Explorantative Code Quality Documents

Haris Mumtaz, Shahid Latif, Fabian Beck, and Daniel Weiskopf

InfoVis’19

Presented By:
Syed Ishtiaque Ahmad
Good code quality is needed for efficiently developing maintainable and extendable software

**Goal:**

Self-explanatory system

(Explanation + Exploration)

Specially for

less experienced software developers | less technical stakeholders
Approach

11 Quality Metrics

Analysis

Explanation (Text Generation)

Exploration (Visualization)

Explorationation (Code Quality Document)

4 Quality Attributes
4 Code Smell
Bugs
Data and Analysis

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Software Metric</th>
<th>Acronym</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>Weighted methods per class</td>
<td>wmc</td>
</tr>
<tr>
<td></td>
<td>Maximum cyclomatic complexity</td>
<td>max.cc</td>
</tr>
<tr>
<td>Coupling</td>
<td>Afferent coupling</td>
<td>ca</td>
</tr>
<tr>
<td></td>
<td>Affective coupling</td>
<td>ce</td>
</tr>
<tr>
<td>Cohesion</td>
<td>Lack of cohesion of methods</td>
<td>lcom3</td>
</tr>
<tr>
<td>Inheritance</td>
<td>Depth of inheritance</td>
<td>dit</td>
</tr>
<tr>
<td></td>
<td>Number of children</td>
<td>noc</td>
</tr>
<tr>
<td>Other</td>
<td>Average method complexity</td>
<td>amc</td>
</tr>
<tr>
<td></td>
<td>Lines of code</td>
<td>loc</td>
</tr>
<tr>
<td></td>
<td>Number of public methods</td>
<td>nmp</td>
</tr>
<tr>
<td></td>
<td>Number of bugs</td>
<td>ng</td>
</tr>
</tbody>
</table>

4 quality attributes

11 metrics

Used to detect 4 types of code smell
- Large Class
- Lazy Class
- Functional Decomposition
- Spaghetti Code

To show bug-proneness

Use these software metrics with threshold to measure quality attributes

Based on Filó, T.G. et al. work on “A Catalogue of Thresholds for Object-Oriented Software Metrics”
1. Summary Text

The document presents the code quality aspects of Lucene 2.0—-it has 10 packages and 195 classes.

Quality Attributes

An analysis of software metrics along four quality attributes shows generally mixed quality.

- **Complexity**: The code complexity is okay as 51 classes are rated as having low quality, still fewer than the ones rated as good (94) or regular (50).
- **Coupling**: The coupling between classes is good as only eleven classes are rated as having low quality, but many as good.
- **Cohesion**: The cohesion within classes is as follows: 89 classes are rated as having low quality, still fewer than the ones rated as good (17) or regular.
- **Inheritance**: The usage of class inheritance is good as only eleven classes are rated as having low quality, but many as good.

2. Overview Visualization

An overview of software quality in terms of complexity, coupling, cohesion, inheritance, and other metrics (loc, amc, npm, bug). The class `StandardTokenizerTokenManager` is selected and drawn in black.

3. Details

**Code Smells**

We have detected eight class-level code smells, among them one Large Class, six cases of Functional Decomposition, and one case of Spaghetti Code. Some classes carry multiple smells. For instance, `StandardTokenizerTokenManager` has three smells: Large Class, Functional Decomposition, and Spaghetti Code.

**Bug History**

With respect to past and present bugs, 91 of the classes have been associated with bugs. The class `StandardTokenizerTokenManager` has both associated bugs and identified code smells.
An analysis of software metrics along four quality attributes shows generally mixed quality. 

**Complexity:** The code complexity is **okay:** as 51 classes are rated as having low quality, still fewer than the ones rated as **good** (94) or **regular** (50). [-]

**Coupling:** The coupling between classes is **good:** as only eleven classes are rated as having low quality, but many as **good** (114). [+]

**Cohesion:** The cohesion within classes is **okay:** as 89 classes are rated as having low quality, still fewer than the ones rated as **good** (17) or **regular** (89). [+]

**Inheritance:** The usage of class inheritance is **good:** as only eleven classes are rated as having low quality, but many as **good** (135). [+]

**Code Smells**

We have detected eight class-level code smells, among them one Large Class, six cases of Functional Decomposition, and one case of Spaghetti Code. Some classes carry multiple smells. For instance, **StandardTokenizerTokenManager** has three smells: Large Class, Functional Decomposition, and Spaghetti Code.

**Bug History**

With respect to past and present bugs, 91 of the classes have been associated with bugs. The class **StandardTokenizerTokenManager** has both associated bugs and identified code smells.

Embedded visualization shows actual value of the software metrics in the respective category.
Overview Visualization

An overview of software quality in terms of complexity (wmc, max cc), coupling (ca, ce), cohesion (lcom3), inheritance (noc, dit) and other metrics (loc, amc, npm, bug).

Gray lines (left) and dots (right) represent classes. The current selection is drawn in black □ and contains three classes: QueryParserTokenManager (loc: 2855), QueryParser (loc: 2830), and StandardTokenizerTokenManager (loc: 3709).
Details

Educational

Background: Complexity Metrics

Complexity metrics estimate how difficult it is to understand the respective code (not to be confused with “computational complexity”, which refers to the runtime resources an algorithm consumes). Based on a complexity computation on method level, we consider two perspectives: First, weighted methods per class (wmc) sum all method complexity values for a class. Second, to also highlight classes that contain few high-complexity methods but many low-complexity ones, the maximal cyclomatic complexity (max_cc) takes into consideration the maximum of all the method-level complexity values of class.

Within the analyzed classes, IndexReader has the highest value with respect to weighted methods per class (wmc) and StandardTokenizerTokenManager the highest value with respect to maximum cyclomatic complexity (max_cc).

We use thresholds values of weighted methods per class (wmc) and maximum cyclomatic complexity (max_cc) for categorizing coupling as low, regular, or good.

Low: wmc > 34 OR max_cc > 4
Regular: not low AND (wmc > 11 OR max_cc > 2)
Good: all other cases

The detailed embedded visualization can be expanded by clicking on the [+] icon. Each bar in this visualization represents a class and the classes are grouped by packages.

Methodological

Data-driven explanations

[X] Class StandardTokenizerTokenManager

Class StandardTokenizerTokenManager is the largest class of the project (loc: 3709) with, considering its size, a relatively small public interface (nppm: 7). It has low quality with respect to the attributes complexity and cohesion. Moreover, it carries Large Class, Functional Decomposition, and Spaghetti Code smells.

```java
package org.apache.lucene.analysis.standard;
import java.io.*;

public class StandardTokenizerTokenManager implements StandardTokenizerC {
    public java.io.PrintStream debugStream = System.out;
    public void setDebugStream(java.io.PrintStream ds) (debugStream = ds);
    private final int jjMoveStringLiteralDfa0_0() {
```
DEMO

https://vis-tools.paluno.uni-due.de/cqd/
Interaction model

Transient selection on hovering over a class name anywhere highlights:

- text-vis: polyline in parallel coordinates, dot in scatterplot
- text-emvis: bar in embedded visualization
- text-text: other occurrence of class name in the text

Persistent selection on clicked: encoded by black color (good for comparing classes)

Persistent range selection on the axes of parallel coordinates
Design Process and Evaluation

Formative Evaluation Iteration # 1

• 4 participants ( 3 PhD, 1 postdoc )
• Mix of visualization and software experts
• Study included 3 phases (45 minutes)
  - Identify different aspect of code quality in a document
  - Participant reviewed features of the system and provided feedback
  - Interview the participants asking general questions
Design Process and Evaluation

Formative Evaluation Iteration # 2

- 3 previous participants (2 PhD + 1 postdoc) + 1 new participant (PhD)
- New participant is currently conducting visualization research and has a software engineering background
- Study included 2 phases (30 minutes)
  - Participants reviewed features of the system and provided feedback
  - Interview the participants focusing on specific improvements
Results

Iteration #1
- Added methodological and educational explanation
- Added interaction between all representations (only text-vis interaction was present in prototype)

Iteration #2
- All the participants agreed that system improved overall
- More information about the bug history was desired
Recommendations for Interactive Documents

Consider brushing text, really!

You just learned on the sides!

Educational
Data-driven
Methodological

Captions! And make them dynamic!
What-Why-How Analysis

| What: Data | Java project source code (Xerces 1.2, Lucene 2.0, Forrest); Multivariate data; |
| What: Derived | 11 metrics (4 quality attributes, 4 code smell, number of bugs) |
| Why: Tasks | Self explanatory system to teach and report about software code quality |
| How: Encode | Parallel coordinates; Scatterplots; bar charts; Consistent colouring; Glyphs; |
| How: Facet | Multiple view panel coordinated with link highlighting and colouring |
| How: Manipulate | Hover and click interaction to link texts, visuals and embedded visuals in a bidirectional way; Brushing interaction with mouse press and hold for parallel coordinates; |
| How: Reduce | Filter class by brushing parallel coordinates axes |
| Scale | Java project source code consisting about 200 ~ 300 classes |
Strengths and Weaknesses

PROS
- Provides more context to the data and explain the findings in detail
- Follows an incremental design process
- Provides recommendations for interactive documents with multivariate data

CONS
- Does not provide solution to occlusion problems with scatterplot
- Unable to view and compare all four quality attributes at once
- Using same person for second iteration of design evaluation introduces biasness
References

1. Talk: https://vimeo.com/370669433
2. Tool: https://vis-tools.paluno.uni-due.de/cqd/
3. Paper: https://www.computer.org/csdl/journal/tg/5555/01/08807349/1cG6mtDwLNm
Thank you!
Questions?