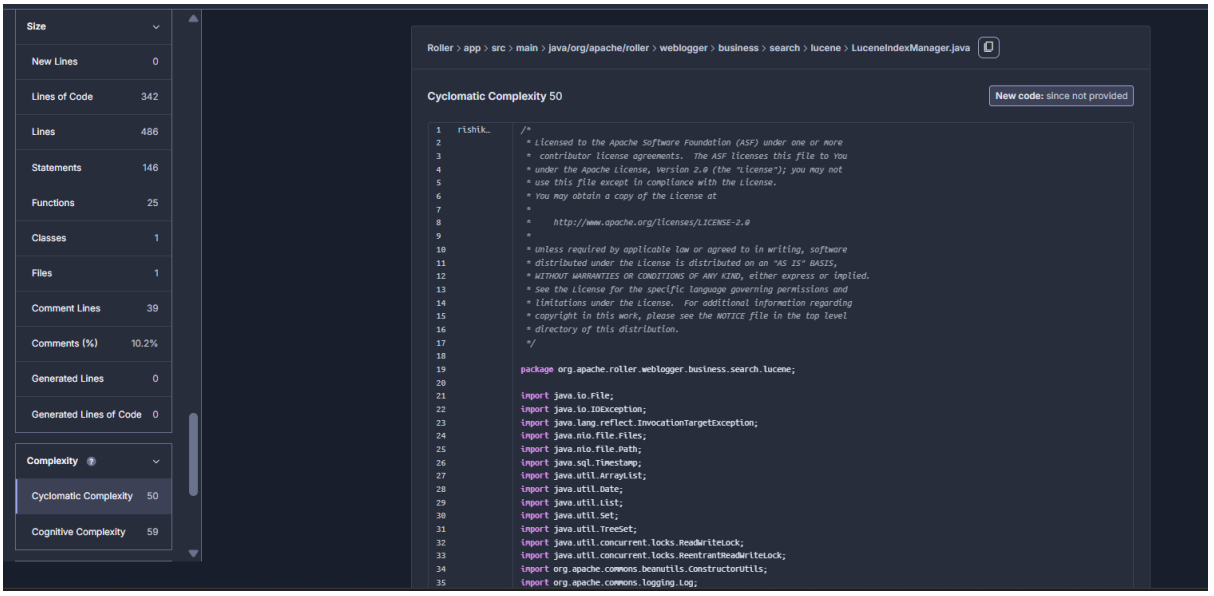
Uml reference:



- Acts as a **dependency hub** with very high fan-in and fan-out
- **Tightly coupled** to many unrelated modules

- Violates **single responsibility principle**
- Changes in this class can **impact many other classes**
- Difficult to **test, maintain, and evolve** independently
- Reduces overall **modularity and system stability**

4. **LuceneIndexManager**
   Design smell: Spaghetti Code
   Spaghetti code refers to source code with a convoluted, tangled, and unstructured control flow, making it difficult to understand, maintain, or extend.
   SonarQube Reference:



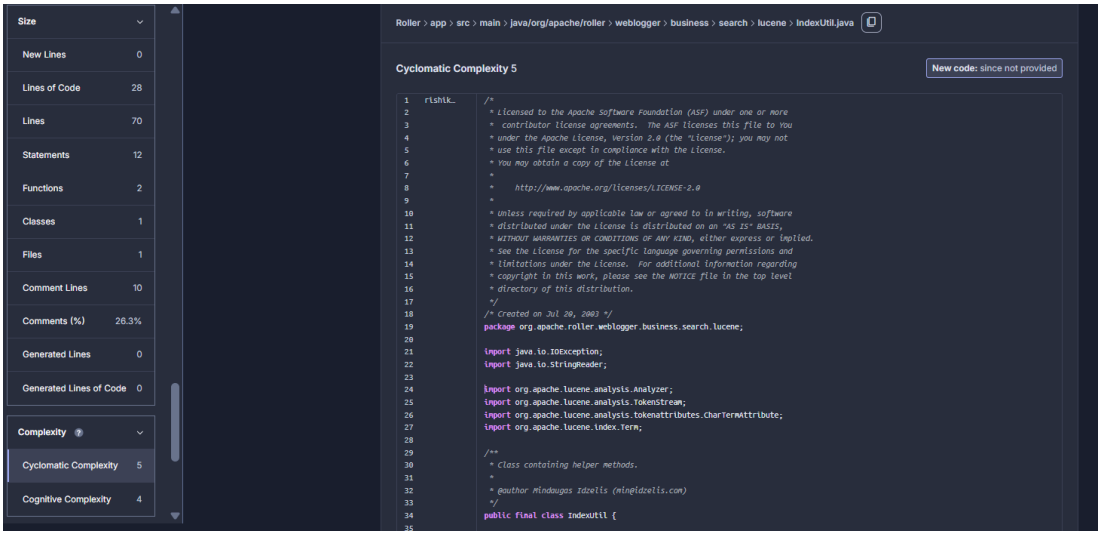
   From the picture we can see that,
   LOC: 342
   Cognitive complexity: 59
   Cyclomatic complexity: 50
   Indicators of Spaghetti code:
   1. **Very high cyclomatic complexity** (50)
      - Many branching paths hence tangled flow
   2. **Very high cognitive complexity** (59)
      - Nested conditionals mixed concerns, jumps in logic
   3. **High LOC in a single class/method**
      - 342 LOC → suggests long, monolithic logic blocks

5. **IndexUtil**

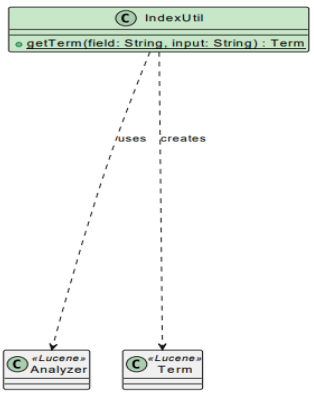
Design smell: Lazy class

It only has one static method that wraps Lucene's Term creation, which is minimal functionality that doesn't justify a separate class

SonarQube reference:



Justification:

1. **Single method** - It only has one public static method `getTerm()`
2. **Minimal responsibility** - Just wraps Lucene token analysis into Term creation
3. **No meaningful abstraction** - It's just a "helper" class with no real identity
4. **Unnecessary indirection** - The ~20 lines of logic could easily live in the classes that actually use it

UML justification: