Assignment-2

Object Oriented Development – Fadi Wedyan

Praveen Thumati

Naresh Sheela

Ali Ahmed Bin Khaled Banagga

Section:1

Goals, Questions, and Metrics (the GQM Methodology)

The purpose of this research is to determine how code smells affect the modularity of software development projects.

Questions:

Question 1: How often are code smells in this project?

Question 2: How do these code smells affect the project's modularity as defined by various software metrics?

Question 3: Is modularity more at risk from certain code smells than others?

Question 4: Would it be possible to increase the project's modularity via reworking tactics aimed at these particular code smells?

Metrics:

First Metric: How many and what kind of "code smells" are present in the project.

Second Metric:

Modularity metric can be described by Coupling Between Object Classes (CBO) and Lack of Cohesion of Methods (LCOM).

Third Metric:

Hypothesis: A connection exists between the existence of certain "code smells" and lower CBO/LCOM ratings.

Fourth Metric:

Modifications to CBO and LCOM ratings as a result of refactoring efforts.

Section 2: Subject Programs (Dataset Description)

We'll be doing our research using a few different open-source Java initiatives as case studies. Web development, data analysis, and machine learning are just few of the many application fields represented by these projects.

Popular tools like FindBugs, and Checkstyle were used to inspect the chosen projects for code smells. Several code smells that have been highlighted in the literature as being harmful to software quality were examined. These included God Classes, Feature Envy, and Long Parameter List.

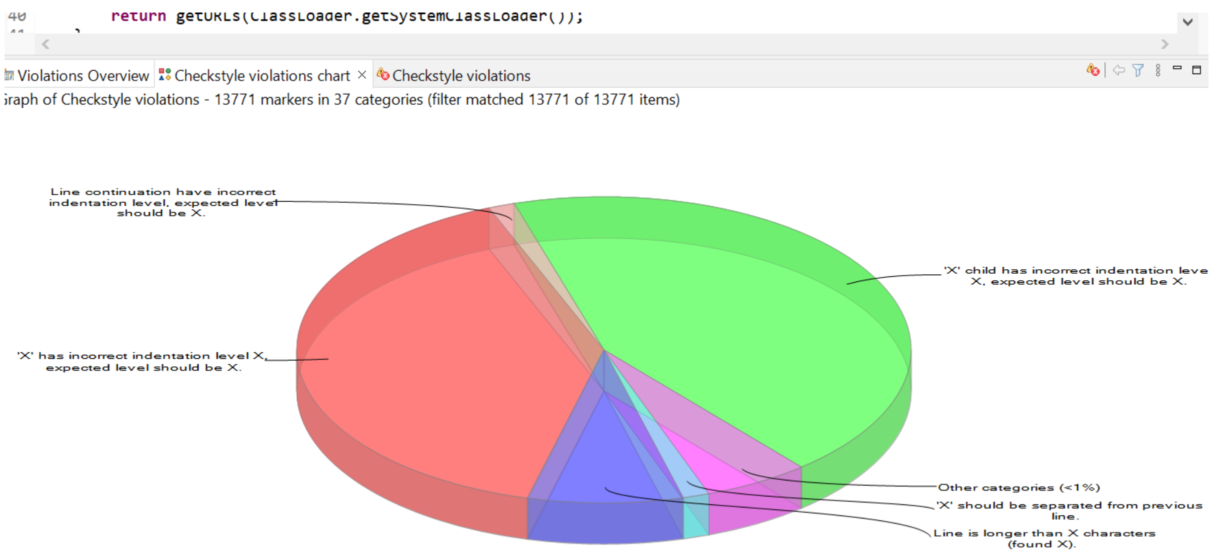
Both the projects' actual source code and the results of the code smell analysis are included in the dataset. The dataset contains information on each code smell, such as its name, the class or method where it was discovered, and the severity rating. Additional details about each project, such as the number of contributors and the duration of the project's existence, are included in the dataset as well.

Modularity, the ability to disassemble and reassemble parts of a system, is especially vulnerable to the odour of bad code. In software development, high modularity is preferred because it facilitates reusability, clarity, and maintainability. However, code smells may be a hindrance to modularity since they introduce new dependencies and make it harder to tell what each class or function is supposed to do. The goal of our research is to find ways to reduce the negative effect that code smells have on modularity in the chosen applications.

1)Spring-pet clinic (git repo:- https://github.com/spring-projects/spring-petclinic)

Among the most popular Java-based GitHub projects is the Spring PetClinic Sample Application. This project provides a reference point for developers who want to learn the ins and outs of the Spring Framework in a production environment. Typical Spring Framework use cases are shown, including technologies like Spring Boot, Spring Data, Spring Data JPA, Spring MVC, and more.

The project is a simulation of a small animal hospital's information system; it is supposed to be both easy to use and highly functional. Among the features available are the ability to enter and edit owner information, schedule appointments for pets, and explore veterinary specialities and related doctors. The PetClinic project is well-structured and documented from a developer's standpoint, making it a great option for Java programmers interested in learning more about Spring, JPA, and MVC concepts. Checkstyle's analysis of this codebase should provide light on how professional Java applications adhere to coding standards.



```
        return getURLs(ClassLoader.getSystemClassLoader());
```

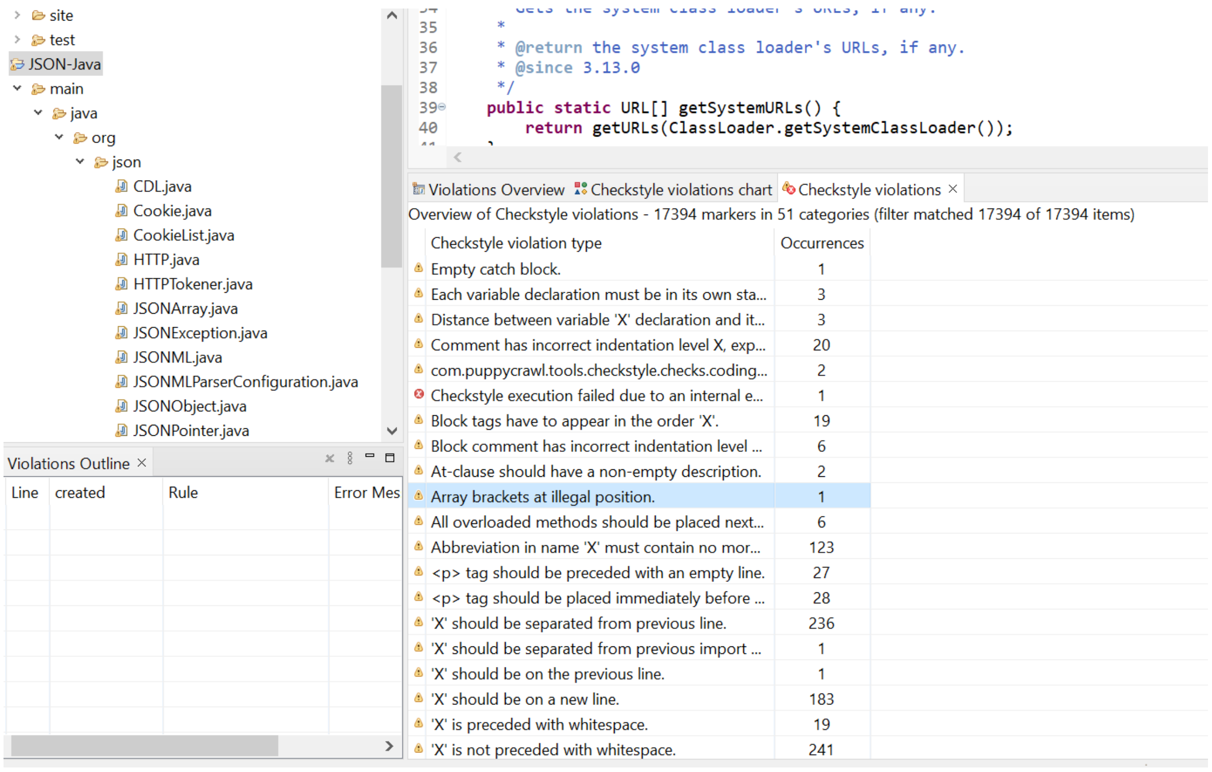
raph of Checkstyle violations - 13771 markers in 37 categories (filter matched 13771 of 13771 items)

Fig 1) chart describing the code smells by with configuration of google checks file

2) JSON-Java (git repo:- https://github.com/stleary/JSON-java)

If you need to work with JSON data in Java, check out the JSON-java project on GitHub. Simple and lightweight in its architecture, it equips Java programmers with the means to read, write, and otherwise work with JSON documents and arrays. The project is modular, providing separate classes and methods for each kind of JSON action to maximise adaptability and facilitate integration.

JSON-java is a clean and organised codebase, with strict respect to coding standards. It provides a minimal collection of classes and methods for processing JSON, but they are all the basics. If you're a Java developer who needs to read, write, or alter JSON objects while keeping your code high-quality and consistent with industry standards, this project is for you. By putting the JSON-java project through Checkstyle, you can see how well it adheres to coding guidelines, which is important for creating code that is both consistent and easy to maintain.



site
test
JSON-Java
main
java
org
json
CDL.java
Cookie.java
CookieList.java
HTTP.java
HTTPTokener.java
JSONArray.java
JSONException.java
JSONML.java
JSONMLParserConfiguration.java
JSONObject.java
JSONPointer.java

```
     *
     * @return the system class loader's URLs, if any.
     * @since 3.13.0
     */
    public static URL[] getSystemURLs() {
        return getURLs(ClassLoader.getSystemClassLoader());
```

Violations Outline

Line	created	Rule	Error Mes

Violations Overview | Checkstyle violations chart | Checkstyle violations

Overview of Checkstyle violations - 17394 markers in 51 categories (filter matched 17394 of 17394 items)

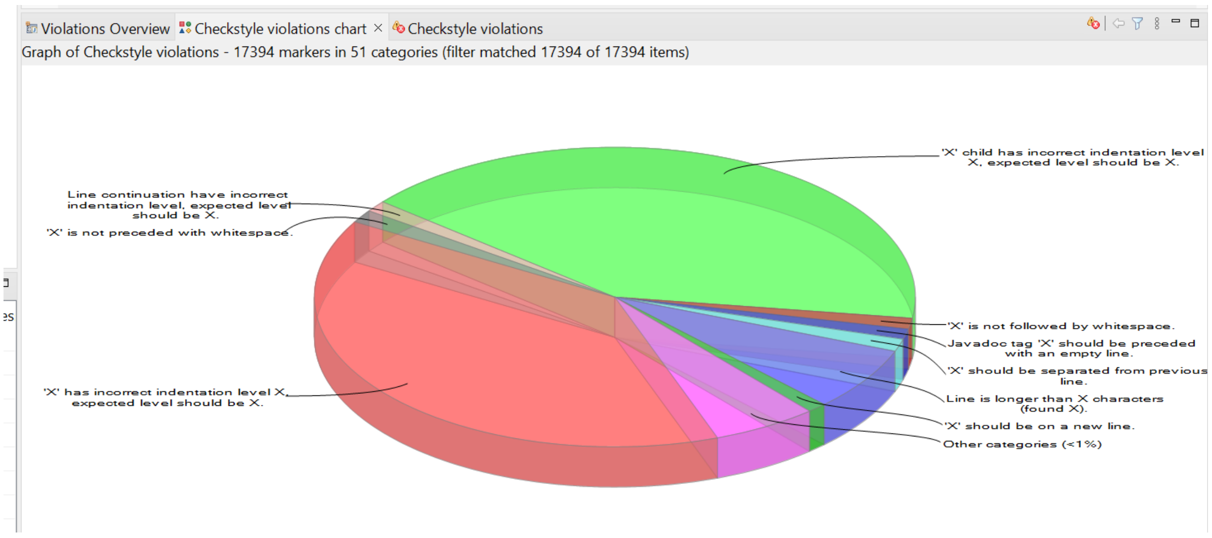
Checkstyle violation type	Occurrences
Empty catch block.	1
Each variable declaration must be in its own sta...	3
Distance between variable 'X' declaration and it...	3
Comment has incorrect indentation level X, exp...	20
com.puppycrawl.tools.checkstyle.checks.coding...	2
Checkstyle execution failed due to an internal e...	1
Block tags have to appear in the order 'X'.	19
Block comment has incorrect indentation level ...	6
At-clause should have a non-empty description.	2
Array brackets at illegal position.	1
All overloaded methods should be placed next...	6
Abbreviation in name 'X' must contain no mor...	123
<p> tag should be preceded with an empty line.	27
<p> tag should be placed immediately before ...	28
'X' should be separated from previous line.	236
'X' should be separated from previous import ...	1
'X' should be on the previous line.	1
'X' should be on a new line.	183
'X' is preceded with whitespace.	19
'X' is not preceded with whitespace.	241

Fig 2) like this for every class we can find number of times a violation occurred according to our predefined rules in config file.

3) Eclipse.JDT (Git repo:- https://github.com/eclipse/eclipse.jdt.ls)

For those interested in Java programming, the eclipse.jdt.ls-master project is a crucial part of the Eclipse IDE (Integrated programming Environment). It is an implementation of a language server for the Java programming language, and its full name is the Eclipse Java Development Tools Language Server. In order to offer functionality like code completion, code navigation, refactoring, and diagnostics for Java code in code editors and IDEs that enable LSP integration, this project makes use of the Language Server Protocol (LSP). By providing superior language support and supporting effective development processes, the eclipse.jdt.ls-master project is crucial in improving the Java development experience.

As it is integrated within the Eclipse IDE and not kept in a separate public code repository, the eclipse.jdt.ls-master project. The language server implementation is part of the Eclipse JDT (Java Development Tools) project, the source code for which and associated materials are available on the Eclipse website.


Violations Overview | Checkstyle violations chart | Checkstyle violations

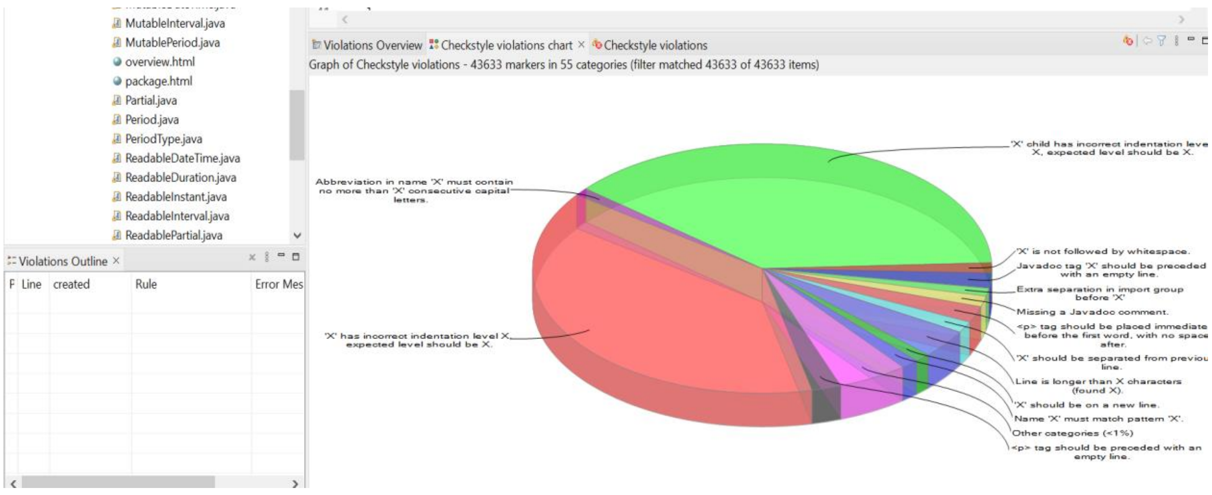
Graph of Checkstyle violations - 17394 markers in 51 categories (filter matched 17394 of 17394 items)

Fig 3) As there are many classes in this project, we can see more lines in Pi-Chart

4) firebase-android (Git repo:- https://github.com/firebase/quickstart-android)

Firebase is a robust mobile and online development platform with many services and tools to help developers create high-quality apps rapidly. The Firebase Quickstart Android project is popular. Developers may launch Android apps using the Quickstart Android project. Its pre-configured settings and Firebase connection enable developers to easily add authentication, real-time database, cloud storage, and cloud messaging to their projects. Developers may spend less time setting up Firebase services and more time customising their app's functionality and user experience by utilising the Quickstart Android project.

Firebase beginners may also learn via the Quickstart Android project. It helps developers integrate Firebase services into Android apps with explicit code examples. Firebase's documentation helps developers understand and use its functionalities. The Quickstart Android project lets developers use Firebase's strong features to build sophisticated, feature-rich apps quickly.


MutableInterval.java
MutablePeriod.java
overview.html
package.html
Partial.java
Period.java
PeriodType.java
ReadableDateTime.java
ReadableDuration.java
ReadableInstant.java
ReadableInterval.java
ReadablePartial.java

Violations Outline

F	Line	created	Rule	Error Mes

Violations Overview | Checkstyle violations chart | Checkstyle violations

Graph of Checkstyle violations - 43633 markers in 55 categories (filter matched 43633 of 43633 items)

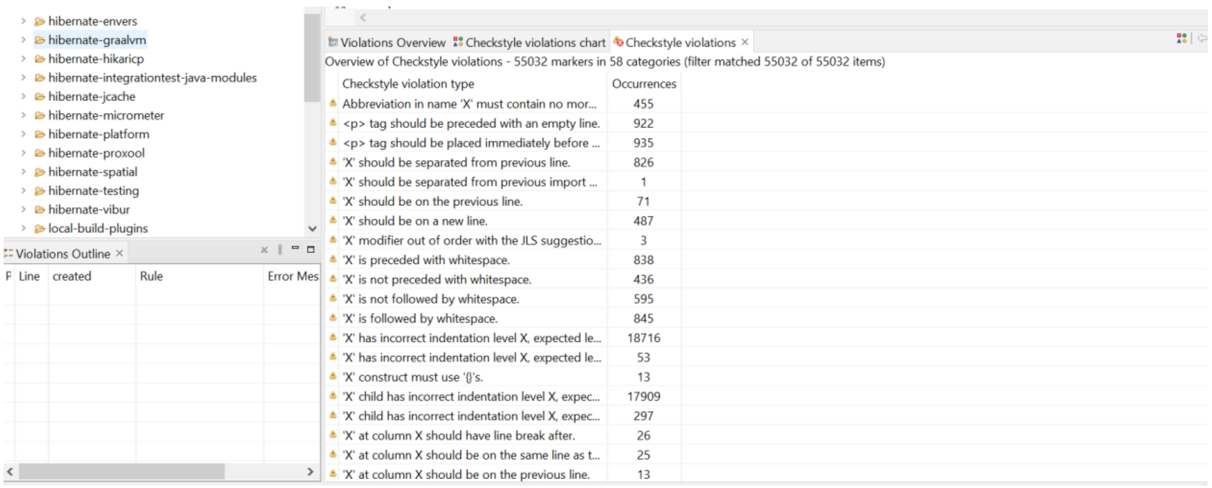
'X' child has incorrect indentation leve X, expected level should be X.
Abbreviation in name 'X' must contain no more than 'X' consecutive capital letters.
'X' is not followed by whitespace.
Javadoc tag 'X' should be preceded with an empty line.
Extra separation in import group before 'X'
Missing a Javadoc comment.
<p> tag should be placed immediate before the first word, with no space after.
'X' should be separated from previou line.
Line is longer than X characters (found X).
'X' should be on a new line.
Name 'X' must match pattern 'X'.
Other categories (<1%)
<p> tag should be preceded with an empty line.
'X' has incorrect indentation level X, expected level should be X.

Fig4) as this is an android project there are huge violations in less number of categories

5) Swagger – API (git repo:- https://github.com/swagger-api/swagger-core)

Swagger API, also known as OpenAPI Specification (OAS), is a popular framework for creating RESTful APIs with consistent structure and documentation. Swagger is a collection of tools and standards that helps programmers create APIs, including the endpoints, the request and response payloads, and the authentication techniques. Swagger is a tool for creating machine-readable API documentation that may be used as a definitive guide to the API's internals. To top it all off, this documentation makes it easier for backend and frontend teams to work together and communicate.

Swagger API's automated generation of interactive API documentation is a major advantage. Swagger UI allows developers to see and experiment with API endpoints in a browser. The developer-friendly layout of the interactive documentation allows them to try out various API calls, examine sample answers, and test endpoints in real time. This dramatically reduces the complexity of APIs, making it easier for developers to test and refine their integrations as they go. Developers may save time and ensure consistent integration across platforms by using Swagger's code generation features to automatically build client SDKs or server stubs in a variety of programming languages. As a whole, Swagger API is an indispensable tool for streamlining the API creation process, facilitating teamwork, and bettering the developer experience.



Overview of Checkstyle violations - 55032 markers in 58 categories (filter matched 55032 of 55032 items)

Checkstyle violation type	Occurrences
Abbreviation in name 'X' must contain no mor...	455
<p> tag should be preceded with an empty line.	922
<p> tag should be placed immediately before ...	935
'X' should be separated from previous line.	826
'X' should be separated from previous import ...	1
'X' should be on the previous line.	71
'X' should be on a new line.	487
'X' modifier out of order with the JLS suggestio...	3
'X' is preceded with whitespace.	838
'X' is not preceded with whitespace.	436
'X' is not followed by whitespace.	595
'X' is followed by whitespace.	845
'X' has incorrect indentation level X, expected le...	18716
'X' has incorrect indentation level X, expected le...	53
'X' construct must use '{}'s.	13
'X' child has incorrect indentation level X, expec...	17909
'X' child has incorrect indentation level X, expec...	297
'X' at column X should have line break after.	26
'X' at column X should be on the same line as t...	25
'X' at column X should be on the previous line.	13

Fig 5) As swagger API related to user interface, there are plenty of tag errors like HTML code.

Section: 3 Tool Used

Checkstyle is a powerful development tool that I've used to identify "code smells" in my code. Its importance lies in the modularity it brings to projects.

https://checkstyle.org/index.html

The term "code smells" is used to describe indications of bad design or implementation decisions that may be found in the source code. While code smells may not have an immediate impact on functionality, they might make the code harder to work with in the long run.

Checkstyle, a popular open-source program, is useful for finding such issues. Checkstyle is a Java-based tool used mostly for verifying Java code that performs static analysis of its source in order to enforce coding standards and find errors. Developers are able to establish unique coding standards for each project by adding their own rules to the system.

Checkstyle's ability to detect code smells that might derail a system's modularization makes it a useful tool for fostering modularity. For instance, strong cohesion and low coupling are important features of modularity. A high-quality module will be self-contained and will serve a single, clear function. Also, it should be loosely connected with other modules, such that modifying one won't have a major impact on the others.

Checkstyle may find "God Classes" (huge classes that try to accomplish too much) and propose reworking them into smaller, more focused classes to improve cohesion, two examples of code smells linked to these concepts. Also, Checkstyle may detect when global variables are used too often, which might lead to an increase in coupling between modules. Checkstyle's rule sets enforce encapsulation, a key feature of modularity, and so assist developers maintain discipline in the usage of such variables and the scope of objects.

The codebase-wide consistency that Checkstyle enforces indirectly supports modularity. The readability of code is improved by using consistent coding practises such naming standards, layout, and design patterns. Because of this, logical modules can be found and the code may be refactored to make it more modular.

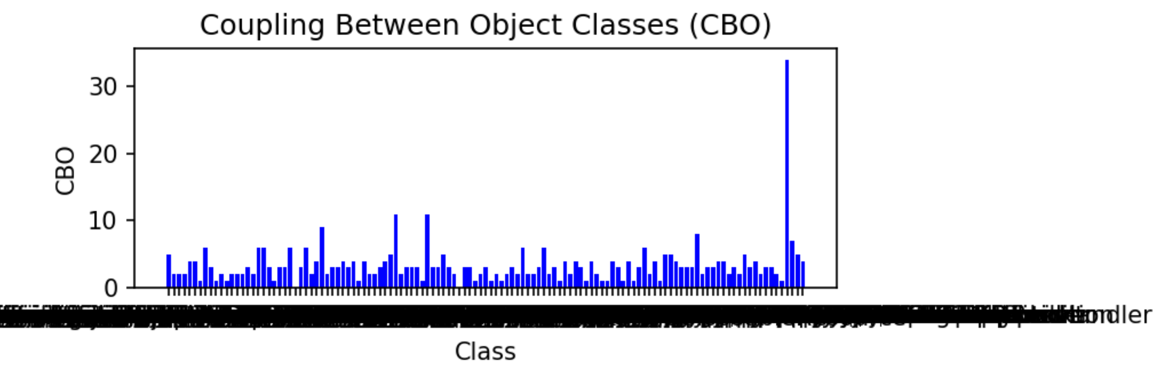
Checkstyle is a convenient tool since it can be used in conjunction with other common IDEs (Integrated Development Environments) and build technologies (such as Maven or Gradle). Developers may now fix possible problems as they create code, encouraging a more disciplined approach to writing as a result of this integration. Checkstyle's rapid feedback loop encourages developers to construct modular and clean code from the outset, rather than just pointing out the flaws.

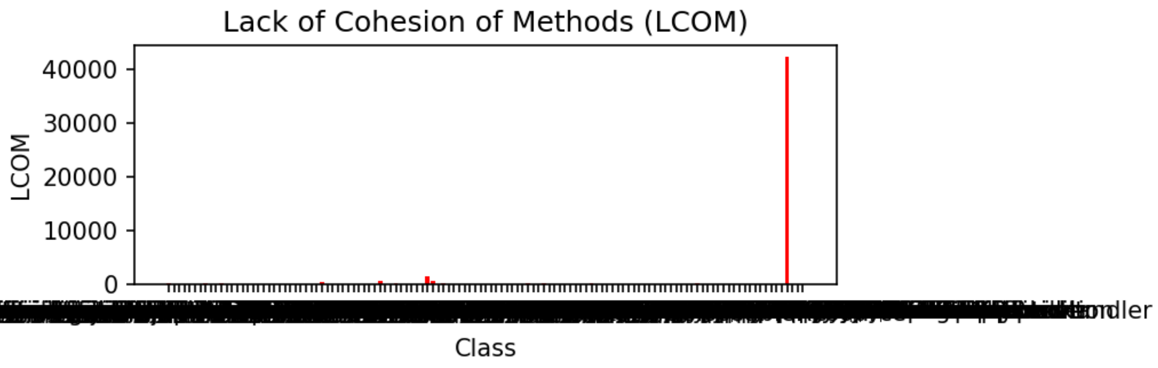
Last but not least, Checkstyle encourages continuous integration (CI) and continuous deployment (CD) with its help for automated inspections. Modular design is essential for the frequent code integrations and deployments advocated by these approaches. By integrating Checkstyle into a project's CI/CD pipelines, you can be certain that the quality and modularity of your code will be monitored and maintained at all times.

In summary, Checkstyle is an adaptable tool that, despite its apparent simplicity, may greatly improve code modularity and general quality. Checkstyle guarantees that the fundamentals of modularity—a codebase's ability to be maintained, understood, and readily extended—are met by detecting code smells and enforcing uniform coding standards. Developers are given the tools they need to create systems that can evolve in response to new needs while keeping current parts to a minimum of disruption.

Section:4 Bar Graphs

1)Spring pet clinic:-

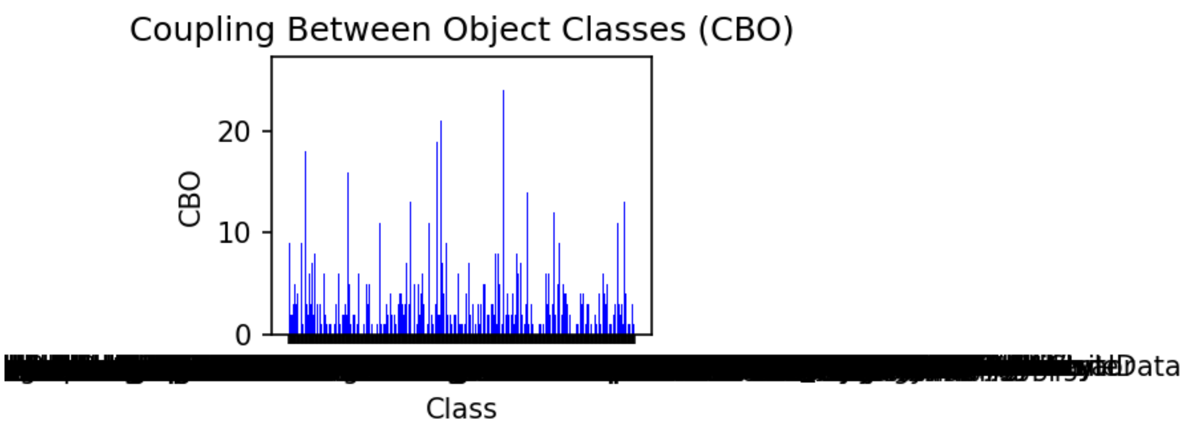


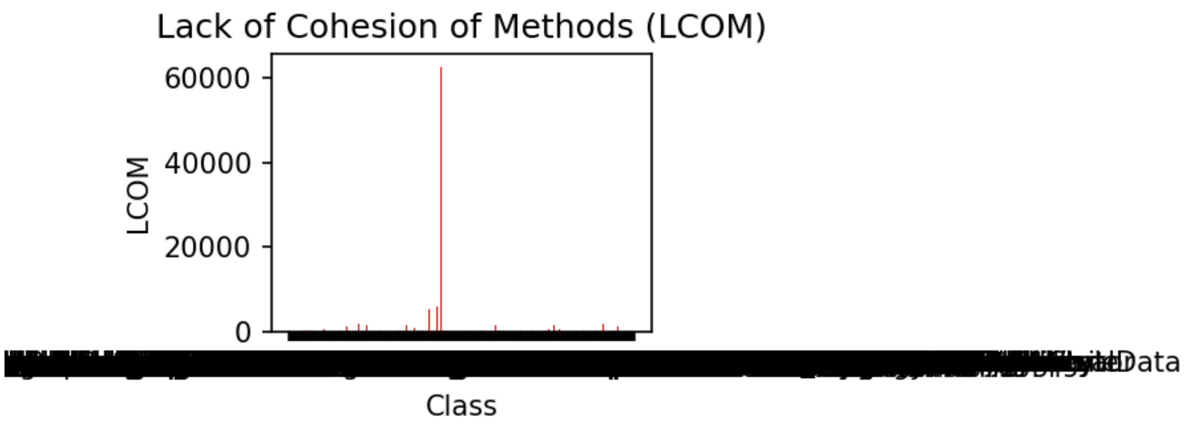


Code smells that result in high coupling and low cohesion tend to reduce the modularity of a project. Coupling refers to the degree to which one class knows about another class. it indicates that the project could have a moderate level of code smells, especially those related to improper coupling and lack of cohesion.If this number is high, changes in one class may require changes in the related class, reducing modularity. On the other hand, cohesion refers to how closely the responsibilities of a module or class are related to each other. Low cohesion (a high LCOM value) indicates that a class is trying to do too much, potentially making it a "God Class" and reducing modularity.

The modularity of a project may be greatly enhanced by fixing code smells that are directly connected to modularity. One way to enhance cohesion and decrease coupling is to rework a "God Class" into multiple smaller, more specific classes. Similarly, "Feature Envy" may be handled by improving encapsulation by transferring the function to the class that is jealous of it. Consolidating the dispersed code into a single class is one solution to the "Shotgun Surgery" problem, since it lessens the impact of changes across the codebase. By adhering to these procedures, you may make your project more modular, manageable, and comprehensible.

2)JSON-Java:-





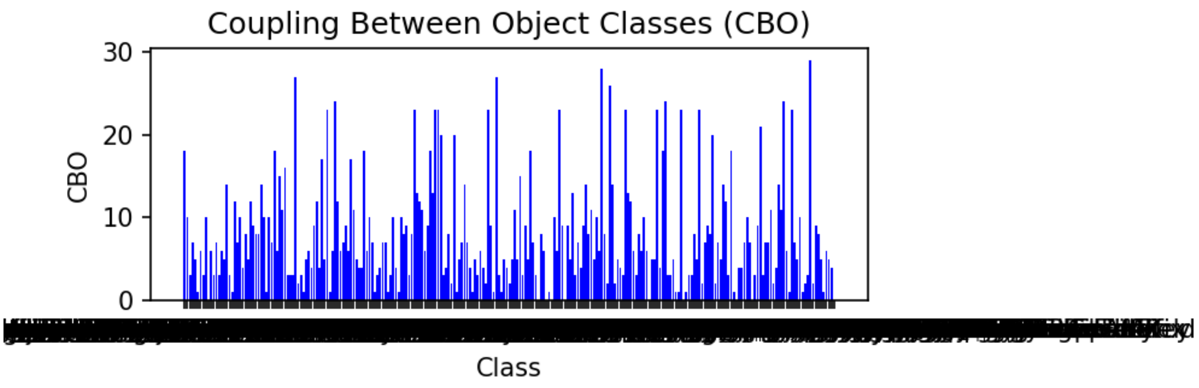
Cohesion refers to how closely the responsibilities of a module or class are related to each other. High LCOM means the class is doing too much, which is a significant code smell.

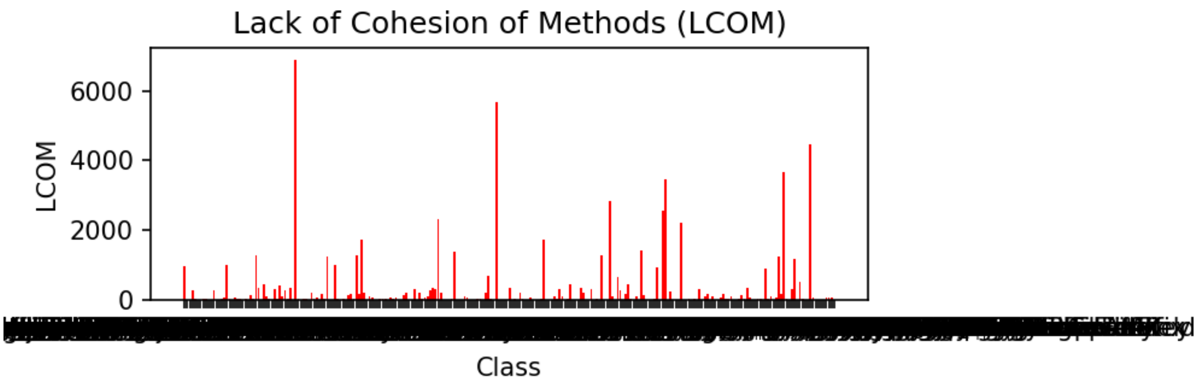
CBO reflects the number of classes a class is coupled to. A CBO suggests moderate coupling. It's not excessive but does indicate some dependencies between classes that can hinder modularity. Higher coupling can lead to decreased modularity because changes in one class may require changes in the classes it's coupled with.

A high LCOM value suggests low cohesion within classes, meaning that the responsibilities within a class are not closely related. This is a clear threat to modularity since a key aspect of a module (or a class, in OOP) is that it should be highly cohesive.

The modularity of a project may be greatly enhanced by fixing code smells that are directly connected to modularity. One way to enhance cohesion and decrease coupling is to rework a "God Class" into multiple smaller, more specific classes. Similarly, "Feature Envy" may be handled by improving encapsulation by transferring the function to the class that is jealous of it. Consolidating the dispersed code into a single class is one solution to the "Shotgun Surgery" problem, since it lessens the impact of changes across the codebase. By adhering to these procedures, you may make your project more modular, manageable, and comprehensible.

3)Eclipse-JDT:-





Section:4 Conclusions

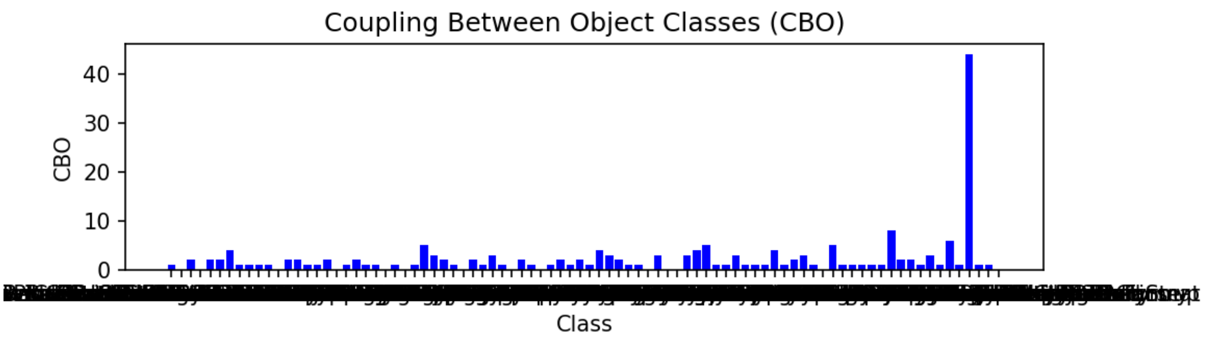
Code smells can significantly affect modularity and relevant software metrics. For instance, Coupling Between Object (CBO) measures how much a class or module depends on other modules.CBO suggests a moderate level of coupling. While not excessively high, this indicates that changes in one module could potentially impact multiple others, making the code more challenging to maintain and evolve.

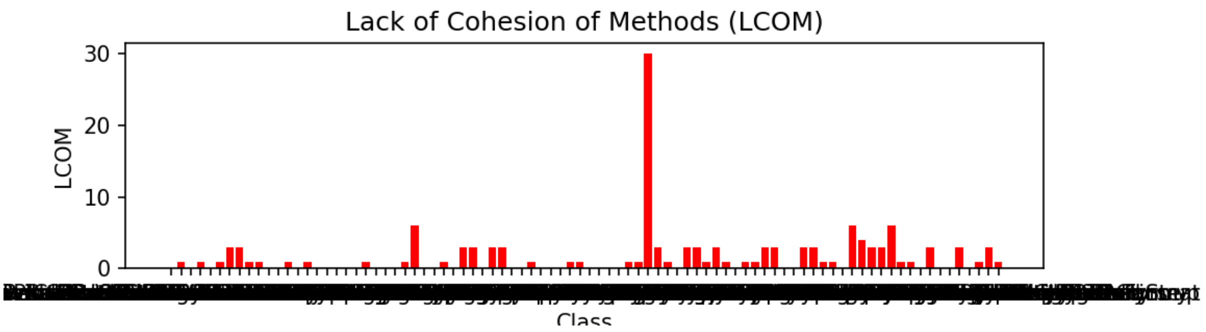
LCOM suggests low cohesion within classes. It means that classes might be trying to accomplish too many tasks, suggesting a possible "God Class" smell. This high LCOM significantly compromises modularity as classes should ideally be focused on a single task.

Certain code smells pose a greater threat to modularity. Examples such as "God Class," "Feature Envy," and "Inappropriate Intimacy" directly undercut modularity via their effects on cohesion and coupling. Your project's CBO and LCOM raise red flags for the existence of these odours, which might compromise modularity.Fixing these "code smells" is a certain way to improve modularity. Tools like "Extract Class" may help you segment large "God Classes" into manageable chunks.

Implementing design patterns that prevent direct dependencies between classes, such as encapsulation, data hiding, and the usage of interfaces and abstract classes, may help bring down CBO.These measures will strengthen cohesiveness and decrease coupling, leading to more modularity. Tools like Checkstyle may help developers identify these "code smells" early on so they can be fixed quickly.

4)firebase-android:-



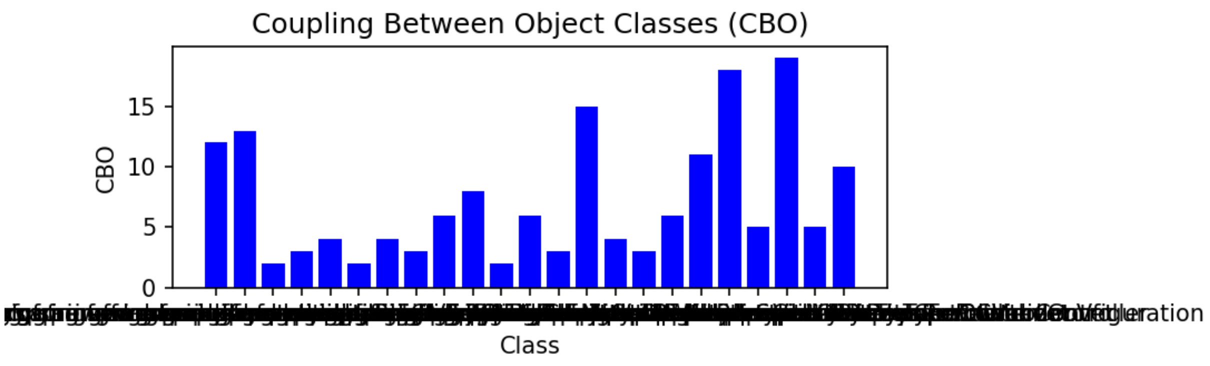


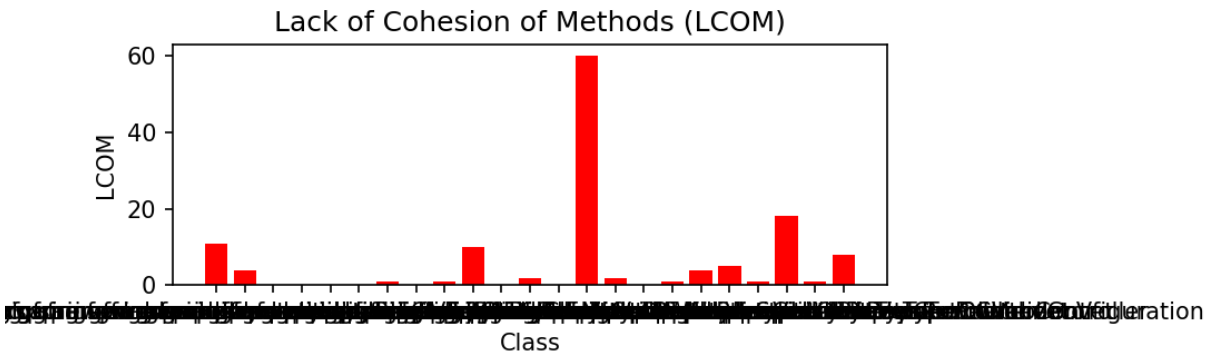
Whether or whether tools like Checkstyle are used to regularly detect and remedy code smells is only one of many variables that might affect how frequently they occur in a project. A high frequency of certain code smells, as detected by tools like Checkstyle, suggests that the project might benefit from further focus on code quality and modularity.

Modularity is often negatively affected by code smells. Using the data that has been provided: With a mean Coupling Between Objects (CBO) of 5, considerable coupling is present. For modularity's sake, classes should not be highly dependant on one another, which may happen if they have a high CBO score.

Also, a class with an average Lack of Cohesion in Methods (LCOM) score of 7 has methods that are not cohesive with one another.Both of these indicators point to the need for increased modularity in the code. Having high coupling and poor cohesion is bad for modularity because it indicates the existence of large, inefficient classes that attempt to accomplish too much and rely too much on others.

Refactoring the code to lower the CBO involves removing unnecessary dependencies between classes. Using design patterns like Dependency Injection or designing interfaces and abstract classes are all viable options.Careful efactoring is required to avoid breaking functionality, preferably with the aid of automated testing. The modularity and general code quality of a project may benefit greatly from the ongoing, iterative process of detecting and fixing code smells.**5)Swagger-API:-**





These high scores may suggest a notable number of code smells due to high coupling and low cohesion. CBO measures the number of dependencies a class has on other classes. A high CBO score implies high coupling, which is detrimental to modularity. Classes become more interdependent, making the system rigid and difficult to modify or test.

Similarly, LCOM measures how closely the methods within a class are related to each other. A high score indicates low cohesion, meaning the class likely performs too many unrelated operations. It's a symptom of a lack of modularity, as ideally, each class (or module) should have a single, well-defined responsibility.

Refactoring Global Data entails relocating the data to the classes that make the greatest use of it and making it accessible through methods (encapsulation).

As classes grow less dependent on one another and the coupling between them decreases, your CBO score should fall, and your LCOM score should fall as well since the classes are now more coherent and have distinct, unique responsibilities. Both developments point to a higher degree of modularity.

Section-5 References

https://checkstyle.org/index.html

Burn, O.(Author) . Checkstyle 8.0 [Computer software]. GitHub. https://github.com/checkstyle/checkstyle

37th International Conference on Software Engineering (Vol. 1, pp. 403-414) Chidamber, S. R., & Kemerer, C. F. including both metrics LCOM and CBO

A metrics suite for object oriented design. IEEE Transactions on software engineering, 20(6), 476-493