

1. JPAWebblogManagerImpl

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:

The image shows a SonarQube interface with a dark theme. At the top, a status bar indicates 'Adaptability | Not focused'. Below this, a message says 'Refactor this method to reduce its Cognitive Complexity from 19 to the 15 allowed.' with a link to 'Java:S3776'. It also states 'Cognitive Complexity of methods should not be too high' and 'Software qualities impacted: Maintainability (High)'. There are filters for 'Open', 'KRishika01', 'Code Smell', and 'Critical'. A navigation bar includes 'Where is the Issue?', 'Why is this an Issue?', 'How can I fix it?', 'Activity', and 'More info'. The main area shows the file path 'app/.../roller/weblogger/business/jpa/JPAWebblogManagerImpl.java' and a 'See all Issues' link. A code editor displays lines 268-280 of the file. Lines 268-271 show a method call to 'this.strategy.store(newWeblog);', 'this.strategy.flush();', and 'this.addWebblogContents(newWeblog);'. Line 272 shows a closing brace. Line 273 shows the start of the 'addWebblogContents' method. A tooltip over the method name says 'Refactor this method to reduce its Cognitive Complexity from 19 to the 15 allowed.' Line 274 shows 'throws WebloggerException {'. Line 275 is empty. Line 276 shows a comment '// grant weblog creator ADMIN permission'. Line 277 shows 'List<String> actions = new ArrayList<>();'. Line 278 shows 'actions.add(WeblogPermission.ADMIN);'. Line 279 shows 'roller.getUserManager().grantWeblogPermission(''. Line 280 shows 'newWeblog, newWeblog.getCreator(), actions);'.

Adaptability | Not focused

Refactor this method to reduce its Cognitive Complexity from 19 to the 15 allowed. [Java:S3776](#)

Cognitive Complexity of methods should not be too high

Software qualities impacted: Maintainability **High**

☐ Open ☒ KRishika01 ☒ Code Smell ☒ Critical

Where is the Issue? Why is this an Issue? How can I fix it? Activity More info

app/.../roller/weblogger/business/jpa/JPAWebblogManagerImpl.java [See all Issues](#)

Show 267 more lines

```
268 rishik... this.strategy.store(newWeblog);
269 this.strategy.flush();
270 this.addWebblogContents(newWeblog);
271 }
272
273 private void addWebblogContents(Weblog newWeblog)
274
275
276 // grant weblog creator ADMIN permission
277 List<String> actions = new ArrayList<>();
278 actions.add(WeblogPermission.ADMIN);
279 roller.getUserManager().grantWeblogPermission(
280 newWeblog, newWeblog.getCreator(), actions);
```

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 `addWebblogContents` is more interested in members of the type: `Weblogger`.

2. JPAWeblogEntryManagerImpl

Design Smell - Insufficient Modularization

SonarQube Reference

The screenshot shows a SonarQube interface with a dark theme. At the top, a message states: "Refactor this method to reduce its Cognitive Complexity from 27 to the 15 allowed." Below this, it says "Cognitive Complexity of methods should not be too high" with a link to "java:S3776". The "Software qualities impacted" section shows "Maintainability" with a "High" severity. On the right, a sidebar lists "Tags" (brain-overload), "Line affected" (L182), "Effort" (17 min), and "Introduced" (2 days ago). Below the message bar, there are tabs: "Where is the Issue?", "Why is this an issue?", "How can I fix it?", "Activity", and "More info". The "Where is the Issue?" tab is active, showing a code editor with Java code. The code is for the `saveWeblogEntry` method in `JPAWeblogEntryManagerImpl`. The method has a cognitive complexity of 27, indicated by a red squiggly line under the method name. The code includes a Javadoc comment, a `@Override` annotation, and a `public void` signature. The method body contains several nested `if` statements and a `try` block. The first `if` statement checks if `entry.getCategory() == null`. The second `if` statement checks if `cat == null`. The code is highlighted with a red squiggly line under the method name.

- 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

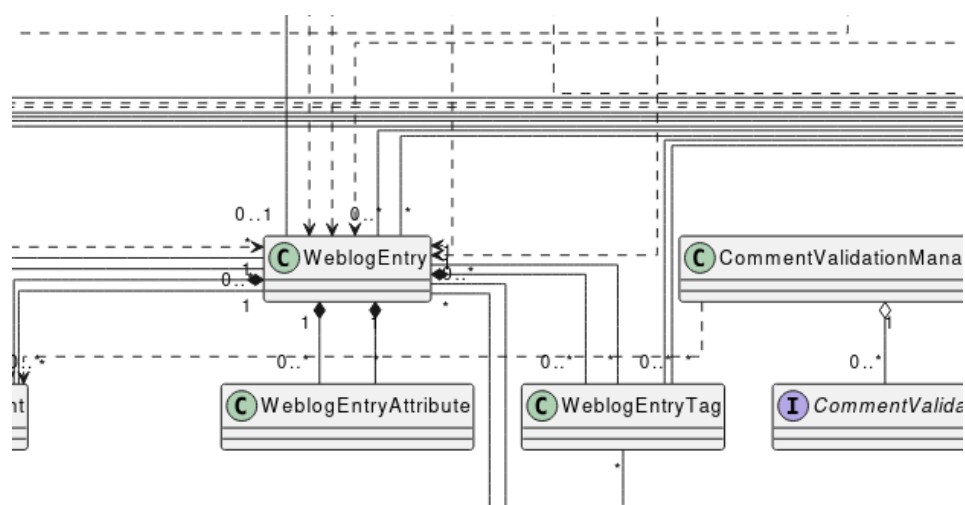
Design smell - hub-like modularisation

Designite reference:

The tool detected the smell in this class because this class has high number of incoming as well as outgoing dependencies. Incoming dependencies: `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:

The screenshot shows the SonarQube interface for the file `Roller > app > src > main > java/org/apache/roller > weblogger > business > search > lucene > LuceneIndexManager.java`. The left sidebar displays the following metrics:

Size	Value
New Lines	0
Lines of Code	342
Lines	486
Statements	146
Functions	25
Classes	1
Files	1
Comment Lines	39
Comments (%)	10.2%
Generated Lines	0
Generated Lines of Code	0

The 'Complexity' section shows:

- Cyclomatic Complexity: 50
- Cognitive Complexity: 59

The main area displays the source code of `LuceneIndexManager.java`. The cyclomatic complexity of 50 is highlighted in the top right corner of the code editor. The code includes a license header and various imports, followed by the class definition and methods.

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:

Size

▼

New Lines

0

Lines of Code

28

Lines

70

Statements

12

Functions

2

Classes

1

Files

1

Comment Lines

10

Comments (%)

28.3%

Generated Lines

0

Generated Lines of Code

0

Complexity

▼

Cyclomatic Complexity

5

Cognitive Complexity

4

Roller > app > src > main > java > org/apache/roller > weblogger > business > search > lucene > IndexUtil.java

Cyclomatic Complexity 5

New code: since not provided

```

1  r1$Hk...
2  /*
3   * Licensed to the Apache Software Foundation (ASF) under one or more
4   * contributor license agreements.  The ASF licenses this file to you
5   * under the Apache License, Version 2.0 (the "License"); you may not
6   * use this file except in compliance with the License.
7   * You may obtain a copy of the License at
8   *
9   *   http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.  For additional information regarding
16  * copyright in this work, please see the NOTICE file in the top level
17  * directory of this distribution.
18  */
19  /* Created on Jul 26, 2003 */
20  package org.apache.roller.weblogger.business.search.lucene;
21
22  import java.io.IOException;
23  import java.io.StringReader;
24
25  import org.apache.lucene.analysis.Analyzer;
26  import org.apache.lucene.analysis.TokenStream;
27  import org.apache.lucene.analysis.tokenattributes.CharTermAttribute;
28  import org.apache.lucene.index.Term;
29
30  /**
31   * Class containing helper methods.
32   *
33   * @author mindspage Izdeilis (mind@izdeilis.com)
34   */
35  public final class IndexUtil {
  
```

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:

