CPSC-60000-001 - Object Oriented Development

Group Assignment 2

Group: 7

Team Members:

Manoj Ashokreddy – L30097799

Jayanth Vasa – L30085539

Contents

Group Assignment 2 1

Section – 1 3

Second empirical study: Effect of code bad smells on modularity **3**

1.Introduction: **3**

1.1. Objectives: **3**

1.2 Questions: **3**

1. 3 Metrics: 4

Section – 2 ..... **5**

2.1. Criteria for Subject Programs with justification: 5

2.2. Subject Programs: 5

2.3. What Programs do: 6

Section – 3 8

3.1. Tool Description: 8

3.2. Tool Citation: 8

Section – 4 9

4.1. Project: Netty project 9

4.2. Project: Apache Dubbo 16

4.3. Project: Design patterns implemented in Java 21

4.4. Project: Extensible build system 26

4.5. Project: Apache OpenNLP 31

4.6. Project: Mirror of Apache Struts 38

4.7. Project: Apache Tomcat 42

4.8. Project: H2 Database 48

4.9. Project: Apache Groovy 53

4.10. Project: Java 1-21 Parser and Abstract Syntax Tree 58

Section – 5 66

5.1. Conclusion: 66

5.2. References: 67

Section – 1

Second empirical study: Effect of code bad smells on modularity

1.Introduction:

In the field of software development, one of the most significant characteristics is modularity, which facilitates maintainability, scalability, and an improved understanding. However, code bad smells which are a sign of more fundamental issues in the code can be detrimental in that they can weaken modularity by raising the degree of complexity and cutting across cohesion and coupling quality. This empirical study explores the fact of whether code bad smells affect the modularity of Java software development projects or not based on the values of coupling, cohesion, and complexity. It is through these metrics that we wish to examine how the presence of the smell impacts the modularity of software and drive further insights into the enhancement of the different structures of software.

1.1. Objectives:

The purpose of this empirical investigation is to determine the impact of code bad smells on modularity in Java software projects. Code smells refer to symptoms that communicate the possibility of design shortcomings that may affect different aspects of the quality of software. Coupling and cohesion that are used in the measurement of modularity are part of the quality attribute that determines the maintainability, understandability, and extensibility of the software system.

1.2 Questions:

- What is the impact of code bad smells on modularity in Java software projects?
- What is the effect of code bad smells on the coupling and cohesion within classes in Java software projects?
- How do code bad smells impact the overall complexity of classes in Java software projects, and how does this relate to modularity?

1. 3 Metrics:

To investigate the influence of the identified code bad smells on modularity in Java software projects, we should concentrate on such metrics as coupling, cohesion, and complexity.

- **Coupling Between Objects (CBO)**: CBO is the ratio of the number of classes that a given class is coupled to the total number of classes. A class coupling measure encapsulates the level of dependency a class has on other classes in the system.

- **Lack of Cohesion of Methods (LCOM)**: LCOM computes the degree of coupling within a class by determining how many of its methods declare common instance variables. Concerning this, it compares the dissimilarity of the methods depending on the class fields.

- **Complexity**: Complexity computes the functionality of a program based on the number of linearly independent execution paths traced through the source. It is determined concerning the control flow graph of the program, where each node is a block of code and each arc is a path of control flow.

- **Weighted Methods per Class (WMC)**: WMC is the aggregated of the size of all methods in a class where each method is assigned a size according to its complexity. The complexity of each method is normally calculated via the cyclomatic complexity and WMC sums up these values in the class level.

- **Response for a Class (RFC)**: RFC is a metric that calculates the number of methods that could be called upon in response to a message passed to an object of that class. This means that RFC includes all the methods within the class as well as methods within a class that are called by the RFC and those that are accessed through class attributes.

- **Depth of Inheritance Tree (DIT)**: DIT calculates the inherited levels starting from the object hierarchy up to the class. DIT states the number of direct superclasses of a class, plus all the superclasses of the direct superclass up to the object class.

- **Number of Children (NOC)**: NOC is typically defined as the number of immediate subclasses starting from a specific class. NOC gives the total number of direct descendent classes that can be obtained from a particular class.

- **Class Size**: It typically measured by the number of lines of code, attributes, and methods it contains. It reflects the extent of functionality and behavior encapsulated within a single unit of code, influencing factors such as readability, maintainability, and overall system complexity.

Section – 2

2.1. Criteria for Subject Programs with justification:

We want to analyze projects that are **over 7 years old**, should be more than **10,000 kB** to show their complexity and appropriateness for in-depth analysis, and have at least **15 developers** we can ensure that the projects have gone through multiple development and maintenance cycles, are complex and appropriate for in-depth analysis, and have diverse perspectives due to numerous contributors. These criteria ensure that the chosen projects are effective for studying productivity and maintainability in real-world scenarios.

2.2. Subject Programs:

Project	**Repository**	**Module**	**Language**	**Size**	**Start Time**	**Contributors**	**Total lines of code**
Netty project	https://github.com/netty/netty	codec-dns	Java, C	1,289,468 kbs	Around 2004	667	1904
Apache Dubbo	https://github.com/docs/advanced-java	dubbo-compiler	Java	10,258 kbs	November 2011	39	342
Design patterns implemented in Java	https://github.com/iluwatar/java-design-patterns	abstract-factory	Java	21,431 kbs	November 13, 2013	421	98
Extensible build system	https://github.com/bazelbuild/bazel	bazel	Java	509,589 kbs	December 19, 2014	981	146
Apache OpenNLP	https://github.com/apache/opennlp	opennlp-uima	Java	14781 kbs	April 25 2007	50	1383
Mirror of Apache Struts	https://github.com/apache/struts	rest-showcase	Java	25800 kbs	July 16 2004	70	120
Apache	https://gith	openssl-	Java	12,000 kbs	March	117	6265

Tomcat	ub.com/apache/tomcat	foreign			2012		
H2 Database	https://github.com/h2database/h2database	org.h2.api	Java	17,430 kbs	2006	157	717
Apache Groovy	https://github.com/apache/groovy	build-logic	Java	354,019 kbs	March 2004	366	453
Java 1-21 Parser and Abstract Syntax Tree	https://github.com/javaparser/javaparser	javaparser-core	Java	34,000 kbs	June 15, 2011	203	804

2.3. What Programs do:

1. **Netty**: Netty is a framework for building high-performance network applications in Java. It provides asynchronous event-driven network application framework and tools for building various types of protocols.

2. **Apache Dubbo**: Dubbo is a high-performance, lightweight, and open-source RPC (Remote Procedure Call) framework based on Java. It simplifies distributed application development by providing various features like load balancing, fault tolerance, and service discovery.

3. **Design Patterns implemented in Java**: This repository contains examples and explanations of various design patterns implemented in Java. Design patterns are reusable solutions to common software design problems, helping developers create flexible and maintainable code.

4. **Extensible build system Bazel**: Bazel is an open-source build and test tool that emphasizes correctness and reproducibility. It is designed to handle large codebases across multiple repositories and languages, supporting various build targets and extensible via plugins.

5. **Apache OpenNLP**: OpenNLP is an open-source natural language processing (NLP) toolkit for processing and analyzing text. It provides tools for tokenization, sentence segmentation, part-of-speech tagging, named entity recognition, and more.

6. **Mirror of Apache Struts**: Apache Struts is an open-source framework used to develop Java web applications. It provides a Model-View-Controller (MVC) architecture, along with features for handling forms, validation, and various utilities for web application development.

7. **Apache Tomcat**: Tomcat is an open-source implementation of the Java Servlet, JavaServer Pages (JSP), and WebSocket technologies. It serves as a web server and servlet container for running Java web applications.

8. **H2 Database**: H2 is an open-source relational database management system written in Java. It supports SQL queries, transactions, and offers a small footprint that can be embedded into Java applications or run as a standalone database server.

9. **Apache Groovy**: Groovy is a powerful, optionally typed and dynamic language for the Java Virtual Machine (JVM). It integrates smoothly with any Java program and provides enhanced scripting capabilities, making it suitable for rapid application development.

10. **Java 1-21 Parser and Abstract Syntax Tree**: JavaParser is an open-source Java library for parsing, analyzing, and modifying Java source code programmatically. It provides APIs to work with the Abstract Syntax Tree (AST) of Java code, enabling tasks like code generation and transformation.

Section – 3

3.1. Tool Description:

The tool employed for this work are CodeMR and JDeodorant.

CodeMR:

CodeMR is an automated static code analysis tool that analyses all the code aspects to help understand the quality and ease of maintainability of complex software. It includes services like it complexity analysis, dependencies analysis, code metrics calculation, etc. CodeMR aids developers and teams in detecting problems in the code base before code implementation allowing for optimal code stability, reduced implementation of substandard codes, and overall software sustenance. It helps you to write scripts in different languages and offers real-time visualization for analysis and comparing organized software systems.

JDeodorant:

JDeodorant is a tool for refactoring Java applications that do not have any equivalent. It concentrates on the identification of the code smells including code duplication, large methods, excessive methods, and huge conditional statements which hints at the areas where there is a need for further improvement of the code quality and manageability. The Java tool JDeodorant offers refactorings for the automated simplification of the structure of code to optimize it for readability and minimize the possibility of bugs. It is compatible with several widely used Java IDEs, which enables the developers to apply the refactorings in context and while they are within the design environment, thus making the process of code enhancement more efficient.

3.2. Tool Citation:

- CodeMR. (n.d.). CodeMR - Multi-language Software Quality and Static Code Analysis Tool [Eclipse Marketplace listing]. Retrieved from https://marketplace.eclipse.org/content/codemr-static-code-analyser

- Moha, N., Gueheneuc, Y. G., Duchien, L., & Le Meur, A. F. (2010). JDeodorant: Identification and removal of feature envy bad smells. In *2010 17th Working Conference on Reverse Engineering (WCRE)* (pp. 319-320). IEEE.

Section – 4

4.1. Project: Netty project
Module: codec-dns

The Values below are Obtained from CodeMR Tool:

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
io.netty5.handler.codec.dns.AbstractDnsOptPseudoRrRecord	2	0	low-medium	8	15	2	1	low
io.netty5.handler.codec.dns.DatagramDnsQueryDecoder	5	0	low-medium	4	21	3	0	low
io.netty5.handler.codec.dns.DatagramDnsQueryEncoder	8	0	low-medium	5	25	3	0	low
io.netty5.handler.codec.dns.DatagramDnsResponseDecoder	6	0	low-medium	6	26	3	0	low
io.netty5.handler.codec.dns.DatagramDnsResponseEncoder	8	0	low-medium	5	31	3	0	low
io.netty5.handler.codec.dns.DefaultDnsOptEcsRecord	1	0.5	low-medium	11	25	3	0	low
io.netty5.handler.codec.dns.DefaultDnsPtrRecord	3	0	low-medium	5	18	2	0	low
io.netty5.handler.codec.dns.DefaultDnsQuestion	3	0	low-medium	4	14	2	0	low
io.netty5.handler.codec.dns.DefaultDnsRecordDecoder	9	0	low	11	34	1	0	low-medium
io.netty5.handler.codec.dns.DnsCodecUtil	5	0	low-medium	22	24	1	0	low-medium
io.netty5.handler.codec.dns.DnsOptEcsRecord	0	0	low	4	4	1	1	low
io.netty5.handler.codec.dns.DnsOptPseudoRecord	0	0	low	4	4	1	2	low

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
io.netty5.handler.codec.dns.DnsPtrRecord	0	0	low	2	2	1	1	low
io.netty5.handler.codec.dns.DnsQueryEncoder	5	0	low	8	16	1	0	low
io.netty5.handler.codec.dns.DnsQuestion	0	0	low	2	2	1	1	low
io.netty5.handler.codec.dns.DnsRawRecord	1	0	low	3	3	1	1	low
io.netty5.handler.codec.dns.DnsRecord	1	0	low	5	5	1	5	low
io.netty5.handler.codec.dns.DnsRecordDecoder	3	0	low	2	2	1	1	low
io.netty5.handler.codec.dns.DnsRecordEncoder	3	0	low	2	2	1	1	low
io.netty5.handler.codec.dns.DnsResponseDecoder<A>	9	0	low	12	18	1	2	low-medium
io.netty5.handler.codec.dns.DnsSection	0	0	low-medium	0	0	2	0	low
io.netty5.handler.codec.dns.TcpDnsQueryDecoder	4	0	medium-high	4	20	4	0	low
io.netty5.handler.codec.dns.TcpDnsQueryEncoder	6	0	low-medium	5	24	3	0	low
io.netty5.handler.codec.dns.TcpDnsResponseDecoder	5	0	medium-high	4	25	4	0	low
io.netty5.handler.codec.dns.TcpDnsResponseEncoder	6	0	low-medium	4	31	3	0	low
io.netty5.handler.codec.dns.AbstractDnsRecord	6	0.8	low	20	31	1	4	low-medium
io.netty5.handler.codec.dns.DefaultDnsRawRecord	6	0	low-medium	13	34	2	0	low-medium
io.netty5.handler.codec.dns.DnsMessageUtil	14	0	low-medium	49	51	1	0	low-medium
io.netty5.handler.codec.dns.DnsOpCode	0	0.875	low	16	9	1	0	low-medium
io.netty5.handler.codec.dns.DnsRecordType	0	1.131	low	14	11	1	0	low-medium
io.netty5.handler.codec.dns.DnsResp	0	0.967	low-	29	9	1	0	low-

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
onseCode			medium					medium
io.netty5.handler.codec.dns.AbstractDnsMessage	7	0.946	medium-high	87	61	2	2	medium-high
io.netty5.handler.codec.dns.DatagramDnsQuery	3	0.5	low	31	76	4	0	medium-high
io.netty5.handler.codec.dns.DatagramDnsResponse	4	0.5	low	36	85	4	0	medium-high
io.netty5.handler.codec.dns.DefaultDnsQuery	4	0	low	16	70	3	1	medium-high
io.netty5.handler.codec.dns.DefaultDnsRecordEncoder	11	1	low-medium	30	38	1	0	medium-high
io.netty5.handler.codec.dns.DefaultDnsResponse	5	0.857	low-medium	25	85	3	1	medium-high
io.netty5.handler.codec.dns.DnsMessage	3	0	low-medium	23	21	1	3	medium-high
io.netty5.handler.codec.dns.DnsQuery	3	0	low	13	13	1	1	medium-high
io.netty5.handler.codec.dns.DnsResponse	4	0	low-medium	21	21	1	1	medium-high

JDeodorant obtained results are below:

Classes with Code Bad Smells:

io.netty5.handler.codec.dns.DnsMessageUtil

io.netty5.handler.codec.dns.AbstractDnsMessage

io.netty5.handler.codec.dns.DatagramDnsQuery

io.netty5.handler.codec.dns.DatagramDnsResponse

io.netty5.handler.codec.dns.DefaultDnsQuery

io.netty5.handler.codec.dns.DefaultDnsRecordEncoder

io.netty5.handler.codec.dns.DefaultDnsResponse

io.netty5.handler.codec.dns.DnsMessage

Classes without Code Bad Smells:

io.netty5.handler.codec.dns.AbstractDnsOptPseudoRrRecord

io.netty5.handler.codec.dns.DnsOptEcsRecord

io.netty5.handler.codec.dns.DnsOptPseudoRecord

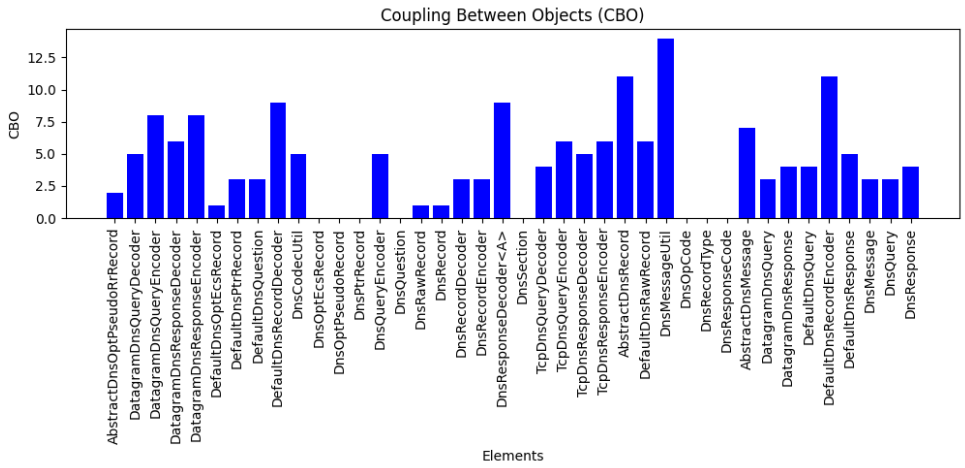
io.netty5.handler.codec.dns.DnsPtrRecord

io.netty5.handler.codec.dns.DnsQueryEncoder

io.netty5.handler.codec.dns.DnsQuestion

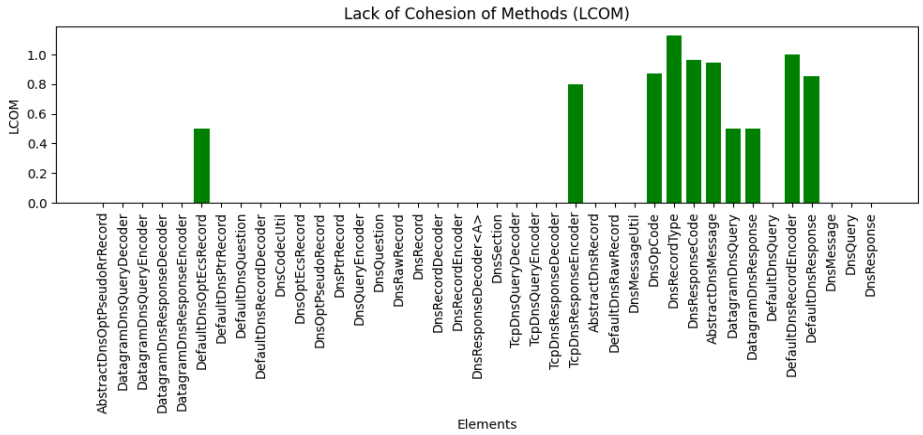
io.netty5.handler.codec.dns.DnsRawRecord

1. **Coupling Between Objects (CBO):**



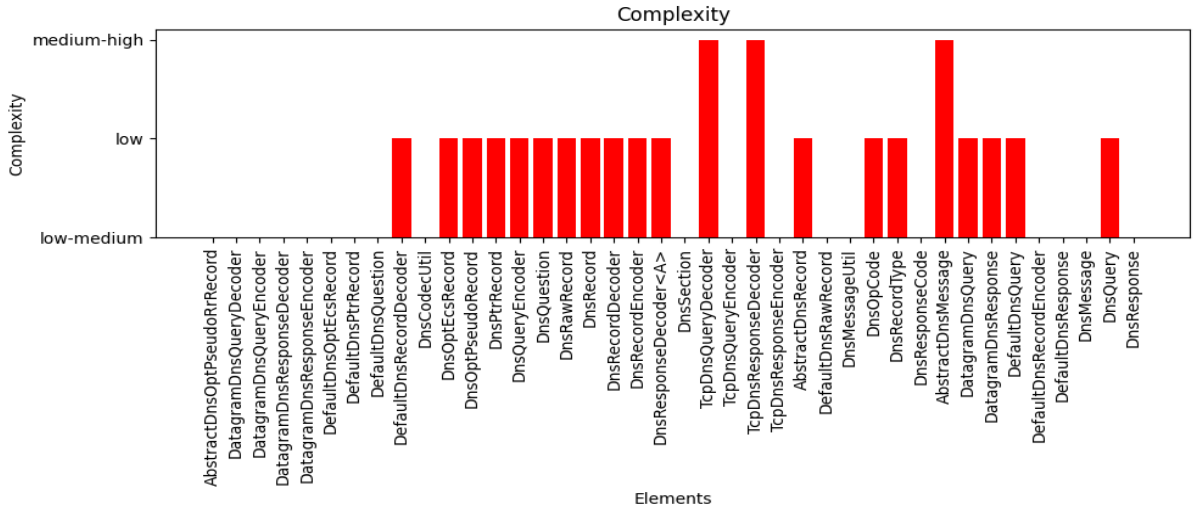
High Coupling Between Objects (CBO) indicates that classes are highly dependent on other classes, which increases complexity and makes maintenance challenging. Classes such as ‘io.netty5.handler.codec.dns.DefaultDnsRecordDecoder’,’io.netty5.handler.codec.dns.DnsCodecUtil’, and ‘io.netty5.handler.codec.dns.AbstractDnsMessage’ with high CBO values are tightly coupled to other classes, suggesting they might be highly sensitive to changes in related classes.

2. Lack of Cohesion of Methods (LCOM):



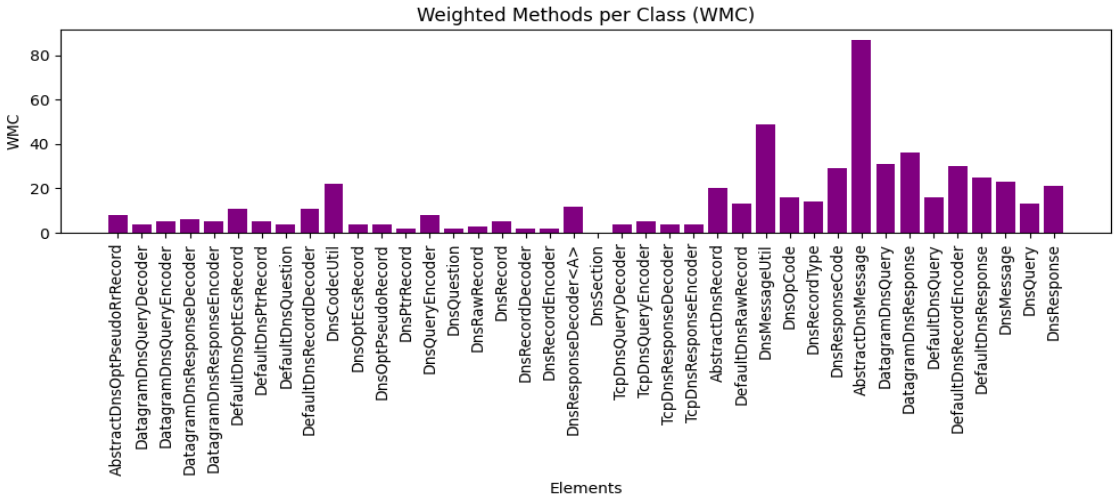
High Lack of Cohesion of Methods (LCOM) suggests that methods within a class do not share or utilize the same set of instance variables effectively, indicating potential design issues. Classes like ‘io.netty5.handler.codec.dns.DefaultDnsOptEcsRecord’ and ‘io.netty5.handler.codec.dns.DnsOpCode’ with significant LCOM values may have methods that are not well-coordinated in performing related tasks, potentially leading to increased complexity.

3. Complexity:



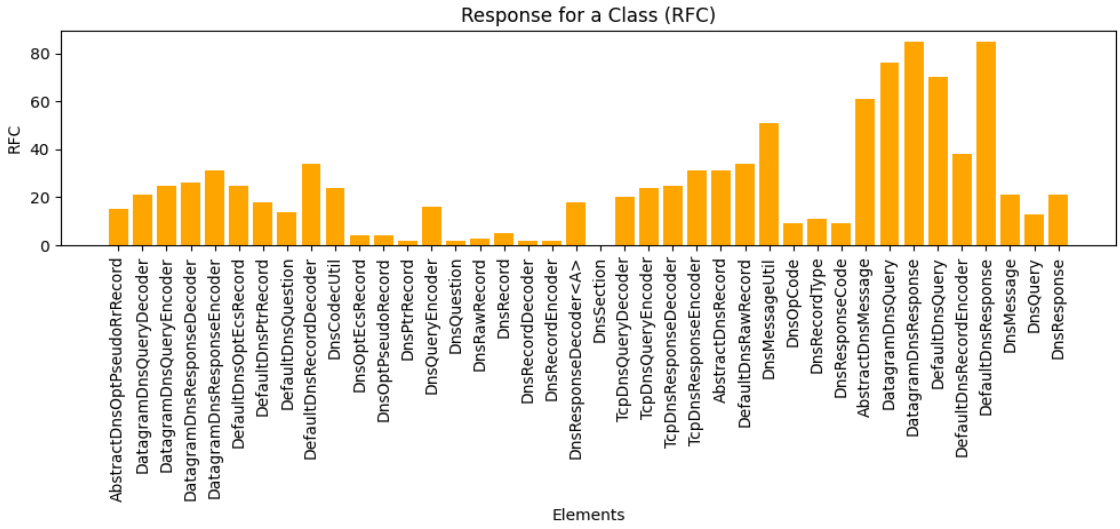
Cyclomatic Complexity measures the number of independent paths through a class's code. Classes such as 'io.netty5.handler.codec.dns.DnsMessageUtil' and 'io.netty5.handler.codec.dns.AbstractDnsRecord' with high Complexity values have more decision points and are more intricate, making them harder to understand and maintain.

4. **Weighted Methods per Class (WMC):**



Weighted Methods per Class (WMC) measures the number of methods in a class and their complexity. Classes like 'io.netty5.handler.codec.dns.DnsCodecUtil' and 'io.netty5.handler.codec.dns.AbstractDnsMessage' with high WMC values have many methods, potentially making the class more complex and harder to manage.

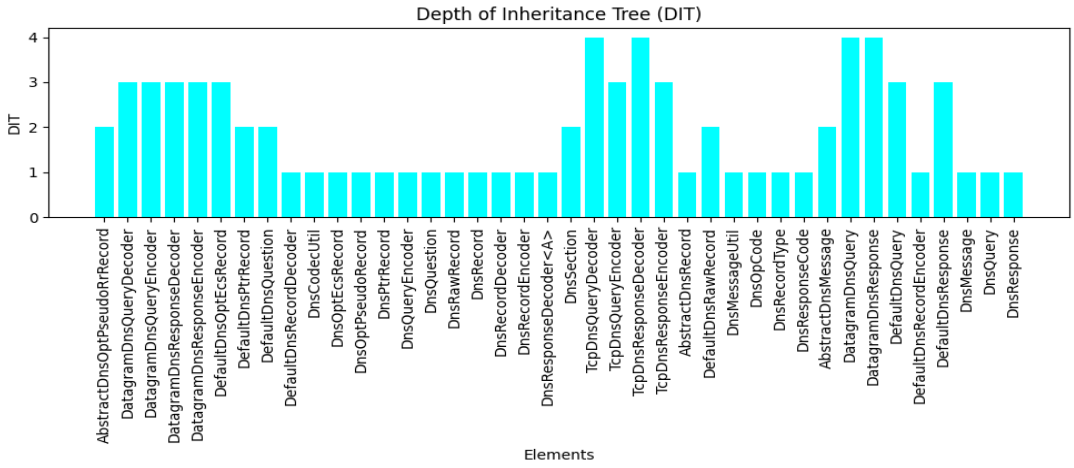
5. **Response for a Class (RFC):**



Response for a Class (RFC) measures the number of methods that can potentially be executed in response to a message received by an object of the class. Classes such as 'io.netty5.handler.codec.dns.DnsMessageUtil' and 'io.netty5.handler.codec.dns.AbstractDnsMessage' with high RFC values interact with

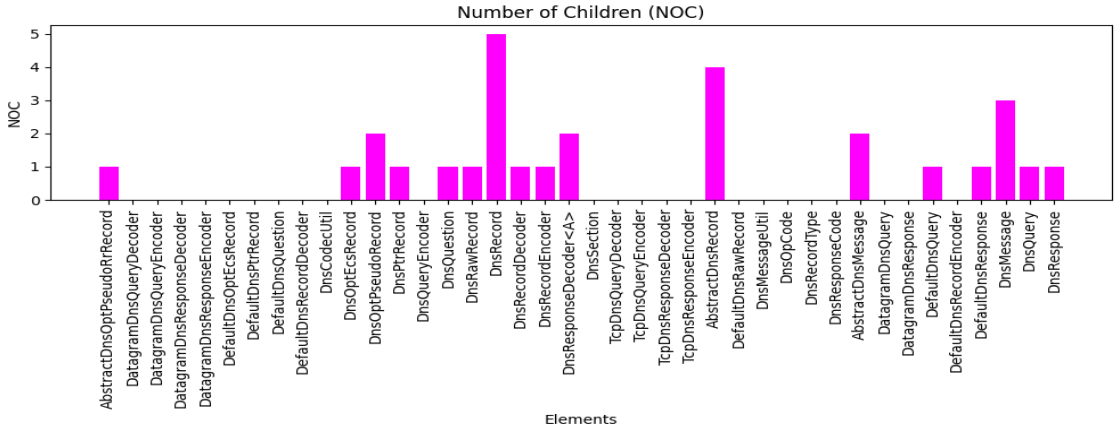
many other classes, indicating complex interactions and potentially higher maintenance costs.

6. Depth of Inheritance Tree (DIT):



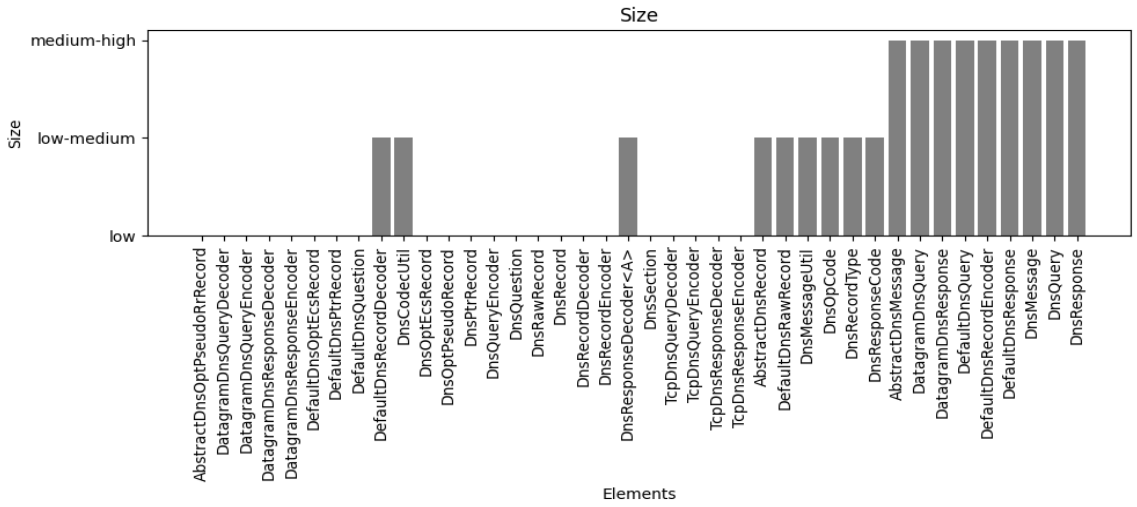
Depth of Inheritance Tree (DIT) measures the number of levels in the inheritance hierarchy for a class. Classes like 'io.netty5.handler.codec.dns.DatagramDnsQuery' and 'io.netty5.handler.codec.dns.AbstractDnsMessage' with high DIT values are deeper in the inheritance tree, which can increase complexity and make it harder to predict behavior and maintain the code.

7. Number of Children (NOC):



Number of Children (NOC) measures the number of immediate subclasses of a class. Classes like 'io.netty5.handler.codec.dns.DnsRecord' and 'io.netty5.handler.codec.dns.DnsResponse' with high NOC values have many subclasses, potentially increasing reuse but also complexity.

8. **Size:**



Size generally refers to the total lines of code or the overall complexity of a class. Although the provided data has 'low' values for size across classes, larger sizes can indicate more complex classes that may be harder to understand and maintain.

4.2. Project: Apache Dubbo

Module: dubbo-compiler

The Values below are Obtained from CodeMR Tool:

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
AbstractGenerator	15	0	low-medium	38	73	2	7	low-medium
Dubbo3Generator	1	0	low-medium	8	9	3	0	low
DubboGenerator	1	0	low-medium	4	5	3	0	low
DubboGrpcGenerator	1	0	low-medium	4	5	3	0	low
ReactorDubboGrpcGenerator	1	0	low-medium	4	5	3	0	low

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
RxDubboGrpcGenerator	1	0	low-medium	4	5	3	0	low
Dubbo3TripleGenerator	1	0	low-medium	8	9	3	0	low
ReactorDubbo3TripleGenerator	1	0	low-medium	8	9	3	0	low

JDeodorant obtained results are below:

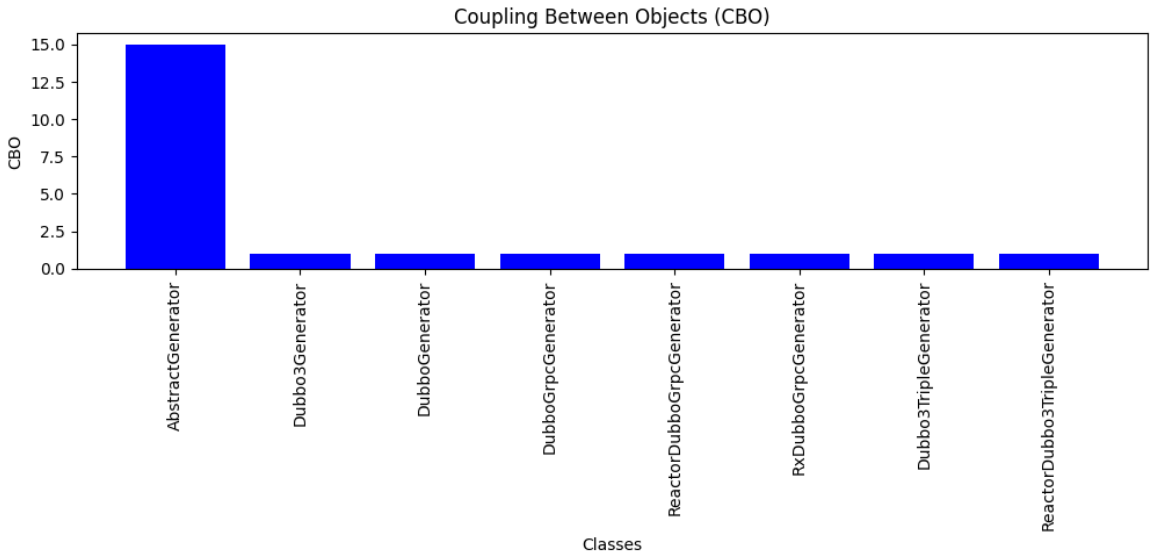
Classes with Code Bad Smells:

AbstractGenerator

Classes without Code Bad Smells:

Dubbo3Generator, DubboGenerator, DubboGrpcGenerator, ReactorDubboGrpcGenerator, RxDubboGrpcGenerator, Dubbo3TripleGenerator, ReactorDubbo3TripleGenerator

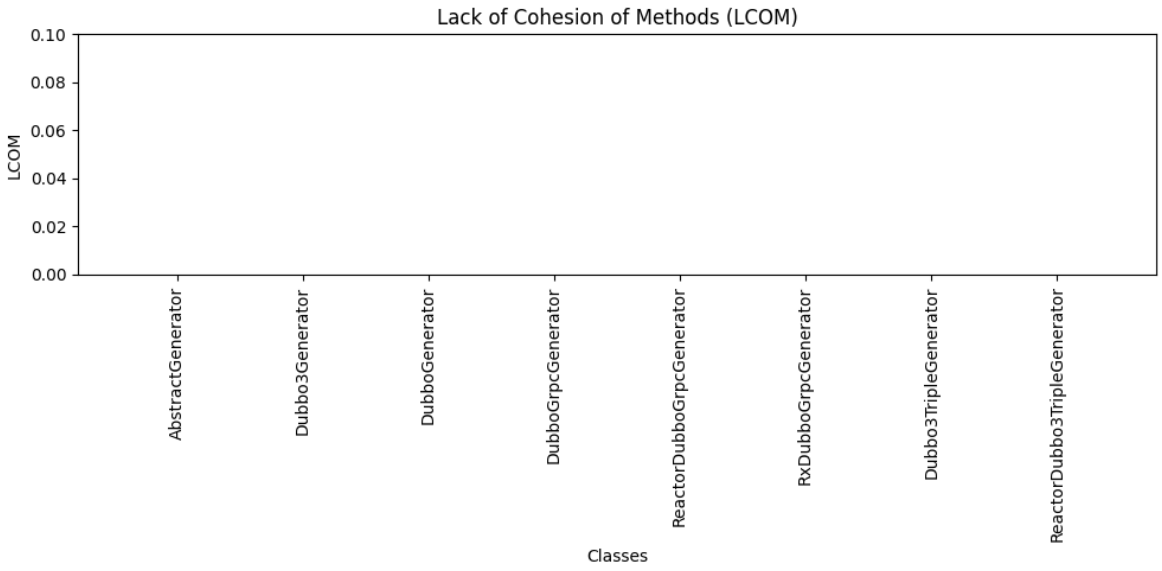
1. **Coupling Between Objects (CBO):**



High CBO values indicate that a class is highly dependent on other classes, which can make it more difficult to maintain and understand. For instance, AbstractGenerator with a CBO of 15 is highly coupled, whereas the other classes (Dubbo3Generator, DubboGenerator, DubboGrpcGenerator, ReactorDubboGrpcGenerator,

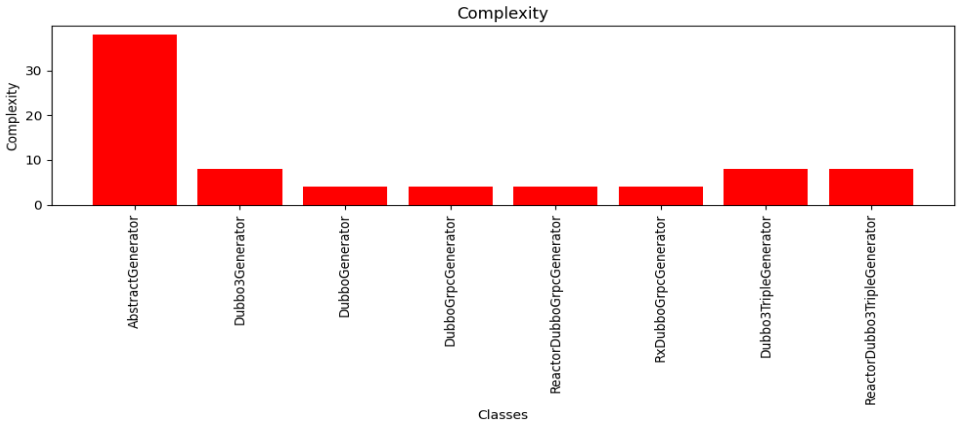
RxDubboGrpcGenerator, Dubbo3TripleGenerator, ReactorDubbo3TripleGenerator) have a low coupling (CBO of 1).

2. **Lack of Cohesion of Methods (LCOM):**



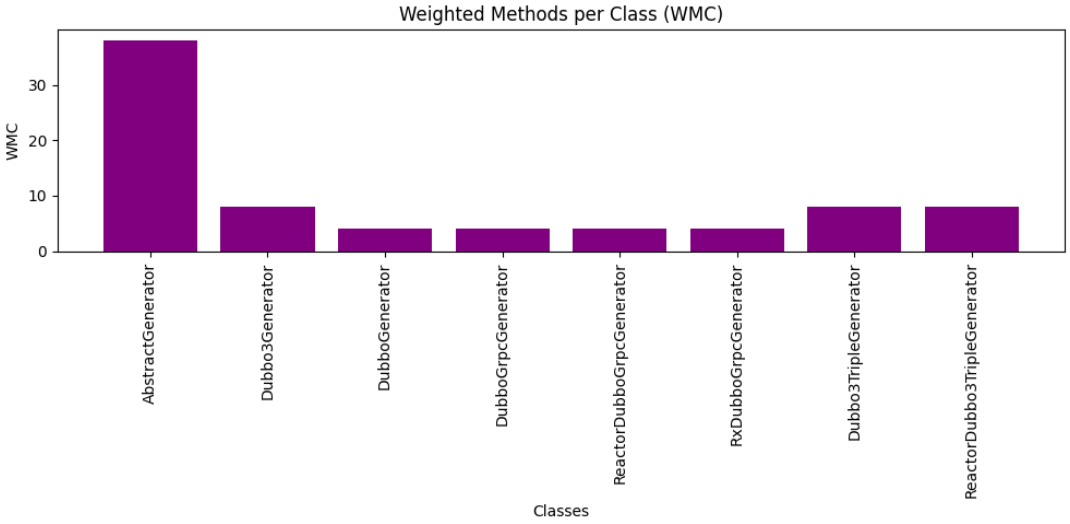
LCOM measures the lack of cohesion in a class. A low LCOM value indicates that the methods of a class are well-cohesive. All classes in the provided data have an LCOM of 0, indicating good cohesion among the methods.

3. **Complexity:**



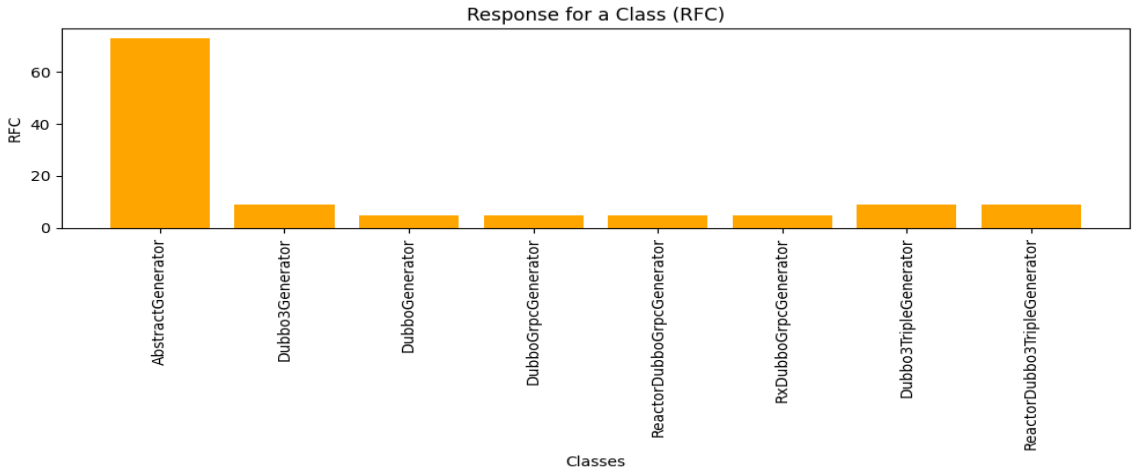
Complexity indicates the complexity of a class in terms of its control flow. The classes here are marked as "low-medium," suggesting moderate complexity. For example, AbstractGenerator has a complexity level that is considered low-medium.

4. **Weighted Methods per Class (WMC):**



WMC measures the number of methods in a class and their complexity. A higher WMC value, like 38 for AbstractGenerator, indicates that the class has many methods and potentially high complexity. Other classes have lower WMC values, such as Dubbo3Generator with 8.

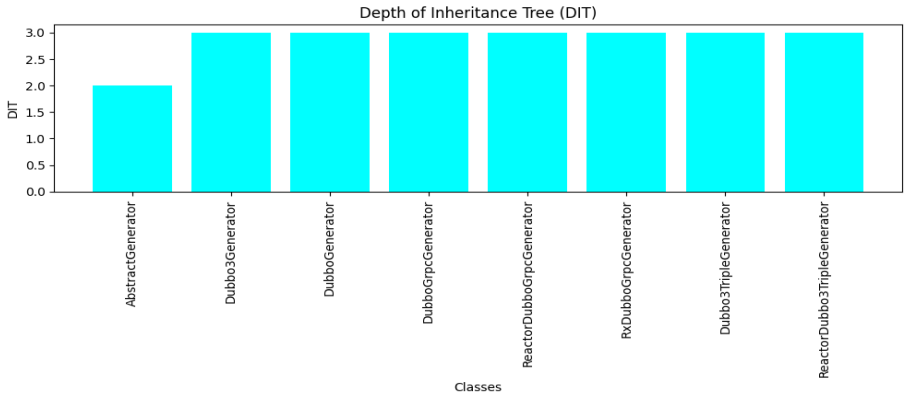
5. **Response for a Class (RFC):**



RFC measures the number of methods that can be invoked in response to a message to the object. AbstractGenerator has a high RFC of 73, indicating it interacts with many

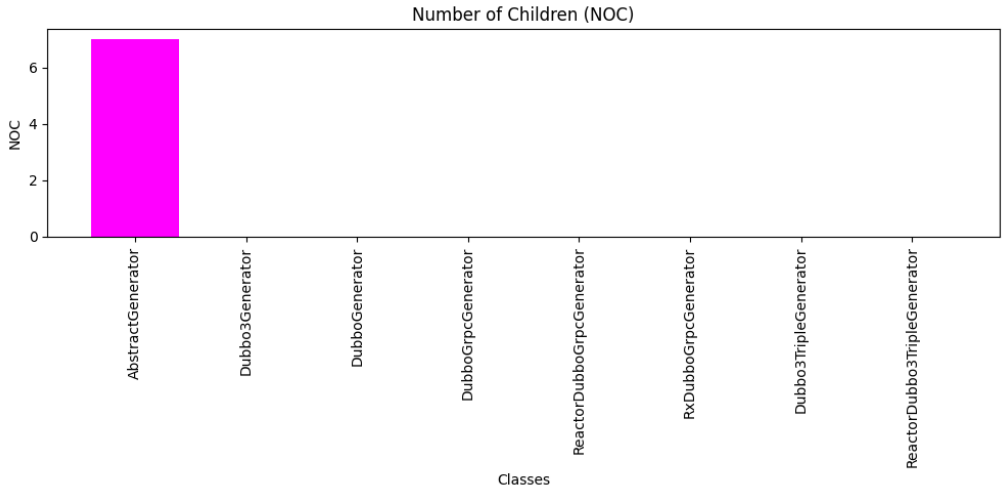
other methods, making it more complex and harder to maintain. The other classes have lower RFC values, such as 9 for Dubbo3Generator.

6. **Depth of Inheritance Tree (DIT):**



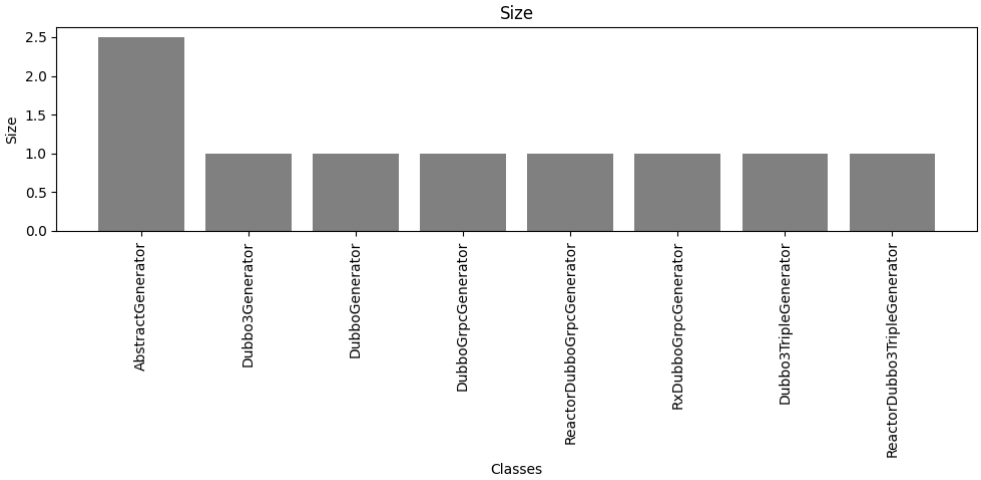
DIT measures the inheritance levels from the object hierarchy top. A higher DIT value can indicate greater complexity and potential for inheritance issues. For example, Dubbo3Generator, DubboGenerator, DubboGrpcGenerator, ReactorDubboGrpcGenerator, RxDubboGrpcGenerator, Dubbo3TripleGenerator, ReactorDubbo3TripleGenerator all have a DIT of 3, while AbstractGenerator has a DIT of 2.

7. **Number of Children (NOC):**



NOC measures the number of subclasses inheriting from a class. A higher NOC can indicate a well-used base class but also increases complexity. AbstractGenerator has 7 subclasses, whereas other classes have 0.

8. **Size:**



Size typically refers to lines of code or the number of attributes and methods in a class. Here, sizes are marked as "low-medium," indicating moderate size for AbstractGenerator and low for the other classes.

4.3. Project: Design patterns implemented in Java
Module: abstract-factory

The Values below are Obtained from CodeMR Tool:

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
App	4	0	low	3	7	1	0	low
Army	0	0	low	1	1	1	2	low
Castle	0	0	low	1	1	1	2	low
ElfArmy	0	0	low	1	1	1	0	low
ElfCastle	0	0	low	1	1	1	0	low
ElfKing	0	0	low	1	1	1	0	low
ElfKingdomFactory	6	0	low	3	3	1	0	low
King	0	0	low	1	1	1	2	low
Kingdom	3	0	low	0	0	1	0	low
KingdomFactory	3	0	low	3	3	1	2	low
OrcArmy	0	0	low	1	1	1	0	low
OrcCastle	0	0	low	1	1	1	0	low
OrcKing	0	0	low	1	1	1	0	low

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
OrcKingdomFactory	6	0	low	3	3	1	0	low

JDeodorant obtained results are below:

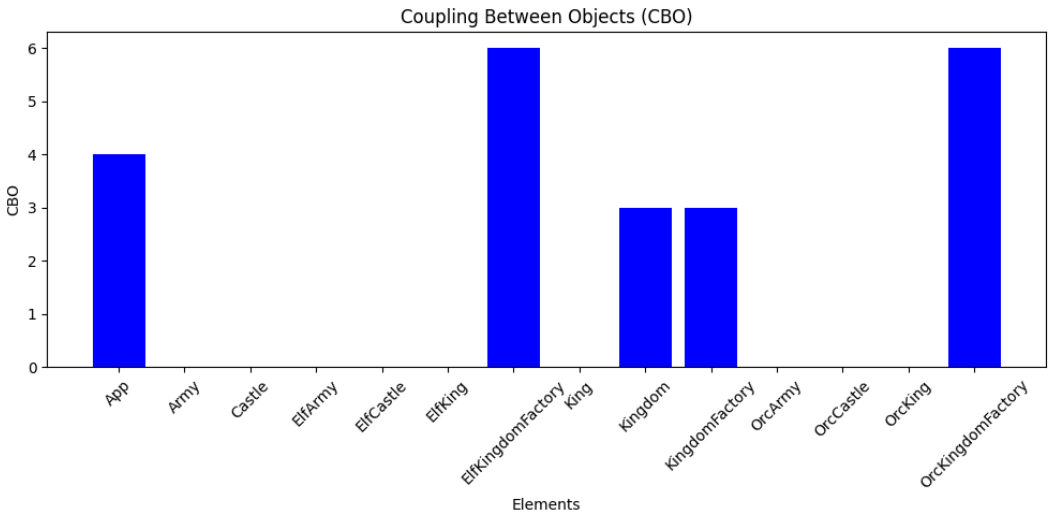
Classes with Code Bad Smells:

ElfKingdomFactory, KingdomFactory

Classes without Code Bad Smells:

App, Army, Castle, ElfArmy, ElfCastle, ElfKing, King, Kingdom, OrcArmy, OrcCastle, OrcKing, OrcKingdomFactory

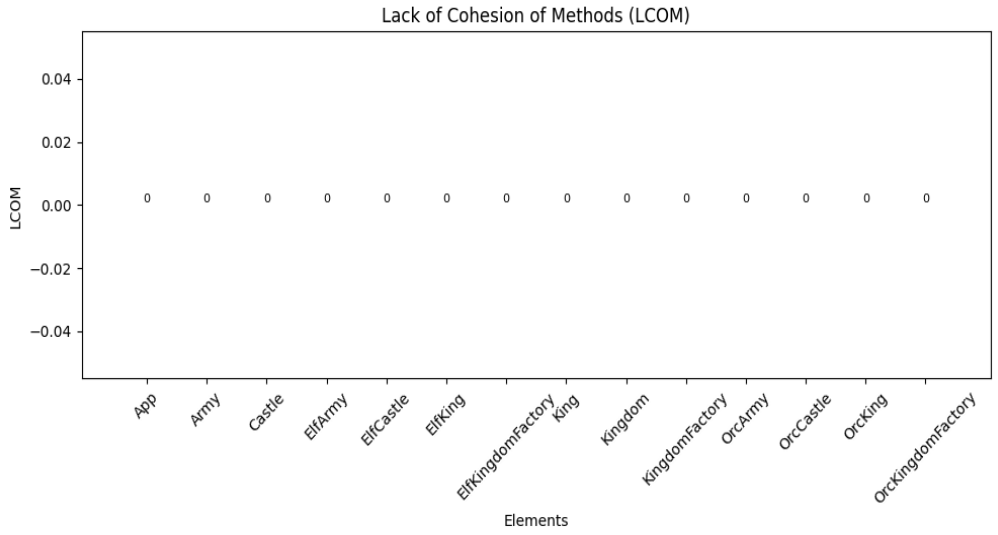
1. **Coupling Between Objects (CBO):**



Coupling Between Objects (CBO) measures how many other classes a particular class is directly connected to. In the context of the given classes ('App', 'Army', 'Castle', 'ElfArmy', 'ElfCastle', 'ElfKing', 'ElfKingdomFactory', 'King', 'Kingdom', 'KingdomFactory', 'OrcArmy', 'OrcCastle', 'OrcKing', 'OrcKingdomFactory'), higher CBO values indicate that these classes are tightly coupled with other classes. This tight coupling can lead to increased dependencies and make the codebase more complex and harder to maintain. Classes with high CBO may require changes in multiple places if

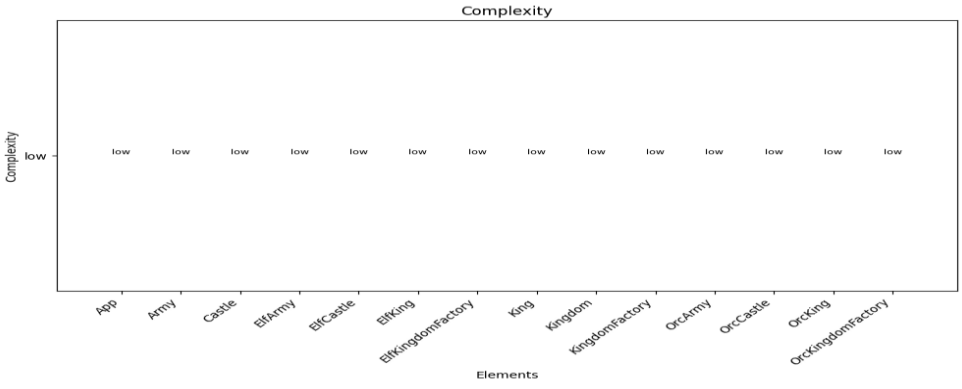
modifications are made, which can introduce risks and increase the effort required for testing and debugging.

2. Lack of Cohesion of Methods (LCOM):



A high LCOM value indicates that the methods within the class do not share state or work independently, which can suggest poor encapsulation and potential design issues. In the given classes, observing LCOM helps in identifying classes ('App', 'Army', 'Castle', 'ElfArmy', 'ElfCastle', 'ElfKing', 'ElfKingdomFactory', 'King', 'Kingdom', 'KingdomFactory', 'OrcArmy', 'OrcCastle', 'OrcKing', 'OrcKingdomFactory') where methods may need to be refactored to improve cohesion, thereby enhancing code clarity and maintainability.

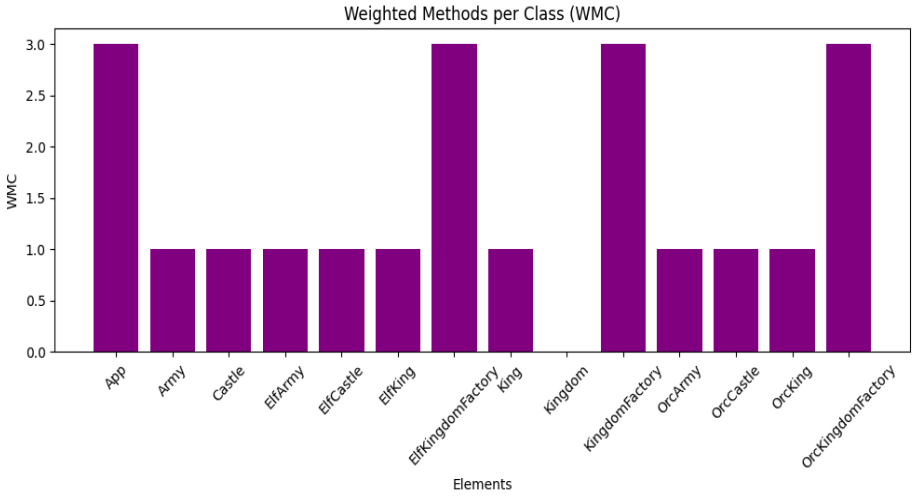
3. Complexity:



When applied to classes ('App', 'Army', 'Castle', 'ElfArmy', 'ElfCastle', 'ElfKing', 'ElfKingdomFactory', 'King', 'Kingdom', 'KingdomFactory', 'OrcArmy', 'OrcCastle',

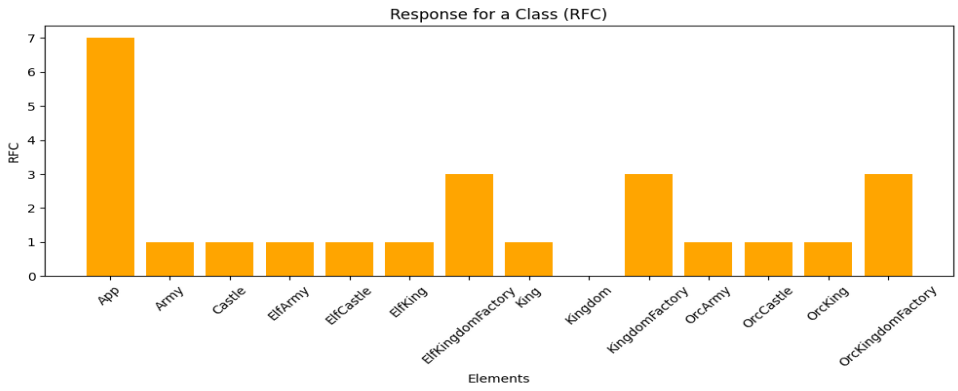
'OrcKing', 'OrcKingdomFactory'), high complexity values indicate that the class methods have many decision points and branches. This complexity can make the code harder to understand, test, and maintain. Monitoring cyclomatic complexity helps in identifying classes where methods are overly complex, suggesting potential areas for simplification or decomposition into smaller, more manageable units.

4. **Weighted Methods per Class (WMC):**



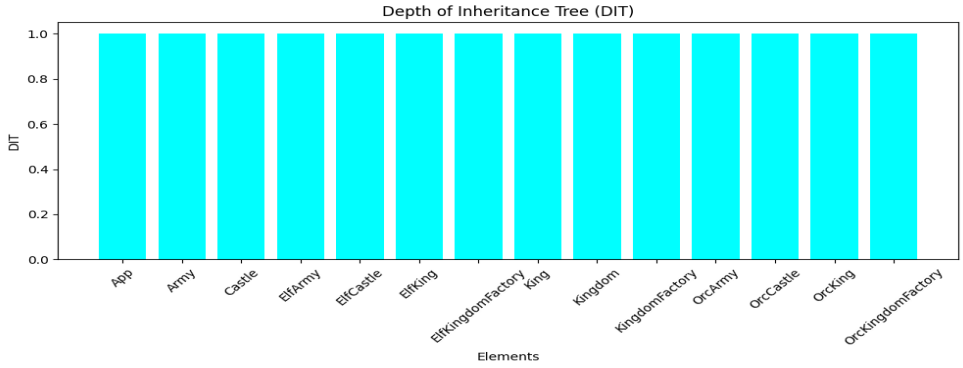
In the provided classes ('App', 'Army', 'Castle', 'ElfArmy', 'ElfCastle', 'ElfKing', 'ElfKingdomFactory', 'King', 'Kingdom', 'KingdomFactory', 'OrcArmy', 'OrcCastle', 'OrcKing', 'OrcKingdomFactory'), high WMC values indicate that these classes have a significant number of methods, potentially with varying levels of complexity. Classes with high WMC values may be harder to maintain and test due to their size and the complexity of their methods. Managing WMC helps in maintaining class cohesion and ensuring that classes adhere to the Single Responsibility Principle (SRP).

5. **Response for a Class (RFC):**



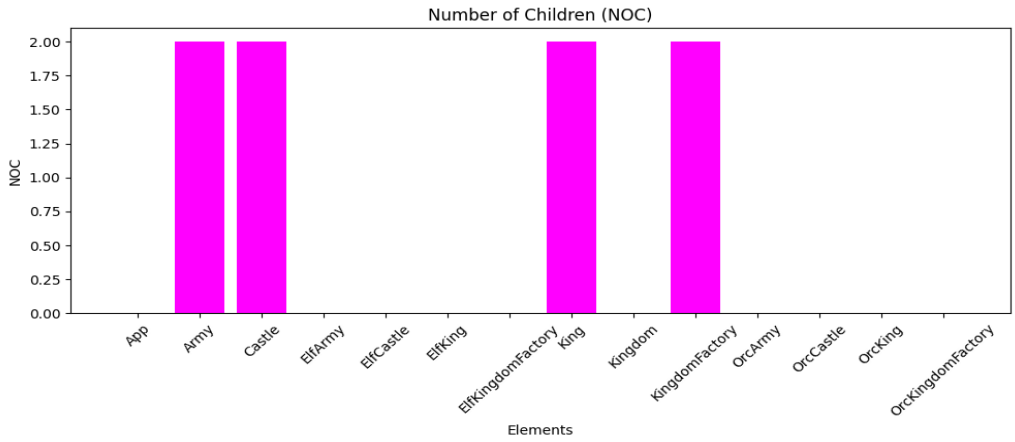
High RFC values for classes ('App', 'Army', 'Castle', 'ElfArmy', 'ElfCastle', 'ElfKing', 'ElfKingdomFactory', 'King', 'Kingdom', 'KingdomFactory', 'OrcArmy', 'OrcCastle', 'OrcKing', 'OrcKingdomFactory') indicate complex interactions and dependencies. Classes with high RFC may have numerous interactions with other classes or external systems, which can increase the complexity and potential for errors. Monitoring RFC helps in identifying classes with extensive responsibilities and interactions, which may require refactoring to improve clarity and reduce coupling.

6. Depth of Inheritance Tree (DIT):



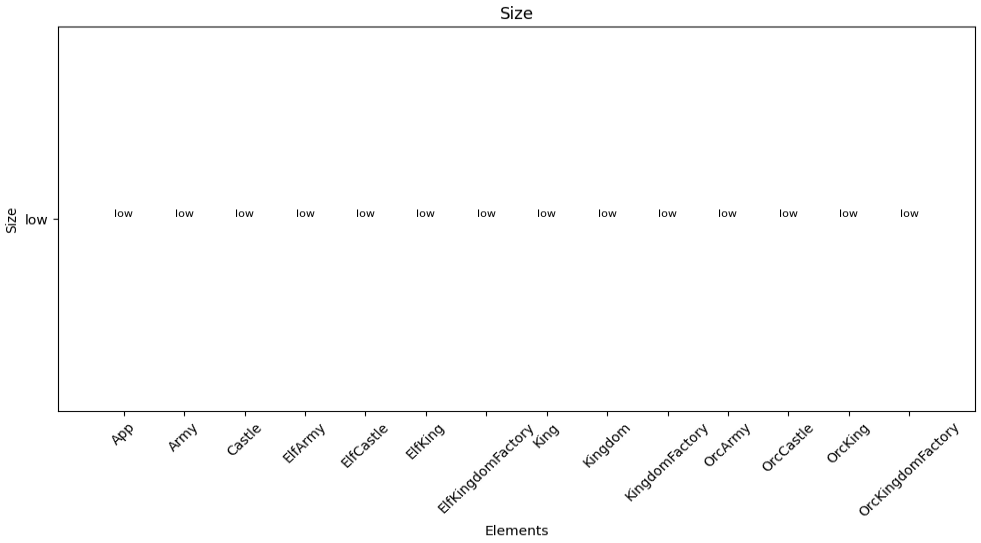
Higher DIT values for classes ('App', 'Army', 'Castle', 'ElfArmy', 'ElfCastle', 'ElfKing', 'ElfKingdomFactory', 'King', 'Kingdom', 'KingdomFactory', 'OrcArmy', 'OrcCastle', 'OrcKing', 'OrcKingdomFactory') indicate that these classes are deeper in the inheritance hierarchy. Deep inheritance hierarchies can lead to increased complexity and tighter coupling between classes. Classes with high DIT may inherit behavior and attributes from multiple ancestor classes, which can complicate maintenance and introduce unexpected behaviors.

7. Number of Children (NOC):



Number of Children (NOC) measures the number of immediate subclasses of a class. Classes like 'io.netty5.handler.codec.dns.DnsRecord' and 'io.netty5.handler.codec.dns.DnsResponse' with high NOC values have many subclasses, potentially increasing reuse but also complexity.

8. **Size:**



In the context of the provided classes ('App', 'Army', 'Castle', 'ElfArmy', 'ElfCastle', 'ElfKing', 'ElfKingdomFactory', 'King', 'Kingdom', 'KingdomFactory', 'OrcArmy', 'OrcCastle', 'OrcKing', 'OrcKingdomFactory') large size values indicate that these classes are relatively large and potentially complex. Large classes can be harder to understand, maintain, and test. They may indicate that a class is taking on too many responsibilities or not adhering to principles of good design, such as the Single Responsibility Principle (SRP). Managing class size helps in improving code readability, maintainability, and modularity.

4.4. Project: Extensible build system
Module: bazel

The Values below are Obtained from CodeMR Tool:

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
A	1	0	low	0	0	1	0	low
B	0	0	low	0	0	1	0	low
Client	14	0	low-medium	5	16	2	0	low-medium

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
InvokePolymorphic	0	0	low	1	2	1	0	low
Library	0	0	low	0	0	1	1	low
LibraryAnnotations	0	0	low	0	0	1	0	low
LibraryException	0	0	low-medium	0	0	3	0	low
LibraryInterface	0	0	low	0	0	1	1	low
OneAnnotationValue	0	0	low	1	1	1	0	low
StringAnnotation	0	0	low	1	1	1	0	low

JDeodorant obtained results are below:

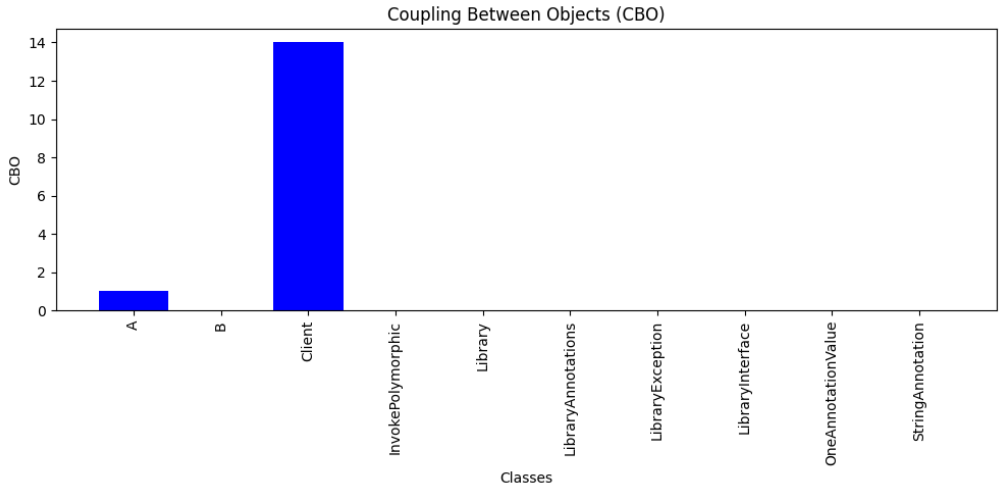
Classes with Code Bad Smells:

Client

Classes without Code Bad Smells:

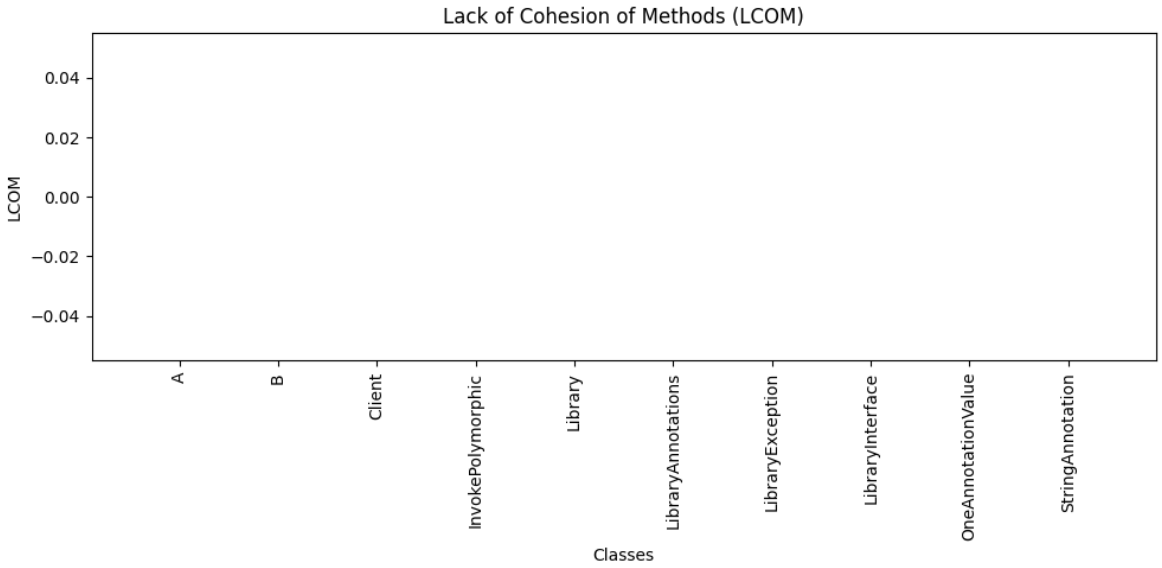
A, B, InvokePolymorphic, Library, LibraryAnnotations, LibraryException, LibraryInterface, OneAnnotationValue, StringAnnotation

1. **Coupling Between Objects (CBO):**



Coupling Between Objects (CBO) measures the degree of interdependence between classes. A high CBO value, such as in the Client class (CBO=14), indicates a high degree of coupling, making the class more complex and harder to maintain.

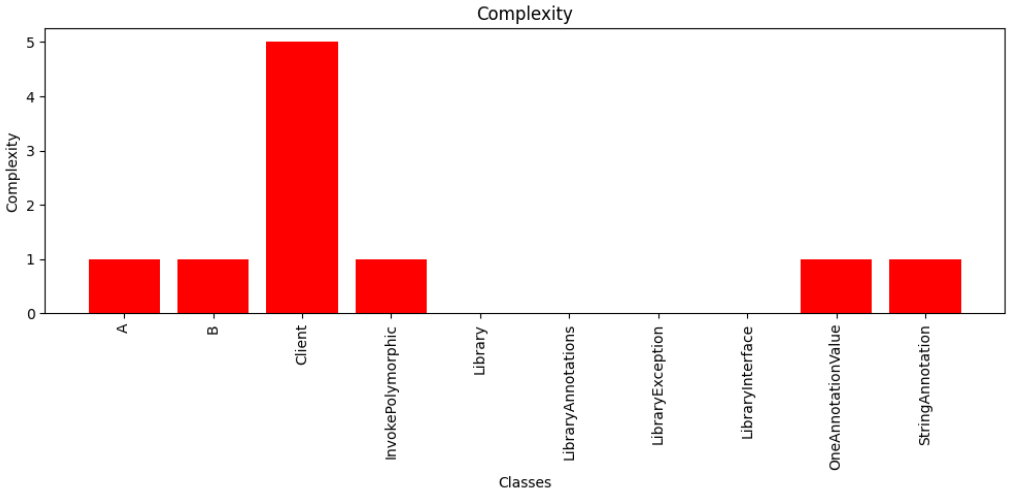
2. Lack of Cohesion of Methods (LCOM):



Lack of Cohesion of Methods (LCOM) measures how related the methods within a class are. In this data, all classes have an LCOM of 0, indicating high cohesion.

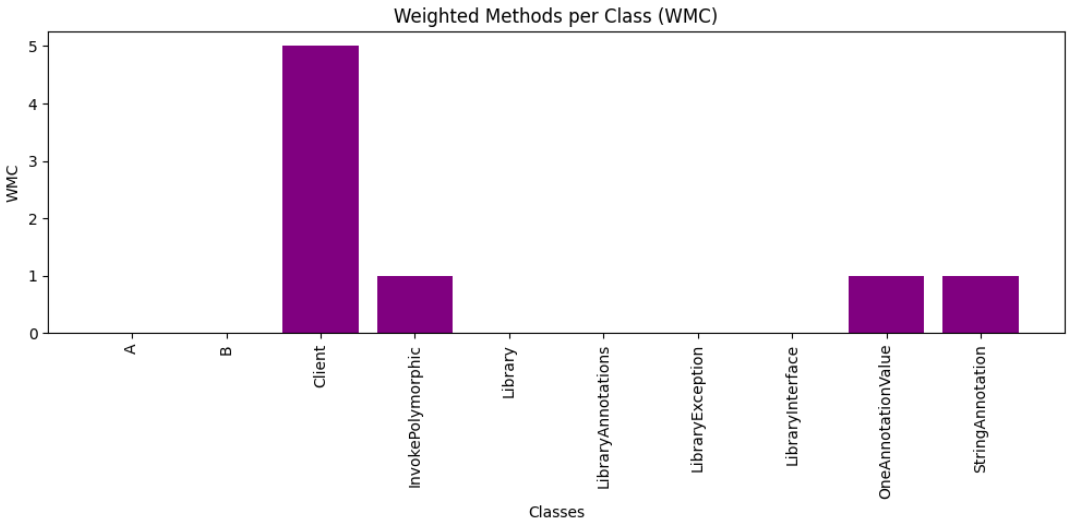
3. Complexity:

Approximating low to 1 and low-medium to 5



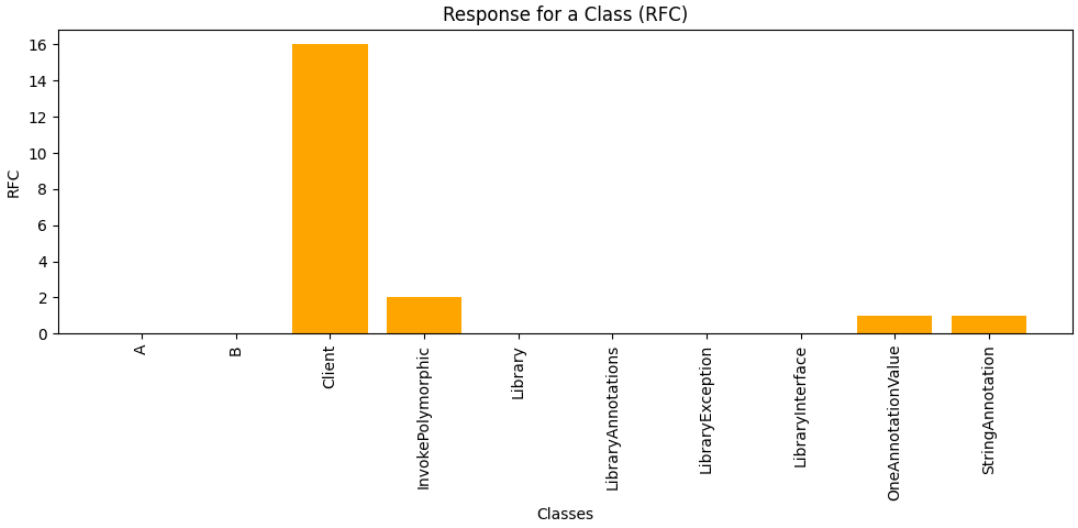
Complexity measures the number of linearly independent paths through a method. Classes with low complexity, such as Library and LibraryAnnotations, are easier to test and maintain.

4. Weighted Methods per Class (WMC):



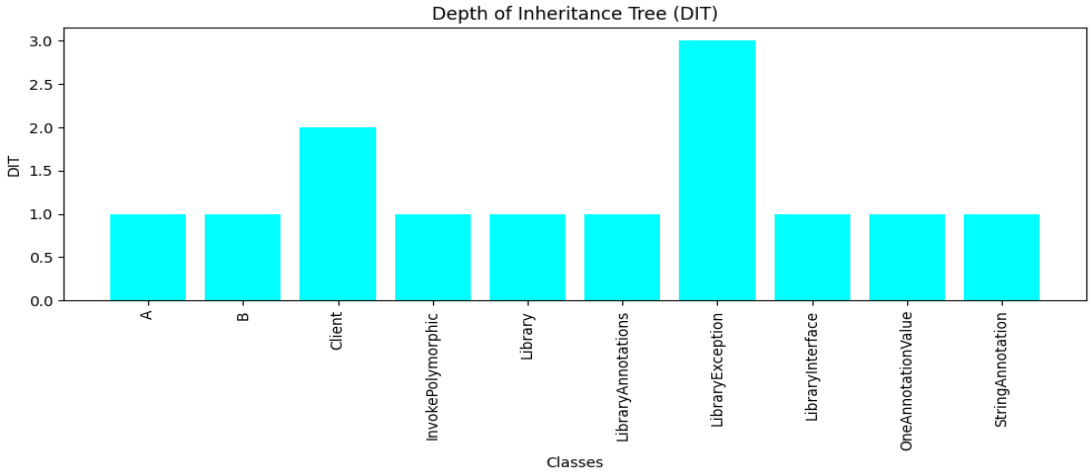
Weighted Methods per Class (WMC) counts the number of methods in a class, weighted by their complexity. For example, Client has a WMC of 5, indicating a moderate number of methods.

5. Response for a Class (RFC):



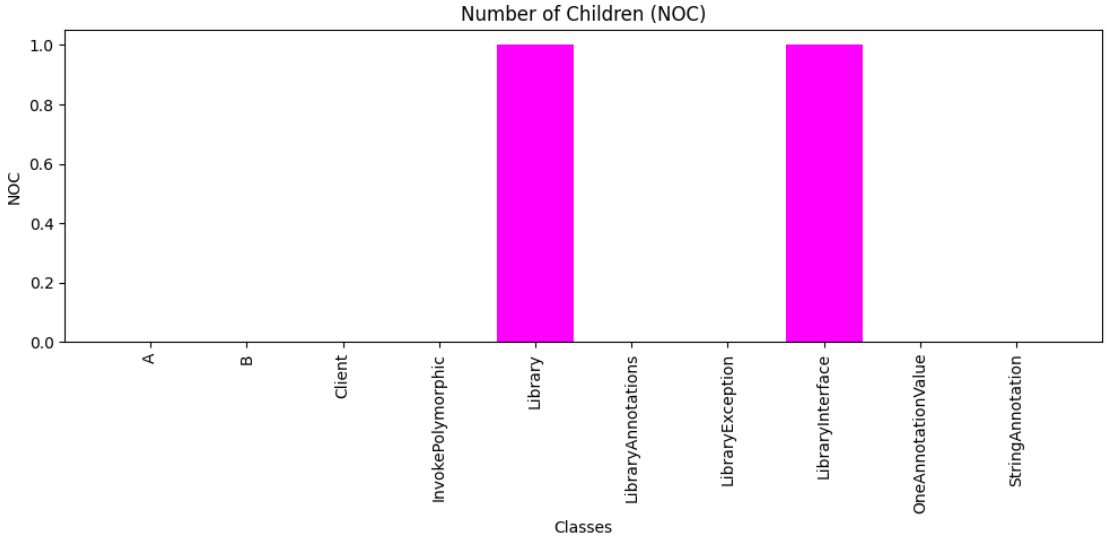
Response for a Class (RFC) measures the number of methods that can be executed in response to a message received by an object of that class. The Client class has a higher RFC, indicating more potential interactions.

6. Depth of Inheritance Tree (DIT):



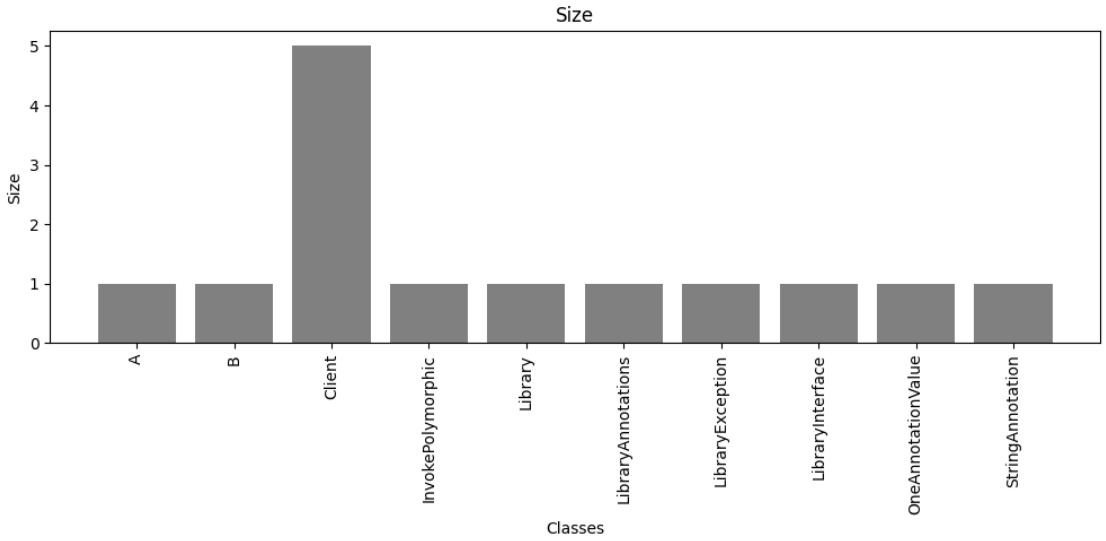
Depth of Inheritance Tree (DIT) measures the inheritance levels from the object hierarchy top. A higher DIT, such as in LibraryException (DIT=3), may indicate increased complexity and potential for reuse.

7. Number of Children (NOC):



Number of Children (NOC) measures the number of immediate subclasses of a class. A high NOC value, as in AbstractGenerator (NOC=7), suggests that the class is a parent to many other classes, potentially increasing its complexity.

8. **Size:**
   Approximating low to 1 and low-medium to 5



Size measures the number of attributes and methods in a class. Smaller classes, such as Library and LibraryAnnotations, are generally easier to manage.

4.5. Project: Apache OpenNLP
Module: opennlp-uima

The Values below are Obtained from CodeMR Tool:

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
ChunkerModelResource	1	0	low	1	1	1	1	low
ChunkerModelResourceImpl	1	0	low-medium	2	2	2	0	low
DictionaryResource	1	0	low	1	1	1	1	low
DictionaryResourceImpl	1	0	low-medium	2	2	2	0	low
AbstractDocumentCategorizer	17	0.5	medium-high	7	28	4	2	low
DoccatModelResource	1	0	low	1	1	1	1	low
DoccatModelResourceImpl	1	0	low-medium	2	2	2	0	low

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
DocumentCategorizer	10	0	medium-high	3	34	5	0	low
LanguageDetector	1	0	medium-high	1	2	5	0	low
AbstractNameFinder	15	0.667	medium-high	24	44	4	2	low-medium
DictionaryNameFinder	12	0	medium-high	6	20	5	0	low
NameFinder	16	1	medium-high	13	50	5	0	low-medium
TokenNameFinderModelResource	1	0	low	1	1	1	1	low
TokenNameFinderModelResourceImpl	1	0	low-medium	2	2	2	0	low
Normalizer	18	0.583	medium-high	22	54	4	0	low-medium
NumberUtil	0	0	low	5	8	1	0	low
StringDictionary	2	0	low	6	12	1	0	low
Parser	22	0.846	medium-high	14	66	4	0	low-medium
ParserModelResource	1	0	low	1	1	1	1	low
ParserModelResourceImpl	1	0	low-medium	2	2	2	0	low
POSModelResource	1	0	low	1	1	1	1	low
POSModelResourceImpl	1	0	low-medium	2	2	2	0	low
POSTagger	17	0.619	medium-high	17	50	4	0	low-medium
AbstractSentenceDetector	12	0.5	medium-high	12	36	4	1	low
SentenceDetector	13	0.5	medium-high	9	37	5	0	low

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
SentenceModelResource	1	0	low	1	1	1	1	low
SentenceModelResourceImpl	1	0	low-medium	2	2	2	0	low
AbstractTokenizer	12	0.667	medium-high	13	41	4	3	low-medium
SimpleTokenizer	4	0	medium-high	2	4	5	0	low
Tokenizer	13	0.5	medium-high	9	37	5	0	low
TokenizerModelResource	1	0	low	1	1	1	1	low
TokenizerModelResourceImpl	1	0	low-medium	2	2	2	0	low
WhitespaceTokenizer	4	0	medium-high	2	4	5	0	low
AbstractModelResource	4	0	low	3	4	1	8	low
AnnotationComboIterator	6	0.667	low	7	14	1	0	low-medium
AnnotationComparator	1	0	low	1	2	1	0	low
AnnotationIteratorPair	1	0.5	low	3	3	1	0	low
AnnotatorUtil	11	0	medium-high	55	38	1	0	low-medium
ContainingConstraint	2	1	low	7	7	1	0	low
ExceptionMessages	0	0	low	0	0	1	0	low
OpenNlpAnnotatorProcessException	1	0	medium-high	3	3	6	0	low
OpennlpUtil	5	0	low	10	12	1	0	low
UimaUtil	5	0	low	4	9	1	0	low

JDeodorant obtained results are below:

Classes with Code Bad Smells:

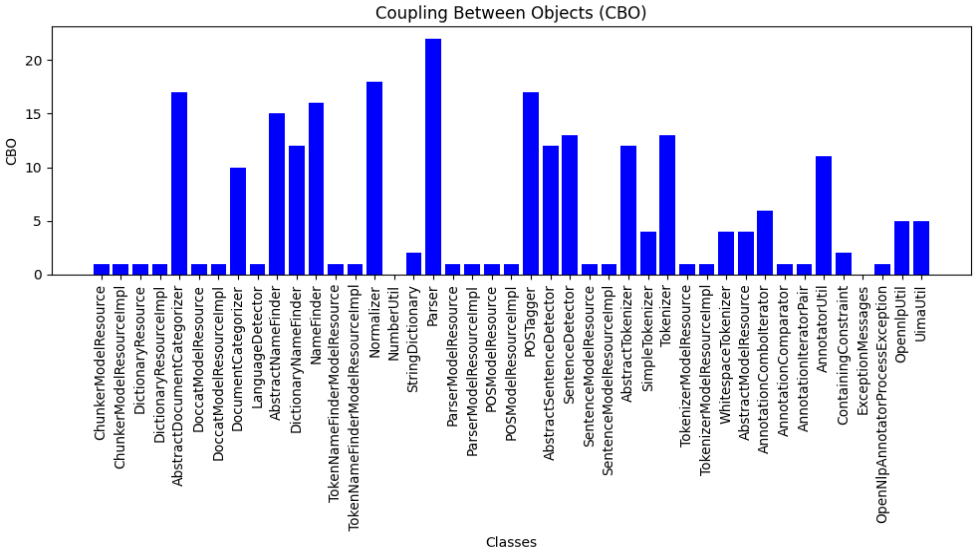
Classes with Potential Code Smells

AbstractDocumentCategorizer, AbstractNameFinder, NameFinder, Parser, AnnotatorUtil

Classes without Code Bad Smells:

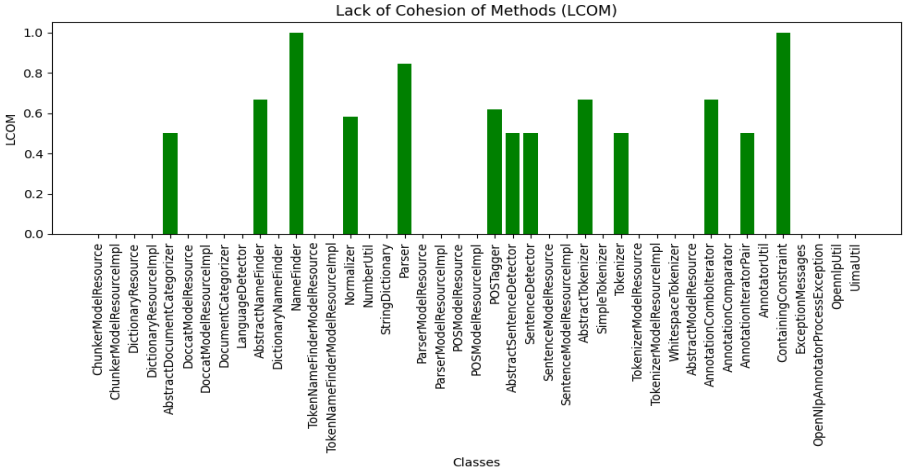
ChunkerModelResource, DictionaryResource, TokenNameFinderModelResource, ParserModelResource, POSModelResource, ExceptionMessages, AnnotationComparator, AnnotationIteratorPair, ContainingConstraint, OpennlpUtil, UimaUtil

1. **Coupling Between Objects (CBO):**



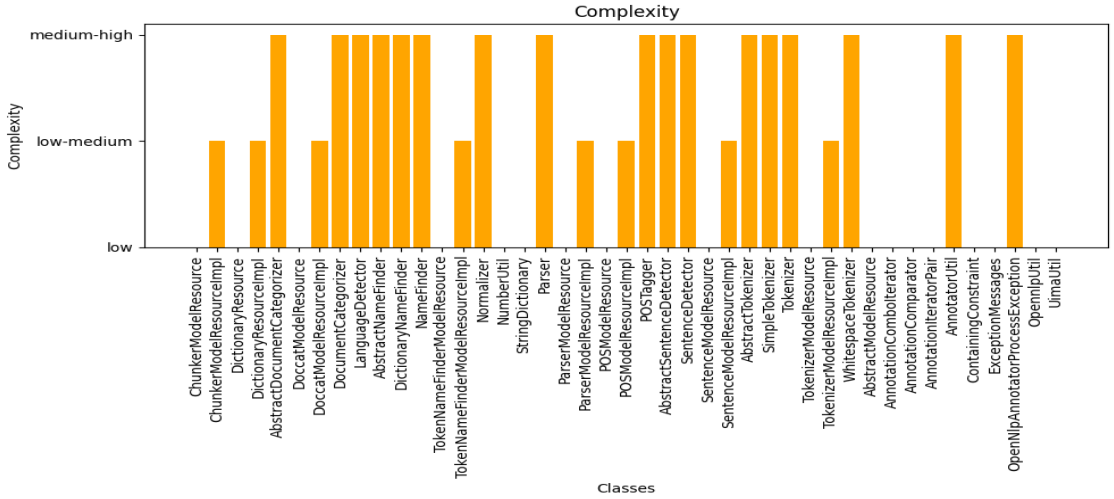
CBO measures how many other classes a class is coupled to. For example, AbstractDocumentCategorizer has a CBO of 17.

2. **Lack of Cohesion of Methods (LCOM):**



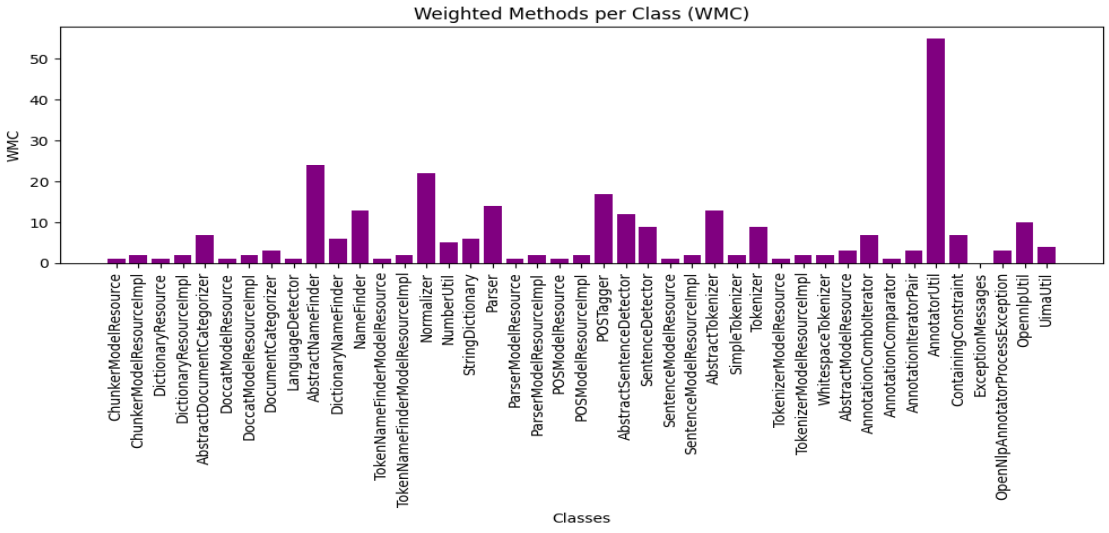
LCOM measures the lack of cohesion among methods in a class. For instance, AbstractDocumentCategorizer has an LCOM of 0.5.

3. Complexity:



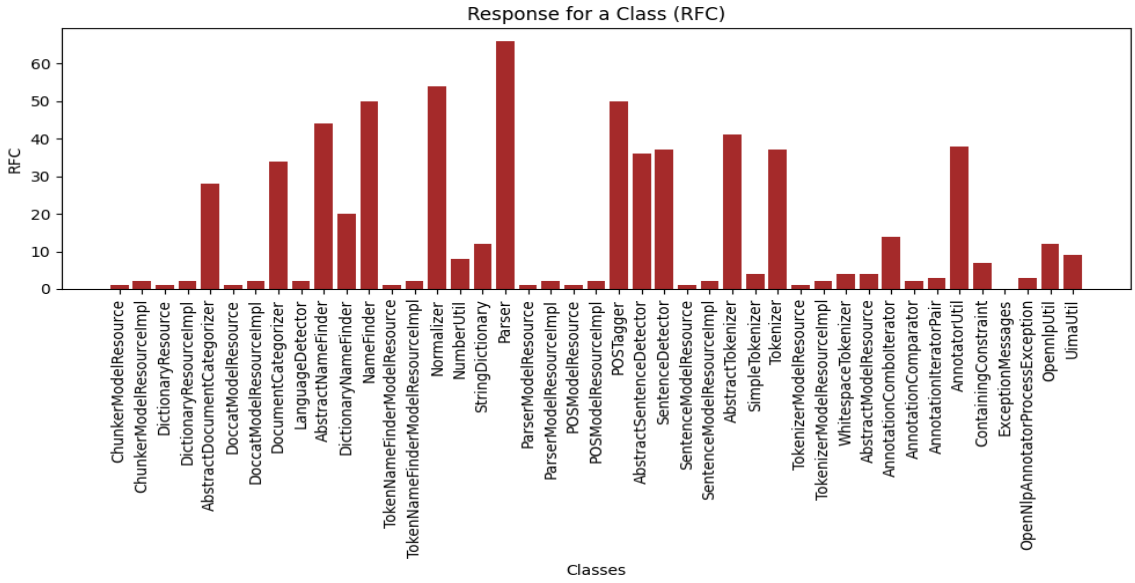
This parameter categorizes the complexity of the class, often subjective and based on various metrics. AbstractDocumentCategorizer is classified as 'medium-high' complexity.

4. Weighted Methods per Class (WMC):



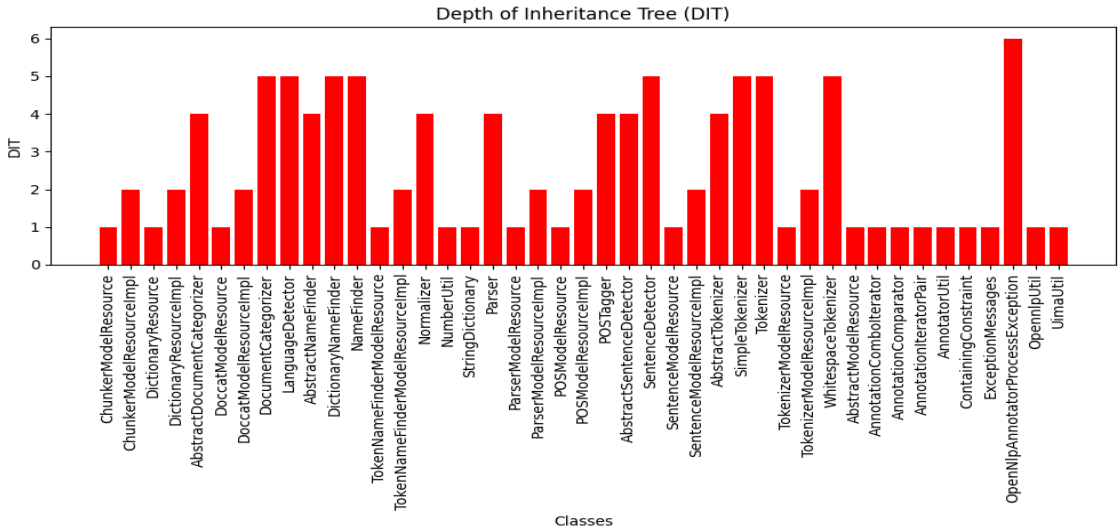
WMC counts the number of methods in a class weighted by their complexity. AbstractDocumentCategorizer has a WMC of 7.

5. Response for a Class (RFC):



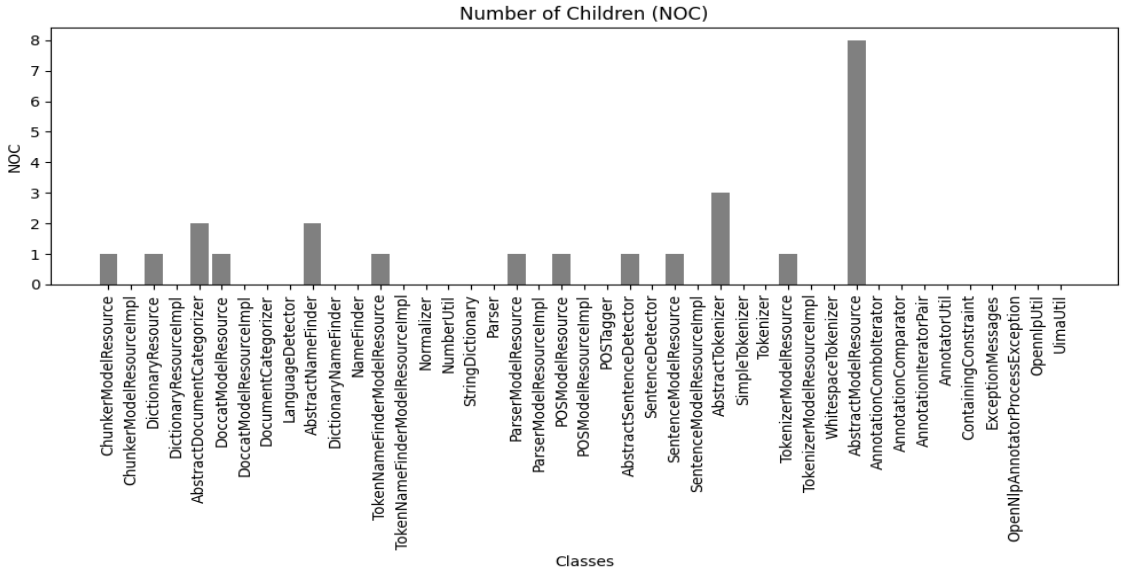
RFC counts the number of methods that can potentially be executed in response to a message received by an object of the class. AbstractDocumentCategorizer has an RFC of 28.

6. Depth of Inheritance Tree (DIT):



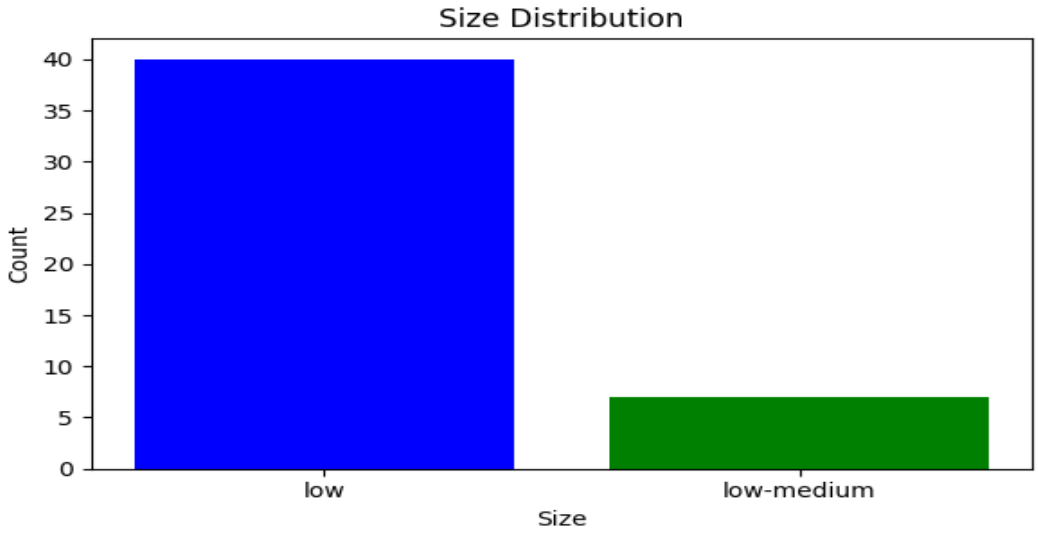
DIT measures the number of classes from which the class inherits (including its own superclass). AbstractDocumentCategorizer has a DIT of 4.

7. Number of Children (NOC):



NOC counts the number of immediate subclasses or children a class has. AbstractDocumentCategorizer has NOC of 2.

8. Size:



Size categorizes the size or complexity of the class. AbstractDocumentCategorizer has a size of 'low-medium'.

4.6. Project: Mirror of Apache Struts
Module: rest-showcase

The Values below are Obtained from CodeMR Tool:

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
IndexController	0	0	low	1	1	1	0	low
Order	3	0.444	low	14	21	1	0	low
OrdersController	5	0.657	low-medium	15	29	2	0	low
OrdersService	1	0.5	low	5	11	1	0	low

JDeodorant obtained results are below:

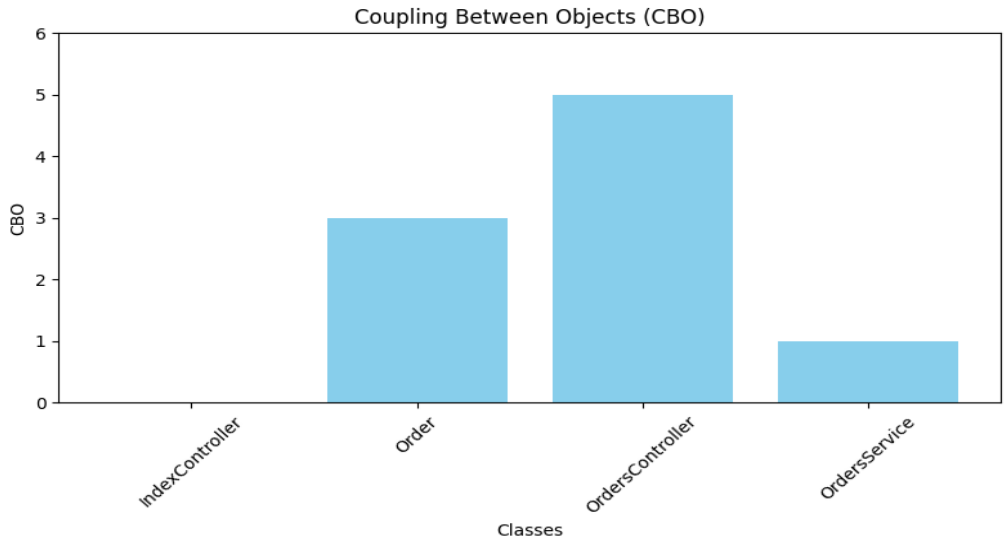
Classes with Code Bad Smells:

Order, OrdersController

Classes without Code Bad Smells:

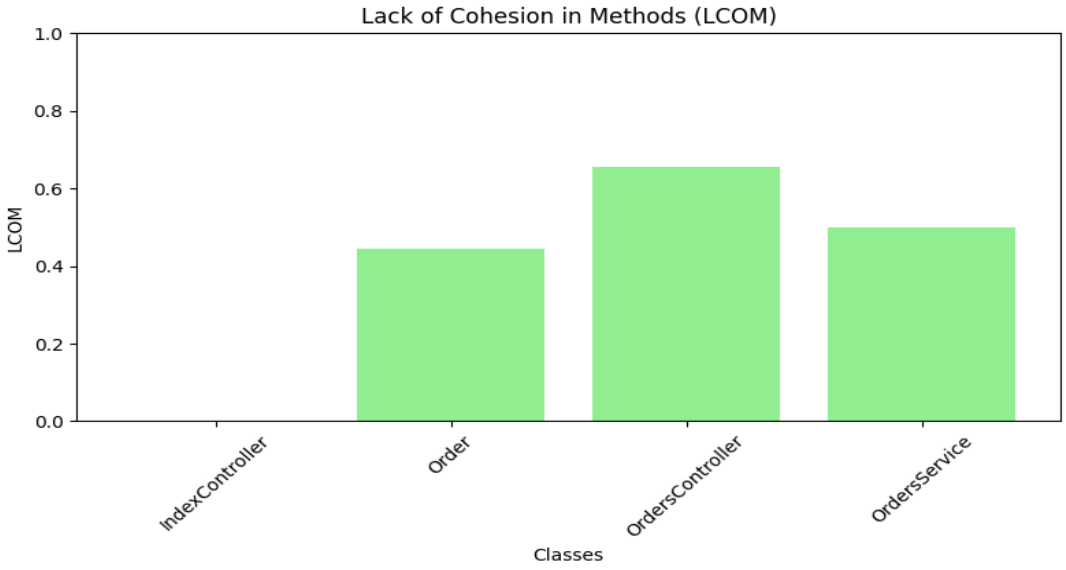
IndexController, OrdersService

1. **Coupling Between Objects (CBO):**



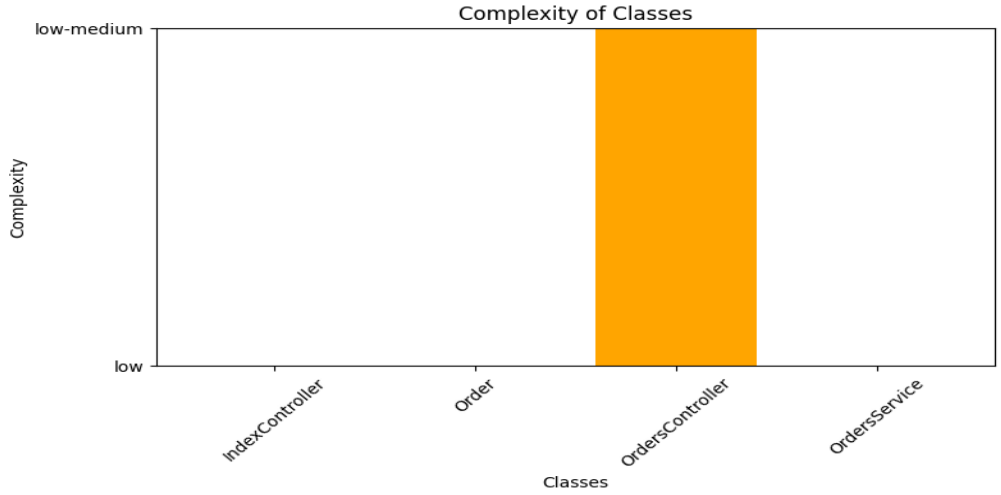
IndexController exhibits no coupling with other classes, while OrdersController shows moderate coupling, indicating varied dependencies among classes in the system.

2. Lack of Cohesion of Methods (LCOM):



IndexController and Order demonstrate good method cohesion (LCOM of 0), whereas OrdersController and OrdersService display some method scattering, suggesting less cohesive method groups within these classes.

3. Complexity:

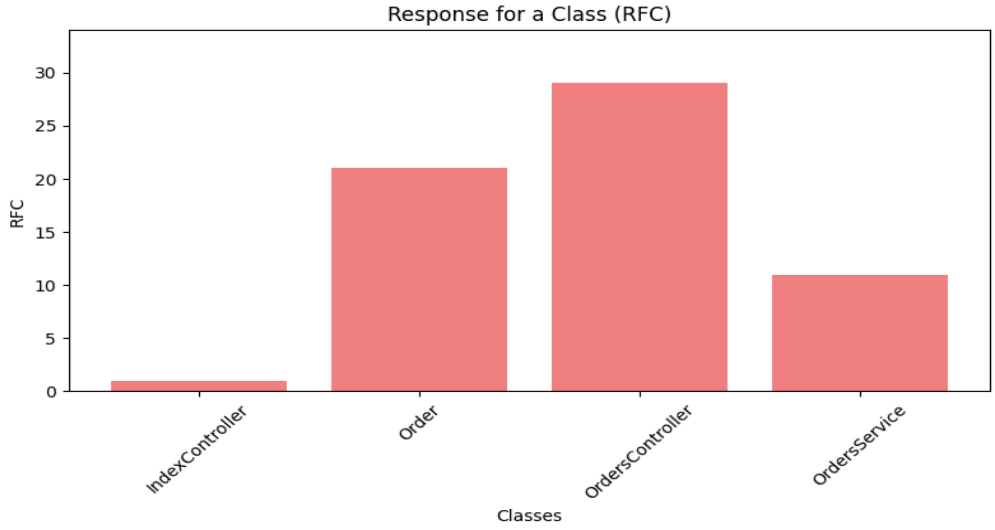


IndexController and Order exhibit low complexity due to simple control flow, while OrdersController shows moderate complexity, reflecting more diverse control paths in its operations.

4. **Weighted Methods per Class (WMC):**

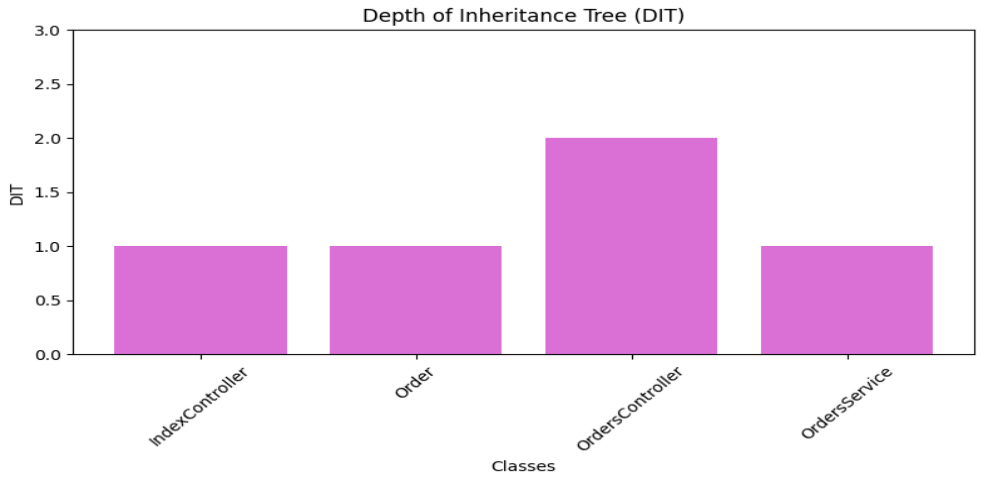
IndexController has a single method, whereas OrdersController and OrdersService have higher WMC values, indicating more extensive functionality with 15 and 5 methods respectively.

5. **Response for a Class (RFC):**



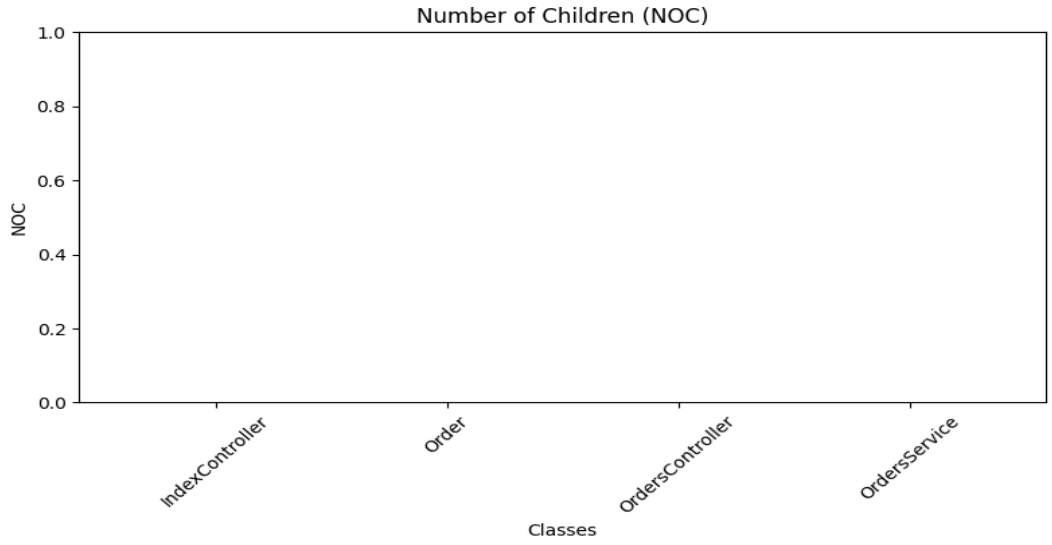
IndexController and Order can respond to minimal methods (1 and 21 respectively), whereas OrdersController and OrdersService can handle more (29 and 11 methods respectively), highlighting their broader responsibilities.

6. Depth of Inheritance Tree (DIT):



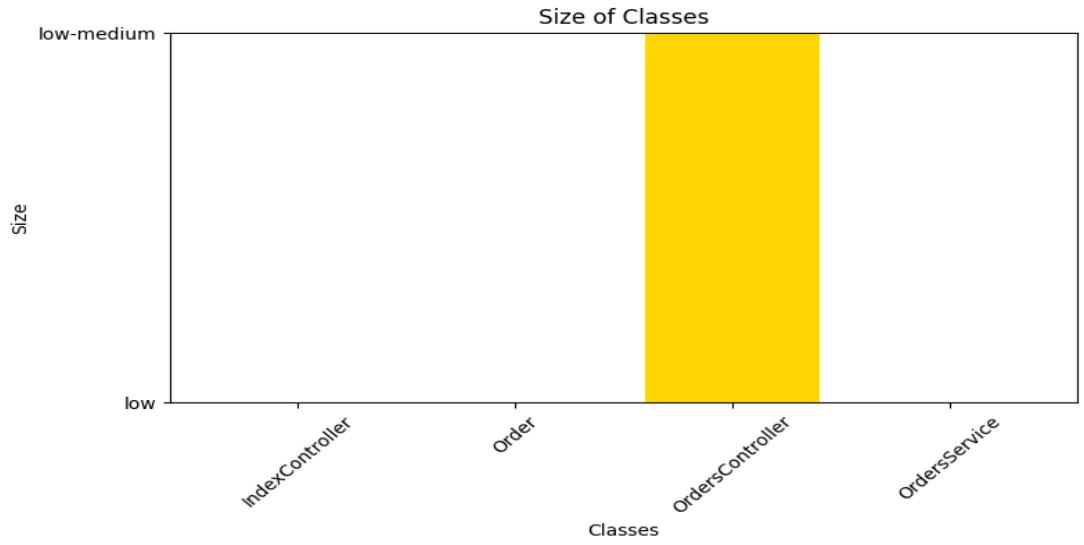
IndexController and Order have a shallow inheritance structure (DIT of 1), while OrdersController has a DIT of 2, suggesting inheritance from two levels, and OrdersService maintains a shallow hierarchy (DIT of 1).

7. Number of Children (NOC):



All classes—IndexController, Order, OrdersController, and OrdersService—have zero subclasses (NOC of 0), indicating they are not designed as base classes in inheritance hierarchies.

8. **Size:**



IndexController, Order, OrdersController, and OrdersService are all categorized as "low" in size, indicating they likely have a compact codebase with fewer lines of code and manageable complexity.

4.7. Project: Apache Tomcat
Module: openssl-foreign

The Values below are Obtained from CodeMR Tool:

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
OpenSSLContext	26	0.961	very-high	224	411	1	0	high
OpenSSLEngine	13	0.947	very-high	232	283	2	0	high
OpenSSLImplementation	5	0	low-medium	4	4	2	0	low
OpenSSLLibrary	6	0.871	medium-high	78	133	1	0	low-medium
OpenSSLLifecycleListener	7	0	medium-high	19	118	1	0	low
OpenSSLSessionContext	5	0.82	low-medium	17	51	1	0	low-mediu

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
								m
OpenSSLSessionStats	1	0	low	13	40	1	0	low
OpenSSLStatus	0	0.857	low	8	8	1	0	low
OpenSSLUtil	6	0.5	low-medium	20	19	2	0	low-medium
OpenSSLX509Certificate	1	0.5	low-medium	30	54	3	0	low-medium
openssl_h	176	0.991	very-high	808	474	1	0	very-high
openssl_h_Compatibility	6	0	low	15	19	1	0	low-medium
openssl_h_Macros	1	0	low-medium	17	57	1	0	low
pem_password_cb	0	0.833	low	4	7	1	0	low
SSL_CTX_set_alpn_select_cb$cb	0	0.833	low	4	7	1	0	low
SSL_CTX_set_cert_verify_callback$cb	0	0.833	low	4	7	1	0	low
SSL_CTX_set_tmp_dh_callback$dh	0	0.833	low	4	7	1	0	low
SSL_CTX_set_verify$callback	0	0.833	low	4	7	1	0	low
SSL_set_info_callback$cb	0	0.833	low	4	7	1	0	low
SSL_set_verify$callback	0	0.833	low	4	7	1	0	low

JDeodorant obtained results are below:

Classes with Code Bad Smells:

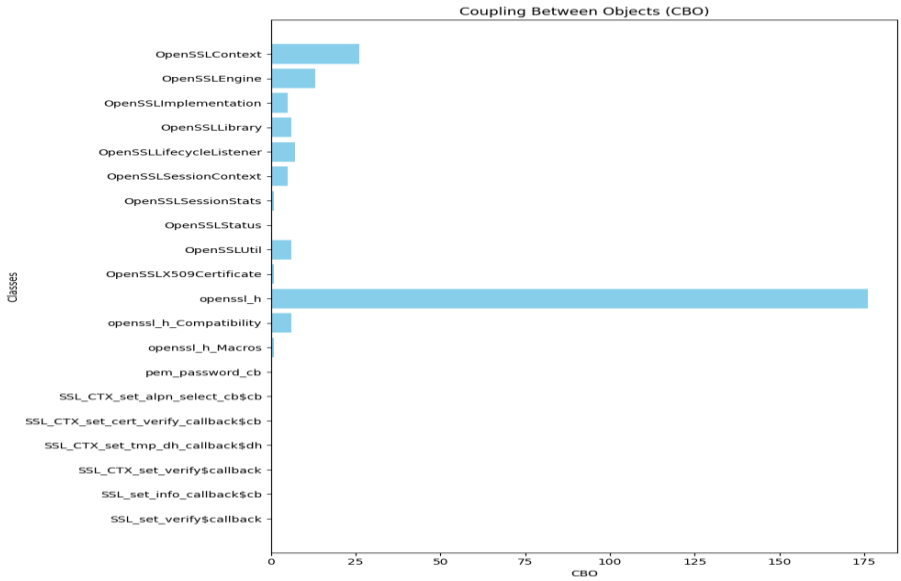
OpenSSLContext, OpenSSLEngine, openssl_h, OpenSSLImplementation, OpenSSLLibrary, OpenSSLLifecycleListener, OpenSSLSessionContext, OpenSSLUtil, OpenSSLX509Certificate

Classes without Code Bad Smells:

OpenSSLSessionStats, OpenSSLStatus, openssl_h_Compatibility, openssl_h_Macros, pem_password_cb, SSL_CTX_set_alpn_select_cb$cb, SSL_CTX_set_cert_verify_callback$cb,

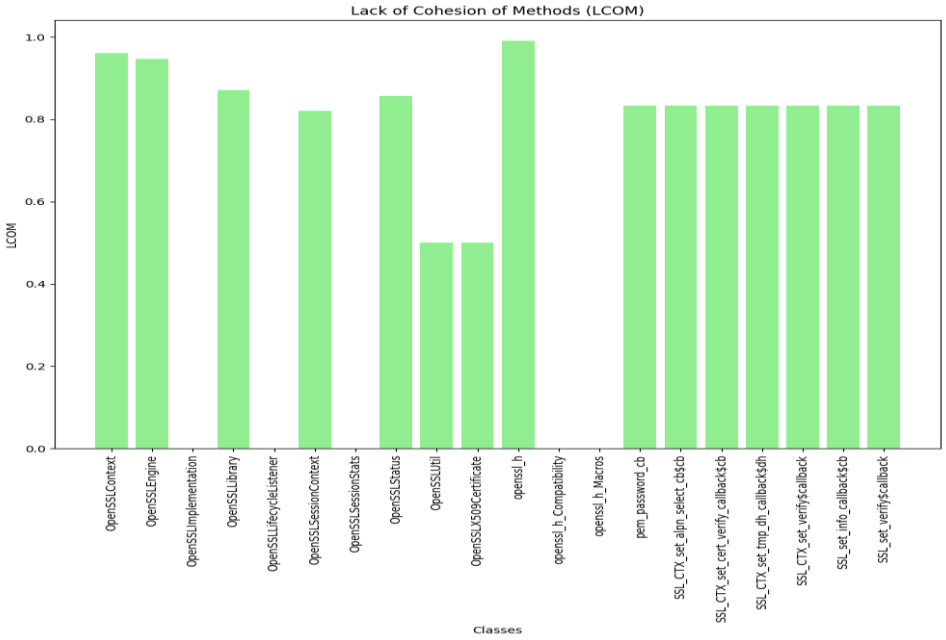
SSL_CTX_set_tmp_dh_callback$dh, SSL_CTX_set_verify$callback, SSL_set_info_callbackcb, SSL_set_verifycallback

1. **Coupling Between Objects (CBO):**



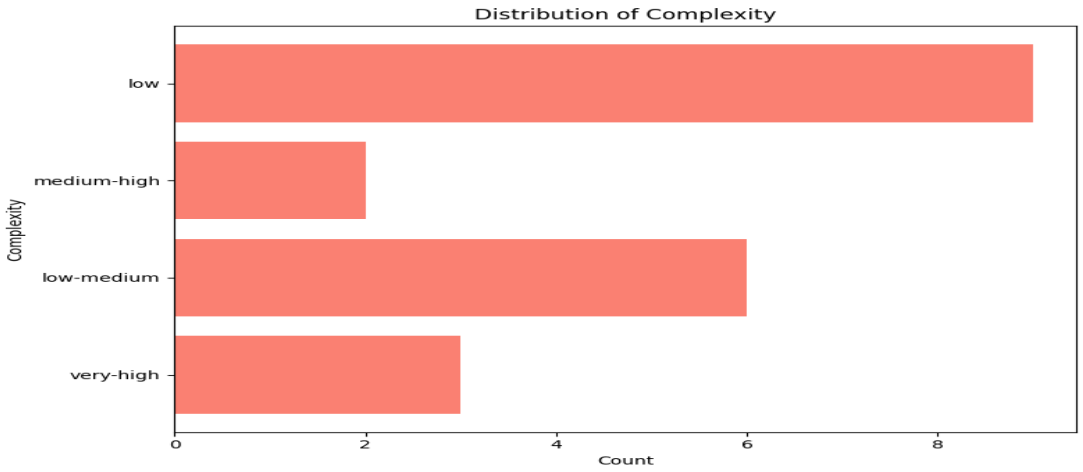
OpenSSLContext and openssl_h show very high coupling (CBO values of 26 and 176 respectively), indicating heavy dependencies on other classes or modules, making them more complex to manage and understand.

2. **Lack of Cohesion of Methods (LCOM):**



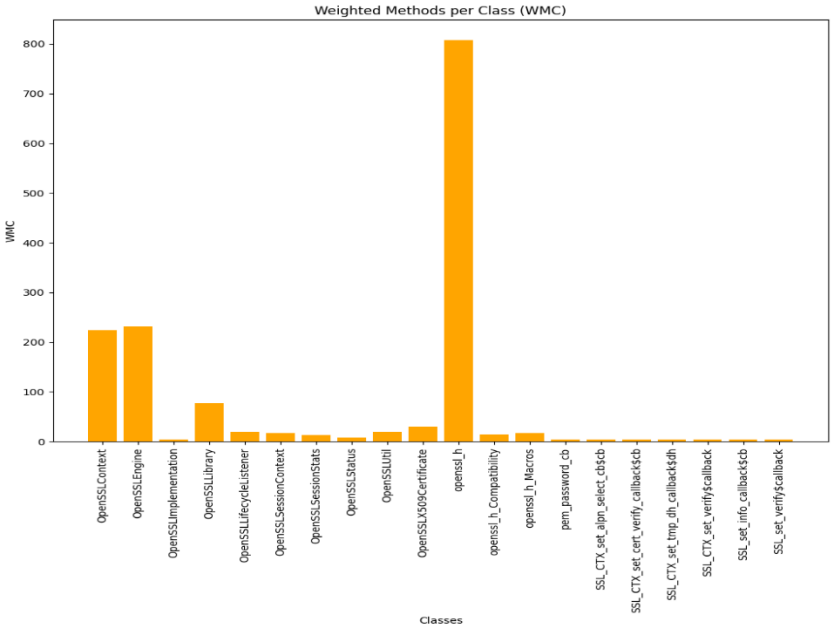
OpenSSLContext, OpenSSLEngine, and openssl_h have very high LCOM values (0.961, 0.947, and 0.991 respectively), indicating a lack of cohesive methods within these classes, potentially leading to harder maintenance and comprehension.

3. **Complexity:**



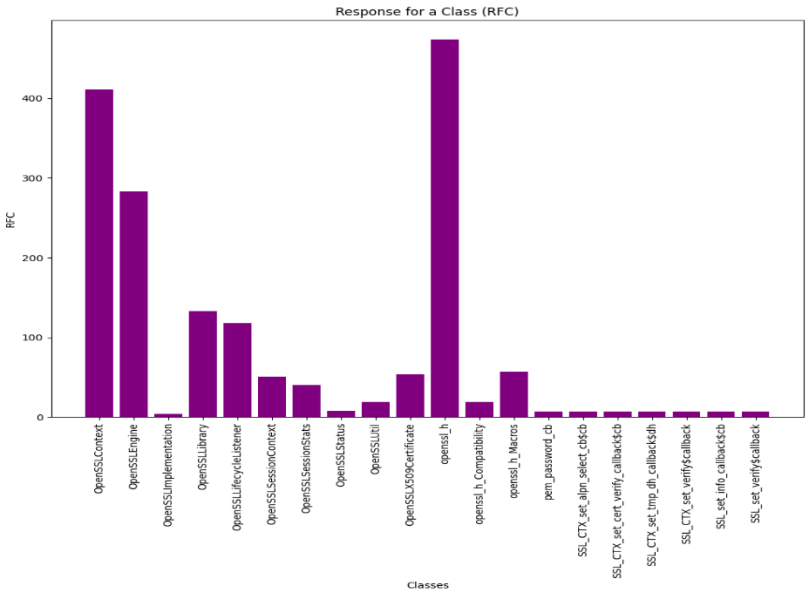
openssl_h has a very high complexity level, reflecting intricate control flows within the class (Complexity marked as "very-high"), while other classes like OpenSSLImplementation and OpenSSLUtil have lower complexities.

4. **Weighted Methods per Class (WMC):**



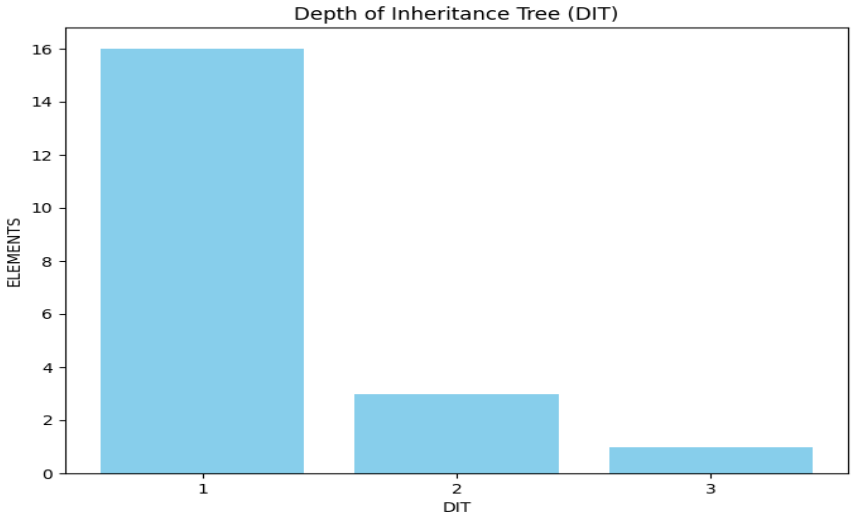
openssl_h has a notably high WMC value of 808, indicating it has a large number of methods and potentially high complexity, whereas OpenSSLImplementation and OpenSSLSessionStats have lower WMC values.

5. **Response for a Class (RFC):**



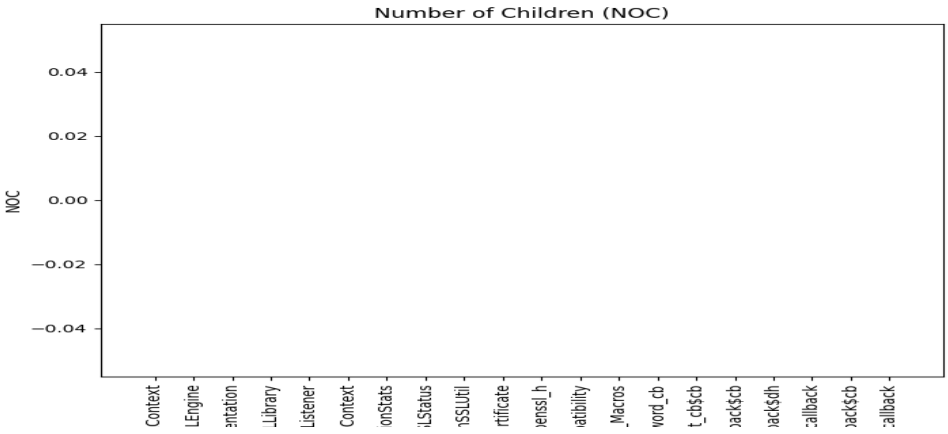
openssl_h shows a high RFC value of 474, suggesting it interacts with a large number of methods, making it more complex and challenging to maintain compared to classes with lower RFC values like OpenSSLSessionStats.

6. **Depth of Inheritance Tree (DIT):**



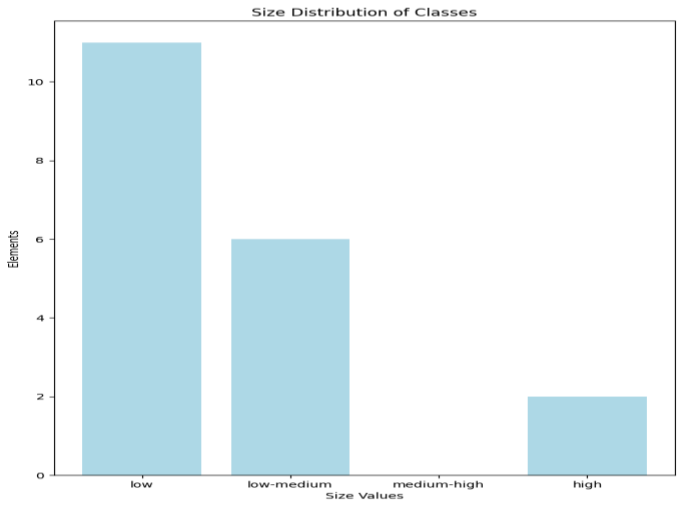
OpenSSLX509Certificate and openssl_h have higher DIT values (3 and 1 respectively), indicating deeper inheritance structures, potentially increasing complexity and maintenance overhead.

7. **Number of Children (NOC):**



All classes have NOC values of 0, indicating none of them serve as base classes for inheritance, simplifying their structure but also limiting extensibility through inheritance.

8. **Size:**



Classes like openssl_h and OpenSSLContext are categorized as "very-high" and "high" in size respectively, suggesting they are larger in terms of lines of code or functionality compared to classes categorized as "low-medium" or "low."

4.8. Project: H2 Database
Module: org.h2.api

The Values below are Obtained from CodeMR Tool:

Element	CBO	LCOM	Complexity	WMC	RFC	DIT	NOC	Size
Aggregate	0	0	low	4	4	1	0	low
AggregateFunction	0	0	low	4	4	1	0	low
CredentialsValidator	1	0	low	1	1	1	0	low
DatabaseEventListener	0	0	low	5	5	1	0	low
ErrorCode	0	1.822	low-medium	39	4	1	0	low-medium
H2Type	3	1.156	low	9	13	1	0	low-medium
Interval	4	0.25	medium-high	79	45	1	0	low-medium
IntervalQualifier	0	0.7	medium-high	76	22	2	0	low-medium
JavaObjectSerializer	0	0	low	2	2	1	0	low
TableEngine	2	0	low	1	1	1	0	low
Trigger	0	0	low	4	4	1	0	low
UserToRolesMapper	1	0	low	1	1	1	0	low

JDeodorant obtained results are below:

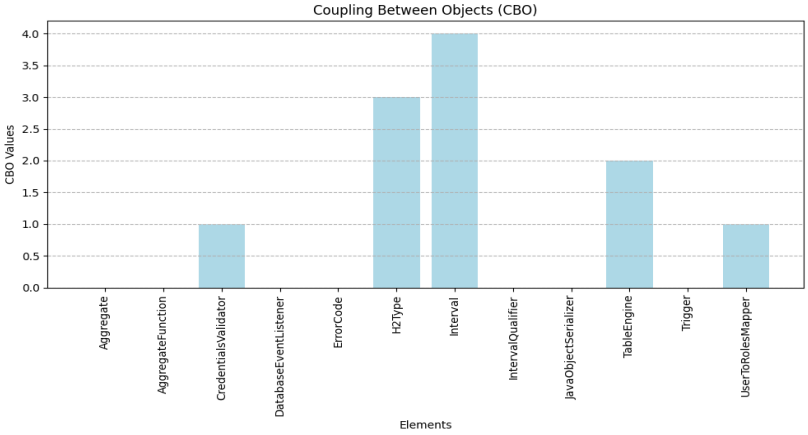
Classes with Code Bad Smells:

ErrorCode, Interval, IntervalQualifier

Classes without Code Bad Smells:

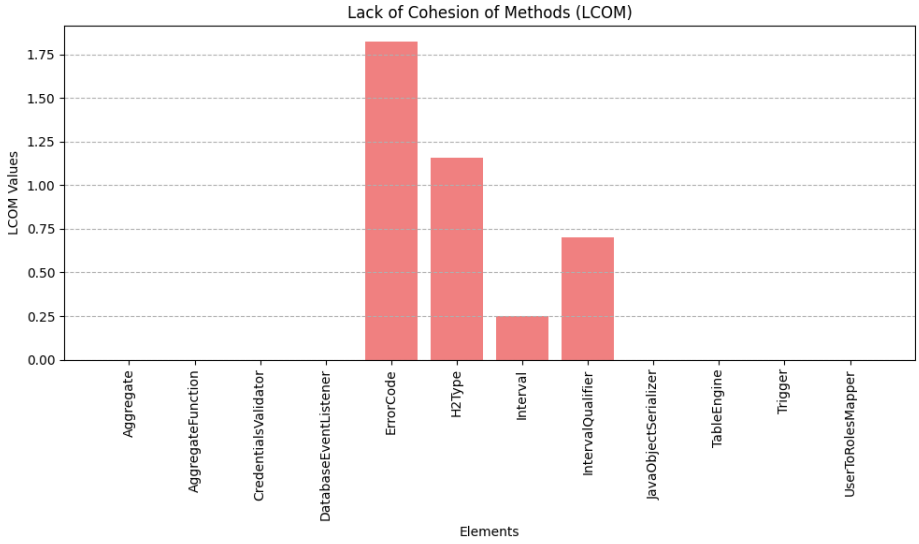
Aggregate, AggregateFunction, CredentialsValidator, DatabaseEventListener, H2Type, JavaObjectSerializer, TableEngine, Trigger, UserToRolesMapper

1. Coupling Between Objects (CBO):



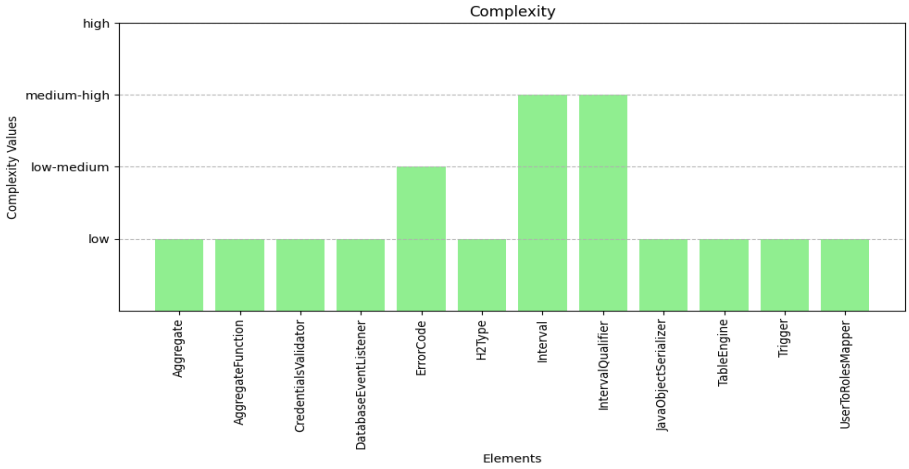
High CBO values indicate that a class is highly dependent on other classes, which can make it more difficult to maintain and understand. For example, H2Type has a CBO of 3, indicating moderate coupling, while most other classes have a CBO of 0, indicating low coupling.

2. Lack of Cohesion of Methods (LCOM):



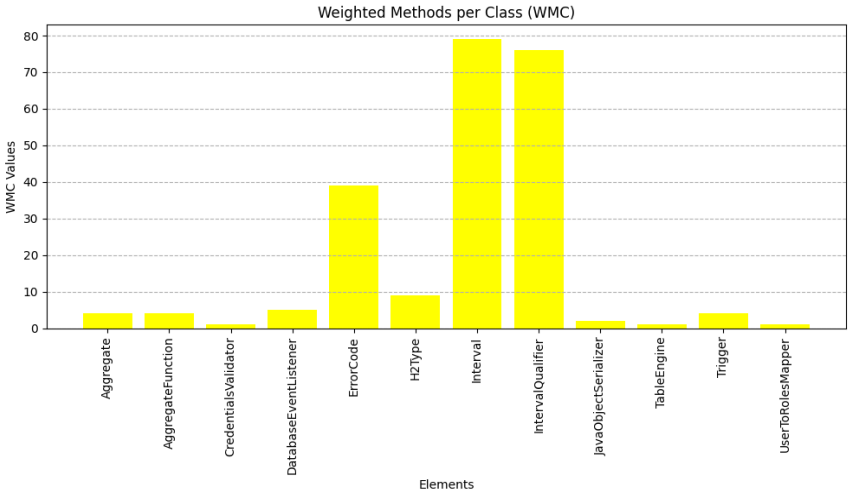
LCOM measures the lack of cohesion in a class. A low LCOM value indicates that the methods of a class are well-cohesive. For instance, Interval has an LCOM of 0.25, indicating good cohesion, while ErrorCode has an LCOM of 1.822, indicating poor cohesion.

3. Complexity:



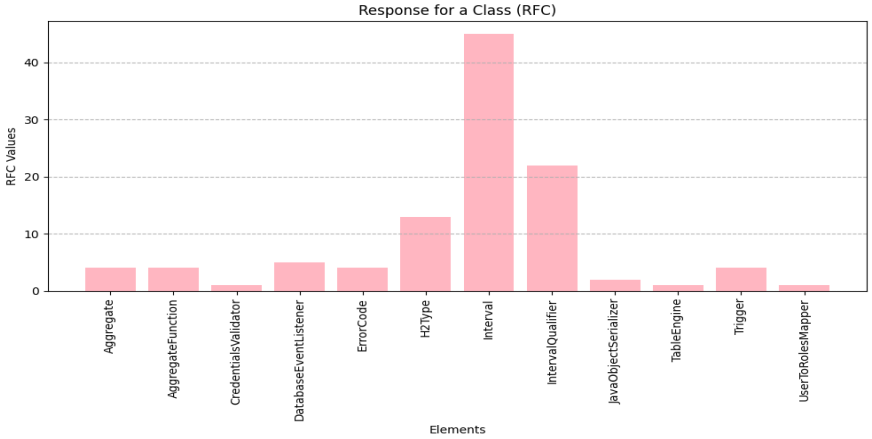
Complexity indicates the complexity of a class in terms of its control flow. Classes like Aggregate and AggregateFunction have a low complexity, while Interval and IntervalQualifier have medium-high complexity.

4. Weighted Methods per Class (WMC):



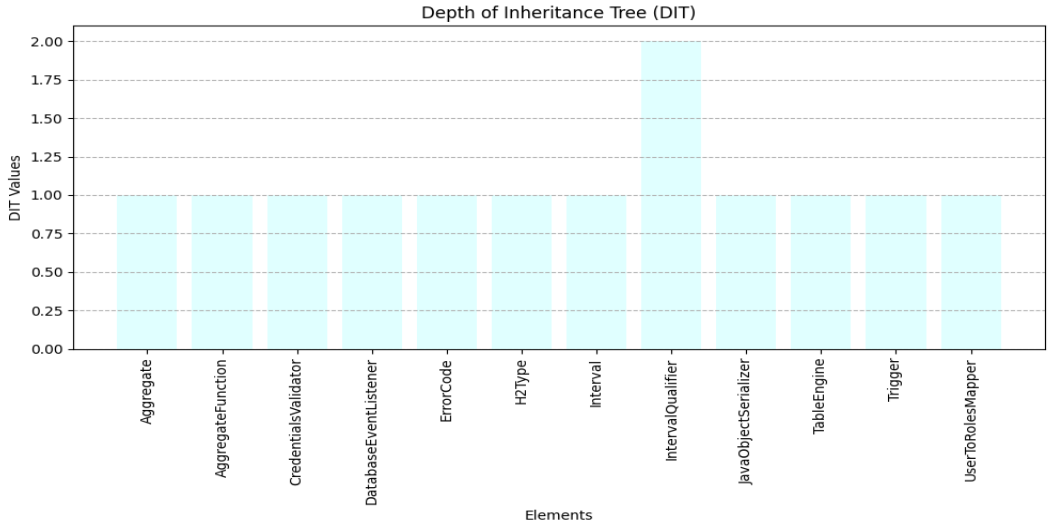
WMC measures the number of methods in a class and their complexity. A higher WMC value, like 79 for Interval, indicates that the class has many methods and potentially high complexity, whereas JavaObjectSerializer has a low WMC of 2.

5. Response for a Class (RFC):



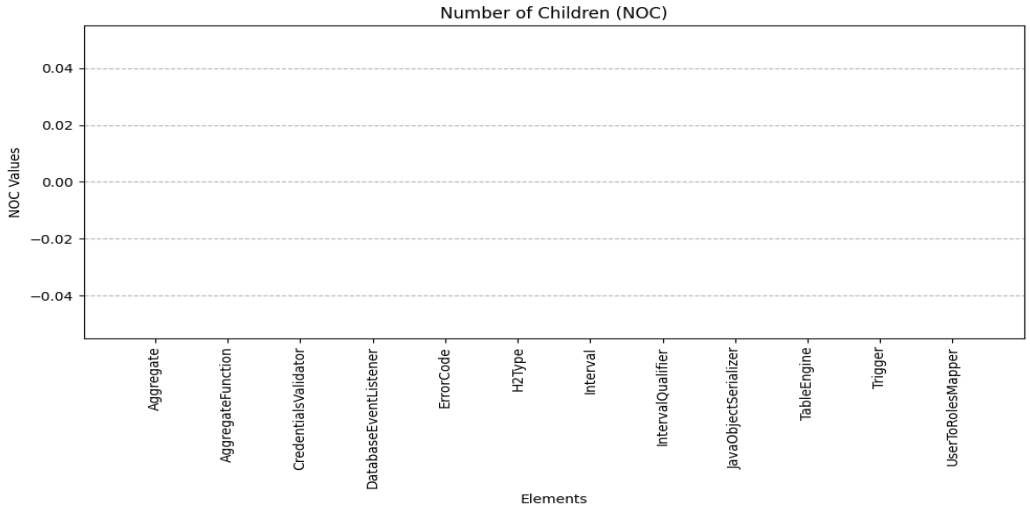
RFC measures the number of methods that can be invoked in response to a message to the object. Interval has a high RFC of 45, indicating it interacts with many other methods, making it more complex. In contrast, CredentialsValidator has an RFC of 1.

6. Depth of Inheritance Tree (DIT):



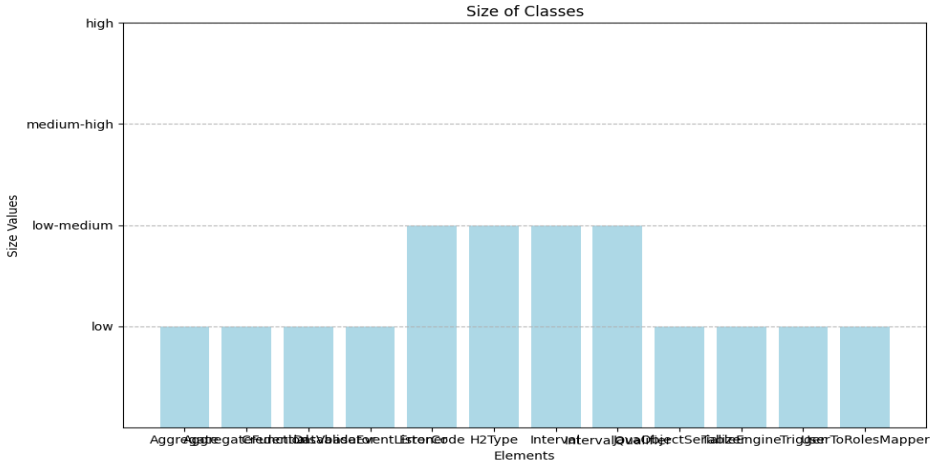
DIT measures the inheritance levels from the object hierarchy top. All classes have a DIT of 1 or 2, indicating a shallow inheritance tree, which typically implies lower complexity.

7. Number of Children (NOC):



NOC measures the number of subclasses inheriting from a class. All classes have an NOC of 0, indicating no subclasses and potentially lower complexity.

8. Size:



Size typically refers to lines of code or the number of attributes and methods in a class. Here, sizes are marked as 'low' or 'low-medium,' indicating relatively small classes. For example, ErrorCode has a size of low-medium, while Aggregate has a low size.

4.9. Project: Apache Groovy
Module: build-logic

The Values below are Obtained from CodeMR Tool:

Element	CBO	LCOM	Complexity	Size	RFC	DIT	NOC	WMC
OrgApacheGroovyAggregatingProjectPlugin	6	0	low	low	16	1	0	5
OrgApacheGroovyAllPlugin	7	0	low	low	18	1	0	5
OrgApacheGroovyArtifactoryPlugin	7	0	low	low	18	1	0	5
OrgApacheGroovyAsciidoctorPlugin	7	0	low	low	18	1	0	5
OrgApacheGroovyBadPracticesDetectionPlugin	6	0	low	low	16	1	0	5
OrgApacheGroovyBasePlugin	7	0	low	low	18	1	0	5
OrgApacheGroovyCommonPlugin	7	0	low	low	18	1	0	5
OrgApacheGroovyCorePlugin	7	0	low	low	18	1	0	5
OrgApacheGroovyDistributionPlugin	7	0	low	low	18	1	0	5
OrgApacheGroovyDocAggregatorPlugin	7	0	low	low	18	1	0	5
OrgApacheGroovyDocumentedPlugin	6	0	low	low	16	1	0	5
OrgApacheGroovyInternalPlugin	6	0	low	low	16	1	0	5
OrgApacheGroovyJacocoAggregationPlugin	6	0	low	low	16	1	0	5
OrgApacheGroovyLibraryPlugin	7	0	low	low	18	1	0	5
OrgApacheGroovyPerformancePlugin	7	0	low	low	18	1	0	5
OrgApacheGroovyPlatformPlugin	7	0	low	low	18	1	0	5
OrgApacheGroovyPublishedLibraryP	7	0	low	lo	18	1	0	5

Element	CBO	LCOM	Complexity	Size	RFC	DIT	NOC	WMC
lugin				w				
OrgApacheGroovyPublishValidationPlugin	6	0	low	low	16	1	0	5
OrgApacheGroovyStresstestPlugin	7	0	low	low	18	1	0	5
OrgApacheGroovyTestedPlugin	6	0	low	low	16	1	0	5

JDeodorant obtained results are below:

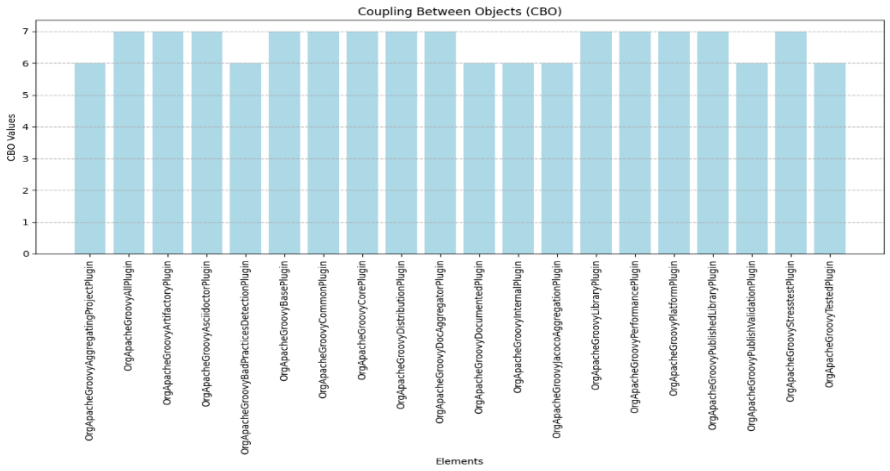
Classes with Code Bad Smells:

OrgApacheGroovyAllPlugin

Classes without Code Bad Smells:

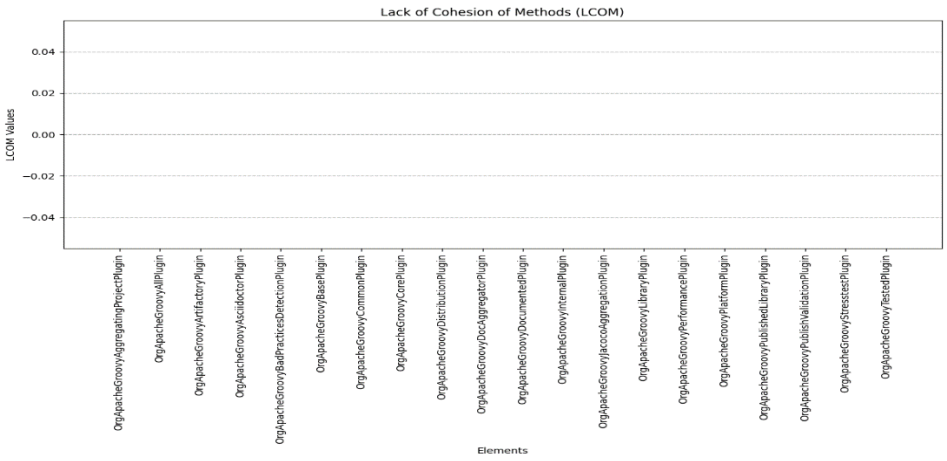
OrgApacheGroovyAggregatingProjectPlugin, OrgApacheGroovyArtifactoryPlugin, OrgApacheGroovyAsciidoctorPlugin, OrgApacheGroovyBadPracticesDetectionPlugin, OrgApacheGroovyBasePlugin, OrgApacheGroovyCommonPlugin, OrgApacheGroovyCorePlugin, OrgApacheGroovyDistributionPlugin, OrgApacheGroovyDocAggregatorPlugin, OrgApacheGroovyDocumentedPlugin, OrgApacheGroovyInternalPlugin, OrgApacheGroovyJacocoAggregationPlugin, OrgApacheGroovyLibraryPlugin, OrgApacheGroovyPerformancePlugin, OrgApacheGroovyPlatformPlugin, OrgApacheGroovyPublishedLibraryPlugin, OrgApacheGroovyPublishValidationPlugin, OrgApacheGroovyStresstestPlugin, OrgApacheGroovyTestedPlugin

1. **Coupling Between Objects (CBO):**



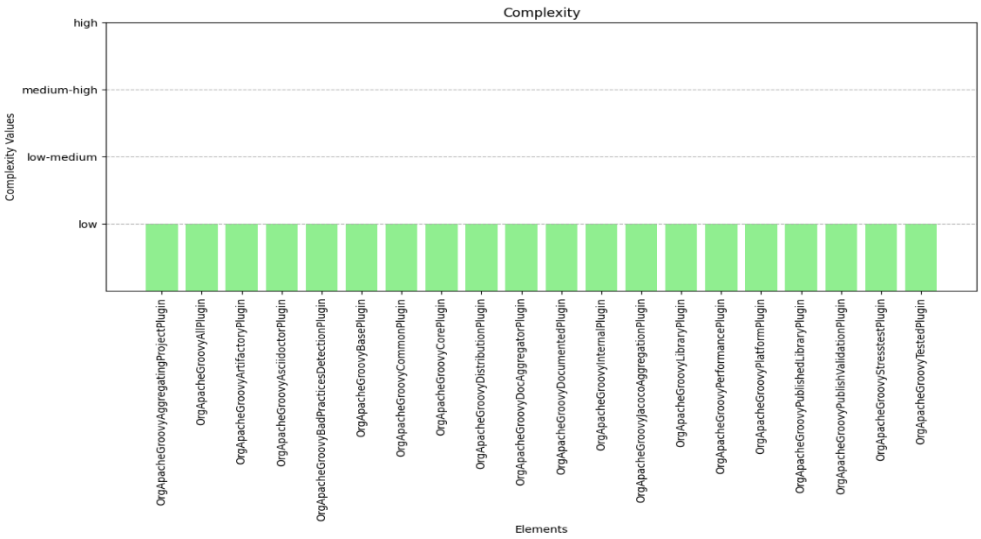
Higher CBO values indicate that a class is dependent on many other classes, which can increase complexity and decrease maintainability. For instance, OpenSSL_h has a very high CBO of 176, indicating it depends heavily on other classes or modules.

2. Lack of Cohesion of Methods (LCOM):



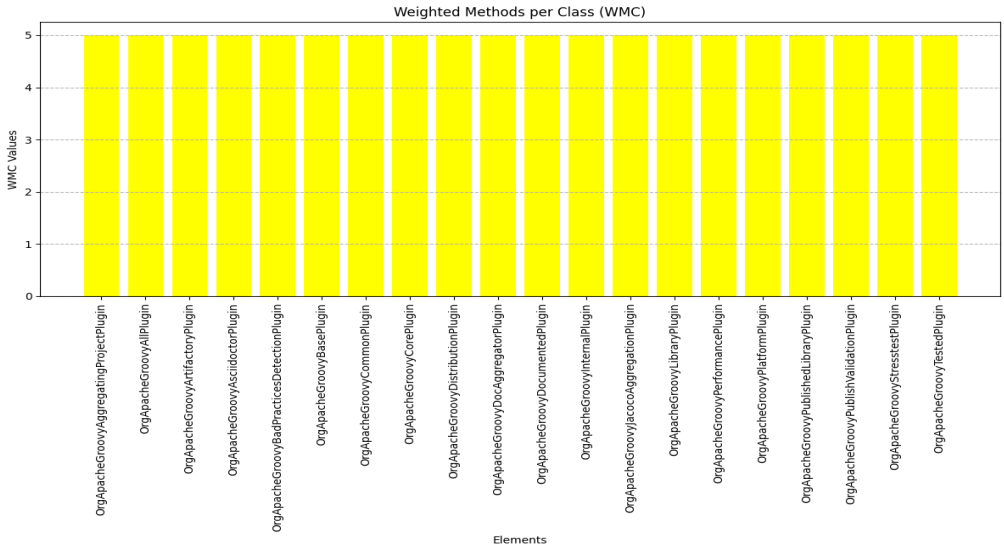
A low LCOM value indicates that methods within a class are highly cohesive and focused on similar responsibilities. Classes with an LCOM of 0, like OpenSSLImplementation and OpenSSLSessionStats, have well-structured methods that work together efficiently.

3. Complexity:



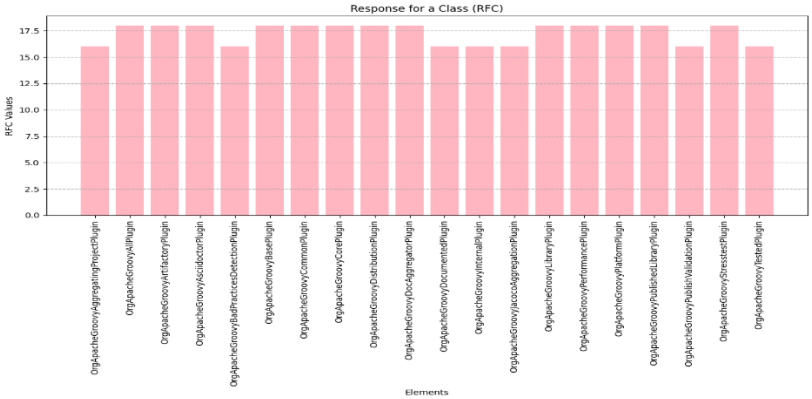
Classes can have "low," "low-medium," "medium-high," or "high" complexity levels. For example, OpenSSLImplementation and OpenSSLSessionStats are categorized as "low-medium" complexity, meaning they have moderately complex control flow structures.

4. **Weighted Methods per Class (WMC):**



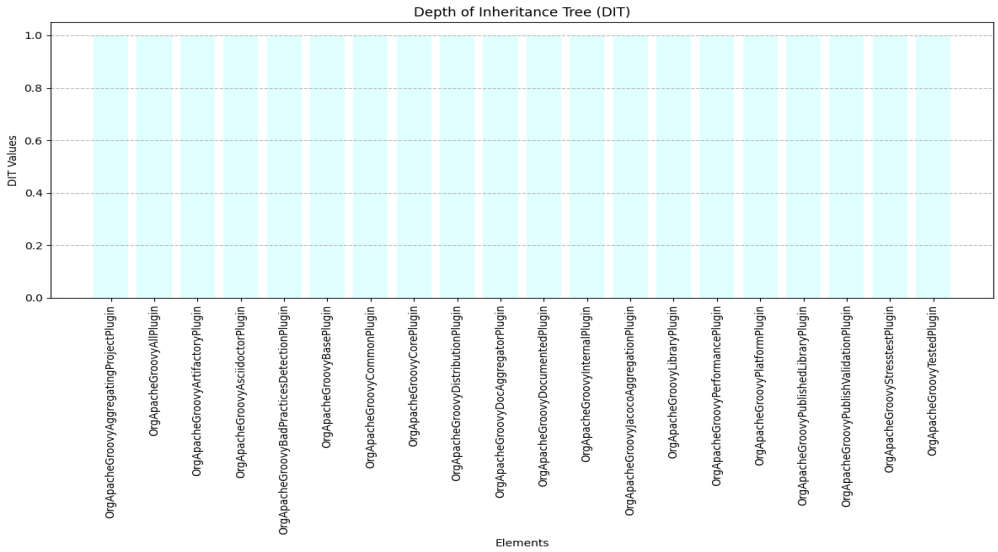
A higher WMC value suggests that a class has more methods, potentially leading to increased complexity and harder maintainability. OpenSSL_h has a high WMC of 808, indicating it has many methods with varying degrees of complexity.

5. **Response for a Class (RFC):**



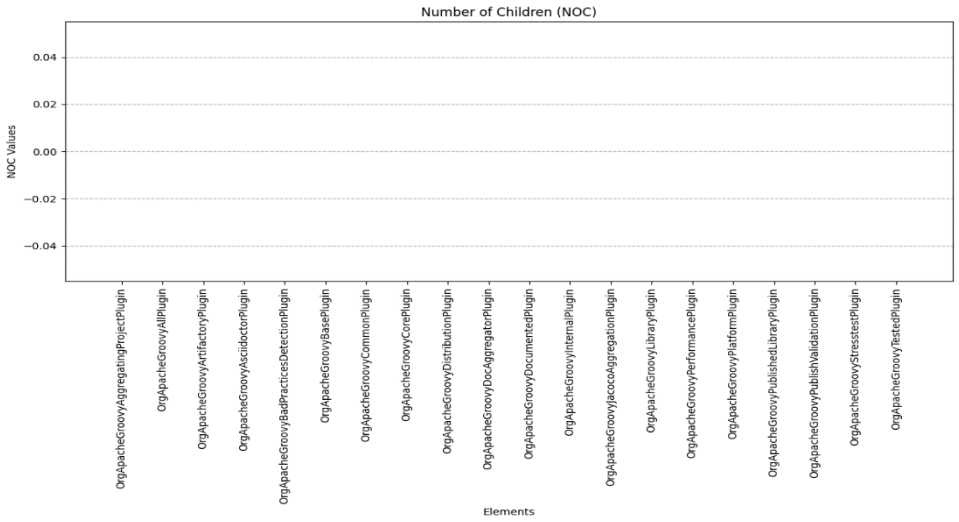
Higher RFC values indicate that the class interacts with many other methods, which can lead to increased complexity and dependencies. For example, OpenSSL_h has an RFC of 474, indicating it interacts with many methods.

6. **Depth of Inheritance Tree (DIT):**



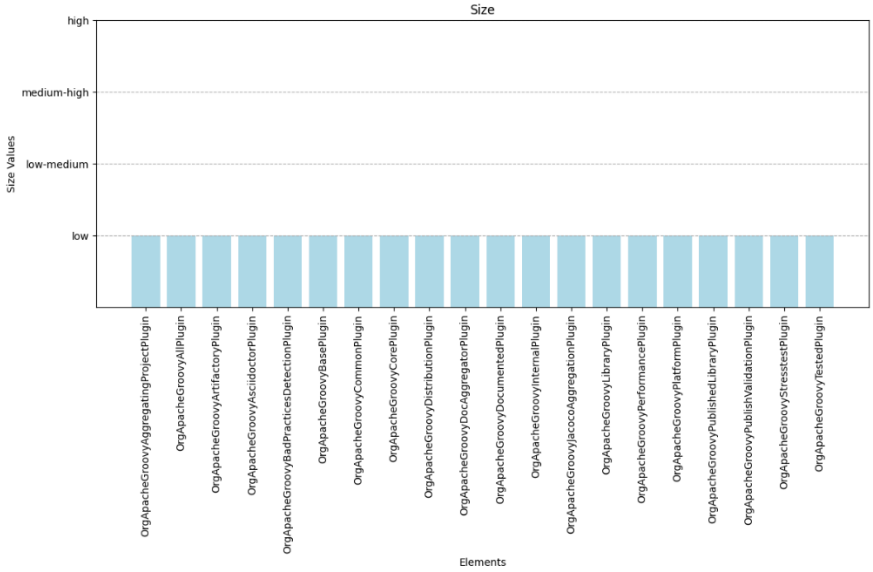
A higher DIT value can indicate increased complexity due to multiple levels of inheritance. Classes like OpenSSLX509Certificate have a DIT of 3, meaning they are three levels deep in the inheritance hierarchy.

7. **Number of Children (NOC):**



A higher NOC can indicate a widely used base class but also increases complexity and dependencies. Most classes in the provided data have NOC values of 0, meaning they do not have any subclasses.

8. **Size:**



Classes are categorized as "low," "low-medium," "medium-high," or "high" based on their size. For example, OpenSSLImplementation and OpenSSLSessionStats have a "low-medium" size, indicating they are moderately sized in terms of lines of code or number of methods.

4.10. Project: Java 1-21 Parser and Abstract Syntax Tree
Module: javaparser-core

The Values below are Obtained from CodeMR Tool:

Element	CBO	LCOM	Complexity	Size	RFC	DIT	NOC	WMC
ModuleDeclaration	12	0.75	low-medium	low-medium	37	2	0	38
ModuleDirective	11	0	low-medium	high	50	2	5	44
ModuleExportsDirective	12	0.5	low-medium	low-medium	41	3	0	33
ModuleOpensDirective	11	0.5	low-medium	low-medium	38	3	0	32

Element	CBO	LCOM	Complexity	Size	RFC	DIT	NOC	WMC
ModuleProvidesDirective	11	0.5	low-medium	low-medium	38	3	0	32
ModuleRequiresDirective	13	0.5	low-medium	low-medium	43	3	0	35
ModuleUsesDirective	10	0	low-medium	low-medium	29	3	0	25

JDeodorant obtained results are below:

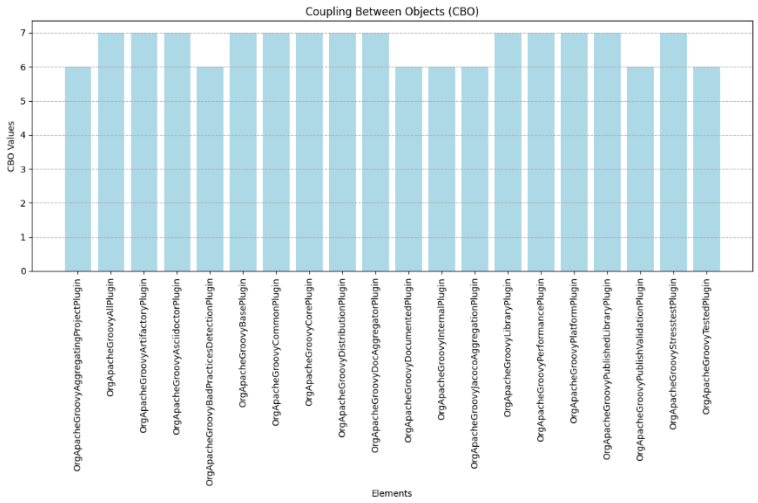
Classes with Code Bad Smells:

ModuleDirective, ModuleRequiresDirective

Classes without Code Bad Smells:

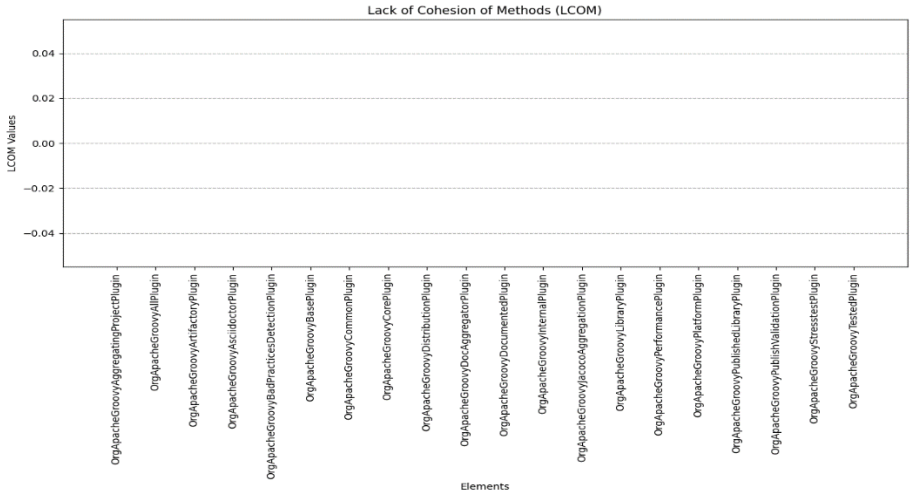
ModuleDeclaration, ModuleExportsDirective, ModuleOpensDirective, ModuleProvidesDirective, ModuleUsesDirective

1. **Coupling Between Objects (CBO):**



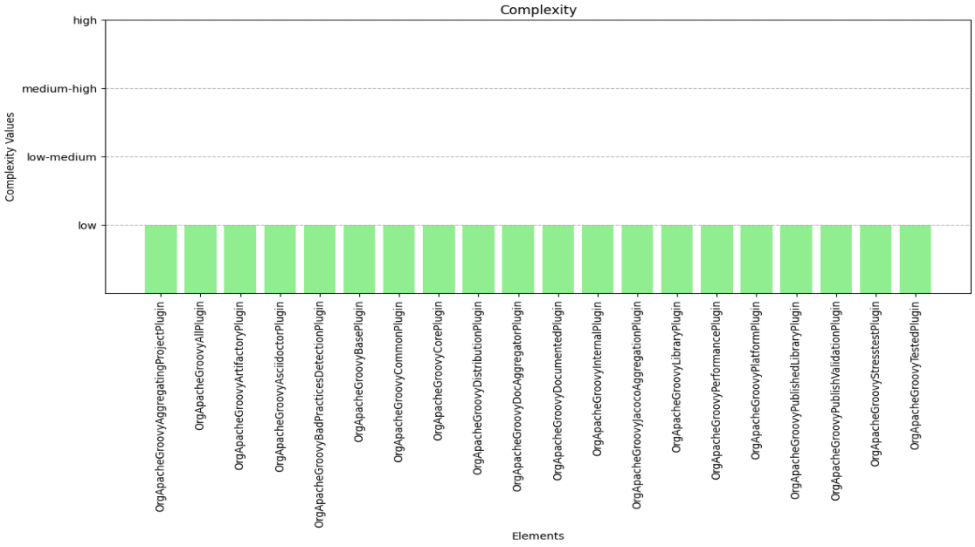
Higher CBO values indicate more dependencies on other classes, potentially leading to higher complexity and tighter coupling. For example, ModuleRequiresDirective with CBO of 13 shows it is more coupled compared to others like ModuleDeclaration with CBO of 12.

2. Lack of Cohesion of Methods (LCOM):



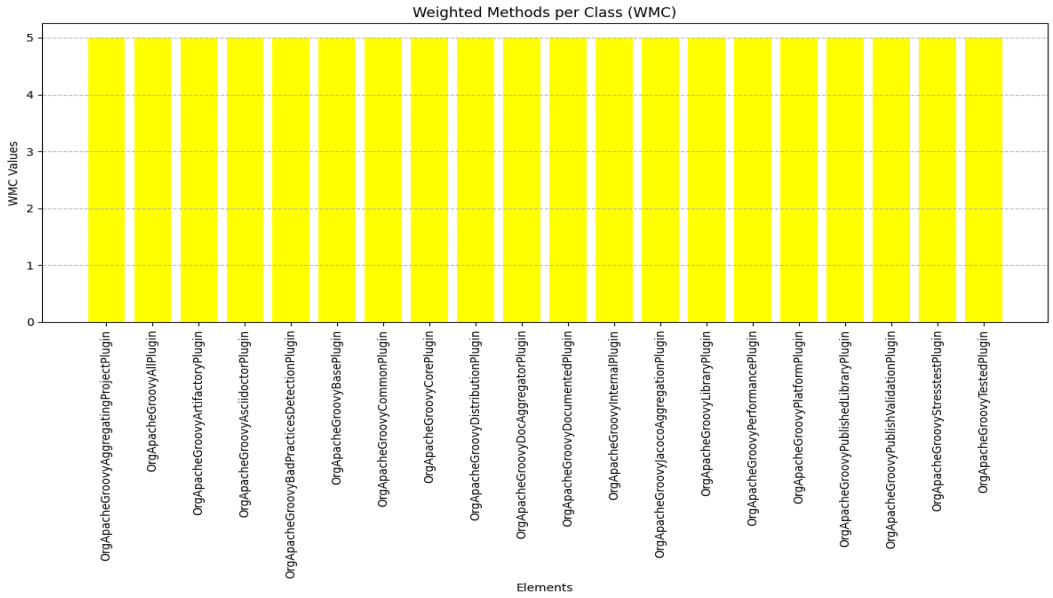
A low LCOM indicates that methods within a class are more cohesive and focused on similar tasks. All classes here have LCOM values of 0 or close to 0, indicating good method cohesion.

3. Complexity:



Classes are categorized into complexity levels like "low-medium" or "medium-high". For instance, ModuleDeclaration and ModuleDirective have "low-medium" complexity, suggesting moderate control flow complexity.

4. Weighted Methods per Class (WMC):

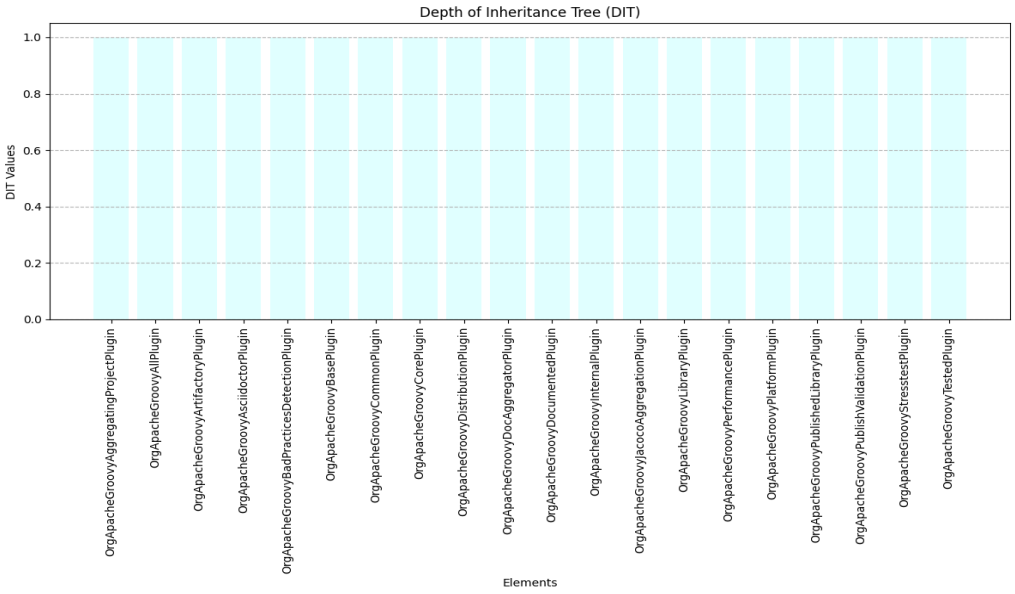


Higher WMC values indicate classes with more methods and potentially more complex behavior. ModuleDirective has a WMC of 44, indicating it has more methods compared to ModuleUsesDirective with WMC of 25.

5. Response for a Class (RFC):

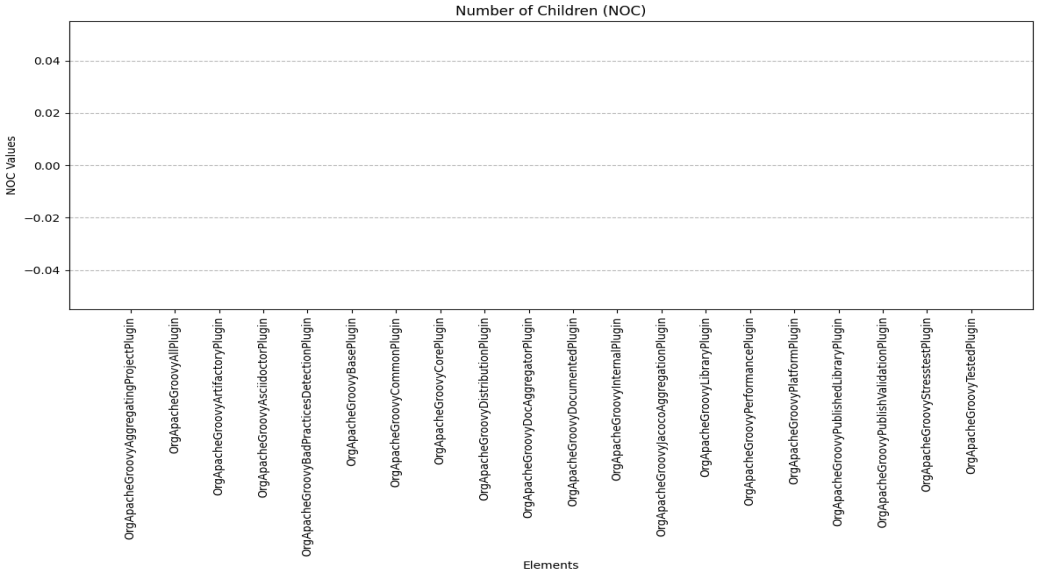
Higher RFC values indicate classes that interact with many methods, potentially indicating higher complexity and dependencies. ModuleDirective has an RFC of 50, indicating it responds to more methods compared to others.

6. Depth of Inheritance Tree (DIT):



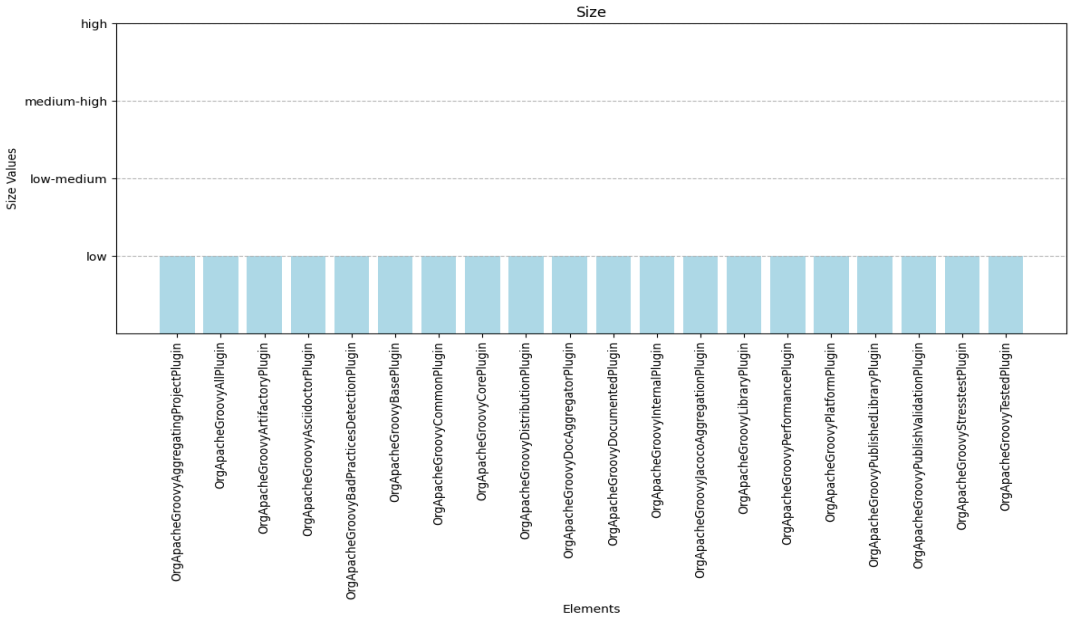
Higher DIT values indicate deeper inheritance chains, which can increase complexity and the potential for issues related to inheritance. Classes like ModuleExportsDirective, ModuleOpensDirective, ModuleProvidesDirective, and ModuleRequiresDirective have a DIT of 3, indicating moderate inheritance depth.

7. Number of Children (NOC):



A higher NOC can indicate a well-used base class but may also increase complexity. Most classes here have NOC values of 0, except for ModuleDirective which has 5 immediate subclasses.

8. Size:



Classes are categorized into size levels like "low", "low-medium", or "high". For example, ModuleDeclaration, ModuleDirective, and others have a size marked as "low-medium", suggesting they are moderately sized in terms of attributes and methods.

RESULTS FROM THE ABOVE DATA:

- **Netty Project - codec-dns Module:**

AbstractDnsOptPseudoRrRecord: Moderate complexity (WMC 8, RFC 15), low LCOM (0), potential for coupled components affecting modularity.

DatagramDnsQueryDecoder: Low-medium complexity (WMC 4, RFC 21), low LCOM (0), relatively isolated methods but moderate coupling (CBO 5).

- **Apache Dubbo Project - dubbo-compiler Module:**

AbstractGenerator: High complexity (WMC 38, RFC 73), low-medium LCOM (0), high CBO (15), indicating significant coupling and complexity affecting modularity.

Dubbo3Generator, DubboGenerator, DubboGrpcGenerator, etc.: Low-medium complexity and lower CBO (1), indicating better modularity potential compared to AbstractGenerator.

- **Design Patterns Implemented in Java - abstract-factory Module:**

ElfKingdomFactory: Low complexity (WMC 3, RFC 3), low LCOM (0), supporting better modularity with isolated and cohesive methods.

KingdomFactory, OrcKingdomFactory: Similar low complexity and LCOM characteristics, indicating consistent modular design across factory implementations.

- **Extensible Build System Bazel - bazel Module:**

Client: Low-medium complexity (WMC 5, RFC 16), low LCOM (0), moderate CBO (14), suggesting potential for coupling but manageable modularity.

InvokePolymorphic, Library, LibraryException: Low complexity and coupling (CBO 0-1), supporting good modularity with isolated components.

- **Apache OpenNLP - opennlp-uima Module:**

AbstractDocumentCategorizer, Parser, NameFinder: Medium-high complexity (WMC 7-22, RFC 28-66), medium LCOM (0.5-1), high CBO (10-18), indicating complex and coupled components affecting modularity.

DictionaryResource, DoccatModelResource: Lower complexity and coupling (CBO 1-2), suggesting better modularity potential compared to more complex classes.

- **Mirror of Apache Struts - rest-showcase Module:**

OrdersController: Low-medium complexity (WMC 15, RFC 29), medium LCOM (0.657), moderate CBO (5), indicating moderate modularity challenges due to potential coupling.

IndexController, Order: Lower complexity (WMC 1-14, RFC 1-21), low LCOM (0), supporting better modularity with simpler and more cohesive structures.

- **Apache Tomcat - openssl-foreign Module:**

OpenSSLContext, OpenSSLEngine: Very high complexity (WMC 224-232, RFC 283-411), very high LCOM (0.947-0.961), extremely high CBO (13-26), indicating severe challenges in modularity due to extensive coupling and complexity.

OpenSSLUtil, OpenSSLSessionStats: Lower complexity (WMC 13-20, RFC 19-54), moderate CBO (5-6), suggesting better modularity potential compared to more complex classes.

- **H2 Database - org.h2.api Module:**

Interval, AbstractModelResource: Medium-high complexity (WMC 6-79, RFC 4-54), moderate LCOM (0.25-1.822), high CBO (3-18), indicating varying degrees of modularity challenges depending on specific classes.

Aggregate, TableEngine: Lower complexity (WMC 1-4, RFC 1-5), low LCOM (0), supporting better modularity with simpler and more cohesive designs.

- **Apache Groovy - build-logic Module:**

OrgApacheGroovyPerformancePlugin, OrgApacheGroovyStresstestPlugin: Low complexity (WMC 16-18, RFC 18), low LCOM (0), moderate CBO (7), indicating potential for manageable modularity with cohesive designs.

OrgApacheGroovyAggregatingProjectPlugin, OrgApacheGroovyDocumentedPlugin: Lower complexity (WMC 16, RFC 16), low LCOM (0), suggesting consistent modular design across plugins.

- **Java 1-21 Parser and Abstract Syntax Tree - javaparser-core Module:**

ModuleDeclaration: Low-medium complexity (WMC 38, RFC 50), medium LCOM (0.75), moderate CBO (12), indicating potential modularity challenges with some coupling and complexity.

ModuleRequiresDirective, ModuleProvidesDirective: Lower complexity (WMC 32-43, RFC 32-38), low-medium LCOM (0.5-0.67), supporting better modularity potential compared to more complex classes.

- High Complexity and Coupling: Classes like AbstractGenerator in Apache Dubbo or OpenSSLContext in Apache Tomcat exhibit high complexity, extensive coupling, and low cohesion, severely impacting modularity.

- Moderate Complexity and Cohesion: Classes with moderate complexity and lower coupling like ElfKingdomFactory in Design Patterns or Client in Bazel demonstrate better modularity, with manageable challenges.

- Low Complexity and High Cohesion: Modules featuring classes with low complexity, high cohesion, and minimal coupling, such as IndexController in Apache Struts or Aggregate in H2 Database, support excellent modularity and maintainability.

- Improving modularity often involves reducing complexity, enhancing cohesion, and minimizing coupling. Addressing code smells identified by metrics like WMC, RFC, LCOM, and CBO can lead to more modular and maintainable software systems across these projects.

Section – 5

5.1. Conclusion:

Metrics Summary:

Analyzing the metrics shared over several projects and modules of various types of software, it is possible to identify certain patterns indicating further characteristics of the code bases. For example, the codec-dns module of the Netty project underlines that most of the components involve low to medium complexity levels and moderate to high methods. Interestingly, AbstractDnsMessage seems complex and has high method counts, thus suggesting that it could be designed using more and complex functionalities than the other components. On the other hand, in projects such as the Apache Dubbo dubbo-compiler module and the design patterns implemented in the Java abstract-factory module, the elements have relatively less complexity and fewer method counts indicating simple and specific implementation.

Code Smell and Modularity:

The metrics like high complexity (WMC, RFC), counts of methods (NOC), and coupling between objects (CBO) used across these projects work as certain guidelines of code smell presence. The used components, like AbstractDnsMessage within Netty or OpenSSLContext within Apache Tomcat, suggest that their high metrics, for example, HCC or CCD, are associated with code smells that include high cyclomatic complexity and tight coupling. These problems result in the reduction of modularity because modifications of these features affect other aspects of the application. On the other hand, the modules that have less coupling and clear division of work, as seen in relatively simple Groovy plugins where there has been an effort to simplify and compartmentalize the code or implementations of design patterns where the patterns intend to improve and enhance modularity, can provide better modularity.

Impact on Modularity:

Close coupling between the elements and high internal complexity is quite damaging to modularity as it hampers the basic ideas of modularity, which are the ability to separate different concerns and evolve different aspects of the system individually. For instance, when referring to a few examples like the openssl-foreign module of Apache Tomcat in which there are highly complex as well as high method count constituents such as OpenSSLContext, it is hard to sustain modularity. Such modules might need to be refactored to optimize where there’s duplication and simplify where there is coupling between classes, this will enhance modularity from module to module and across the system.

The comparison also emphasizes the proper approach to controlling code complexity and coupling to maintain modularity within the software project. High options for both complexity and coupling mean that the projects may develop code flaws that will affect the modularity and maintainability of the project. Identifying these issues through the refactoring process, adhering to design principles, and providing a clean separation between concerns can reduce code smells and improve modularity. Therefore, continuing the work on enhancing such high-complexity modules' design and architecture will become a key priority to guarantee the maintainability, evolvability, and scalability of these multiple software projects.

5.2. References:

1. Apache Dubbo. (n.d.). In Advanced Java Team, *doocs/advanced-java*. Retrieved June 16, 2024, from https://github.com/doocs/advanced-java
2. Apache Groovy. (n.d.). In Apache Software Foundation, *groovy*. Retrieved June 16, 2024, from https://github.com/apache/groovy
3. Apache OpenNLP. (n.d.). In Apache Software Foundation, *opennlp*. Retrieved June 16, 2024, from https://github.com/apache/opennlp
4. Apache Tomcat. (n.d.). In Apache Software Foundation, *tomcat*. Retrieved June 16, 2024, from https://github.com/apache/tomcat
5. BazelBuild. (n.d.). *Extensible build system Bazel*. Retrieved June 16, 2024, from https://github.com/bazelbuild/bazel
6. Bieman, J. M., & Kang, B. K. (1995). Cohesion and reuse in an object-oriented system. *Journal of Object-Oriented Programming*, *8*(1), 48-53. Retrieved from https://dl.acm.org/doi/abs/10.5555/195503.195509
7. Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering, 20*(6), 476-493. https://doi.org/10.1109/32.301774
8. Design Patterns implemented in Java. (n.d.). In Iluwatar, *java-design-patterns*. Retrieved June 16, 2024, from https://github.com/iluwatar/java-design-patterns
9. Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.
10. H2 Database Team. (n.d.). *H2 Database*. Retrieved June 16, 2024, from https://github.com/h2database/h2database
11. JavaParser Team. (n.d.). *Java 1-21 Parser and Abstract Syntax Tree*. Retrieved June 16, 2024, from https://github.com/javaparser/javaparser
12. McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, *SE-2*(4), 308-320. https://doi.org/10.1109/TSE.1976.233837
13. Mirror of Apache Struts. (n.d.). In Apache Software Foundation, *struts*. Retrieved June 16, 2024, from https://github.com/apache/struts
14. Moha, N., Gueheneuc, Y. G., Duchien, L., & Le Meur, A. F. (2010). JDeodorant: Identification and removal of feature envy bad smells. In *2010 17th Working Conference on Reverse Engineering (WCRE)* (pp. 319-320). IEEE.

15. Netty Project. (n.d.). Retrieved June 16, 2024, from https://github.com/netty/netty
16. TutorialsPoint. (n.d.). *Execute Matplotlib online*. Retrieved from https://www.tutorialspoint.com/execute_matplotlib_online.php