Exploranative 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

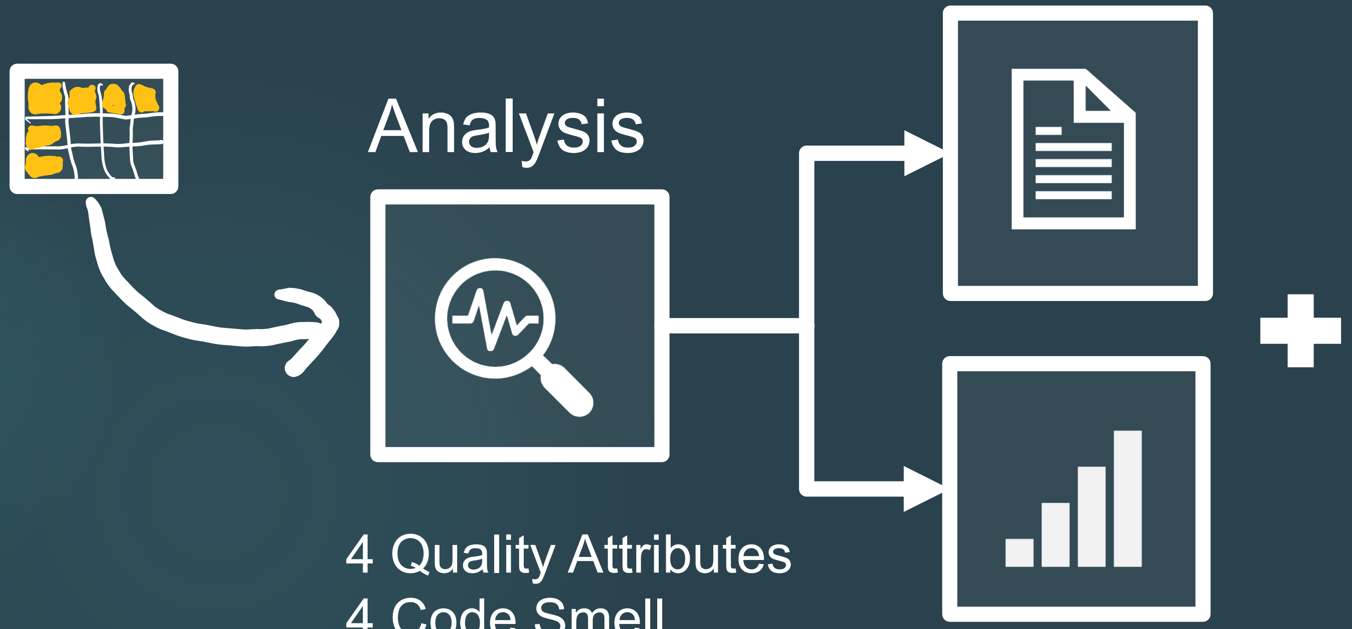
Goal:

Self-explanatory system

(Explanation + Exploration)

Specially for
less experienced software developers | less technical stakeholders

Approach



11 Quality Metrics

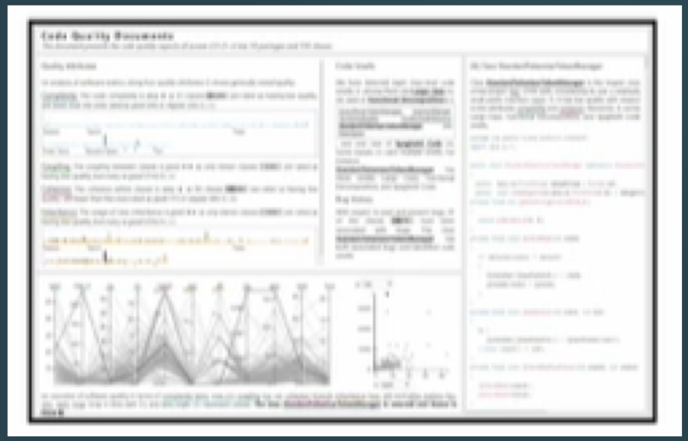
Analysis

4 Quality Attributes
4 Code Smell
Bugs

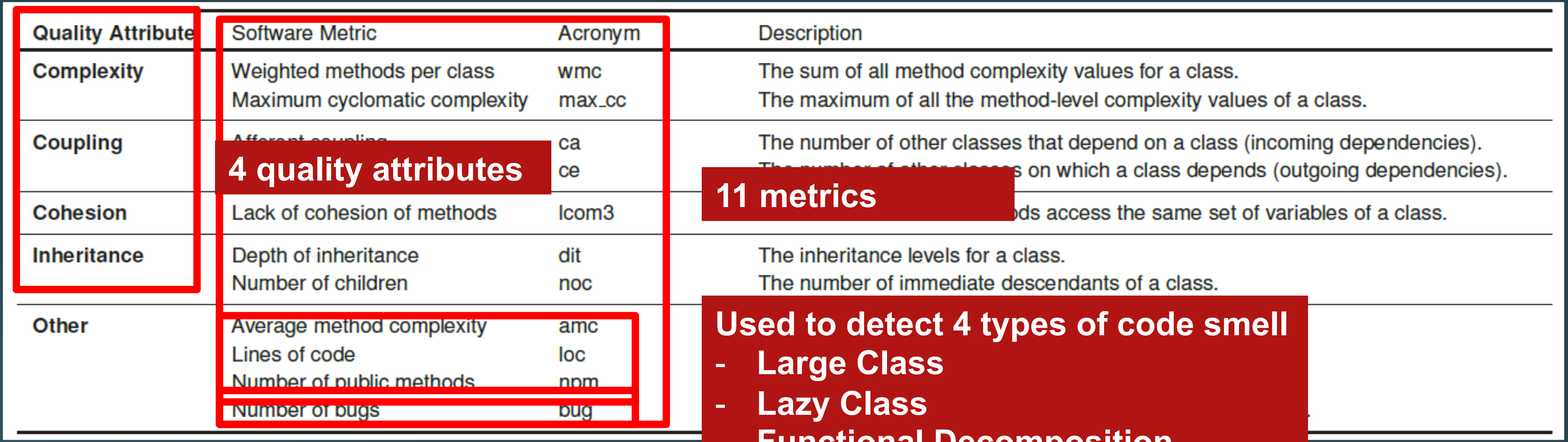
Explanation
(Text Generation)

Exploration
(Visualization)

Exploranation
(Code Quality Document)



Data and Analysis



Quality Attribute	Software Metric	Acronym	Description
Complexity	Weighted methods per class	wmc	The sum of all method complexity values for a class.
	Maximum cyclomatic complexity	max_cc	The maximum of all the method-level complexity values of a class.
Coupling	Afferent coupling	ca	The number of other classes that depend on a class (incoming dependencies).
	[illegible]	ce	The number of other classes on which a class depends (outgoing dependencies).
Cohesion	Lack of cohesion of methods	lcom3	[illegible] ods access the same set of variables of a class.
Inheritance	Depth of inheritance	dit	The inheritance levels for a class.
	Number of children	noc	The number of immediate descendants of a class.
Other	Average method complexity	amc	[illegible]
	Lines of code	loc	[illegible]
	Number of public methods	npm	[illegible]
	Number of bugs	bug	[illegible]

4 quality attributes

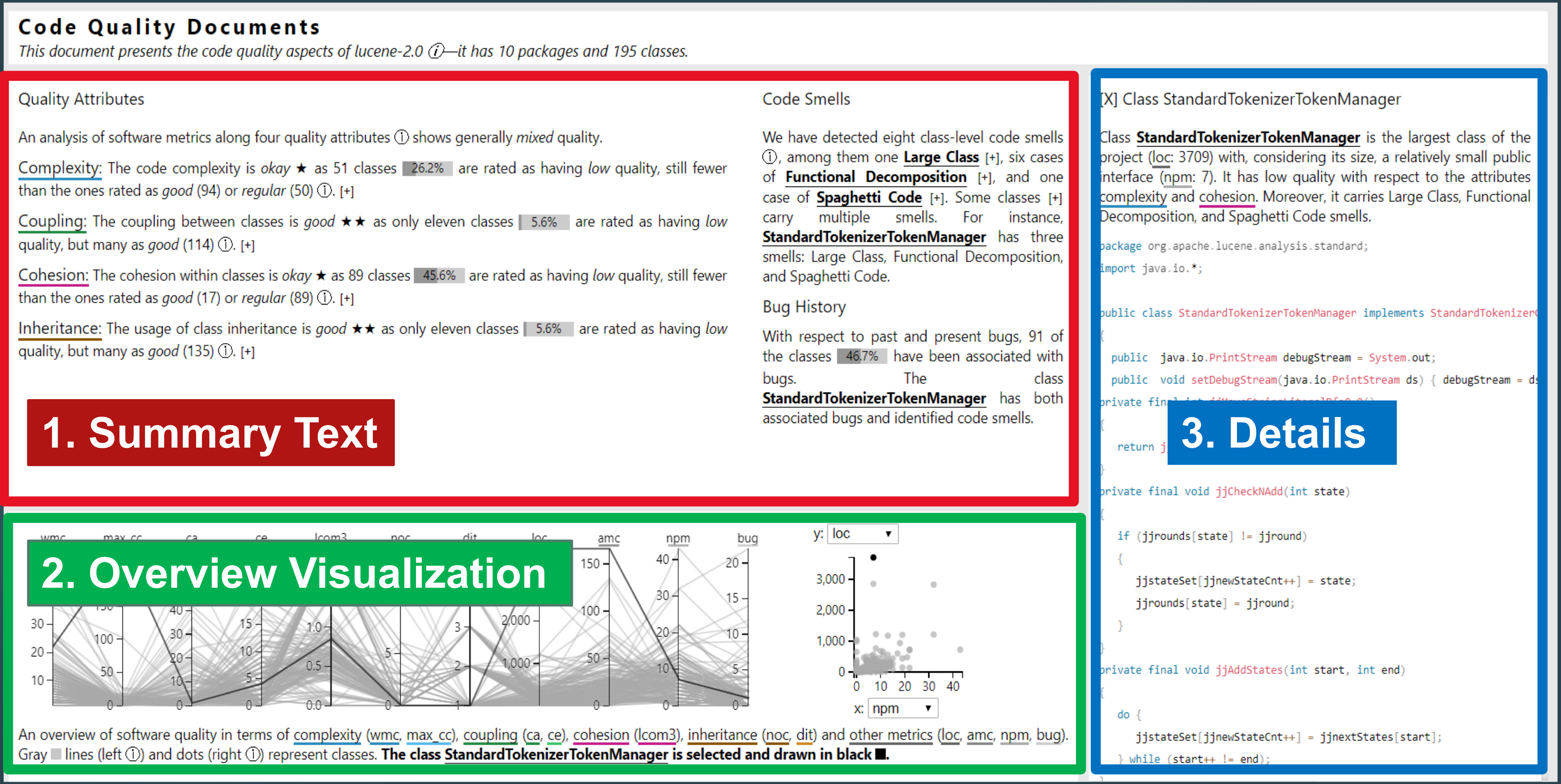
11 metrics

To show bug-proneness

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

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”



Code Quality Documents

This 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 26.2% 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 5.6% are rated as having *low* quality, but many as *good* (114) ⓘ. [+]

Cohesion: The cohesion within classes is *okay* ★ as 89 classes 45.6% 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 5.6% are rated as having *low* quality, but many as *good* (135) ⓘ. [+]

1. Summary Text

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 46.7% have been associated with bugs. The class **StandardTokenizerTokenManager** has both associated bugs and identified code smells.

2. 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 class StandardTokenizerTokenManager is selected and drawn in black ■.**

[X] Class StandardTokenizerTokenManager

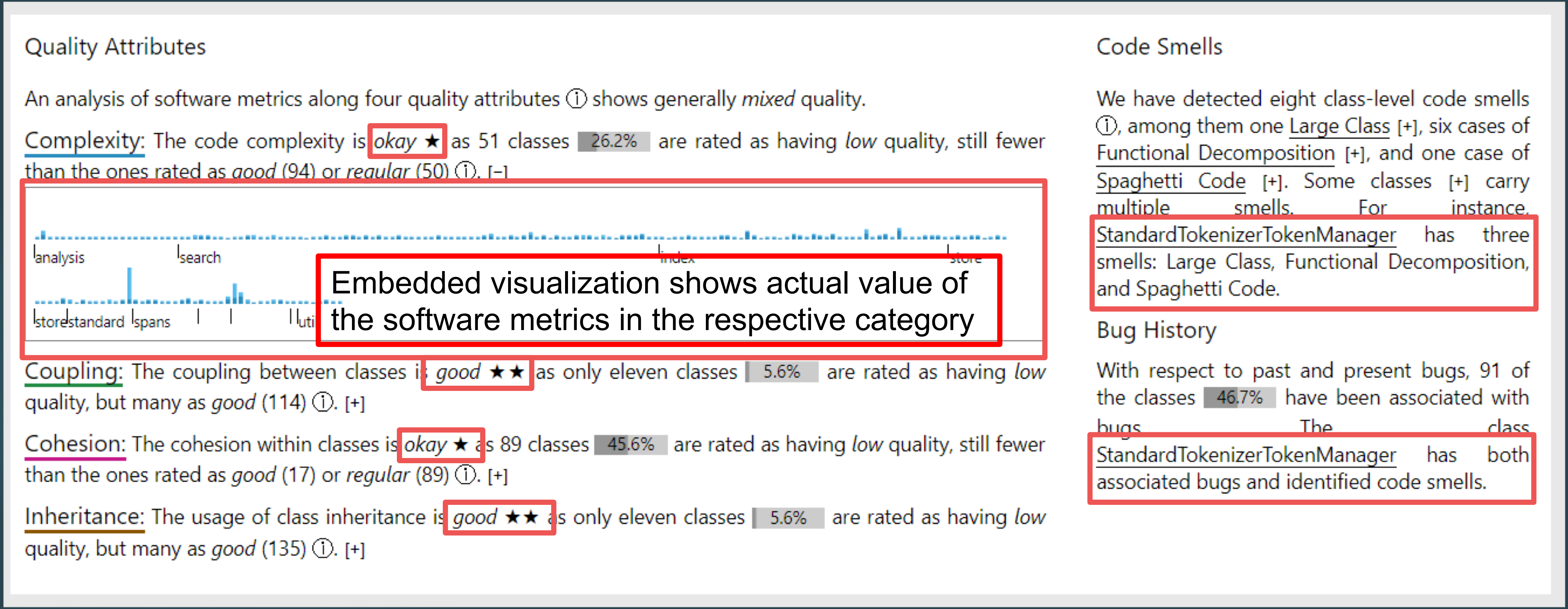
Class **StandardTokenizerTokenManager** is the largest class of the project (loc: 3709) with, considering its size, a relatively small public interface (npm: 7). It has low quality with respect to the attributes complexity and cohesion. Moreover, it carries Large Class, Functional Decomposition, and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

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

public class StandardTokenizerTokenManager implements StandardTokenizer(
{
  public  java.io.PrintStream debugStream = System.out;
  public  void setDebugStream(java.io.PrintStream ds) { debugStream = ds
private fin
{
   return j
}
private final void jjCheckNAdd(int state)
{
   if (jjrounds[state] != jjround)
   {
      jjstateSet[jjnewStateCnt++] = state;
      jjrounds[state] = jjround;
   }
}
private final void jjAddStates(int start, int end)
{
   do {
      jjstateSet[jjnewStateCnt++] = jjnextStates[start];
   } while (start++ != end);
```

3. Details

Summary Text



Quality Attributes

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

Complexity: The code complexity is *okay* ★ as 51 classes 26.2% are rated as having *low* quality, still fewer than the ones rated as *good* (94) or *regular* (50) ⓘ. [–]

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

Coupling: The coupling between classes is *good* ★★ as only eleven classes 5.6% are rated as having *low* quality, but many as *good* (114) ⓘ. [+]

Cohesion: The cohesion within classes is *okay* ★ as 89 classes 45.6% 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 5.6% are rated as having *low* quality, but many as *good* (135) ⓘ. [+]

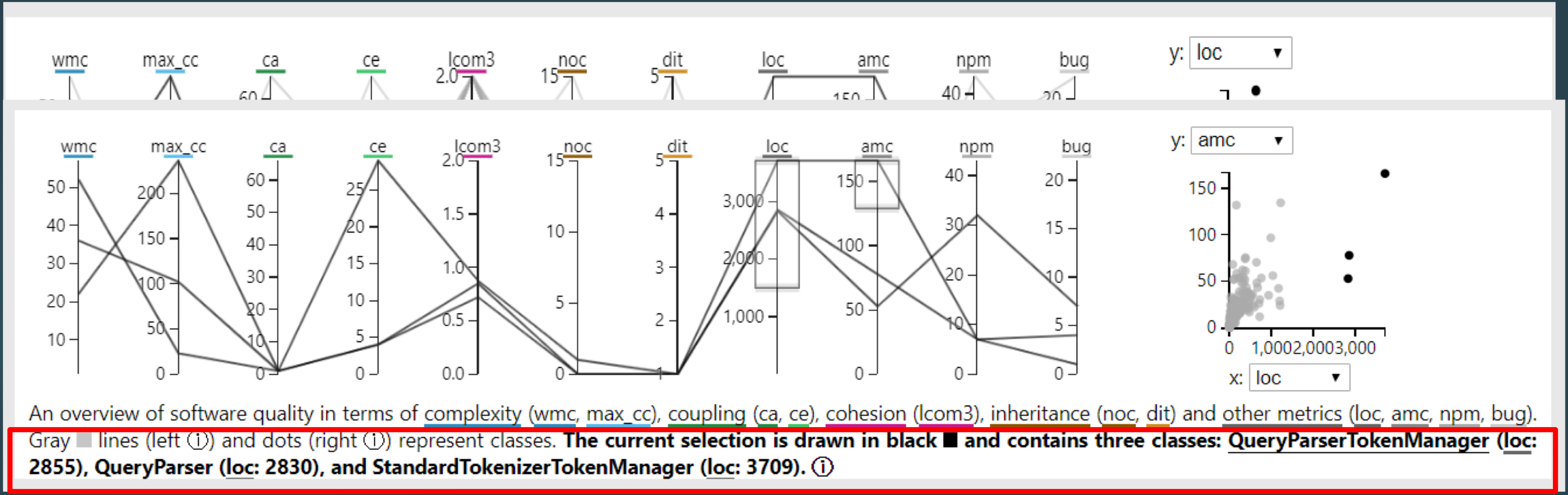
Code Smells

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 46.7% have been associated with bugs. The class StandardTokenizerTokenManager has both associated bugs and identified code smells.

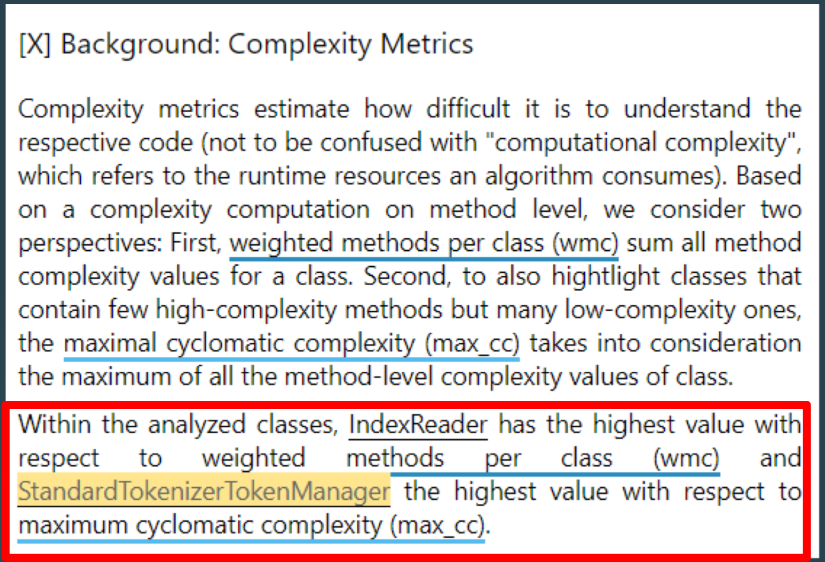
Overview Visualization



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

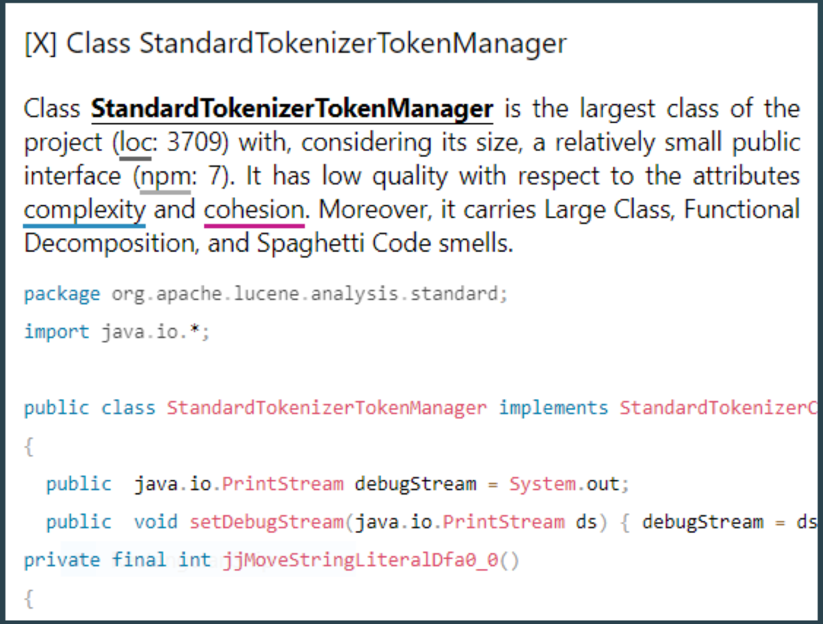


[X] Background: Complexity Metrics

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 hightlight 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).

Data-driven explanations

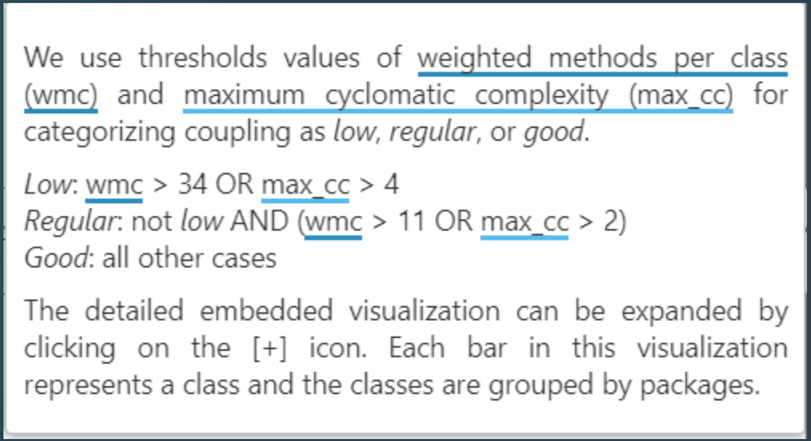


[X] Class StandardTokenizerTokenManager

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

```
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()
{
```



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

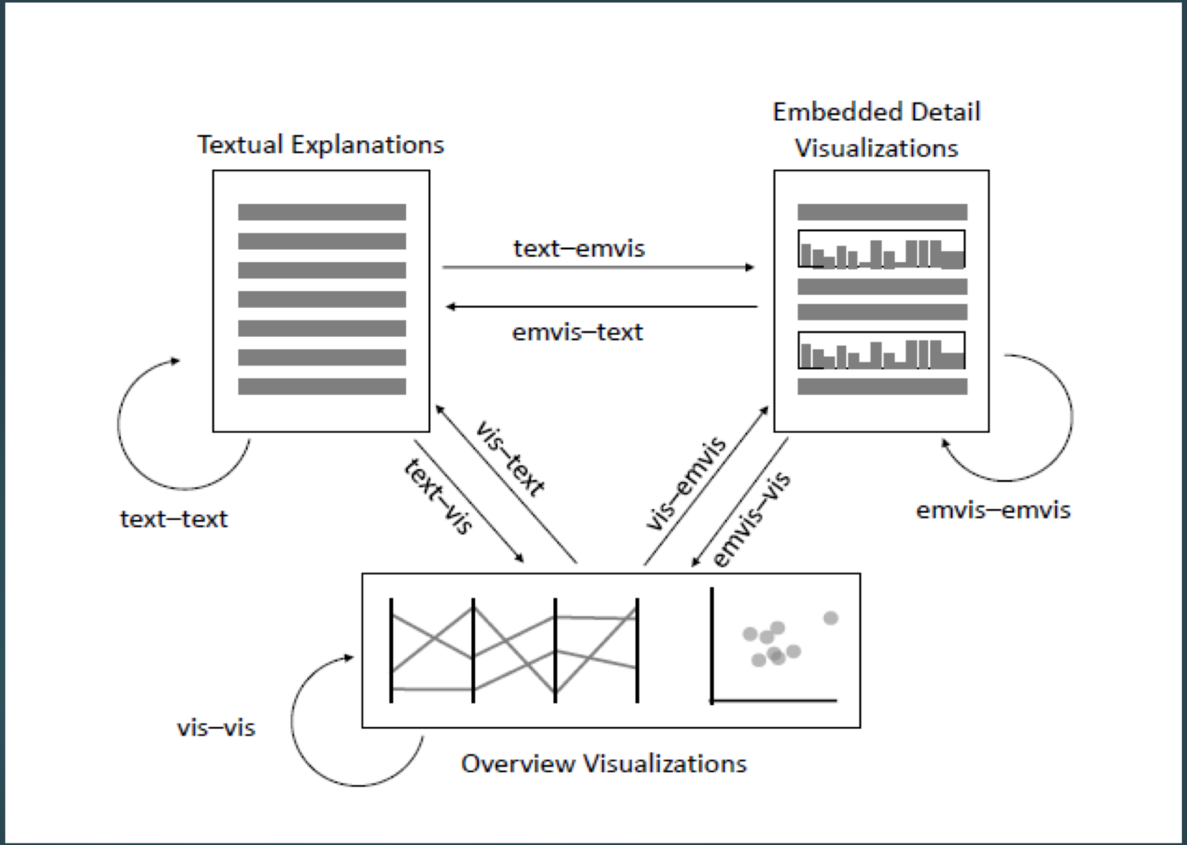
The detailed embedded visualization can be expanded by clicking on the [+] icon. Each bar in this visualization represents a class and the classes are grouped by packages.

Methodological

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)

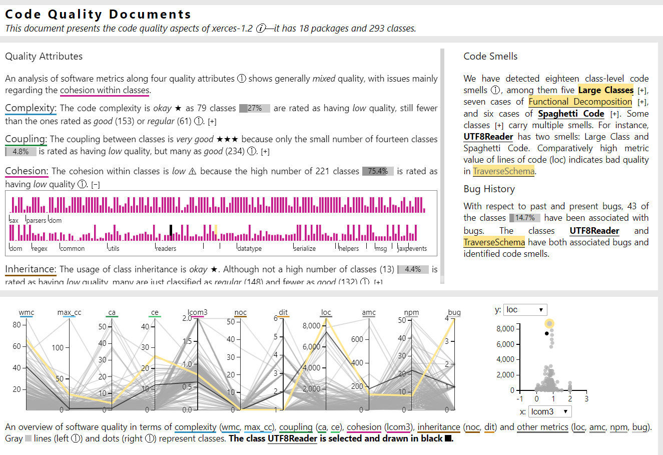
Iteration #2

- All the participants agreed that system improved overall

- More information about the bug history was desired

Recommendations for Interactive Documents

Consider brushing text, really!



Code Quality Documents

This document presents the code quality aspects of xerces-1.2 ⓘ—it has 18 packages and 293 classes.

Quality Attributes

An analysis of software metrics along four quality attributes ⓘ shows generally *mixed* quality, with issues mainly regarding the cohesion within classes.

Complexity: The code complexity is *okay* ★ as 79 classes 27% are rated as having *low* quality, still fewer than the ones rated as *good* (153) or *regular* (61) ⓘ. [+]

Coupling: The coupling between classes is *very good* ★★★ because only the small number of fourteen classes 4.8% is rated as having *low* quality, but many as *good* (234) ⓘ. [+]

Cohesion: The cohesion within classes is *low* ⚠ because the high number of 221 classes 75.4% is rated as having *low* quality ⓘ. [−]

Inheritance: The usage of class inheritance is *okay* ★. Although not a high number of classes (13) 4.4% is rated as having *low* quality, many are just classified as *regular* (148) and fewer as *good* (132) ⓘ. [+]

Code Smells

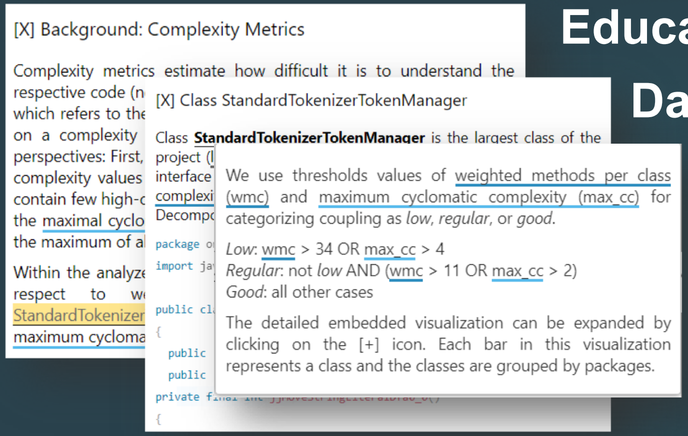
We have detected eighteen class-level code smells ⓘ, among them five **Large Classes** [+], seven cases of Functional Decomposition [+], and six cases of **Spaghetti Code** [+]. Some classes [+] carry multiple smells. For instance, **UTF8Reader** has two smells: Large Class and Spaghetti Code. Comparatively high metric value of lines of code (loc) indicates bad quality in TraverseSchema.

Bug History

With respect to past and present bugs, 43 of the classes 14.7% have been associated with bugs. The classes **UTF8Reader** and TraverseSchema have both associated bugs and identified code smells.

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 class UTF8Reader is selected and drawn in black ■.**

You just learned on the sides!



[X] Background: Complexity Metrics

Complexity metrics estimate how difficult it is to understand the respective code (n
which refers to the
on a complexity
perspectives: First,
complexity values
contain few high-c
the maximal cyclo
the maximum of al

Within the analyze
respect to w
StandardTokenizer
maximum cycloma

[X] Class StandardTokenizerTokenManager

Class **StandardTokenizerTokenManager** is the largest class of the
project (l
interface
complexi
Decompo

```
package o
import ja
public cl
{
  public
  public
private final int jjMoveStringLiteralDfa0_0()
{
```

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

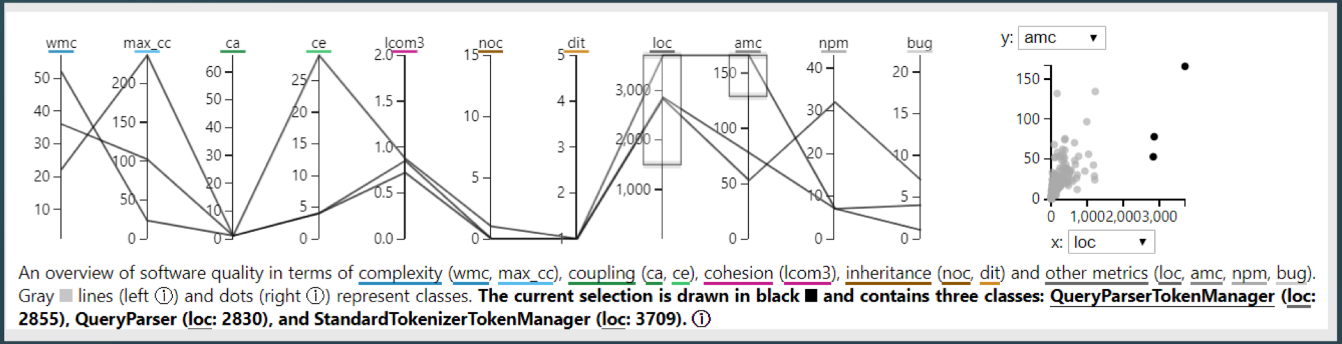
The detailed embedded visualization can be expanded by clicking on the [+] icon. Each bar in this visualization represents a class and the classes are grouped by packages.

Educational

Data-driven

Methodological

Captions! And make them dynamic

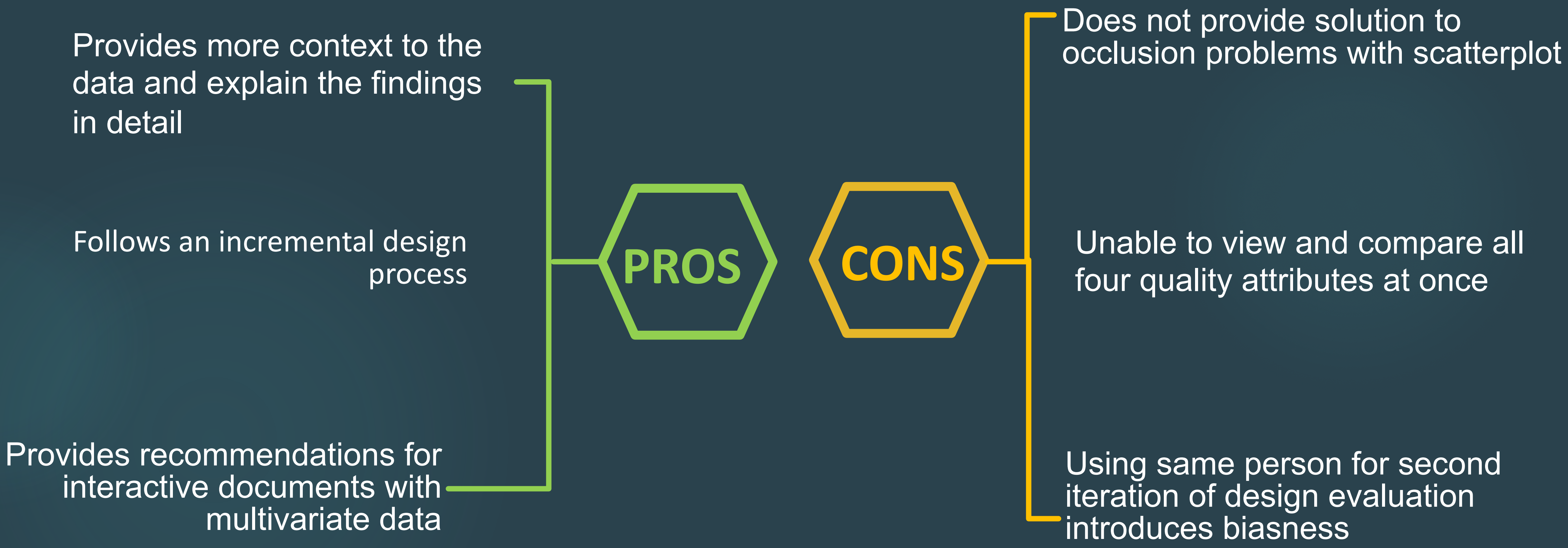


An overview of software quality in terms of complexity (wmc, max_cc), coupling (ca, ce), cohesion (lcom3), inheritance (noc, dit) and other metrics (loc, amc, npm, bug). Gray lines (left ⓘ) and dots (right ⓘ) represent classes. **The current selection is drawn in black ■ and contains three classes: QueryParserTokenManager (loc: 2855), QueryParser (loc: 2830), and StandardTokenizerTokenManager (loc: 3709).** ⓘ

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?