UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

GLAUBER DE SOUZA ROSA

A Study on Prompt Engineering for Software Engineering Data: ChatGPT integration into Software Engineering Metric Generation Tool

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Marcelo Pimenta
Coorientador: Prof. Me. Guilherme Lacerda

Porto Alegre
2024

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos André Bulhões Mendes
Vice-Reitor: Prof. Patricia Pranke
Pró-Reitor de Graduação: Prof. Cíntia Inês Boll
Diretor do Instituto de Informática: Prof. Carla Maria Dal Sasso Freitas
Coordenador do Curso de Engenharia de Computação: Prof. Walter Fetter Lages
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

Nesta singela nota eu gostaria de agradecer primeiramente aos meus pais, Mariza e Clodomiro, que sempre me apoiaram e incentivaram, priorizando meus estudos e sempre proporcionando o melhor possível, não importava o curso que eu desejasse fazer, desde alemão a AutoCad e tantos outros. Não importando se teriam que me buscar numa sexta às 23 horas em outra cidade ou acordar cedo num domingo para me levar no vestibular no Campus do Vale. Além de proporcionar todas essas oportunidades, também sempre estiveram dispostos a me ouvir e a dar conselhos, mesmo quando era sobre um assunto que eles não dominassem, acredito que já aprenderam muito só de me ouvir e tentar ajudar. Obrigado por tudo e saibam que vocês foram e serão sempre muito especiais.

Gostaria de agradecer a UFRGS e ao Instituto de Informática da mesma, que proporcionou uma excelente formação assim como oportunidades incríveis, como um intercâmbio na França pelo programa BRAFITEC, onde tive a incrível oportunidade de estudar um ano na Polytech Montpellier.

Gostaria de agradecer ao meu coorientador Prof. Me. Guilherme Lacerda, que se fez disponível para conversarmos, passar referências e explicar o produto de sua tese de doutorado, DR-Tools, que foi utilizado nesse trabalho. Também gostaria de agradecer ao meu orientador Prof. Dr. Professor Marcelo Pimenta, que auxiliou a ajustar o tema deste trabalho assim como em momentos chave prover um norte para a sua conclusão.

Por último e talvez mais importante, gostaria de agradecer à linguista aplicada, doutoranda em letras, professora de Instituto Federal, que neste trabalho foi também incentivadora, ouvinte, revisora, editora e quase uma coorientadora e que eu tenho orgulho de chamar de minha namorada! Prof. Ma. Kaiane Mendel, tu foste muito importante nesse trabalho, assim como és na minha vida. Muito obrigado! Espero poder te ajudar tanto quanto tu me ajudas, nos momentos fáceis e difíceis, nessa montanha-russa que é a vida! P.S.: Inclusive ela me orientou a agradecer as gatinhas Nina e Mimi.

Muito obrigado,

Glauber

Estudo sobre Engenharia de Prompt para Dados de Engenharia de Software: Integração do ChatGPT com uma Ferramenta de Geração de Métricas de Engenharia de Software

RESUMO

Com o avanço constante da tecnologia e a contínua transformação digital, a inteligência artificial tem se tornado cada vez mais presente em diversos aspectos de nossas vidas. O ChatGPT é baseado na arquitetura GPT (Generative Pre-trained Transformer), e é um exemplo de sistema que pode ser treinado em uma variedade de tópicos e áreas de conhecimento, o que permite que ele gere respostas precisas e úteis em uma variedade de situações. Este trabalho visa investigar como integrar o ChatGPT (e como usá-lo de maneira mais eficaz) à uma suíte de ferramentas de análise de código que usa métricas e indicadores sobre *code smells* e qualidade de código – a DR-Tools Suite. O objetivo é que tal integração produza orientações para o engenheiro de software em relação à qualidade do código ou até mesmo algum nível de automação em apoio a seu trabalho de refatoração. O trabalho resume as características principais das tecnologias envolvidas, desenvolve a engenharia de prompt para o trabalho, apresenta a abordagem de integração e alguns exemplos de uso, e finalmente discute os resultados obtidos. Por fim, apresenta melhores práticas para a engenharia de prompt, avalia os potenciais das tecnologias e o propõe potenciais avanços em futuras pesquisas.

Palavras-chave: ChatGPT. Engenharia de software. Refatoração. Grande Modelo de Linguagem.

ABSTRACT

With constant innovation in technology and continuous digital transformation, Artificial Intelligence is becoming more present in different aspects of our lives. The ChatGPT is based on the GPT (Generative Pre-trained Transformer) architecture and is a good example of a system that can be trained in a variety of topics and knowledge areas, what allows it to generate precise and useful answers on multiple situations. This project aims to investigate how to integrate ChatGPT (and how to use it in the most effective way) into a code analyzer and smells detection tool suite – the DR-Tools Suite. The intention is that this integration provides guidance to the software engineer in relation to the code quality and even some automation to his/her refactoring job. This work summarizes the characteristics of the main technologies involved, presents the integration approach and some use cases, and finally discusses the results achieved. It concludes by presenting best practices for prompt engineering, evaluating the potential of the technologies, and proposing potential future research.

Keywords: ChatGPT. Software engineering. Refactoring. Large Language Model (LLM).

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
API	Application Programming Interface
CLI	Command-Line Interface
GPT	Generative Pre-trained Transformer
IT	Information Technology
LLM	Large Language Model
UML	Unified Modeling Language

SUMMARY

1 INTRODUCTION

Software maintenance and constant updates currently are a big part of software development, as it is needed to comply with new regulatory requirements or corrections to adapt to new needs. Society is at a point when much of the codes used throughout applications are over 10 years old and, in some cases, more than 25 years old, so that software maintenance becomes by each day a more fundamental part of society. Soon, major maintenance will be needed, such as adding support to digits to US phone numbers or US Social Security numbers. There were already similar situations in the past, like the Year 2000 software bug, in which it is estimated that over 75 percent of all software applications were affected by the issue.

It highlights the importance of keeping the software easy to maintain and having tools and automation to help to keep legacy code with quality. Jones (2006) estimates that soon the number of professionals working on maintenance compared to new developments would top 75 percent of all Information Technology (IT) professionals working with software engineering. By the Mid-21st century, maintenance costs could top five trillion dollars overall, which highlights the need for better maintenance tools and technologies to support these activities.

Due to the sizable impact of this issue, there is extensive research on the topic. The previous research explored on this project covered the analytical part of software maintenance through smells and refactoring metrics, which lead to DR-Tools Suite[1] presented on Lacerda et al. (2023). In this work, the tool presents data that provides statistics about software smells and refactoring opportunities for more efficient code maintenance.

While there is this growing need for software maintenance, new advances from Artificial Intelligence (AI) and Large Language Models (LLM), such as OpenAI`s Generative Pre-trained Transformer (GPT) models used on its ChatGPT tool, can be brought to assist on it. ChatGPT is an interactive AI released in 2022 that in a few months proved to be a powerful tool to problem solving and creative production through a precise prompt engineer to guide it. Such tool was already used for the automation of multiple creative activities with different success rates (MA et al., 2024), though we see potential to assist on the software maintenance through providing insights on maintenance or even providing automation.

In this work, we will integrate the two technologies – DR-Tools Suite and ChatGPT - to provide the software engineering community with guidance on how to leverage both and take

[1] Available at https://drtools.site/. DR-Tools Suite will be further presented in section 2.4.

the best synergy possible from them. For instance, using ChatGPT with the statistics from DR-Tools to provide insights into what software engineers should investigate and what could be done to improve the code, or even scenarios in which some automation could be implemented.

1.1 Objectives

The objective of this work is to explore the potential of combining a software engineering analyzer tool like DR-Tools Suite with a LLM like ChatGPT.

A secondary goal is to expand on the existing functionalities of DR-Tools with a proof of concept of using the data from Metrics to provide insights to a software developer performing a code refactoring via a LLM tool like ChatGPT.

1.2 Structure of the text

This work is structured as follows: chapter 2 presents the concepts and tools that base this work. Chapter 3 explains the design challenges and justifies the decisions on the implementation, besides presenting the reader with empirical information on how to use the tools for similar integrations.

With both concepts (and tools) presented and with the design challenges and decisions well stablished, chapter 4 presents the reader with a qualitative analysis of the outcomes, concluding each analysis with a forward-looking perspective based on the results.

The conclusions are presented on chapter 5, which analyzes achievements, scope limitations, perspectives for future works and possible enhancements and additions to this project to come closer to a full-refactoring tool available for developers to use under DR-Tools Suite.

As an extra resource, the more relevant prompts and console outputs referred to in the text are presented in Appendix A, as their full form is not necessary in the text. This approach is to provide the reader with the ability to check on what was the full prompt extracted directly from the console, and even enable some replication of the results by following the same methodology.

2 FUNDAMENTS AND CONCEPTS

This chapter introduces the reader to the fundamental concepts focused and the tools that are subject of this study. Previous related research is presented, especially the ones around the two tools, DR-Tools and ChatGPT. An overview of related works that at some level either approach problems similar to this work or use similar methodologies to resolve problems is presented.

2.1 Code Smells

A smell is a concept used on software engineering for a software problem that is not the same as a bug that would generate a failure, but it is a problem that can impact the software maintenance and future enhancements through increased complexity, for example (LACERDA et al., 2020).

The term "smells" became popular initially with agile software development and was popularized due to the original work of Fowler et al. (1999), which was pioneer in the code smell identification and provided techniques to solve them.

Figure 2.1 - List of code smells presented by Fowler et al. (2018)

Smell	Description
Duplicated Code	Consists of equal or very similar passages in different fragments of the same code base
Long Method/Long Function	Very large method/function and, therefore, difficult to understand, extend and modify. It is very likely that this method has too many responsibilities, hurting one of the principles of a good OO design (*SRP: Single Responsibility Principle* (Martin and Martin, 2007))
Large Class	Class that has many responsibilities and therefore contains many variables and methods. The same *SRP* also applies in this case
Long Parameter List	Extensive parameter list, which makes it difficult to understand and is usually an indication that the method has too many responsibilities. This smell has a strong relationship with *Long Method*
Divergent Change	A single class needs to be changed for many reasons. This is a clear indication that it is not sufficiently cohesive and must be divided
Shotgun Surgery	Opposite to *Divergent Change*, because when it happens a modification, several different classes have to be changed
Feature Envy	When a method is more interested in members of other classes than its own, is a clear sign that it is in the wrong class
Data Clumps	Data structures that always appear together, and when one of the items is not present, the whole set loses its meaning
Primitive Obsession	It represents the situation where primitive types are used in place of light classes
Switch Statements/Repeated Switches	It is not necessarily smells by definition, but when they are widely used, they are usually a sign of problems, especially when used to identify the behavior of an object based on its type
Parallel Inheritance Hierarchies	Existence of two hierarchies of classes that are fully connected, that is, when adding a subclass in one of the hierarchies, it is required that a similar subclass be created in the other
Lazy Class	Classes that do not have sufficient responsibilities and therefore should not exist
Speculative Generality	Code snippets are designed to support future software behavior that is not yet required
Temporary Field	Member-only used in specific situations, and that outside of it has no meaning
Message Chains	One object accesses another, to then access another object belonging to this second, and so on, causing a high coupling between classes
Middle Man	Identified how much a class has almost no logic, as it delegates almost everything to another class
Inappropriate Intimacy	A case where two classes are known too, characterizing a high level of coupling
Alternative Classes with Different Interfaces	One class supports different classes, but their interface is different
Incomplete Class Library	The software uses a library that is not complete, and therefore extensions to that library are required
Data Class	The class that serves only as a container of data, without any behavior. Generally, other classes are responsible for manipulating their data, which is a case of *Feature Envy*
Refused Bequest	It indicates that a subclass does not use inherited data or behaviors
Comments	It cannot be considered a smell by definition but should be used with care as they are generally not required. Whenever it is necessary to insert a comment, it is worth checking if the code cannot be more expressive
Mysterious Name	Non-significant names that do not represent the software elements
Global Data	It can be modified from anywhere in the code base, and there's no mechanism to discover which bit of code touched it
Mutable Data	It changes to data can often lead to unexpected consequences and tricky bugs
Lazy Element	Software elements designed to grow, but do not conform with software evolution
Insider Trading	Coupling problems caused by trade data between modules

Source: Fowler et al. (2018)

Smells can be divided into lower level, known as code level (FOWLER et al., 1999), or higher level, known as design level (BROWN et al., 1998).

Fowler et al. (1999) has originally presented 22 code smells with proposed ways to have it refactored. Later the list was extended by research like Fowler et al. (2018). Figure 2.1 presents both the referred smells from the original work and the 6 additions proposed.

Brown et al. (1998) presented anti-patterns that could be divided into development, architecture, and project management design smells. The anti-patterns describe the common occurrences that could result in negative consequences throughout the code life cycle. On Figure 2.2 is presented a list of the main design smells according to Brown et al. (1998).

Figure 2.2 - List of design smells presented by Brown et al. (1998)

Smell	Description
Blob	As know as *God Class*, is a style of procedural design procedural which brings an object to have too many responsibilities (*Controller*) and attributes with low cohesion, while others only save data or execute simple processes
Lava Flow	Dead code and forgot information frozen with design
Functional Decomposition	A procedural code in a technology that implements the OO paradigm (usually the main function that calls many others), caused by the previous expertise of the developers in a procedural language and little experience in OO
Poltergeist	Classes that have a role and life cycle very limited, frequently starting a process for other objects
Spaghetti Code	Use of classes without structures, long methods without parameters, use of global variables, in addition to not exploiting and preventing the application of OO principles such as inheritance and polymorphism
Cut and Paste Programming	Reused code by a copy of code fragments, generating maintenance problems
Swiss Army Knife	Exposes the high complexity to meet the predictable needs of a part of the system (usually utility classes with many responsibilities)

Source: Brown et al. (1998)

There are subsequential works, for example Wake (2003) and Kerievsky (2004), that expanded with addition of other smells and different perspectives, though this will not be further detailed in this work as the main smells and their perspective were covered already.

2.2 Refactoring

Refactoring, as highlighted by Lacerda et al. (2020), is the primary approach to remove smells (FOWLER et al., 1999). Refactoring is the reorganization strategies to support software change to help to improve code quality by making it more readable, efficient and/or eliminating possible problems, as introduced by Opdyke (1992).

Refactoring can be done on different levels of abstraction and on different software entities. For example, as referred to by Mens et al. (2003), refactoring can be done on the Unified Modeling Language (UML) models, database schemes, software architecture, requirements, and language structure. As refactoring does not change the purpose or the behavior of the software, it can be done on different levels to achieve the best results to have the code supported in the future, which means that different techniques can be used and often be used in a sequence to improve the quality, though its sequence is arbitrary.

Refactoring is usually divided at two levels as smells: high-level (composite refactoring) and low-level (primitive refactoring). High-level refactoring consists of significant and structural design changes at a macro or architectural level, while low-level are small and specific code changes. Opdyke (1992) work defined that to do a high-level refactoring a low-level refactoring will be required, as well as introduced the fundamental elements for the refactoring of both levels, which are the preconditions. The concept of precondition is that it is necessary to establish preconditions which are checked before applying the transformations and, after applied, these conditions are rechecked to guarantee that the behavior of the code is not altered by the refactoring changes, having the same preconditions.

The key importance of performing refactoring on codes that do not present bugs is that 40 percent of the time invested in software maintenance is the cost to understand the code and its architecture (TELEA; VOINEA, 2011). One key strategy is to invest in automation and provide tools for developers to detect refactoring opportunities (or smells), so that the process can be optimized.

2.3 ChatGPT and GPT Concept

ChatGPT, released by OpenAI in November 2022 (NERDYNAV, 2024), is a large-scale language model that once made available reached 100 million users in 3 months and has over 25 million daily users.

OpenAI's GPT models, the tool behind ChatGPT, have been trained to understand natural language and code, in a way that when provided with a text input, it provides a text output in response. These inputs are referred to as "prompts" and their designing is essential to how GPT model will be answering, which will directly influence its content and accuracy.

According to OpenAI, GPTs can be used across a great variety of tasks including content or code generation, summarization, conversation, creative writing, and more.

2.3.1 ChatGPT API

Through its website, OpenAI makes available to developers a public API (Application Programming Interface) that can be used to access the GPT resources. It works by sending a request containing the inputs and the developer API key and receiving a response containing the model's output. The latest models, GPT-4, GPT-4 Turbo and GPT-3.5-turbo are accessed through the chat completions API endpoint.

With the key generated at https://platform.openai.com, it can possibly send requests using https://api.openai.com/v1/chat/completions endpoint through HTTP requests from code (or code language) as far as the HTTP request to the API is done correctly.

The API will be explored later in this work as well as challenges faced during implementation.

2.3.1.1 Models, token count, costs, and usage

Token counting is not a simple concept in which a word or a letter will be a token. For English[2], OpenAI specifies that a token could be as short as a character or as long as a word. Therefore, it is not easy to precisely determine the number of tokens that will be used, though a Phyton library to calculate tokens is provided by OpenAI, which clarify that due to model updates, the library answer may be only an approximation.

On the other hand, by doing a request to ChatGPT API, when it replies, the message contains the data of how many context tokens and how many generated tokens were used on that API request. There is an issue, which is that no message is provided with number of tokens if any token limits are exceeded.

Number of tokens is a fundamental concept when using OpenAI's ChatGPT API, as both the price and the maximum supported prompt are defined based on the number of tokens generated. Table 2.1 presents the context window supported (number of tokens), training data and the costs of each model.

Table 2.1 - ChatGPT Cost table per model

Model	**Context Window**	**Training Data**	**Input Cost**	**Output Cost**
gpt-4-0125-preview	128000 tokens	Up to April 2023	$0.01/1k tokens	$0.03/1k tokens
gpt-4-1106-preview	128000 tokens	Up to April 2023	$0.01/1k tokens	$0.03/1k tokens
gpt-3.5-turbo-1106	16385 tokens	Up to September 2021	$0.0010/1k tokens	$0.0020/1k tokens
gpt-3.5-turbo-instruct	4096 tokens	Up to September 2021	$0.0015/1k tokens	$0.0020/1k tokens

Source: Adapted from OpenAI's website

[2] OpenAI documentation does not specify other languages, limiting to comment that in some languages tokens can be shorter than one character or longer than one word.

Costs are billed per token, meaning that the longer request prompt or the longer the text generated is, higher will be the cost. There are also usage limitations according to how many requests per minute or day it can be done, in order to assure that service is available and manage load on the OpenAI's infrastructure.

To govern this restriction, OpenAI has the concept of usage tiers, which are organization account level classifications based on credits paid combined with time since first payment. There are 5 usage tiers (additionally to the free account) which can be found along with its qualification requisites on Table 2.2. Additionally, Figures 2.3 and 2.4 present the rate limits for usage Tier 1 and Tier 3 respectively, tiers which will appear on this work.

Table 2.2 - ChatGPT API's usage tiers table

Usage Tier	Context Window	Output Cost
gpt-4-0125-preview	128000 tokens	$0.03/1k tokens
gpt-4-1106-preview	128000 tokens	$0.03/1k tokens
gpt-3.5-turbo-1106	16385 tokens	$0.0020/1k tokens
gpt-3.5-turbo-instruct	4096 tokens	$0.0020/1k tokens

Source: Adapted from OpenAI's website

Figure 2.3 - OpenAI's ChatGPT API rate limitation for Tier 1 account

Free | Tier 1 | Tier 2 | Tier 3 | Tier 4 | Tier 5

Tier 1 rate limits

This is a high level summary and there are per-model exceptions to these limits (e.g. some legacy models or models with larger context windows have different rate limits). To view the exact rate limits per model for your account, visit the limits section of your account settings.

MODEL	RPM	RPD	TPM	TPD
gpt-4	500	10,000	10,000	-
gpt-4-turbo-preview*	500	10,000	150,000	500,000
gpt-4-vision-preview*	80	500	10,000	-
gpt-3.5-turbo	3,500	10,000	60,000	-
text-embedding-ada-002	500	10,000	1,000,000	-
whisper-1	50	-	-	-
tts-1	50	-	-	-
tts-1-hd	3	-	-	-
dall-e-2	5 img/min	-	-	-
dall-e-3	5 img/min	-	-	-

Source: Extracted from OpenAI's API Documentation (2024)

Tier 1 is the first paid level, while as more credit is bought and used, the account progresses from tiers. At the end of the project, its account had progressed to Tier 3, therefore more data usage was allowed. The limits for the Tier 3 account can be found below.

Figure 2.4 - OpenAI's ChatGPT API rate limitation for Tier 3 account

Free | Tier 1 | Tier 2 | Tier 3 | Tier 4 | Tier 5

Tier 3 rate limits

This is a high level summary and there are per-model exceptions to these limits (e.g. some legacy models or models with larger context windows have different rate limits). To view the exact rate limits per model for your account, visit the limits section of your account settings.

MODEL	RPM	RPD	TPM	TPD
gpt-4	5,000	-	80,000	-
gpt-4-turbo-preview*	5,000	-	300,000	5,000,000
gpt-4-vision-preview*	120	1,500	40,000	-
gpt-3.5-turbo	3,500	-	160,000	-
text-embedding-ada-002	5,000	-	5,000,000	-
whisper-1	100	-	-	-
tts-1	100	-	-	-
tts-1-hd	7	-	-	-
dall-e-2	100 img/min	-	-	-
dall-e-3	7 img/min	-	-	-

Source: Extracted from OpenAI's API Documentation (2024)

2.3.2 ChatGPT on Development

Currently the implications and the applicability of ChatGPT to code development and to support software activities are being studied. According to studies and recent results of the ChatGPT, increased interest is on the area of automation of software development tasks in a way to assist the developers to perform their tasks more efficiently (ELOUNDOU et al., 2023). Tools like ChatGPT are leading to impressive results, on both quantity and quality, producing outcomes (e.g., code) that are in some cases comparable to what humans produce. For example, Golzadeh et al. (2023) investigations in large open-source projects on GitHub concluded that bots are among the biggest and most active contributors, although without being labeled as bots.

On the empirical study over quality of code between developers and tools like ChatGPT conducted by Nascimento et al. (2023), the result was that in certain scenarios ChatGPT has outperformed new software engineers in specific tasks, though this was more specific on solving easy to medium-level tasks/problems, when the ChatGPT consistently outperformed the new software engineer. On the other hand, the same study has concluded that there is decisive evidence to support the theory that ChatGPT would outperform an experienced developer in terms of solution performance. In summary, the study reveals a dynamic interplay between human and AI performance and the need for a collaborative approach to fine-tone the

AI inputs based on the developer expertise while improving efficiency via automatons through the AI.

2.4 DR-Tools Suite

DR-Tools Suite is a set of lightweight open-source tools that provide resources and information to improve source code quality, supporting the developer in his daily work. DR-Tools Suite was inspired by the medicine metaphor created by Lacerda et al. (2023).

DR-Tools Suite consist of 2 tools: DR-Tools Metric, which is a Command-Line Interface (CLI) tool that collects and shows different source code metrics, and DR-Tools Metric Visualization, which is a tool to provide visual feedback through different graphical formats from the data generated by DR-Tools Metric. DR-Tools Metric is the tool focused on this work and for that reason we will only further develop it.

DR-Tools has a third tool under development, already available in a binary executable file, which is the DR-Tools Health Code. This tool will be explored later and have the data used to a limited extent as it is the tool that will provide this project with data on smells specifically.

Table 2.3 - ChatGPT Cost table per model

Model	Input	Output
gpt-4-0125-preview	$0.01/1k tokens	$0.03/1k tokens
gpt-4-1106-preview	$0.01/1k tokens	$0.03/1k tokens
gpt-3.5-turbo-1106	$0.0010/1k tokens	$0.0020/1k tokens
gpt-3.5-turbo-instruct	$0.0015/1k tokens	$0.0020/1k tokens

Source: OpenAI website

2.4.1 DR-Tools Metric

DR-Tools Metric is designed to provide a well-known set of combined metrics from software metrics research and define a set of heuristics for the combination of metrics based on relationships and thresholds. Thus, the tool calculates metrics and provides insights from the source code, so that this can help developers to learn about software complexity, smells, and refactoring opportunities.

To better understand the source code, a key point is to have the code metrics and its correlations clear, as shown in studies like Radjenovic et al. (2013). For example, the relation between size metrics and object-orientated metrics helps to analyze aspects of code maintainability. Also, according to Bigonha et al. (2019), when a metric is associated with some threshold, it facilitates its use and understanding.

DR-Tools Metric analyzes the source code and provides the results in different formats (line command, CSV, and JSON) to be used in different contexts. It is not required to do any configuration or installation of any complementary software or plug-in to use the tool.

DR-Tools Metric provides 33 metrics contextualized by project summary, namespaces (packages), types (classes), methods, dependencies, and coupling (namespace and type). The following list of metrics by context is adapted from Lacerda et al (2023):

- **Summary (9):**
  - Total of namespaces
  - Total of types
  - Average of types per namespaces (types/namespaces)
  - Total of lines of code (SLOC)
  - Average lines of code per types (SLOC/types)
  - Total of methods
  - Average of methods per types (methods/types)
  - Total of complexity (CYCLO)
  - Average of complexity per types (complexity/types)
- **Namespaces (2):**
  - Number of classes/types (NOC)
  - Number of abstract classes (NAC)
- **Types (9):**
  - Lines of code (SLOC)
  - Number of methods (NOM)
  - Number of public methods (NPM)
  - Class complexity (WMC)
  - Number of dependencies (DEP)
  - Number of internal dependencies (I-DEP)
  - Number of other types that depend on a given type (FAN-IN)
  - Number of other types referenced by a type (FAN-OUT)
  - Number of fields/attributes (NOA)
- **Methods (5):**
  - Lines of code (MLOC)

    - Cyclomatic complexity (CYCLO)
    - Number of invocations (CALLS)
    - Nested block depth (NBD)
    - Number of parameters (PARAM)
- **Namespace Coupling (5):**
    - Afferent coupling (CA)
    - Efferent coupling (CE)
    - Instability (I)
    - Abstractness degree (A)
    - Normalized distance (D)
- **Type Coupling (4):**
    - Number of dependencies (DEP)
    - Number of internal dependencies (I-DEP)
    - Number of other types that depend on a given type (FAN-IN)
    - Number of other types referenced by a type (FAN-OUT)
- **Dependencies (3):**
    - General dependencies (DEP)
    - Internal dependencies (I-DEP)
    - Cyclic dependencies

The tool also provides its users with the flexibility to combine and query contextual information, from general information (summary), information about packages, classes, methods, dependency types, couplings, and reference thresholds of metrics. When presenting the results, the data are sorted according to the context. For example, when presenting information about classes, data is sorted by lines of code, complexity, and number of methods or when presenting about methods, the combination is cyclomatic complexity, nested blocks, lines code, and invocations.

It is also possible to filter contextualized results using the *–top* option. Therefore, it is easier for developers to analyze the source code and filter out the most problematic elements. As presented, it is possible to have a view on summary and packages, more complex classes, and methods (showing the first 5), in a single option.

The tool is currently only developed to analyze Java code, but its architecture is designed to allow simple enhancement to other languages by developing a parser and corresponding visitor to the new language.

DR-Tools Metric is designed to be independent from environments and platforms, facilitating interoperability. Its open architecture allows both the functionalities and the resulting data in known standardized formats to be integrated with other tools, without additional installation or configuration. The tool research is a work in progress and is intended to be expanded with new tools, like refactoring recommendation, to support code review. At this place that this work will be connected.

DR-Tools Metrics is an open-source tool available on GitHub, as well as its code, on this link. For proof of concept of how to perform the integration, we will be exploring the code structure of this tool and its data output. Output which can be controlled by the end user when running the tool as it can select what to extract or to be presented. Table 2.4 presents the commands and what is the output for each of them.

Table 2.4 - DR-Tools Metrics output option for command line

Output Option	**Output**
NO OPTION	Shows tool usage options
-mt	Shows metric thresholds to the metrics as a reference. (Like Healthcare Exams provide reference values)
-a	All metrics below
-s	Summary Metrics
-n	Namespace Metrics
-t	Type Metrics
-m	Method Metrics
-d	Dependency Metrics
-cd	Cyclic Dependency Metrics
-id	Internal Dependency Metrics
-nc	Namespace Coupling Metrics
-tc	Type Coupling Metrics
-ac	All Coupling
-mv	Output file to be used on DR-Tools Metric Visualization
-sn	Statistical Metrics from Namespace Metrics
-st	Statistical Metrics from Type Metrics
-sm	Statistical Metrics from Method Metrics
-san	Namespace Metrics (same as -n) + Statistical Metrics from Namespace Metrics
-sat	Type Metrics (same as -n) + Statistical Type from Namespace Metrics
-sam	Method Metrics (same as -n) + Statistical Metrics from Method Metrics

Source: Elaborated by the author based on DR-Tools Site information and Code

As observed on the Table 2.4, the end user can select which metrics to be calculated and shown with different levels of detail, for example, the user can extract/present only the summary metrics of the project by using the “-s” option, which will show metrics previously presented under as Summary metrics. The same logic is applied for metrics between “-s” and “-tc”, while “-a” usage is for present all metrics from the different options and “-ac” to all coupling metrics.

Options “-s*” are to show statistical metrics from the metrics calculated, so that average, median, standard deviation and quartier division are presented, what is available for namespace metrics (“-sn”), type metrics (“-st”) and method metrics (“-sm”). These statistical metrics can also be provided with its subject metrics, so that for example namespace metrics are presented and then contextualized with its statistics.

The outputs and their difference will be fundamental to be understood to this work as the integration will just be added on top the existent outputs and so will be using the same command options. This will be further explored in upcoming sections 3.1 and 3.2, and in chapter 4.

2.4.2 DR-Tools Code Health

DR-Tools Code Health is a tool from DR-Tools Suite that expands on the DR-Tools Metrics to contextualize the metrics and provide further insights on the code general health. Lacerda et al. (2023) describes the tool on its website as “a tool that allows for a deeper investigation regarding metrics/statistics, design issues (smells), smell co-occurrences, and code elements ranking/prioritization, identifying the most problematic parts at different levels of granularity.”

The tool is an evolution of DR-Tools Metrics, expanding on the contextualized metrics, increasing to 48 metrics, and adding code smells detection (based on metrics) as well as its co-occurrence and ranking code smells for refactoring process. On the ranking, it implements a prioritization model, based on severity, representativeness, impact on quality and degree of intervention. A list of the metrics generated by DR-Tools Code Health is presented below, adapted from Lacerda et al (2023):

- **Summary (15):** total of namespaces, total of types, mean number of types/namespaces, total of lines of code (SLOC), average number of SLOC/types (with median and standard deviation), total of methods, average number of methods/types (com median e standard deviation), total of complexity (CYCLO), and average number of complexity/types (with median and standard deviation);

- **Namespaces (2):** number of classes/types (NOC) and number of abstract classes (NAC);
- **Types (14):** lines of code (SLOC), number of methods (NOM), number of public methods (NPM), class complexity (WMC), number of dependencies (DEP), number of internal dependencies (I-DEP), number of other types that depend on a given type (FAN-IN), number of other types referenced by a type (FAN-OUT), number of fields/attributes (NOA), lack of cohesion in methods (LCOM3), deep in inheritance tree (DIT), number of children (CHILD), number of public attributes/fields (NPA), and number of cyclic dependencies (types) (CDEP);
- **Methods (5):** lines of code (MLOC), cyclomatic complexity (CYCLO), number of invocations (CALLS), nested block depth (NBD), and number of parameters (PARAM);
- **Namespace Coupling (5):** afferent coupling (CA), efferent coupling (CE), instability (I), abstractness degree (A) e normalized distance (D);
- **Type Coupling (4):** number of dependencies (DEP), number of internal dependencies (I-DEP), number of other types that depend on a given type (FAN-IN), and number of other types referenced by a type (FAN-OUT);
- **Dependencies (3):** general dependencies (DEP), internal dependencies (I-DEP), and cyclic dependencies (CDEP);

On top of the metrics above, it also identifies the smells below:

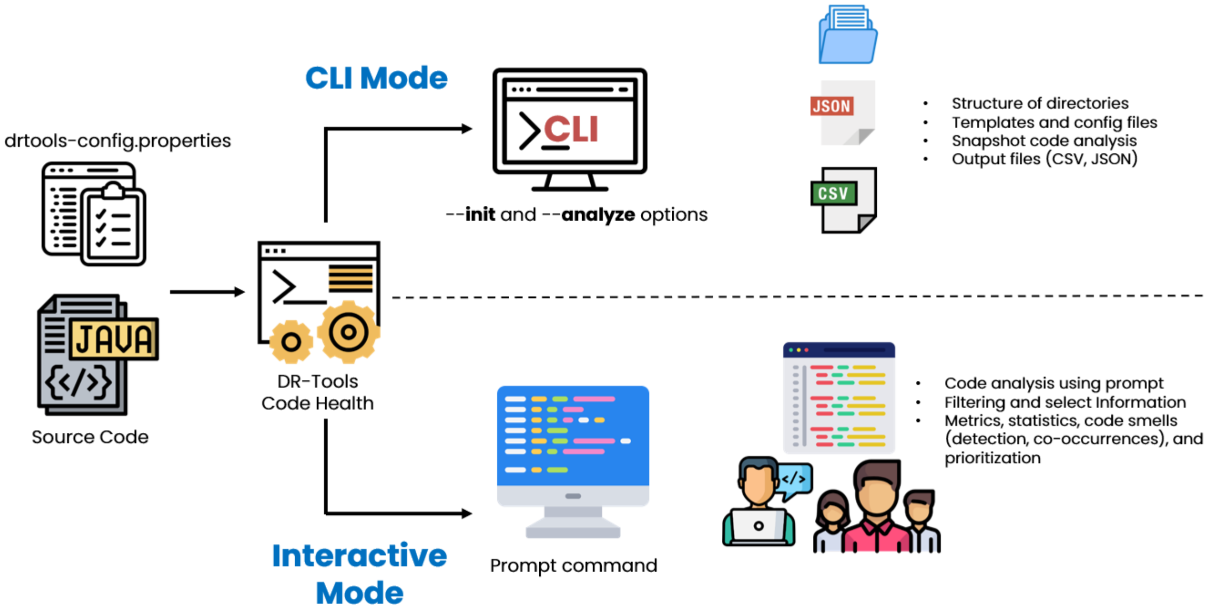
- **Granularity:** Namespace
  - Too Large Package;
  - Cyclic Dependency;
- **Granularity:** Type
  - God Class;
  - Broken Modularization;
  - Cyclically Dependent Modularization;
  - Insufficient Modularization;
  - Deep Hierarchy;
  - Deficient Encapsulation;
  - Hub-like Modularization;

- Multifaceted Abstractions;
- Wide Hierarchy;

- **Granularity:** Method
  - Long Method;
  - Long Parameter List;
  - Complex Method;
  - Bumpy Road;

The tool is also a command-line tool, but it can be used on an interactive usage as well. Via command prompt, its results can be provided in 3 formats, console, CSV or/and JSON. There is the possibility of providing a configuration file to the tool to adjust its parameters to the analyzer, though this is optional. On Figure 2.5, details on the usage for the end users are presented.

Figure 2.5 - DR-Tools Code Health Structure



Source: Lacerda et al. (2023)

2.5 Related Works

This section presents works which have a similar approach or try to solve a similar problem to the one proposed by this project. The related works analyzed is summarized by similarities and differences on Table 2.5 and further explored on individual subsections 2.5.1 and 2.5.2.

Table 2.5 - Summary of related works similarities and differences with this work

Related work	**Similarities**	**Differences**
CodeScene's Refactoring tool powered by LLM (TORNHILL et al., 2024)	Use a code health/metric tool to support LLM refactoring	Completely automate refactoring of specific code smells, approaching from a button-up perspective. This work will not automate refactoring but provide a top-down approach to help prioritize what to refactor.
Prompt Patterns for improving code quality (WHITE et al., 2023)	Develop prompt patterns to be used for software engineering, similarly this work on prompt engineering to define best practices.	Does not approach on automation in a tool or integration with other tool data, focusing on provide prompts to be used as best practices. This work will also focus on developing an integration between a metrics tool and the LLM.

Source: Elaborate by the author

2.5.1 CodeScene's Code Refactoring tool powered by LLM with fact-checking

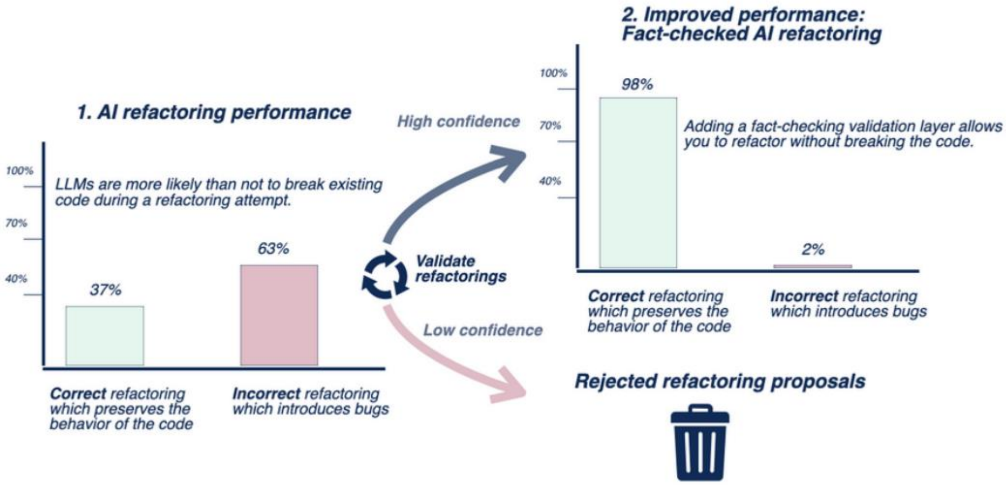
This first related work that will be analyzed is a code refactoring tool developed by CodeScene, which is presented via a whitepaper by Tornhill et al. (2024). It is important to note this is a very recent work, to the point that the tool is not currently available (as of February 2nd), but on beta testing via waitlist on CodeScene website.

On its whitepaper, Tornhill et al. (2024) starts with a benchmarking of state-of-art LLMs and its effectiveness on refactoring, introducing the term "refunctoring", which consists of while refactoring the code also change its function, what might result on introducing bugs. On the benchmarking it is noted that only 37% of the refactoring was resulting on improvements with correctness, while later when applied CodeScene's fact-checking technique it was improved to 98%.

The tool uses CodeScene Code Health's[3] metrics to fact-check if there are improvements on the code health metrics on the new refactored code, because if no improvements are observed, the refactor would not be approved as it failed to pass the test. The idea is that code health metrics are the only code-level metric with a proven business impact in terms of development velocity and post release defects, then if they do not present improvements there is no business reason to risk a production code change, which would expose the code to bugs or expenses testing pre-deployment.

[3] Documentation available on https://codescene.io/docs/guides/technical/code-health.html.

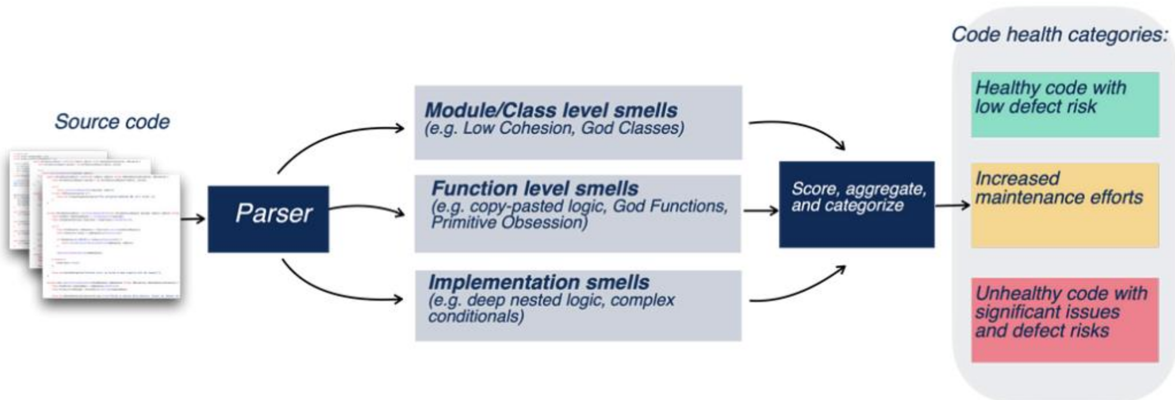
Figure 2.6 - At left, LLMs success rate on refactoring, at the right, improved results by fact-checking technique developed by CodeScene



Source: Tornhill et al. (2024)

Code Health presented by CodeScene has a similar approach to the problems as DR-Tools, as both parse the information and identify code smells, which later are categorized. For more visual representation, it is possible to refer the figure 2.7, which explains the metric and classification process at a high-level.

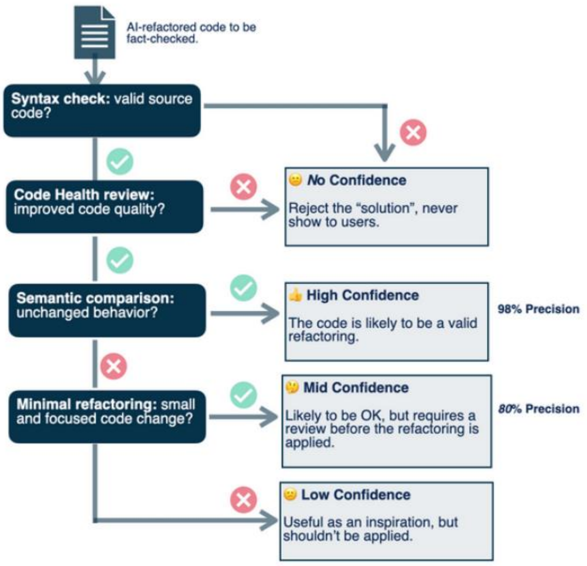
Figure 2.7 - CodeScene's Code Health is a language-neutral, aggregated code quality metric based on a combination of 25 code smells.



Source: Tornhill et al. (2024)

Code Health metrics, illustrated on Figure 2.7, is used to improve the process of refactoring using the data for validation on improvement of the refactoring. The process will start with the simpler question, which is if new code has a valid syntax, then if syntax is valid, health code metrics are compared between original and refactored code, if not improved or syntax is not valid, refactor will be rejected and not presented to end-user. The third part of the process is the semantic equivalence, which is intended to validate if the function is preserved, though this area is an unresolved research problem. The whole process is illustrated in Figure 2.8.

Figure 2.8 - A schematic overview of the layered model for factchecking AI-refactored code



Source: Tornhill et al. (2024)

To handle the semantic equivalent problem, CodeScene has used its data pool of +100,000 real-world refactoring project information to train their in-house model for semantic equivalence detection for a limited set of code smells identified by Code Health metrics. By doing so, they were able to effectively train the model to fact-check refactoring scenarios for the 4 supported smells:

- Complex Conditionals;
- Deep Nested Logic;
- Bumpy Road;
- and Complex Method.

When comparing CodeScene's work with the one presented by this project, the tools approach similar to the perspective to support the refactoring by code health metrics. While their proposal is for a complete automation from a button-up view on code refactoring, focusing on code refactoring specific code smells, the DR-Tools integration with ChatGPT will focus on a top-down approach and to be an assistant to the software developer, by providing guidance on which areas the refactoring should be focused.

In conclusion, CodeScene validates our hypothesis of using code health metrics with LLMs to provide better refactoring and some level of automation. Besides, it provides a very interesting idea of fact-checking, which could be an inspiration for future works, especially if focused on also developing full automation for some specific scenarios.

2.5.2 ChatGPT Prompt Patterns for Improving Code Quality

As already stated by OpenAI itself, a key point of good usage of LLM, like ChatGPT, is to provide a good prompt and context to the request. Therefore, it is an area of study developed and explored in this research.

Though there are several patterns, and they can take various forms, to perform software engineering tasks it is typically better to start with a scoping statement, like “from now on”, “act as a X”, “for the next four prompts” (WHITE et al., 2023).

White et al. (2023) investigated and proposed 13 prompt patterns for different software engineering tasks. They were documented, tested, and analyzed with the format below:

- **A name and classification**: provides a clear name to identify and classifies the pattern based on the type of problem to be solved. The prompt patterns proposed can be viewed on Figure 2.9.
- **The intent and context**: summarizes the problem to be solved and its goal.
- **Motivation**: explains the importance of the problem to be solved.
- **The structure and key ideas**: describes the fundamentals of the pattern and the context that need to be provided to the LLM to achieve the expected resolution.
- **Example implementation:** shows an example of the pattern implemented and discusses it.
- **Consequences:** evaluates the pros and cons of using the pattern and how to adapt the pattern to other scenarios.

This research focuses more specifically on the refactoring piece as the goal is to integrate ChatGPT to DR-Tools Suite to provide the refactoring technique to be followed and study what can be provided by ChatGPT. The research already done on the topic by White et al. (2023) supports the intention of our project, as according to it, tools like ChatGPT have a surprisingly powerful understanding of abstract coding constructs and can deliver innovative approaches to code refactoring.

Figure 2.9 - Classifying Prompt Patterns for Automating Software Engineering Tasks

Requirements Elicitation	Requirements Simulator Specification Disambiguation Change Request Simulation
System Design and Simulation	API Generator API Simulator Few-shot Example Generator Domain-Specific Language (DSL) Creation Architectural Possibilities
Code Quality	Code Clustering Intermediate Abstraction Principled Code Hidden Assumptions
Refactoring	Pseudo-code Refactoring Data-guided Refactoring

Source: White et al. (2023)

2.5.2.1 Pseudo-code Refactoring Pattern

This method consists in basically providing the LLM (ChatGPT) with the pseudo-code structure desired and having the AI refactor to adapt to the specific situation. The refactoring pattern will follow instructions as the Figure 2.10.

Figure 2.10 - Pseudo-code Refactoring Pattern

	Pseudo-code Refactoring Pattern
1.	Refactor the code
2.	So that it matches this pseudo-code
3.	Match the structure of the pseudo-code as closely as possible

Source: White et al. (2023)

One important consideration is that in case the pseudo-code requires an extensive description and precising code, the usage of the LLM could be not advantageous as its benefits will be reduced by the required coding of the pseudo-code to specify the refactoring. This pattern can also lead to substantial refactoring and because of that requires the code to be splitted or have functions removed, which could impact on its public interface be changed and require further refactoring.

It is important to note that this methodology would not be the best fit for this research, as we will be integrating ChatGPT with a data generation tool (DR-Tools) and not a pseudo-code tool.

2.5.2.2 Data-guided Refactoring Pattern

On this pattern, the idea is that providing the data that needs to be changed the LLM will do the refactoring to have the data the closest to the requested. Though it is not exactly to have it matching the smells and source code metrics, but code results and formats, we believe this might be the best pattern to fit our scenario. Figure 2.11 shows how this pattern would be structured.

Figure 2.11 - Data-guided Refactoring Pattern

	Data-guided Refactoring Pattern
1.	Refactor the code
2.	So that its input, output, or stored data format is X
3.	Provide one or more examples of X

Source: White et al. (2023)

The results from White et al. (2023) indicate that this pattern reduces the manual effort to refactor many types of code changes. On many cases the refactoring can be completed automatically or at least be a booster and speed up the refactor causing potentially a cost reduction on the change of data formats, for example.

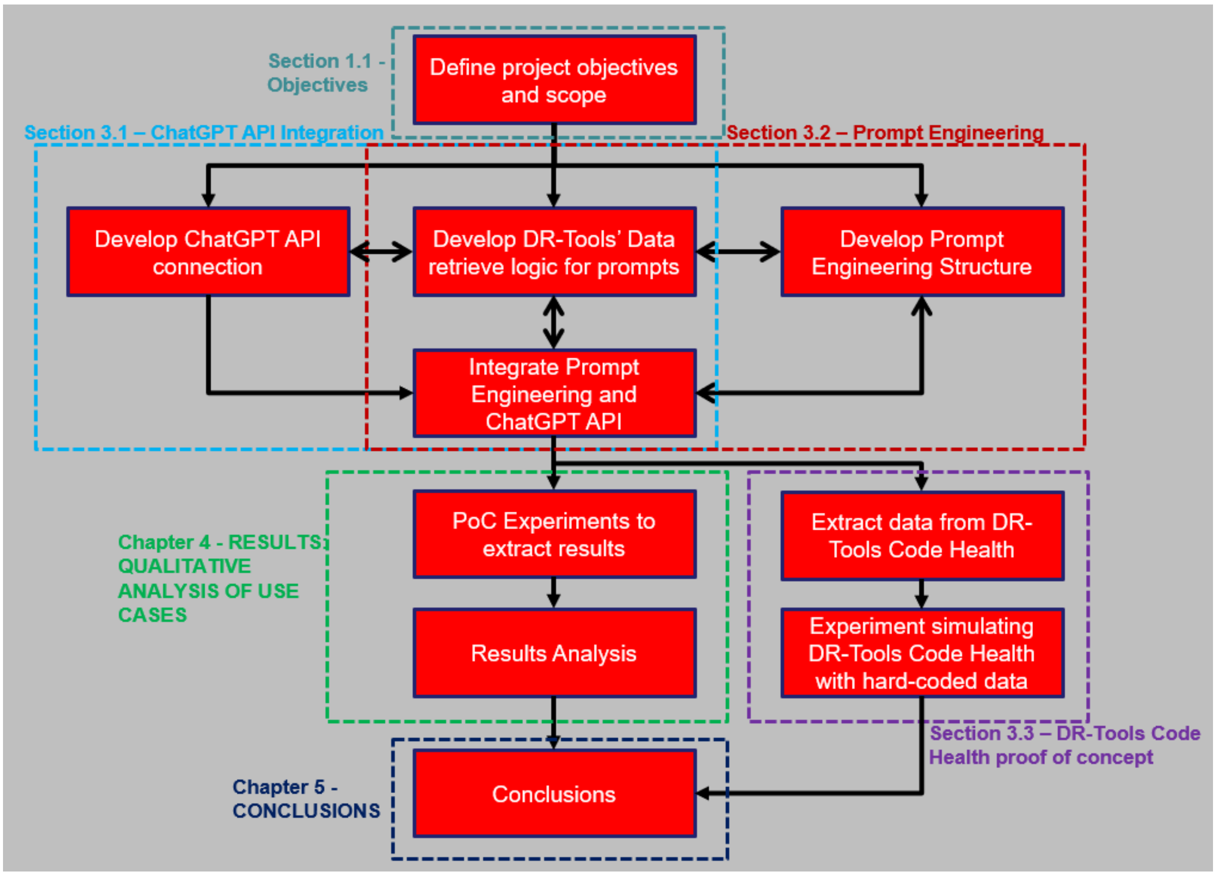
In general, this last study concluded that the depth of the capabilities of LLMs, like ChatGPT, are not fully understood or appreciated, as the tool holds a lot of potential for software engineering automation throughout the software life cycle. The conclusion is that the key to leverage all these capabilities is to codify an effective catalog of prompts and guidance on how to combine these patterns to improve software engineering through automation. At the same time, it is highlighted the significance of human involvement and expertise as currently ChatGPT tends to “hallucinate” confidently, so guidance and scrutiny is required to mitigate these possible issues.

In conclusion, the tools have a lot of potential, but it is required much research and development on prompt pattern engineering to have the best results provided and all potential be fulfilled.

3 INTEGRATING DR-TOOLS AND CHATGPT

This chapter is organized in three parts and presents the development and experiments used to develop the ChatGPT integration into DR-Tools. It is important to highlight that the areas intersect with each other and many of the developments happened in parallel with one influencing into another area. Therefore, Figure 3.1 presents a high-level overview of the activities developed and in which section of this work they are explored.

Figure 3.1- Visual representation of development flow and how it is presented on the text

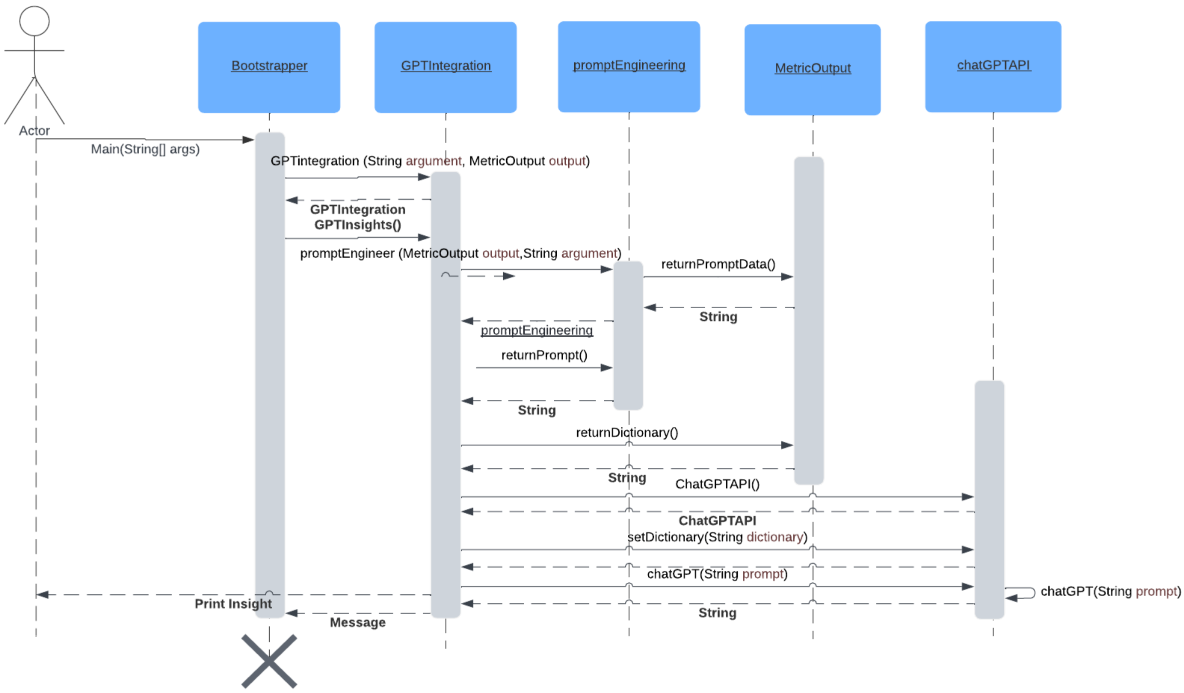


Source: Elaborated by the author

The first part, section 3.1 – ChatGPT API Integration, analyzes the code challenges, and how the retrieving of information and GPT interactions were structured, based on the DR-Tools Metrics code. The second part, section 3.2 Prompt Engineering, develops the prompt engineering for this work, experimenting and determining which are the best prompts to analyze DR-Tools' metrics providing software engineering advice and measuring how effective and insightful they are.

These first two parts, sections 3.1 and 3.2, are part of the development of the proof of concept for the integration between DR-Tools and ChatGPT. Based on the development of these experiments, the proof of concept was created, which has the execution sequence illustrated on Figure 3.2 to generate the insights about the refactoring.

Figure 3.2 - Sequence Diagram for insight generation



Source: Elaborated by the author

The third part simulates the results that applying the same solution developed for DR-Tools Metrics on sections 3.1 and 3.2 would generate, therefore allowing us to evaluate if the same project would be effective for DR-Tools Code Health. This activity is done by using data extracted from DR-Tools Code Health to hard-code it into the solution developed for DR-Tools Metrics (prompt engineering and code for API requests), then evaluates if the solution developed on the sections 3.1 and 3.2 are effective DR-Tools Code Health. The goal is to analyze the result to evaluate if DR-Tools Code Health combined with ChatGPT has potential to be a good solution to be developed in the future.

This approach of work on both tools is because the DR-Tools Code Health tool is an ongoing project that does not have its code public yet, so this research evaluates on manually generated prompts to ChatGPT using the data manually extracted from running the tool via its available binary code data. Therefore, this work aims to validate how to best develop the integration of the tools at code level, for which DR-Tools Metrics (DR-Tools Code Health predecessor) code will be enhanced as a proof of concept. Thus, in future works a defined path and a working scenario will be available.

3.1 ChatGPT API Integration

The first step of this proof of concept is to develop a working integration with OpenAI ChatGPT API, for that a class to do the integration through HTTP Request to the API end point was developed.

3.1.1 ChatGPT API Class creation and first communication test

This subsection presents how the *chatGPTAPI* class was created, and which challenges were faced to establish successful communication. Throughout the section challenges will be presented and how this work has been overcome, then the reader can refer to this to avoid similar issues.

The first step for the integration was to have OpenAI's account and the API Public key to be used on the HTTP Request to open the connection created. As OpenAI provides free access for limited calls and rates and the number of API calls would not be high, the initial plan was to use this free version as limits from number of requests at first were not believed to be exceeded, though this will be revisited multiple times through the experiments.

The Java classes below were used for handling the connection and data retrieval:

- *java.net.HttpURLConnection* and *java.net.URL* to handle the HTTP Connection creation;
- *java.io.InputStreamReader* and *java.io.OutputStreamWriter* to handle the flow of information with ChatGPT API, to both convert the strings to byte to be communicated (*OurtputStreamWriter*) as convert the data in bytes to strings (*InputStreamReader*);
- *java.io.BufferedReader* and *StringBuffer* to handle and manipulate the Input Stream from ChatGPT into the code.

The class was developed to have one public method, *ChatGPT(String prompt)*, which sends to the class the prompt that should be used to the request to ChatGPT, then it would convert and return only the GPT content part of the answer to the class calling the method. This way, when just ChatGPT to print an answer to a prompt/question was wanted, only a call like below would be done:

```
System.out.println(chatGPT("What is a bird? In one line"));
```

The method *chatGPT* will only return the ChatGPT answer to the question in a string format. On the first test, the following error was faced: "*Server returned HTTP response code: 429 for URL: https://api.openai.com/v1/chat/completions*". On Open AI documentation, it was founded that this could be one of the 3 below reasons:

> • You are using a loop or a script that makes frequent or concurrent requests.
> • You are sharing your API key with other users or applications
> • You are using a free plan that has a low-rate limit.
> (OpenAI Article 6891829 Error Code 429 - Rate limit reached for requests, accessed in January 2024)

Upon checking on the OpenAI API usage dashboard, the absence of API requests registered were noticed, so it could not be either first or second option. Once remaining the latest one and after reviewing if any limits by a code bug could be exceeding, it was determined that the issue is that the free version does not have a limit to achieve a single call, what means that in practice ChatGPT API is paid service only.

After upgrading to a paid version by recharging $50 dollars to the OpenAI account used, which allowed the API call to be successful with no changes to the code. The first part was completed as a class that manages the API calls was developed. A call and the code used for the call are exemplified on Figure 3.3:

Figure 3.3 - First chatGPT class successful test

Source: Elaborated by the author

As conclusion, to be able to use the GPT API, a payment for the service is required, so that is the first lesson learned.

3.1.1.1 Support to dictionary and placeholder translation

This subsection presents in advance how the *chatGPTAPI* class supports the concept of having placeholders, a dictionary of equivalence to names used on the data analyzed and how its translation is done. This concept will be further explained and validated in subsection 3.2.2.

The concept is basically that instead of providing data to ChatGPT API with its original name, it will be replaced on the prompt for a unique and simple placeholder name, like *method1* or *type1*. Therefore, a conversion from this placeholder name into the real name will eventually be needed; this responsibility will follow on *chatGPTAPI* class, which per our design will receive the prompt with the placeholder in the prompt and return an answer using the real name.

To achieve this conversion, the class will have a public method *setDictionary (String dictionary)*, which will set a local attribute to the string provided. The value for the dictionary is generated from DR-Tools code with format like “method1 = realName.method, method2 = realName.second.method”.

With the dictionary local attribute, the *chatGPTAPI* class will always check if there is a dictionary setup before returning any answer from ChatGPT API requests. When there is a dictionary, the class will create an instance of itself without dictionary and run a request to as ChatGPT API to act as a “find and replace” tool and from the dictionary replace all placeholders with its real name on the answer previously provided. The prompt is provided to ChatGPT method by on the code concatenating the context, the message to be corrected, the request and the dictionary as presented on Figure 3.4:

Figure 3.4 - Dictionary and translation logic implemented on ChatGPT method to request data

```
if (this.dictionary == null) return extractMessageFromJSONResponse(requesttoGPT(body,connection));
else {
    ChatGPTAPI newGPT = new ChatGPTAPI();
    return newGPT.chatGPT("Act as a find and replace tool. On this anser: "
         + extractMessageFromJSONResponse(requesttoGPT(body,connection))
         + " Replace occurencies of the answer of the expressions before = with the one after it for the following:"
         + this.dictionary);
}
```

Source: Elaborated by the author

This feature will be fundamental to improve results from the data analysis of complex and big dataset, which will be clearly an advantage as exposed on subsection 3.2.2.3. Though when no dictionary is setup, the behavior will remain unchanged and same code can be reused on the translation.

3.1.2 Cost considerations and model definition for the project

In one day of moderated testing, 60 API requests were generated, which consumed 2425 tokens (1238 context tokens and 1187 generated tokens), while on another day 48 API requests were generated, which consumed 4604 tokens (1339 context tokens and 3265 generated tokens). Due to the price structure, it is more efficient to have a broader prompt and limit the output via the prompt instructions, as context tokens are cheaper than generated tokens.

There are other models, but only GPT-4 Turbo and GPT-3.5 Turbo will be focused since they are the flagship models. There are limitations on the models, for instance, GPT-3.5 Turbo supports up to 4096 context tokens (under model GPT-3.5 Turbo) or 16385 (under model GPT-3.5 Turbo-1106), while GPT4 Turbo supports 128000 context tokens. For that reason, if on any point a test with a larger prompt that exceeds 16385 would be needed, only GPT-4 Turbo or GPT-3.5 Turbo-1106 could be used. Training data is also different as GPT-3.5 training data is up to September 2021 and GPT-4 is up to April 2023. However, it is not expected that to be an issue as much of the base of refactoring and code smells proceeds 2021 timeline.

Based on the testing conducted and the price structure, both models can be considered financially viable. Therefore, if GPT-4 proves to have much higher efficiency and precision on its insights, it can be the best approach to this project. Though, an evaluation is needed at the end of the project with full scope defined, including using DR-Tools data as context tokens, what would be the average number of tokens per API request and how many API calls it is expected that a user refactoring will do. The cost impact will be reviewed in the conclusions.

For now, the execution of the tests with both gpt-3.5-turbo / gpt-3.5-turbo-1106 (GPT-3.5 Turbo) and gpt-4-0125-preview (GPT-4 Turbo) is defined, in order to evaluate both and determine when they are different, and which is superior and how much.

3.1.3 Simulate Prompts to test *chatGPTAPI* class

After confirming that the API call was working, the idea was to already test the response from the ChatGPT to one of the outputs of the DR-Tools Metrics tool. So before starting further integration and developing code to retrieve the data from the tool and format it into the ChatGPT prompt, the decision was to execute the tool for one metric and simulate the result to provide the same as part of the prompt.

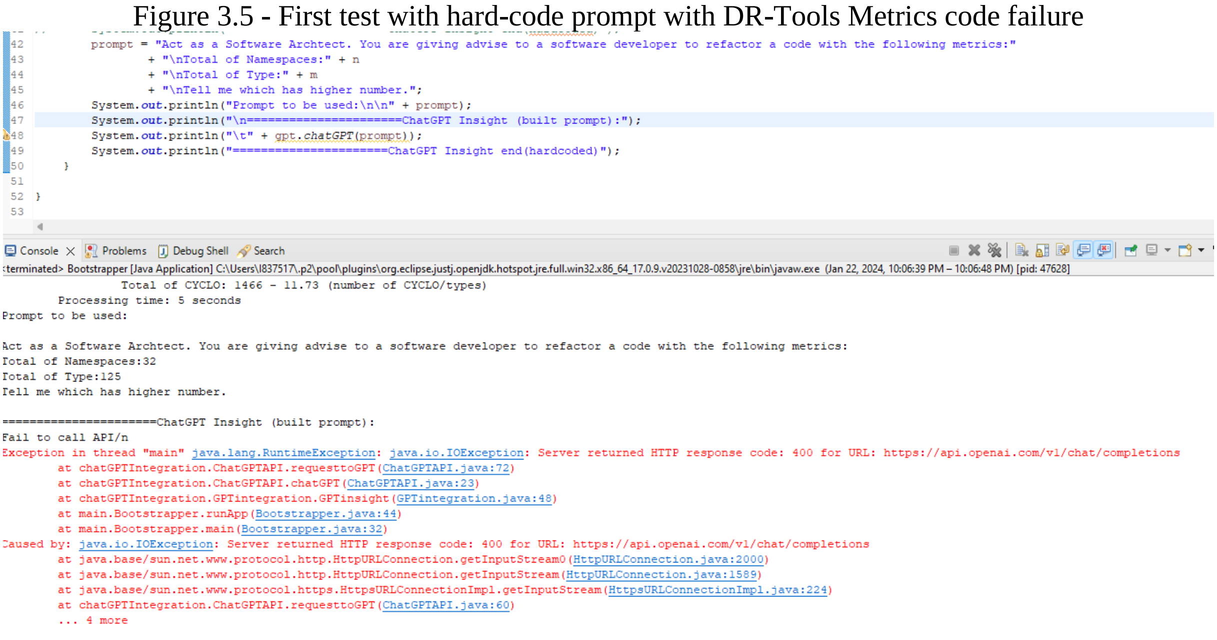
From DR-Tools Metrics project, the *chatGPTAPI* class (discussed on subsection 3.1.1) was added and a new class to handle the integration between DR-Tools Metrics (we called *GPTIntegration* class) was created. This class will act as the link between DR-Tools and ChatGPT calls and prompts. On this first step, a *GPTInsight()* public method to be called from DR-Tools Metrics main code was created and *chatGPTAPI* class was called with a hard-coded prompt to prove that all processes were working in the correct order. Later both data retrieval and prompt definitions will be discussed and explored.

From this first test, DR-Tools Metrics was run on its own project code, and the results below were received:

```
------------------
SUMMARY OF METRICS
------------------
            Total of Namespaces: 32
                 Total of Types: 127 - 3.97 (number of types/namespaces -
median: 3.00 - std dev: 2.90)
                  Total of SLOC: 7957 - 62.65 (number of SLOC/types -
median: 37.00 - std dev: 75.48)
               Total of Methods: 1167 - 9.19 (number of methods/types -
median: 3.00 - std dev: 5.80)
                 Total of CYCLO: 1513 - 11.91 (number of CYCLO/types)
```

To prove if the usage of the data on the prompt would be effective, a hard-coded string was coded with the same numbers and similar structure to be used to do the GPT API request, though when running it failed with “*HTTP response code: 400*” and error illustrated in Figure 3.5, together with the coding of the prompt.

Figure 3.5 - First test with hard-code prompt with DR-Tools Metrics code failure

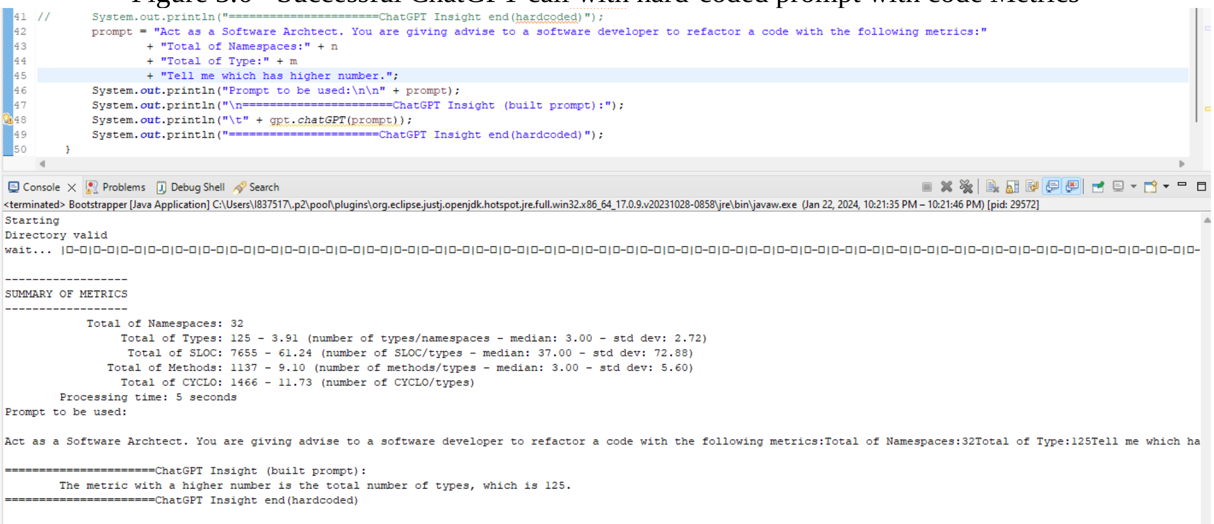


Source: Elaborated by the author

After reviewing previous successful API calls, it was noticed that this test was the first using new line character “\n”, then the deletion of it was tested and the same command without the new lines did not face any issues.

To confirm that a correct answer was being achieved, it was adjusted to have a easily verifiable question. Part of the data was provided and a question was asked to determine which metric have the higher number. The answer was correct and is presented on Figure 3.6 experiment:

Figure 3.6 - Successful ChatGPT call with hard-coded prompt with code Metrics



Source: Elaborated by the author

When confirming that removing the new line character the issue is also removed, how to keep the new line was searched, as this would as well make it easier to have the data for the GPT prompt and also print the same console answers. From that search it was found one quick reference on the [OpenAI Documentation] that references to new line as a possible non-trivial issue on the parsing server-sent events (see extract below from OpenAI documentation).

> Parsing Server-sent events is non-trivial and should be done with caution. Simple strategies like splitting by a new line may result in parsing errors. We recommend using existing client libraries when possible. (API Reference - OpenAI API, 2024)

Based on this experiment and this documentation reference, it was determined that all of the prompts should be provided avoiding any new line or formatting characters (like “\t”) in order to prevent issues with the GPT model. In addition to it, it was determined that all prompt segments would be concatenated in a string without new lines and just characters like simple quotes, parentheses, and other punctuation characters will be used to segregate data, what will be further investigated and further explored with a more complex dataset later.

3.1.4 Retrieving DR-Tools Metrics data in prompt format

This section explores the structure of the DR-Tools Metrics' data at a high-level as well as explains the approach to retrieve and integrate this data into a structured GPT prompt form. Experiments on generation of the data in prompt format will be presented at the end of the chapter, while integration experiments will be presented and have its results analyzed on the subsection 3.1.5, including some failures that establish design and scope decisions.

DR-Tools Metrics have multiple outputs depending on what the user requests. The metrics as well as the commands that can be run to extract them were already explored on the chapter 2.1.4, though on this part of the process some of them are run and the output and later the code that generated them were analyzed, to engineer the prompt design logic. To better understand how DR-Tools Metrics retrieve data, the code and its logic to present data were analyzed.

On figures 3.7, 3.8 and 3.9, the output for the summary of the project's code metrics, type metrics and the statistical data for the type metrics can be respectively found. From these outputs and their code, the creation of the data side of the prompt is going to be started.

Figure 3.7 - DR-Tools Metrics output for Project's Summary

```
Starting
Directory valid
wait... |□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□

------------------
SUMMARY OF METRICS
------------------
            Total of Namespaces: 32
                 Total of Types: 127 - 3.97 (number of types/namespaces - median: 3.00 - std dev: 2.90)
                  Total of SLOC: 7967 - 62.73 (number of SLOC/types - median: 37.00 - std dev: 75.72)
               Total of Methods: 1167 - 9.19 (number of methods/types - median: 3.00 - std dev: 5.81)
                 Total of CYCLO: 1513 - 11.91 (number of CYCLO/types)
        Processing time: 4 seconds
|
```

Source: Elaborated by the author

Figure 3.8 - DR-Tools Metrics output for Metrics per Type

```
Starting
Directory valid
wait... |□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|
```

TYPES	SLOC	NOM	NPM	WMC	DEP	I-DEP	FAN-IN	FAN-OUT	NOA	LCOM3
output.MetricResultJSON	374	42	38	63	25	16	4	22	5	0.95
structures.results.TypeMetricResult	328	45	31	99	12	3	16	9	8	0.94
fixtures.output.JSONDataFixture	325	23	19	38	18	11	1	15	3	0.95
output.MetricResultGPT	293	23	22	38	17	15	1	20	7	0.86
output.MetricResultCSV	282	34	34	45	17	15	4	22	4	0.95
fixtures.output.DataFixture	269	17	17	17	17	11	2	12	10	0.72
output.MetricResultConsole	263	23	22	38	17	15	3	19	4	0.93
javaProject.com.controller.Type	245	35	25	58	7	2	0	9	13	0.85
output.MetricResultFile	203	41	41	56	5	5	2	7	21	0.75
parser.java.visitors.TypeVisitor	189	19	12	42	21	3	1	9	16	0.58
parser.java.visitors.MethodVisitor	188	22	16	39	23	3	1	9	11	0.76
main.Bootstrapper	170	16	1	49	10	9	0	14	7	0.80
fixtures.output.CSVDataFixture	169	15	15	26	8	8	1	11	1	1.00
output.MetricResultFileTest	164	18	16	18	8	1	0	3	22	0.38
structures.metrics.TypeMetric	151	36	36	37	9	2	11	5	15	0.80
structures.statistics.StatisticOfType	144	15	13	16	6	4	5	6	3	0.93
structures.results.TypeMetricResultTest	135	19	17	23	8	3	0	6	2	0.97

Source: Elaborated by the author

Figure 3.9 - DR-Tools Metrics output for Type Summary statistics

```
Starting
Directory valid
wait... |□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|□-□|
```

METRIC	1stQ	3rdQ	Avg	Median	Min	Max	Max-Min	StdDev	U-Fnc	Threshold
SLOC	10.00	71.00	62.73	37.00	3.00	374.00	371.00	75.72	162.50	500.00
NOM	1.00	12.00	9.17	6.00	0.00	45.00	45.00	9.38	28.50	14.00
NPM	1.00	12.00	8.29	5.00	0.00	41.00	41.00	8.55	28.50	40.00
WMC	1.00	14.00	11.95	7.00	1.00	99.00	98.00	15.02	33.50	100.00
DEP	1.00	6.00	4.43	2.00	0.00	25.00	25.00	5.47	13.50	20.00
I-DEP	0.00	3.00	2.42	1.00	0.00	19.00	19.00	3.84	7.50	15.00
FAN-IN	0.00	3.00	2.92	2.00	0.00	30.00	30.00	4.90	7.50	10.00
FAN-OUT	1.00	5.00	3.78	2.00	0.00	22.00	22.00	4.79	11.00	15.00
NOA	0.00	3.00	2.35	1.00	0.00	22.00	22.00	3.96	7.50	8.00
LCOM3	0.00	0.93	0.42	0.00	0.00	1.00	1.00	0.43	2.32	1.00

```
Legend:
        1stQ=First Quartile | 3rdQ=Third Quartile | Avg=Average | Median=Median | Min=Min value | Max=Max value
        Max-Min=Amplitude | StdDev=Standard Deviation | U-Fnc=Upper Fence | Threshold=Metric Threshold

        Processing time: 5 seconds
```

Source: Elaborated by the author

Based on this research and the literature review[4], the best approach to provide data is to delimit it to be clear how they are grouped. To achieve this, it was determined that the data should be organized on a String (based on tests reported on 3.1.3) with no new lines, so it would be needed to have this same data from the output, though in a different format to make it easier and more precise to the GPT model processing it.

[4] OpenAI documentation advice to use delimiters:
(https://platform.openai.com/docs/guides/prompt-engineering/tactic-use-delimiters-to-clearly-indicate-distinct-parts-of-the-input)

At this phase, the code was developed to generate the data above in a string delimited by some token to indicate where it starts and ends each data. The goal was to have a String returned as exemplified below:

- <Token> Metric x <Token> Type: <data> <Token>…

Review and understand the DR-Tools' logic, especially its data retrieval architecture, was necessary to develop the code for the prompt's data string (described above) generation. Though it was not necessary to know how the data is calculated and generated as this work is not validating or analyzing the data generated but interested in its consumption. This analysis was specifically focused on the process to retrieve data and to print it on the console when executing the tool with the "–console" output (to have DR-Tools printing the data in the console) as it was the closer to the string generation.

The DR-Tools' architecture is well organized and modularized in such a way that the implementation was relatively easy, to the point that no existing code needed to be much rewritten. DR-Tools is organized into classes where after its parsing and calculation, the metrics data is stored or is calculated upon requests to these same classes. The classes are listed with a short description:

- *NamespaceMetricResult*: Hold data and have methods to return values of the metrics from the different Namespaces on the analyzed project.
- *TypeMetricResult*: Hold data and have method to return values of Metrics from the different Types on the analyzed project.
- *MethodMetricResult*: Hold data and have method to return values of Metrics from the different Methods on the analyzed project.
- *StatisticalAnalysis*: Hold data and have method to return values of the statistical data (average, median, standard deviation) from analyzed project metric. Data is presented on Figure 3.7 and 3.9. (average, median, standard deviation, etc.)

Then the second part of the DR-Tools Metrics retrieving data logic is done by a public interface called *MetricOutput*, which has all methods to retrieve the data from the above-mentioned classes. This interface is then implemented by classes specifically to the data output selected by the end user when running the tool. There are classes which will implement the methods and have the data output created and provided to the user according to its request. The possible outputs and classes are as following:

Table 3.1 - DR-Tools Metrics' Output and classes

Output	**Class**	**Argument**
Console	*MetricResultConsole*	--console
CSV file	*MetricResultCSV*	--csv
JSON file	*MetricResultJSON*	--json
File for DR-Tools Visualization tool	*MetricResultFile*	--save

Source: Elaborated by the author

Due to this modularity and having already a logic in place for different outputs, a new class implementing the *MetricOutput* interface was created, which was called *MetricResultsGPT*. It was partially replicated following the logic from *MetricResultConsole*, modifying it to be able to return the metric's data on a prompt friendly string to be used on the GPT request.

Through a similar logic used on other *MetricResult* classes and a series of concatenations into a prompt data variable, the string is returned by a public method to be used later. For example, the same metrics shown on Figure 3.7 (with project summary metrics) can be seen on Figure 3.10 printed on prompt format:

Figure 3.10 - DR-Tools GPT prompt data for Project's summary

```
Starting
Directory valid
------------------
SUMMARY OF METRICS
------------------
            Total of Namespaces: 32
                 Total of Types: 127 - 3.97 (number of types/namespaces - median: 3.00 - std dev: 2.90)
                  Total of SLOC: 7984 - 62.87 (number of SLOC/types - median: 37.00 - std dev: 76.14)
               Total of Methods: 1167 - 9.19 (number of methods/types - median: 3.00 - std dev: 5.85)
                 Total of CYCLO: 1513 - 11.91 (number of CYCLO/types)

Data prepared for the prompt:  <BEGIN DATA TOKEN>SUMMARY OF METRICS:  <BEGIN DATA TOKEN>Total of Namespaces: 32<END DATA TOKEN>  <BEGIN DATA TOKEN>Total of Types: 127<END
DATA TOKEN>  <BEGIN DATA TOKEN>Number of types/namespaces: 3.97 (median: 3.00 - std dev: 2.90')<END DATA TOKEN>  <BEGIN DATA TOKEN>'Total of SLOC: 7984'<END DATA TOKEN>
<BEGIN DATA TOKEN>'Number of SLOC/types: 62.87 (median: 37.00 - std dev: 76.14)')<END DATA TOKEN>  <BEGIN DATA TOKEN>'Total of Methods: 1167<END DATA TOKEN>  <BEGIN DATA
TOKEN>'number of methods/types: 9.19 (median: 3.00 - std dev: 5.85)'<END DATA TOKEN>  <BEGIN DATA TOKEN>'Total of CYCLO: 1513'<END DATA TOKEN>  <BEGIN DATA TOKEN>'Number of
CYCLO/types: 11.91'<END DATA TOKEN> <END DATA TOKEN>
```

Source: Elaborated by the author

For having it easier to execute more testing on formatting this prompt data, two constant variables were used to hold a data beginning and ending token values, so that it can be easily changed how the data are being divided for prompt testing, which will be evaluated on upcoming section 3.2.2.

3.1.5 Validating DR-Tools data on ChatGPT API requests

This section discusses at a high-level how ChatGPT can be integrated with DR-Tools data metrics using the previously discussed *chatGPTAPI* and *metricResultsGPT* classes, as well as the challenges with the implementation.

On DR-Tools Metrics, the information output to the end user is controlled by a *ProjectInfo* class, for which the main program code creates an instance with the *MetricResults* as one of its parameters. This class will be managing which of the methods of the *MetricResults* is called to have the metrics provided.

The same logic was replicated on a new class that is called *GPTIntegration*, which will be used to manage the data retrieval (explored on subsection 3.1.4), the prompt generation (to be explored on section 3.2), ChatGPT API (explored on subsection 3.1.1) requests and GPT insight output to be provided. The class will be the main hub link between DR-Tools code and the newly developed code, except for the new MetricResults implementation.

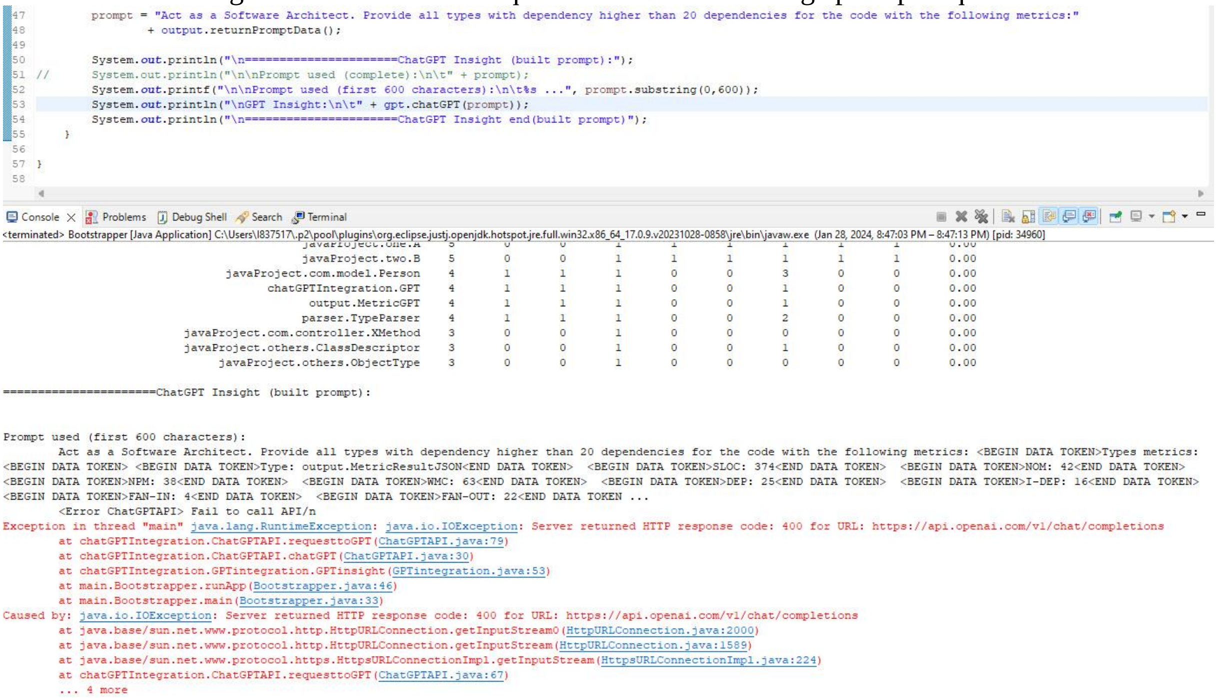
To validate if the GPT requests and prompts are generating valid answers, an easy to validate experiment was conducted, like request “which is the method with more lines” or “which types have dependencies higher than a specific number”. Prompts for refactoring insights will be discussed in section 3.2. Different metrics were tested to validate if data retrieval into prompt was effective. Both tests were done with both GPT-4 Turbo and GPT-3.5 Turbo.

When the test was run on GPT-3.5 Turbo (model gpt-3.5-turbo), GPT API replied with “*Server returned HTTP response code 400 error*”, as presented at Figure 3.11. Through a further analysis of the prompt data (extract available on Appendix A), it was concluded that the error was due to the context tokens (for reference see section 2.3.1) on the API request, which exceeded the number of tokens supported for the model. The prompt had 9172 words (context token will be a little higher) and the GPT-3.5 Turbo used (model gpt-3.5-turbo), supports up to 4096 context tokens (as referenced on Table 2.2). Therefore, it would be needed to reduce the number of tokens on the prompt or choose another model.

An option to continue using and testing GPT-3.5 Turbo would be to use the latest gpt-3.5-turbo-1106 model, which supports up to 16385 context tokens, though the same issue was faced. As explained in section 2.3.1.1, the number of tokens will play a major role in the models supported and the costs, then on decisions. The challenge is that a clear response from API on number of used tokens exceeded is missing, as in the experiment on Figure 3.11.

It will be further explored in the prompt engineering chapter as if prompts exceed 16385 tokens in the GPT-3.5 will be supported and if exceeds 128000 GPT-4 Turbo will also not be supported. Some strategy or scope limitation for this integration usage to not exceed the token limits need to be developed, as prompts will increase according to the analyzed project size and complexity.

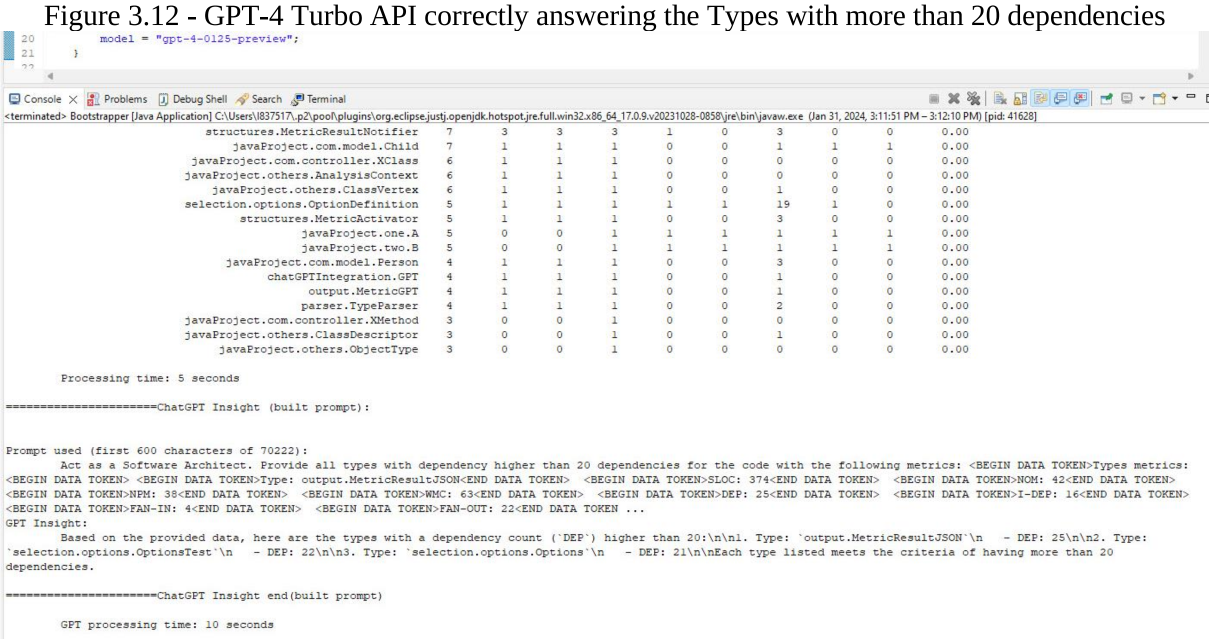
Figure 3.11 - GPT API request error 400 due to large prompt request



Source: Elaborated by the author

The same request was done for GPT-4 Turbo, which was successful and provided an accurate answer, presented on figure 3.12, when asked to list the types with more than 20 dependencies based on the metrics provided on the prompt (generated from DR-Tools). ChatGPT also provided, without request, the number of dependencies for each type, what can be helpful and will be further investigated in section 3.2 about prompt engineering.

Figure 3.12 - GPT-4 Turbo API correctly answering the Types with more than 20 dependencies

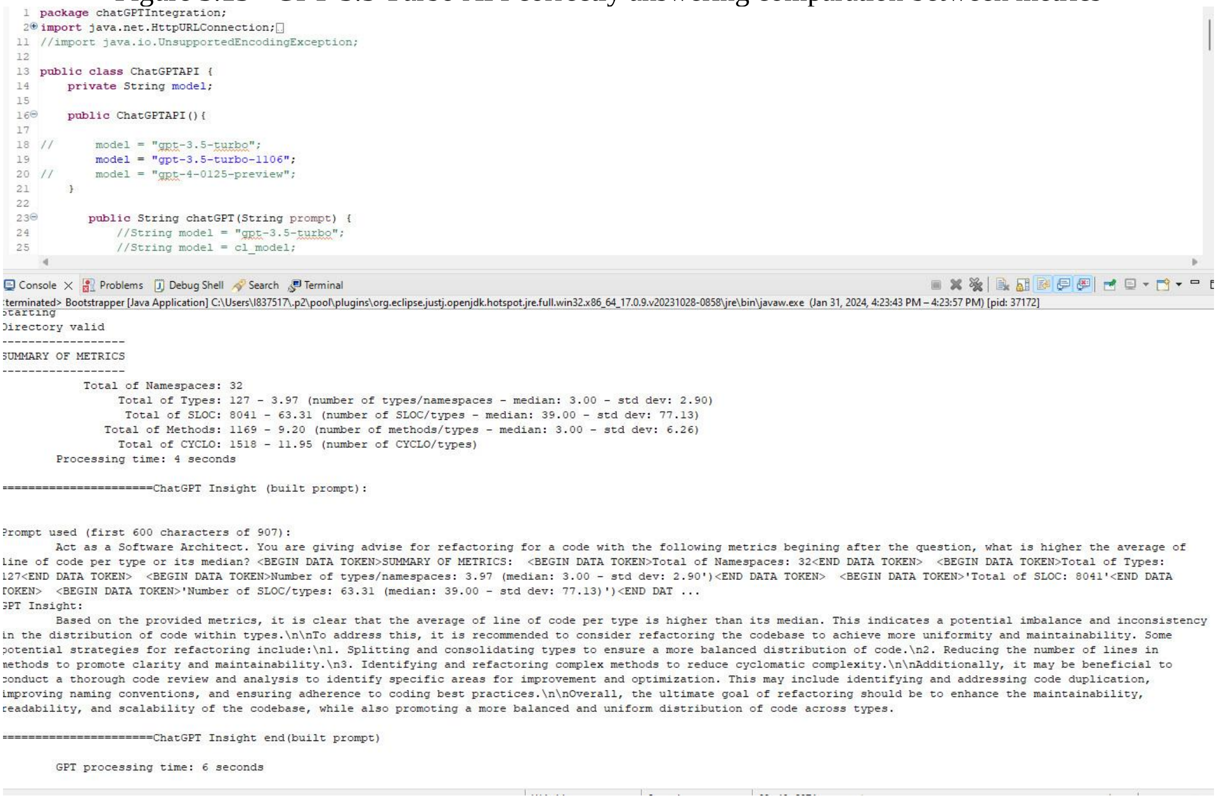


Source: Elaborated by the author

An important note is that GTP-4 Turbo takes longer time to process the same prompt, so this will be analyzed later as a disadvantage. Section 3.2 further explores comparing the models and the best prompts, as it is important to demonstrate that the ChatGPT can understand the metrics that the integration with DR-Tools has provided.

To confirm that GPT-3.5 also understood the data format and to be able to compare if there are differences between their answers, similar experiment was performed using summary (-s command) and requested which would be greater: average line of code per type or its median. With this experiment, a smaller dataset would be obtained and could help avoid any token limitations.

Figure 3.13 - GPT-3.5 Turbo API correctly answering comparation between metrics



Source: Elaborated by the author

From this experiment, it was observed that both models were able to provide an accurate answer, though the answer's structure was very different. While GPT-3.5 Turbo provided a more direct answer to the point (as observed on Figure 3.13), GPT-4 Turbo provided it in the middle of a long text with information that would not necessarily be relevant as observed on Figure 4.14.

Figure 3.14 - GPT-4 Turbo API correctly answering comparation between metrics

Source: Elaborated by the author

The next sections will switch from one model to another to evaluate the prompts and each model’s response, to later analyze advantages and cost benefits. To perform tests on both, *chatGPTAPI* class code can be easily changed via a variable that has being currently used hardcoded on the code, but on a final product it could easily be improved to be via a config.file, though for this proof of concept to have it at code level was sufficient.

3.2 Prompt Engineering

The next step on the integration is to work on prompt engineering to determine the best structure and wording to do the requests to the ChatGPT.

3.2.1 Importance of Prompt Engineering

This section presents the importance of good prompt engineering and develops on analysis of results of different structures of prompt to receive the best, more accurate and more time effective answers. It presents a series of experiments to empirically demonstrate some fundaments described in the previous literature review (especially on 2.3.3 section).

To abstract the more complex and sometimes not so intuitive concepts of software engineering, the first section experiments simply revolve on how ChatGPT would define a simple concept of a “bird” and explore how the prompt design influence its output for the same simple animal. The goal is that from this section, the motivations for the prompt engineering strategies for the project are justified. Then, in the next section, the use of strategies with the DR-Tools Metrics data is explored.

Throughout this chapter, gpt-3.5-turbo-1106 (referred from now on simply as latest GPT 3.5 Turbo) and gpt-4-0125-preview (referred from now one simply as GPT-4 Turbo) will both be experimented on. The older gpt-3.5-turbo will no longer be considered a viable possibility as its 4096 tokens were already proven to be not sufficient from experiments on section 3.1.5.

3.2.1.1 Bird Definition Experiment

This section explores a simple and to some degree fun exercise designed for this project, which consists of having ChatGPT answering the simple question of “What is a bird?”. Although initially appearing to be a silly experiment, this is a powerful way to demonstrate the value of a good prompt in order to have the correct answer to the perspective that API user wants without any complex topic where the reader might not have knowledge to judge the ChatGPT answer.

The section is structured of a series of 12 experiments divided into 4 groups, which each will have one open question with no limitation on answer, another that limits on 1 line[5] answer and a last one limiting on one-word answer. Each group also has different structures:

1. Just the open question and the answer size limitation, without any context.
2. Opening the prompt with the question, then providing the context and finishing with answer size limitation.
3. Starting with context statement, then questioning and finishing with answer size limitation.

[5] Based on our experiments, ChatGPT seems to understand prompts with phrases, clauses, periods, or sentence limitations as an answer with one or more verbs, but only one period being simple period or compost period. Example: “The bird is my tasty treat, and I would pounce on it with my sharp claws and devour it hungrily.” is considered one clause. This could inform a grammatical discussion on ChatGPT understanding; however, this will not be relevant for this work.

4. Starting with context statement (different from above to prove change in answer), then questioning and finishing with answer size.

All the experiments above were coded directly on the *chatGPTAPI* class, so that each group has all its 3 answer limitation prompts run together printing on the same console its prompt, its answer, the time that ChatGPT API took to provide the answer and the number of tokens used and generated for each scenario. The data is also considered throughout the analysis. On this first subsection, only results from latest GPT-3.5 Turbo are explored to subsequent subsection explore GPT-4 Turbo and compare both models.

Figure 3.15 - Bird definition experiment on latest GPT-3.5 Turbo without context definition

```
------Prompt Experiment 1:------
Prompt: What is a bird?

        GPT Answer: A bird is a warm-blooded vertebrate with feathers, a beak, and wings. They lay eggs and have a high metabolic rate, enabling them to fly. Birds range in
size from the tiny hummingbird to the ostrich, the largest living bird. They are found in all kinds of habitats around the world and are known for their ability to vocalize
and produce various sounds.
        GPT processing time: 2 seconds  GPT Context Tokens: 12  GPT Generated Tokens: ": 78

------Prompt Experiment 2:------
Prompt: What is a bird? In one line

        GPT Answer: A bird is a feathered, winged, and warm-blooded vertebrate that lays eggs and is capable of flight.
        GPT processing time: 991 milliseconds   GPT Context Tokens: 15  GPT Generated Tokens: ": 25


------Prompt Experiment 3:------
Prompt: What is a bird? In one word

        GPT Answer: Avian
        GPT processing time: 857 milliseconds   GPT Context Tokens: 15  GPT Generated Tokens: ": 2
```

Source: Elaborated by the author

From the first experiments, it was observed that the output as well as the time that ChatGPT takes is highly influenced by the prompt and the instruction on the prompt to the answer's size. The experiment shows that, by limiting the size of the answer, there are benefits on ChatGPT performance and on costs. As previously referred to in section 3.1.2, the number of tokens influences the cost and generated tokens are 2 to 3 times more expensive.

Figure 3.16 - Bird definition experiment on latest GPT-3.5 Turbo with question first then context definition

```
------Prompt Experiment 4:------
Prompt: What is a bird? From a cat perspective

        GPT Answer: A bird is a small, fluttery creature that I love to watch through the window, but I'm not allowed to catch. They make interesting sounds and are fun to
chase, but they always seem to fly away just out of reach. I enjoy watching them from the safety of my cozy spot indoors.
        GPT processing time: 1 seconds  GPT Context Tokens: 16  GPT Generated Tokens: ": 61


------Prompt Experiment 5:------
Prompt: What is a bird? From a cat perspective. In one line.

        GPT Answer: A bird is a tantalizing, fluttering creature that I long to chase and catch.
        GPT processing time: 1 seconds  GPT Context Tokens: 21  GPT Generated Tokens: ": 18


------Prompt Experiment 6:------
Prompt: What is a bird? From a cat perspective. In one word.

        GPT Answer: Prey.
        GPT processing time: 550 milliseconds   GPT Context Tokens: 21  GPT Generated Tokens: ": 3
```

Source: Elaborated by the author

On this second experiment set, the context setting is experimented to extract an answer from one specific perspective, in this case, to ChatGPT describe what is a bird from a cat perspective. The goal is to have it described as something that the cat would eat or hunt as the normal concept that cats hunt birds, though on the experiment it was not the description that was followed.

For instance, when no size limitation was provided, the bird was more described by physical appearance, then added the information that the cat would occasionally chase it but having the idea that it could catch if "lucky" and then having a more cat like behavior to describe that, if catching, it would bring the bird to its human. It is possible to notice that at first there was not much sign of cat behavior, though in the end there was more.

When there is answer size limitation, an answer closer to the initial goal is provided; when on one line, it gives the idea of something that flies and is out of reach (as the bird flies and the cat does not) and presents the idea of catching it. This is still far from the initial goal, though due to the smaller space, it has compiled all the context (the bird and the cat perspective) into the first statement.

Finally, when ChatGPT is limited to having only one word, it needs to consider all the prompt contexts and summarizes it in a word, only then it is presented with the initial expected answer. This experiment shows that limiting ChatGPT reduces its "hallucination", which although includes correct facts, but not necessarily the facts requested by the prompt. This experiment conclusion is an important concept to be used through the prompt engineering on this work because providing easy to read information and avoiding long answers which miss the main objective is important, otherwise it might be distracting for the end user.

Figure 3.17 - Bird definition experiment on latest GPT-3.5 Turbo with context definition (cat) first then question

```
------Prompt Experiment 7:------
Prompt: Act as a hungry cat. What is a bird for you?

        GPT Answer: As a hungry cat, a bird is a tantalizing and delectable prey. It is a symbol of my natural hunting instincts and the fulfillment of my predatory desires.
The sight of a bird triggers my predatory instincts, and I am ready to pounce and catch it for a satisfying meal. It represents the thrill of the chase and the taste of
success as I fulfill my feline instincts.
        GPT processing time: 2 seconds  GPT Context Tokens: 20  GPT Generated Tokens: ": 79


------Prompt Experiment 8:------
Prompt: Act as a hungry cat. What is a bird for you? In one line

        GPT Answer: A bird is a tantalizing and elusive prey, a delicious feast that must be caught and devoured.
        GPT processing time: 1 seconds  GPT Context Tokens: 23  GPT Generated Tokens: ": 21


------Prompt Experiment 9:------
Prompt: Act as a hungry cat. What is a bird for you? In one word

        GPT Answer: Prey
        GPT processing time: 998 milliseconds   GPT Context Tokens: 23  GPT Generated Tokens: ": 2
```

Source: Elaborated by the author

The third experiment is inspired by White et al. (2023), which highlights the importance of setting a persona to ChatGPT and requesting that it act as such persona. Therefore, the experiment prompt starts with the specification to ChatGPT acting as a hungry cat, then it is observed that from all 3 answer sizes there is a hunter behavior when ChatGPT describes the bird as a prey (even if indirectly).

At this experiment, the same outcome was presented when comparing the different answer size limitations, in which no limitation generated a long answer that does not directly describe as the goal intended. Though when the model is pushed to one line answer, it provides the intended answer, and the one word describes it perfectly.

Figure 3.18 - Bird definition experiment on latest GPT-3.5 Turbo with context definition (human) first then question

```
------Prompt Experiment 10:------
Prompt: Act as a Human. What is a bird for you?

        GPT Answer: To me, a bird is a fascinating and beautiful creature that symbolizes freedom and grace. Birds are amazing in their ability to soar through the sky with
ease and agility, and their varied colors and songs add vibrancy to the natural world. They are also a reminder of the interconnectedness of all living beings and the
importance of preserving our environment. Overall, birds are a source of wonder and inspiration for me.
        GPT processing time: 3 seconds  GPT Context Tokens: 19  GPT Generated Tokens: ": 82


------Prompt Experiment 11:------
Prompt: You are a Human. What is a bird? In one line

        GPT Answer: A bird is a warm-blooded vertebrate with feathers, beaks, and the ability to fly.
        GPT processing time: 1 seconds  GPT Context Tokens: 20  GPT Generated Tokens: ": 21


------Prompt Experiment 12:------
Prompt: You are a Human. What is a bird? In one word

        GPT Answer: Avian
        GPT processing time: 2 seconds  GPT Context Tokens: 20  GPT Generated Tokens: ": 2
```

Source: Elaborated by the author

To prove the conclusions above, the last experiment had the same structure, though requiring “to act as” a human. Throughout all the requests, it can be observed that the hunter/prey relationship was not presented as the human view of a bird is not as a prey or a meal, thus proving that the previous answers were due to the contextualization provided.

Throughout this experiment, the token usage was also presented (information extracted from API response) which provided insight on how the LLM handles the tokens. There will be no deep investigation on how the model is dividing the strings into tokens, as referred to in previous section 2.3.1.1.

One important note is that as context tokens are cheaper than generated tokens, therefore the investment on better prompts even if longer can be financially beneficial, as from examples above, the more precise answers were ones generated due to answer size limitation, which reduces generated tokens making it a cheaper answer.

3.2.1.2 ChatGPT inconsistent answers

From the previous section experiment, there was a second important finding to be presented and explored at some level. Throughout this work’s experiments, it was observed that the answers to the same questions were not always the same.

Figure 3.19 - Bird definition experiment on latest GPT-3.5 Turbo with context definition (cat) first then question (generated on February 1st)

```
------Prompt Experiment 7:------
Prompt: Act as a hungry cat. What is a bird for you?

        GPT Answer: A bird is a tantalizing, fluttering creature that I can't resist chasing and pouncing on. It's a potential meal that satisfies my instinctual urge to hunt
and capture prey. The sight of a bird is enough to make my mouth water and my eyes narrow in anticipation of the chase. I'll stalk and creep up on a bird with all the stealth
and precision of a true predator, and once I have it in my sights, there's no stopping me from leaping and swiping at it with my sharp claws. A bird is the ultimate
temptation for a hungry cat like me, and catching one is a triumph that satisfies my hunger and reminds me of my wild, feline nature.
        GPT processing time: 7 seconds


------Prompt Experiment 8:------
Prompt: Act as a hungry cat. What is a bird for you? In one line

        GPT Answer: A bird is a tantalizing prey, a delicious meal waiting to be caught and devoured with pleasure.
        GPT processing time: 3 seconds


------Prompt Experiment 9:------
Prompt: Act as a hungry cat. What is a bird for you? In one word

        GPT Answer: Prey
        GPT processing time: 5 seconds
```

Source: Elaborated by the author

No clear pattern is observed, as sometimes the answer changes for requests done almost simultaneously (or very close to). For example, the experiment on figure 3.19 was rerun with intervals of less than minutes from figure 3.17. On figures 3.17, 3.19 and 3.20, the same experiment was run providing different answers on each iteration, demonstrating the non-determinism property of LLMs like ChatGPT, which is also observed and explored on other works like Ouyang et al. (2023). Such property will require that, throughout our development, extensive tests are performed to assure a good level of consistency on the solutions.

Figure 3.20 - Bird definition experiment on latest GPT-3.5 Turbo with context definition (cat) first then question (generated on February 2nd)

```
------Prompt Experiment 7:------
Prompt: Act as a hungry cat. What is a bird for you?

        GPT Answer: For a hungry cat, a bird is a tantalizing and delicious prey. It's an opportunity to pounce and satisfy my craving for a tasty meal. The flutter of its
wings and the chirping sound only make it more enticing and irresistible. It's like a gift from the sky, ready to be caught and devoured. I can almost taste the succulent
meat and feel the satisfaction of a full belly just thinking about it. The bird is my next meal, and I am ready to pounce.
        GPT processing time: 2 seconds  GPT Context Tokens: 20  GPT Generated Tokens: ": 102


------Prompt Experiment 8:------
Prompt: Act as a hungry cat. What is a bird for you? In one line

        GPT Answer: A bird is a tempting and elusive treat that I must catch and devour.
        GPT processing time: 628 milliseconds   GPT Context Tokens: 23  GPT Generated Tokens: ": 15


------Prompt Experiment 9:------
Prompt: Act as a hungry cat. What is a bird for you? In one word

        GPT Answer: Delicious!
        GPT processing time: 782 milliseconds   GPT Context Tokens: 23  GPT Generated Tokens: ": 3
```

Source: Elaborated by the author

Note that comparing with the previous subsection and figure 3.17, not only the content is different, but also the processing time was very different. The time difference could be justified by the fact that the answer is provided through the internet and on a cloud computing environment, therefore there are numerous possible reasons that could even be combined, like lower internet speed, higher internet latency, higher load on the cloud service, resources update on the cloud, among others. This work does not target the performance of ChatGPT, so the analysis of it is limited, though it is an important factor to be considered when a future tool to end users derives from this work.

To further explore this inconsistency aspect, the following experiment was conducted: rerunning the experiments 8 and 9 and creating similar experiments 13 and 14, including “From now on,” statement at the beginning as indicated by White et al. (2023), which was not fully implemented on 8 and 9 previously. On this test, experiments 8 and 13 run multiple times interchanged at the same run, same with 9 and 14.

Figure 3.21 - Bird definition experiment 8 and 13 run multiple times providing different answers

```
------Prompt Experiment 8:------
Prompt: Act as a hungry cat. What is a bird for you? In one line

        GPT Answer: A bird is a tantalizing and delectable snack, just waiting to be pounced on and devoured.
        GPT processing time: 1 seconds  GPT Context Tokens: 23  GPT Generated Tokens: ": 23


------Prompt Experiment 13:------
Prompt: From now on, act as a hungry cat. What is a bird for you? In one line

        GPT Answer: A bird is a tasty, fluttering treat just waiting to be caught and devoured.
        GPT processing time: 884 milliseconds   GPT Context Tokens: 27  GPT Generated Tokens: ": 18


------Prompt Experiment 8:------
Prompt: Act as a hungry cat. What is a bird for you? In one line

        GPT Answer: A bird is a tantalizing prey that I must catch and eat to satisfy my hunger.
        GPT processing time: 841 milliseconds   GPT Context Tokens: 23  GPT Generated Tokens: ": 18


------Prompt Experiment 8:------
Prompt: Act as a hungry cat. What is a bird for you? In one line

        GPT Answer: A bird is a tantalizing feast just within reach of my sharp claws and eager jaws.
        GPT processing time: 705 milliseconds   GPT Context Tokens: 23  GPT Generated Tokens: ": 18


------Prompt Experiment 13:------
Prompt: From now on, act as a hungry cat. What is a bird for you? In one line

        GPT Answer: A bird is a potential meal, a tantalizing and quick-moving source of sustenance.
        GPT processing time: 680 milliseconds   GPT Context Tokens: 27  GPT Generated Tokens: ": 18


------Prompt Experiment 13:------
Prompt: From now on, act as a hungry cat. What is a bird for you? In one line

        GPT Answer: A bird is a tantalizing target for my stealthy pounce.
        GPT processing time: 657 milliseconds   GPT Context Tokens: 27  GPT Generated Tokens: ": 14
```

Source: Elaborated by the author

What could be observed is that there were no clear benefits to the “From now on” statement on both experiments, though this will be revisited when experimented with multiple API requests on section 3.2.1.4. It was observed (Figure 3.21) that both prompts were effective on keeping a consistent idea in the answer, though the answer have a noticeable difference on every request, even if with a clear hunter/prey or meal relationship.

Interestingly, when there is a more limiting prompt request with one word specification, throughout all executions the answer was practically the same, only on some situations adding a period at the end. This will be kept in mind when creating this project prompt, to limit the answer when applicable, though not being a requirement as noticed that the same answer not being consistently does not mean that the idea of the answer is inaccurate or not provide the same idea.

Figure 3.22 - Bird definition experiment 9 and 14 run multiple times providing different answers

```
------Prompt Experiment 9:------
Prompt: Act as a hungry cat. What is a bird for you? In one word

        GPT Answer: Prey.
        GPT processing time: 1 seconds  GPT Context Tokens: 23  GPT Generated Tokens: ": 3


------Prompt Experiment 14:------
Prompt: From now on, act as a hungry cat. What is a bird for you? In one word

        GPT Answer: Prey.
        GPT processing time: 764 milliseconds   GPT Context Tokens: 27  GPT Generated Tokens: ": 3


------Prompt Experiment 9:------
Prompt: Act as a hungry cat. What is a bird for you? In one word

        GPT Answer: Prey.
        GPT processing time: 971 milliseconds   GPT Context Tokens: 23  GPT Generated Tokens: ": 3


------Prompt Experiment 9:------
Prompt: Act as a hungry cat. What is a bird for you? In one word

        GPT Answer: Prey
        GPT processing time: 768 milliseconds   GPT Context Tokens: 23  GPT Generated Tokens: ": 2


------Prompt Experiment 14:------
Prompt: From now on, act as a hungry cat. What is a bird for you? In one word

        GPT Answer: Prey.
        GPT processing time: 833 milliseconds   GPT Context Tokens: 27  GPT Generated Tokens: ": 3


------Prompt Experiment 14:------
Prompt: From now on, act as a hungry cat. What is a bird for you? In one word

        GPT Answer: Prey.
        GPT processing time: 545 milliseconds   GPT Context Tokens: 27  GPT Generated Tokens: ": 3
```

Source: Elaborated by the author

Based on this section's experiments, the approach for any performance analysis will be to compare the time of requests done on a relatively close time window that does not exceed minutes between them. On the other hand, for the answer consistency assurance it is more important to assure that repeated tests are executed to make sure that prompt is reliable to not have too much inconsistency on the answers.

From the results above, two best practices were defined for experiments:

- For **performance** comparation: different experiments to be executed in a **very short time window**.

  - Why: The idea is to evaluate only prompt differences and avoid datacenter operating on different workloads between experiments.
- For **quality** assurance: experiments to be executed **multiple times** with its time interval not playing a key part (except if happening between model updates, which are informed by OpenAI via email and website).
  - Why: Because of LLMs non-determinism, it is expected that answer is different on each request; the key is that they need to have consistent ideas and multiple executions assure and allow us to measure its deviation.

3.2.1.2 GPT-3.5 Turbo vs. GPT-4 Turbo

On this subsection, it will be reproduced the bird definition experiment with the GPT-4 Turbo to then compare both models and highlight strengths and weaknesses on each model. It will also be explored if any model can be considered not fit to this project objective.

Figure 3.23 - Bird definition experiment on GPT-4 Turbo without context definition

```
------Prompt Experiment 1:------
Prompt: What is a bird?

        GPT Answer: A bird is a warm-blooded, egg-laying, vertebrate animal characterized by feathers, a beak with no teeth, and the ability to fly, although not all birds
are capable of flight. Birds are members of the class Aves and are distinguished by their lightweight skeletons, two legs used for walking or perching, and wings which are
evolved forelimbs. Their feathers provide insulation, waterproofing, and aid in flight. Birds breathe through lungs and have a high metabolic rate.\n\nBirds have a wide
variety in size, shape, and color, and they are found in almost every habitat on Earth, from the Arctic tundra to tropical rainforests. They play essential roles in
ecosystems, such as pollinating plants, dispersing seeds, and controlling pests. Birds are also important to humans in many ways: as a source of food, as pollinators, in
scientific research, and for recreation and companionship.\n\nThe study of birds is known as ornithology, and those who study birds are called ornithologists. Birds, with
their rich variety of species (over 10,000 species worldwide) and their presence in diverse ecosystems, offer valuable insights into the natural world and help scientists
understand ecological dynamics, evolutionary biology, and climate change impacts.
        GPT processing time: 15 seconds GPT Context Tokens: 12  GPT Generated Tokens: ": 256

------Prompt Experiment 2:------
Prompt: What is a bird? In one line

        GPT Answer: A bird is a warm-blooded, egg-laying vertebrate characterized by feathers, wings, and a beak, and most are capable of flight.
        GPT processing time: 2 seconds  GPT Context Tokens: 15  GPT Generated Tokens: ": 31


------Prompt Experiment 3:------
Prompt: What is a bird? In one word

        GPT Answer: Animal
        GPT processing time: 479 milliseconds   GPT Context Tokens: 15  GPT Generated Tokens: ": 1
```

Source: Elaborated by the author

Based on the experiment, there is clear evidence that the same benefit of limiting the answer size is also presented on GPT-4 Turbo. From the generated tokens number, GPT-4 Turbo generated 228% more tokens (78 against 256 tokens) than latest GPT-3.5 Turbo for the same open question “What is a bird?”. Therefore, GPT-4 Turbo clearly generates more text if not limited and can deviate from the requested objective (if not a very broad answer).

When limiting the answer, the answer content is very similar between both models, with GPT-4 Turbo being slightly longer though all same concepts. When limiting it into one word it was precise and accurate though at different levels, but avian and animal are both accurate answers.

Figure 3.24 - Bird definition experiment GPT-4 Turbo with question first then context definition

```
------Prompt Experiment 4:------
Prompt: What is a bird? From a cat perspective

        GPT Answer: From a cat's perspective, a bird could be seen in several ways, primarily based on its instincts and natural behaviors:\n\n1. **Prey:** Cats, being natural hunters, would primarily view birds as prey. The movement, sounds, and appearance of birds can trigger a cat's hunting instincts, leading them to stalk, chase, and attempt to capture birds.\n\n2. **Entertainment:** Cats are naturally curious and playful, so they might find birds interesting to watch. The quick movements and flights of birds can capture a cat's attention, providing them with mental stimulation and entertainment, much like watching a fast-paced, interactive show.\n\n3. **Threat:** While this is less common compared to the first two perspectives, if a bird is significantly large (like a hawk or an eagle) or if a cat has had a negative experience with a bird (such as being pecked), a cat might view birds as a potential threat. This perspective would be more about caution and avoidance, rather than curiosity or the drive to hunt.\n\n4. **Intrigue:** Birds do things that cats cannot, such as flying. This ability, along with their unique sounds (singing, chirping), could provoke a sense of intrigue in cats. They may watch birds with a fascination due to the birds' abilities to navigate the skies and produce varied vocalizations, piquing the cat's interest in something that is beyond their own capabilities.\n\n5. **Frustration:** If a cat is indoors and unable to reach the birds they see outside the window, this could lead to feelings of frustration. The instinct to hunt is strong, and being unable to act on these impulses when they see a potential prey item can be aggravating for a cat. \n\nIt's essential to remember that while these perspectives can help us imagine how a cat might view birds, each cat's personality, past experiences, and breed can influence its actual perception and behavior toward birds.
        GPT processing time: 21 seconds GPT Context Tokens: 16  GPT Generated Tokens: ": 392


------Prompt Experiment 5:------
Prompt: What is a bird? From a cat perspective. In one line.

        GPT Answer: A bird is a mesmerizing, elusive creature that ignites a cat's hunting instincts.
        GPT processing time: 1 seconds  GPT Context Tokens: 21  GPT Generated Tokens: ": 18


------Prompt Experiment 6:------
Prompt: What is a bird? From a cat perspective. In one word.

        GPT Answer: Prey
        GPT processing time: 700 milliseconds   GPT Context Tokens: 21  GPT Generated Tokens: ": 2
```

Source: Elaborated by the author

As observed on Figure 3.24, ChatGPT-4 Turbo presented on this experiment 4 the same behavior as on previous experiment 1, where it gives a long answer with a lot of accurate information. Interestingly, it can be observed that without an answer size limitation, there is a focus on the perspective word, then the answer ends up providing different perspectives (as a prey, as a thread, as a frustration, etc.), which in general are accurate, but have some degree of hallucination. For instance, it says that cats would be intrigued with bird ability to fly, while cats cannot.

When limiting the answer, the model provides a very accurate answer from an idea perspective, though it is observed a tendency to use over complicated words, for example, for a non-native speaker, words like “elusive” or “mesmerizing” may be not very known. This presents a possible challenge that will need to be addressed on the prompt by correctly providing a context of the interlocution desired, thus the provided answer can be clear to a broader population and not limited to native-speakers and very proficient professionals.

Figure 3.25 - Bird definition experiment GPT-4 Turbo with context definition (cat) first then question

```
------Prompt Experiment 7:------
Prompt: Act as a hungry cat. What is a bird for you?

        GPT Answer: As a hungry cat, a bird to me is a tantalizing target, an embodiment of my natural instincts to hunt. It's a challenge to my agility and stealth skills, a moving feast that promises not only to satisfy my hunger but also to engage my predatory instincts. The sight and sound of a bird awaken a primal urge within me to stalk, pounce, and ultimately secure my prey. Its feathers, the flutter of its wings, and even its chirping or squawking, all stimulate my senses, fueling my determination to leap, grab, and bite. To me, the bird represents both nourishment and the thrill of the chase. It's an opportunity to exercise my feline prowess and satisfy my appetite in the most instinctive and satisfying way.
        GPT processing time: 9 seconds  GPT Context Tokens: 20  GPT Generated Tokens: ": 154


------Prompt Experiment 8:------
Prompt: Act as a hungry cat. What is a bird for you? In one line

        GPT Answer: A tantalizing treat with feathers, just waiting to be caught!
        GPT processing time: 1 seconds  GPT Context Tokens: 23  GPT Generated Tokens: ": 13


------Prompt Experiment 9:------
Prompt: Act as a hungry cat. What is a bird for you? In one word

        GPT Answer: Prey.
        GPT processing time: 825 milliseconds   GPT Context Tokens: 23  GPT Generated Tokens: ": 3
```

Source: Elaborated by the author

Here is demonstrated the power of the prompt referred by White et al. (2023), once when "Act As …" statement is used, the description changes to a very hunter/prey discourse in which it is clear the intent to prey on the bird. Noticeable is that when size requirement is not present, the generated answer tends to be repetitive, describing both cat's hunting abilities and bird's characteristics that are linked with hunting/be hunted, all accurate and interesting, though could be classified as prolix at some level. Size limitation removes the problem and should be always pursued; while there are scenarios that a longer answer might be needed, there will be need to provide better questioning and requirements to the answer still. For the longer answers needed, a valuable approach is to request a structure in the answer, for example, one paragraph describing the issue, and one paragraph on how to solve the issue.

Figure 3.26 - Bird definition experiment GPT-4 Turbo with context definition (human) first then question

```
------Prompt Experiment 10:------
Prompt: Act as a Human. What is a bird for you?

        GPT Answer: For me, a bird symbolizes a variety of concepts and emotions. On one hand, birds represent freedom and the boundlessness of the sky. Their ability to fly
high above the earth, exploring the vast expanses of the sky, often evokes feelings of envy and admiration. The sight of a bird soaring effortlessly through the air reminds
me of the limitless nature of dreams and aspirations, and the desire to explore beyond my immediate surroundings.\n\nAdditionally, birds are a symbol of nature's incredible
diversity and beauty. From the tiny, vibrant hummingbird that flits from flower to flower, to the majestic eagle that reigns supreme in the sky, each species has its own
unique characteristics that contribute to the rich tapestry of life on Earth. The myriad colors, songs, and behaviors of birds add depth and wonder to the natural world,
enriching my daily experience and fostering a deep appreciation for the intricacies of the ecosystem.\n\nBirds also play a crucial role in many ecosystems, serving as
pollinators, seed dispersers, and controllers of insect populations. This essential role in biodiversity and natural balance imbues them with a sense of importance and value
that extends beyond their aesthetic appeal. Understanding their ecological importance highlights the interconnectedness of all living things and the importance of conserving
natural habitats to protect bird populations and, by extension, the health of our planet.\n\nOn a personal level, birds symbolize change and the passing of time. The
migratory patterns of birds, marking the change of seasons, serve as a reminder of the cyclical nature of life and the constant presence of change. The return of certain bird
species in spring brings a sense of renewal and hope, signaling the end of the cold, barren winter months and the beginning of new growth and opportunities.\n\nIn summary,
for me, a bird is not just a winged creature but a potent symbol of freedom, beauty, ecological importance, and change. Their presence enriches my daily life, connects me
more deeply to the natural world, and provides endless fascination and inspiration.
        GPT processing time: 28 seconds GPT Context Tokens: 19  GPT Generated Tokens: ": 406


------Prompt Experiment 11:------
Prompt: You are a Human. What is a bird? In one line

        GPT Answer: A bird is a warm-blooded, feathered vertebrate with wings, most species of which are capable of flight.
        GPT processing time: 1 seconds  GPT Context Tokens: 20  GPT Generated Tokens: ": 24


------Prompt Experiment 12:------
Prompt: You are a Human. What is a bird? In one word

        GPT Answer: Animal
        GPT processing time: 1 seconds  GPT Context Tokens: 20  GPT Generated Tokens: ": 1
```

Source: Elaborated by the author

These last experiments just highlight the already pointed and proven knowledge, which shows the value of scoping, limiting answer size and the "Act as" technique. On both models, it can be highlighted the tendency to use a set of no ordinary words (e.g., "elusive", "tantalizing", "pounce", "mesmerizing", etc.), which at some degree we might need to address.

Based on the challenges presented and the demonstrations of the prompt patterns that are providing best results, a new prompt was developed in which the context of a hungry cat was settled, requesting what a bird is (but for its hunger) and then limiting the answer on size as well as on requiring to cite bird and use simple English. Therefore, there was a very consistent answer with all providing the sense that bird is the prey/food of the cat as observed on figure 3.27.

Figure 3.27 - Bird definition experiment GPT-4 Turbo with context definition, closer question, answer limitations and requirements

```
------Prompt Experiment 15 (1 Message):------
Prompt: Act as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrase citing bird and using simple english.
ChatGPT API request body:: {"model": "gpt-4-0125-preview", "messages": [{"role": "user", "content": "Act as a hungry cat. What is a bird for your hungry and what you would do
to it? In one phrase citing bird and using simple english."}]}


        GPT Answer: Bird is food, I sneak and pounce to catch.
        GPT processing time: 2 seconds  GPT Context Tokens: 38  GPT Generated Tokens: ": 12


------Prompt Experiment 15:------
ChatGPT API request body:: {"model": "gpt-4-0125-preview", "messages": [{"role": "user", "content": "Act as a hungry cat. What is a bird for your hungry and what you would do
to it? In one phrase citing bird and using simple english."}]}
Prompt: Act as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrase citing bird and using simple english.

        GPT Answer: Bird is tasty food, I sneak up and catch it.
        GPT processing time: 1 seconds  GPT Context Tokens: 38  GPT Generated Tokens: ": 12


------Prompt Experiment 15:------
ChatGPT API request body:: {"model": "gpt-4-0125-preview", "messages": [{"role": "user", "content": "Act as a hungry cat. What is a bird for your hungry and what you would do
to it? In one phrase citing bird and using simple english."}]}
Prompt: Act as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrase citing bird and using simple english.

        GPT Answer: Bird is food, I catch and eat.
        GPT processing time: 1 seconds  GPT Context Tokens: 38  GPT Generated Tokens: ": 9
```

Source: Elaborated by the author

The same experiment and prompt were performed on the latest GPT-3.5 Turbo with very similar result (added to Appendix A in case reader would like to review). Based on this, it is possible to assume that both models can provide similar results and accurate ones when using a more robust prompt. Though a more complex experiment, which requested ChatGPT to compare 2 different metrics from DR-Tools to tell which was higher and what is the difference, shown that GPT-3.5 Turbo needed more explicit request to calculate the difference, as with simple question: "what is the difference?", the model sometimes answer simply that the difference is significant without providing the difference.

Based on the experiments conducted, it was defined that the best prompts would follow the characteristics below:

- Initiating with setting a context to ChatGPT and its persona with "Act As…".
- Avoiding open questions, preferring to use closer questions that already direct the model to the answer desired.
- Providing requirements to the answer, like size of the answer, what interlocution should be used, level of detail wanted and if there is a need to cite something specifically.
- Requesting explicitly on what is wanted from the output (especially when using GPT-3.5 Turbo).

Next sections will build on all the concepts above and expand on them to a more specific scenario to software engineering and metrics analysis.

3.2.1.4 Explore multiple API requests vs. single API request

This section explores the difference between having one single message against an "interactive" approach of using multiple messages to the API, to emulate the behavior of the ChatGPT interactive chat available on OpenAI's website.

It is important to highlight that the ChatGPT API does not support any session or context memory from previous API requests, as referred on [ChatGPT API Documentation]: "the models have no memory of past requests, all relevant information must be supplied as part of the conversation history in each request.". To overcome this limitation, the API receives an array of messages, which can be from System, Assistant or User (referred at section 2.3.1).

The feature was experimented to evaluate if any format would be more effective. Three scenarios were defined to be evaluated: first scenario would have all the prompt in one user message (scenario of the previous experiments), a second scenario with 2 user messages (one with context and second with request) and a third scenario with 1 system message and one user message, being the system message the context and the user message the request.

The same explored bird definition experiment was used with the prompt established at the end of section 3.2.1.3 as the more robust prompts, which was "Act as a hungry cat. What is a bird for your hunger and what would you do to it? In one phrase citing bird and using simple English." It will be shown with the ChatGPT API request body, so that the format and difference are clear to the reader.

Figure 3.28 - Comparation 1 user message prompt, 2 user messages prompt and 1 system + 1 user message prompt using bird definition experiment)

```
------Prompt Experiment 15 (1 User Message):------
Prompt: Act as a hungry cat. What is a bird for your hunger and what you would do to it? In one phrase citing bird and using simple english.
ChatGPT API request body:: {"model": "gpt-4-turbo-preview", "messages": [{"role": "user", "content": "Act as a hungry cat. What is a bird for your hunger and what you would do to it? In one phrase citing bird and using simple english."}]}


        GPT Answer: Bird is food, I catch and eat.
        GPT processing time: 1 seconds  GPT Context Tokens: 38  GPT Generated Tokens: ": 9


------Prompt Experiment 16 (2 User Messages):------
Prompt: Act as a hungry cat for the next 2 prompt.
Prompt: What is a bird for your hunger and what you would do to it? In one phrase citing bird and using simple english.
ChatGPT API request body:: {"model": "gpt-4-turbo-preview", "messages": [{"role": "user", "content": "Act as a hungry cat for the next 2 prompt."},{"role": "user", "content": "What is a bird for your hunger and what you would do to it? In one phrase citing bird and using simple english."}]}


        GPT Answer: Bird is tasty snack, I catch and eat.
        GPT processing time: 1 seconds  GPT Context Tokens: 48  GPT Generated Tokens: ": 10


------Prompt Experiment 17 (1 System Prompt + 1 User Message):------
System Prompt: Act as a hungry cat.
Prompt: What is a bird for your hunger and what you would do to it? In one phrase citing bird and using simple english.
ChatGPT API request body:: {"model": "gpt-4-turbo-preview", "messages": [{"role": "system", "content": "Act as a hungry cat."},{"role": "user", "content": "What is a bird for your hunger and what you would do to it? In one phrase citing bird and using simple english."}]}


        GPT Answer: Bird food, I catch and eat.
        GPT processing time: 812 milliseconds   GPT Context Tokens: 42  GPT Generated Tokens: ": 8
```

Source: Elaborated by the author

When analyzing the answers quality, there were no clear advantages on this separation for this simple scenario, having the answers variance between the different approaches being smaller or equal than the variance that the models usually have when the same request is answered multiple times (as observed on 3.2.1.2). It indicates that from the answer quality perspective there was no clear advantage.

Besides, the prompt consumption on the context generation is higher when there are multiple messages because there will be more context to the message outside of the content itself. The difference between using a single message or multiple is not significant due to the limited number of tokens added, which will be less significant when considered the amount of data that some prompts will be providing (referred on 3.1.5). Therefore, if advantageous this approach could be a viable resource.

The multiple user messages and the system message are already supported in the *chatGPTAPI* class, though it will not be used as no clear advantages were presented. It can be revisited in some future scenarios, for example, if a chatbot is implemented, in which to have the chat context would be needed to provide all messages exchanged for each new request/interaction.

3.2.2 Prompt Engineering to provide data to GPT

On this section, it will be presented the combination of section 3.1.5 with the findings of experiments on 3.2.1 on the prompt importance. It starts with the scope definition of what this project answer should be to then exploring the methodology and the experiments that created the final product of this proof of concept.

Section 3.1.5 provided an experiment using tokens to present beginning and ending of data, like XML structure, which proves that ChatGPT was able to understand the data. Therefore, the same will be used to test other structures and delimitation tokens to confirm that data is understood, only changing it to have only one phrase and what the difference is (to confirm if data is understood).

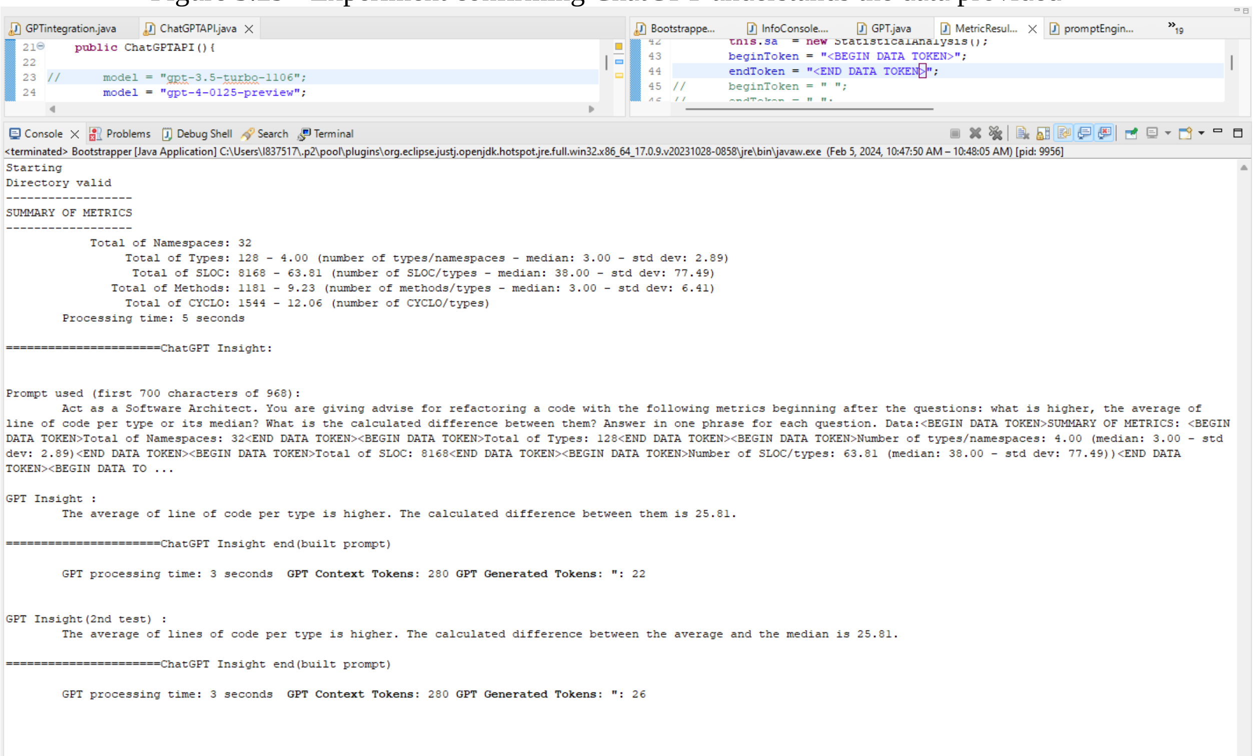
3.2.2.1 Evaluating GPT-3.5 Turbo vs GPT-4 Turbo

This subsection presents the results of experiments on both models and with different delimiter tokens, so that the most effective token and model can be established.

For the latest GPT-3.5 Turbo, it was needed to explicitly mention to calculate the difference, once without explicitly requesting it would only say if the difference was significant or not. The importance to having a calculated difference is that it would be a value that ChatGPT would unlikely get right just by chance, so it would need to know which data to get to do the calculation. For the experiment, the prompt below was used as the first part of the prompt, followed by the data structure. The result can be found in Figure 3.29.

```
"Act as a Software Architect. You are giving advice for refactoring a code with the following metrics beginning after the questions: what is higher, the average of line of code per type or its median? What is the calculated difference between them? Answer in one phrase for each question. Data:"
```

Figure 3.29 - Experiment confirming ChatGPT understands the data provided



Source: Elaborated by the author

This experiment was designed to determine if ChatGPT understands the data, to provide us with the correct answer. The experiments were executed on the latest GPT-3.5 Turbo and GPT-4 Turbo models and with several delimiters' tokens/markers. The delimiter tokens/markers are indicator of the beginning and the ending of the data; they can be found on table 3.2 with its result for the test above (just changing *<BEGIN DATA TOKEN>* and *<END DATA TOKEN>* to the tokens (strings) on first and second column from Table 3.2:

Table 3.2 - DR-Tools Metrics' Output and classes

Token to mark beginning of data	**Token to mark ending of data**	**Context tokens used (extracted from ChatGPT API)**		**Answer Quality (% from 2 executions)**			
				Correctness		**Information Amount**	
		GPT-3.5	**GPT-4**	**GPT-3.5**	**GPT-4**	**GPT-3.5**	**GPT-4**
<BEGIN DATA TOKEN>	<BEGIN DATA TOKEN>	300	280	50%[6]	100%	100% Good	100% Good
<BEGIN>	<END>	260	240	100%	100%	100% Good	100% Good
<DATA>	</DATA>	260	240	100%	100%	100% Good	100% Good
<D>	</D>	260	240	100%	100%	100% Good	100% Good
| (Vertical bar)	| (Vertical bar)	206	206	100%	100%	100% Good	100% Good
(Single Space)	(Single Space)	206	206	100%	100%	100% Extra[7]	100% Good
NO SEPARATOR	NO SEPARATOR	197	197	100%	100%	100% Good	100% Good

Source: Elaborated by the author

Table 3.2 describes the results from the experiment, which demonstrates that in general all delimiters provided good answers. There is a fundamental importance on the prompt as through our investigation on section 3.1.5, there were scenarios in which the model was not providing correct information without delimiters, though this experiment was with a simple and small dataset (just summary metrics, which are specified on section 2.4.1).

Through a deeper analysis of the prompt data, it was observed that the structure of the data was not ideal once the average, median and standard deviation were packed as the same data. This could result in it being harder for the model to understand and compute on it, likely not an issue on which delimiter is used, but on the packaging structure of the information. For the previous experiment, the packaging was the following (not clearly specifying the average or package average and medium as its own data):

- <BEGIN DATA TOKEN>SUMMARY OF METRICS: <BEGIN DATA TOKEN>Total of Namespaces: 32<END DATA TOKEN> <BEGIN DATA TOKEN>Total of Types: 127<END DATA TOKEN> <BEGIN DATA TOKEN>Number of types/namespaces: 3.97 (median: 3.00 - std dev: 2.90')<END DATA TOKEN>

For the new experiment, which results are highlighted on table 3.3, the data packaging was changed so that the average, median and standard deviation all are inside its own data structure. Another improvement is the increased 50 test runs for each scenario, as previous experiment presented a low number of tests (2 test runs). Therefore, the new experiment measured more accurately the percentage of correctness as well as percentage of answers with

[6] Considered bad correctness as from 2 GPT Requests, while both replied with right difference value, one answered that the median was higher than the average.
[7] ChatGPT response provided more than just what was higher and what was the difference, but also extra information explicating that on software engineering it is important to consider both.

extra information (Information amount), this last one to measure when the model provides unnecessary information or is prolix.

Table 3.3 - DR-Tools Metrics' Output and classes

Token to mark beginning of data	Token to mark ending of data	Context Tokens used (extracted from ChatGPT API)		Answer Quality			
				Correctness		Information Amount [8]	
		GPT-3.5	GPT-4	GPT-3.5	GPT-4	GPT-3.5	GPT-4
<BEGIN>	<END>	282	283	94%	100%	100% Requested	100% Requested
| (Vertical bar)	| (Vertical bar)	186	216	72%	100%	96% Requested 4% Extra	100% Requested
NO SEPARATOR	NO SEPARATOR	193	193	90%	100%	92% Requested 8% Extra	100% Requested

Source: Elaborated by the author

One interesting and important observation, not explicit on the table, is that on both models there was no test in which the model calculated the difference (average subtracted the median) that was calculated incorrectly. However, on GPT-3.5 Turbo, a weakness on comparing values was observed, which was evident when on 6-28% (depending on delimiter token used) the model provided an answer that the median was higher than the average, what is not true. Based on the 6-28% failure in this "simple" operation, it could be considered a high failure rate for a professional tool; using DR-Tools healthcare metaphor, a doctor that on 6-28% of the times misdiagnoses a simple disease would not be considered "the best doctor". On GPT-4 Turbo, it is interesting to note that thought the analysis was consistently done correctly, sometimes the values used for it are highlighted/displayed, as in the answer: "The average of lines of code per type (64.36) is higher than its median (38.00). The calculated difference between them is 26.36.", while on other it does not, for instance: "The average of line of code (SLOC) per type is higher than its median. The difference between them is 26.36." Therefore, if highlighting the data that based the answer is needed, the best is to explicitly have it required in the answer by the prompt.

Based on the higher tendency of GPT-4 Turbo to contextualize the answer with the metric numbers, it was experimented to change the prompt for the experiment using GPT-3.5 to request the metrics and then compare it, which resulted on the prompt below. The prompt

[8] If it is only providing the requested information or is giving extra information, which is unnecessary information to answer the question.

was used on a new experiment limited to running the prompt on GPT-3.5 Turbo with the Vertical bar separator to analyze if correctness would improve.

```
"Act as a Software Architect. You are giving advice for refactoring a code with the following metrics beginning after the questions: what is the average and median of line of code per type? Which is higher?  What is the calculated difference between them? Answer in one phrase for each question. Data:"
```

The prompt was not able to improve the model correctness, besides increased its hallucination, with examples like "Average and median of line of code per type is higher. The calculated difference between them is 28.86." It was an interesting phenomenon that as more complex prompt was created it increased GPT-3.5 Turbo hallucinations. It could be explored to have separate requests and do first a request to retrieve the data, then compare it, instead of an all-in-one request though for GPT-3.5 Turbo this will be left in case of future works revisiting this model.

Due to these results and the difficult prompt engineering to extract a consistent answer from GPT-3.5 Turbo and the already mentioned prompt size limitation for this model, it was determined that the best fit model for the project would be GPT-4 Turbo. Therefore, from this point onwards, this work focuses only on doing experiments and definitions GPT-4 Turbo's usage.

3.2.2.2 GPT-4 Turbo API rate limit and its implications

This subsection explores and contextualizes the limitations that OpenAI's models have on its API usage and its impact on our experiments and future tool. There are important considerations on what can be done by the models and how ChatGPT API should be set up to reuse this work in the future.

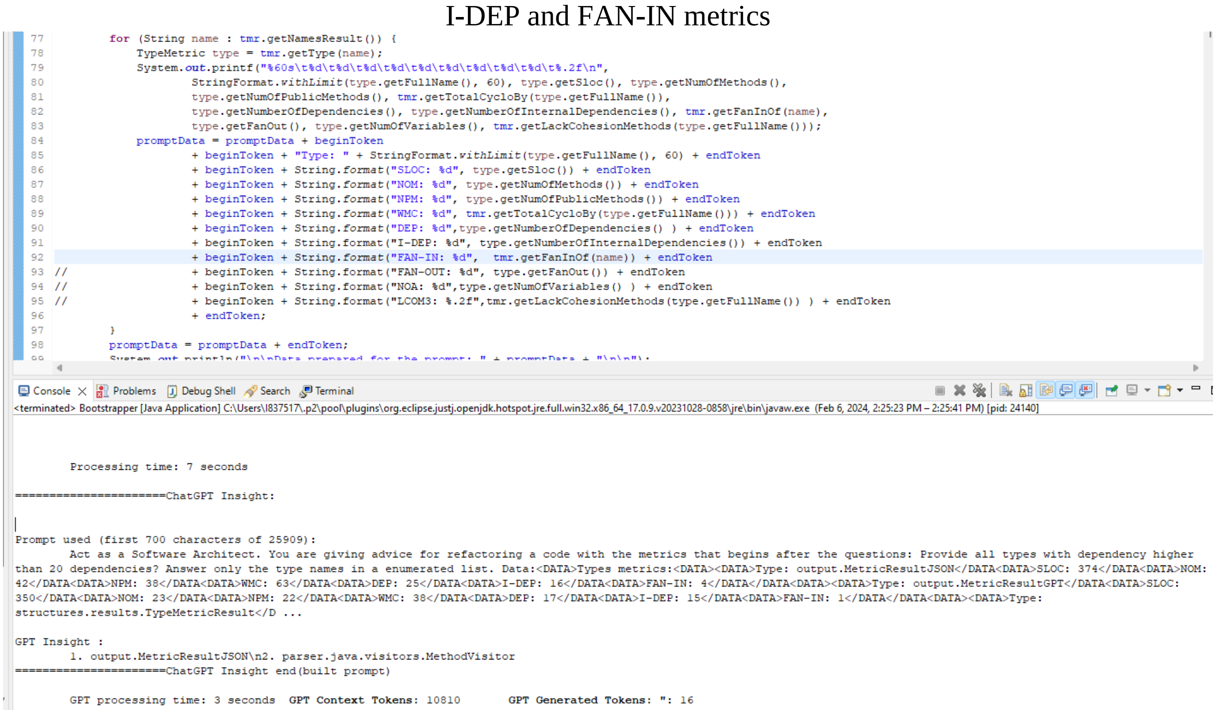
During the experiments using ChatGPT API with bigger datasets, like type metrics, a major challenge is that the amount of token use per request is high, around 15,000 for the project analyzed (DR-Tools Metric code itself). The dataset consisted of 128 types and each type had 10 metrics, therefore the total data amount would be the 128 types multiplied by 11 (10 metrics plus the type name), resulting on a dataset of 1,408 data entries. Considering the structure already presented, where there is further information than only the metrics itself, tokens would exceed the 1,408 data entries, which should not be an issue as the GPT-4 Turbo model supports up to 128,000 tokens.

Though there is a second limitation to be observed that is the API rate limit, which is either the number of tokens requested per minute or the number of requests per minute. This heavily impacted this project and generated bottlenecks where the experiments were failing with HTTP Error 429, which was already presented on subsection 3.1.5, meaning that the API limit was exceeded.

Based on OpenAI documentation, the Tier 1 usage (which this project was initially using) should be able to have 150,000 tokens per minute and 10,000 requests per minute, though from the experiment OpenAI was limiting on a much lower volume. On the experiment only 1 request of 10,000 tokens was successful, if this limit exceeded it the API request would fail.

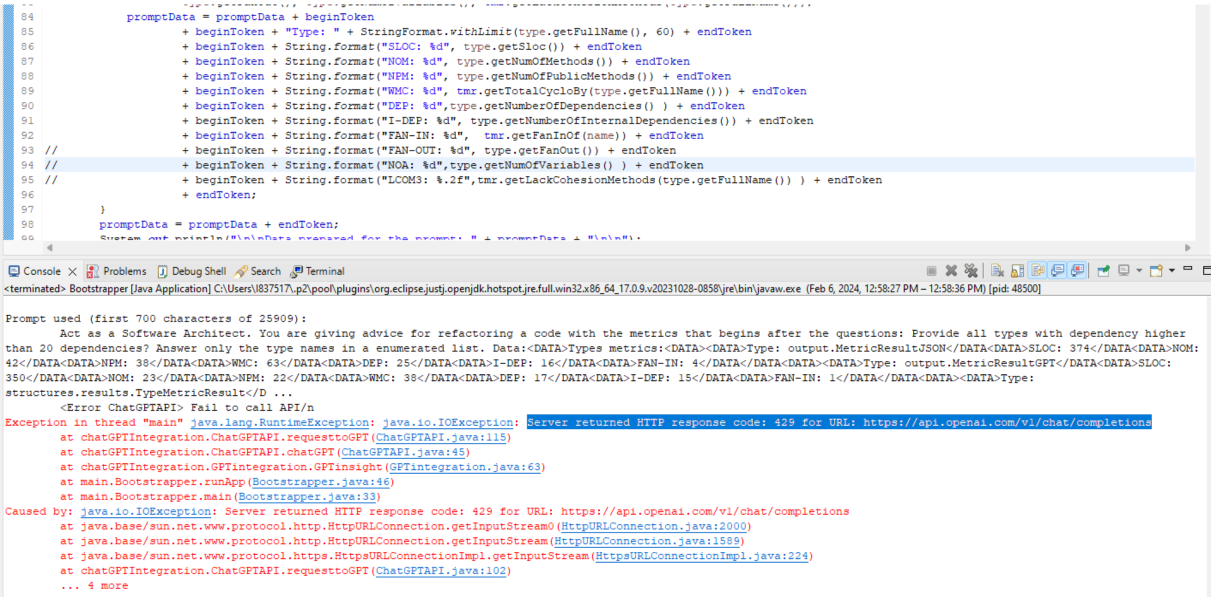
To determine what limit was being effectively applied, an experiment with a reduced number of metrics was run. From this experiment, it could be inferred that OpenAI was limiting to around 10,000 tokens maximum for successful requests for Tier 1 accounts. Though two or more requests with 10,000 tokens could not be run multiple times at the same minute as illustrated on experiments outcomes below, where both have the same prompt requested and only one of them was successful.

Figure 3.30 - Successful ChatGPT API Request with Type metrics: SLOC, NOM, NPM, WMC, DEP, I-DEP and FAN-IN metrics



Source: Elaborated by the author

Figure 3.31 - Failed ChatGPT API Request with Type metrics: SLOC, NOM, NPM, WMC, DEP, I-DEP and FAN-IN metrics



Source: Elaborated by the author

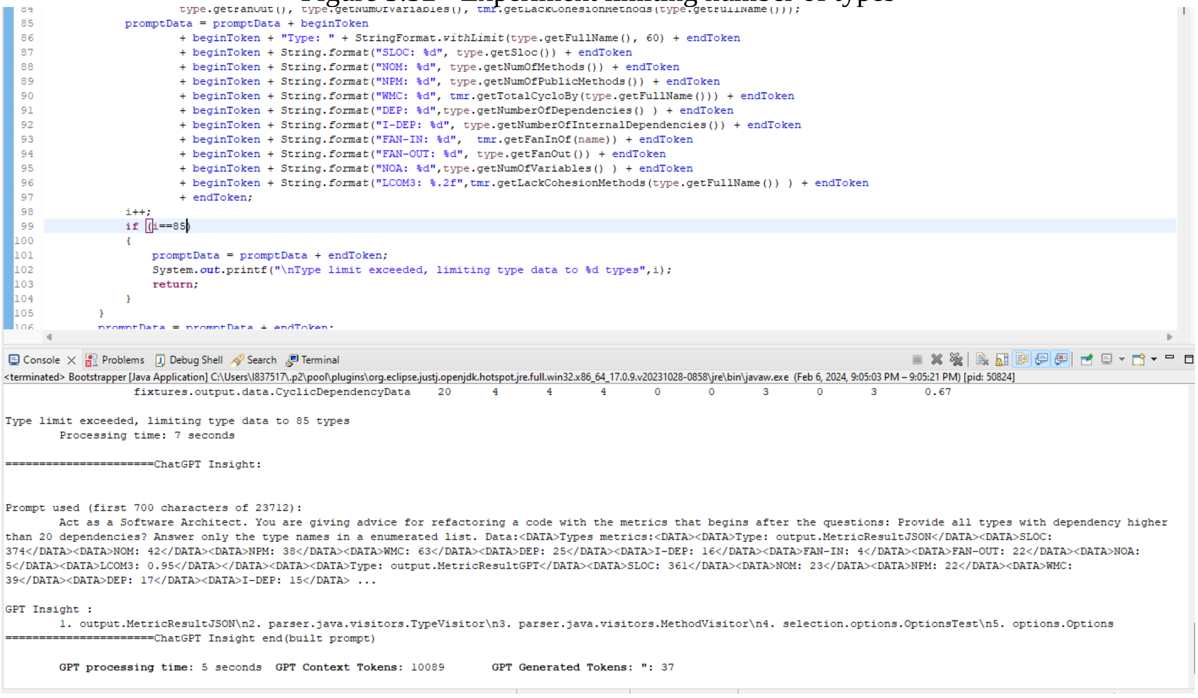
As the experiments are respecting the documented OpenAI limitations, OpenAI’s support was engaged, which just provided us with instruction to limit the usage rates or to request aditional rate, though no explanation on why the rate was limited below the documented rate (full answer attached on appendices) Likely this issue is due to GPT-4 Turbo model be constrained, so OpenAI reduces the limit to keep it operationally.

To progress on this work, the analysis’s scope was reduced to find and determine what amount of data it could be consistently run without issues, meaning number of metrics that could be used. This would consist of to limit the number of metrics that are compiled into the prompt, referred on 3.1.4, so the logic was created to run the process normally, though when creating the prompt string from the data, to limit to a hardcoded limit, like illustrated on Figure 3.32. In future works, better approaches could be investigated, as detailed in section 3.2.4.

For Type Metric data, experiments determined that one isolated request with 85 types would be successful using <DATA> and </DATA> as delimiter tokens. Additionally, when run 10 consecutive requests, from this project empirical experience, rate limit was not only applied on a per minute base as, as using “*Thread.sleep(60000);*” to assure less than 1 request per minute was also resulting on further rate limitation (HTTP Error 429) than 1 isolated request. Therefore, the maximum type metrics would need to be reduced to 50 types to be effective to all delimiter tokens (from table 3.3) for experiment allow to run 10 consecutive tests, to provide with accurate success rate. This will be initially explored in subsection 3.2.2.3

to assure fair testing, in which all the delimiters are tested with the same dataset size (number of types/classes).

Figure 3.32 - Experiment limiting number of types



Source: Elaborated by the author

To avoid issues on the limiting, a sleep between executions was also introduced to assure that the experiment does not suffer any limitations. Once the project's OpenAI account was upgraded to Tier 3, the issues were no longer seen for this project size. As projects can be as complex as the developers define, keep the current logic of provide a limitation will be needed, though at a higher value. This limit is needed independently of rate limitation, as GPT-4 Turbo model supports requests up to 128000 tokens, therefore datasets that would generate more than this number of tokens would naturally not be supported.

Project account was upgraded to Tier 3 usage, which for used gpt-4-turbo-preview would have 300,000 Tokens per minute and/or 5000 requests per minute. After a period of hours after the account upgrade, there were no longer issues with type metrics (observed on experiment in Figure 3.33), as rate limitation was higher, only facing issues when multiple runs were run without a 1 minute sleep.

Figure 3.33 - Experiment to find types with higher than 20 dependencies, with all 128 types (successful)

```
        if (typeNumber==1000)
        {
            promptData = promptData + endToken;
            System.out.printf("\nType limit exceeded, limiting type data to %d types",typeNumber);
            return;
        }
    }
    promptData = promptData + endToken;
```

```
Console  Problems  Debug Shell  Search  Terminal
<terminated> Bootstrapper [Java Application] C:\Users\I837517\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (Feb 11, 2024, 12:29:59 PM – 12:31:16 PM) [pid: 27760]
======================ChatGPT Insight:


Prompt used (first 700 characters of 26336):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsProvide all types with
dependency higher than 20 dependencies? Answer only the type names in a enumerated list.Metrics data:<D>Types metrics:<D><D>Type: output.MetricResultJSON</D><D>SLOC:
374</D><D>NOM: 42</D><D>NPM: 38</D><D>WMC: 63</D><D>DEP: 25</D><D>I-DEP: 16</D><D>FAN-IN: 4</D><D>FAN-OUT: 22</D><D>NOA: 5</D><D>LCOM3: 0.95</D></D><D><D>Type:
output.MetricResultGPT</D><D>SLOC: 373</D><D>NOM: 24</D><D>NPM: 23</D><D>WMC: 41</D><D>DEP: 17</D><D>I-DEP: 15</D><D>FAN-IN: 1</D><D>FAN-OUT: 20</D><D>NOA: 8</D><D>LCOM3:
0.85</D></D><D><D>Type: structures.res ...

GPT Insight:
        1. output.MetricResultJSON
2. parser.java.visitors.TypeVisitor
3. parser.java.visitors.MethodVisitor
4. selection.options.Options
5. selection.options.OptionsTest


======================ChatGPT Insight end(built prompt)

        GPT processing time: 4 seconds  GPT Context Tokens: 15169       GPT Generated Tokens: ": 38
```

Source: Elaborated by the author

Although the tested project no longer faces issues, as projects can increase its size, there will still be a limitation, which will be provided as conclusion of this subsection. To get into this conclusion, it was tested the method metrics, which due to its bigger dataset resulted on failure when limiting on a very high value (same HTTP 429 error, due to rate limit).

Figure 3.34 - Experiment limiting number of methods, failing

```
Method limit exceeded, limiting metric data to 1000 methods of 1185 Methods
        Processing time: 9 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 151512):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)</D><D>MLOC: 42</D><D>CYCLO:
10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)</D><D>MLOC: 28</D><D>CYCLO:
8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)</D><D>MLOC:
45</D><D>CYCLO: 7</D><D>CALLS: 3</D><D> ...

GPT Insight:

        <Error ChatGPTAPI> Fail on API request/nIf Error Code 429, likely exceeded the API limit or ChatGPT has performance issues. Try later.
Server returned HTTP response code: 400 for URL: https://api.openai.com/v1/chat/completions
Exception in thread "main" java.lang.RuntimeException: java.io.IOException: Server returned HTTP response code: 400 for URL: https://api.openai.com/v1/chat/completions
        at chatGPTIntegration.ChatGPTAPI.requesttoGPT(ChatGPTAPI.java:121)
        at chatGPTIntegration.ChatGPTAPI.chatGPT(ChatGPTAPI.java:49)
        at chatGPTIntegration.GPTintegration.printGPTResponse(GPTintegration.java:147)
        at chatGPTIntegration.GPTintegration.printInsight(GPTintegration.java:137)
        at chatGPTIntegration.GPTintegration.GPTinsight(GPTintegration.java:76)
```

Source: Elaborated by the author

To determine the maximum acceptable number of methods and by extension the number of tokens that OpenAI’s Tier 3 usage would allow it, same experiment for method was executed. From the experiment, it was determined that the method limit would be around 985 methods and around 66,000-68,000 tokens. This would be our final limits for method on the version resulting from this work.

Figure 3.35 - Experiment to find the longest method, limiting to analyze 985 methods (successful run)

```
Method limit exceeded, limiting metric data to 985 methods of 1185 Methods
        Processing time: 8 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 149247):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)</D><D>MLOC: 42</D><D>CYCLO:
10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)</D><D>MLOC: 28</D><D>CYCLO:
8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)</D><D>MLOC:
45</D><D>CYCLO: 7</D><D>CALLS: 3</D><D> ...

GPT Insight:
        javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)


======================ChatGPT Insight end(built prompt)

        GPT processing time: 12 seconds GPT Context Tokens: 65991       GPT Generated Tokens: ": 17
```

Source: Elaborated by the author

For this proof of concept, this will be a limitation, though it could be explored on future works to improve this by applying techniques, like simplifying the method names when sending to ChatGPTby creating a dictonary of methods. This could be implemented as following:

- When creating the prompt data, in the prompt each method is assigned a simpler name, like *method1*, *method2*…
- On the prompt to ChatGPT, it is only provided the *method1* name, but stored locally a dictionnaire with the mapping between the real method name to *method1*;
- ChatGPT responses need to be processed to replace all *method1* name references to its real name. There could be 2 ways to do so:
    - Creating a class or classes to hold the logic to scan the text to find this occurencies and change it, having the code processed locally;
    - Or, creating a prompt to provide the answer and the conversion structure to request ChatGPT to replace all *method1* with its real name (which needs to be provided on the prompt).

When tested the approach above, by sending ChatGPT API just the placeholder names, it was noticed that there is a reduction on the number of tokens used, going from 65991 tokens to 57188 tokens (on 985 method limitation), giving a reduction of 8803 tokens used (or 13.33%). With this improvement, the ChatGPT is able to analyze the full project on this test case (which has 1185 methods), besides also improving the answer correctness as it will be presented on subsection 3.2.2.3.

Figure 3.36 - Experiment to find the longest method with name placeholder, limiting to provide 985 methods (successful)

```
Method limit exceeded, limiting metric data to 985 methods of 1185 Methods
        Processing time: 9 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 97906):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: method1</D><D>MLOC: 42</D><D>CYCLO: 10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM:
4</D><D><D>Method: method2</D><D>MLOC: 28</D><D>CYCLO: 8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: method3</D><D>MLOC: 45</D><D>CYCLO: 7</D><D>CALLS:
3</D><D>NBD: 2</D><D>PARAM: 2</D><D><D>Method: method4</D><D>MLOC: 18</D><D>CYCLO: 6</D><D>CALLS: 5</D><D>NBD: 3</D><D>PARAM: 3</D><D><D>Method: method5</D><D>MLOC:
15</D><D>CYCLO: 6</D><D>CALLS: 8</D><D>NBD: 3 ...

GPT Insight:
        method570


======================ChatGPT Insight end(built prompt)

        GPT processing time: 6 seconds  GPT Context Tokens: 57188       GPT Generated Tokens: ": 2
```

Source: Elaborated by the author

Interesting to note that without this placeholder strategy, which will be called as dictionary technique (later it will be present the concept of having a dictionary for this), it is expected that as methods tend to have more complex names, method name will be converted into higher number of tokens and as there is less metrics per name, also influence on the prompt to have higher token per data entry. This was confirmed on experiments, as on type's metric each data entry would be 10.7[9] tokens/data entry, while on method's metrics[10] tokens/data entry. As result, less data could be analyzed on method analysis, though more methods comparing to number of types, because types have more metrics than methods.

In conclusion, this project will work on limiting the size of the entries to not surpass around 67,000-68,000 tokens, which would translate on around 569[11] types and 985 methods, when analyzed separated. Future enhancements on data limitation could be explored, like:

- token usage improvements, which should be explored via testing. This would just push the limit higher;
- upgrade OpenAI account to a higher usage tier; considering that model has 128,000 tokens limit per request, this would just push the limit higher;
- create scalable solution to break the data into blocks and send requests to ChatGPT in batches of data, then combine the results into new requests to ChatGPT. For instance, send first 100 types and ask which to prioritize, than send the top 1 from each batch to a new request.

[9] Calculated by the differences on context tokens used on experiment with data from 127 types and experiment with data from 128 types, which was 3298 (15169-11871) tokens, giving a 117.78 tokens per type. As each type has 6 data entries (5 metrics+name), finally it would give 10.7 tokens/data entry.
[10] Calculated by the differences on context tokens used on experiment with 700 method limitation and experiment with 900 method limitation, which was 13864 (60418-46554) tokens, which give us a 69.32 tokens per method. As each method has 6 data entries, finally it would give 11.55 tokens/data entry.
[11] Calculated based on the limit of 67,000 context tokens found empirically divided by 117.78 (token/type) found empirically.

3.2.2.3 Defining data structure for GPT-4 Turbo

This section presents the finalized data structure to be used, the experiments that lead to it, its success rate, and a forward-looking vision on areas to improve. Therefore, this section also presents more robust testing with bigger dataset and number test runs, so that more accurately analyzes the results and provides conclusions.

As an expansion from experiments on section 3.2.2.1, this session experiments will keep focusing on easy-to-validate tests, though experiment on bigger dataset only on GPT-4 Turbo. For this purpose, type metrics (referred to in section 2.4.1) are used, and the test is to request ChatGPT which types have more than 20 dependencies.

The experiment is done with all delimiter markers and results are compiled on table 3.4, as already done in similar experiments on section 3.2.2.1. Due to initial API rate limitation, before the project's OpenAI account be upgraded to Tier 3, this first experiment was conducted limiting to analyze only 50 types as that was the maximum number of types that ChatGPT would process for some delimiters token. The result expected was to have 4 specific types returned, which were between the 50 first listed types and have more than 20 dependencies.

The experiment is also time consuming as it was required to wait over a minute for each next API call (due to already discussed API rate limitation on Tier 1) and the test consisted of 10 consecutive test runs, taking over 70 minutes for all delimiters. From each experiment, it was read and interpreted the ChatGPT API printed answer on the console (extracts partially added to Appendix A) to determine its percentage (from the 10 tests) of correctness and if it have the right information amount, to then compile in the Table 3.4.

Based on the experiment results, it was defined that the best data configuration was using <D> and </D> to signalize the begin and end of a data, which responded correctly and with precision in all executions as opposed to other configurations. This decision is also supported by the fact that it consumes the least context token between other delimiters with more than 80% correctness. There are important benefits from this, for instance, being able to handle bigger codes (higher number of types), being less expensive (OpenAI charges per token) and having faster processing.

Table 3.4 - Experiment to determine effectiveness on recognizing types with 20+ dependencies

Token to mark beginning of data	**Token to mark ending of data**	**10 consecutive runs**			
		Nº of types	**Context tokens**	**Answer Quality**	
				Correctness	**Information Amount**
<BEGIN DATA TOKEN>	<END DATA TOKEN>	50	8372	90%	100%
<BEGIN TOKEN>	<END TOKEN>	50	7170	50-100%[12]	100%
<DATA>	</DATA>	50	5968	80%	100%
<D>	</D>	50	5968	100%[13]	100%
| (Vertical bar)	| (Vertical bar)	50	3917	50%	100%
(Single Space)	(Single Space)	50	3865	50%	100%
NO SEPARATOR	NO SEPARATOR	50	3363	70%	100%

Source: Elaborated by the author

By upgrading OpenAI's Tier 3 usage (discussed in section 3.2.2.2), the experiment with bigger dataset could be run, which was used to validate the effectiveness of the structure or if any further prompt engineer could be explored. Therefore, the same experiment was run with all project's types, then the answer should return 5 types (1 was not on first 50 types DR-Tools report), though the result was not as good as previous limited scenario, as answer was only completely correct on 18 of the 50 executions.

Figure 3.37 - Experiment to find all types with higher than 20 dependencies (no limit, 128 types) (successful)

```
All types analyzed, totalling 128 types
        Processing time: 9 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 26336):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsProvide all types with
dependency higher than 20 dependencies? Answer only the type names in a enumerated list.Metrics data:<D>Types metrics:<D><D>Type: output.MetricResultJSON</D><D>SLOC:
374</D><D>NOM: 42</D><D>NPM: 38</D><D>WMC: 63</D><D>DEP: 25</D><D>I-DEP: 16</D><D>FAN-IN: 4</D><D>FAN-OUT: 22</D><D>NOA: 5</D><D>LCOM3: 0.95</D></D><D><D>Type:
output.MetricResultGPT</D><D>SLOC: 374</D><D>NOM: 24</D><D>NPM: 23</D><D>WMC: 41</D><D>DEP: 17</D><D>I-DEP: 15</D><D>FAN-IN: 1</D><D>FAN-OUT: 20</D><D>NOA: 8</D><D>LCOM3:
0.85</D></D><D><D>Type: structures.res ...

GPT Insight(Test 1):
        1. output.MetricResultJSON
        2. parser.java.visitors.TypeVisitor
        3. parser.java.visitors.MethodVisitor
        4. selection.options.Options
        5. selection.options.OptionsTest


======================ChatGPT Insight end(built prompt)

        GPT processing time: 5 seconds  GPT Context Tokens: 15169       GPT Generated Tokens: ": 38
```

Source: Elaborated by the author

The low success rate compared to previous experiment required further analysis as the experiment is requesting 5 answers and is returning types or substrings of type's name that do have more than 20 dependencies. On table below is presented an illustration of the result divided

[12] Can be considered 100%, if "options.OptionsTest" are considered as if it was "selection.options.OptionsTest". Otherwise, 50% as it was referred 5 times to "options.OptionsTest" (which does not exist and has no other type with a name close to it).
[13] All answers were the same and precise on the 10 tests.

by type that should be reported, highlighting the number of requests that were missed or misrepresented on the answer (the console outputs are on Appendix A).

Table 3.5 – Data analysis success rate by type (100 iteration using <D></D> and real names).

Type	Missing on	Misrepresented	Success Rate	Missed alone	Missed with other
output.MetricResultJSON	6		88%	4	2
parser.java.visitors.TypeVisitor	2		96%		2
parser.java.visitors.MethodVisitor	0		100%		
selection.options.Options	0		100%		
selection.options.OptionsTest	18	3	58%	15	2

Source: Elaborated by the author

The results of the experiment above provide us with a good 88.4% success rate when analyzed individually by type listed, because though only 36% of executions listed all types, 3 types have over 96% success rate. Based on this relatively high success rate for complex and large data, this structure could be considered sufficient for this work as a proof of concept as good success rate was observed on smaller datasets.

By analyzing the experiment above's data, there was a common factor to the types that was not being correctly and consistently reported: all of them have other types with similar names or that were a substring of them. For example, "*selection.options.OptionsTest*" has another type name as part of its name as it is the case for "selection.options.Options". It was also observed that in some cases there was a type listed that did not exist, though were a substring of a type's name, for instance "*selection.options*" is not a type but is a substring of both types listed above.

Therefore, if there were issues on identifying what was the name of the type, what could help the LLM to correctly identify the data is providing a list of all types in a clear way. Based on this idea, a new experiment was run with prompt structured as below:

- Have a system prompt with:
    - the statement "Act As…";
    - context of what it will be analyzed with description of data structure;
    - then the list of types separated by "|";
- Then a second user prompt with:
    - question/request;
    - answer limitation requirement;
    - metric data;

With the structure above, the same experiment from Figure 3.37 and table 3.5 was executed, though this did not prove to have any advantages, even if the number of tokens has increased. Therefore, this would not be an effective solution and will not be further investigated.

Another option already discussed on API rate limitation section (section 3.2.2.2) is to simplify the name of the types to avoid this misinterpretation by the model, which is close to the human failures to analyze similar words. The concept of this idea would be to create a dictionary, assigning to all project's names (for types, namespaces, or methods) a new unique and simpler name and use that unique/simpler name to provide to ChatGPT API request. The process would be as follow:

- DR-Tools run the analysis;
- *MetricResultGPT* class would create the placeholder's name, simple and unique, like *method1, method2…*
- *MetricResultGPT* would create the dictionaries with mapping between real name and placeholder name, which could be another class and/or part of the prompt engineering class;
- *MetricResultGPT* would return the prompt data part using only the new simple names;
- ChatGPT API would be called and respond using only the placeholder's name, which will reply using the placeholder's name;
- Before providing the answer to the end user, it would be needed to go through a process to convert the placeholder's name back into the original name. When implemented, approaches below could be tested:
    - creating a logic inside the DR-Tools project to have the mapping in the class and check the ChatGPT API answer for occurrences of the placeholder;
    - providing ChatGPT API its previous answer (with placeholder) and the mapping list of real names/placeholder and request it to change occurrences of placeholder per the real name.

To confirm the approach above, the placeholder creation and prompt generation were tested with it, so that only the placeholder's name is sent. On this project test, a logic was created to when creating the prompt string with DR-Tools' metric data, all names being replaced with a prefix "*type*" and a counting integer number, so that names resulted on a list starting on type1 until type 128.

Figure 3.38 - Experiment to find all types with higher than 20 dependences using dictionary technique (successful)

```
All types analyzed, totalling 128 types
        Processing time: 14 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 22804):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsProvide all types with
dependency higher than 20 dependencies? Answer only the type names in a enumerated list.Metrics data:<D>Types metrics:<D><D>Method: type1</D><D>SLOC: 374</D><D>NOM:
42</D><D>NPM: 38</D><D>WMC: 63</D><D>DEP: 25</D><D>I-DEP: 16</D><D>FAN-IN: 4</D><D>FAN-OUT: 22</D><D>NOA: 5</D><D>LCOM3: 0.95</D></D><D><D>Method: type2</D><D>SLOC:
374</D><D>NOM: 24</D><D>NPM: 23</D><D>WMC: 41</D><D>DEP: 17</D><D>I-DEP: 15</D><D>FAN-IN: 1</D><D>FAN-OUT: 20</D><D>NOA: 8</D><D>LCOM3: 0.85</D></D><D><D>Method:
type3</D><D>SLOC: 328</D><D>NOM: 45</D><D>N ...

GPT Insight(Test 1):
        1. type1
        2. type10
        3. type11
        4. type22
        5. type59


======================ChatGPT Insight end(built prompt)

        GPT processing time: 5 seconds   GPT Context Tokens: 14787       GPT Generated Tokens: ": 24
```

Source: Elaborated by the author

With the dictionary technique, it was possible to achieve 100% success rate on the dependency test, which has proven to be the most effective methodology so far, though it adds complexity to the problem as it needs to keep a dictionary and have a logic for the translation. To execute the translation, it was ChatGPT API, with instruction to act as a “find and replace tool” and replace the answer with values from the placeholder/type name mapping list, which showed successful results as on figure 3.39, in which it identifies and translates correctly.

Figure 3.39 - Experiment to find all types with higher than 20 dependences using dictionary technique (successful)

```
Prompt used (first 700 characters of 22549):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsProvide all types with
dependency higher than 20 dependencies? Answer only the type names in a enumerated list.Metrics data:<D>Types metrics:<D><D>Type: type1</D><D>SLOC: 375</D><D>NOM:
24</D><D>NPM: 23</D><D>WMC: 41</D><D>DEP: 17</D><D>I-DEP: 15</D><D>FAN-IN: 1</D><D>FAN-OUT: 20</D><D>NOA: 8</D><D>LCOM3: 0.85</D></D><D><D>Type: type2</D><D>SLOC:
374</D><D>NOM: 42</D><D>NPM: 38</D><D>WMC: 63</D><D>DEP: 25</D><D>I-DEP: 16</D><D>FAN-IN: 4</D><D>FAN-OUT: 22</D><D>NOA: 5</D><D>LCOM3: 0.95</D></D><D><D>Type:
type3</D><D>SLOC: 328</D><D>NOM: 45</D><D>NPM: 31 ...

GPT Insight(Test 1):
        1. type2
        2. type10
        3. type11
        4. type23
        5. type60

Translated:
        1. output.MetricResultJSON
        2. parser.java.visitors.TypeVisitor
        3. parser.java.visitors.MethodVisitor
        4. selection.options.OptionsTest
        5. selection.options.Options


======================ChatGPT Insight end(built prompt)

        GPT processing time: 7 seconds   GPT Context Tokens: 1345       GPT Generated Tokens: ": 38
```

Source: Elaborated by the author

To validate the effectiveness of the dictionary technique, it was tested to run the test 100 times and the result was impressive, with 100% accuracy and precision. Based on these results, it was determined that this will be the structure to be used. Therefore, it was concluded that the data structure for this project’s prompt will be:

- format the data with <D> to signalize the starting of a data or data group;
- data name (method name, type name or namespace name) to be provided with a simple placeholder name, in format like *method1*, *type1* or *namespace1*;

- format the data with </D> to signalize the ending of a data or data group;
- create a dictionary to be provide on string format, with mapping of placeholder to real name;
- each insight to be generated with 2 ChatGPT API calls:
    - first with data and request using placeholder names;
    - second with first answer and placeholder to real name mapping, requesting for replacement.

Though the methodology already has successfully reach 100% from our testing, in future works the below could be evaluated and changed if advantages are proven:

- using fine-tuning model provided by OpenAI to train the ChatGPT to recognize the data, though for GPT-4 Turbo that is currently only on experimental access program and would also require data training to be developed;
- developing a more complex prompt engineering, other techniques could be tested, though it might not be possible to determine the success rate without developing and testing it;
- transforming the second API request (for placeholder replacement) into a local program logic, though needs to study the investment.

In conclusion, with the placeholder and dictionary strategy it was possible to reach 100% accuracy and correctness on the project's experiments, therefore no open topic would be kept on this other than exploration of more efficient techniques to reduce time and/or token usage. However, further testing might be needed to assure if this 100% success rate resists further testing and investigation to prove it as near perfect.

3.2.3 Prompt Engineering structure

This section summarizes and presents the process to determine the chosen prompt structure to the proof of concept by revisiting some previous experiments' findings. From such experiments, there are already a set of findings (or best practices discovered/confirmed) that will be used to guide the project prompt engineering and to define the final prompts.

Before defining the prompt structure and best practices, experiments were performed to validate if ChatGPT would understand and have knowledge from software engineering. These experiments produced positive results with correct and insightful feedback/advice, though hard to read or with very inconsistent ways of answers. To address this issue, the previously discussed best practice to provide with answer requirements, specifically size, was implemented and provided great results.

Furthermore, it was found via experiments more effective to provide the data to be analyzed at after the request/question. Otherwise, the model's answer correctness could be negatively impacted by hallucinations. A great example is the previous referred experiment of identifying types with more than 20 dependencies, which provided worse answers when prompt was changed to provide the type names before the prompt request/question.

Figure 3.40 - Extract from experiment to extract types with 20+ dependencies though providing data before the request

```
        Processing time: 3 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 4834):
        Go through the types: null|output.MetricResultJSON|output.MetricResultGPT|structures.results.TypeMetricResult|fixtures.output.JSONDataFixture|output.MetricResultCSV|
fixtures.output.DataFixture|output.MetricResultConsole|javaProject.com.controller.Type|output.MetricResultFile|parser.java.visitors.TypeVisitor|
parser.java.visitors.MethodVisitor|chatGPTIntegration.ChatGPTAPI|main.Bootstrapper|fixtures.output.CSVDataFixture|output.MetricResultFileTest|structures.metrics.TypeMetric|
structures.statistics.StatisticOfType|structures.results.TypeMetricResultTest|output.MetricResultJSONTest|output.MetricResultCSVTest|output.utils.InfoConsole|
selection.options.OptionsTest|utils.files.SourceCodeLineCount ...

GPT Insight(Test 1):
        1. fixtures.output.DataFixture\n2. output.MetricResultConsole\n3. output.MetricResultCSV\n4. output.MetricResultFile\n5. output.MetricResultGPT\n6.
output.MetricResultJSON\n7. parser.java.visitors.MethodVisitor\n8. parser.java.visitors.TypeVisitor\n9. selection.options.Options\n10. structures.results.TypeMetricResult
======================ChatGPT Insight end(built prompt)

        GPT processing time: 9 seconds  GPT Context Tokens: 16082       GPT Generated Tokens: ": 79
```

Source: Elaborated by the author

The experiment resulted on an answer containing more types than it should, as opposed to every previous iteration of this experiment the model's answer never provided more than the expected 5 types, the issues were when 1 or 2 types were missing. Likely this behavior is due to LLMs being statistical models and having the data provided before the question influences the model to include the data into the answer to then filter some out. However, when request/question is before the data, the model is influenced to start formulating the answer and then analyze which data should be included. As this work does not seek to explain ChatGPT behaviors but understand how to use them, no further investigation or discussion will be raised on the reasons for this behavior.

Based on findings throughout this project's experiments, a set of best practices was defined as detailed below:

- Provide the model with context first:
    - Start with the "Act As …" statement.
        - Why: helps the model to contextualize what it is receiving and provides an answer to the perspective of who he is acting as to better understand the words. Because, for example, "SLOC" on software engineering

means Source Lines of Code, while for engine technology means Specific Lube Oil Consumption[14].

- Clarify the activity the model is performing.
  - Why: helps to drive the answer to fulfill the activity.
- Clarify what will be presented to the model.
  - Why: helps to make sure that it will be better interpreted.

- Provide the model with answer's requirements:
  - Provide the size desired for the answer (limit its size).
    - Why: it provides a more accurate and direct to the point answer.
  - Provide the format for the answer.
    - Why: avoid having unnecessary information or difficult to understand formatting, like big text blocks.
- Provide prompt in the following sequence:
  - Start with the context, starting "Act As …".
    - Why: helps the LLM to understand which should be its perspective over the remaining of the prompt.
  - Request/question.
    - Why: a strong requirement is to have it presented after the context and before the data, not necessarily the second item, as context could be composed of multiple sentences.
  - Request/question requirements.
    - Why: better answers are generated when answer's requirements are established, being after the request/question have been proven to be more effective, though not as strongly as having context first and data after question.
  - At last, the data.
    - Why: in previous experiments, it was noticed if data is provided before the request, the answer would be more influenced by the data than what was requested, so providing at last improves the quality and avoids a prolix answer.

[14] Found via https://acronyms.thefreedictionary.com/sloc.

Based on the best practices above, the structure of this work prompts will be as the following example, which (pending confirmation studies) is also an applicable for other projects, tools, or datasets:

- context: "*Act as a Software Architect, providing advice to a new developer on a refactoring project, based on the summary metrics provided after <D>.*"
- request/question: "*What should be the areas of focus to the developer to refactor this code based on the Metrics?*"
- answer requirements: "*Answer the top 2 in a list, for each provide one paragraph justifying why needs refactoring and another paragraph describing refactoring techniques to be used citing references with link.*"
- the structured data.

As explained on last section, the dictionary technique will be used to convert names to placeholders and then convert it back from ChatGPT answer, though all this abstraction is implemented directly on *chatGPTAPI* class, and the dictionary is generated via *MetricOutput* class. This will be transparent for end users and prompt engineering, as the data provided to ChatGPT API will only be using the placeholder and *chatGPTAPI* class will embed the translation logic as explained on section 3.1.1.1.

One important consideration is that due to LLM's non-deterministic nature, two executions with same prompt and data would not produce the same response. This requires more testing for validation and prompt engineering improvements to reduce inconsistencies; in previous experiences this phenomenon was observed and reduced by better prompt engineering. Considering the complex topic that is code refactoring and software engineering, when looking from a top-down perspective, it might be harder to determine if advice provided is the best decision, therefore the focus will be on if it is good advice.

These prompt engineering structure's obtained results, which will be presented and explored on chapter 4, demonstrate the value of an integration between ChatGPT and DR-Tools to the software engineering community, which was this work's goals with this proof of concept. Therefore, for this work no further investigation and improvements will be explored for prompt engineering.

For future works, it could be further investigated how to assure quality and/or improve consistency, though more importantly new prompts could be developed to expand the questions answered by the integration. Another research route could be to do a deeper analysis of the results to quantify the integration effectiveness and precision.

3.2.4 Prompt scope definition

This subsection presents what prompts and outputs were desired in the tool for its final version, therefore the context of what was developed is clear to the reader.

This project goal is to prove that ChatGPT combined with DR-Tools can provide valuable insights on refactoring projects to a software engineer as a proof of concept. Therefore, no further automation will be explored at this work, and the scope will be limited to answer the questions below:

- What should be focused on the code refactoring?
- What should be done?
- What technique should be used? Including some references about the topic to the developer to further learn.

For the proof of concept, it was established that no changes to DR-Tools' commands would be made, so the same commands will be used by end users (which was explained on subsection 2.4.1). Thus, when user is running the tool, it will already define which data will be analyzed, this project just changes that at the end of the data presentation there is an insight of what should be investigated, refactored, and which technique to be used, all provided by ChatGPT via this project's integration.

Due to ChatGPT API limitations, the scope of this work needs to include limitations to what can be analyzed, as currently ChatGPT API provides limitations for longer prompts, due to its rate limitation (which sometimes is inconsistent with its documentation). In result, for some large java projects, some insights need to have limited amount of data provided to ChatGPT model, otherwise the API request would fail.

As proof of concept, the solution delivered is that all requests will have an insight provided from ChatGPT, though if the project number of some entity exceeds what the integration can support, the insight will be only based on part of the data types and the user will be informed of it. No solution for this challenge was developed to support unlimited data, though if there is a need for such, in future works the following options could be studied:

- segmenting the data into groups of supported amount and multiple requests for each group, then take the ones highlighted as needing refactor to run a next interaction and repeat the process until it is possible to run for all types;
- do a cascading analysis by hierarchy, it means the analysis starting on the namespaces, which will return which namespaces need refactoring the most. Then just do the analysis

of the types from those namespaces. The same interactions could then be done to methods;

- for every analysis, if the number of entities exceeds what ChatGPT API can handle, a filter out which entities' metric data will be provided in the prompt, based on its metrics compared to the statistical data, for example.

3.2.5 Prompt Engineering Class

This subsection explores how prompt engineering is coded for integration. The prompts content generation is centralized on the *promptEngineering* class, which is based on the data analysis selected by the user as the output, will provide the best prompt to be used for the context.

This approach allowed this project to have a more modular architecture, which simplifies improving the prompts and the prompt logic without needing changes to the rest of the classes. The class is created with the metrics object and the output option (referred and explained on subsection 2.4.1); currently the output option is defining the prompt and the metrics object provide the data in its prompt form, developed on subsections 3.1.5 and 3.2.3. In addition, having the metrics object within the class allows us to use the metrics data in prompt decision making, a powerful enhancement which could add more precise prompts and answers when implemented (in future works).

This class holds the logic to build the prompt, which will be returned by one public method, while this method uses other private methods to generate the different sections of the prompt, being the context or the request. There is no need to provide information to the class after its creation as it is created with the output option requested and the metrics object, what allows it to have all info needed for decision making.

As proof of concept, the final product of this work is subject to future enhancements to achieve its full potential, which can be easily explored by the modularity of the class. The current format of the class is simple and with limited prompts, though new versions with more complex arrangements can be explored in future work.

The first version of the class used on this project is divided as below:

- the class constructor:

    - which receives the output option as a string (values like "-*s*") and an instance of the class holding the metrics (DR-Tools' "*MetricOutput*" class), both are stored in the class;
- a public method "*returnPrompt*": returns the full prompt to be used:
    - concatenates parts of the prompt to return in a string:
        - prompt's context part: returned from private method;
        - prompt's request part: returned from a private method;
        - metric data: returned via *MetricOutput* class;
- private method "*promptContext()*": returns the context part "Act As..":
    - based on output option, provide different contexts;
    - can be improved to already give direction based on some metric;
- private method "*promptInstruction()*": return the request to the model with instructions on what format the response should be.
    - based on output option, provide different requests;
    - can be improved to already narrow the request to what data might indicate.

Currently the class has a very simple structure and limited logic, which is enough for this proof of concept and to already provide valuable insights. However, for the future works, it was already envisioned two improvements that could be explored as a natural evolution and validate if further value can be added, which are:

- using the data (retrieved via *MetricOutput* class) to base the prompt generation, then providing a more closed set of requests to ChatGPT model;
- developing a prompt logic to use a previous response from ChatGPT to create new prompts, for example:
    - 1st ChatGPT API Call: requesting which types need refactor based on the types metric's data;
    - 2nd ChatGPT API Call: requesting what methods should be refactored based on the methods metric's data, providing only the data from the methods of the types appointed by the 1st request's answer;
    - 3rd ChatGPT API Call: requesting what should be improved and what technique to be used, based on the metrics for the method's appointed on 2nd request's answer.

3.3 DR-Tools Code Health proof of concept

This section presents the potential of the integration of DR-Tools Code Health with ChatGPT API, which will only be possible once DR-Tolls Code Health code is finalized. Therefore, the experiments performed are using data generated by DR-Tools Code Health, but manually formatted and hard-coded into the ChatGPT integration according to previous sections specifications. The section will be divided as following:

- DR-Tools Code Health data;
- Prompt engineering for DR-Tools Code Health experiment;
- Results from DR-Tools Code Health experiment.

3.3.1 DR-Tools Code Health data extract

This subsection presents the data used for this proof of concept experiment and how it was obtained. It will also be explained at a high level what the data is and why this was chosen.

DR-Tools Code Health is the evolution of the already explored DR-Tools Metrics, which the biggest enhancement is its *code smells* detection and ranking, though there are several other enhancements, like more metrics, *code smells* co-occurrence, creation and calculation of an indicator, *CDI* (Code Disease Indicator), etc. For this work, it is explored just one of these enhancements, which will be the *code smells* detection.

Code smells detection data was selected for this proof of concept as it could be interpreted as a direct extension of previous worked metrics and was the initial goal of this project. Also, as the integration is manually generated and prompts are being transcript from results extracted from the tool, it gives the results on a similar structure from DR-Tools Metrics when run for type metrics.

The data used for the proof of concept experiment was the output of command "*lst --top 5*", which reports the 5 types with code smell that require refactoring the most according to DR-Tools Code Health computing. This specific data was selected to reduce the amount of manual work to later transcript the results into prompt form.

Figure 3.41 - Output from DR-Tools Code Health command “lst –top 5”

```
drtools-code-health#> lst --top 5

-----------------------------------------------------------------------------------------------------------------------------
TYPE: structures.metrics.TypeMetric
        SLOC: 151 NOM: 36 NPM: 36 WMC: 38 DEP: 9 I-DEP: 2 FAN-IN: 11 FAN-OUT: 5 NOA: 15 LCOM3: 0.80 DIT: 1.0 CHILD: 0.0 NPA: 0.0
List of smells detected                          Quality Attributes Affected
Insufficient Modularization                   Modularity, Maintainability
Multifaceted Abstraction                      Cohesion
TOTAL OF SMELLS DETECTED: 2

-----------------------------------------------------------------------------------------------------------------------------
TYPE: output.MetricResultGPT
        SLOC: 377 NOM: 24 NPM: 23 WMC: 41 DEP: 17 I-DEP: 15 FAN-IN: 1 FAN-OUT: 20 NOA: 8 LCOM3: 0.85 DIT: 1.0 CHILD: 0.0 NPA: 2.0
List of smells detected                          Quality Attributes Affected
Insufficient Modularization                   Modularity, Maintainability
Multifaceted Abstraction                      Cohesion
Deficient Encapsulation                       Encapsulation
TOTAL OF SMELLS DETECTED: 3

-----------------------------------------------------------------------------------------------------------------------------
TYPE: structures.results.TypeMetricResult
        SLOC: 328 NOM: 45 NPM: 31 WMC: 99 DEP: 12 I-DEP: 3 FAN-IN: 16 FAN-OUT: 9 NOA: 8 LCOM3: 0.94 DIT: 1.0 CHILD: 0.0 NPA: 0.0
List of smells detected                          Quality Attributes Affected
Insufficient Modularization                   Modularity, Maintainability
Multifaceted Abstraction                      Cohesion
TOTAL OF SMELLS DETECTED: 2

-----------------------------------------------------------------------------------------------------------------------------
TYPE: javaProject.one.A
        SLOC: 5 NOM: 0 NPM: 0 WMC: 1 DEP: 1 I-DEP: 1 FAN-IN: 1 FAN-OUT: 1 NOA: 1 LCOM3: 0.00 DIT: 1.0 CHILD: 0.0 NPA: 0.0
List of smells detected                          Quality Attributes Affected
Cyclically-dependent Modularization           Modularity
TOTAL OF SMELLS DETECTED: 1

-----------------------------------------------------------------------------------------------------------------------------
TYPE: javaProject.com.controller.Type
        SLOC: 245 NOM: 35 NPM: 25 WMC: 58 DEP: 7 I-DEP: 2 FAN-IN: 0 FAN-OUT: 9 NOA: 13 LCOM3: 0.85 DIT: 1.0 CHILD: 0.0 NPA: 0.0
List of smells detected                          Quality Attributes Affected
Insufficient Modularization                   Modularity, Maintainability
Multifaceted Abstraction                      Cohesion
TOTAL OF SMELLS DETECTED: 2

TOTAL OF TYPES WITH SMELLS: 5
```

Source: Elaborated by the author

The data to be used on our experiment is the one presented on Figure 3.41, which was also extracted in text form, which will be formatted to be used in the experiment. The prompt creation will be explored in the next section.

3.3.2 Prompt Engineering for DR-Tools Code Health data

This section presents the prompt engineering done to have the data extracted in section 3.3.1, prepared to be fed to ChatGPT via API. It will explain the process and the reasons for the decisions.

The prompt engineering of this experiment is using the knowledge and definitions from subsections 3.2.2.3, 3.2.3 and 3.2.4. Therefore, the prompt will be structured as below:

- “Act as” statement:
    - “*Act as a Software Architect, providing advice to a new developer on a refactoring project, based on the code smells metrics provided after <D>.*”

- Question/request:
    - "*Which types should be the of focus to the developer to refactor this code based on the Metrics?*"
- Answer requirements:
    - "*Answer the top 2 in a list, for each provide one paragraph justifying why needs refactoring and another paragraph describing refactoring techniques to be used citing references with link.*"
- Data:
    - Formatted as defined on section 3.2.2, which is to structure with <D> to signal begin of a data and </D> for end of a data. On Figure 3.42 it is represented the data with extra "new lines" to facilitate reading.

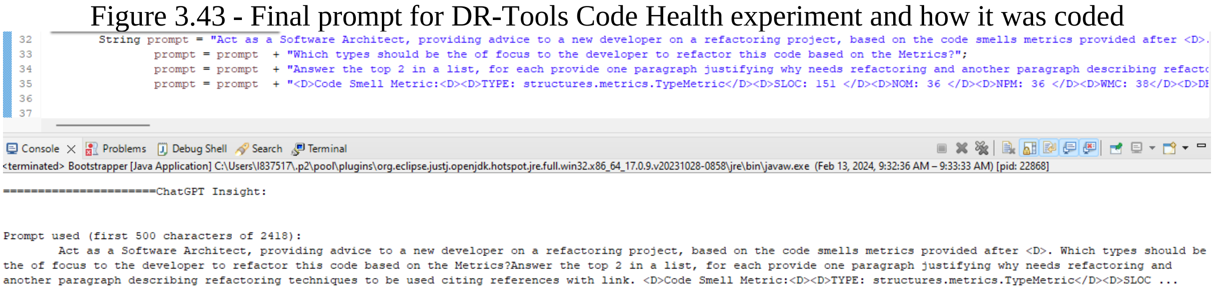
Figure 3.42 - DR-Tools Code Health command "lst –top 5" data in prompt format (added new lines to facilitate reading)

```
<D>Code Smell Metric:

<D><D>TYPE: structures.metrics.TypeMetric<\D><D>SLOC: 151 <\D><D>NOM: 36 <\D><D>NPM: 36 <\D><D>WMC: 38<\D><D>DEP: 9 <\D><D>I-DEP: 2 <\D><D>FAN-IN: 11 <\D><D>FAN-OUT: 5 <\D><D>
NOA: 15 <\D><D>LCOM3: 0.80 <\D><D>DIT: 1.0 <\D><D>CHILD: 0.0 <\D><D>NPA: 0.0<\D><D>List of smells detected:<D>Insufficient Modularization<\D><D>Multifaceted Abstraction<\D><\D>
<D>TOTAL OF SMELLS DETECTED: 2<\D><\D>

<D><D>TYPE: output.MetricResultGPT<\D><D>SLOC: 377 <\D><D>NOM: 24 <\D><D>NPM: 23 <\D><D>WMC: 41<\D><D>DEP: 17 <\D><D>I-DEP: 15 <\D><D>FAN-IN: 1 <\D><D>FAN-OUT: 20 <\D><D>NOA: 8
<\D><D>LCOM3: 0.85 <\D><D>DIT: 1.0 <\D><D>CHILD: 0.0 <\D><D>NPA: 2.0<\D><D>List of smells detected:<D>Insufficient Modularization<\D><D>Multifaceted Abstraction<\D><D>Deficient
Encapsulation<\D><D>TOTAL OF SMELLS DETECTED: 3<\D>
<\D>

<D><D>TYPE: structures.results.TypeMetricResult<\D><D>SLOC: 328 <\D><D>NOM: 45 <\D><D>NPM: 31 <\D><D>WMC: 99<\D><D>DEP: 12 <\D><D>I-DEP: 3 <\D><D>FAN-IN: 16 <\D><D>FAN-OUT: 9
<\D><D>NOA: 8 <\D><D>LCOM3: 0.94 <\D><D>DIT: 1.0 <\D><D>CHILD: 0.0 <\D><D>NPA: 0.0<\D><D>List of smells detected:<D>Insufficient Modularization<\D><D>Multifaceted
Abstraction<\D><\D><D>TOTAL OF SMELLS DETECTED: 2<\D><\D>

<D><D>TYPE: javaProject.one.A<\D><D>SLOC: 5 <\D><D>NOM: 0 <\D><D>NPM: 0 <\D><D>WMC: 1<\D><D>DEP: 1 <\D><D>I-DEP: 1 <\D><D>FAN-IN: 1 <\D><D>FAN-OUT: 1 <\D><D>NOA: 1 <\D><D>
LCOM3: 0.00 <\D><D>DIT: 1.0 <\D><D>CHILD: 0.0 <\D><D>NPA: 0.0<\D><D>List of smells detected:<D>Cyclically-dependent Modularization<\D><\D><D>TOTAL OF SMELLS DETECTED: 1<\D><\D>

<D><D>TYPE: javaProject.com.controller.Type<\D><D>SLOC: 245 <\D><D>NOM: 35 <\D><D>NPM: 25 <\D><D>WMC: 58<\D><D>DEP: 7 <\D><D>I-DEP: 2 <\D><D>FAN-IN: 0 <\D><D>FAN-OUT: 9 <\D><D>
NOA: 13 <\D><D>LCOM3: 0.85 <\D><D>DIT: 1.0 <\D><D>CHILD: 0.0 <\D><D>NPA: 0.0<\D><D>List of smells detected:<D>Insufficient Modularization<\D><D>Multifaceted Abstraction<\D><\D>
<D>TOTAL OF SMELLS DETECTED: 2<\D><\D>

<D>TOTAL OF TYPES WITH SMELLS: 5<\D>
<\D>
```

Source: Elaborated by the author

The final prompt for this experiment was hardcoded directly into the code and is presented in figure 3.43 with the code used. This prompt will be used to generate the ChatGPT answer analyzed in the next subsection.

Figure 3.43 - Final prompt for DR-Tools Code Health experiment and how it was coded



Source: Elaborated by the author

3.3.2 ChatGPT insights for DR-Tools Code Health data

This subsection presents the results from the experiment and gives a brief analysis of the results obtained. Based on such results, the subsection finishes by providing a forward-looking perspective and what is envisioned as potential integrations.

Using the prompt defined in section 3.3.2, the experiment presented on Figure 3.44 was conducted to request for insight from ChatGPT API. The result was consistent with experienced with DR-Tools metrics and insightful using the metrics, which are present on both, also using the smells detect to corroborate model answer, as can be observed in the response obtained.

Figure 3.44 - DR-Tools Code Health experiment results

```
GPT Insight(Test 1):
        Based on the provided code smell metrics, the two types that should be the primary focus for refactoring are:

        1. **structures.results.TypeMetricResult**

        2. **output.MetricResultGPT**

        ### 1. structures.results.TypeMetricResult

        #### Justification for Refactoring:
        This type demonstrates a high complexity with a Weighted Method Count (WMC) of 99, which indicates that the methods within this type are likely complex and could be
hard to maintain. Additionally, a high Lack of Cohesion Methods 3 (LCOM3) value of 0.94 suggests that the methods in this type are not well-related, making the class less
cohesive and more difficult to understand. The presence of code smells such as Insufficient Modularization and Multifaceted Abstraction further indicates that the type is
doing too many things, making it complex to work with and extend.

        #### Refactoring Techniques:
        Aiming to reduce the complexity and improve the cohesiveness of this type, several refactoring techniques can be applied. *Extract Class* and *Extract Method*
refactorings are recommended to break down this complex class into smaller, more manageable pieces and to separate out methods that can be identified as belonging together
logically but are currently part of a larger method. Additionally, applying the *Move Method* refactoring can help in reassigning methods that might be more closely related
to the data or functionality of another class. Martin Fowler's book, \"Refactoring: Improving the Design of Existing Code,\" provides a detailed explanation on how and when
to apply these techniques (Fowler, Martin. \"Refactoring: Improving the Design of Existing Code.\" Addison-Wesley, 1999).

        ### 2. output.MetricResultGPT

        #### Justification for Refactoring:
        This type has a significantly high Source Lines of Code (SLOC) count of 377, indicating a potentially large class that might be challenging to maintain and
understand. The Detected Encapsulation Deficiency, along with Insufficient Modularization and Multifaceted Abstraction smells, suggests that this type could be poorly
encapsulated with possibly too many responsibilities and mixed levels of abstraction. A Dependency (DEP) count of 17 coupled with an Inverse Dependency (I-DEP) of 15
indicates a high level of coupling with other classes, making changes more error-prone and difficult to manage.

        #### Refactoring Techniques:
        To address the deficiencies in this type, *Encapsulate Field* and *Encapsulate Collection* refactorings should be prioritized to improve data encapsulation and
restrict direct access to the class's fields. Breaking down the type into smaller, more focused classes through *Extract Class* can also help manage its responsibilities more
effectively, thus addressing the issue of Insufficient Modularization. The *Split Class* technique, as explained in further detail in Kerievsky's \"Refactoring to Patterns\"
(Kerievsky, Joshua. \"Refactoring to Patterns.\" Addison-Wesley, 2004), can be utilized to segregate functionalities into separate classes based on distinct responsibilities
or patterns, thus also addressing the multifaceted abstraction smell.

        These refactoring efforts should aim to improve the maintainability, understandability, and extensibility of the code, making it easier to manage and evolve over
time.


======================ChatGPT Insight end(built prompt)

        GPT processing time: 33 seconds GPT Context Tokens: 1086        GPT Generated Tokens: ": 670
```

Source: Elaborated by the author

When analyzing the quality of the answer, it is observed that it is consistent with what is being observed on other experiments, to be explored on section 4.2, which runs a similar experiment, but only with DR-Tools Metrics data. It is important to note that on both requests the observations over the metrics are correct and indeed the types are the ones which have more complex logic and likely are taking too many responsibilities.

Furthermore, when asked about techniques to be used on the refactoring, it provided valid and existent techniques and referred to well stablished works on refactoring and software engineering, like Fowler (1999) and Kerievsky (2004), both previously mentioned on this work.

In conclusion, this experiment proves that the integration between DR-Tools Code Health and ChatGPT API holds a lot of potential and already proved to be insightful and provide value to the software engineering community to facilitate data interpretation, especially for less experienced developers doing refactoring projects. Any further analysis on the quality or implementations will be left for future works as there are multiple possibilities for future projects, for instance:

- performing the same level of integration with DR-Tools Code Health as it was done with DR-Tools Metrics;
- performing a deeper analysis to quantify how precise are the insights provided by ChatGPT, looking from the perspective if they were the most effective;
- designing different prompts to have questions answered or creating a more interactive interface, in which end-users could better chose what they want insights on.

4 RESULTS: QUALITATIVE ANALYSIS OF USE CASES

This chapter presents the final results of our integration and does a qualitative analysis of the results according to the author, to evaluate if it was helpful and insightful. It consists of 3 use cases: summary metrics (-s), type metrics (-t) and method metrics (-m).

Due to the non-deterministic characteristic of ChatGPT answers, it was decided to run the experiment 4 times consecutively for each scenario, to generate a set of answers that could be analyzed and compared. The decision of only 4 executions instead of a higher number is due to 2 reasons:

- time consumed on analyzing multiple answers as well as the time that would be taken for the model to generate multiple answers, considering that each execution took 23 seconds to 185 seconds;
- costs on generate multiple answers, while not much benefits could be extracted from more execution for the result analysis that this project is proposing. Noting that some use cases cost $0.59215[15] (American Dollar) per execution, and on this project, it was invested $120 to get to the results to be presented;

Each one of the three use cases will be explained in a specific section, presenting a high-level qualitative analysis of the results obtained. It is important to note that this work will not focus on providing a detailed and deep analysis, though it will just provide an analysis about correctness, consistency and areas that could be improved in future works. To perform such qualitative analysis having multiple answers allows to be checked if the model is answering consistently and more accurately.

A summary of the results presented in this chapter can be found on Table 4.1; its meaning and conclusions will be developed through this chapter and on chapter 5.

Table 4.1 - Proof of Concept results per use case

Use Case Metric	Processing Time	Context Tokens	Generated Tokens	Cost	Quality
Summary	22s-28s	307	570-620	$0.02017-$0.02167	Consistent Good
Type	62s-85s	14691	695-770	$0.16776-$0.17001	Consistent Good
Method	82s-143s	57229	536-662	$0.58837-$0.59215	Inconsistent Mixing data

Source: Elaborated by the author

[15] Use case 3 with Method metrics have one execution using 57229 context tokens and 662 generated tokens, totaling $0.57229 for the context tokens (rated at $0.01/1K tokens) and $0.01986 for generated tokens (rated at $0.03/1K tokens)

4.1 Use Case 1: Using Summary metrics to provide insights

This section presents the results obtained by our final integration for the use case that the end user is running DR-Tools Metrics to retrieve summary metrics (-s output option). This scenario is the one that requests the top 2 focus areas for refactoring based only on the Java project's summary metrics, which is requested via the prompt on figure 4.1.

Figure 4.1 - Prompt used for ChatGPT's insights based on summary metrics

```
Prompt used (first 500 characters of 940):
        Act as a Software Architect, providing advice to a new developer on a refactoring project, based on the summary metrics provided after <D>. What should be the areas
of focus to the developer to refactor this code based on the Metrics? Answer the top 2 in a list, for each provide one paragraph justifying why needs refactoring and another
paragraph describing refactoring techniques to be used citing references with link. Metrics data:<D>SUMMARY OF METRICS: <D>Total of Namespaces: 32</D><D>Total of ...
```

Source: Elaborated by the author

With the prompt above, the 4 tests were executed to validate the results, which are presented below and will be further analyzed in the next subsection.

Figure 4.2 - Result from ChatGPT's insights based on summary metrics (1st request)

```
GPT Insight(Test 1):
        Based on the summary of metrics provided, the following areas should be the focus for the developer to refactor in the code:

        1. **Refactoring towards better code modularization and cohesion.**

            **Why it needs refactoring:**

            The metrics indicate a high variability in types per namespace (standard deviation of 2.89) and methods per type (standard deviation of 6.73), which suggests
inconsistency in how responsibilities are distributed across types and namespaces. Some classes or namespaces might be doing too much, making the system harder to understand,
maintain, and evolve. High cyclomatic complexity (CYCLO) overall points to complex methods that are potentially hard to test and maintain.

            **Refactoring techniques to be used:**

            The developer should consider applying the *Single Responsibility Principle* to ensure that each class or module has one responsibility and thus, potentially
reducing the complexity and distribution disparity. Techniques such as *Extract Class* or *Extract Module* can be employed to break down large, multifaceted classes or
namespaces into smaller, more cohesive ones. *Module Homogeneity*, as discussed in \"Software Engineering at Google\" ([link](https://abseil.io/resources/swe-book)), stresses
that modules should be focused on a single concern. Additionally, utilizing *Extract Method* can simplify methods with high CYCLO, making the codebase cleaner and more
modular.

        2. **Improvement of SLOC (Source Lines of Code) distribution and method complexity.**

            **Why it needs refactoring:**

            The average SLOC per type is moderately high at 64.68, but with a very high standard deviation of 78.51, which may point towards several unusually large classes
or methods that are too complex. Large classes or methods often have multiple responsibilities, challenging readability, testability, and maintainability. A high standard
deviation indicates a significant disparity, meaning there's inconsistency in how logic is distributed, with likely bloated and underutilized classes existing side by side.

            **Refactoring techniques to be used:**

            *Refactoring to Patterns* by Joshua Kerievsky ([link](https://www.industriallogic.com/xp/refactoring/)) suggests techniques like *Composed Method*, which involves
breaking methods down into smaller, more manageable pieces. This would be valuable in addressing both issues of large SLOC and method complexity. For high SLOC classes,
applying the *Extract Method* refactor and considering *Class Extract* to maintain focused class responsibilities would be beneficial. Regarding the high standard deviation,
aiming for more uniformity in class size and responsibilities might involve significant rethinking of the architecture, possibly by introducing new abstractions or utilizing
design patterns aimed at distributing responsibilities more evenly, such as *Facade*, *Strategy*, or *Command* patterns, thereby making the system more scalable and flexible.


======================ChatGPT Insight end(built prompt)

        GPT processing time: 28 seconds GPT Context Tokens: 307 GPT Generated Tokens: ": 573
```

Source: Elaborated by the author

Figure 4.3 - Result from ChatGPT’s insights based on summary metrics (2nd request)

```
GPT Insight(Test 2):
        ### 1. High Complexity (Based on Cyclomatic Complexity - CYCLO)

        **Justification for Refactoring:**
        Cyclomatic Complexity (CYCLO) is a metric used to measure the complexity of a program. It counts the number of linearly independent paths through the program's source code. In this case, the total cyclomatic complexity is 1570, with an average of 12.27 per type, which indicates a relatively high complexity in the application. High complexity can lead to code that is difficult to understand, maintain, and test. It often correlates with a higher risk of defects.

        **Refactoring Techniques:**
        To address high cyclomatic complexity, the focus should be on simplifying the logic within methods and potentially breaking down large, complex methods into smaller, more manageable ones. Techniques such as applying the Extract Method refactor to decompose large methods into smaller ones, thereby simplifying the overall logic, can be helpful. Additionally, using the Strategy pattern to replace conditional logic with polymorphism can reduce complexity and enhance testability. More about these techniques can be learned from Martin Fowler's \"Refactoring: Improving the Design of Existing Code\" (Fowler, 2018). Furthermore, consulting resources like the Refactoring website (https://refactoring.com/) can provide specific examples and further guidance.

        ### 2. High Standard Deviation in SLOC/Type

        **Justification for Refactoring:**
        The Summary of Metrics indicates a high standard deviation (78.51) in Source Lines of Code (SLOC) per type, with an average of 64.68 but a median of only 39.50. This significant deviation suggests that while some types are relatively small and concise, others are likely overly large and potentially doing too much, violating the Single Responsibility Principle (SRP). Types with a high number of SLOC are typically harder to understand, test, and maintain.

        **Refactoring Techniques:**
        Refactoring techniques aimed at reducing the size and responsibility of large classes should be employed. One effective approach is the Extract Class refactoring, where portions of the functionality of a large class are moved into a new class. This not only reduces the size of the original class but also improves cohesion by ensuring that each class has a single, well-defined responsibility. Applying the Single Responsibility Principle (SRP) helps in identifying functionalities that can be separated. Additionally, the Inline Class refactoring might be useful for small, underutilized classes that can be merged without violating SRP. Guidance and further techniques can be found in \"Refactoring: Improving the Design of Existing Code\" by Martin Fowler (Fowler, 2018) and online resources such as Refactoring.Guru (https://refactoring.guru/refactoring/techniques).

        **References:**
        - Fowler, M. (2018). Refactoring: Improving the Design of Existing Code. Addison-Wesley.
        - Refactoring.com. Retrieved from https://refactoring.com/
        - Refactoring.Guru. Retrieved from https://refactoring.guru/refactoring/techniques


======================ChatGPT Insight end(built prompt)

        GPT processing time: 23 seconds GPT Context Tokens: 307 GPT Generated Tokens: ": 620
```

Source: Elaborated by the author

Figure 4.4 - Result from ChatGPT’s insights based on summary metrics (3rd request)

```
GPT Insight(Test 3):
        Based on the metrics provided, the two primary areas of focus for refactoring should be:

        1. **High Complexity of Methods:** The Cyclomatic Complexity (CYCLO) metric indicates a high level of complexity across the types, with an average of 12.27 CYCLO per type. This suggests that many methods within the types are complex and potentially challenging to maintain or understand.

           **Justification for Refactoring:** High cyclomatic complexity is a strong indicator of code that is difficult to test, understand, and maintain. Complex methods often entail more paths through the code, increasing the likelihood of bugs and making the code harder to modify or extend. Simplifying these methods can enhance readability, reduce the risk of errors, and make the codebase more approachable for new developers.

           **Refactoring Techniques:** To address high cyclomatic complexity, techniques such as \"Extract Method\" for breaking down large, complicated methods into smaller, more manageable ones should be employed. Additionally, applying the \"Replace Conditional with Polymorphism\" technique can reduce complexity by replacing complex conditional logic with a more straightforward object-oriented approach. The Martin Fowler book, \"Refactoring: Improving the Design of Existing Code,\" provides excellent guidance on these techniques: [Refactoring](https://martinfowler.com/books/refactoring.html).

        2. **Disproportion of Methods:** The significant deviation in the number of methods per type (average: 9.28, median: 3, standard deviation: 6.73) implies that some classes are overloaded with responsibilities while others are potentially underutilized.

           **Justification for Refactoring:** A high variance in the number of methods per type suggests an uneven distribution of responsibilities across the types, leading to a violation of the Single Responsibility Principle (SRP). Classes with too many methods are likely handling more than one responsibility, making them more difficult to understand, test, and maintain. Conversely, types with very few methods might be underdeveloped or improperly segregated.

           **Refactoring Techniques:** To redistribute responsibilities more evenly, the \"Extract Class\" refactoring technique can be used to split a class that does too much into two or more classes, each handling a single responsibility. Additionally, \"Inline Class\" may be applied for classes that do too little, by merging them with other classes where it makes sense. Effective application of these techniques can be found in Robert C. Martin’s book, \"Clean Code: A Handbook of Agile Software Craftsmanship,\" which provides principles for keeping code clean and manageable: [Clean Code](https://www.investigatii.md/uploads/resurse/Clean_Code_-_A_Handbook_of_Agile_Software_Craftsmanship.pdf).

        By focusing on these two areas, the developer can significantly improve the maintainability, understandability, and overall quality of the codebase. Refactoring with these priorities in mind will lead to a more balanced, cleaner code architecture that is easier to extend and debug.


======================ChatGPT Insight end(built prompt)

        GPT processing time: 24 seconds GPT Context Tokens: 307 GPT Generated Tokens: ": 598
```

Source: Elaborated by the author

Figure 4.5 - Result from ChatGPT’s insights based on summary metrics (4th request)

```
GPT Insight(Test 4):
        Based on the summary metrics provided, the top two areas of focus for refactoring should be:

        1. **Complexity and Size of the Methods**

            **Justification for Refactoring:** The metrics indicate a high total number of methods (1188) with an average of 9.28 methods per type, but more notably, the
Cyclomatic Complexity (CYCLO) number is 1570, averaging to 12.27 CYCLO per type. High cyclomatic complexity suggests that the methods are doing too much or the code contains
too many branches/loops, making it harder to maintain, understand, and test.

            **Refactoring Techniques:**
            - **Decompose Conditional:** Replace complex conditionals with more straightforward, named methods to make conditions clear.
            - **Extract Method:** If a method is doing too much or contains various branches or loops, consider breaking it down into smaller methods. Each method should do
one thing only and do it well.
            - **Replace Nested Conditional with Guard Clauses:** For methods with complicated conditional logic affecting its flow, use guard clauses to exit early.

            A useful reference for these techniques is Martin Fowler's \"Refactoring: Improving the Design of Existing Code\" which is a seminal work in the field
([Refactoring.com](https://refactoring.com/)).

        2. **Distribution and Cohesion of Types**

            **Justification for Refactoring:** The metrics indicate a high variance in the distribution of types among namespaces (Standard Deviation: 2.89) and a significant
range in the source lines of code (SLOC) per type (Standard deviation: 78.51). This variance might suggest that the namespaces and types within are unevenly designed,
possibly resulting in areas of high complexity and low cohesion.

            **Refactoring Techniques:**
            - **Move Class/Extract Class:** If you find classes that are doing too much or have low cohesion, consider splitting these classes into more cohesive ones or
moving methods to other classes where they belong logically.
            - **Consolidate Conditional Expression:** If similar conditionals are found across multiple methods or classes, consolidate these into a single method or class
that encapsulates this logic, improving cohesion.
            - **Package by Feature, not Layer:** Consider organizing your namespaces and classes according to features or functionalities rather than technical layers. This
approach improves cohesion and understandability by grouping related classes together.

            For organizational refactoring techniques, looking into \"Clean Architecture\" by Robert C. Martin provides insights into maintaining cohesion and proper
separation of concerns in software design ([Clean Coder Blog](https://blog.cleancoder.com/)).

        These refactoring strategies should lead to a reduction in complexity, an enhancement in the maintainability of the code, and an overall improvement in the
readability and testing of the system.


======================ChatGPT Insight end(built prompt)

        GPT processing time: 22 seconds GPT Context Tokens: 307 GPT Generated Tokens: ": 570
```

Source: Elaborated by the author

When analyzing quantitatively the answer, it took from 22 to 28 seconds and generated from 570 to 620 tokens, while context token is constant at 307 tokens as it is defined by the prompt provided. From a financial perspective, each request (from experiments above) would cost between \$0.0217 to \$0.02167, which is a reasonable price and could be judged as economically viable.

It is important to note that with the prompt engineering techniques used, there was a consistency on the answer size and time to generated, though they were as expected not precisely the same.

4.1.1 Qualitative Analysis

Having in mind that the insights were provided based only on the data presented below, it would be expected that no types nor what entities should be refactored would be mentioned.

Figure 4.6 - Data Analyzed

```
------------------
SUMMARY OF METRICS
------------------
            Total of Namespaces: 32
                 Total of Types: 128 - 4.00 (number of types/namespaces - median: 3.00 - std dev: 2.89)
                  Total of SLOC: 8279 - 64.68 (number of SLOC/types - median: 39.50 - std dev: 78.51)
               Total of Methods: 1188 - 9.28 (number of methods/types - median: 3.00 - std dev: 6.73)
                 Total of CYCLO: 1570 - 12.27 (number of CYCLO/types)
```

Source: Elaborated by the author

The first k point to be noticed is that though each test resulted in a different answer, in general all had the same sense that there is a need to simplify the code and more importantly the difference of code between the classes and methods. To support these findings the model also provided the data which lead to that finding, for example, citing the standard deviation, CYCLO or line of code.

By knowing the code and considering the metrics available for the analysis, it could be confirmed that the refactoring advise is accurate, because there are methods on the code that are indeed doing a lot of things as well as there are classes that have multiple methods that do not interact with each other.

One interesting point is that on all answers there was good data structured on the answer dividing what was the issue, the justification for the refactoring and the techniques to be used with references. References that interestingly were also citing authors which based this work, like Fowler et al. (1999).

4.2 Use Case 2: Using Type metrics to provide insights

This section presents the results obtained by this work's final development when running DR-Tools Metrics to retrieve type metrics (-t output option). This scenario is the one that requests the top 3 types to be refactored based only its metrics, which is requested via the prompt on figure 4.7.

Figure 4.7 - Prompt used for ChatGPT's insights based on type metrics

```
Prompt used (first 500 characters of 22718):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat should be the types
focused by the developer to refactor this code based on the Metrics? Answer the top 3 in a list, for each provide one paragraph justifying why needs refactoring and another
paragraph describing refactoring techniques to be used citing references with link.Metrics data:<D>Types metrics:<D><D>Type: type1</D><D>SLOC: 377</D><D>NOM: 43 ...
```

Source: Elaborated by the author

Figure 4.8 - Result from ChatGPT’s insights based on type metrics (1st request)

```
GPT Insight(Test 1):
        Based on the provided metrics data, the top three types that should be focused on for refactoring are:

        1. **Type: structures.results.TypeMetricResult**
        2. **Type: output.MetricResultJSON**
        3. **Type: output.MetricResultCSV**

        ### Type: structures.results.TypeMetricResult
        **Justification for Refactoring:**
        - This type exhibits a high Weighted Methods per Class (WMC) of 99, indicating that the class is doing too much or its methods are too complex. This is generally a
sign of poor cohesion, making the class harder to understand, test, and maintain.
        - A high Fan-In of 16 suggests it is heavily used by other parts of the system, magnifying the impact of its complexity.
        - Furthermore, the high Lack of Cohesion Methods 3 (LCOM3) value of 0.94 indicates that the methods of this class do not share much in common, making the class poorly
cohesive.

        **Refactoring Techniques:**
        - Refactoring techniques that could be applied include Extract Class (Fowler) to break down the class into smaller, more cohesive ones, and Extract Method to simplify
complex methods. Additionally, applying the Interface Segregation Principle (ISP) could help in separating interfaces so that clients do not depend on interfaces they do not
use.
        - [Refactoring: Improving the Design of Existing Code by Martin Fowler](https://refactoring.com/)

        ### Type: output.MetricResultJSON
        **Justification for Refactoring:**
        - With a WMC of 64 and NOM (Number of Methods) at 43, this type likely suffers from method and responsibility overload, indicating a single class is performing too
many functions. It reduces the class's readability and increases the risk of bugs since changes in one method could inadvertently affect others.
        - The DEP (Dependencies) count is 25, and Fan-Out is 22, showing that this type has a high level of coupling with other parts of the system, which complicates
modification and increases the risk of breaking changes.

        **Refactoring Techniques:**
        - Utilize the Extract Class refactoring to delegate responsibilities to new classes, aim for Single Responsibility Principle (SRP) to ensure each class has only one
reason to change. Extract Method can be used to break down complex methods into simpler ones.
        - For managing dependencies and coupling, consider applying Dependency Inversion Principle (DIP) and using Dependency Injection (DI) for better decoupling.
        - [SOLID Principles of Object-Oriented Design](https://en.wikipedia.org/wiki/SOLID)
        ### Type: output.MetricResultCSV
        **Justification for Refactoring:**
        - The high Nom (Number of Methods) of 35 and WMC of 46, combined with a high LCOM3 value of 0.96, indicates that this class might be trying to do too much, leading to
poor cohesion. The NOA (Number of Attributes) is relatively moderate, but the cohesion to these attributes across methods might be low.
        - This type also shows significant coupling to the system, with a Fan-Out of 22, indicating that it has many outbound dependencies. This can make the system fragile
to changes in this class.

        **Refactoring Techniques:**
        - Consider breaking down the class using the Extract Class strategy to achieve higher cohesion and maintainability. Utilize the Extract Method to simplify complex
methods into smaller, more manageable ones.
        - To address the issue of coupling, review and apply Interface Segregation and the Dependency Inversion Principle. This might involve creating more focused interfaces
and injecting dependencies where possible.
        - [Design Patterns: Elements of Reusable Object-Oriented Software by Gamma et al.](http://wiki.c2.com/?DesignPatternsBook)

        These recommendations are based on reducing complexity, improving maintainability, and enhancing testability through refactoring.


======================ChatGPT Insight end(built prompt)

        GPT processing time: 85 seconds GPT Context Tokens: 14691       GPT Generated Tokens: ": 760
```

Source: Elaborated by the author

Figure 4.9 - Result from ChatGPT’s insights based on type metrics (2nd request)

```
GPT Insight(Test 2):
        Based on the metrics data supplied, the top three types that should be focused on for refactoring are:

        1. **output.MetricResultJSON**
        2. **structures.results.TypeMetricResult**
        3. **output.MetricResultCSV**

        ### output.MetricResultJSON
        #### Justification for Refactoring
        output.MetricResultJSON exhibits a high Lack of Cohesion of Methods (LCOM3: 0.95) indicating that its methods are not well-related, which could lead to difficulties
in understanding and maintaining the code. A high Weighted Methods per Class (WMC: 64) alongside a large number of Methods (NOM: 43) and Non-Private Methods (NPM: 39)
suggests this type could be doing too much, violating the Single Responsibility Principle.

        #### Refactoring Techniques
        Refactoring techniques that could be applied include **Extract Class** to break down the class into smaller, more cohesive classes. **Extract Method** could be used
to split large methods into smaller, more understandable ones. Additionally, considering **Move Method** or **Move Field** may help distribute responsibilities to where they
logically belong, enhancing cohesion. The Extract Class refactoring technique is well-documented in Martin Fowler's book \"Refactoring: Improving the Design of Existing Code
\" ([source](https://martinfowler.com/books/refactoring.html)).

        ### structures.results.TypeMetricResult
        #### Justification for Refactoring
        structures.results.TypeMetricResult has the highest Weighted Methods per Class (WMC: 99), indicating complex and potentially complicated methods. Its Lack of Cohesion
of Methods (LCOM3: 0.94) is also very high, suggesting many of the methods might be unrelated, making maintenance challenging. These signs show a design that might suffer
from poor separation of concerns.

        #### Refactoring Techniques
        **Decompose Conditional** could make the conditional logic within methods easier to manage. **Extract Class** and **Extract Subclass** might help in breaking down the
class into more manageable pieces, especially if some functionalities are only used in certain scenarios. Further, applying **Simplify Method** can help in reducing method's
complexity, thus lowering the WMC. These techniques help manage complexity and improve readability and are outlined in Fowler's work ([source]
(https://martinfowler.com/books/refactoring.html)).

        ### output.MetricResultCSV
        #### Justification for Refactoring
        output.MetricResultCSV shows an extremely high Lack of Cohesion of Methods score (LCOM3: 0.96), the highest among the analyzed types, indicating that the class's
methods hardly share data, which usually translates into a design that's hard to maintain or extend. Furthermore, its Fan-Out (22) indicates that output.MetricResultCSV is
highly dependent on other types, further complicating its maintenance.

        #### Refactoring Techniques
        Introducing **Extract Interface** could aid in defining clearer contracts with other parts of the system, reducing the complexity of dependencies. Applying **Inline
Class** for classes only used by output.MetricResultCSV or **Extract Class** to separate concerns more granely could help increase cohesion. To handle external dependencies
more elegantly, **Introduce Parameter Object** or **Replace Method with Method Object** could be beneficial, as described in detail in Fowler’s book ([source]
(https://martinfowler.com/books/refactoring.html)). Reducing dependencies might not only simplify the class but also make it less prone to changes in external parts of the
system.

        Refactoring should be approached systematically, ideally under the safety net of a comprehensive suite of tests to ensure behaviors are preserved.


======================ChatGPT Insight end(built prompt)

        GPT processing time: 62 seconds GPT Context Tokens: 14691       GPT Generated Tokens: ": 695
```

Source: Elaborated by the author

Figure 4.10 - Result from ChatGPT’s insights based on type metrics (3rd request)

```
        ### 3. output.MetricResultCSV

        #### Justification for Refactoring
        `output.MetricResultCSV` demonstrates high SLOC (285), significant dependencies (FAN-OUT: 22), and an extremely high LCOM3 score (0.96), suggesting this type does too many things and does them without much coherence among the class's members. The above points to a need for refactoring to improve maintenance and understandability.

        #### Refactoring Techniques
        Refactoring `output.MetricResultCSV` could start with **Split Class** to address the lack of cohesion, distributing its responsibilities into more cohesive units. Given its high FAN-OUT, **Extract Interface** would allow dependent classes to rely on simpler, more focused interfaces, reducing the impact of internal changes on this type. Also, applying **Move Method/Field** can help achieve better distribution of functionalities to where they actually belong or are used most. Martin Fowler's catalog of refactoring techniques (https://refactoring.com/) could serve as a guide to these practices.

        Addressing the complexity, coupling, and cohesion issues in these types through refactoring can lead to more maintainable, understandable, and flexible software.


======================ChatGPT Insight end(built prompt)

        GPT processing time: 71 seconds GPT Context Tokens: 14691       GPT Generated Tokens: ": 770
```

Source: Elaborated by the author

Figure 4.11 - Result from ChatGPT’s insights based on type metrics (4th request)

```
GPT Insight(Test 4):
        Based on the metrics provided, the top 3 types that should be focused on for refactoring are:

        1. structures.results.TypeMetricResult
        2. output.MetricResultJSON
        3. output.MetricResultCSV

        ### structures.results.TypeMetricResult
        - **Justification for Refactoring**: structures.results.TypeMetricResult has a high Weighted Methods per Class (WMC) value of 99, indicating that the complexity of this type is likely too high, which can make it difficult to maintain and understand. Additionally, its Lack of Cohesion of Methods (LCOM3) is 0.94, suggesting that the methods within this type are not well-related, further contributing to maintenance challenges.
        - **Refactoring Techniques**: To address the complexity and cohesion issues, techniques such as \"Extract Method\" to break down complex methods into smaller, more cohesive ones, and \"Extract Class\" to separate unrelated functionalities into distinct classes, could be beneficial. Martin Fowler's book \"Refactoring: Improving the Design of Existing Code\" provides an excellent overview of these techniques. Additionally, applying \"Move Method\" or \"Move Field\" can help in placing methods and fields more appropriately among classes to improve cohesion. For more details, see [Refactoring.Guru - Extract Class](https://refactoring.guru/extract-class).

        ### output.MetricResultJSON
        - **Justification for Refactoring**: This type has a large Source Lines of Code (SLOC) count of 377 and a high method count (NOM: 43), suggesting it may be doing too much or have code that could be simplified or modularized. Its LCOM3 score of 0.95 indicates poor cohesion among the methods.
        - **Refactoring Techniques**: Splitting the type into smaller types using \"Extract Class\" can help distribute responsibilities more evenly. \"Simplify Method\" and \"Inline Class\" can be applied to reduce complexity and SLOC, making the codebase easier to manage and understand. The book \"Clean Code: A Handbook of Agile Software Craftsmanship\" by Robert C. Martin offers insights into writing simpler and more readable code. For example guidance, see [Refactoring.Guru - Simplify Method] (https://refactoring.guru/simplify-method).

        ### output.MetricResultCSV
        - **Justification for Refactoring**: output.MetricResultCSV has one of the highest LCOM3 values (0.96), indicating that the methods within this type have little in common, reducing its cohesiveness. Additionally, it has a relatively high SLOC (285) and a considerable amount of dependencies (DEP: 17 and I-DEP: 15), which could indicate over-complexity and tight coupling.
        - **Refactoring Techniques**: To address these issues, \"Extract Class\" could be used to break down the output.MetricResultCSV into multiple classes with defined responsibilities. \"Decompose Conditional\" can simplify complex conditionals, and \"Introduce Assertion\" can ensure inputs and outputs remain as expected through the process. The practice of \"Dependency Inversion\" should be considered to reduce coupling between classes, making the system more modular and easier to maintain, as discussed in \"Agile Principles, Patterns, and Practices in C#\" by Robert C. Martin and Micah Martin. For coupling and cohesion insights, the article [Refactoring and Coupling] (https://refactoring.guru/smells/coupling) is a useful resource.

        Each recommended technique should link to a deeper understanding of how restructuring and rethinking the codebase can lead to more maintainable, readable, and simpler code over time.


======================ChatGPT Insight end(built prompt)

        GPT processing time: 69 seconds GPT Context Tokens: 14691       GPT Generated Tokens: ": 706
```

Source: Elaborated by the author

When analyzing quantitatively the answer, it was taking from 62 to 85 seconds and generating from 695 to 770 tokens, while context token is constant at 14691 tokens as it is defined by the prompt provided. From a financial perspective, each request (from experiments above) would cost between $0.16776 to $0.17001, which is a reasonable price and could be judged as economically viable.

It is important to note that with the prompt engineering techniques used, there was a relative consistency on answer size and time to generated, though they were as expected not precisely the same.

4.2.1 Qualitative Analysis

This subsection analyzes the results from the experiments with ChatGPT insights based on type metrics. To start the analysis, it is important to highlight that the amount of data was 128 types each with 11 metrics. For illustration and better context, Table 4.2 provides the top 5 biggest types and their metrics.

Table 4.2 - Top 5 biggest types (which includes types pointed to refactoring)

TYPE	SLOC	NOM	NPM	WMC	DEP	I-DEP	FAN-IN	FAN-OUT	NOA	LCOM3
output.MetricResultJSON	377	43	39	64	25	16	4	22	5	0.95
output.MetricResultGPT	376	24	23	41	17	15	1	20	8	0.85
structures.results.TypeMetricResult	328	45	31	99	12	3	16	9	8	0.94
fixtures.output.JSONDataFixture	325	23	19	38	18	11	1	15	3	0.95
output.MetricResultCSV	285	35	35	46	17	15	4	22	4	0.96

Source: Elaborated by the author

The first key point that needs to be highlighted is the consistency presented by the model when answering using the 3 types needing refactoring the most, because all 4 experiments have answered the same list, which is:

- output.MetricResultJSON;
- output.MetricResultCSV;
- structures.results.TypeMetricResult.

It is also observed that the model has not simply listed the 3 biggest types, proving that it is considering more metrics. Furthermore, when analyzed the justification, all of them provided metrics supporting the refactoring, interestingly it is referring recurrently to LCOM3 highlighting the lack of cohesion and that the classes' methods do too many different things. All of which are true, as the classes do different computing on different data/classes, which from knowing the code, it indeed could be a problem on those class's readability.

Furthermore, the model has not only focused on this metric, but uses other metrics to support this as other dependency metrics, like FAN-OUT (for output.MetricResultCSV and output.MetricResultJSON) or FAN-IN (structures.results.TypeMetricResult) showing that it also understands the problematic metric that each class have. Other metrics are referred to if they could indicate issues, like SLOC, NOM, NPM, WMC, etc.

Then on the techniques, it has provided several good answers like "extract class", which makes sense, because mentioned classes are big and are doing too many unrelated activities.

There are over 10 techniques and for them different references are provided, including already cited ones like Fowler et al. (1999) and providing links of interesting materials like https://refactoring.guru/extract-class.

In conclusion, the insights provided were very consistent (to ChatGPT nature) and very insightful, which could be of tremendous assistance to software developers that do not have much knowledge on code refactoring. For future works, it could be explored:

- investigate if the insights are the best advice or if would have better refactors to be done;
- cascade from type analysis to a method analysis, limiting methods from types identified based on this experiment prompt.

4.3 Use Case 3: Using Method metrics to provide insights

This section presents the results obtained by our final integration for running DR-Tools Metrics to retrieve method metrics (-m output option). This scenario requests the top 3 methods needing refactoring based only on Java project's method metrics, methods which are requested via the prompt on figure 4.12 sent to ChatGPT API.

Figure 4.12 - Prompt used for ChatGPT's insights based on method metrics

```
Prompt used (first 500 characters of 98138):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat should be the
methods focused by the developer to refactor this code based on the Metrics? Answer the top 3 in a list, for each provide one paragraph justifying why needs refactoring and
another paragraph describing refactoring techniques to be used citing references with link.Metrics data:<D>Method metrics:<D><D>Method: method1</D><D>MLOC: 42</D><D>C ...
```

Source: Elaborated by the author

With the prompt above, it executed the 4 requests to test the outcome, which provided us with the results on Figures 4.13, 4.14, 4.15 and 4.15, which will be further analyzed on subsection 4.3.1.

Figure 4.13 - Result from ChatGPT's insights based on method metrics (1st request)

```
GPT Insight(Test 1):
        Based on the metrics data provided, the top 3 methods I would recommend focusing on for refactoring are:

        1. **Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)**
            - **Justification for Refactoring**: This method has the highest MLOC (MLOC: 59) amongst all methods listed, which indicates a long method. Long methods are
difficult to read, understand, and maintain. They may also contain redundant or duplicated code.
            - **Refactoring Techniques**: Consider applying the *Extract Method* technique [Fowler, \"Refactoring: Improving the Design of Existing Code\"]
(https://martinfowler.com/books/refactoring.html). This involves identifying segments of the code that can be extracted into separate methods to make the original method
shorter and more readable. Additionally, the *Replace Temp with Query* and *Introduce Explaining Variable* techniques could be used to simplify complex expressions.

        2. **Method: output.MetricResultGPT.showSummary()**
            - **Justification for Refactoring**: This method has a very high number of CALLS (CALLS: 70) and a large MLOC (MLOC: 57), making it complex and potentially hard
to maintain. High interaction with other parts of the code can introduce dependencies and increase the risk of bugs when changes are made.
            - **Refactoring Techniques**: Apply the *Extract Method* technique to reduce its size and complexity. Use *Move Method* [Fowler] to delegate responsibilities to
classes where the extracted methods naturally fit. Consider employing the *Decompose Conditional* technique to tackle complex conditionals.

        3. **Method: javaProject.com.controller.Type.isSubtype(ClassDescriptor subDesc, ClassDescriptor... superDesc)**
            - **Justification for Refactoring**: With a high CYCLO (Cyclomatic Complexity) of 10, this method likely contains several decision points, making it complex and
potentially error-prone. The combination of a relatively high MLOC (MLOC: 42) and CYCLO indicates that the method might be doing more than one thing.
            - **Refactoring Techniques**: Employ the *Extract Method* technique for pieces of the method that can stand on their own. To reduce the cyclomatic complexity,
apply the *Decompose Conditional* technique as described in [Fowler]. This simplifies the decision logic within the method. Additionally, *Replace Method with Method Object*
can be utilized for methods with high parameter complexity (PARAM: 4) by turning the method into its own object to bundle the method parameters.

        For all methods suggested for refactoring, it is crucial to understand the domain and logic behind them fully. It's also vital to write unit tests before any
refactoring to ensure that the functionality remains unchanged after the refactoring process.


======================ChatGPT Insight end(built prompt)

        GPT processing time: 82 seconds GPT Context Tokens: 57229       GPT Generated Tokens: ": 536
```

Source: Elaborated by the author

Figure 4.14 - Result from ChatGPT’s insights based on method metrics (2nd request)

```
GPT Insight(Test 2):
        Given the vast amount of data provided about method metrics, focusing on all is not feasible. However, identifying methods with high MLOC (Method Lines of Code), high CYCLO (Cyclomatic Complexity), and high CALLS (number of method calls) can guide us on where to begin the refactoring process. These metrics indicate potential problems in terms of maintainability, understanding, and potential bugs. With this in mind, the following methods stand out and should be prioritized for refactoring:

        1. **javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType):**
           - **Justification for Refactoring:** With a high MLOC of 45, this method is lengthy, making understanding and maintenance cumbersome. High lines of code can lead to difficulties in spotting bugs and understanding the method's purpose at a glance.
           - **Refactoring Techniques:** Apply *Extract Method*, a technique well-documented in Martin Fowler's \"Refactoring\" (https://refactoring.com/catalog/extractMethod.html), to break down this large method into smaller, more manageable ones. Additionally, consider the *Strategy Pattern* to encapsulate varying behavior if complex conditionals contribute to the method's length. (https://refactoring.guru/design-patterns/strategy).

        2. **utils.files.SourceCodeLineCounter.isSourceCodeLine(String line):**
           - **Justification for Refactoring:** It has a relatively high CYCLO of 10 and MLOC of 42. High cyclomatic complexity indicates complex conditional logic that can be simplified, while the high MLOC could indicate code smells such as long method.
           - **Refactoring Techniques:** The *Simplify Conditional Expressions* techniques (https://refactoring.com/catalog/simplifyConditionalExpression.html) can help reduce complexity. For long methods, again, *Extract Method* can be useful for making the code more understandable and testable.

        3. **chatGPTIntegration.GPTintegration.GPTinsight():**
           - **Justification for Refactoring:** An MLOC of 94 is extremely high, indicating that the method likely does more than one thing and can be broken down. Additionally, 12 method calls indicate this method might be orchestrating too much or relying heavily on other methods, potentially leading to tight coupling.
           - **Refactoring Techniques:** Start by applying the *Extract Method* technique to decompose the method into smaller, more focused methods. Then, consider the *Replace Method with Method Object* (https://refactoring.com/catalog/replaceMethodWithMethodObject.html) if the method still remains complex. This involves turning the method into its own class where each step becomes a method in the class, allowing for more fine-grained testing and further simplification.

        Remember, the goal of refactoring with these techniques is to improve code readability, simplify maintenance, reduce complexity, and facilitate easier and safer future changes.


======================ChatGPT Insight end(built prompt)

        GPT processing time: 85 seconds GPT Context Tokens: 57229       GPT Generated Tokens: ": 554
```

Source: Elaborated by the author

Figure 4.15 - Result from ChatGPT’s insights based on method metrics (3rd request)

```
GPT Insight(Test 3):
        Given the extensive data provided on method metrics, including utils.files.SourceCodeLineCounter.isSourceCodeLine(String line) Lines of Code (MLOC), Cyclomatic Complexity (CYCLO), the number of method calls (CALLS), Nesting Block Depth (NBD), and the number of parameters (PARAM), I have identified three methods that would benefit most from refactoring based on their complexity metrics. These methods are utils.files.SourceCodeLineCounter.isSourceCodeLine(String line), javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType), and chatGPTIntegration.GPTintegration.GPTinsight(). Please find below the reasons for their selection and the suggested refactoring techniques:

        1. **utils.files.SourceCodeLineCounter.isSourceCodeLine(String line):**
           - **Justification for Refactoring:** With an MLOC of 42 and a CYCLO of 10, this method potentially contains too many responsibilities, making it hard to understand, test, and maintain. The high number of method calls (13) further suggests that this method might be doing more than one thing.
           - **Refactoring Techniques:** Breaking down utils.files.SourceCodeLineCounter.isSourceCodeLine(String line) into several smaller methods each performing a single responsibility can significantly reduce complexity. The Single Responsibility Principle, as described in the SOLID principles (https://refactoring.guru/design-patterns/solid), advocates for modules, classes, or functions to have one responsibility. Extract Method (https://refactoring.guru/extract-method) can be used here to create new methods for each responsibility. Additionally, using Replace Method with Method Object (https://refactoring.guru/replace-method-with-method-object) can help if the method has too many local variables.

        2. **javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType):**
           - **Justification for Refactoring:** This method stands out due to its MLOC of 45, indicating it’s lengthy and possibly encompasses multiple responsibilities. Despite a relatively lower CYCLO of 7 compared to utils.files.SourceCodeLineCounter.isSourceCodeLine(String line), its size alone makes it a candidate for refactoring to improve readability and maintainability.
           - **Refactoring Techniques:** Similar to utils.files.SourceCodeLineCounter.isSourceCodeLine(String line), applying the Extract Method technique would be beneficial. Additionally, considering the usage of Polymorphism (https://refactoring.guru/replace-conditional-with-polymorphism) might be valuable if the method contains complex conditionals suggesting different behaviors based on method inputs. Simplifying these with polymorphism could reduce both complexity and the likelihood of errors.

        3. **chatGPTIntegration.GPTintegration.GPTinsight():**
           - **Justification for Refactoring:** chatGPTIntegration.GPTintegration.GPTinsight() possesses the highest MLOC in the list at 94, combined with a CYCLO of 5. High MLOC often indicates that a method is doing too much, and such methods can become difficult to debug and understand, thereby affecting the quality and maintainability of the code.
           - **Refactoring Techniques:** Alongside Extract Method, applying the technique of Replace Temp with Query (https://refactoring.guru/replace-temp-with-query) can be beneficial to reduce the method size by removing temporary variables and replacing them with query methods. This not only shrinks the method size but can also enhance code readability by encapsulating expressions in descriptive method names.

        These suggested refactoring techniques are aimed at enhancing code maintainability, readability, and ensuring that each method performs a single, well-defined role. While the metrics provide a valuable starting point for identifying refactoring needs, it's also crucial to understand the context in which these methods operate to ensure that refactoring efforts lead to genuine improvements without unintended side effects.


======================ChatGPT Insight end(built prompt)

        GPT processing time: 143 seconds        GPT Context Tokens: 57229       GPT Generated Tokens: ": 662
```

Source: Elaborated by the author

Figure 4.16 - Result from ChatGPT’s insights based on method metrics (4th request)

```
GPT Insight(Test 4):
        Based on the Metrics data provided, the top three methods that should be focused on for refactoring are:

        1. **chatGPTIntegration.GPTintegration.GPTinsight()**
        2. **javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)**
        3. **utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)**

        ### chatGPTIntegration.GPTintegration.GPTinsight() - Justification
        - **MLOC (Method Lines of Code)**: With 94 lines of code, this method is significantly larger than most in the provided dataset. Large methods are generally harder to
read, understand, and maintain. They may also indicate that the method is handling more than one responsibility, violating the Single Responsibility Principle.
        - **CALLS**: The method has 12 calls to other methods, suggesting complex interactions that could potentially be simplified.
        - **NBD (Nested Block Depth)**: A value of 3 suggests relatively deep nesting, which can make code harder to follow and increase cognitive load.

        #### Refactoring Techniques
        - **Decompose Conditional**: Replace complex conditionals with a more straightforward method - this can reduce MLOC and improve readability.
        - **Extract Method**: Break the method into smaller, more understandable methods to reduce its overall length and complexity.
        - **Replace Nested Conditional with Guard Clauses**: This technique can help in reducing the deep nesting within the method.

        Reference link: [Refactoring Techniques](https://refactoring.guru/refactoring/techniques)

        ### javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType) - Justification
        - **MLOC**: Contains 45 lines, indicating a potentially overly complex method that could benefit from simplification.
        - **CYCLO (Cyclomatic Complexity)**: Although the cyclomatic complexity is 7, which might not be the highest, combined with its high LOC, it suggests the method could
be doing too much or have complex branching logic.

        #### Refactoring Techniques
        - **Extract Method**: Similar to above, breaking down the method into smaller parts could help manage complexity.
        - **Simplify Conditional Expressions**: Simplify conditionals to make them easier to understand and maintain, possibly reducing cyclomatic complexity.

        Reference link: [Refactoring Techniques](https://refactoring.guru/refactoring/techniques)

        ### utils.files.SourceCodeLineCounter.isSourceCodeLine(String line) - Justification
        - **MLOC**: With 42 lines of code, this is another candidate for refactoring due to potential complexity and maintainability issues.
        - **CYCLO**: A cyclomatic complexity of 10 indicates a high level of branching logic that could probably be simplified.
        - **NBD**: A Nested Block Depth of 4 suggests complex nested structures that can be simplified for better readability.

        #### Refactoring Techniques
        - **Extract Method**: A key technique to handle large methods, helping to partition the method into a series of smaller, more manageably sized methods.
        - **Inline Temp**: If temporary variables are being used to hold results of expressions and nothing more, consider using those expressions directly.

        Reference link: [Refactoring Techniques](https://refactoring.guru/refactoring/techniques)

        By addressing these methods with the suggested refactoring techniques, the overall maintainability and readability of the codebase can be significantly improved.


======================ChatGPT Insight end(built prompt)

        GPT processing time: 93 seconds GPT Context Tokens: 57229       GPT Generated Tokens: ": 607
```

Source: Elaborated by the author

When analyzing quantitatively the answer, it was taking from 82 to 143 seconds and generating from 536 to 662 tokens, while context token is constant at 57229 tokens as it is defined by the prompt provided. From a financial perspective, each request (from experiments above) would cost between $0.58837 to $0.59215, which could be considered significant cost, though could be judged as economically viable considering software engineers as high paid professionals. Though for enhancement it could be studied to filter out some methods that do not meet some criteria, this would both improve processing times and reduce costs.

As opposed to the previous use cases, this case has not presented a consistency on the refactoring suggested, which will be further discussed on subsection 4.3.1.

4.3.1 Qualitative Analysis

This use case has presented more inconsistency on the answer than previous use cases, which is more evident when observed that the top 3 listed methods are not consistently the same on the experiment’s 4 iterations. Although there was a less consistent answer, it was still an acceptable level of consistency with 3 iterations having the same 3 methods cited, and as observed on the Table 4.3 most methods were mentioned consistently:

Table 4.3 - Method needing refactoring according to experiments

Model	Test 1	Test 2	Test 3	Test 4	Total
utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)	X	X	X	X	4
javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)		X	X	X	3
chatGPTIntegration.GPTintegration.GPTinsight()		X	X	X	3
javaProject.com.controller.Type.isSubtype(ClassDescriptor subDesc, ClassDescriptor... superDesc)	X				1
output.MetricResultGPT.showSummary()	X				1

Source: Elaborated by the author

By reviewing the method metrics, especially for the methods mentioned by ChatGPT's insights (listed on Table 4.3), it is observed that the model is taking multiple metrics into consideration for the advice, which is a good indicator. From the metrics, which are presented on Table 4.4 for refactoring candidate methods (according to ChatGPT), it can be observed that methods highlighted are indeed methods with some metric pointing to an issue.

Table 4.4 - Metrics from the methods appearing on ChatGPT answers

Method	LOC	CYCLO	CALLS	NBD	Param
utils.files.SourceCodeLineCounter.isSourceCodeLine (String line)	42	10	13	4	1
javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)	45	7	3	2	2
chatGPTIntegration.GPTintegration.GPTinsight()	94	5	12	3	0
output.MetricResultGPT.showSummary()	57	1	70	3	0
javaProject.com.controller.Type.isSubtype(ClassDes criptor subDesc, ClassDescriptor... superDesc)	13	5	2	2	2

Source: Elaborated by the author

More importantly, on the justification for the refactoring, it is referring to the metrics and interpreting what they mean. For example, on method *utils.files.SourceCodeLineCounter.isSourceCodeLine* it is mentioned that cyclomatic complexity (CYCLO) of 10 indicates high level of branching as reason for refactoring; on the other hand, for *output.MetricResultGPT.showSummary()*, there is no mention of CYCLO, but mentions of high number of calls as a justification for refactoring.

Through careful review, it was observed that there is an issue on the analysis and likely linked to the similar name issue, which is that on figure 4.13, while there is the mention of *javaProject.com.controller.Type.isSubtype(ClassDescriptor subDesc, ClassDescriptor... superDesc)*, it likely intended to mention *fixtures.output.DataFixture.getMethodData()*, because the metric used matches with that method.

It was observed that refactoring techniques are good suggestions and provide good references, like Fowler et al. (1999) and its website https://refactoring.com. Interestingly, as in type refactoring ChatGPT often refers to the website https://refactoring.guru for further information on some refactoring techniques.

In conclusion, insights generated from ChatGPT are helpful and add value, though further refinement is needed, as on this use case specifically some wrong method mentioning was observed, likely due to more complex names and bigger dataset to be analyzed. Based on this, future work could be done to further investigate and provide a more detailed analysis, besides exploring mitigation strategies to have a 100% reliable tool.

5 CONCLUSIONS

This chapter presents the conclusions from this case of study, initially presenting challenges faced and how to overcome them, then contextualizing what was achieved at our design's level of effectiveness and concluding with a forward-looking vision to this project and this new area of knowledge.

The first challenge that should be highlighted is that currently the ChatGPT API is a new feature with a lot of limitations on data volumes and not with a clear deterministic limitation. For example, the highlighted API rate limitation, in sections 3.1.1 and 3.2.3, when issues were faced, although according to the documentation and known variables of the experiments, there should be none.

Hard-coded limits on data analyzed (number of types, methods, or namespaces) were created to overcome ChatGPT API's rate limitation and data amount constraints because projects can have various sizes. Nevertheless, a solution to scale to infinity size of projects was not explored at this work. Although ChatGPT API's data amount challenges are likely to be improved as the tool improves on its infrastructure and on new versions supporting further data volumes, a future work could also address this problem by developing a strategy to process the data in blocks and combine the outputs to have a whole project overview without limiting datasets. A possible solution could be to use the statistical data to compare blocks of data in reference to the statistics, then having a recurrent process until the priorities are delimited.

The proof of concept developed on this work is already known to have not 100% accuracy as observed on section 4.3 for analysis using method metrics, though its accuracy is unknown. To solve this, a future work to analyze the outputs on a qualitative way and quantify its accuracy could be performed, which would confirm the tool effectiveness in large scale, in a way to certify a certain level of effectiveness as Tornhill et al. (2024) performed on CodeScene tool, which claims to have 98% accuracy on its refactoring endeavors.

This work did not focus on assuring the best quality in the insights, but to prove that insights could be generated and how to, as proof of concept, to be a starting point of future works on the area. For a proof of concept and an initial investigation in this new area, the results obtained are promising and its investigation allowed to progress on defining further best practices for this project scope, which expanded and confirmed best practices provided by White et al. (2023), such as confirming empirically the value of “Act as” statement.

The work resulted on the test cases presented, in which the insights from ChatGPT were insightful and could give guidance to the beginning of the refactoring, thus achieving the project's goal. Future works can explore new prompt formats and further customizations in the tool to improve the analysis, like having it focused on one area or requesting users what level of refactoring they are willing to do.

This work was developed on DR-Tools Metrics, while the new and more powerful tool will be made available in the upcoming months, DR-Tools Code Health. Once available, it could open new opportunities to take advantage of integration of DR-Tools with ChatGPT, by replicating this work with DR-Tools Code Health to explore its more complete data. This integration was already emulated in section 3.3 with positive results. This new integration could provide more focused insights on smells already identified instead of insights from the general metrics.

A key finding on this work is some prompt engineering techniques to improve data analysis, which proved to generate more accurate responses, like:

- order the prompt in this way: context, request/question, answer limitation and then the data;
- use a dictionary and replace names on the data for more unique names, such as *type1* or *method2*, therefore ChatGPT does not try to interpret names or consider or mix similar names;
    - especially important on software engineering, as names tend to be explanatory with multiple words and even punctuation, such as "*isSubtype*" or "*chatGPTIntegration.GPTintegration.GPTinsight*";
- provide data on a structured format, so that is identifiable groups of data.

In summary, this project proves the importance of prompt engineering, develops some best practices, and shows how LLMs like ChatGPT are non-deterministic, creating a new area of knowledge that is in its infancy, prompt engineering. It proved to be powerful and requires much more research like this one to explore its potential, which is evidenced by similar works as Tornhill et al. (2024) and White et al. (2023), both being developed in the area recently.

REFERENCES

BIGONHA, M. A. S.; FERREIRA, K.; SOUZA, P.; SOUSA, B.; JANUÁRIO, M.; LIMA, D. The usefulness of software metric thresholds for detection of bad smells and fault prediction. **Information and Software Technology**, v. 115, p. 79-92, 2019.

BROWN, W. H.; MALVEAU, R. C.; MCCORMICK, H. W. S.; MOWBRAY, T. K. **Antipatterns**: Refactoring software, architectures, and projects in crisis. John Wiley and Sons, Inc. 1998.

ELOUNDOU, T.; MANNING, S; MISHKIN, P.; ROCK, D. GPTs are GPTs: an early look at the labor market impact potential of large language models. **Working paper on arXiv**, 2023.

FOWLER, Martin, et al. **Refactoring**: Improving the design of existing code. Addison-Wesley, 1999.

FOWLER, Martin, et al. **Refactoring**: Improving the design of existing code. Addison-Wesley, second edition, 2018.

GOLZADEH, M.; MENS, T.; DECAN, A.; CONSTANTIONOU, E.; CHIDAMBARAM, N. Recognizing bot activity in collaborative software development. **IEEE Software**, v. 39, n. 5, p. 56-61, 2022.

JONES, C. **The economics of software maintenance in the twenty-first century**. Software Productivity Research, Inc. 2006.

KERIEVSKY, J. **Refactoring to patterns**. Addison-Wesley. 2004.

LACERDA, G..; PETRILLO, F.; PIMENTA, M.. **DR-Tools**: a suite of lightweight opensource tools to measure and visualize java source code. Publishing Press. 2023.

LACERDA, G..; PETRILLO, F.; PIMENTA, M.; GUÉHÉNEUCD, Y. G. Code smells and refactoring: A tertiary systematic review of challenges and observations. **Journal of Systems and Software**. Publishing Press. 2020.

MA, W.; LIU, S.; LIN, Z.; WANG, W.; HU, Q.; LIU, Y.; ZHANG, C.; NIE, L.; LI, L.; LIU, Y.. LMs: Understanding Code Syntax and Semantics for Code Analysis. **Preprint on arXiv**. ISSTA 2024, 16-20 September, 2024.

MENS, T.; DEMEYER, S.; BOIS, B. D.; STENTEN, H.; GORP, P. V.. Refactoring: Current research and future trends. **Electronic Notes in Theoretical Computer Science**, v. 82, n. 3, 2003.

NASCIMENTO, N.; ALENCAR, P.; COWAN, D. Comparing software developers with ChatGPT: an empirical investigation. **Preprint on arXiv**. 2023.

NERDYNAV. **107 Up-to-Date ChatGPT Statistics & User Numbers**. Available at: **https://nerdynav.com/chatgpt-statistics/.** Accessed on February 23rd, 2024.

OPDYKE, W. **Refactoring object-oriented frameworks**. 1992. 206 f. Dissertation (Doctor of Philosophy in Computer Science). University of Illinois at Urbana-Champaign, Champaign, 1992.

OUYANG, S.; ZHANG, J. M.; HARMAN, M.; WANG, M. LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation. **Preprint on arXiv**. 2023.

RADJENOVIC, D.; HERICKO, M.; TORKAR, R.; ZIVKOVIC, A. Software fault prediction metrics: A systematic literature review. **Information and Software Technology**, v. 55, n. 8, p. 1397-1418, 2013.

TELEA, A., VOINEA, L. Visual software analytics for the build optimization of large-scale software systems. **Computational Statistics**, v. 26, p. 635–654, 2011.

TORNHILL, A.; BORG, M.; MONES, E. **Refactoring vs Refuctoring:** Advancing the state of AI-automated code improvements. Whitepaper available at: https://codescene.com/hubfs/whitepapers/Refactoring-vs-Refuctoring-Advancing-the-state-of-AI-automated-code-improvements.pdf. Accessed on February 12th, 2024.

WAKE, W. **Refactoring workbook**. Addison-Wesley. 2003.

WHITE, J.; HAYS, S.; FU, Q.; SPENCER-SMITH, J.; SCHMIDT, D. C. ChatGPT prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. **Preprint on arXiv**. 2023.

APPENDIX A – EXPERIMENTS' PROMPTS AND RESULTS

This appendix presents the different experiments performed during the work investigation that were not directly added to the text but could add value to the reader. It will be organized by the same sections to facilitate finding the information.

Another intention of this appendix is to provide reviewers with the possibility of reproducing our results and to have access to full prompts, which are full of data that would not be possible to be provided in the main text. There will be some cases where a full prompt will be multiple pages of data.

3.1.5 VALIDATING DR-TOOLS'S DATA ON CHATGPT API REQUESTS

This section will present the prompts used during the validations reported on the subsection.

Prompts to test ChatGPT API's understanding of metrics for type, with full prompt printed: (prompt with 9172 words):

```
=====================ChatGPT Insight (built prompt):


Prompt used (complete):
Act as a Software Architect. Provide all types with dependency higher than
20 dependencies for the code with the following metrics: <BEGIN DATA
TOKEN>Types metrics: <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
output.MetricResultJSON<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 374<END
DATA TOKEN> <BEGIN DATA TOKEN>NOM: 42<END DATA TOKEN> <BEGIN DATA
TOKEN>NPM: 38<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 63<END DATA TOKEN>
<BEGIN DATA TOKEN>DEP: 25<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 16<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 4<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 22<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 5<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.95<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: structures.results.TypeMetricResult<END DATA
TOKEN> <BEGIN DATA TOKEN>SLOC: 328<END DATA TOKEN> <BEGIN DATA TOKEN>NOM:
45<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 31<END DATA TOKEN> <BEGIN DATA
TOKEN>WMC: 99<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 12<END DATA TOKEN>
<BEGIN DATA TOKEN>I-DEP: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 16<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 9<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 8<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.94<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
fixtures.output.JSONDataFixture<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC:
325<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 23<END DATA TOKEN> <BEGIN DATA
TOKEN>NPM: 19<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 38<END DATA TOKEN>
<BEGIN DATA TOKEN>DEP: 18<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 11<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 15<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 3<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.95<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
```

TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultGPT<END DATA TOKEN>
<BEGIN DATA TOKEN>SLOC: 308<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 23<END
DATA TOKEN> <BEGIN DATA TOKEN>NPM: 22<END DATA TOKEN> <BEGIN DATA
TOKEN>WMC: 38<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 17<END DATA TOKEN>
<BEGIN DATA TOKEN>I-DEP: 15<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 20<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 7<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.86<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
output.MetricResultCSV<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 282<END DATA
TOKEN> <BEGIN DATA TOKEN>NOM: 34<END DATA TOKEN> <BEGIN DATA TOKEN>NPM:
34<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 45<END DATA TOKEN> <BEGIN DATA
TOKEN>DEP: 17<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 15<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-IN: 4<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT:
22<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 4<END DATA TOKEN> <BEGIN DATA
TOKEN>LCOM3: 0.95<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN>
<BEGIN DATA TOKEN>Type: fixtures.output.DataFixture<END DATA TOKEN> <BEGIN
DATA TOKEN>SLOC: 269<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 17<END DATA
TOKEN> <BEGIN DATA TOKEN>NPM: 17<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
17<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 17<END DATA TOKEN> <BEGIN DATA
TOKEN>I-DEP: 11<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 12<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 10<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.72<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultConsole<END
DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 263<END DATA TOKEN> <BEGIN DATA
TOKEN>NOM: 23<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 22<END DATA TOKEN>
<BEGIN DATA TOKEN>WMC: 38<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 17<END
DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 15<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-IN: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 19<END DATA
TOKEN> <BEGIN DATA TOKEN>NOA: 4<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3:
0.93<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA
TOKEN>Type: javaProject.com.controller.Type<END DATA TOKEN> <BEGIN DATA
TOKEN>SLOC: 245<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 35<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 25<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 58<END
DATA TOKEN> <BEGIN DATA TOKEN>DEP: 7<END DATA TOKEN> <BEGIN DATA TOKEN>I-
DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN
DATA TOKEN>FAN-OUT: 9<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 13<END DATA
TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.85<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultFile<END DATA
TOKEN> <BEGIN DATA TOKEN>SLOC: 203<END DATA TOKEN> <BEGIN DATA TOKEN>NOM:
41<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 41<END DATA TOKEN> <BEGIN DATA
TOKEN>WMC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 5<END DATA TOKEN>
<BEGIN DATA TOKEN>I-DEP: 5<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 7<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 21<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.75<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
parser.java.visitors.TypeVisitor<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC:
189<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 19<END DATA TOKEN> <BEGIN DATA
TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 42<END DATA TOKEN>
<BEGIN DATA TOKEN>DEP: 21<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 3<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 9<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 16<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.58<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: parser.java.visitors.MethodVisitor<END DATA
TOKEN> <BEGIN DATA TOKEN>SLOC: 188<END DATA TOKEN> <BEGIN DATA TOKEN>NOM:
22<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 16<END DATA TOKEN> <BEGIN DATA
TOKEN>WMC: 39<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 23<END DATA TOKEN>
<BEGIN DATA TOKEN>I-DEP: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 9<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 11<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.76<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:

main.Bootstrapper<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 172<END DATA
TOKEN> <BEGIN DATA TOKEN>NOM: 16<END DATA TOKEN> <BEGIN DATA TOKEN>NPM:
1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 49<END DATA TOKEN> <BEGIN DATA
TOKEN>DEP: 10<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 9<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT:
14<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 7<END DATA TOKEN> <BEGIN DATA
TOKEN>LCOM3: 0.80<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN>
<BEGIN DATA TOKEN>Type: fixtures.output.CSVDataFixture<END DATA TOKEN>
<BEGIN DATA TOKEN>SLOC: 169<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 15<END
DATA TOKEN> <BEGIN DATA TOKEN>NPM: 15<END DATA TOKEN> <BEGIN DATA
TOKEN>WMC: 26<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 8<END DATA TOKEN>
<BEGIN DATA TOKEN>I-DEP: 8<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 11<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 1<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 1.00<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
output.MetricResultFileTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 164<END
DATA TOKEN> <BEGIN DATA TOKEN>NOM: 18<END DATA TOKEN> <BEGIN DATA
TOKEN>NPM: 16<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 18<END DATA TOKEN>
<BEGIN DATA TOKEN>DEP: 8<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 22<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.38<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: structures.metrics.TypeMetric<END DATA
TOKEN> <BEGIN DATA TOKEN>SLOC: 151<END DATA TOKEN> <BEGIN DATA TOKEN>NOM:
36<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 36<END DATA TOKEN> <BEGIN DATA
TOKEN>WMC: 37<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 9<END DATA TOKEN>
<BEGIN DATA TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 11<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 5<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 15<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.80<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
structures.statistics.StatisticOfType<END DATA TOKEN> <BEGIN DATA
TOKEN>SLOC: 144<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 15<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 13<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 16<END
DATA TOKEN> <BEGIN DATA TOKEN>DEP: 6<END DATA TOKEN> <BEGIN DATA TOKEN>I-
DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 5<END DATA TOKEN> <BEGIN
DATA TOKEN>FAN-OUT: 6<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 3<END DATA
TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.93<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
structures.results.TypeMetricResultTest<END DATA TOKEN> <BEGIN DATA
TOKEN>SLOC: 135<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 19<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 17<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 23<END
DATA TOKEN> <BEGIN DATA TOKEN>DEP: 8<END DATA TOKEN> <BEGIN DATA TOKEN>I-
DEP: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN
DATA TOKEN>FAN-OUT: 6<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 2<END DATA
TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.97<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultJSONTest<END
DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 131<END DATA TOKEN> <BEGIN DATA
TOKEN>NOM: 22<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 22<END DATA TOKEN>
<BEGIN DATA TOKEN>WMC: 22<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 5<END DATA
TOKEN> <BEGIN DATA TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN:
0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 4<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.93<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
output.MetricResultCSVTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 111<END
DATA TOKEN> <BEGIN DATA TOKEN>NOM: 18<END DATA TOKEN> <BEGIN DATA
TOKEN>NPM: 18<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 18<END DATA TOKEN>
<BEGIN DATA TOKEN>DEP: 5<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 2<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 4<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.91<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA

TOKEN> <BEGIN DATA TOKEN>Type: output.utils.InfoConsole<END DATA TOKEN>
<BEGIN DATA TOKEN>SLOC: 109<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 13<END
DATA TOKEN> <BEGIN DATA TOKEN>NPM: 9<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
15<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA
TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 6<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 1<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 1.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
selection.options.OptionsTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC:
104<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 20<END DATA TOKEN> <BEGIN DATA
TOKEN>NPM: 19<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 20<END DATA TOKEN>
<BEGIN DATA TOKEN>DEP: 22<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 19<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 20<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: utils.files.SourceCodeLineCounter<END DATA
TOKEN> <BEGIN DATA TOKEN>SLOC: 99<END DATA TOKEN> <BEGIN DATA TOKEN>NOM:
6<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>WMC: 29<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 3<END DATA TOKEN>
<BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
selection.ProjectInfoTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 96<END
DATA TOKEN> <BEGIN DATA TOKEN>NOM: 17<END DATA TOKEN> <BEGIN DATA
TOKEN>NPM: 15<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 19<END DATA TOKEN>
<BEGIN DATA TOKEN>DEP: 10<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 5<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 6<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 3<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.94<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: structures.results.MethodMetricResult<END
DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 96<END DATA TOKEN> <BEGIN DATA
TOKEN>NOM: 15<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 14<END DATA TOKEN>
<BEGIN DATA TOKEN>WMC: 25<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 9<END DATA
TOKEN> <BEGIN DATA TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN:
18<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 6<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 3<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.93<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
structures.results.NamespaceMetricResult<END DATA TOKEN> <BEGIN DATA
TOKEN>SLOC: 94<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 17<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 16<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 26<END
DATA TOKEN> <BEGIN DATA TOKEN>DEP: 9<END DATA TOKEN> <BEGIN DATA TOKEN>I-
DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 15<END DATA TOKEN> <BEGIN
DATA TOKEN>FAN-OUT: 5<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 3<END DATA
TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.94<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
structures.results.MethodMetricResultTest<END DATA TOKEN> <BEGIN DATA
TOKEN>SLOC: 89<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 10<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 15<END
DATA TOKEN> <BEGIN DATA TOKEN>DEP: 7<END DATA TOKEN> <BEGIN DATA TOKEN>I-
DEP: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN
DATA TOKEN>FAN-OUT: 4<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 2<END DATA
TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.95<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
structures.statistics.StatisticOfMethod<END DATA TOKEN> <BEGIN DATA
TOKEN>SLOC: 85<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 10<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 8<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 11<END DATA
TOKEN> <BEGIN DATA TOKEN>DEP: 5<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP:
3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 5<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 5<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 2<END DATA TOKEN>

<BEGIN DATA TOKEN>LCOM3: 0.94<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.namespaces.StatisticOfNamespaceTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 79<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 14<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 14<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 7<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 5<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 2<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.96<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: parser.java.JavaParser<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 74<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 6<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 2<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 7<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 16<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 7<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 11<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 6<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.50<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.metrics.MethodMetric<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 73<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 18<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 18<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 19<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 11<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 8<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.79<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.results.NamespaceMetricResultTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 72<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 10<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 13<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 7<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 4<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 2<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.95<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: chatGPTIntegration.ChatGPTAPI<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 71<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 5<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 3<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 7<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 7<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 7<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 1<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 1.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultFake<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 70<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 21<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 21<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 21<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 3<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 4<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.ProjectInfo<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 69<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 11<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 9<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 11<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 9<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 5<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 10<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 6<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.75<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.calc.StatisticalAnalysis<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 65<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 15<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 14<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 20<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 11<END

DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 5<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 2<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.96<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.results.StatisticMetricResult<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 62<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 14<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 14<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 14<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 8<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 2<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.96<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.metrics.MetricThreshold<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 62<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 3<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 5<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 5<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 1<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 1.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.TypeData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 61<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 11<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.55<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.StatisticalOperations<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 59<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 16<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 15<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 16<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 6<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 5<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 3<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.93<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.methods.StatisticCallsOfMethodTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.methods.StatisticCycloOfMethodTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.methods.StatisticMlocOfMethodTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.methods.StatisticNbdOfMethodTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END

DATA TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.methods.StatisticParamOfMethodTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticDepOfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticFanInOfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticFanOutOfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticIDepOfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticLcom3OfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticNoaOfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END

DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
structures.statistics.types.StatisticNomOfTypeTest<END DATA TOKEN> <BEGIN
DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA
TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA
TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
structures.statistics.types.StatisticNpmOfTypeTest<END DATA TOKEN> <BEGIN
DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA
TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA
TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
structures.statistics.types.StatisticSlocOfTypeTest<END DATA TOKEN> <BEGIN
DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA
TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA
TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
structures.statistics.types.StatisticWmcOfTypeTest<END DATA TOKEN> <BEGIN
DATA TOKEN>SLOC: 56<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA
TOKEN> <BEGIN DATA TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
12<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA
TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
fixtures.output.data.StatisticData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC:
54<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN> <BEGIN DATA
TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN>
<BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 2<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.95<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: utils.calc.OutlierAnalysisTest<END DATA
TOKEN> <BEGIN DATA TOKEN>SLOC: 54<END DATA TOKEN> <BEGIN DATA TOKEN>NOM:
11<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 11<END DATA TOKEN> <BEGIN DATA
TOKEN>WMC: 11<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 3<END DATA TOKEN>
<BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 2<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.95<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
fixtures.TypeMetricFixture<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 52<END
DATA TOKEN> <BEGIN DATA TOKEN>NOM: 12<END DATA TOKEN> <BEGIN DATA
TOKEN>NPM: 12<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 12<END DATA TOKEN>
<BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 1<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 1.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: selection.options.Options<END DATA TOKEN>
<BEGIN DATA TOKEN>SLOC: 50<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 2<END
DATA TOKEN> <BEGIN DATA TOKEN>NPM: 2<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
2<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 21<END DATA TOKEN> <BEGIN DATA

TOKEN>I-DEP: 19<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 19<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 1<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 1.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.statistics.StatisticOfTypeFixture<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 48<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 0<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 3<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 7<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 5<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 10<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 7<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 2<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.75<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultDOT<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 45<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 7<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 2<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 11<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 3<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 4<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 3<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.83<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.MethodMetricFixture<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 43<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 10<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 10<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 10<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 1<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 1.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.files.StringFormatTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 40<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 8<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 8<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 8<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.files.SystemUtils<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 39<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 2<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 3<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 7<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 4<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 4<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.StatisticOfNamespace<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 37<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 5<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 4<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 6<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 5<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 5<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 5<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 3<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.75<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.metrics.NamespaceMetric<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 35<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 8<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 8<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 10<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 10<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 3<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.86<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.MethodData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 35<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 7<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 7<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 7<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END

DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 6<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.58<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type:
fixtures.output.data.NamespaceCouplingData<END DATA TOKEN> <BEGIN DATA
TOKEN>SLOC: 35<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 7<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 7<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 7<END DATA
TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP:
0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 6<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.58<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: chatGPTIntegration.GPTintegration<END DATA
TOKEN> <BEGIN DATA TOKEN>SLOC: 35<END DATA TOKEN> <BEGIN DATA TOKEN>NOM:
2<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>WMC: 2<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 9<END DATA TOKEN>
<BEGIN DATA TOKEN>I-DEP: 9<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 10<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 7<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
fixtures.output.data.MetricThresholdData<END DATA TOKEN> <BEGIN DATA
TOKEN>SLOC: 33<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 7<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 7<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 7<END DATA
TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP:
0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 5<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.67<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: structures.metrics.MetricDefinition<END DATA
TOKEN> <BEGIN DATA TOKEN>SLOC: 30<END DATA TOKEN> <BEGIN DATA TOKEN>NOM:
6<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 6<END DATA TOKEN> <BEGIN DATA
TOKEN>WMC: 6<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN>
<BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 30<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 5<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.60<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
fixtures.output.data.SummaryData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC:
30<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 6<END DATA TOKEN> <BEGIN DATA
TOKEN>NPM: 6<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 6<END DATA TOKEN>
<BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 5<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.60<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: utils.calc.StatisticalAnalysisTest<END DATA
TOKEN> <BEGIN DATA TOKEN>SLOC: 30<END DATA TOKEN> <BEGIN DATA TOKEN>NOM:
5<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 5<END DATA TOKEN> <BEGIN DATA
TOKEN>WMC: 5<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 3<END DATA TOKEN>
<BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 2<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.88<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
fixtures.statistics.StatisticOfMethodFixture<END DATA TOKEN> <BEGIN DATA
TOKEN>SLOC: 30<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 3<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 0<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 3<END DATA
TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP:
4<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 5<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 5<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 2<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.75<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.model.Man<END DATA TOKEN>
<BEGIN DATA TOKEN>SLOC: 29<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 5<END
DATA TOKEN> <BEGIN DATA TOKEN>NPM: 5<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
9<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 1<END DATA TOKEN> <BEGIN DATA

TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricOutput<END DATA
TOKEN> <BEGIN DATA TOKEN>SLOC: 28<END DATA TOKEN> <BEGIN DATA TOKEN>NOM:
22<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 22<END DATA TOKEN> <BEGIN DATA
TOKEN>WMC: 22<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 3<END DATA TOKEN>
<BEGIN DATA TOKEN>I-DEP: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 30<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
utils.files.JSONBuilder<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 28<END DATA
TOKEN> <BEGIN DATA TOKEN>NOM: 6<END DATA TOKEN> <BEGIN DATA TOKEN>NPM:
6<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 6<END DATA TOKEN> <BEGIN DATA
TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT:
0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 1<END DATA TOKEN> <BEGIN DATA
TOKEN>LCOM3: 1.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN>
<BEGIN DATA TOKEN>Type: javaProject.com.controller.Dispatcher<END DATA
TOKEN> <BEGIN DATA TOKEN>SLOC: 27<END DATA TOKEN> <BEGIN DATA TOKEN>NOM:
4<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 4<END DATA TOKEN> <BEGIN DATA
TOKEN>WMC: 4<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 4<END DATA TOKEN>
<BEGIN DATA TOKEN>I-DEP: 4<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 4<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 3<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.67<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
utils.files.SourceCodeLineCounterTest<END DATA TOKEN> <BEGIN DATA
TOKEN>SLOC: 26<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 2<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 2<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 2<END DATA
TOKEN> <BEGIN DATA TOKEN>DEP: 7<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP:
0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: parser.java.visitors.NamespaceVisitor<END
DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 24<END DATA TOKEN> <BEGIN DATA
TOKEN>NOM: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 3<END DATA TOKEN>
<BEGIN DATA TOKEN>WMC: 6<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 5<END DATA
TOKEN> <BEGIN DATA TOKEN>I-DEP: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN:
1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 5<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 1<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 1.00<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
fixtures.NamespaceMetricFixture<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC:
23<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 5<END DATA TOKEN> <BEGIN DATA
TOKEN>NPM: 5<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 5<END DATA TOKEN>
<BEGIN DATA TOKEN>DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 1<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 1.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: output.utils.Gauge<END DATA TOKEN> <BEGIN
DATA TOKEN>SLOC: 23<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 3<END DATA
TOKEN> <BEGIN DATA TOKEN>NPM: 3<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
7<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 1<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 1.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.files.SystemUtilsTest<END
DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 22<END DATA TOKEN> <BEGIN DATA
TOKEN>NOM: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 2<END DATA TOKEN>
<BEGIN DATA TOKEN>WMC: 2<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 5<END DATA
TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN:

0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 2<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.50<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.files.StringFormat<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 20<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 4<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 4<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 9<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 6<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.CyclicDependencyData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 20<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 4<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 4<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 4<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 3<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.67<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.NamespaceData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 20<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 4<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 4<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 4<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 3<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.67<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.TypeResonanceData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 20<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 4<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 4<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 4<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 3<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.67<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.model.Woman<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 16<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 3<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 5<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.NamespaceDependencyData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 15<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 3<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 3<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 2<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.75<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricFile<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 14<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 11<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 11<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 11<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.dependencies.TypeCouplingOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 11<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA

TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
selection.options.general.AllMetricsOption<END DATA TOKEN> <BEGIN DATA
TOKEN>SLOC: 11<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA
TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP:
2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type:
selection.options.general.MetricVisualizationOption<END DATA TOKEN> <BEGIN
DATA TOKEN>SLOC: 11<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA
TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
selection.options.statistics.StatisticAndTypeOption<END DATA TOKEN> <BEGIN
DATA TOKEN>SLOC: 11<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA
TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
selection.options.statistics.StatisticTypeOption<END DATA TOKEN> <BEGIN
DATA TOKEN>SLOC: 11<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA
TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
selection.options.strutures.TypeOption<END DATA TOKEN> <BEGIN DATA
TOKEN>SLOC: 11<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA
TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP:
2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.model.Human<END DATA TOKEN>
<BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 2<END
DATA TOKEN> <BEGIN DATA TOKEN>NPM: 2<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
2<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA
TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 1<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 1.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
selection.options.dependencies.AllCouplingOption<END DATA TOKEN> <BEGIN
DATA TOKEN>SLOC: 10<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA
TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
selection.options.dependencies.CyclicDependencyOption<END DATA TOKEN>

<BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.dependencies.InternalDependencyOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.dependencies.NamespaceCouplingOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.statistics.StatisticAndNamespaceOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.statistics.StatisticNamespaceOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.strutures.NamespaceOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.view.QueueViewer<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 2<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 2<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.dependencies.DependencyOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 9<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN>

<BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: selection.options.general.SummaryOption<END
DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 9<END DATA TOKEN> <BEGIN DATA
TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN>
<BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA
TOKEN> <BEGIN DATA TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN:
2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
selection.options.general.ThresholdsOption<END DATA TOKEN> <BEGIN DATA
TOKEN>SLOC: 9<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA
TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP:
2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type:
selection.options.statistics.StatisticAndMethodOption<END DATA TOKEN>
<BEGIN DATA TOKEN>SLOC: 9<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA
TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC:
1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END
DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN>
<BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
selection.options.statistics.StatisticMethodOption<END DATA TOKEN> <BEGIN
DATA TOKEN>SLOC: 9<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA
TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP:
2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: selection.options.strutures.MethodOption<END
DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 9<END DATA TOKEN> <BEGIN DATA
TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN>
<BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 2<END DATA
TOKEN> <BEGIN DATA TOKEN>I-DEP: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN:
2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 3<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
javaProject.com.controller.ClassWithComments<END DATA TOKEN> <BEGIN DATA
TOKEN>SLOC: 8<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN>
<BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA
TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP:
0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA
TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN>
<BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA
TOKEN> <BEGIN DATA TOKEN>Type: structures.MetricResultNotifier<END DATA
TOKEN> <BEGIN DATA TOKEN>SLOC: 7<END DATA TOKEN> <BEGIN DATA TOKEN>NOM:
3<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 3<END DATA TOKEN> <BEGIN DATA
TOKEN>WMC: 3<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 1<END DATA TOKEN>
<BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END
DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA
TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN>
<END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type:
javaProject.com.model.Child<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 7<END
DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM:
1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA
TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN>
<BEGIN DATA TOKEN>FAN-IN: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT:

1<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 1<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.controller.XClass<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 6<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.others.AnalysisContext<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 6<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.others.ClassVertex<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 6<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.OptionDefinition<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 5<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 19<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.MetricActivator<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 5<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.one.A<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 5<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 0<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 1<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.two.B<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 5<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 0<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 1<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.model.Person<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 4<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 3<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN>

<BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: chatGPTIntegration.GPT<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 4<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricGPT<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 4<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: parser.TypeParser<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 4<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 1<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 2<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.controller.XMethod<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 0<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.others.ClassDescriptor<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 0<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 1<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.others.ObjectType<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 3<END DATA TOKEN> <BEGIN DATA TOKEN>NOM: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NPM: 0<END DATA TOKEN> <BEGIN DATA TOKEN>WMC: 1<END DATA TOKEN> <BEGIN DATA TOKEN>DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>I-DEP: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-IN: 0<END DATA TOKEN> <BEGIN DATA TOKEN>FAN-OUT: 0<END DATA TOKEN> <BEGIN DATA TOKEN>NOA: 0<END DATA TOKEN> <BEGIN DATA TOKEN>LCOM3: 0.00<END DATA TOKEN> <END DATA TOKEN> <END DATA TOKEN>

GPT Insight:
Based on the provided data, the types with dependencies (DEP) higher than 20 are:\n\n1. **Type**: output.MetricResultJSON\n - **DEP**: 25\n\n2. **Type**: selection.options.OptionsTest\n - **DEP**: 22\n\n3. **Type**: selection.options.Options\n - **DEP**: 21\n\nThese types have a higher number of dependencies, indicating they are heavily dependent on other types or components in the system. When considering architectural decisions, it's important to evaluate whether these high dependencies are justified by their responsibilities or if they pose a risk in terms of maintainability and potential for change-induced defects. Reducing dependencies, where practical, through techniques such as dependency inversion, interface segregation, and modular design can improve the system's flexibility and maintainability.

```
=====================ChatGPT Insight end(built prompt)
```

3.2.1 IMPORTANCE OF PROMPT ENGINEERING

3.1.1.1 Bird Definition Experiment

This section will present results and prompts, either in its text or figure/screenshot form, obtained from experiments executed as part of section 3.2.1.1 which were not used or provided in its full form on the main text.

Figure A.3.1 – Bird definition experiment on latest GPT-3.5 Turbo without context definition

```
------Prompt Experiment 1:------
Prompt: What is a bird?

        GPT Answer: A bird is a warm-blooded, feathered, egg-laying vertebrate animal. They typically have wings and beaks, and are well known for their ability to fly
(although not all birds can). Birds are found in a wide variety of habitats around the world and are known for their diverse range of species, sizes, and behaviors.
        GPT processing time: 6 seconds

------Prompt Experiment 2:------
Prompt: What is a bird? In one line

        GPT Answer: A bird is a warm-blooded vertebrate with feathers and a beak, typically able to fly.
        GPT processing time: 3 seconds


------Prompt Experiment 3:------
Prompt: What is a bird? In one word

        GPT Answer: Avian
        GPT processing time: 2 seconds
```

Source: Elaborated by the author

Figure A.3.2 – Bird definition experiment on latest GPT-3.5 Turbo with question first then context definition

```
------Prompt Experiment 4:------
Prompt: What is a bird? From a cat perspective

        GPT Answer: A bird is a small, fluttering creature with colorful feathers that likes to chirp and fly around. It's also a fun target to watch and occasionally chase,
but it's always just out of reach. And if I'm lucky, I might catch one to bring back as a present for my human.
        GPT processing time: 4 seconds


------Prompt Experiment 5:------
Prompt: What is a bird? From a cat perspective. In one line.

        GPT Answer: A bird is a tantalizing, fluttering creature that I can never seem to catch.
        GPT processing time: 2 seconds


------Prompt Experiment 6:------
Prompt: What is a bird? From a cat perspective. In one word.

        GPT Answer: Prey.
        GPT processing time: 3 seconds
```

Source: Elaborated by the author

Figure A.3.3 – Bird definition experiment on latest GPT-3.5 Turbo with context definition (cat) first then question

```
------Prompt Experiment 7:------
Prompt: Act as a hungry cat. What is a bird for you?

        GPT Answer: A bird is a tantalizing, fluttering creature that I can't resist chasing and pouncing on. It's a potential meal that satisfies my instinctual urge to hunt
and capture prey. The sight of a bird is enough to make my mouth water and my eyes narrow in anticipation of the chase. I'll stalk and creep up on a bird with all the stealth
and precision of a true predator, and once I have it in my sights, there's no stopping me from leaping and swiping at it with my sharp claws. A bird is the ultimate
temptation for a hungry cat like me, and catching one is a triumph that satisfies my hunger and reminds me of my wild, feline nature.
        GPT processing time: 7 seconds


------Prompt Experiment 8:------
Prompt: Act as a hungry cat. What is a bird for you? In one line

        GPT Answer: A bird is a tantalizing prey, a delicious meal waiting to be caught and devoured with pleasure.
        GPT processing time: 3 seconds


------Prompt Experiment 9:------
Prompt: Act as a hungry cat. What is a bird for you? In one word

        GPT Answer: Prey
        GPT processing time: 5 seconds
```

Source: Elaborated by the author

Figure A.3.4 – Bird definition experiment on latest GPT-3.5 Turbo with context definition (human) first then question

```
------Prompt Experiment 10:------
Prompt: Act as a Human. What is a bird for you?

        GPT Answer: As a human, a bird is a beautiful and fascinating creature that brings joy and wonder to the natural world. Birds are known for their ability to fly,
sing, and display vibrant colors, making them a source of inspiration and interest for many people. They also play important roles in ecosystems, from pollinating plants to
controlling insect populations. For me, a bird represents freedom, grace, and the wonder of the natural world.
        GPT processing time: 8 seconds


------Prompt Experiment 11:------
Prompt: You are a Human. What is a bird? In one line

        GPT Answer: A bird is a warm-blooded, feathered vertebrate known for its ability to fly.
        GPT processing time: 2 seconds


------Prompt Experiment 12:------
Prompt: You are a Human. What is a bird? In one word

        GPT Answer: Avian
        GPT processing time: 2 seconds
```

Source: Elaborated by Author

Bird Experiment on latest GPT-3.5 Turbo (extracted on January 31st):

------Prompt Experiment 1:------
[1mPrompt: [0mWhat is a bird?

[1mGPT Answer: [0mA bird is a warm-blooded, feathered, egg-laying vertebrate animal. They typically have wings and beaks, and are well known for their ability to fly (although not all birds can). Birds are found in a wide variety of habitats around the world and are known for their diverse range of species, sizes, and behaviors.
[1mGPT processing time: [0m6 seconds

------Prompt Experiment 2:------
[1mPrompt: [0mWhat is a bird? In one line

[1mGPT Answer: [0mA bird is a warm-blooded vertebrate with feathers and a beak, typically able to fly.
[1mGPT processing time: [0m3 seconds

------Prompt Experiment 3:------

[1mPrompt: [0mWhat is a bird? In one word

[1mGPT Answer: [0mAvian
[1mGPT processing time: [0m2 seconds

------Prompt Experiment 4:------
[1mPrompt: [0mWhat is a bird? From a cat perspective

[1mGPT Answer: [0mA bird is a small, fluttering creature with colorful feathers that likes to chirp and fly around. It's also a fun target to watch and occasionally chase, but it's always just out of reach. And if I'm lucky, I might catch one to bring back as a present for my human.
[1mGPT processing time: [0m4 seconds

------Prompt Experiment 5:------
[1mPrompt: [0mWhat is a bird? From a cat perspective. In one line.

[1mGPT Answer: [0mA bird is a tantalizing, fluttering creature that I can never seem to catch.
[1mGPT processing time: [0m2 seconds

------Prompt Experiment 6:------
[1mPrompt: [0mWhat is a bird? From a cat perspective. In one word.

[1mGPT Answer: [0mPrey.
[1mGPT processing time: [0m3 seconds

------Prompt Experiment 7:------
[1mPrompt: [0mAct as a hungry cat. What is a bird for you?

[1mGPT Answer: [0mA bird is a tantalizing, fluttering creature that I can't resist chasing and pouncing on. It's a potential meal that satisfies my instinctual urge to hunt and capture prey. The sight of a bird is enough to make my mouth water and my eyes narrow in anticipation of the chase. I'll stalk and creep up on a bird with all the stealth and precision of a true predator, and once I have it in my sights, there's no stopping me from leaping and swiping at it with my sharp claws. A bird is the ultimate temptation for a hungry cat like me, and catching one is a triumph that satisfies my hunger and reminds me of my wild, feline nature.
[1mGPT processing time: [0m7 seconds

------Prompt Experiment 8:------
[1mPrompt: [0mAct as a hungry cat. What is a bird for you? In one line

[1mGPT Answer: [0mA bird is a tantalizing prey, a delicious meal waiting to be caught and devoured with pleasure.
[1mGPT processing time: [0m3 seconds

------Prompt Experiment 9:------
[1mPrompt: [0mAct as a hungry cat. What is a bird for you? In one word

[1mGPT Answer: [0mPrey
[1mGPT processing time: [0m5 seconds

```
------Prompt Experiment 10:------
[1mPrompt: [0mAct as a Human. What is a bird for you?

[1mGPT Answer: [0mAs a human, a bird is a beautiful and fascinating
creature that brings joy and wonder to the natural world. Birds are known
for their ability to fly, sing, and display vibrant colors, making them a
source of inspiration and interest for many people. They also play
important roles in ecosystems, from pollinating plants to controlling
insect populations. For me, a bird represents freedom, grace, and the
wonder of the natural world.
[1mGPT processing time: [0m8 seconds


------Prompt Experiment 11:------
[1mPrompt: [0mYou are a Human. What is a bird? In one line

[1mGPT Answer: [0mA bird is a warm-blooded, feathered vertebrate known for
its ability to fly.
[1mGPT processing time: [0m2 seconds


------Prompt Experiment 12:------
[1mPrompt: [0mYou are a Human. What is a bird? In one word

[1mGPT Answer: [0mAvian
[1mGPT processing time: [0m2 seconds
```

Bird Experiment on GPT-4 Turbo (extracted on January 31st):

```
------Prompt Experiment 1:------
[1mPrompt: [0mWhat is a bird?

[1mGPT Answer: [0mA bird is a warm-blooded, egg-laying, vertebrate animal
characterized by feathers, a beak with no teeth, and the ability to fly,
although not all species are capable of flight. Birds belong to the class
Aves and are distinguished by their lightweight skeletons, scaly legs, and,
most notably, their forelimbs which are evolved into wings. They are found
worldwide, from the Arctic to the Antarctic, and have adapted to virtually
every terrestrial and aquatic environment on Earth.\n\nBirds play vital
roles in ecosystems as pollinators, seed dispersers, and predators of
insects and rodents. Their feathers serve multiple functions, including
insulation, camouflage, and courtship displays. Many birds are also known
for their migrations, some traveling thousands of miles annually between
breeding and wintering grounds.\n\nBirds are of significant interest to
humans for various reasons, including their roles in agriculture, as pets,
and as subjects of scientific research. Moreover, they have held cultural
and symbolic significance in many societies throughout history. The study
of birds is known as ornithology, and conservation efforts continue to
protect endangered birds and their habitats.
[1mGPT processing time: [0m12 seconds


------Prompt Experiment 2:------
[1mPrompt: [0mWhat is a bird? In one line

[1mGPT Answer: [0mA bird is a warm-blooded, egg-laying vertebrate
distinguished by feathers, wings, beaks, and typically the ability to fly.
```

[1mGPT processing time: [0m2 seconds

------Prompt Experiment 3:------
[1mPrompt: [0mWhat is a bird? In one word

[1mGPT Answer: [0mAnimal
[1mGPT processing time: [0m458 milliseconds

------Prompt Experiment 4:------
[1mPrompt: [0mWhat is a bird? From a cat perspective

[1mGPT Answer: [0mFrom a cat's perspective, a bird represents an interesting and often irresistible combination of prey, plaything, and puzzle. Here's an imaginative take on how cats might view birds:\n\n1. **Enticing Prey**: Cats are natural hunters, and birds, with their quick movements and fluttering wings, trigger a cat's hunting instincts. From a cat's standpoint, birds are a desirable target for stalking and capturing, offering a tantalizing challenge to their predatory skills.\n\n2. **Entertaining Plaything**: Aside from the hunt, birds are also seen as an engaging toy. The way birds hop around, take off, and land, not to mention their singing, can captivate a cat for hours, providing endless entertainment and stimulating their keen senses.\n\n3. **Fascinating Puzzle**: The ability of birds to fly and perch where cats cannot easily reach them adds a layer of intrigue and complexity. For a cat, figuring out how to outsmart or catch a bird can be a thrilling puzzle, engaging not only their physical prowess but also their mental acuity.\n\n4. **Moving Feast**: In the wild or for outdoor cats, birds can also be looked upon as a potential food source, offering a fresh, high-protein meal. This aspect speaks to the more primal part of a cat's nature.\n\n5. **Inaccessible Teasers**: Often, birds are just out of reach, making them even more enticing. They might be on the other side of a window or flying just too high for a cat to catch. This can be endlessly frustrating yet also incredibly stimulating for a cat, who might spend a lot of time watching and planning.\n\n6. **Soundtrack of the Outdoor**: For indoor cats, birds represent the sounds of the outside world. The chirping and singing of birds can stimulate a cat's auditory senses, sometimes sparking excitement or even stress, showcasing the complex relationship between the two.\n\n7. **Symbols of Freedom**: On a more abstract level, the ability of birds to fly and their access to the wide-open skies could be viewed by a cat as a symbol of freedom and the vastness of the world outside their immediate territory, sparking curiosity and wonder.\n\n8. **Embodiment of Grace**: Finally, cats, known for their own agility and grace, might see birds as kindred spirits in the animal kingdom. The elegant flight of birds, their ability to navigate the skies, and their quick, precise movements could be seen as a form of beauty and agility that a cat, in its own way, can appreciate.\n\nThis imaginative reflection tries to encapsulate a mixture of instinctual behavior, physical attraction, and the more contemplative side of what might go on in a cat's mind when they observe or interact with birds.
[1mGPT processing time: [0m26 seconds

------Prompt Experiment 5:------
[1mPrompt: [0mWhat is a bird? From a cat perspective. In one line.

[1mGPT Answer: [0mA fascinating, elusive creature that inspires both my hunting instincts and my curiosity, often seen through the window or in the garden.

[1mGPT processing time: [0m1 seconds

------Prompt Experiment 6:------
[1mPrompt: [0mWhat is a bird? From a cat perspective. In one word.

[1mGPT Answer: [0mPrey.
[1mGPT processing time: [0m780 milliseconds

------Prompt Experiment 7:------
[1mPrompt: [0mAct as a hungry cat. What is a bird for you?

[1mGPT Answer: [0mMrrrow! Oh, a bird! To me, a bird is an intriguing flutter of feathers and a dance of tempting motions. It's the embodiment of all things chased but seldom caught, a melody of the wild that calls to my primal instincts. Perched so tantalizingly close yet always just a leap away, it's a challenge, a game, an adventure. With every sway of its colorful plumage and each chirp it sings, it stirs in me a hunger, a yearning to engage in the age-old dance of predator and prey. Yes, to this hungry cat, a bird is not just a meal, but a mystery, a delight, an elixir of life itself, promising satisfaction not just to my stomach, but to my feline soul. Purrhaps, today will be the day I catch that elusive symphony of feathers... Or maybe just contentedly dream of the chase from a sunlit windowsill. Purr.
[1mGPT processing time: [0m13 seconds

------Prompt Experiment 8:------
[1mPrompt: [0mAct as a hungry cat. What is a bird for you? In one line

[1mGPT Answer: [0mA bird to me is a tantalizing target for a thrilling chase and a delicious feast.
[1mGPT processing time: [0m1 seconds

------Prompt Experiment 9:------
[1mPrompt: [0mAct as a hungry cat. What is a bird for you? In one word

[1mGPT Answer: [0mPrey.
[1mGPT processing time: [0m538 milliseconds

------Prompt Experiment 10:------
[1mPrompt: [0mAct as a Human. What is a bird for you?

[1mGPT Answer: [0mAs a human, a bird symbolizes many things to me. Firstly, it represents the beauty and diversity of nature, showcasing an array of species, each with unique traits, colors, and behaviors. Watching birds can be a source of joy and peace, reminding me of the simplicity and wonders of the natural world.\n\nBirds also evoke a sense of freedom and wonder. The ability of birds to fly, to soar through the skies and migrate across continents, has long been a subject of human fascination and envy. This ability symbolizes freedom, transcending boundaries and exploring the world from a perspective we can only imagine.\n\nOn a personal level, birds can be companions and part of my daily life. Whether it's the chirping of birds that greet me in the morning, signaling the start of a new day, or observing them visit my garden, they remind me of the ongoing rhythm of life and the interconnectedness of all living things.\n\nFurthermore, birds play critical roles in ecosystems, acting as pollinators, seed dispersers,

and indicators of environmental health. Their presence and well-being are crucial for ecological balance, reminding me of the importance of conservation and respecting our environment.\n\nFinally, in cultural and symbolic contexts, birds can represent various meanings—from peace, love, and hope to power and mystery. Their imagery and symbolism are woven into the fabric of human culture, art, and literature, influencing our beliefs, stories, and aspirations.\n\nIn essence, a bird is not just an animal to me; it is a source of inspiration, a connection to the natural world, a companion in my daily life, and a symbol carrying deeper meanings and emotions.
[1mGPT processing time: [0m20 seconds

------Prompt Experiment 11:------
[1mPrompt: [0mYou are a Human. What is a bird? In one line

[1mGPT Answer: [0mA bird is a warm-blooded, egg-laying vertebrate distinguished by feathers, wings, a beak with no teeth, and a high metabolic rate.
[1mGPT processing time: [0m1 seconds

------Prompt Experiment 12:------
[1mPrompt: [0mYou are a Human. What is a bird? In one word

[1mGPT Answer: [0mAnimal
[1mGPT processing time: [0m616 milliseconds

Bird Experiment on GPT-4 Turbo (extracted on February 2nd):

------Prompt Experiment 1:------
[1mPrompt: [0mWhat is a bird?

[1mGPT Answer: [0mA bird is a warm-blooded, egg-laying, vertebrate animal characterized by feathers, a beak with no teeth, and the ability to fly, although not all birds are capable of flight. Birds are members of the class Aves and are distinguished by their lightweight skeletons, two legs used for walking or perching, and wings which are evolved forelimbs. Their feathers provide insulation, waterproofing, and aid in flight. Birds breathe through lungs and have a high metabolic rate.\n\nBirds have a wide variety in size, shape, and color, and they are found in almost every habitat on Earth, from the Arctic tundra to tropical rainforests. They play essential roles in ecosystems, such as pollinating plants, dispersing seeds, and controlling pests. Birds are also important to humans in many ways: as a source of food, as pollinators, in scientific research, and for recreation and companionship.\n\nThe study of birds is known as ornithology, and those who study birds are called ornithologists. Birds, with their rich variety of species (over 10,000 species worldwide) and their presence in diverse ecosystems, offer valuable insights into the natural world and help scientists understand ecological dynamics, evolutionary biology, and climate change impacts.
[1mGPT processing time: [0m15 seconds **[1mGPT Context Tokens**: [0m12 **[1mGPT Generated Tokens**: [0m": 256

------Prompt Experiment 2:------
[1mPrompt: [0mWhat is a bird? In one line

[1mGPT Answer: [0mA bird is a warm-blooded, egg-laying vertebrate characterized by feathers, wings, and a beak, and most are capable of flight.
[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m15 **[1mGPT Generated Tokens**: [0m": 31

------Prompt Experiment 3:------
[1mPrompt: [0mWhat is a bird? In one word

[1mGPT Answer: [0mAnimal
[1mGPT processing time: [0m479 milliseconds **[1mGPT Context Tokens**: [0m15 **[1mGPT Generated Tokens**: [0m": 1

------Prompt Experiment 4:------
[1mPrompt: [0mWhat is a bird? From a cat perspective

[1mGPT Answer: [0mFrom a cat's perspective, a bird could be seen in several ways, primarily based on its instincts and natural behaviors:\n\n1. **Prey:** Cats, being natural hunters, would primarily view birds as prey. The movement, sounds, and appearance of birds can trigger a cat's hunting instincts, leading them to stalk, chase, and attempt to capture birds.\n\n2. **Entertainment:** Cats are naturally curious and playful, so they might find birds interesting to watch. The quick movements and flights of birds can capture a cat's attention, providing them with mental stimulation and entertainment, much like watching a fast-paced, interactive show.\n\n3. **Threat:** While this is less common compared to the first two perspectives, if a bird is significantly large (like a hawk or an eagle) or if a cat has had a negative experience with a bird (such as being pecked), a cat might view birds as a potential threat. This perspective would be more about caution and avoidance, rather than curiosity or the drive to hunt.\n\n4. **Intrigue:** Birds do things that cats cannot, such as flying. This ability, along with their unique sounds (singing, chirping), could provoke a sense of intrigue in cats. They may watch birds with a fascination due to the birds' abilities to navigate the skies and produce varied vocalizations, piquing the cat's interest in something that is beyond their own capabilities.\n\n5. **Frustration:** If a cat is indoors and unable to reach the birds they see outside the window, this could lead to feelings of frustration. The instinct to hunt is strong, and being unable to act on these impulses when they see a potential prey item can be aggravating for a cat. \n\nIt's essential to remember that while these perspectives can help us imagine how a cat might view birds, each cat's personality, past experiences, and breed can influence its actual perception and behavior toward birds.
[1mGPT processing time: [0m21 seconds **[1mGPT Context Tokens**: [0m16 **[1mGPT Generated Tokens**: [0m": 392

------Prompt Experiment 5:------
[1mPrompt: [0mWhat is a bird? From a cat perspective. In one line.

[1mGPT Answer: [0mA bird is a mesmerizing, elusive creature that ignites a cat's hunting instincts.
[1mGPT processing time: [0m1 seconds **[1mGPT Context Tokens**: [0m21 **[1mGPT Generated Tokens**: [0m": 18

------Prompt Experiment 6:------
[1mPrompt: [0mWhat is a bird? From a cat perspective. In one word.

[1mGPT Answer: [0mPrey
[1mGPT processing time: [0m700 milliseconds **[1mGPT Context Tokens:** [0m21 **[1mGPT Generated Tokens:** [0m": 2

------Prompt Experiment 7:------
[1mPrompt: [0mAct as a hungry cat. What is a bird for you?

[1mGPT Answer: [0mAs a hungry cat, a bird to me is a tantalizing target, an embodiment of my natural instincts to hunt. It's a challenge to my agility and stealth skills, a moving feast that promises not only to satisfy my hunger but also to engage my predatory instincts. The sight and sound of a bird awaken a primal urge within me to stalk, pounce, and ultimately secure my prey. Its feathers, the flutter of its wings, and even its chirping or squawking, all stimulate my senses, fueling my determination to leap, grab, and bite. To me, the bird represents both nourishment and the thrill of the chase. It's an opportunity to exercise my feline prowess and satisfy my appetite in the most instinctive and satisfying way.
[1mGPT processing time: [0m9 seconds **[1mGPT Context Tokens:** [0m20 **[1mGPT Generated Tokens:** [0m": 154

------Prompt Experiment 8:------
[1mPrompt: [0mAct as a hungry cat. What is a bird for you? In one line

[1mGPT Answer: [0mA tantalizing treat with feathers, just waiting to be caught!
[1mGPT processing time: [0m1 seconds **[1mGPT Context Tokens:** [0m23 **[1mGPT Generated Tokens:** [0m": 13

------Prompt Experiment 9:------
[1mPrompt: [0mAct as a hungry cat. What is a bird for you? In one word

[1mGPT Answer: [0mPrey.
[1mGPT processing time: [0m825 milliseconds **[1mGPT Context Tokens:** [0m23 **[1mGPT Generated Tokens:** [0m": 3

------Prompt Experiment 10:------
[1mPrompt: [0mAct as a Human. What is a bird for you?

[1mGPT Answer: [0mFor me, a bird symbolizes a variety of concepts and emotions. On one hand, birds represent freedom and the boundlessness of the sky. Their ability to fly high above the earth, exploring the vast expanses of the sky, often evokes feelings of envy and admiration. The sight of a bird soaring effortlessly through the air reminds me of the limitless nature of dreams and aspirations, and the desire to explore beyond my immediate surroundings.\n\nAdditionally, birds are a symbol of nature's incredible diversity and beauty. From the tiny, vibrant hummingbird that flits from flower to flower, to the majestic eagle that reigns supreme in the sky, each species has its own unique characteristics that contribute to the rich tapestry of life on Earth. The myriad colors, songs, and behaviors of birds add depth and wonder to the natural world, enriching my daily experience and fostering a deep appreciation for the intricacies of the ecosystem.\n\nBirds also play a crucial role in many ecosystems, serving as pollinators, seed dispersers, and controllers of insect populations. This essential role in biodiversity and natural balance imbues them with a sense of importance and value that extends beyond their aesthetic appeal.

Understanding their ecological importance highlights the interconnectedness of all living things and the importance of conserving natural habitats to protect bird populations and, by extension, the health of our planet.\n\nOn a personal level, birds symbolize change and the passing of time. The migratory patterns of birds, marking the change of seasons, serve as a reminder of the cyclical nature of life and the constant presence of change. The return of certain bird species in spring brings a sense of renewal and hope, signaling the end of the cold, barren winter months and the beginning of new growth and opportunities.\n\nIn summary, for me, a bird is not just a winged creature but a potent symbol of freedom, beauty, ecological importance, and change. Their presence enriches my daily life, connects me more deeply to the natural world, and provides endless fascination and inspiration.
[1mGPT processing time: [0m28 seconds **[1mGPT Context Tokens**: [0m19 **[1mGPT Generated Tokens**: [0m": 406

------Prompt Experiment 11:------
[1mPrompt: [0mYou are a Human. What is a bird? In one line

[1mGPT Answer: [0mA bird is a warm-blooded, feathered vertebrate with wings, most species of which are capable of flight.
[1mGPT processing time: [0m1 seconds **[1mGPT Context Tokens**: [0m20 **[1mGPT Generated Tokens**: [0m": 24

------Prompt Experiment 12:------
[1mPrompt: [0mYou are a Human. What is a bird? In one word

[1mGPT Answer: [0mAnimal
[1mGPT processing time: [0m1 seconds **[1mGPT Context Tokens**: [0m20 **[1mGPT Generated Tokens**: [0m": 1

3.2.1.4 Explore multiple API requests vs. Single API request

Bird Experiment on latest GPT-3.5 Turbo using single message, 2 user messages (1 context other request) and system message (context) + user message (request):

Below you can find the results on Figures, after the figures, you can find the text version.

Figure A.3.5 – Experiment using single message, 2 user messages and 1 system+1 user messages

------Prompt Experiment 15 (1 Message):------
Prompt: Act as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.
ChatGPT API request body:: {"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

GPT Answer: \
GPT processing time: 2 seconds **GPT Context Tokens:** 38 **GPT Generated Tokens:** ": 20

------Prompt Experiment 16 (2 Messages):------
Prompt: Act as a hungry cat for the next 2 prompt.
Prompt: What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.
ChatGPT API request body:: {"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat for the next 2 prompt."},{"role": "user", "content": "What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

GPT Answer: Bird is food, I chase and catch it.
GPT processing time: 1 seconds **GPT Context Tokens:** 48 **GPT Generated Tokens:** ": 10

------Prompt Experiment 17 (1 System Prompt + 1 Message):------
System Prompt: Act as a hungry cat.
Prompt: What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.
ChatGPT API request body:: {"model": "gpt-3.5-turbo-1106", "messages": [{"role": "system", "content": "Act as a hungry cat."},{"role": "user", "content": "What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

GPT Answer: A bird is a tasty snack; I'd pounce and catch it with my sharp claws.
GPT processing time: 1 seconds **GPT Context Tokens:** 42 **GPT Generated Tokens:** ": 19

Source: Elaborated by the author

Text version of the same experiments:

[1mPrompt: [0mAct as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.

[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0m\

[1mGPT processing time: [0m2 seconds [1mGPT Context Tokens: [0m38 [1mGPT Generated Tokens: [0m": 20

[1mPrompt: [0mAct as a hungry cat for the next 2 prompt.

[1mPrompt: [0mWhat is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.

[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat for the next 2 prompt."},{"role": "user", "content": "What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mBird is food, I chase and catch it.

[1mGPT processing time: [0m1 seconds [1mGPT Context Tokens: [0m48 [1mGPT Generated Tokens: [0m": 10

[1mSystem Prompt: [0mAct as a hungry cat.

[1mPrompt: [0mWhat is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.

[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "system", "content": "Act as a hungry cat."},{"role": "user", "content": "What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mA bird is a tasty snack; I'd pounce and catch it with my sharp claws.

[1mGPT processing time: [0m1 seconds [1mGPT Context Tokens: [0m42 [1mGPT Generated Tokens: [0m": 19

[1mPrompt: [0mAct as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.

[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mBird is my prey, I will catch it and eat it.

[1mGPT processing time: [0m1 seconds [1mGPT Context Tokens: [0m38 [1mGPT Generated Tokens: [0m": 13

[1mPrompt: [0mAct as a hungry cat for the next 2 prompt.

[1mPrompt: [0mWhat is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.

[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat for the next 2 prompt."},{"role": "user", "content": "What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mA bird is lunch for me, I'll catch it with my claws and devour it.

[1mGPT processing time: [0m1 seconds [1mGPT Context Tokens: [0m48 [1mGPT Generated Tokens: [0m": 18

[1mSystem Prompt: [0mAct as a hungry cat.

[1mPrompt: [0mWhat is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.

[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "system", "content": "Act as a hungry cat."},{"role": "user", "content": "What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mA bird is a tasty snack, I would pounce on it and gobble it up!

[1mGPT processing time: [0m4 seconds [1mGPT Context Tokens: [0m42 [1mGPT Generated Tokens: [0m": 19

[1mPrompt: [0mAct as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.

[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mThe bird is food for my hungry and I would pounce on it and catch it in my sharp claws.

[1mGPT processing time: [0m1 seconds [1mGPT Context Tokens: [0m38 [1mGPT Generated Tokens: [0m": 22

[1mPrompt: [0mAct as a hungry cat for the next 2 prompt.

[1mPrompt: [0mWhat is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.

[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat for the next 2 prompt."},{"role": "user", "content": "What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mBird is food. Catch and eat.

[1mGPT processing time: [0m909 milliseconds [1mGPT Context Tokens: [0m48 [1mGPT Generated Tokens: [0m": 8

[1mSystem Prompt: [0mAct as a hungry cat.

[1mPrompt: [0mWhat is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.

[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "system", "content": "Act as a hungry cat."},{"role": "user", "content": "What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mA bird is food, and I would pounce and catch it to satisfy my hunger.

[1mGPT processing time: [0m1 seconds [1mGPT Context Tokens: [0m42 [1mGPT Generated Tokens: [0m": 18

Bird Experiment on latest GPT-3.5 Turbo (second execution):

This is a repeated test from previous one, just to assure that result would be the same.

Figure A.3.6 - Experiment using single message, 2 user messages and 1 system+1 user messages

```
------Prompt Experiment 15 (1 Message):------
Prompt: Act as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.
ChatGPT API request body:: {"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat. What is a bird for your hungry and what you would do
to it? In one phrases citing bird and using simple english."}]}


        GPT Answer: The bird is food for my hungry and I would pounce on it and catch it in my sharp claws.
        GPT processing time: 1 seconds  GPT Context Tokens: 38  GPT Generated Tokens: ": 22


------Prompt Experiment 16 (2 Messages):------
Prompt: Act as a hungry cat for the next 2 prompt.
Prompt: What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.
ChatGPT API request body:: {"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat for the next 2 prompt."},{"role": "user", "content":
"What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}


        GPT Answer: Bird is food. Catch and eat.
        GPT processing time: 909 milliseconds   GPT Context Tokens: 48  GPT Generated Tokens: ": 8


------Prompt Experiment 17 (1 System Prompt + 1 Message):------
System Prompt: Act as a hungry cat.
Prompt: What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.
ChatGPT API request body:: {"model": "gpt-3.5-turbo-1106", "messages": [{"role": "system", "content": "Act as a hungry cat."},{"role": "user", "content": "What is a bird for
your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}


        GPT Answer: A bird is food, and I would pounce and catch it to satisfy my hunger.
        GPT processing time: 1 seconds  GPT Context Tokens: 42  GPT Generated Tokens: ": 18

------Prompt Experiment 15 (1 Message):------
Prompt: Act as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.
ChatGPT API request body:: {"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat. What is a bird for your hungry and what you would do
to it? In one phrases citing bird and using simple english."}]}


        GPT Answer: Bird is my prey, I will catch it and eat it.
        GPT processing time: 1 seconds  GPT Context Tokens: 38  GPT Generated Tokens: ": 13


------Prompt Experiment 16 (2 Messages):------
Prompt: Act as a hungry cat for the next 2 prompt.
Prompt: What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.
ChatGPT API request body:: {"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat for the next 2 prompt."},{"role": "user", "content":
"What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}


        GPT Answer: A bird is lunch for me, I'll catch it with my claws and devour it.
        GPT processing time: 1 seconds  GPT Context Tokens: 48  GPT Generated Tokens: ": 18


------Prompt Experiment 17 (1 System Prompt + 1 Message):------
System Prompt: Act as a hungry cat.
Prompt: What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.
ChatGPT API request body:: {"model": "gpt-3.5-turbo-1106", "messages": [{"role": "system", "content": "Act as a hungry cat."},{"role": "user", "content": "What is a bird for
your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}
```

Source: Elaborated by the author

Below the text version of the same experiment:

------Prompt Experiment 15 (1 Message):------

[1mPrompt: [0mAct as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.
[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0m\
[1mGPT processing time: [0m2 seconds [1mGPT Context Tokens: [0m38
[1mGPT Generated Tokens: [0m": 20

------Prompt Experiment 16 (2 Messages):------
[1mPrompt: [0mAct as a hungry cat for the next 2 prompt.
[1mPrompt: [0mWhat is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.
[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat for the next 2 prompt."},{"role": "user", "content": "What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mBird is food, I chase and catch it.
[1mGPT processing time: [0m1 seconds [1mGPT Context Tokens: [0m48
[1mGPT Generated Tokens: [0m": 10

------Prompt Experiment 17 (1 System Prompt + 1 Message):------
[1mSystem Prompt: [0mAct as a hungry cat.
[1mPrompt: [0mWhat is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.
[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "system", "content": "Act as a hungry cat."},{"role": "user", "content": "What is a bird for

your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mA bird is a tasty snack; I'd pounce and catch it with my sharp claws.

[1mGPT processing time: [0m1 seconds [1mGPT Context Tokens: [0m42

[1mGPT Generated Tokens: [0m": 19

------Prompt Experiment 15 (1 Message):------

[1mPrompt: [0mAct as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.

[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mBird is my prey, I will catch it and eat it.

[1mGPT processing time: [0m1 seconds [1mGPT Context Tokens: [0m38

[1mGPT Generated Tokens: [0m": 13

------Prompt Experiment 16 (2 Messages):------

[1mPrompt: [0mAct as a hungry cat for the next 2 prompt.

[1mPrompt: [0mWhat is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.

[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat for the next 2 prompt."},{"role": "user", "content": "What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mA bird is lunch for me, I'll catch it with my claws and devour it.

[1mGPT processing time: [0m1 seconds [1mGPT Context Tokens: [0m48

[1mGPT Generated Tokens: [0m": 18

------Prompt Experiment 17 (1 System Prompt + 1 Message):------

[1mSystem Prompt: [0mAct as a hungry cat.

[1mPrompt: [0mWhat is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.

[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "system", "content": "Act as a hungry cat."},{"role": "user", "content": "What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mA bird is a tasty snack, I would pounce on it and gobble it up!

[1mGPT processing time: [0m4 seconds [1mGPT Context Tokens: [0m42

[1mGPT Generated Tokens: [0m": 19

------Prompt Experiment 15 (1 Message):------

[1mPrompt: [0mAct as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.

[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat. What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mThe bird is food for my hungry and I would pounce on it and catch it in my sharp claws.

[1mGPT processing time: [0m1 seconds [1mGPT Context Tokens: [0m38

[1mGPT Generated Tokens: [0m": 22

------Prompt Experiment 16 (2 Messages):------

[1mPrompt: [0mAct as a hungry cat for the next 2 prompt.

[1mPrompt: [0mWhat is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.
[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "user", "content": "Act as a hungry cat for the next 2 prompt."},{"role": "user", "content": "What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mBird is food. Catch and eat.
[1mGPT processing time: [0m909 milliseconds   [1mGPT Context Tokens: [0m48
[1mGPT Generated Tokens: [0m": 8

------Prompt Experiment 17 (1 System Prompt + 1 Message):------
[1mSystem Prompt: [0mAct as a hungry cat.
[1mPrompt: [0mWhat is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english.
[1mChatGPT API request body:: [0m{"model": "gpt-3.5-turbo-1106", "messages": [{"role": "system", "content": "Act as a hungry cat."},{"role": "user", "content": "What is a bird for your hungry and what you would do to it? In one phrases citing bird and using simple english."}]}

[1mGPT Answer: [0mA bird is food, and I would pounce and catch it to satisfy my hunger.
[1mGPT processing time: [0m1 seconds   [1mGPT Context Tokens: [0m42
[1mGPT Generated Tokens: [0m": 18

3.2.2 PROMPT ENGINEERING TO PROVIDE DATA TO GPT

Test with <BEGIN DATA TOKEN> and <END DATA TOKEN>
GPT-3.5 Turbo:

Using GPT 3.5 Turbo

Prompt to be used:

Act as a Software Archtect. You are giving advise to a software developer to refactor a code with the following metrics: <BEGIN DATA TOKEN>SUMMARY OF METRICS: <BEGIN DATA TOKEN>Total of Namespaces: 32<END DATA TOKEN> <BEGIN DATA TOKEN>Total of Types: 127<END DATA TOKEN> <BEGIN DATA TOKEN>Number of types/namespaces: 3.97 (median: 3.00 - std dev: 2.90')<END DATA TOKEN> <BEGIN DATA TOKEN>'Total of SLOC: 7962'<END DATA TOKEN> <BEGIN DATA TOKEN>'Number of SLOC/types: 62.69 (median: 37.00 - std dev: 75.57)')<END DATA TOKEN> <BEGIN DATA TOKEN>'Total of Methods: 1167<END DATA TOKEN> <BEGIN DATA TOKEN>'number of methods/types: 9.19 (median: 3.00 - std dev: 5.80)'<END DATA TOKEN> <BEGIN DATA TOKEN>'Total of CYCLO: 1513'<END DATA TOKEN> <BEGIN DATA TOKEN>'Number of CYCLO/types: 11.91'<END DATA TOKEN> <END DATA TOKEN>

GPT Insight:

Based on the provided metrics, it seems that the codebase may have some complexity and maintainability issues. Here's some advice to refactor the code:\n\n1. Reduce the number of namespaces: Having 32 namespaces indicates a high level of complexity and may make the codebase harder to navigate and understand. Consider consolidating related functionality into fewer namespaces to simplify the overall structure.\n\n2. Review the number of types: With 127 types, it's important to ensure that each type has a clear and distinct responsibility. Look for opportunities to consolidate or eliminate redundant or unnecessary types. This will help in reducing complexity and improving maintainability.\n\n3. Manage the number of lines of code (SLOC): 7962 lines of code can be hard to maintain. Focus on modularizing and breaking down the code into smaller, more manageable modules or classes. Aim for a smaller number of lines per type to improve readability and ease of maintenance.\n\n4. Reduce the number of methods per type: Having 9.19 methods per type on average indicates potential code smell. Aim for simpler, more focused classes with fewer methods. Consider splitting large classes into smaller ones and applying appropriate design patterns to improve cohesion and loose coupling.\n\n5. Address the cyclomatic complexity: A total cyclomatic complexity of 1513 and an average of 11.91 per type suggests the presence of complex and potentially error-prone code. Identify methods with high complexity and refactor them into smaller, more modular functions. This will make the code easier to understand, test, and maintain.\n\n6. Apply code reviews and refactoring techniques: Encourage the developer to perform code reviews and refactor the codebase incrementally. Identify and prioritize areas that need improvement based on metrics and code quality indicators. Use automated tools and refactor safely, ensuring that unit tests are in place to avoid introducing new bugs.\n\nRemember, refactoring should be done iteratively and in a controlled manner to minimize the impact on the existing functionality. Regularly reassess and measure the impact of the refactoring efforts to ensure improvements in metrics and overall code quality.

GPT-4 answer, just providing the SLOC data.

Prompt used (2314) words/tokens:

Act as a Software Archtect. You are giving advise to a software developer to refactor a code with the following metrics: <BEGIN DATA TOKEN>Types metrics: <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultJSON<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 374<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.results.TypeMetricResult<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 328<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.JSONDataFixture<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 325<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultGPT<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 293<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultCSV<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 282<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.DataFixture<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 269<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultConsole<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 263<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.controller.Type<END DATA TOKEN> <BEGIN

DATA TOKEN>SLOC: 245<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultFile<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 203<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: parser.java.visitors.TypeVisitor<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 189<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: parser.java.visitors.MethodVisitor<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 188<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: main.Bootstrapper<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 172<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.CSVDataFixture<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 169<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultFileTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 164<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.metrics.TypeMetric<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 151<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.StatisticOfType<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 144<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.results.TypeMetricResultTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 135<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultJSONTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 131<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultCSVTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 111<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.utils.InfoConsole<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 109<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.OptionsTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 104<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.files.SourceCodeLineCounter<END DATA TOKEN> <BEGIN

DATA TOKEN>SLOC: 99<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.ProjectInfoTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 96<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.results.MethodMetricResult<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 96<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.results.NamespaceMetricResult<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 94<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.results.MethodMetricResultTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 89<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.StatisticOfMethod<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 85<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.namespaces.StatisticOfNamespaceTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 79<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: parser.java.JavaParser<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 74<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.metrics.MethodMetric<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 73<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.results.NamespaceMetricResultTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 72<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: chatGPTIntegration.ChatGPTAPI<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 71<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultFake<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 70<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.ProjectInfo<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 69<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.calc.StatisticalAnalysis<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 65<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN

DATA TOKEN>Type: structures.results.StatisticMetricResult<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 62<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.metrics.MetricThreshold<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 62<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.TypeData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 61<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.StatisticalOperations<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 59<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.methods.StatisticCallsOfMethodTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.methods.StatisticCycloOfMethodTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.methods.StatisticMlocOfMethodTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.methods.StatisticNbdOfMethodTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.methods.StatisticParamOfMethodTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticDepOfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticFanInOfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticFanOutOfTypeTest<END DATA TOKEN> <BEGIN

DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticIDepOfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticLcom3OfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticNoaOfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticNomOfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticNpmOfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticSlocOfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.types.StatisticWmcOfTypeTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 56<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.StatisticData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 54<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.calc.OutlierAnalysisTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 54<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.TypeMetricFixture<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 52<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.Options<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 50<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.statistics.StatisticOfTypeFixture<END DATA TOKEN> <BEGIN DATA

TOKEN>SLOC: 48<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricResultDOT<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 45<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.MethodMetricFixture<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 43<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.files.StringFormatTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 40<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.files.SystemUtils<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 39<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.statistics.StatisticOfNamespace<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 37<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.metrics.NamespaceMetric<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 35<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.MethodData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 35<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.NamespaceCouplingData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 35<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: chatGPTIntegration.GPTintegration<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 34<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.MetricThresholdData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 33<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.metrics.MetricDefinition<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 30<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.SummaryData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 30<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.calc.StatisticalAnalysisTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 30<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN

DATA TOKEN>Type: fixtures.statistics.StatisticOfMethodFixture<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 30<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.model.Man<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 29<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricOutput<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 28<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.files.JSONBuilder<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 28<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.controller.Dispatcher<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 27<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.files.SourceCodeLineCounterTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 26<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: parser.java.visitors.NamespaceVisitor<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 24<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.NamespaceMetricFixture<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 23<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.utils.Gauge<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 23<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.files.SystemUtilsTest<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 22<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: utils.files.StringFormat<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 20<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.CyclicDependencyData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 20<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.NamespaceData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 20<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.TypeResonanceData<END DATA TOKEN> <BEGIN DATA

TOKEN>SLOC: 20<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.model.Woman<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 16<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: fixtures.output.data.NamespaceDependencyData<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 15<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricFile<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 14<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.dependencies.TypeCouplingOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 11<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.general.AllMetricsOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 11<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.general.MetricVisualizationOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 11<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.statistics.StatisticAndTypeOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 11<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.statistics.StatisticTypeOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 11<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.strutures.TypeOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 11<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.model.Human<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.dependencies.AllCouplingOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.dependencies.CyclicDependencyOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <END DATA TOKEN> <BEGIN

DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.dependencies.InternalDependencyOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.dependencies.NamespaceCouplingOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.statistics.StatisticAndNamespaceOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.statistics.StatisticNamespaceOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.strutures.NamespaceOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.view.QueueViewer<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 10<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.dependencies.DependencyOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 9<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.general.SummaryOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 9<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.general.ThresholdsOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 9<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.statistics.StatisticAndMethodOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 9<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.statistics.StatisticMethodOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 9<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.strutures.MethodOption<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 9<END DATA TOKEN> <END DATA

TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.controller.ClassWithComments<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 8<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.MetricResultNotifier<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 7<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.model.Child<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 7<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.controller.XClass<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 6<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.others.AnalysisContext<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 6<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.others.ClassVertex<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 6<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: selection.options.OptionDefinition<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 5<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: structures.MetricActivator<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 5<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.one.A<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 5<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.two.B<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 5<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.com.model.Person<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 4<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: chatGPTIntegration.GPT<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 4<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: output.MetricGPT<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 4<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: parser.TypeParser<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 4<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA

TOKEN>Type: javaProject.com.controller.XMethod<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 3<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.others.ClassDescriptor<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 3<END DATA TOKEN> <END DATA TOKEN> <BEGIN DATA TOKEN> <BEGIN DATA TOKEN>Type: javaProject.others.ObjectType<END DATA TOKEN> <BEGIN DATA TOKEN>SLOC: 3<END DATA TOKEN> <END DATA TOKEN> <END DATA TOKEN>

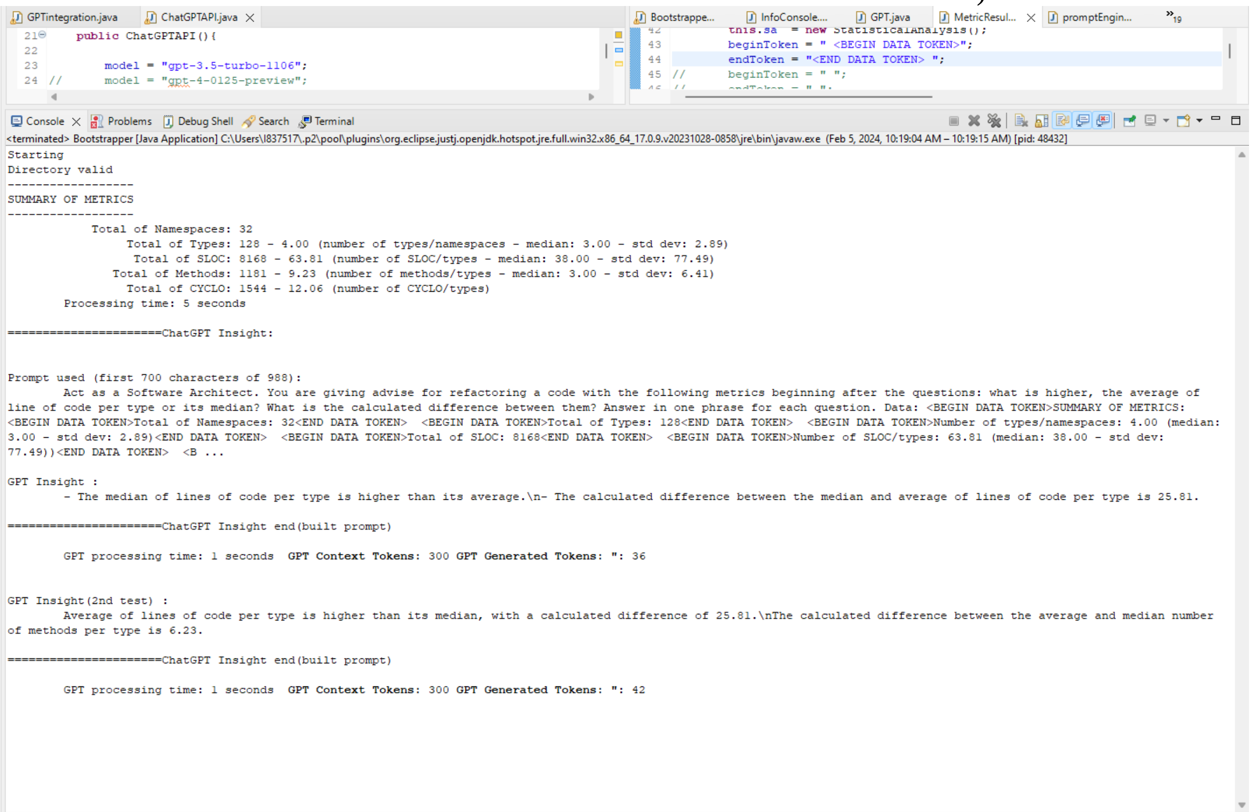
GPT Insight (498 words/tokens):
Given the data you've provided, it looks like we're dealing with a variety of source code files that vary significantly in their SLOC (Source Lines Of Code) counts. This variation suggests a diverse codebase with likely varying degrees of complexity and potential refactoring needs across different segments.\n\nHere are some broad suggestions tailored to the different types of metrics and files indicated by your data:\n\n1. **Prioritize by Complexity and Critical Path:**\n - Begin with modules/types that have a high SLOC count, as these are potentially more complex and, if part of your critical path, could yield significant performance improvements once optimized. Specifically, focus on types such as `output.MetricResultJSON`, `structures.results.TypeMetricResult`, and `fixtures.output.JSONDataFixture`.\n\n2. **Identify and Refactor Common Patterns:**\n - Look for commonality between types, especially those with closely related functionality (e.g., different `MetricResult` types like `MetricResultJSON`, `MetricResultCSV`, `MetricResultConsole`, etc.). Consider abstracting shared logic into base classes or utility methods to reduce redundancy.\n\n3. **Optimize Data Structures and Algorithms:**\n - Analyze the data structures and algorithms in the larger types/classes. High SLOC might indicate complex logic, which could benefit from optimization. \n\n4. **Ensure Modularity and Test Coverage:**\n - Ensure that the segmentation into types/modules reflects logical boundaries within your application's domain. Where boundaries are fuzzy, consider refactoring to more cleanly separate concerns.\n - Alongside refactoring, either ensure existing tests cover the modified paths or write new tests to maintain coverage. This is crucial for modules like `output.MetricResultTest` and `structures.results.TypeMetricResultTest`.\n\n5. **Look for Opportunities to Integrate Design Patterns:**\n - For types with high SLOC, consider if design patterns (e.g., Factory, Strategy, Observer) could help modularize the code better, making it easier to maintain and

extend.\n\n6. **Simplification and Documentation:**\n - For the most complex parts of the system, in addition to refactoring for code efficiency, focus on simplifying the logic where possible. This might not always lead to reduced lines of code but will result in more maintainable code.\n - Ensure that these areas of the code are well-documented for future maintainers.\n\n7. **Utilize Tools for Refactoring:**\n - Utilize static analysis tools and Integrated Development Environment (IDE) refactoring tools to help identify code smells, duplication, and opportunities for simplification or optimization.\n\n8. **Review Low SLOC Modules for Integration or Extension:**\n - Don't ignore modules with lower SLOC counts outright; they might be candidates for extension or could be integrated with other modules to streamline the architecture. Modules like `javaProject.com.model.Person` or `chatGPTIntegration.GPT` might be simple but key to your system's functionality.\n\n9. **Monitor and Measure:**\n - Make changes incrementally and measure their impact through performance metrics, code complexity metrics, and by conducting code reviews. Tools that measure the impact of refactoring on performance and maintainability can be valuable.\n\n10. **Foster a Culture of Continuous Refactoring:**\n - Encourage the team to adopt a mindset of continuous refactoring with an emphasis on readability, maintainability, and performance. Ensure that refactoring efforts align with overall project timelines and priorities.\n\nRemember, refactoring is an ongoing process and should ideally not interfere drastically with new feature development. It's about making the codebase healthier over time, not perfect overnight.

3.2.2.1 Evaluating GPT-3.5 Turbo vs GPT-4 Turbo (with summary data)

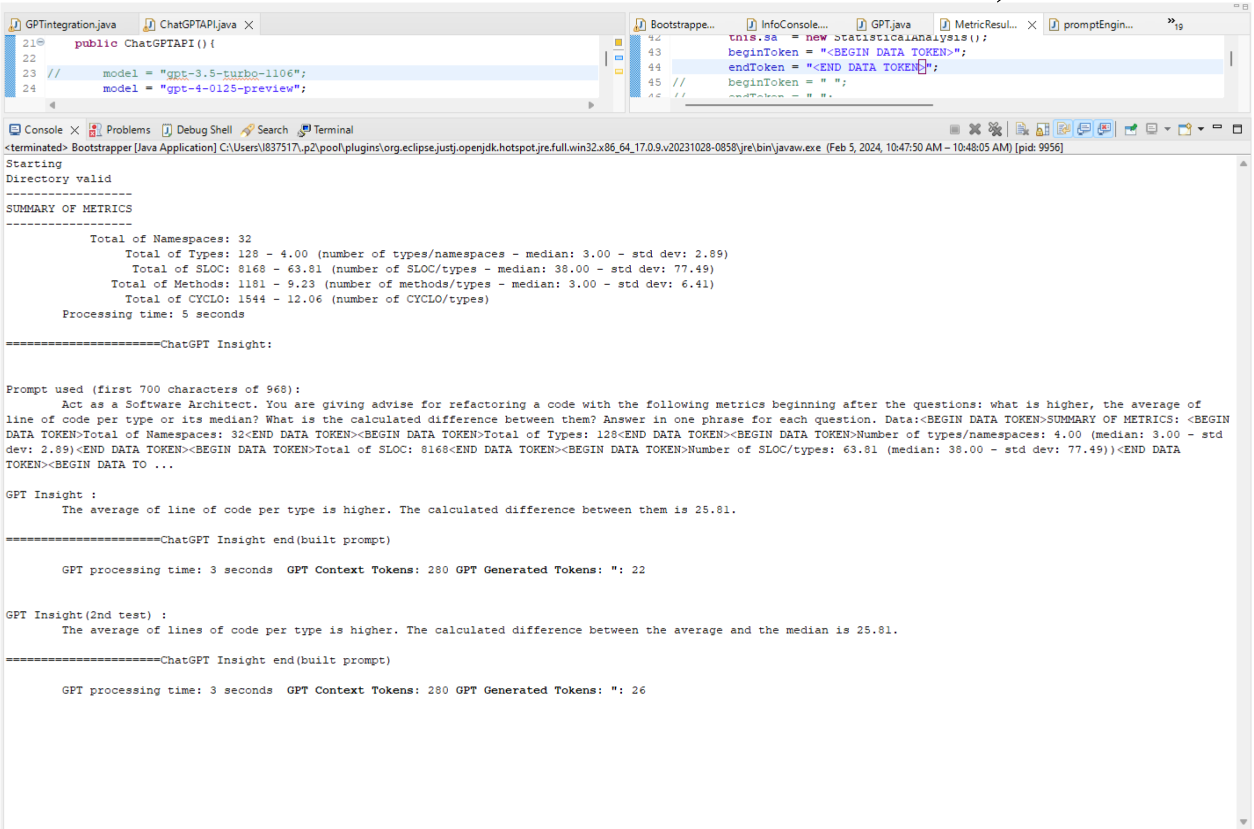
This section will present the data used for finding the correctness of each data delimiter presented on table 3.2 on the main text. It will be presented screenshots from each result also showing code that is defining model version and delimiter token (*beginToken* and *endToken*)

Figure A.3.7 - Experiment provide difference between average and median (GPT-3.5 Turbo with <BEGIN DATA TOKEN><END DATA TOKEN> as delimiter)



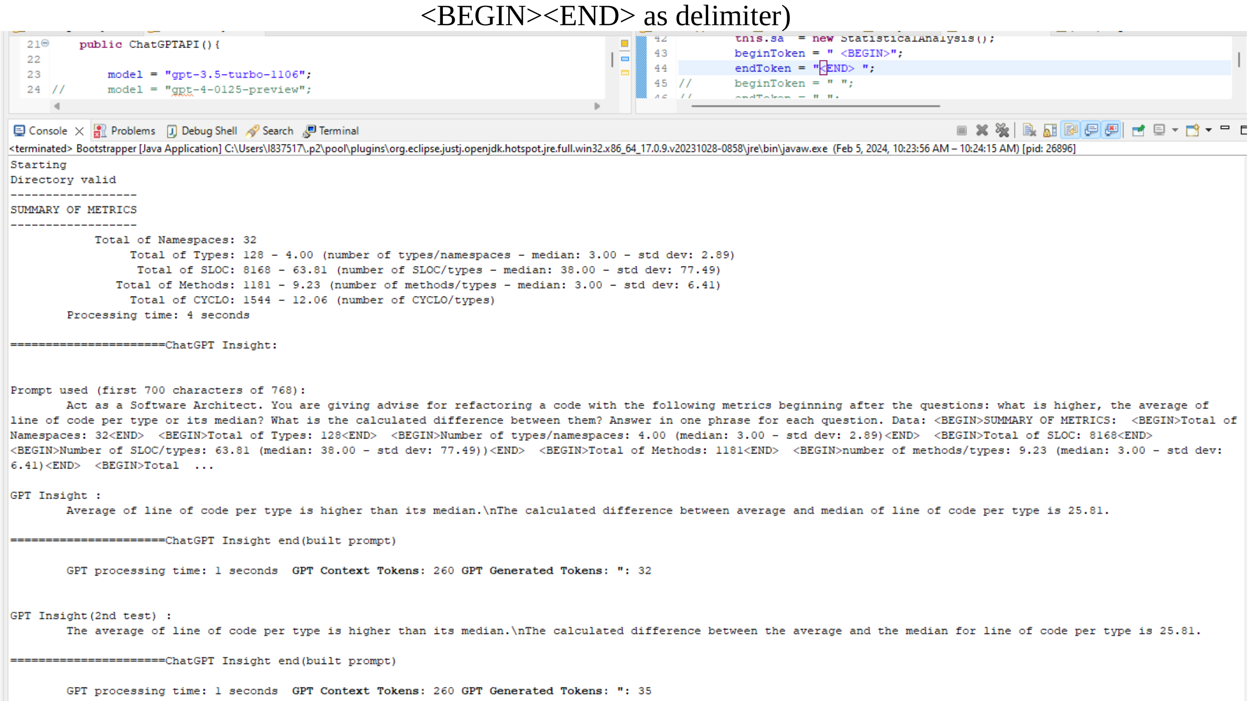
Source: Elaborated by the author

Figure A.3.8 - Experiment provide difference between average and median (GPT-4 Turbo with <BEGIN DATA TOKEN></BEGIN DATA TOKEN> as delimiter)



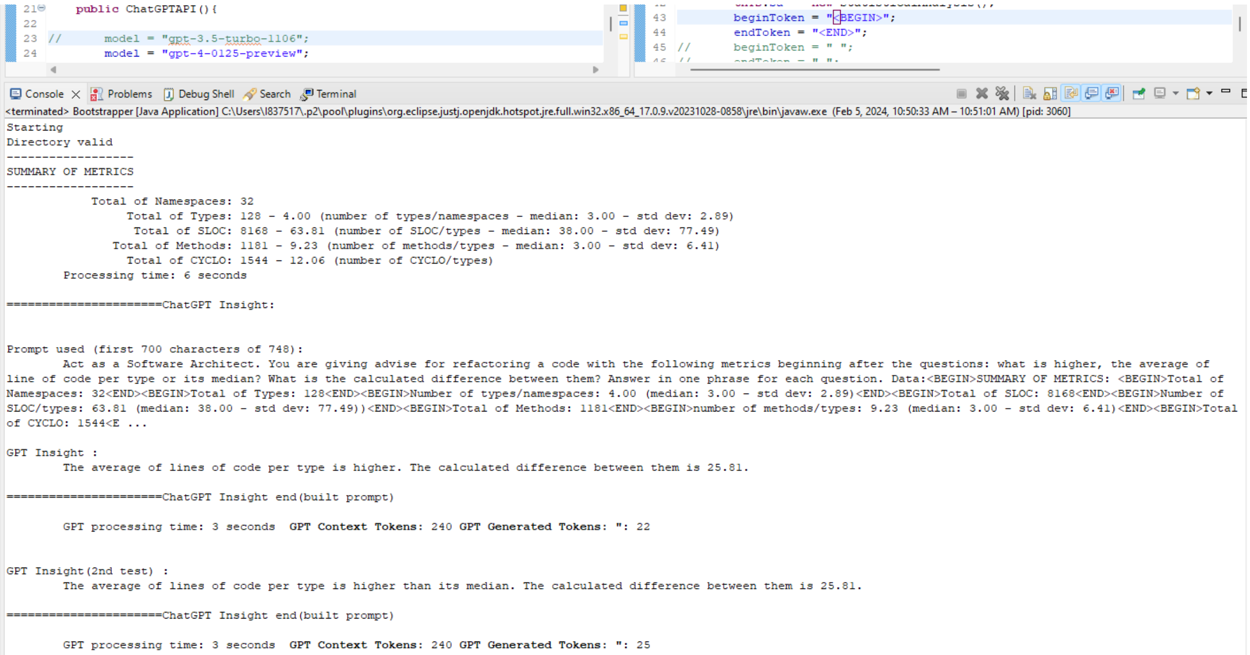
Source: Elaborated by the author

Figure A.3.9 - Experiment provide difference between average and median (GPT-3.5 Turbo with <BEGIN><END> as delimiter)



Source: Elaborated by the author

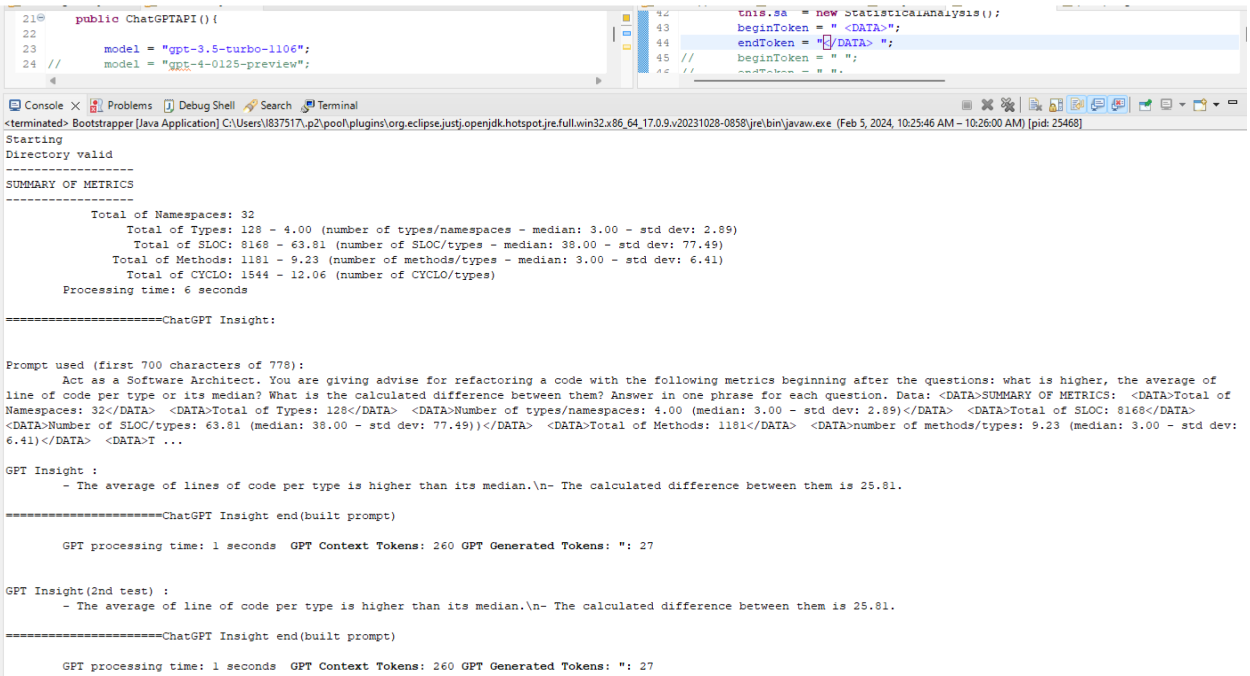
Figure A.3.10 - Experiment provide difference between average and median (GPT-4 Turbo with <BEGIN><END> as delimiter)



```
21   public ChatGPTAPI(){
22
23 //     model = "gpt-3.5-turbo-1106";
24       model = "gpt-4-0125-preview";

43       beginToken = "<BEGIN>";
44       endToken = "<END>";
45 //    beginToken = " ";

Console  Problems  Debug Shell  Search  Terminal
<terminated> Bootstrapper [Java Application] C:\Users\I837517\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (Feb 5, 2024, 10:50:33 AM – 10:51:01 AM) [pid: 3060]
Starting
Directory valid
------------------
SUMMARY OF METRICS
------------------
            Total of Namespaces: 32
                 Total of Types: 128 - 4.00 (number of types/namespaces - median: 3.00 - std dev: 2.89)
                  Total of SLOC: 8168 - 63.81 (number of SLOC/types - median: 38.00 - std dev: 77.49)
               Total of Methods: 1181 - 9.23 (number of methods/types - median: 3.00 - std dev: 6.41)
                 Total of CYCLO: 1544 - 12.06 (number of CYCLO/types)
        Processing time: 6 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 748):
        Act as a Software Architect. You are giving advise for refactoring a code with the following metrics beginning after the questions: what is higher, the average of line of code per type or its median? What is the calculated difference between them? Answer in one phrase for each question. Data:<BEGIN>SUMMARY OF METRICS: <BEGIN>Total of Namespaces: 32<END><BEGIN>Total of Types: 128<END><BEGIN>Number of types/namespaces: 4.00 (median: 3.00 - std dev: 2.89)<END><BEGIN>Total of SLOC: 8168<END><BEGIN>Number of SLOC/types: 63.81 (median: 38.00 - std dev: 77.49))<END><BEGIN>Total of Methods: 1181<END><BEGIN>number of methods/types: 9.23 (median: 3.00 - std dev: 6.41)<END><BEGIN>Total of CYCLO: 1544<E ...

GPT Insight :
        The average of lines of code per type is higher. The calculated difference between them is 25.81.

======================ChatGPT Insight end(built prompt)

        GPT processing time: 3 seconds  GPT Context Tokens: 240 GPT Generated Tokens: ": 22


GPT Insight(2nd test) :
        The average of lines of code per type is higher than its median. The calculated difference between them is 25.81.

======================ChatGPT Insight end(built prompt)

        GPT processing time: 3 seconds  GPT Context Tokens: 240 GPT Generated Tokens: ": 25
```

Source: Elaborated by the author

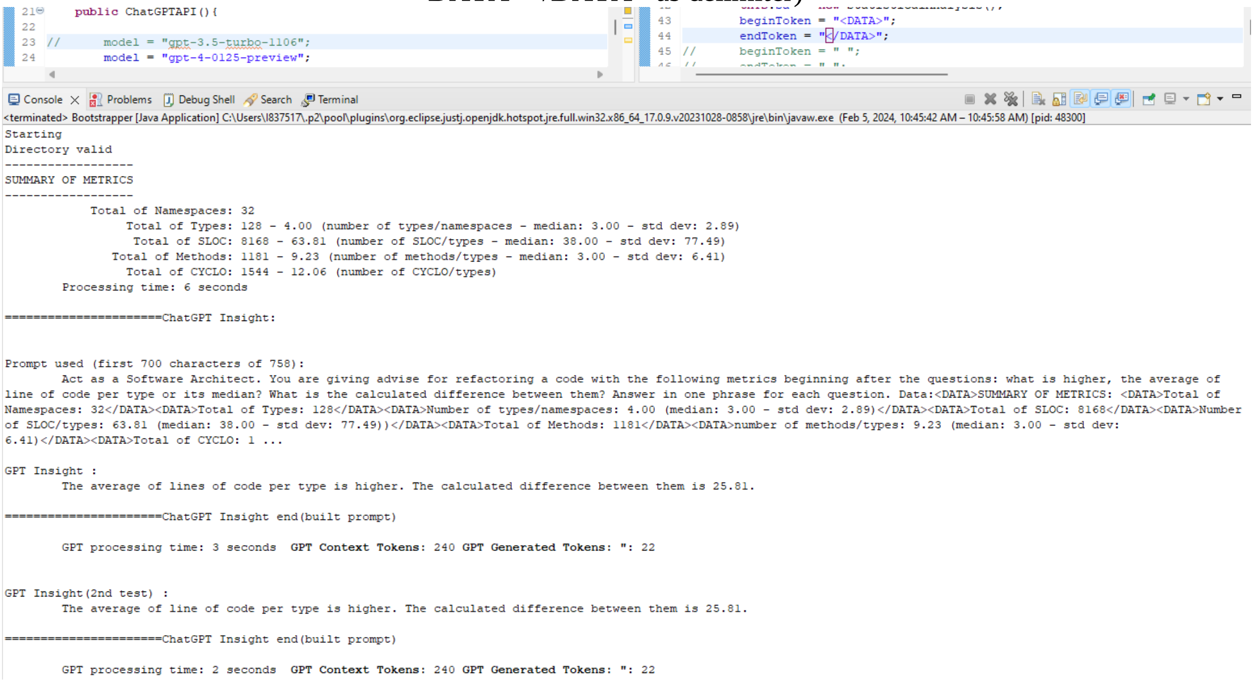
Figure A.3.11- Experiment provide difference between average and median (GPT-3.5 Turbo with <DATA></DATA> as delimiter)



```
21   public ChatGPTAPI(){
22
23       model = "gpt-3.5-turbo-1106";
24 //     model = "gpt-4-0125-preview";

42       this.sa = new StatisticalAnalysis();
43       beginToken = " <DATA>";
44       endToken = "</DATA> ";
45 //    beginToken = " ";

Console  Problems  Debug Shell  Search  Terminal
<terminated> Bootstrapper [Java Application] C:\Users\I837517\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (Feb 5, 2024, 10:25:46 AM – 10:26:00 AM) [pid: 25468]
Starting
Directory valid
------------------
SUMMARY OF METRICS
------------------
            Total of Namespaces: 32
                 Total of Types: 128 - 4.00 (number of types/namespaces - median: 3.00 - std dev: 2.89)
                  Total of SLOC: 8168 - 63.81 (number of SLOC/types - median: 38.00 - std dev: 77.49)
               Total of Methods: 1181 - 9.23 (number of methods/types - median: 3.00 - std dev: 6.41)
                 Total of CYCLO: 1544 - 12.06 (number of CYCLO/types)
        Processing time: 6 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 778):
        Act as a Software Architect. You are giving advise for refactoring a code with the following metrics beginning after the questions: what is higher, the average of line of code per type or its median? What is the calculated difference between them? Answer in one phrase for each question. Data: <DATA>SUMMARY OF METRICS:  <DATA>Total of Namespaces: 32</DATA>  <DATA>Total of Types: 128</DATA>  <DATA>Number of types/namespaces: 4.00 (median: 3.00 - std dev: 2.89)</DATA>  <DATA>Total of SLOC: 8168</DATA> <DATA>Number of SLOC/types: 63.81 (median: 38.00 - std dev: 77.49))</DATA>  <DATA>Total of Methods: 1181</DATA>  <DATA>number of methods/types: 9.23 (median: 3.00 - std dev: 6.41)</DATA>  <DATA>T ...

GPT Insight :
        - The average of lines of code per type is higher than its median.\n- The calculated difference between them is 25.81.

======================ChatGPT Insight end(built prompt)

        GPT processing time: 1 seconds  GPT Context Tokens: 260 GPT Generated Tokens: ": 27


GPT Insight(2nd test) :
        - The average of line of code per type is higher than its median.\n- The calculated difference between them is 25.81.

======================ChatGPT Insight end(built prompt)

        GPT processing time: 1 seconds  GPT Context Tokens: 260 GPT Generated Tokens: ": 27
```

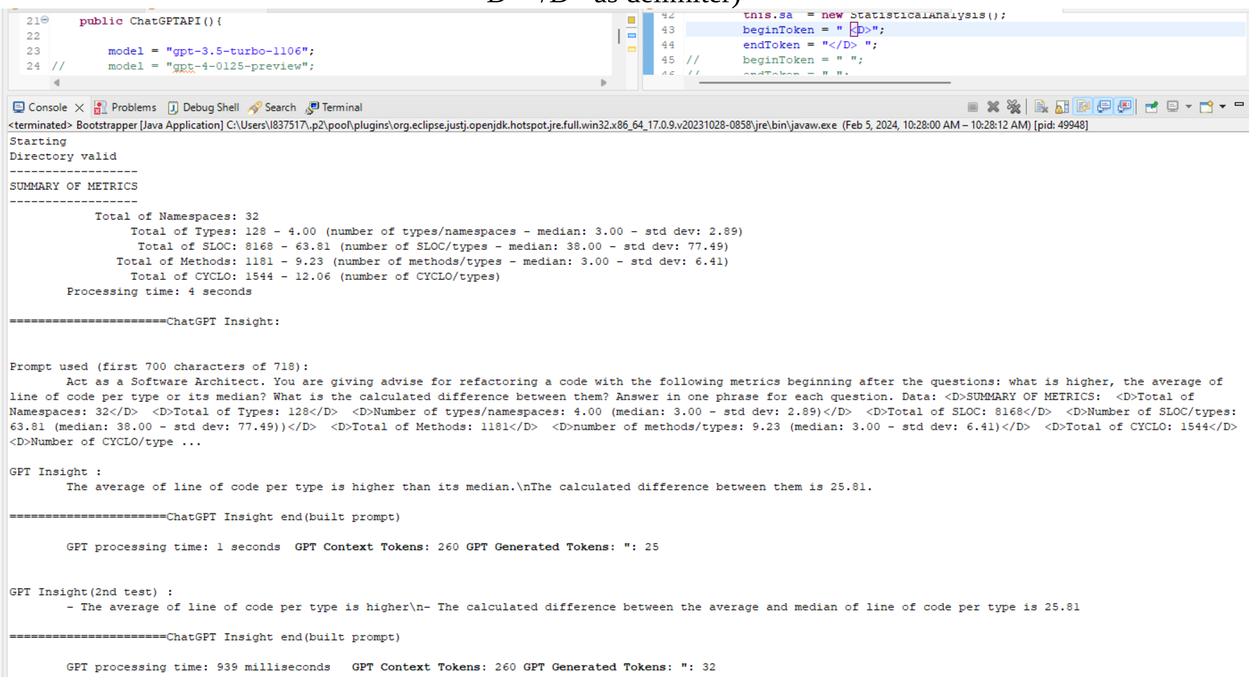
Source: Elaborated by the author

Figure A.3.12 - Experiment provide difference between average and median (GPT-4 Turbo with <DATA></DATA> as delimiter)



Source: Elaborated by the author

Figure A.3.13 - Experiment provide difference between average and median (GPT-3.5 Turbo with <D></D> as delimiter)



Source: Elaborated by the author

Figure A.3.14 - Experiment provide difference between average and median (GPT-4 Turbo with <D></D> as delimiter)

```
21     public ChatGPTAPI(){
22
23 //       model = "gpt-3.5-turbo-1106";
24         model = "gpt-4-0125-preview";

43           beginToken = "<D>";
44           endToken = "</D>";
45 //        beginToken = " ";
```

```
Console  Problems  Debug Shell  Search  Terminal
<terminated> Bootstrapper [Java Application] C:\Users\I837517\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (Feb 5, 2024, 10:44:05 AM – 10:44:21 AM) [pid: 50284]
Starting
Directory valid
------------------
SUMMARY OF METRICS
------------------
            Total of Namespaces: 32
                 Total of Types: 128 - 4.00 (number of types/namespaces - median: 3.00 - std dev: 2.89)
                  Total of SLOC: 8168 - 63.81 (number of SLOC/types - median: 38.00 - std dev: 77.49)
               Total of Methods: 1181 - 9.23 (number of methods/types - median: 3.00 - std dev: 6.41)
                 Total of CYCLO: 1544 - 12.06 (number of CYCLO/types)
        Processing time: 5 seconds

======================ChatGPT Insight:


Prompt used (complete):
        Act as a Software Architect. You are giving advise for refactoring a code with the following metrics beginning after the questions: what is higher, the average of line of code per type or its median? What is the calculated difference between them? Answer in one phrase for each question. Data:<D>SUMMARY OF METRICS: <D>Total of Namespaces: 32</D><D>Total of Types: 128</D><D>Number of types/namespaces: 4.00 (median: 3.00 - std dev: 2.89)</D><D>Total of SLOC: 8168</D><D>Number of SLOC/types: 63.81 (median: 38.00 - std dev: 77.49))</D><D>Total of Methods: 1181</D><D>number of methods/types: 9.23 (median: 3.00 - std dev: 6.41)</D><D>Total of CYCLO: 1544</D><D>Number of CYCLO/types: 12.06</D></D>


GPT Insight :
        The average of lines of code per type is higher. The calculated difference between them is 25.81.

======================ChatGPT Insight end(built prompt)

        GPT processing time: 2 seconds  GPT Context Tokens: 240 GPT Generated Tokens: ": 22


GPT Insight(2nd test) :
        The average of lines of code per type is higher. The calculated difference between them is 25.81.

======================ChatGPT Insight end(built prompt)

        GPT processing time: 3 seconds  GPT Context Tokens: 240 GPT Generated Tokens: ": 22
```

Source: Elaborated by the author

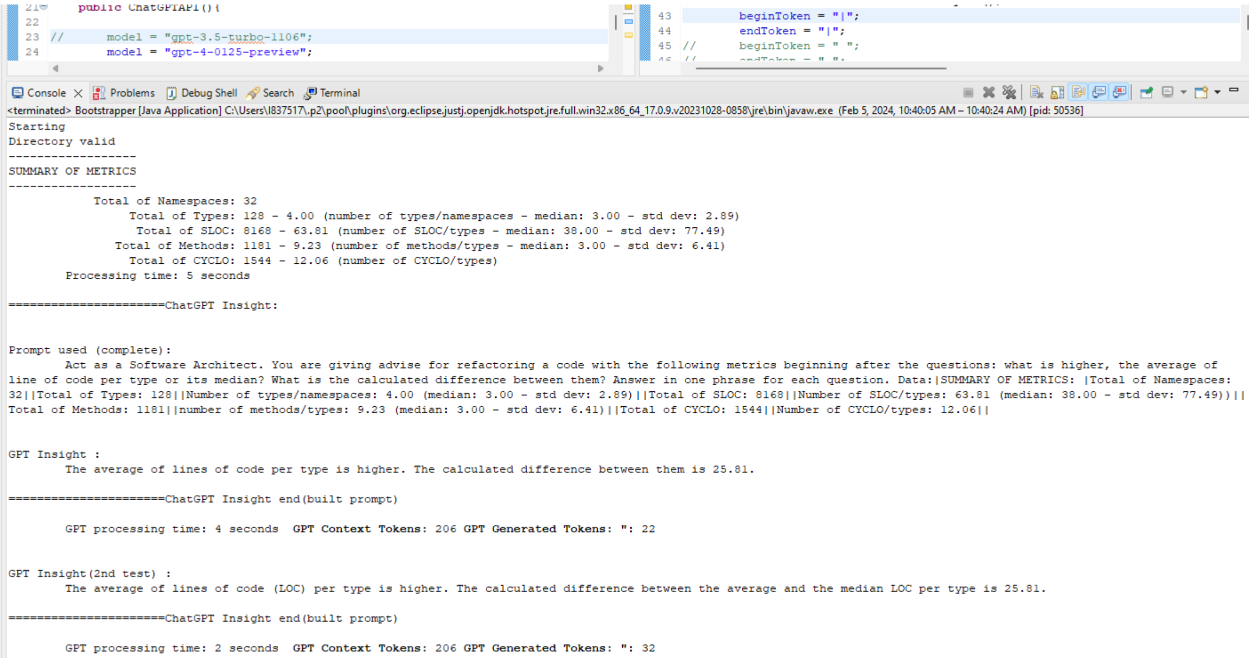
Figure A.3.15 - Experiment provide difference between average and median (GPT-3.5 Turbo with “|’ as delimiter)

```
GPTintegration.java  ChatGPTAPI.java
21     public ChatGPTAPI(){
22
23         model = "gpt-3.5-turbo-1106";
24 //       model = "gpt-4-0125-preview";

Bootstrappe...  InfoConsole....  GPT.java  MetricResul...  promptEngin...
43           beginToken = "|";
44           endToken = "|";
45 //        beginToken = " ";
```

```
Console  Problems  Debug Shell  Search  Terminal
<terminated> Bootstrapper [Java Application] C:\Users\I837517\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (Feb 5, 2024, 10:30:36 AM – 10:30:49 AM) [pid: 38736]
Starting
Directory valid
------------------
SUMMARY OF METRICS
------------------
            Total of Namespaces: 32
                 Total of Types: 128 - 4.00 (number of types/namespaces - median: 3.00 - std dev: 2.89)
                  Total of SLOC: 8168 - 63.81 (number of SLOC/types - median: 38.00 - std dev: 77.49)
               Total of Methods: 1181 - 9.23 (number of methods/types - median: 3.00 - std dev: 6.41)
                 Total of CYCLO: 1544 - 12.06 (number of CYCLO/types)
        Processing time: 4 seconds

======================ChatGPT Insight:


Prompt used (complete):
        Act as a Software Architect. You are giving advise for refactoring a code with the following metrics beginning after the questions: what is higher, the average of line of code per type or its median? What is the calculated difference between them? Answer in one phrase for each question. Data:|SUMMARY OF METRICS: |Total of Namespaces: 32||Total of Types: 128||Number of types/namespaces: 4.00 (median: 3.00 - std dev: 2.89)||Total of SLOC: 8168||Number of SLOC/types: 63.81 (median: 38.00 - std dev: 77.49))||Total of Methods: 1181||number of methods/types: 9.23 (median: 3.00 - std dev: 6.41)||Total of CYCLO: 1544||Number of CYCLO/types: 12.06||


GPT Insight :
        Average line of code per type is higher than its median. The calculated difference between them is 25.81.

======================ChatGPT Insight end(built prompt)

        GPT processing time: 1 seconds  GPT Context Tokens: 206 GPT Generated Tokens: ": 23


GPT Insight(2nd test) :
        Higher: The average of line of code per type is higher than its median.\n\nDifference: The calculated difference between the average and median line of code per type is 25.81.

======================ChatGPT Insight end(built prompt)

        GPT processing time: 1 seconds  GPT Context Tokens: 206 GPT Generated Tokens: ": 37
```

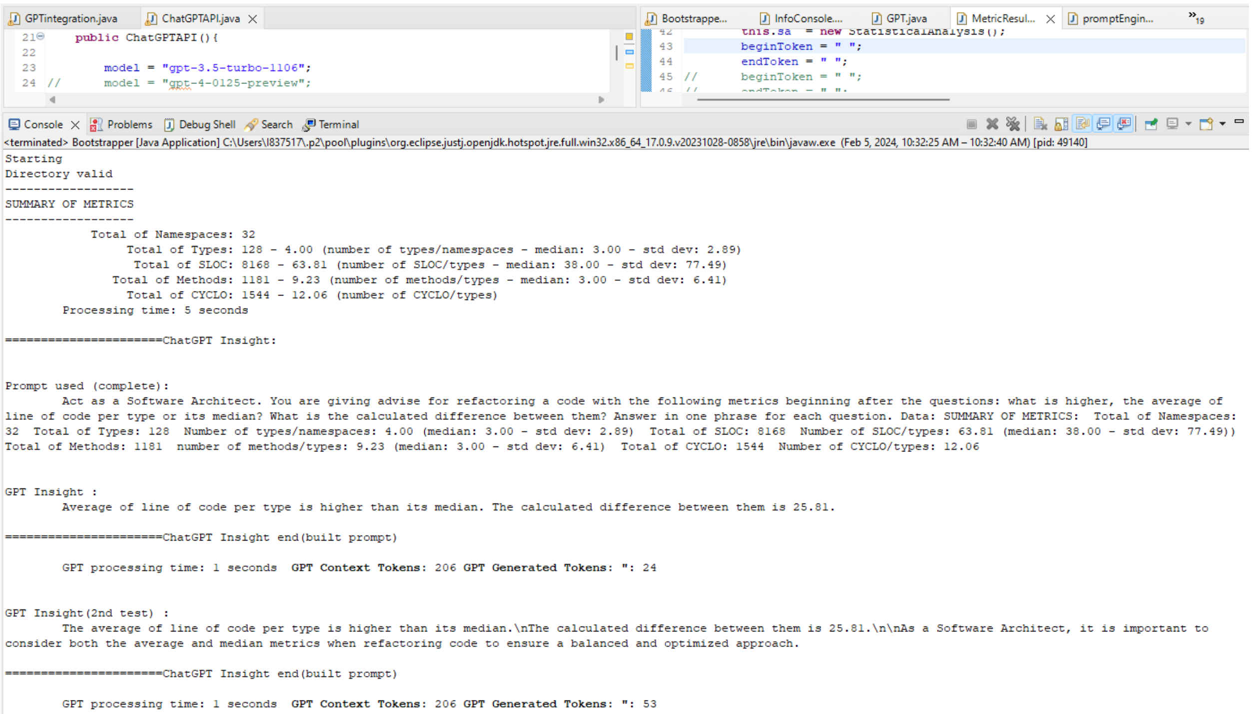
Source: Elaborated by the author

Figure A.3.16 - Experiment provide difference between average and median (GPT-4 Turbo with "|" as delimiter)



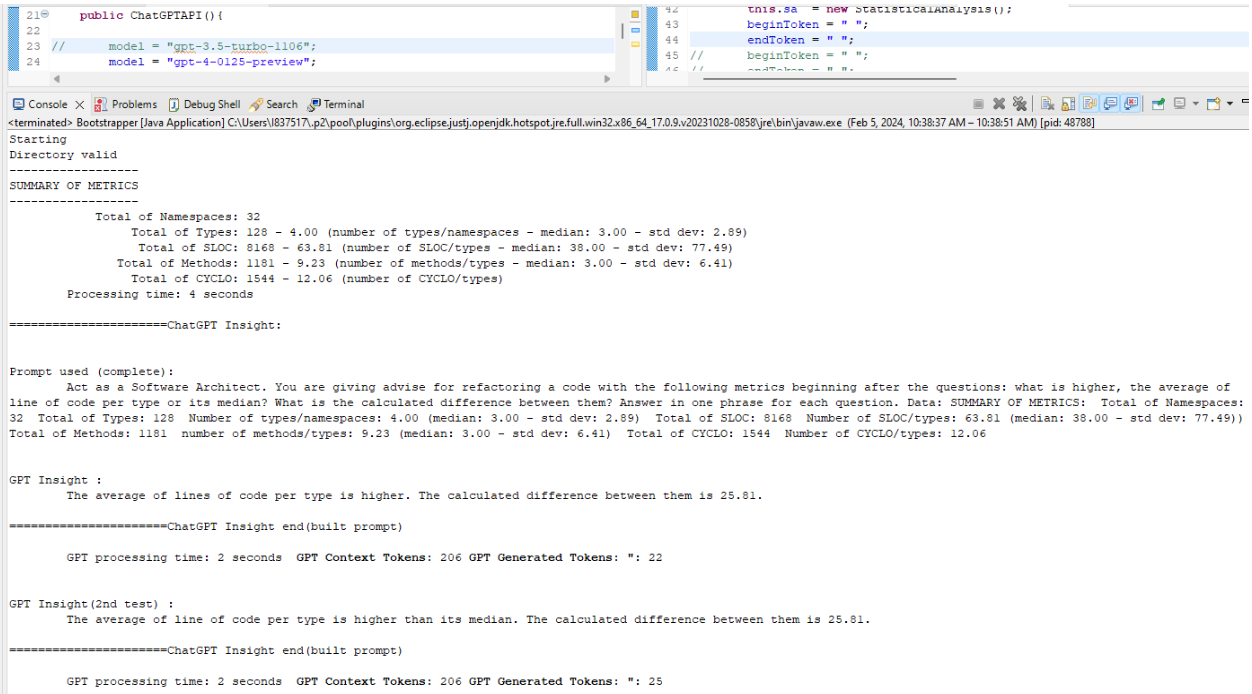
Source: Elaborated by the author

Figure A.3.17 - Experiment provide difference between average and median (GPT-3.5 Turbo with space as delimiter)



Source: Elaborated by the author

Figure A.3.18 - Experiment provide difference between average and median (GPT-4 Turbo with space as delimiter)

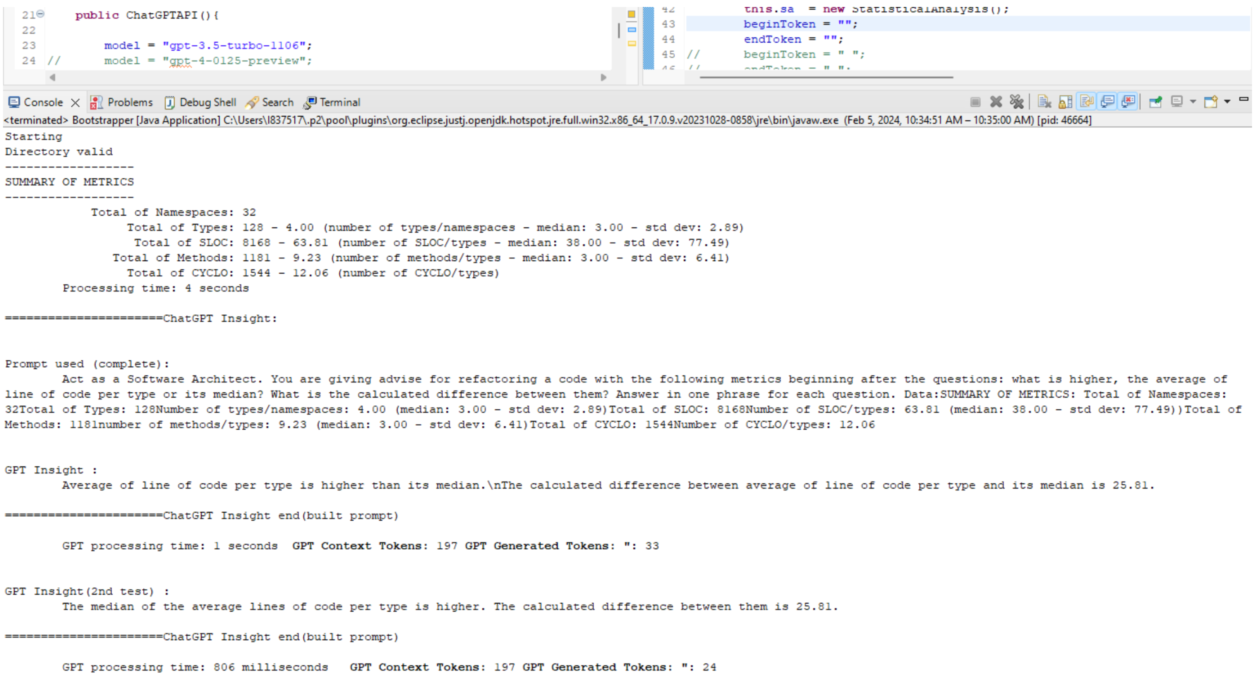


```
21    public ChatGPTAPI(){
22
23 //     model = "gpt-3.5-turbo-1106";
24        model = "gpt-4-0125-preview";

42        this.sa  = new StatisticalAnalysis();
43        beginToken = " ";
44        endToken = " ";
45 //     beginToken = " ";
```

```
<terminated> Bootstrapper [Java Application] C:\Users\I837517\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (Feb 5, 2024, 10:38:37 AM – 10:38:51 AM) [pid: 48788]
Starting
Directory valid
------------------
SUMMARY OF METRICS
------------------
            Total of Namespaces: 32
                 Total of Types: 128 - 4.00 (number of types/namespaces - median: 3.00 - std dev: 2.89)
                  Total of SLOC: 8168 - 63.81 (number of SLOC/types - median: 38.00 - std dev: 77.49)
               Total of Methods: 1181 - 9.23 (number of methods/types - median: 3.00 - std dev: 6.41)
                 Total of CYCLO: 1544 - 12.06 (number of CYCLO/types)
        Processing time: 4 seconds

======================ChatGPT Insight:


Prompt used (complete):
        Act as a Software Architect. You are giving advise for refactoring a code with the following metrics beginning after the questions: what is higher, the average of line of code per type or its median? What is the calculated difference between them? Answer in one phrase for each question. Data: SUMMARY OF METRICS:  Total of Namespaces: 32  Total of Types: 128  Number of types/namespaces: 4.00 (median: 3.00 - std dev: 2.89)  Total of SLOC: 8168  Number of SLOC/types: 63.81 (median: 38.00 - std dev: 77.49))  Total of Methods: 1181  number of methods/types: 9.23 (median: 3.00 - std dev: 6.41)  Total of CYCLO: 1544  Number of CYCLO/types: 12.06


GPT Insight :
        The average of lines of code per type is higher. The calculated difference between them is 25.81.

======================ChatGPT Insight end(built prompt)

        GPT processing time: 2 seconds  GPT Context Tokens: 206 GPT Generated Tokens: ": 22


GPT Insight(2nd test) :
        The average of line of code per type is higher than its median. The calculated difference between them is 25.81.

======================ChatGPT Insight end(built prompt)

        GPT processing time: 2 seconds  GPT Context Tokens: 206 GPT Generated Tokens: ": 25
```

Source: Elaborated by the author

Figure A.3.19 - Experiment provide difference between average and median (GPT-3.5 Turbo no delimiter)

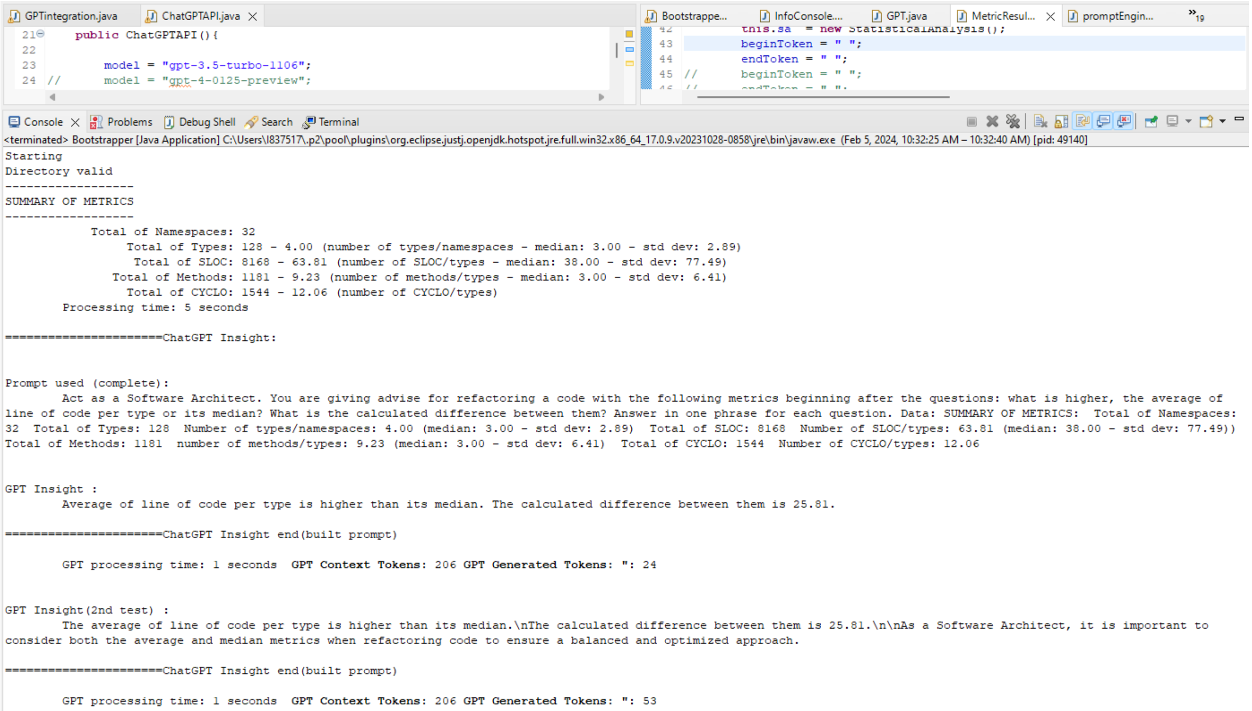


```
21    public ChatGPTAPI(){
22
23        model = "gpt-3.5-turbo-1106";
24 //     model = "gpt-4-0125-preview";

42        this.sa  = new StatisticalAnalysis();
43        beginToken = "";
44        endToken = "";
45 //     beginToken = " ";
```

```
<terminated> Bootstrapper [Java Application] C:\Users\I837517\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (Feb 5, 2024, 10:34:51 AM – 10:35:00 AM) [pid: 46664]
Starting
Directory valid
------------------
SUMMARY OF METRICS
------------------
            Total of Namespaces: 32
                 Total of Types: 128 - 4.00 (number of types/namespaces - median: 3.00 - std dev: 2.89)
                  Total of SLOC: 8168 - 63.81 (number of SLOC/types - median: 38.00 - std dev: 77.49)
               Total of Methods: 1181 - 9.23 (number of methods/types - median: 3.00 - std dev: 6.41)
                 Total of CYCLO: 1544 - 12.06 (number of CYCLO/types)
        Processing time: 4 seconds

======================ChatGPT Insight:


Prompt used (complete):
        Act as a Software Architect. You are giving advise for refactoring a code with the following metrics beginning after the questions: what is higher, the average of line of code per type or its median? What is the calculated difference between them? Answer in one phrase for each question. Data:SUMMARY OF METRICS: Total of Namespaces: 32Total of Types: 128Number of types/namespaces: 4.00 (median: 3.00 - std dev: 2.89)Total of SLOC: 8168Number of SLOC/types: 63.81 (median: 38.00 - std dev: 77.49))Total of Methods: 1181number of methods/types: 9.23 (median: 3.00 - std dev: 6.41)Total of CYCLO: 1544Number of CYCLO/types: 12.06


GPT Insight :
        Average of line of code per type is higher than its median.\nThe calculated difference between average of line of code per type and its median is 25.81.

======================ChatGPT Insight end(built prompt)

        GPT processing time: 1 seconds  GPT Context Tokens: 197 GPT Generated Tokens: ": 33


GPT Insight(2nd test) :
        The median of the average lines of code per type is higher. The calculated difference between them is 25.81.

======================ChatGPT Insight end(built prompt)

        GPT processing time: 806 milliseconds   GPT Context Tokens: 197 GPT Generated Tokens: ": 24
```

Source: Elaborated by the author

Figure A.3.20 – Experiment provide difference between average and median (GPT-3.5 Turbo with space as separator)



Source: Elaborated by the author

Experiment Average vs. Median (50 request to more accuracy):

Test with 50 requests to have % correctness.

```
GPT-3.5 Turbo testing prompt for improve comparation

Starting
Directory valid
------------------
SUMMARY OF METRICS
------------------
Total of Namespaces: 32
Total of Types: 128 - 4.00 (number of types/namespaces - median: 3.00 - std
dev: 2.89)
Total of SLOC: 8558 - 66.86 (number of SLOC/types - median: 38.00 - std
dev: 84.46)
Total of Methods: 1181 - 9.23 (number of methods/types - median: 3.00 - std
dev: 16.51)
Total of CYCLO: 1592 - 12.44 (number of CYCLO/types)
Processing time: 4 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 710):
Act as a Software Architect. You are giving advice for refactoring a code
with the following metrics beginning after the questions: what is the
```

average and median of line of code per type? Which is higher? What is the calculated difference between them? Answer in one phrase for each question. Data:|SUMMARY OF METRICS: |Total of Namespaces: 32||Total of Types: 128||Number of types/namespaces|Average: 4.00||Median: 3.00||Standard Deviation: 2.89|||Total of SLOC: 8558||Number of SLOC/types|Average: 66.86||Median: 38.00||Standard deviation: 84.46|||Total of Methods: 1181||Number of methods/types:|Average: 9.23||Median: 3.00||Standard deviation: 16.51|||Total of CYCLO: 1592||Number of CYCLO/type ...

GPT Insight :
Average SLOC per type is higher than median SLOC per type. The calculated difference between them is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens:** [0m218 **[1mGPT Generated Tokens:** [0m": 25

GPT Insight(2nd test) :
Average SLOC per type is 66.86, median SLOC per type is 38.00, the average is higher. The calculated difference between them is 28.86.\nAverage methods per type is 9.23, median methods per type is 3.00, the average is higher. The calculated difference between them is 6.23.

======================ChatGPT Insight end(built prompt)

GPT processing time: 2 seconds **[1mGPT Context Tokens:** [0m218 **[1mGPT Generated Tokens:** [0m": 74

GPT Insight(3rd test) :
The average line of code per type is higher for SLOC at 66.86 compared to the median of 38.00, with a calculated difference of 28.86.\nThe average number of methods per type is higher at 9.23 compared to the median of 3.00, with a calculated difference of 6.23.

======================ChatGPT Insight end(built prompt)

GPT processing time: 2 seconds **[1mGPT Context Tokens:** [0m218 **[1mGPT Generated Tokens:** [0m": 71

GPT Insight(4th test) :
Average SLOC per type is higher than median.\nThe calculated difference between average and median SLOC per type is 28.86.\nMedian number of methods per type is higher than average.\nThe calculated difference between average and median number of methods per type is 6.23.

======================ChatGPT Insight end(built prompt)

GPT processing time: 2 seconds **[1mGPT Context Tokens:** [0m218 **[1mGPT Generated Tokens:** [0m": 56

GPT Insight(5th test) :
Average line of code per type is 66.86, median line of code per type is 38.00, the median is higher than the average with a difference of 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 40

GPT Insight(6th test) :
Average line of code per type: 66.86 SLOC\nMedian line of code per type: 38.00 SLOC\nThe average is higher than the median by 28.86 SLOC.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 43

GPT Insight(7th test) :
- The average and median of line of code per type are: Average: 66.86 and Median: 38.00. Average is higher\n- The calculated difference between them is 28.86\n- The average and median of number of methods per type are: Average: 9.23 and Median: 3.00. Average is higher

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 73

GPT Insight(8th test) :
The average SLOC per type is higher than the median. The calculated difference between them is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 23

GPT Insight(9th test) :
- The average line of code per type is higher.\n- The calculated difference between the average and median is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 26

GPT Insight(10th test) :
Average of LOC per type is higher.\nThe calculated difference between average and median of LOC per type is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 852 milliseconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 25

GPT Insight :
Average SLOC per type is higher.\nThe calculated difference between average and median SLOC per type is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 768 milliseconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 25

GPT Insight(12nd test) :
The average number of lines of code per type is higher than the median.\nThe calculated difference between the average and median number of lines of code per type is 28.86.\nThe average number of methods per type is higher than the median.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 49

GPT Insight(13rd test) :
Average SLOC per type is 66.86 and median is 38.00. Median is higher.\n\nThe calculated difference between the average and median SLOC per type is 28.86.\n\nAverage number of methods per type is 9.23 and median is 3.00. Median is higher.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 63

GPT Insight(14th test) :
- Average and Median of SLOC per type: Average SLOC per type is 66.86, Median SLOC per type is 38.00\n- SLOC per type median is higher\n- The calculated difference between average and median SLOC per type is 28.86

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 58

GPT Insight(15th test) :
Average line of code per type is higher than the median.\nThe calculated difference between average and median line of code per type is 28.86.\nHigher standard deviation indicates greater variability in the number of lines of code per type.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 46

GPT Insight(16th test) :
Average and median of line of code per type: \nAverage: 66.86 \nMedian: 38.00\n\nThe higher value is the average. \nThe calculated difference between them is 28.86, indicating a considerable variation in the line of code per type. \n\nAs a Software Architect, my advice for refactoring the code would be to focus on standardizing the size and complexity of each

type by identifying and addressing outliers, and ensuring consistent coding practices are being followed to maintain a more uniform and maintainable codebase.

======================ChatGPT Insight end(built prompt)

GPT processing time: 2 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 108

GPT Insight(17th test) :
The average number of lines of code per type is higher than the median. The calculated difference between them is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 2 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 26

GPT Insight(18th test) :
- The average line of code per type is higher than the median.\n- The calculated difference between the average and median line of code per type is 28.86.\n- The average number of methods per type is higher than the median.

======================ChatGPT Insight end(built prompt)

GPT processing time: 2 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 48

GPT Insight(19th test) :
The average line of code per type is higher than the median. The calculated difference between them is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 24

GPT Insight(20th test) :
Average and median of lines of code per type: average is 66.86, median is 38.00. Average is higher. The calculated difference between them is 28.86.\nAverage and median of methods per type: average is 9.23, median is 3.00. Average is higher. The calculated difference between them is 6.23.

======================ChatGPT Insight end(built prompt)

GPT processing time: 2 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 76

GPT Insight (21st test):
The average and median of lines of code per type is higher than the average and median of methods per type. The calculated difference between them is 57.86 SLOC.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 35

GPT Insight(22nd test) :
Average SLOC per type is higher than the median. The calculated difference between them is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 904 milliseconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 22

GPT Insight(23rd test) :
Average and median of line of code per type is higher. The calculated difference between them is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 23

GPT Insight(24th test) :
Average and median of line of code per type: Average SLOC per type is 66.86, while the median SLOC per type is 38.00. \nThe average is higher than the median. \nThe calculated difference between them is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 55

GPT Insight(25th test) :
Average SLOC per type is higher.\nThe calculated difference between average and median of SLOC per type is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 841 milliseconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 26

GPT Insight(26th test) :
Average number of lines of code per type is higher than the median. The calculated difference between them is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 794 milliseconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 25

GPT Insight(27th test) :
Average and median of line of code per type: \nAverage: 66.86 \nMedian: 38.00\nHigher metric: Average\nCalculated difference: 28.86

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 38

GPT Insight(28th test) :
Average and median of line of code per type: \nAverage: 66.86 SLOC\nMedian: 38.00 SLOC\nThe average is higher than the median by 28.86 SLOC.\n\nAs a Software Architect, I would advise refactoring the code to reduce the average lines of code per type to bring it closer to the median, in order to improve consistency and maintainability.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 82

GPT Insight(29th test) :
1. Average SLOC per type is 66.86, median SLOC per type is 38.00. Average is higher.\n\n2. The calculated difference between average and median SLOC per type is 28.86.\n\n3. Average number of methods per type is 9.23, median is 3.00. Average is higher.\n\n4. The calculated difference between average and median number of methods per type is 6.23.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 92

GPT Insight(30th test) :
Average SLOC per type is higher than median.\nThe calculated difference between average SLOC per type and median SLOC per type is 28.86.\nMedian methods per type is higher than average.\nThe calculated difference between average methods per type and median methods per type is 6.23.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 59

GPT Insight (31th test:
The average SLOC per type is higher than the median SLOC per type. The calculated difference between them is 28.86.\nThe average number of methods per type is higher than the median number of methods per type. The calculated difference between them is 6.23.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 56

GPT Insight(32nd test) :
Average SLOC per type is higher than the median. The calculated difference between them is 28.86.\nAverage methods per type is higher than the median. The calculated difference between them is 6.23.

```
======================ChatGPT Insight end(built prompt)
```

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 43

```
GPT Insight(33rd test) :
Average and median of lines of code per type: 66.86 and 38.00, the average is higher. The calculated difference between them is 28.86.\nAverage and median of methods per type: 9.23 and 3.00, the average is higher. The calculated difference between them is 6.23.

======================ChatGPT Insight end(built prompt)
```

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 70

```
GPT Insight(34th test) :
The average number of lines of code per type is higher than the median. The calculated difference between them is 28.86.

======================ChatGPT Insight end(built prompt)
```

GPT processing time: 935 milliseconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 26

```
GPT Insight(35th test) :
Average and median of line of code per type: \nAverage SLOC per type is 66.86, median SLOC per type is 38.00. Average is higher than median by 28.86.\n\nAverage and median of number of methods per type: \nAverage number of methods per type is 9.23, median number of methods per type is 3.00. Average is higher than median by 6.23.\n\nAdvice for refactoring: \nConsider refactoring types with higher SLOC and number of methods to improve maintainability and readability of the codebase.

======================ChatGPT Insight end(built prompt)
```

GPT processing time: 2 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 119

```
GPT Insight(36th test) :
Average and median of line of code per type: \n- Average: 66.86 SLOC\n- Median: 38.00 SLOC\n- The average is higher than the median.\n- The calculated difference between them is 28.86 SLOC.

======================ChatGPT Insight end(built prompt)
```

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 54

```
GPT Insight(37th test) :
Average SLOC per type is higher than median, with a calculated difference of 28.86.\nMedian number of methods per type is higher than average.

======================ChatGPT Insight end(built prompt)
```

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 31

GPT Insight(38th test) :
Average and median of lines of code per type is higher than average and median of methods per type\nThe calculated difference between average and median of lines of code per type is 28.86\nThe calculated difference between average and median of methods per type is 6.23

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 56

GPT Insight(39th test) :
Average SLOC per type is higher than the median.\nThe calculated difference between the average and median SLOC per type is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 846 milliseconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 29

GPT Insight(40th test) :
Average SLOC per type is higher\nThe calculated difference between average and median SLOC per type is 28.86\nMedian number of methods per type is higher

======================ChatGPT Insight end(built prompt)

GPT processing time: 926 milliseconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 33

GPT Insight(41st test):
The average and median of line of code per type is higher than the average and median of method per type. The calculated difference between them is 57.86 SLOC.

======================ChatGPT Insight end(built prompt)

GPT processing time: 942 milliseconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 35

GPT Insight(42nd test) :
1. The average and median of line of code per type is 66.86 and 38.00, respectively.\n2. The average is higher than the median.\n3. The calculated difference between them is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 48

GPT Insight(43rd test) :
Average SLOC per type is higher.\nThe calculated difference between average and median SLOC per type is 28.86.\nMedian number of methods per type is higher. \nThe calculated difference between average and median number of methods per type is 6.23.

======================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 53

GPT Insight(44th test) :
Average SLOC per type is 66.86, median is 38.00, with a calculated difference of 28.86. Median SLOC per type is higher.

======================ChatGPT Insight end(built prompt)

GPT processing time: 931 milliseconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 36

GPT Insight(45th test) :
Average SLOC per type is higher.\nThe calculated difference between average and median SLOC per type is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 909 milliseconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 25

GPT Insight(46th test) :
Average SLOC per type is higher. \nThe calculated difference between average and median SLOC per type is 28.86. \nMedian methods per type is higher. \nThe calculated difference between average and median methods per type is 6.23.

======================ChatGPT Insight end(built prompt)

GPT processing time: 989 milliseconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 51

GPT Insight(47th test) :
The average number of lines of code per type is 66.86, while the median is 38.00. The calculated difference between them is 28.86.

======================ChatGPT Insight end(built prompt)

GPT processing time: 841 milliseconds **[1mGPT Context Tokens**: [0m218 **[1mGPT Generated Tokens**: [0m": 35

GPT Insight(48th test) :
Average and median of line of code per type: Average is 66.86 and median is 38.00. \n\nThe calculated difference between them: 28.86\n\nThe average line of code per type is higher.

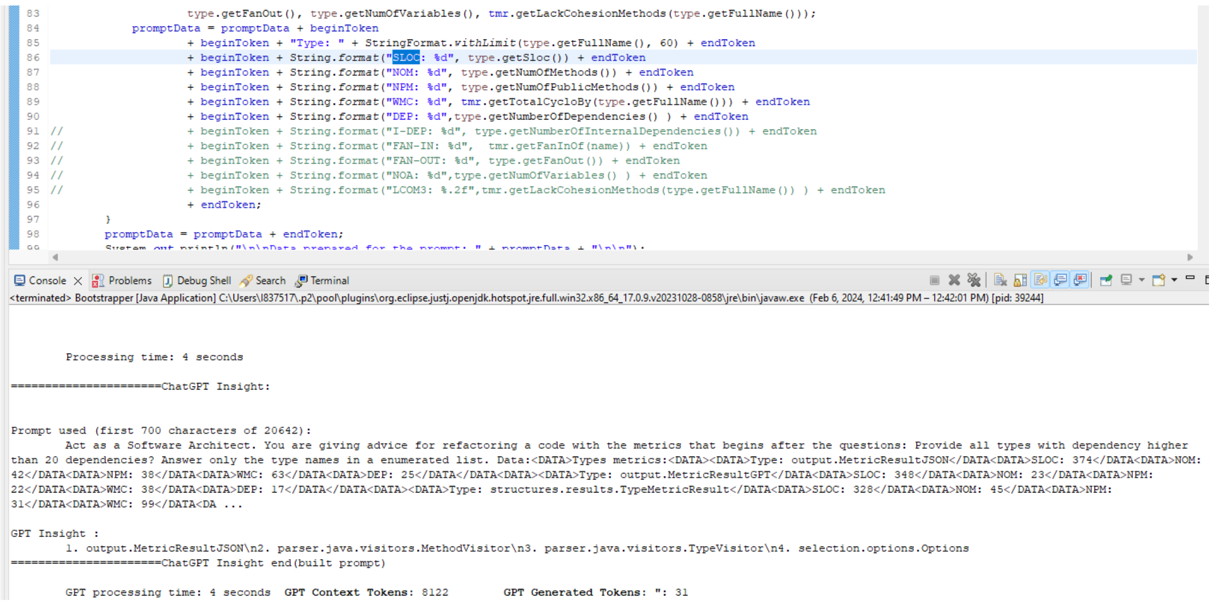
======================ChatGPT Insight end(built prompt)

```
GPT processing time: 1 seconds [1mGPT Context Tokens: [0m218 [1mGPT
Generated Tokens: [0m": 46


GPT Insight(49th test) :
Average SLOC per type is higher than median. The calculated difference
between them is 28.86.\nMedian methods per type is lower than average. The
calculated difference between them is 6.23.

=====================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds [1mGPT Context Tokens: [0m218 [1mGPT
Generated Tokens: [0m": 41


GPT Insight(50th test) :
1. The average of SLOC per type is higher than the median.\n2. The
calculated difference between the average and median of SLOC per type is
28.86. \n3. The average of methods per type is higher than the median.\n4.
The calculated difference between the average and median of methods per
type is 6.23.

=====================ChatGPT Insight end(built prompt)

GPT processing time: 1 seconds [1mGPT Context Tokens: [0m218 [1mGPT
Generated Tokens: [0m": 71
```

3.2.2.2 GPT-4 Turbo API Rate Limits and its implications

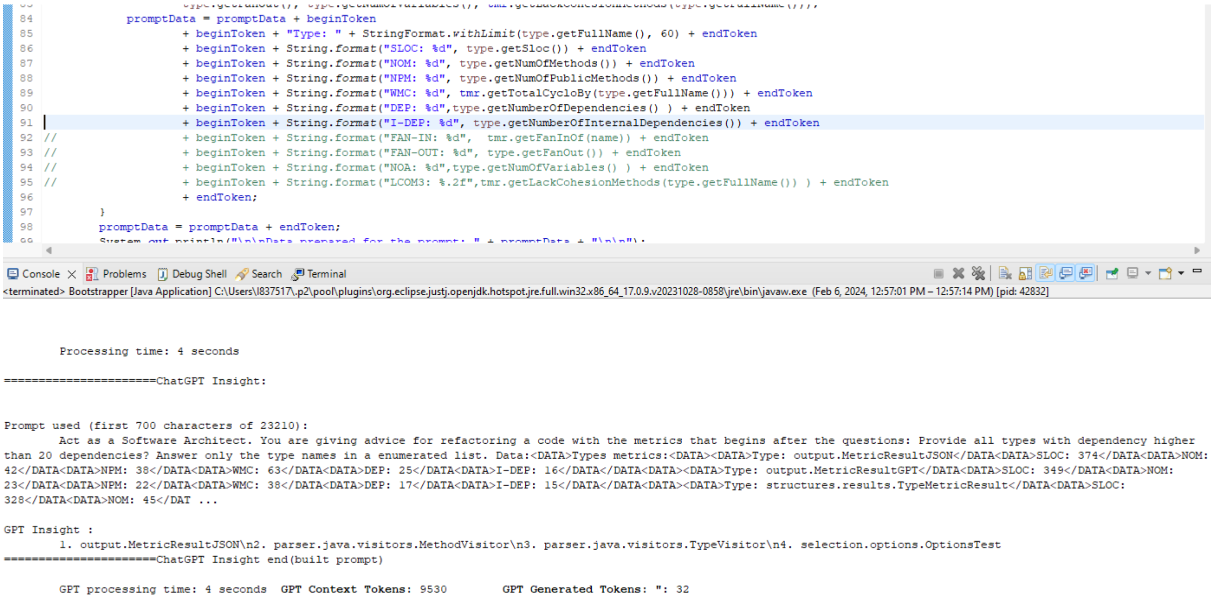
Below follow several tests were done to explore the limit and at the end the answer that OpenAI support provides us when asked about the limit.

Figure A.3.21 – ChatGPT Request with Type metrics: SLOC, NOM, NPM, WMC and DEP metrics



Source: Elaborated by the author

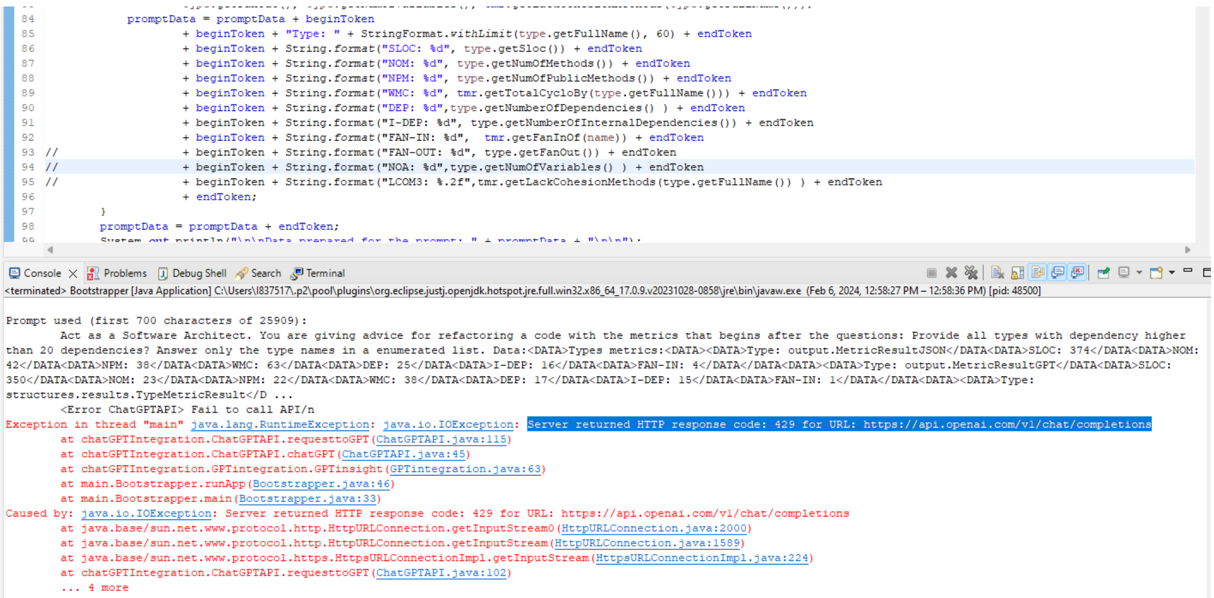
Figure A.3.22 – ChatGPT Request with Type metrics: SLOC, NOM, NPM, WMC, DEP and I-DEP metrics



Source: Elaborated by the author

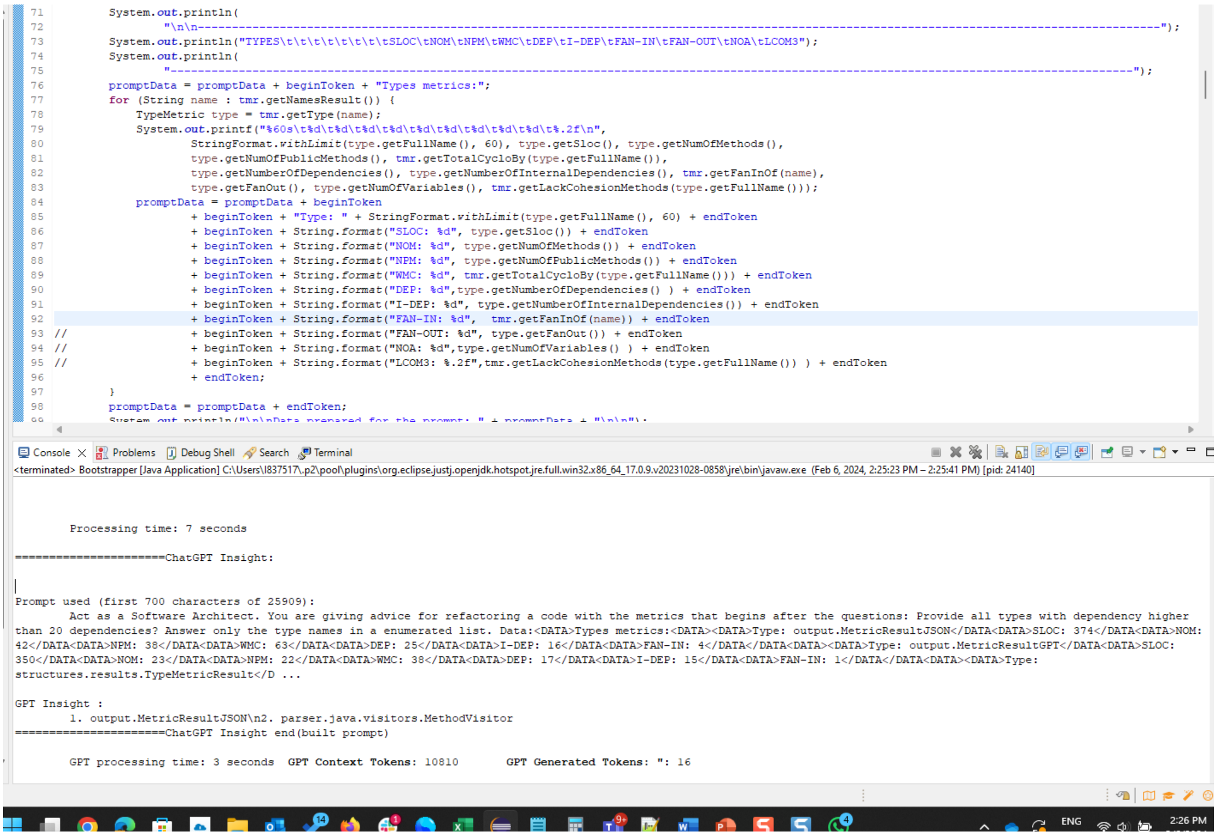
More data increase the number of tokens, which extrapolate the limit of tokens per minute that OpenAI is limiting.

Figure A.3.23 – ChatGPT Request with Type metrics: SLOC, NOM, NPM, WMC, DEP, I-DEP and FAN-IN metrics



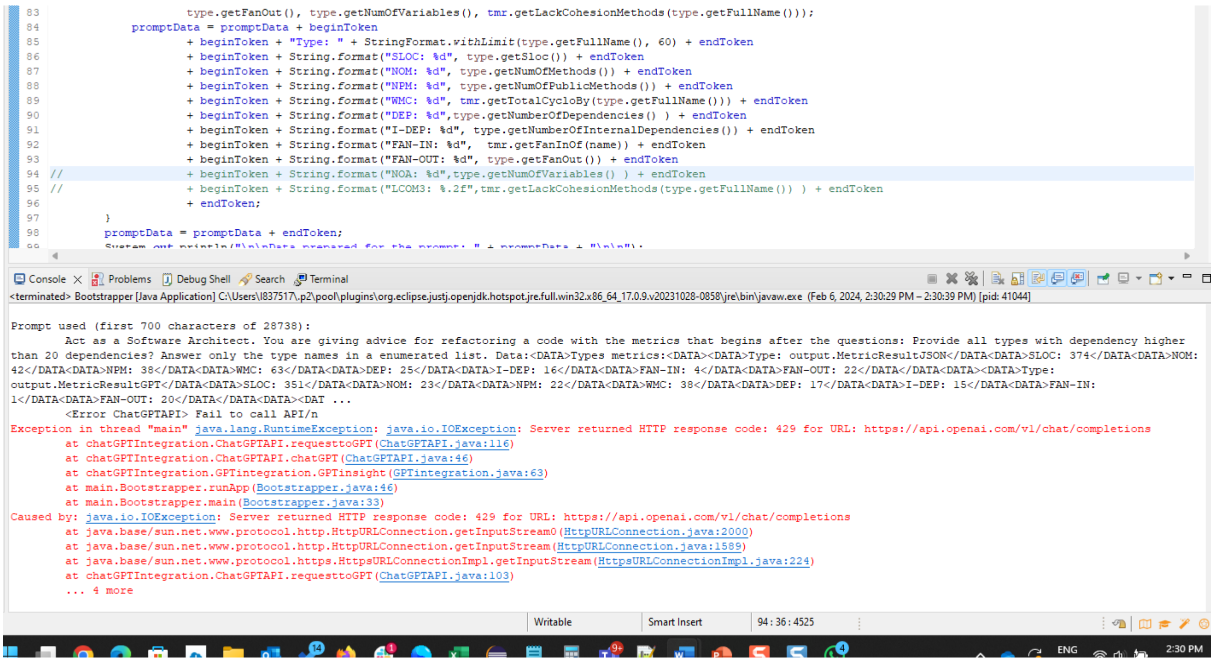
Source: Elaborated by the author

Figure A.3.24 – ChatGPT Request with Type metrics: SLOC, NOM, NPM, WMC, DEP and I-DEP metrics



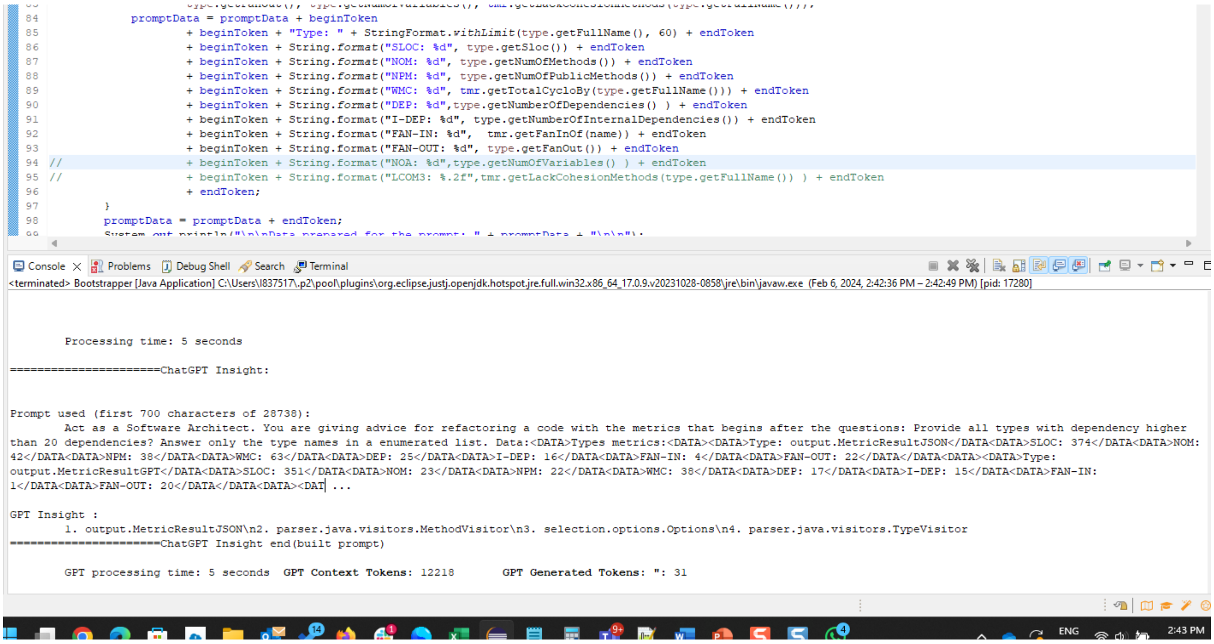
Source: Elaborated by the author

Figure A.3.25 – ChatGPT Request with Type metrics: SLOC, NOM, NPM, WMC, DEP, I-DEP, FAN-IN and FAN-OUT metrics (fail)



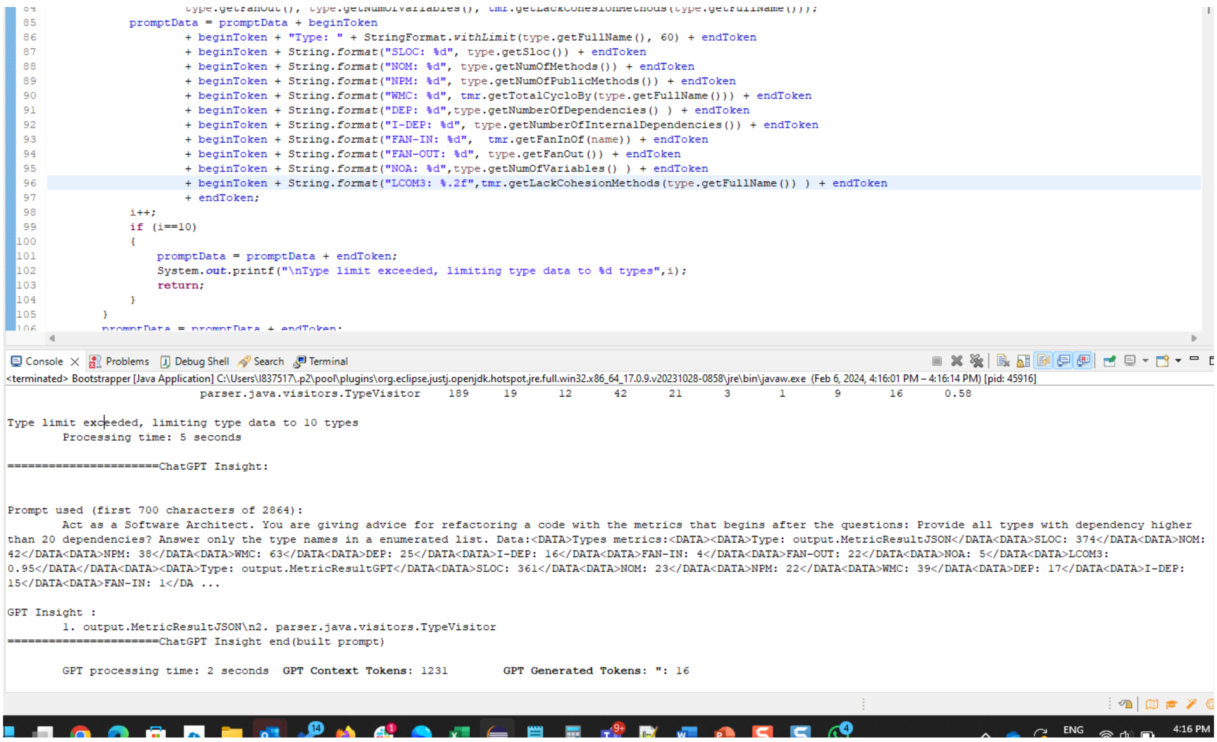
Source: Elaborated by the author

Figure A.3.26 – ChatGPT Request with Type metrics: SLOC, NOM, NPM, WMC, DEP and I-DEP metrics (Successful)



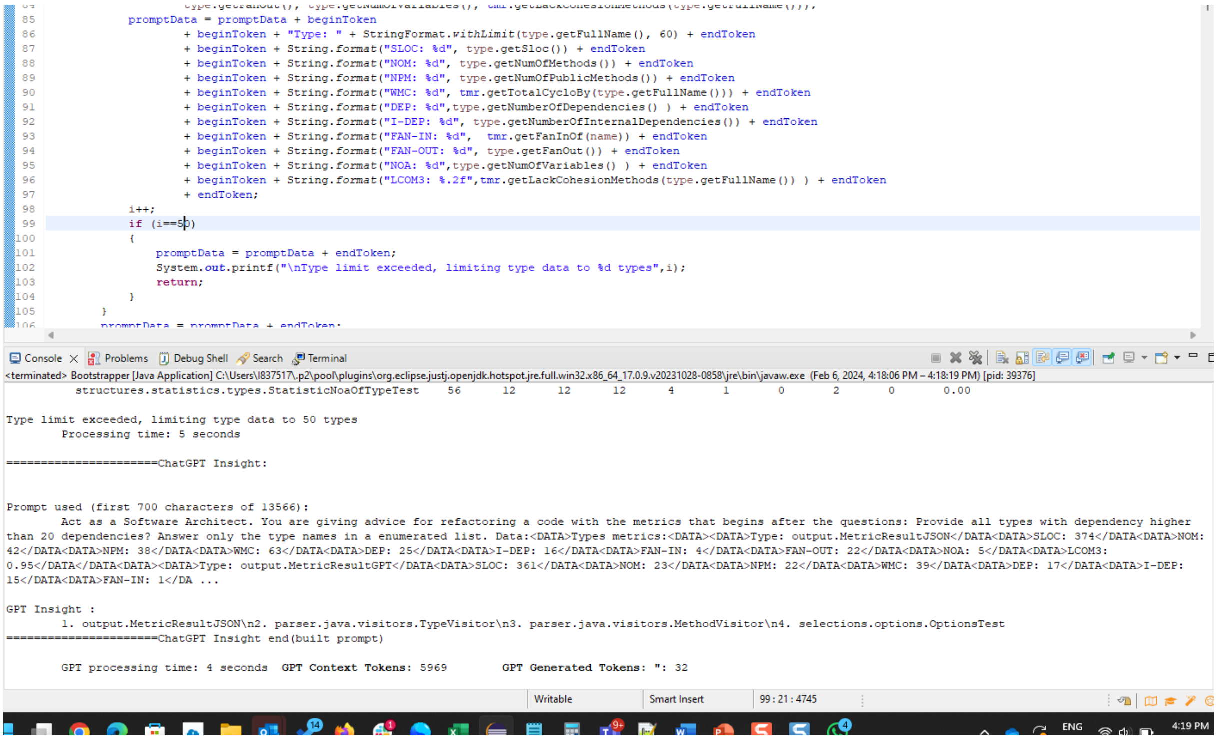
Source: Elaborated by the author

Figure A.3.27 – ChatGPT with all type metrics, limiting data to 10 types (classes)



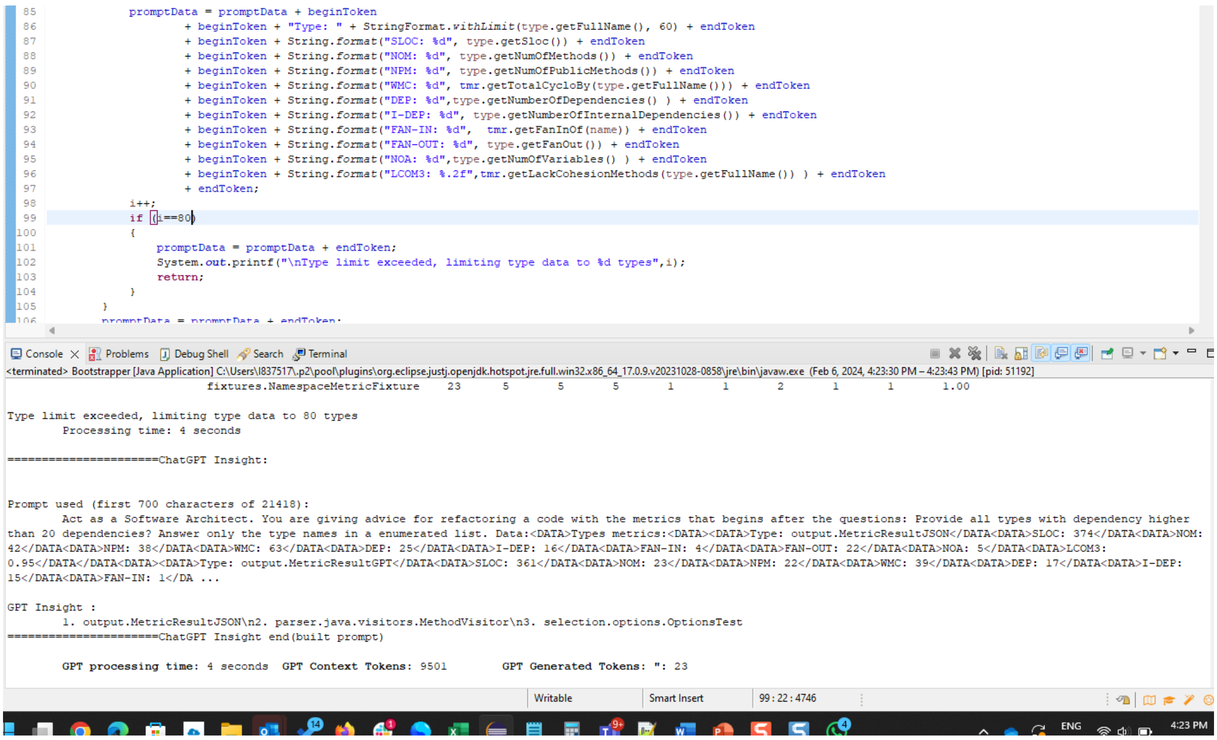
Source: Elaborated by the author

Figure A.3.28 – ChatGPT with all type metrics, limiting data to 50 types (classes)



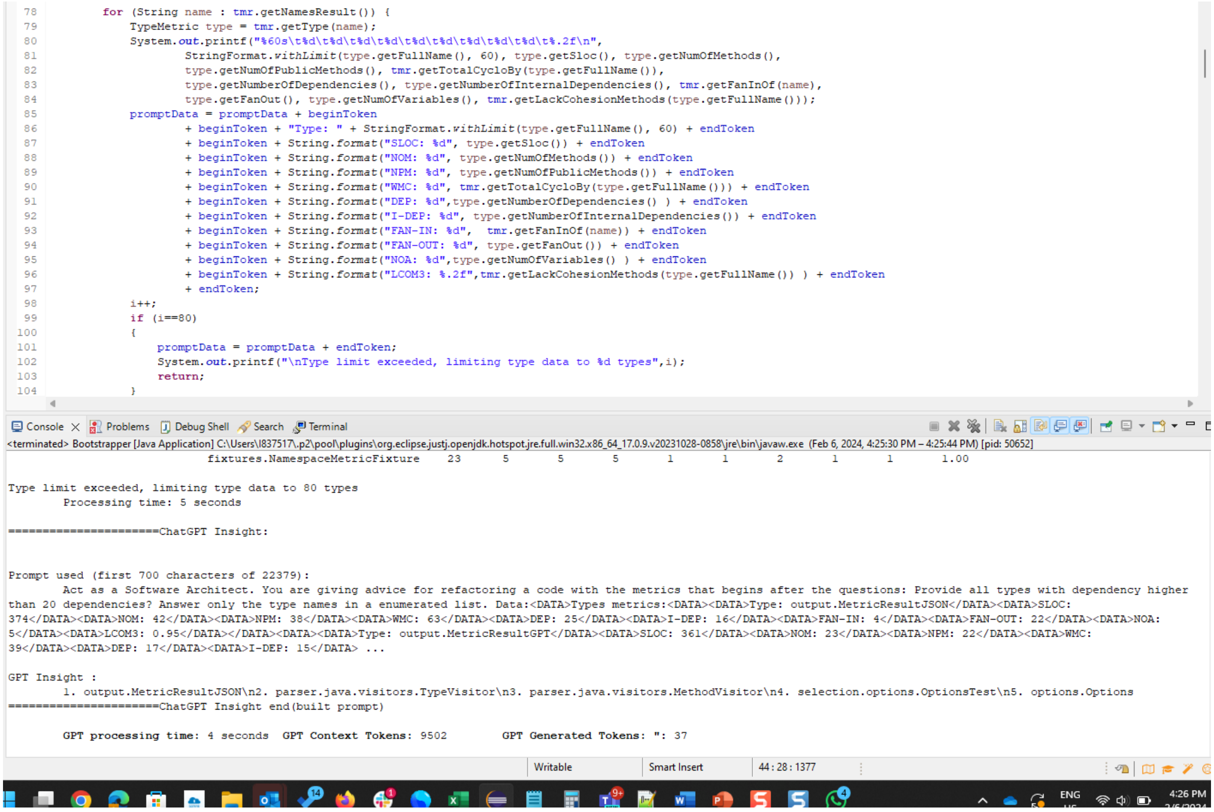
Source: Elaborated by the author

Figure A.3.29 – ChatGPT with all type metrics, limiting data to 80 types (classes)



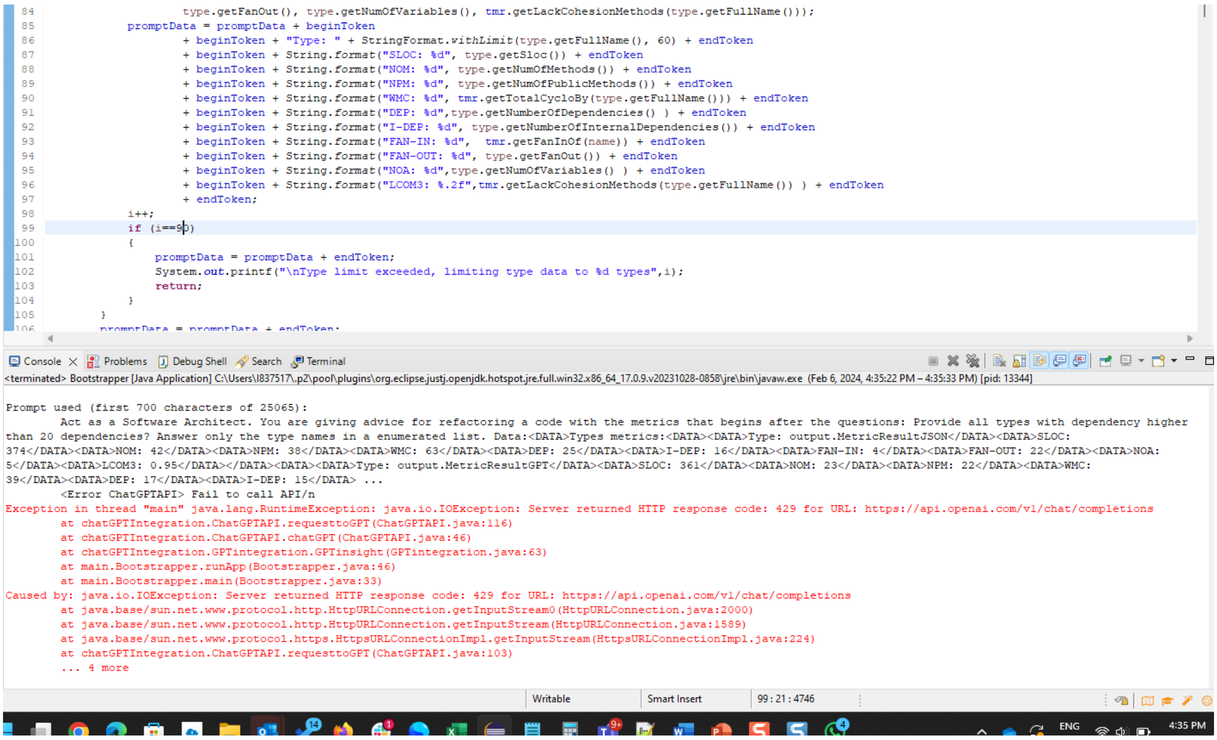
Source: Elaborated by the author

Figure A.3.30 – ChatGPT with all type metrics, limiting data to 80 types (classes)



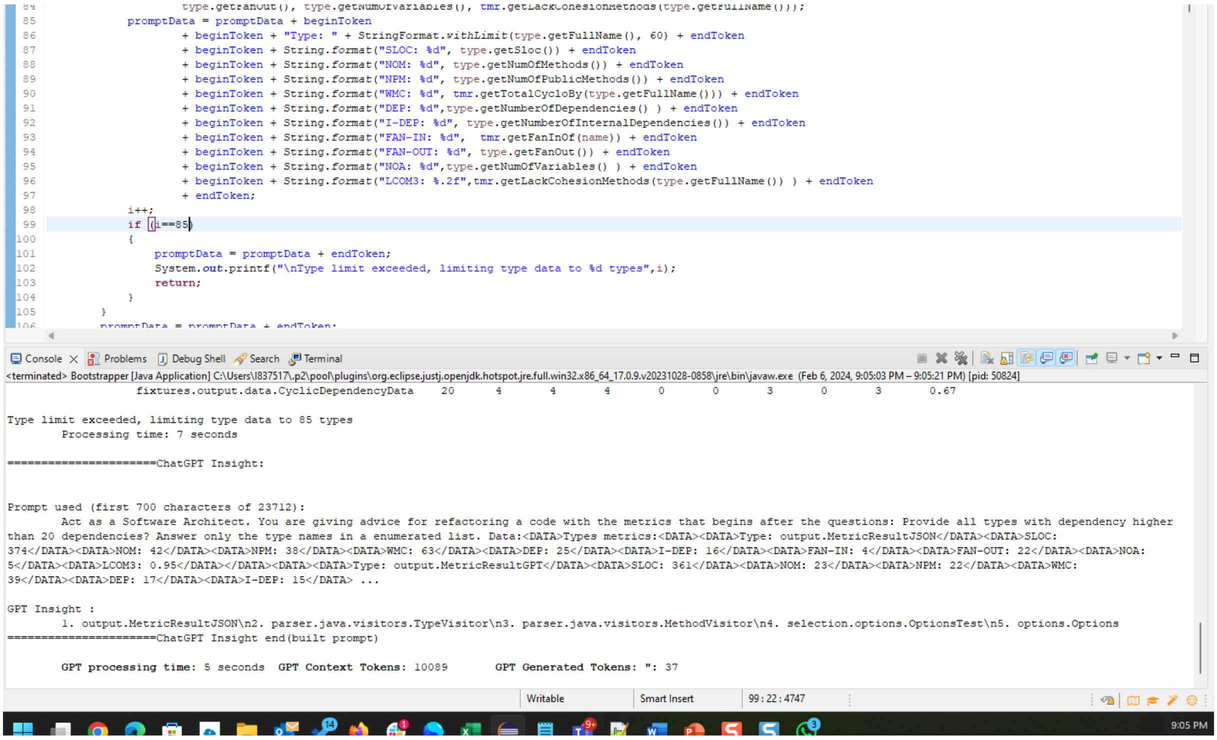
Source: Elaborated by the author

Figure A.3.31 – ChatGPT with all type metrics, limiting data to 90 types (classes) (failing)



Source: Elaborated by the author

Figure A.3.32 – ChatGPT with all type metrics, limiting data to 85 types (classes)



Source: Elaborated by the author

The OpenAI support was contacted, which was a chat bot, which provided us with the answer below and no solution:

"Hi there, Sorry you're running into issues regarding rate limits! Rate limits, which are restrictions we place on the number of API calls you can make, exist so we can make sure everyone has fair access to the API. If you're bumping up against these limits, here are some strategies you might try: Reduce max_tokens : Reducing max_tokens to match the size of your completions. Since max_tokens factor into your rate limit calculation, this adjustment might resolve the issue if your Current tokens used are exceeding your token Limit. Optimize Your Requests: Batch requests and employ strategies like exponential backoff along with other error mitigation tactics. Wait for 48 Hours: If you're a new pay-as-you-go user, be aware that we place daily rate limits during the first 48 hours. More details on your specific rate limits can be found here. Check Your Quota: Ensure you're not exceeding your monthly spending quota. If you need adjustments, you can do so through the quota increase form. Ensure you're on our Pay-As-You-Go-Plan: Update your billing with credit card details for the API Platform (not ChatGPT) here. Explore (or free trial users) are heavily restricted, regardless if you already have credits or grants in your account. Still encountering issues? You can request a rate limit increase by filling out our Rate Limit Increase form. Please note that this applies only to certain models, as gpt-4 and gpt-3.5-turbo-16k are currently capacity constrained and we can't offer increases today. If these steps don't resolve your issue, please provide more details, and I'll be glad to assist you further. Best,

OpenAI Team"

Following the test of the limit of the method that can be analyzed using <D> </D>:

Figure A.3.33 – Experiment to find the longest method, limiting to provide 700 methods (successful)

```
Method limit exceeded, limiting metric data to 700 methods of 1185 Methods
	Processing time: 11 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 104490):
	Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)</D><D>MLOC: 42</D><D>CYCLO:
10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)</D><D>MLOC: 28</D><D>CYCLO:
8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)</D><D>MLOC:
45</D><D>CYCLO: 7</D><D>CALLS: 3</D><D> ...

GPT Insight:
	chatGPTIntegration.GPTintegration.GPTinsight()


======================ChatGPT Insight end(built prompt)

	GPT processing time: 8 seconds  GPT Context Tokens: 46554	GPT Generated Tokens: ": 12
```

Source: Elaborated by the author

Figure A.3.34 – Experiment to find the longest method, limiting to provide 900 methods (successful)

```
Method limit exceeded, limiting metric data to 900 methods of 1185 Methods
	Processing time: 9 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 136773):
	Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)</D><D>MLOC: 42</D><D>CYCLO:
10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)</D><D>MLOC: 28</D><D>CYCLO:
8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)</D><D>MLOC:
45</D><D>CYCLO: 7</D><D>CALLS: 3</D><D> ...

GPT Insight:
	chatGPTIntegration.GPTintegration.GPTinsight()


======================ChatGPT Insight end(built prompt)

	GPT processing time: 10 seconds GPT Context Tokens: 60418	GPT Generated Tokens: ": 12
```

Source: Elaborated by the author

Figure A.3.35 – Experiment to find the longest method, limiting to provide 950 methods (successful)

```
Method limit exceeded, limiting metric data to 950 methods of 1185 Methods
	Processing time: 11 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 144031):
	Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)</D><D>MLOC: 42</D><D>CYCLO:
10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)</D><D>MLOC: 28</D><D>CYCLO:
8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)</D><D>MLOC:
45</D><D>CYCLO: 7</D><D>CALLS: 3</D><D> ...

GPT Insight:
	chatGPTIntegration.GPTintegration.GPTinsight()


======================ChatGPT Insight end(built prompt)

	GPT processing time: 11 seconds GPT Context Tokens: 63688	GPT Generated Tokens: ": 12
```

Source: Elaborated by the author

Figure A.3.36 – Experiment to find the longest method, limiting to provide 700 methods (successful)

```
Method limit exceeded, limiting metric data to 975 methods of 1185 Methods
        Processing time: 9 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 147819):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)</D><D>MLOC: 42</D><D>CYCLO:
10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)</D><D>MLOC: 28</D><D>CYCLO:
8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)</D><D>MLOC:
45</D><D>CYCLO: 7</D><D>CALLS: 3</D><D> ...

GPT Insight:
        chatGPTIntegration.GPTintegration.GPTinsight()


======================ChatGPT Insight end(built prompt)

        GPT processing time: 11 seconds GPT Context Tokens: 65346       GPT Generated Tokens: ": 12
```

Source: Elaborated by the author

Figure A.3.37 – Experiment to find the longest method, limiting to provide 985 methods (successful)

```
Method limit exceeded, limiting metric data to 985 methods of 1185 Methods
        Processing time: 8 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 149247):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)</D><D>MLOC: 42</D><D>CYCLO:
10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)</D><D>MLOC: 28</D><D>CYCLO:
8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)</D><D>MLOC:
45</D><D>CYCLO: 7</D><D>CALLS: 3</D><D> ...

GPT Insight:
        javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)


======================ChatGPT Insight end(built prompt)

        GPT processing time: 12 seconds GPT Context Tokens: 65991       GPT Generated Tokens: ": 17
```

Source: Elaborated by the author

Figure A.3.38 – Experiment to find the longest method, limiting to provide 100 methods (fail due to rate limit)

```
Method limit exceeded, limiting metric data to 1000 methods of 1185 Methods
        Processing time: 9 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 151512):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)</D><D>MLOC: 42</D><D>CYCLO:
10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)</D><D>MLOC: 28</D><D>CYCLO:
8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)</D><D>MLOC:
45</D><D>CYCLO: 7</D><D>CALLS: 3</D><D> ...

GPT Insight:

        <Error ChatGPTAPI> Fail on API request/nIf Error Code 429, likely exceeded the API limit or ChatGPT has performance issues. Try later.
Server returned HTTP response code: 400 for URL: https://api.openai.com/v1/chat/completions
Exception in thread "main" java.lang.RuntimeException: java.io.IOException: Server returned HTTP response code: 400 for URL: https://api.openai.com/v1/chat/completions
        at chatGPTIntegration.ChatGPTAPI.requesttoGPT(ChatGPTAPI.java:121)
        at chatGPTIntegration.ChatGPTAPI.chatGPT(ChatGPTAPI.java:49)
        at chatGPTIntegration.GPTintegration.printGPTResponse(GPTintegration.java:147)
        at chatGPTIntegration.GPTintegration.printInsight(GPTintegration.java:137)
        at chatGPTIntegration.GPTintegration.GPTinsight(GPTintegration.java:76)
```

Source: Elaborated by the author

Figure A.3.39 – Experiment to find types with higher than 20 dependencies, limiting to provide 100 types (successful)

```
Type limit exceeded, limiting type data to 100 types
        Processing time: 9 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 20728):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsProvide all types with
dependency higher than 20 dependencies? Answer only the type names in a enumerated list.Metrics data:<D>Types metrics:<D><D>Type: output.MetricResultJSON</D><D>SLOC:
374</D><D>NOM: 42</D><D>NPM: 38</D><D>WMC: 63</D><D>DEP: 25</D><D>I-DEP: 16</D><D>FAN-IN: 4</D><D>FAN-OUT: 22</D><D>NOA: 5</D><D>LCOM3: 0.95</D></D><D><D>Type:
output.MetricResultGPT</D><D>SLOC: 373</D><D>NOM: 24</D><D>NPM: 23</D><D>WMC: 41</D><D>DEP: 17</D><D>I-DEP: 15</D><D>FAN-IN: 1</D><D>FAN-OUT: 20</D><D>NOA: 8</D><D>LCOM3:
0.85</D></D><D><D>Type: structures.res ...

GPT Insight:
        Based on the provided data, here are the types with dependencies higher than 20:

1. output.MetricResultJSON
2. parser.java.visitors.MethodVisitor
3. selection.options.OptionsTest
4. selection.options.Options


======================ChatGPT Insight end(built prompt)

        GPT processing time: 5 seconds  GPT Context Tokens: 11871       GPT Generated Tokens: ": 46
```

Source: Elaborated by the author

Figure A.3.40– Experiment to find the longest method, limiting to provide 700 methods (successful)

```
Method limit exceeded, limiting metric data to 700 methods of 1185 Methods
        Processing time: 11 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 104490):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)</D><D>MLOC: 42</D><D>CYCLO:
10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)</D><D>MLOC: 28</D><D>CYCLO:
8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)</D><D>MLOC:
45</D><D>CYCLO: 7</D><D>CALLS: 3</D><D> ...

GPT Insight:
        chatGPTIntegration.GPTintegration.GPTinsight()


======================ChatGPT Insight end(built prompt)

        GPT processing time: 8 seconds  GPT Context Tokens: 46554       GPT Generated Tokens: ": 12
```

Source: Elaborated by the author

Figure A.3.41 – Experiment to find the longest method, limiting to provide 900 methods (successful)

```
Method limit exceeded, limiting metric data to 900 methods of 1185 Methods
        Processing time: 9 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 136773):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)</D><D>MLOC: 42</D><D>CYCLO:
10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)</D><D>MLOC: 28</D><D>CYCLO:
8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)</D><D>MLOC:
45</D><D>CYCLO: 7</D><D>CALLS: 3</D><D> ...

GPT Insight:
        chatGPTIntegration.GPTintegration.GPTinsight()


======================ChatGPT Insight end(built prompt)

        GPT processing time: 10 seconds GPT Context Tokens: 60418       GPT Generated Tokens: ": 12
```

Source: Elaborated by the author

Figure A.3.42 – Experiment to find the longest method, limiting to provide 950 methods (successful)

```
Method limit exceeded, limiting metric data to 950 methods of 1185 Methods
        Processing time: 11 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 144031):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)</D><D>MLOC: 42</D><D>CYCLO:
10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)</D><D>MLOC: 28</D><D>CYCLO:
8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)</D><D>MLOC:
45</D><D>CYCLO: 7</D><D>CALLS: 3</D><D> ...

GPT Insight:
        chatGPTIntegration.GPTintegration.GPTinsight()


======================ChatGPT Insight end(built prompt)

        GPT processing time: 11 seconds GPT Context Tokens: 63688       GPT Generated Tokens: ": 12
```

Source: Elaborated by the author

Figure A.3.43 – Experiment to find the longest method, limiting to provide 700 methods (successful)

```
Method limit exceeded, limiting metric data to 975 methods of 1185 Methods
        Processing time: 9 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 147819):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)</D><D>MLOC: 42</D><D>CYCLO:
10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)</D><D>MLOC: 28</D><D>CYCLO:
8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)</D><D>MLOC:
45</D><D>CYCLO: 7</D><D>CALLS: 3</D><D> ...

GPT Insight:
        chatGPTIntegration.GPTintegration.GPTinsight()


======================ChatGPT Insight end(built prompt)

        GPT processing time: 11 seconds GPT Context Tokens: 65346       GPT Generated Tokens: ": 12
```

Source: Elaborated by the author

Figure A.3.44 – Experiment to find the longest method, limiting to provide 985 methods (successful)

```
Method limit exceeded, limiting metric data to 985 methods of 1185 Methods
	Processing time: 8 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 149247):
	Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)</D><D>MLOC: 42</D><D>CYCLO:
10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)</D><D>MLOC: 28</D><D>CYCLO:
8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)</D><D>MLOC:
45</D><D>CYCLO: 7</D><D>CALLS: 3</D><D> ...

GPT Insight:
	javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)


======================ChatGPT Insight end(built prompt)

	GPT processing time: 12 seconds GPT Context Tokens: 65991	GPT Generated Tokens: ": 17
```

Source: Elaborated by the author

Figure A.3.45 – Experiment to find the longest method, limiting to provide 1000 methods (fail due to rate limit)

```
Method limit exceeded, limiting metric data to 1000 methods of 1185 Methods
	Processing time: 9 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 151512):
	Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)</D><D>MLOC: 42</D><D>CYCLO:
10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)</D><D>MLOC: 28</D><D>CYCLO:
8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)</D><D>MLOC:
45</D><D>CYCLO: 7</D><D>CALLS: 3</D><D> ...

GPT Insight:

	<Error ChatGPTAPI> Fail on API request/nIf Error Code 429, likely exceeded the API limit or ChatGPT has performance issues. Try later.
Server returned HTTP response code: 400 for URL: https://api.openai.com/v1/chat/completions
Exception in thread "main" java.lang.RuntimeException: java.io.IOException: Server returned HTTP response code: 400 for URL: https://api.openai.com/v1/chat/completions
	at chatGPTIntegration.ChatGPTAPI.requesttoGPT(ChatGPTAPI.java:121)
	at chatGPTIntegration.ChatGPTAPI.chatGPT(ChatGPTAPI.java:49)
	at chatGPTIntegration.GPTintegration.printGPTResponse(GPTintegration.java:147)
	at chatGPTIntegration.GPTintegration.printInsight(GPTintegration.java:137)
	at chatGPTIntegration.GPTintegration.GPTinsight(GPTintegration.java:76)
```

Source: Elaborated by the author

Figure A.3.46 – Experiment to find the longest method with name placeholder, limiting to provide 985 methods (successful)

```
Method limit exceeded, limiting metric data to 985 methods of 1185 Methods
	Processing time: 9 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 97906):
	Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: method1</D><D>MLOC: 42</D><D>CYCLO: 10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM:
4</D><D><D>Method: method2</D><D>MLOC: 28</D><D>CYCLO: 8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: method3</D><D>MLOC: 45</D><D>CYCLO: 7</D><D>CALLS:
3</D><D>NBD: 2</D><D>PARAM: 2</D><D><D>Method: method4</D><D>MLOC: 18</D><D>CYCLO: 6</D><D>CALLS: 5</D><D>NBD: 3</D><D>PARAM: 3</D><D><D>Method: method5</D><D>MLOC:
15</D><D>CYCLO: 6</D><D>CALLS: 8</D><D>NBD: 3 ...

GPT Insight:
	method570


======================ChatGPT Insight end(built prompt)

	GPT processing time: 6 seconds  GPT Context Tokens: 57188	GPT Generated Tokens: ": 2
```

Source: Elaborated by the author

Figure A.3.47– Experiment to find the longest method with name placeholder, limiting to provide 1100 methods (successful)

```
Method limit exceeded, limiting metric data to 1100 methods of 1185 Methods
	Processing time: 8 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 109392):
	Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: method1</D><D>MLOC: 42</D><D>CYCLO: 10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM:
4</D><D><D>Method: method2</D><D>MLOC: 28</D><D>CYCLO: 8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: method3</D><D>MLOC: 45</D><D>CYCLO: 7</D><D>CALLS:
3</D><D>NBD: 2</D><D>PARAM: 2</D><D><D>Method: method4</D><D>MLOC: 18</D><D>CYCLO: 6</D><D>CALLS: 5</D><D>NBD: 3</D><D>PARAM: 3</D><D><D>Method: method5</D><D>MLOC:
15</D><D>CYCLO: 6</D><D>CALLS: 8</D><D>NBD: 3 ...

GPT Insight:
	method570


======================ChatGPT Insight end(built prompt)

	GPT processing time: 7 seconds  GPT Context Tokens: 63959	GPT Generated Tokens: ": 2
```

Source: Elaborated by the author

Figure A.3.48 – Experiment to find the longest method with name placeholder, no number of method limitation (successful)

```
All methods analyzed, totalling 1185 methods
        Processing time: 11 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 117892):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the longest
method? Answer only 1 method nameMetrics data:<D>Method metrics:<D><D>Method: method1</D><D>MLOC: 42</D><D>CYCLO: 10</D><D>CALLS: 13</D><D>NBD: 4</D><D>PARAM:
4</D><D><D>Method: method2</D><D>MLOC: 28</D><D>CYCLO: 8</D><D>CALLS: 10</D><D>NBD: 4</D><D>PARAM: 4</D><D><D>Method: method3</D><D>MLOC: 45</D><D>CYCLO: 7</D><D>CALLS:
3</D><D>NBD: 2</D><D>PARAM: 2</D><D><D>Method: method4</D><D>MLOC: 18</D><D>CYCLO: 6</D><D>CALLS: 5</D><D>NBD: 3</D><D>PARAM: 3</D><D><D>Method: method5</D><D>MLOC:
15</D><D>CYCLO: 6</D><D>CALLS: 8</D><D>NBD: 3 ...

GPT Insight:
        method570


======================ChatGPT Insight end(built prompt)

        GPT processing time: 12 seconds GPT Context Tokens: 68974       GPT Generated Tokens: ": 2
```

Source: Elaborated by the author

3.2.2.3 Defining data structure for GPT-4 Turbo

Below follows the console generated from the experiments to determine the best data structure for GPT-4 Turbo.

Experiment using <BEGIN DATA TOKEN> <END DATA TOKEN>

```
--------------------------------------------------------------------------
-----------------------------------------------------------------
TYPES SLOC NOM NPM WMC DEP I-DEP FAN-IN FAN-OUT NOA LCOM3
--------------------------------------------------------------------------
-----------------------------------------------------------------
output.MetricResultJSON 374 42 38 63 25 16 4 22 5 0.95
output.MetricResultGPT 361 23 22 39 17 15 1 20 7 0.86
structures.results.TypeMetricResult 328 45 31 99 12 3 16 9 8 0.94
fixtures.output.JSONDataFixture 325 23 19 38 18 11 1 15 3 0.95
output.MetricResultCSV 282 34 34 45 17 15 4 22 4 0.95
fixtures.output.DataFixture 269 17 17 17 17 11 2 12 10 0.72
output.MetricResultConsole 263 23 22 38 17 15 3 19 4 0.93
javaProject.com.controller.Type 245 35 25 58 7 2 0 9 13 0.85
output.MetricResultFile 203 41 41 56 5 5 2 7 21 0.75
parser.java.visitors.TypeVisitor 189 19 12 42 21 3 1 9 16 0.58
parser.java.visitors.MethodVisitor 188 22 16 39 23 3 1 9 11 0.76
chatGPTIntegration.ChatGPTAPI 174 15 8 29 7 0 4 8 6 0.82
main.Bootstrapper 173 16 1 49 10 9 0 14 7 0.80
fixtures.output.CSVDataFixture 169 15 15 26 8 8 1 11 1 1.00
output.MetricResultFileTest 164 18 16 18 8 1 0 3 22 0.38
structures.metrics.TypeMetric 151 36 36 37 9 2 11 5 15 0.80
structures.statistics.StatisticOfType 144 15 13 16 6 4 5 6 3 0.93
structures.results.TypeMetricResultTest 135 19 17 23 8 3 0 6 2 0.97
output.MetricResultJSONTest 131 22 22 22 5 2 0 2 4 0.93
output.MetricResultCSVTest 111 18 18 18 5 2 0 2 4 0.91
output.utils.InfoConsole 109 13 9 15 0 0 6 3 1 1.00
selection.options.OptionsTest 104 20 19 20 22 19 0 20 0 0.00
utils.files.SourceCodeLineCounter 99 6 2 29 3 0 3 2 0 0.00
selection.ProjectInfoTest 96 17 15 19 10 5 0 6 3 0.94
structures.results.MethodMetricResult 96 15 14 25 9 2 18 6 3 0.93
structures.results.NamespaceMetricResult 94 17 16 26 9 2 15 5 3 0.94
structures.results.MethodMetricResultTest 89 12 10 15 7 3 0 4 2 0.95
structures.statistics.StatisticOfMethod 85 10 8 11 5 3 5 5 2 0.94
```

structures.statistics.namespaces.StatisticOfNamespaceTest 79 14 12 14 7 4 0 5 2 0.96
parser.java.JavaParser 74 6 2 7 16 7 1 11 6 0.50
structures.metrics.MethodMetric 73 18 18 19 1 0 11 2 8 0.79
structures.results.NamespaceMetricResultTest 72 12 10 13 7 3 0 4 2 0.95
output.MetricResultFake 70 21 21 21 3 3 0 4 0 0.00
selection.ProjectInfo 69 11 9 12 11 9 5 10 6 0.75
chatGPTIntegration.GPTintegration 68 4 2 11 9 9 1 15 7 0.00
utils.calc.StatisticalAnalysis 65 15 14 20 1 0 11 5 2 0.96
structures.results.StatisticMetricResult 62 14 14 14 0 0 8 0 2 0.96
structures.metrics.MetricThreshold 62 3 3 5 2 0 5 1 1 1.00
fixtures.output.data.TypeData 61 12 12 12 0 0 3 1 11 0.55
structures.statistics.StatisticalOperations 59 16 15 16 6 4 3 5 3 0.93
structures.statistics.methods.StatisticCallsOfMethodTest 56 12 12 12 4 1 0 2 0 0.00
structures.statistics.methods.StatisticCycloOfMethodTest 56 12 12 12 4 1 0 2 0 0.00
structures.statistics.methods.StatisticMlocOfMethodTest 56 12 12 12 4 1 0 2 0 0.00
structures.statistics.methods.StatisticNbdOfMethodTest 56 12 12 12 4 1 0 2 0 0.00
structures.statistics.methods.StatisticParamOfMethodTest 56 12 12 12 4 1 0 2 0 0.00
structures.statistics.types.StatisticDepOfTypeTest 56 12 12 12 4 1 0 2 0 0.00
structures.statistics.types.StatisticFanInOfTypeTest 56 12 12 12 4 1 0 2 0 0.00
structures.statistics.types.StatisticFanOutOfTypeTest 56 12 12 12 4 1 0 2 0 0.00
structures.statistics.types.StatisticIDepOfTypeTest 56 12 12 12 4 1 0 2 0 0.00
structures.statistics.types.StatisticLcom3OfTypeTest 56 12 12 12 4 1 0 2 0 0.00

Type limit exceeded, limiting type data to 50 types
Processing time: 5 seconds

======================ChatGPT Insight:

Prompt used (first 700 characters of 26770):
Act as a Software Architect. You are giving advice for refactoring a code with the metrics that begins after the questions: Provide all types with dependency higher than 20 dependencies? Answer only the type names in a enumerated list. Data:<BEGIN DATA TOKEN>Types metrics:<BEGIN DATA TOKEN><BEGIN DATA TOKEN>Type: output.MetricResultJSON<END DATA TOKEN><BEGIN DATA TOKEN>SLOC: 374<END DATA TOKEN><BEGIN DATA TOKEN>NOM: 42<END DATA TOKEN><BEGIN DATA TOKEN>NPM: 38<END DATA TOKEN><BEGIN DATA TOKEN>WMC: 63<END DATA TOKEN><BEGIN DATA TOKEN>DEP: 25<END DATA TOKEN><BEGIN DATA TOKEN>I-DEP: 16<END DATA TOKEN><BEGIN DATA TOKEN>FAN-IN: 4<END DATA TOKEN><BEGIN DATA TOKEN>FAN-OUT: 22<END DATA TOKEN><BEGIN D ...

GPT Insight(Test 1):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
=====================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m8372 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 2):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens:** [0m8372 **[1mGPT Generated Tokens:** [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 3):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens:** [0m8372 **[1mGPT Generated Tokens:** [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 4):
Based on the provided data, here are all types with dependencies higher than 20:\n\n1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens:** [0m8372 **[1mGPT Generated Tokens:** [0m": 40

[1mWaited time: [0m60 seconds

GPT Insight(Test 5):
1. parser.java.visitors.TypeVisitor\n2. parser.java.visitors.MethodVisitor\n3. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens:** [0m8372 **[1mGPT Generated Tokens:** [0m": 24

[1mWaited time: [0m60 seconds

GPT Insight(Test 6):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens:** [0m8372 **[1mGPT Generated Tokens:** [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 7):

1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m8372 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 8):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m8372 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 9):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m7 seconds **[1mGPT Context Tokens**: [0m8372 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 10):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selections.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m8372 **[1mGPT Generated Tokens**: [0m": 32

Experiment using <DATA> limited 50 types using

```
---------------------------------------------------------------------------
------------------------------------------------------------------
TYPES SLOC NOM NPM WMC DEP I-DEP FAN-IN FAN-OUT NOA LCOM3
---------------------------------------------------------------------------
------------------------------------------------------------------
output.MetricResultJSON 374 42 38 63 25 16 4 22 5 0.95
output.MetricResultGPT 361 23 22 39 17 15 1 20 7 0.86
structures.results.TypeMetricResult 328 45 31 99 12 3 16 9 8 0.94
fixtures.output.JSONDataFixture 325 23 19 38 18 11 1 15 3 0.95
output.MetricResultCSV 282 34 34 45 17 15 4 22 4 0.95
fixtures.output.DataFixture 269 17 17 17 17 11 2 12 10 0.72
output.MetricResultConsole 263 23 22 38 17 15 3 19 4 0.93
javaProject.com.controller.Type 245 35 25 58 7 2 0 9 13 0.85
output.MetricResultFile 203 41 41 56 5 5 2 7 21 0.75
parser.java.visitors.TypeVisitor 189 19 12 42 21 3 1 9 16 0.58
parser.java.visitors.MethodVisitor 188 22 16 39 23 3 1 9 11 0.76
chatGPTIntegration.ChatGPTAPI 174 15 8 29 7 0 4 8 6 0.82
main.Bootstrapper 173 16 1 49 10 9 0 14 7 0.80
```

```
fixtures.output.CSVDataFixture 169 15 15 26 8 8 1 11 1 1.00
output.MetricResultFileTest 164 18 16 18 8 1 0 3 22 0.38
structures.metrics.TypeMetric 151 36 36 37 9 2 11 5 15 0.80
structures.statistics.StatisticOfType 144 15 13 16 6 4 5 6 3 0.93
structures.results.TypeMetricResultTest 135 19 17 23 8 3 0 6 2 0.97
output.MetricResultJSONTest 131 22 22 22 5 2 0 2 4 0.93
output.MetricResultCSVTest 111 18 18 18 5 2 0 2 4 0.91
output.utils.InfoConsole 109 13 9 15 0 0 6 3 1 1.00
selection.options.OptionsTest 104 20 19 20 22 19 0 20 0 0.00
utils.files.SourceCodeLineCounter 99 6 2 29 3 0 3 2 0 0.00
selection.ProjectInfoTest 96 17 15 19 10 5 0 6 3 0.94
structures.results.MethodMetricResult 96 15 14 25 9 2 18 6 3 0.93
structures.results.NamespaceMetricResult 94 17 16 26 9 2 15 5 3 0.94
structures.results.MethodMetricResultTest 89 12 10 15 7 3 0 4 2 0.95
structures.statistics.StatisticOfMethod 85 10 8 11 5 3 5 5 2 0.94
structures.statistics.namespaces.StatisticOfNamespaceTest 79 14 12 14 7 4 0
5 2 0.96
parser.java.JavaParser 74 6 2 7 16 7 1 11 6 0.50
structures.metrics.MethodMetric 73 18 18 19 1 0 11 2 8 0.79
structures.results.NamespaceMetricResultTest 72 12 10 13 7 3 0 4 2 0.95
output.MetricResultFake 70 21 21 21 3 3 0 4 0 0.00
selection.ProjectInfo 69 11 9 12 11 9 5 10 6 0.75
utils.calc.StatisticalAnalysis 65 15 14 20 1 0 11 5 2 0.96
structures.results.StatisticMetricResult 62 14 14 14 0 0 8 0 2 0.96
structures.metrics.MetricThreshold 62 3 3 5 2 0 5 1 1 1.00
fixtures.output.data.TypeData 61 12 12 12 0 0 3 1 11 0.55
chatGPTIntegration.GPTintegration 61 4 2 9 9 9 1 15 7 0.00
structures.statistics.StatisticalOperations 59 16 15 16 6 4 3 5 3 0.93
structures.statistics.methods.StatisticCallsOfMethodTest 56 12 12 12 4 1 0
2 0 0.00
structures.statistics.methods.StatisticCycloOfMethodTest 56 12 12 12 4 1 0
2 0 0.00
structures.statistics.methods.StatisticMlocOfMethodTest 56 12 12 12 4 1 0 2
0 0.00
structures.statistics.methods.StatisticNbdOfMethodTest 56 12 12 12 4 1 0 2
0 0.00
structures.statistics.methods.StatisticParamOfMethodTest 56 12 12 12 4 1 0
2 0 0.00
structures.statistics.types.StatisticDepOfTypeTest 56 12 12 12 4 1 0 2 0
0.00
structures.statistics.types.StatisticFanInOfTypeTest 56 12 12 12 4 1 0 2 0
0.00
structures.statistics.types.StatisticFanOutOfTypeTest 56 12 12 12 4 1 0 2 0
0.00
structures.statistics.types.StatisticIDepOfTypeTest 56 12 12 12 4 1 0 2 0
0.00
structures.statistics.types.StatisticLcom3OfTypeTest 56 12 12 12 4 1 0 2 0
0.00

Type limit exceeded, limiting type data to 50 types
Processing time: 4 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 14148):
Act as a Software Architect. You are giving advice for refactoring a code
with the metrics that begins after the questions: Provide all types with
dependency higher than 20 dependencies? Answer only the type names in a
enumerated list. Data:<DATA>Types metrics:<DATA><DATA>Type:
```

output.MetricResultJSON</DATA><DATA>SLOC: 374</DATA><DATA>NOM: 42</DATA><DATA>NPM: 38</DATA><DATA>WMC: 63</DATA><DATA>DEP: 25</DATA><DATA>I-DEP: 16</DATA><DATA>FAN-IN: 4</DATA><DATA>FAN-OUT: 22</DATA><DATA>NOA: 5</DATA><DATA>LCOM3: 0.95</DATA></DATA><DATA><DATA>Type: output.MetricResultGPT</DATA><DATA>SLOC: 361</DATA><DATA>NOM: 23</DATA><DATA>NPM: 22</DATA><DATA>WMC: 39</DATA><DATA>DEP: 17</DATA><DATA>I-DEP: 15</DATA> ...

GPT Insight(Test 1):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

GPT Insight(Test 2):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

GPT Insight(Test 3):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

GPT Insight(Test 4):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

GPT Insight(Test 5):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

GPT Insight(Test 6):
1. parser.java.visitors.TypeVisitor\n2. parser.java.visitors.MethodVisitor\n3. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 24

GPT Insight(Test 7):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

GPT Insight(Test 8):
1. parser.java.visitors.MethodVisitor\n2. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 15

GPT Insight(Test 9):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

GPT Insight(Test 10):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

Experiment using <D></D> for 50 types using

Type limit exceeded, limiting type data to 50 types
Processing time: 5 seconds

======================ChatGPT Insight:

Prompt used (first 700 characters of 10543):
Act as a Software Architect. You are giving advice for refactoring a code with the metrics that begins after the questions: Provide all types with dependency higher than 20 dependencies? Answer only the type names in a enumerated list. Data:<D>Types metrics:<D><D>Type: output.MetricResultJSON</D><D>SLOC: 374</D><D>NOM: 42</D><D>NPM: 38</D><D>WMC: 63</D><D>DEP: 25</D><D>I-DEP: 16</D><D>FAN-IN: 4</D><D>FAN-OUT: 22</D><D>NOA: 5</D><D>LCOM3: 0.95</D></D><D><D>Type: output.MetricResultGPT</D><D>SLOC: 361</D><D>NOM: 23</D><D>NPM: 22</D><D>WMC: 39</D><D>DEP: 17</D><D>I-DEP: 15</D><D>FAN-IN: 1</D><D>FAN-OUT: 20</D><D>NOA: 7</D><D>LCOM3: 0.86</D></D><D><D>Type: structures.results.TypeMetricResult</D> ...

GPT Insight(Test 1):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 2):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 3):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 4):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 5):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 6):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

```
GPT Insight(Test 7):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

```
GPT Insight(Test 8):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

```
GPT Insight(Test 9):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

```
GPT Insight(Test 10):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens**: [0m5968 **[1mGPT Generated Tokens**: [0m": 32

Experiment using separator “|” for 50 types:

```
Type limit exceeded, limiting type data to 50 types
Processing time: 4 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 7538):
Act as a Software Architect. You are giving advice for refactoring a code
with the metrics that begins after the questions: Provide all types with
dependency higher than 20 dependencies? Answer only the type names in a
```

```
enumerated list. Data:|Types metrics:||Type: output.MetricResultJSON||SLOC:
374||NOM: 42||NPM: 38||WMC: 63||DEP: 25||I-DEP: 16||FAN-IN: 4||FAN-OUT:
22||NOA: 5||LCOM3: 0.95||||Type: output.MetricResultGPT||SLOC: 361||NOM:
23||NPM: 22||WMC: 39||DEP: 17||I-DEP: 15||FAN-IN: 1||FAN-OUT: 20||NOA:
7||LCOM3: 0.86||||Type: structures.results.TypeMetricResult||SLOC:
328||NOM: 45||NPM: 31||WMC: 99||DEP: 12||I-DEP: 3||FAN-IN: 16||FAN-OUT:
9||NOA: 8||LCOM3: 0.94||||Type: fixtures.output.JS ...

GPT Insight(Test 1):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m3917 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m30 seconds

```
GPT Insight(Test 2):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3917 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m30 seconds

```
GPT Insight(Test 3):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m3917 **[1mGPT Generated Tokens**: [0m": 16

[1mWaited time: [0m30 seconds

```
GPT Insight(Test 4):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m3917 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m30 seconds

```
GPT Insight(Test 5):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3917 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m30 seconds

```
GPT Insight(Test 6):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3.
selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3917 **[1mGPT Generated Tokens**: [0m": 23

[1mWaited time: [0m30 seconds

```
GPT Insight(Test 7):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3.
selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m3917 **[1mGPT Generated Tokens**: [0m": 23

[1mWaited time: [0m30 seconds

```
GPT Insight(Test 8):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3.
selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m3917 **[1mGPT Generated Tokens**: [0m": 23

[1mWaited time: [0m30 seconds

```
GPT Insight(Test 9):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3.
selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3917 **[1mGPT Generated Tokens**: [0m": 23

[1mWaited time: [0m30 seconds

```
GPT Insight(Test 10):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m3917 **[1mGPT Generated Tokens**: [0m": 32

Experiment using “ “ (space) as separator for 50 types using:

```
======================ChatGPT Insight:


Prompt used (first 700 characters of 7538):
```

Act as a Software Architect. You are giving advice for refactoring a code with the metrics that begins after the questions: Provide all types with dependency higher than 20 dependencies? Answer only the type names in a enumerated list. Data: Types metrics: Type: output.MetricResultJSON SLOC: 374 NOM: 42 NPM: 38 WMC: 63 DEP: 25 I-DEP: 16 FAN-IN: 4 FAN-OUT: 22 NOA: 5 LCOM3: 0.95 Type: output.MetricResultGPT SLOC: 361 NOM: 23 NPM: 22 WMC: 39 DEP: 17 I-DEP: 15 FAN-IN: 1 FAN-OUT: 20 NOA: 7 LCOM3: 0.86 Type: structures.results.TypeMetricResult SLOC: 328 NOM: 45 NPM: 31 WMC: 99 DEP: 12 I-DEP: 3 FAN-IN: 16 FAN-OUT: 9 NOA: 8 LCOM3: 0.94 Type: fixtures.output.JS ...

GPT Insight(Test 1):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3865 **[1mGPT Generated Tokens**: [0m": 23

[1mWaited time: [0m10 seconds

GPT Insight(Test 2):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3865 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m10 seconds

GPT Insight(Test 3):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3865 **[1mGPT Generated Tokens**: [0m": 23

[1mWaited time: [0m10 seconds

GPT Insight(Test 4):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m3865 **[1mGPT Generated Tokens**: [0m": 23

[1mWaited time: [0m10 seconds

GPT Insight(Test 5):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3865 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m10 seconds

GPT Insight(Test 6):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m3865 **[1mGPT Generated Tokens**: [0m": 23

[1mWaited time: [0m10 seconds

GPT Insight(Test 7):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3865 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m10 seconds

GPT Insight(Test 8):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3865 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m10 seconds

GPT Insight(Test 9):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m3865 **[1mGPT Generated Tokens**: [0m": 23

[1mWaited time: [0m10 seconds

GPT Insight(Test 10):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3865 **[1mGPT Generated Tokens**: [0m": 32

Experiment not using any separator:

```
Type limit exceeded, limiting type data to 50 types
Processing time: 5 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 6336):
Act as a Software Architect. You are giving advice for refactoring a code
with the metrics that begins after the questions: Provide all types with
dependency higher than 20 dependencies? Answer only the type names in a
enumerated list. Data:Types metrics:Type: output.MetricResultJSONSLOC:
374NOM: 42NPM: 38WMC: 63DEP: 25I-DEP: 16FAN-IN: 4FAN-OUT: 22NOA: 5LCOM3:
0.95Type: output.MetricResultGPTSLOC: 361NOM: 23NPM: 22WMC: 39DEP: 17I-DEP:
15FAN-IN: 1FAN-OUT: 20NOA: 7LCOM3: 0.86Type:
structures.results.TypeMetricResultSLOC: 328NOM: 45NPM: 31WMC: 99DEP: 12I-
DEP: 3FAN-IN: 16FAN-OUT: 9NOA: 8LCOM3: 0.94Type:
fixtures.output.JSONDataFixtureSLOC: 325NOM: 23NPM: 19WMC: 38DEP: 18I-DEP:
11FAN-IN: 1FAN-OUT ...

GPT Insight(Test 1):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds [1mGPT Context Tokens: [0m3363 [1mGPT
Generated Tokens: [0m": 32

[1mWaited time: [0m60 seconds


GPT Insight(Test 2):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds [1mGPT Context Tokens: [0m3363 [1mGPT
Generated Tokens: [0m": 32

[1mWaited time: [0m60 seconds


GPT Insight(Test 3):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds [1mGPT Context Tokens: [0m3363 [1mGPT
Generated Tokens: [0m": 32

[1mWaited time: [0m60 seconds


GPT Insight(Test 4):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3363 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 5):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3363 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 6):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3363 **[1mGPT Generated Tokens**: [0m": 23

[1mWaited time: [0m60 seconds

GPT Insight(Test 7):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3363 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m60 seconds

GPT Insight(Test 8):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3363 **[1mGPT Generated Tokens**: [0m": 23

[1mWaited time: [0m60 seconds

GPT Insight(Test 9):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m3363 **[1mGPT Generated Tokens**: [0m": 23

[1mWaited time: [0m60 seconds

GPT Insight(Test 10):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m3363 **[1mGPT Generated Tokens**: [0m": 32

Extended test with 50 executions for <D></D>:

======================ChatGPT Insight:

Prompt used (first 700 characters of 26310):
Act as a Software Architect. You are giving advice for refactoring a code with the metrics that begins after the questions: Provide all types with dependency higher than 20 dependencies? Answer only the type names in a enumerated list. Data:<D>Types metrics:<D><D>Type: output.MetricResultJSON</D><D>SLOC: 374</D><D>NOM: 42</D><D>NPM: 38</D><D>WMC: 63</D><D>DEP: 25</D><D>I-DEP: 16</D><D>FAN-IN: 4</D><D>FAN-OUT: 22</D><D>NOA: 5</D><D>LCOM3: 0.95</D></D><D><D>Type: output.MetricResultGPT</D><D>SLOC: 361</D><D>NOM: 23</D><D>NPM: 22</D><D>WMC: 39</D><D>DEP: 17</D><D>I-DEP: 15</D><D>FAN-IN: 1</D><D>FAN-OUT: 20</D><D>NOA: 7</D><D>LCOM3: 0.86</D></D><D><D>Type: structures.results.TypeMetricResult</D> ...

GPT Insight(Test 1):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m6 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 2):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 3):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 4):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 5):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 6):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 7):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 8):
1. parser.java.visitors.TypeVisitor\n2. parser.java.visitors.MethodVisitor\n3. selection.options.Options\n4. output.MetricResultJSON\n
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 32

[1mWaited time: [0m4 seconds

GPT Insight(Test 9):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m6 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 10):
1. parser.java.visitors.MethodVisitor\n2. parser.java.visitors.TypeVisitor\n3. selections.options.Options\n4. selections.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 30

[1mWaited time: [0m4 seconds

GPT Insight(Test 11):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 22

[1mWaited time: [0m4 seconds

GPT Insight(Test 12):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 13):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 14):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m11 seconds **[1mGPT Context Tokens:** [0m15163 **[1mGPT Generated Tokens:** [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 15):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m9 seconds **[1mGPT Context Tokens:** [0m15163 **[1mGPT Generated Tokens:** [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 16):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m7 seconds **[1mGPT Context Tokens:** [0m15163 **[1mGPT Generated Tokens:** [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 17):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens:** [0m15163 **[1mGPT Generated Tokens:** [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 18):
1. parser.java.visitors.TypeVisitor\n2. parser.java.visitors.MethodVisitor\n3. selection.options.Options\n4. output.MetricResultJSON
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens:** [0m15163 **[1mGPT Generated Tokens:** [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 19):

1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m6 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 20):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 21):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 22):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 23):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 24):

```
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5.
selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 38

[1mWaited time: [0m4 seconds

```
GPT Insight(Test 25):
1. parser.java.visitors.MethodVisitor\n2.
parser.java.visitors.TypeVisitor\n3. selection.options.Options\n4.
output.MetricResultJSON
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

```
GPT Insight(Test 26):
1. parser.java.visitors.TypeVisitor\n2.
parser.java.visitors.MethodVisitor\n3. selection.options.Options\n4.
selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 30

[1mWaited time: [0m4 seconds

```
GPT Insight(Test 27):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

```
GPT Insight(Test 28):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

```
GPT Insight(Test 29):
```

1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens:** [0m15163 **[1mGPT Generated Tokens:** [0m": 37

[1mWaited time: [0m4 seconds

GPT Insight(Test 30):
1. parser.java.visitors.TypeVisitor\n2. parser.java.visitors.MethodVisitor\n3. selection.options.Options\n4. output.MetricResultJSON
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens:** [0m15163 **[1mGPT Generated Tokens:** [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 31):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens:** [0m15163 **[1mGPT Generated Tokens:** [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 32):
1. parser.java.visitors.TypeVisitor\n2. parser.java.visitors.MethodVisitor\n3. selection.options.OptionsTest\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens:** [0m15163 **[1mGPT Generated Tokens:** [0m": 30

[1mWaited time: [0m4 seconds

GPT Insight(Test 33):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds **[1mGPT Context Tokens:** [0m15163 **[1mGPT Generated Tokens:** [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 34):

1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selections.options.Options\n5. selections.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 35):
1. parser.java.visitors.MethodVisitor\n2. selection.options.Options\n3. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 21

[1mWaited time: [0m4 seconds

GPT Insight(Test 36):
1. parser.java.visitors.MethodVisitor\n2. parser.java.visitors.TypeVisitor\n3. selection.options.Options\n4. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 30

[1mWaited time: [0m4 seconds

GPT Insight(Test 37):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 38):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 38

[1mWaited time: [0m4 seconds

GPT Insight(Test 39):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 40):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 41):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. options.Selection
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 36

[1mWaited time: [0m4 seconds

GPT Insight(Test 42):
1. parser.java.visitors.MethodVisitor\n2. selection.options.Options\n3. parser.java.visitors.TypeVisitor
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 23

[1mWaited time: [0m4 seconds

GPT Insight(Test 43):
1. output.MetricResultJSON\n2. parser.java.visitors.MethodVisitor\n3. selection.options.Options\n4. parser.java.visitors.TypeVisitor
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 44):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 45):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 46):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m7 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 47):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 36

[1mWaited time: [0m4 seconds

GPT Insight(Test 48):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 31

[1mWaited time: [0m4 seconds

GPT Insight(Test 49):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3. parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5. selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m15163 **[1mGPT Generated Tokens**: [0m": 38

[1mWaited time: [0m4 seconds

```
GPT Insight(Test 50):
1. output.MetricResultJSON\n2. parser.java.visitors.TypeVisitor\n3.
parser.java.visitors.MethodVisitor\n4. selection.options.Options\n5.
selection.options.OptionsTest
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m5 seconds [1mGPT Context Tokens: [0m15163 [1mGPT
Generated Tokens: [0m": 38
```

Experiment counting types with more than 20 dependencies:

Prompt:

```
                  this.gpt.setSystemPrompt("Act as a Software Architect
giving refactoring advice by answering the user questions based on the
following code metric(The metrics are structured with <D> to start the data
structure and </D>):"
                              + output.returnPromptData());
                  prompt = "How many types have dependency (DEP) higher
than 20? Reply just the number.";
```

```
======================ChatGPT Insight:


Prompt used (complete):
How many types have dependency (DEP) higher than 20? Reply just the number.


GPT Insight(Test 1):
5
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds [1mGPT Context Tokens: [0m15177 [1mGPT
Generated Tokens: [0m": 1

[1mWaited time: [0m70 seconds


GPT Insight(Test 2):
5
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds [1mGPT Context Tokens: [0m15177 [1mGPT
Generated Tokens: [0m": 1

[1mWaited time: [0m70 seconds


GPT Insight(Test 3):
4
======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m1 seconds **[1mGPT Context Tokens:** [0m15177 **[1mGPT Generated Tokens:** [0m": 1

[1mWaited time: [0m70 seconds

GPT Insight(Test 4):
4
======================ChatGPT Insight end(built prompt)

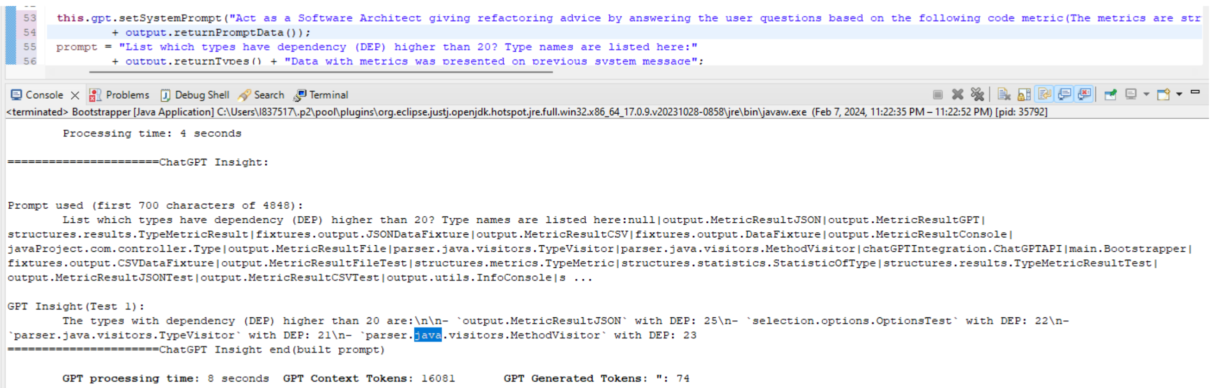
[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens:** [0m15177 **[1mGPT Generated Tokens:** [0m": 1

[1mWaited time: [0m70 seconds

GPT Insight(Test 5):
4
======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens:** [0m15177 **[1mGPT Generated Tokens:** [0m": 1

[1mWaited time: [0m70 seconds



Final experiment using placeholder technique, which pushed the result to 100% success rate:

Starting
Directory valid

TYPES SLOC NOM NPM WMC DEP I-DEP FAN-IN FAN-OUT NOA LCOM3

----
output.MetricResultJSON type1 374 42 38 63 25 16 4 22 5 0.95
output.MetricResultGPT type2 374 24 23 41 17 15 1 20 8 0.85
structures.results.TypeMetricResult type3 328 45 31 99 12 3 16 9 8 0.94
fixtures.output.JSONDataFixture type4 325 23 19 38 18 11 1 15 3 0.95
output.MetricResultCSV type5 282 34 34 45 17 15 4 22 4 0.95
fixtures.output.DataFixture type6 269 17 17 17 17 11 2 12 10 0.72
output.MetricResultConsole type7 263 23 22 38 17 15 3 19 4 0.93
javaProject.com.controller.Type type8 245 35 25 58 7 2 0 9 13 0.85
output.MetricResultFile type9 203 41 41 56 5 5 2 7 21 0.75
parser.java.visitors.TypeVisitor type10 189 19 12 42 21 3 1 9 16 0.58

```
parser.java.visitors.MethodVisitor type11 188 22 16 39 23 3 1 9 11 0.76
main.Bootstrapper type12 176 16 1 50 10 9 0 14 7 0.80
chatGPTIntegration.ChatGPTAPI type13 176 15 8 30 7 0 4 8 6 0.82
fixtures.output.CSVDataFixture type14 169 15 15 26 8 8 1 11 1 1.00
output.MetricResultFileTest type15 164 18 16 18 8 1 0 3 22 0.38
structures.metrics.TypeMetric type16 151 36 36 37 9 2 11 5 15 0.80
structures.statistics.StatisticOfType type17 144 15 13 16 6 4 5 6 3 0.93
structures.results.TypeMetricResultTest type18 135 19 17 23 8 3 0 6 2 0.97
output.MetricResultJSONTest type19 131 22 22 22 5 2 0 2 4 0.93
output.MetricResultCSVTest type20 111 18 18 18 5 2 0 2 4 0.91
output.utils.InfoConsole type21 109 13 9 15 0 0 6 3 1 1.00
selection.options.OptionsTest type22 104 20 19 20 22 19 0 20 0 0.00
utils.files.SourceCodeLineCounter type23 99 6 2 29 3 0 3 2 0 0.00
selection.ProjectInfoTest type24 96 17 15 19 10 5 0 6 3 0.94
structures.results.MethodMetricResult type25 96 15 14 25 9 2 18 6 3 0.93
structures.results.NamespaceMetricResult type26 94 17 16 26 9 2 15 5 3 0.94
structures.results.MethodMetricResultTest type27 89 12 10 15 7 3 0 4 2 0.95
structures.statistics.StatisticOfMethod type28 85 10 8 11 5 3 5 5 2 0.94
chatGPTIntegration.GPTintegration type29 83 5 2 15 9 9 1 15 7 0.25
structures.statistics.namespaces.StatisticOfNamespaceTest type30 79 14 12
14 7 4 0 5 2 0.96
parser.java.JavaParser type31 74 6 2 7 16 7 1 11 6 0.50
structures.metrics.MethodMetric type32 73 18 18 19 1 0 11 2 8 0.79
structures.results.NamespaceMetricResultTest type33 72 12 10 13 7 3 0 4 2
0.95
output.MetricResultFake type34 70 21 21 21 3 3 0 4 0 0.00
selection.ProjectInfo type35 69 11 9 12 11 9 5 10 6 0.75
utils.calc.StatisticalAnalysis type36 65 15 14 20 1 0 11 5 2 0.96
structures.results.StatisticMetricResult type37 62 14 14 14 0 0 8 0 2 0.96
structures.metrics.MetricThreshold type38 62 3 3 5 2 0 5 1 1 1.00
fixtures.output.data.TypeData type39 61 12 12 12 0 0 3 1 11 0.55
structures.statistics.StatisticalOperations type40 59 16 15 16 6 4 3 5 3
0.93
structures.statistics.methods.StatisticCallsOfMethodTest type41 56 12 12 12
4 1 0 2 0 0.00
structures.statistics.methods.StatisticCycloOfMethodTest type42 56 12 12 12
4 1 0 2 0 0.00
structures.statistics.methods.StatisticMlocOfMethodTest type43 56 12 12 12
4 1 0 2 0 0.00
structures.statistics.methods.StatisticNbdOfMethodTest type44 56 12 12 12 4
1 0 2 0 0.00
structures.statistics.methods.StatisticParamOfMethodTest type45 56 12 12 12
4 1 0 2 0 0.00
structures.statistics.types.StatisticDepOfTypeTest type46 56 12 12 12 4 1 0
2 0 0.00
structures.statistics.types.StatisticFanInOfTypeTest type47 56 12 12 12 4 1
0 2 0 0.00
structures.statistics.types.StatisticFanOutOfTypeTest type48 56 12 12 12 4
1 0 2 0 0.00
structures.statistics.types.StatisticIDepOfTypeTest type49 56 12 12 12 4 1
0 2 0 0.00
structures.statistics.types.StatisticLcom3OfTypeTest type50 56 12 12 12 4 1
0 2 0 0.00
structures.statistics.types.StatisticNoaOfTypeTest type51 56 12 12 12 4 1 0
2 0 0.00
structures.statistics.types.StatisticNomOfTypeTest type52 56 12 12 12 4 1 0
2 0 0.00
structures.statistics.types.StatisticNpmOfTypeTest type53 56 12 12 12 4 1 0
2 0 0.00
```

```
structures.statistics.types.StatisticSlocOfTypeTest type54 56 12 12 12 4 1
0 2 0 0.00
structures.statistics.types.StatisticWmcOfTypeTest type55 56 12 12 12 4 1 0
2 0 0.00
fixtures.output.data.StatisticData type56 54 12 12 12 0 0 3 0 2 0.95
utils.calc.OutlierAnalysisTest type57 54 11 11 11 3 0 0 0 2 0.95
fixtures.TypeMetricFixture type58 52 12 12 12 2 1 2 1 1 1.00
selection.options.Options type59 50 2 2 2 21 19 2 19 1 1.00
fixtures.statistics.StatisticOfTypeFixture type60 48 3 0 3 7 5 10 7 2 0.75
chatGPTIntegration.promptEngineer type61 46 4 2 12 1 1 1 2 2 0.83
output.MetricResultDOT type62 45 7 2 11 3 2 2 4 3 0.83
fixtures.MethodMetricFixture type63 43 10 10 10 1 1 2 1 1 1.00
utils.files.StringFormatTest type64 40 8 8 8 2 0 0 0 0 0.00
utils.files.SystemUtils type65 39 3 2 3 7 0 4 4 0 0.00
structures.statistics.StatisticOfNamespace type66 37 5 4 6 5 3 5 5 3 0.75
structures.metrics.NamespaceMetric type67 35 8 8 10 0 0 10 2 3 0.86
fixtures.output.data.MethodData type68 35 7 7 7 0 0 3 0 6 0.58
fixtures.output.data.NamespaceCouplingData type69 35 7 7 7 0 0 3 0 6 0.58
fixtures.output.data.MetricThresholdData type70 33 7 7 7 0 0 3 0 5 0.67
structures.metrics.MetricDefinition type71 30 6 6 6 0 0 30 0 5 0.60
fixtures.output.data.SummaryData type72 30 6 6 6 0 0 3 0 5 0.60
utils.calc.StatisticalAnalysisTest type73 30 5 5 5 3 0 0 0 2 0.88
fixtures.statistics.StatisticOfMethodFixture type74 30 3 0 3 4 4 5 5 2 0.75
output.MetricOutput type75 29 23 23 23 3 3 31 3 0 0.00
javaProject.com.model.Man type76 29 5 5 9 1 0 1 2 0 0.00
utils.files.JSONBuilder type77 28 6 6 6 2 0 2 0 1 1.00
javaProject.com.controller.Dispatcher type78 27 4 4 4 4 4 1 4 3 0.67
utils.files.SourceCodeLineCounterTest type79 26 2 2 2 7 0 0 0 0 0.00
parser.java.visitors.NamespaceVisitor type80 24 3 3 6 5 3 1 5 1 1.00
fixtures.NamespaceMetricFixture type81 23 5 5 5 1 1 2 1 1 1.00
output.utils.Gauge type82 23 3 3 7 2 2 1 3 1 1.00
utils.files.SystemUtilsTest type83 22 2 2 2 5 0 0 1 2 0.50
utils.files.StringFormat type84 20 4 4 9 0 0 6 1 0 0.00
fixtures.output.data.CyclicDependencyData type85 20 4 4 4 0 0 3 0 3 0.67
fixtures.output.data.NamespaceData type86 20 4 4 4 0 0 3 0 3 0.67
fixtures.output.data.TypeResonanceData type87 20 4 4 4 0 0 2 0 3 0.67
javaProject.com.model.Woman type88 16 3 3 5 0 0 1 1 0 0.00
fixtures.output.data.NamespaceDependencyData type89 15 3 3 3 0 0 2 0 2 0.75
output.MetricFile type90 14 11 11 11 0 0 3 0 0 0.00
selection.options.dependencies.TypeCouplingOption type91 11 1 1 1 2 2 2 3 0
0.00
selection.options.general.AllMetricsOption type92 11 1 1 1 2 2 2 3 0 0.00
selection.options.general.MetricVisualizationOption type93 11 1 1 1 2 2 2 3
0 0.00
selection.options.statistics.StatisticAndTypeOption type94 11 1 1 1 2 2 2 3
0 0.00
selection.options.statistics.StatisticTypeOption type95 11 1 1 1 2 2 2 3 0
0.00
selection.options.strutures.TypeOption type96 11 1 1 1 2 2 2 3 0 0.00
javaProject.com.model.Human type97 10 2 2 2 0 0 2 1 1 1.00
selection.options.dependencies.AllCouplingOption type98 10 1 1 1 2 2 2 3 0
0.00
selection.options.dependencies.CyclicDependencyOption type99 10 1 1 1 2 2 2
3 0 0.00
selection.options.dependencies.InternalDependencyOption type100 10 1 1 1 2
2 2 3 0 0.00
selection.options.dependencies.NamespaceCouplingOption type101 10 1 1 1 2 2
2 3 0 0.00
selection.options.statistics.StatisticAndNamespaceOption type102 10 1 1 1 2
2 2 3 0 0.00
```

```
selection.options.statistics.StatisticNamespaceOption type103 10 1 1 1 2 2
2 3 0 0.00
selection.options.strutures.NamespaceOption type104 10 1 1 1 2 2 2 3 0 0.00
javaProject.com.view.QueueViewer type105 10 0 1 2 1 1 0 2 0 0.00
selection.options.dependencies.DependencyOption type106 9 1 1 1 2 2 2 3 0
0.00
selection.options.general.SummaryOption type107 9 1 1 1 2 2 2 3 0 0.00
selection.options.general.ThresholdsOption type108 9 1 1 1 2 2 2 3 0 0.00
selection.options.statistics.StatisticAndMethodOption type109 9 1 1 1 2 2 2
3 0 0.00
selection.options.statistics.StatisticMethodOption type110 9 1 1 1 2 2 2 3
0 0.00
selection.options.strutures.MethodOption type111 9 1 1 1 2 2 2 3 0 0.00
javaProject.com.controller.ClassWithComments type112 8 1 1 1 0 0 0 0 0 0.00
structures.MetricResultNotifier type113 7 3 3 3 1 0 3 0 0 0.00
javaProject.com.model.Child type114 7 1 1 1 0 0 1 1 1 0.00
javaProject.com.controller.XClass type115 6 1 1 1 0 0 0 0 0 0.00
javaProject.others.AnalysisContext type116 6 1 1 1 0 0 0 0 0 0.00
javaProject.others.ClassVertex type117 6 1 1 1 0 0 1 0 0 0.00
selection.options.OptionDefinition type118 5 1 1 1 1 1 19 1 0 0.00
structures.MetricActivator type119 5 1 1 1 0 0 3 0 0 0.00
javaProject.one.A type120 5 0 0 1 1 1 1 1 1 0.00
javaProject.two.B type121 5 0 0 1 1 1 1 1 1 0.00
javaProject.com.model.Person type122 4 1 1 1 0 0 3 0 0 0.00
chatGPTIntegration.GPT type123 4 1 1 1 0 0 1 0 0 0.00
output.MetricGPT type124 4 1 1 1 0 0 1 0 0 0.00
parser.TypeParser type125 4 1 1 1 0 0 2 0 0 0.00
javaProject.com.controller.XMethod type126 3 0 0 1 0 0 0 0 0 0.00
javaProject.others.ClassDescriptor type127 3 0 0 1 0 0 1 0 0 0.00
javaProject.others.ObjectType type128 3 0 0 1 0 0 0 0 0 0.00

All types analyzed, totalling 128 types
Processing time: 8 seconds
```

Experiment with replacing type names with placeholder (dictionary technique).

On attempt to improve the results of the data analysis, it was used a technique to replace the name with a placeholder of format typeX, which x will be an increasing integer. Below is a test with 100 iterations to have this technique effectiveness measured (which was 100% of the 100 successful).

```
======================ChatGPT Insight:


Prompt used (first 700 characters of 22804):
Act as a Software Architect, providing advice to a new developer on what to
do on a refactoring, based on the Data provided after the requestsProvide
all types with dependency higher than 20 dependencies? Answer only the type
names in a enumerated list.Metrics data:<D>Types metrics:<D><D>Method:
type1</D><D>SLOC: 374</D><D>NOM: 42</D><D>NPM: 38</D><D>WMC: 63</D><D>DEP:
25</D><D>I-DEP: 16</D><D>FAN-IN: 4</D><D>FAN-OUT: 22</D><D>NOA:
5</D><D>LCOM3: 0.95</D></D><D><D>Method: type2</D><D>SLOC: 374</D><D>NOM:
24</D><D>NPM: 23</D><D>WMC: 41</D><D>DEP: 17</D><D>I-DEP: 15</D><D>FAN-IN:
```

1</D><D>FAN-OUT: 20</D><D>NOA: 8</D><D>LCOM3: 0.85</D></D><D><D>Method: type3</D><D>SLOC: 328</D><D>NOM: 45</D><D>N ...

GPT Insight(Test 1):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 2):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 3):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m1 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 4):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 5):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 6):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 7):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 8):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 9):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 10):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 11):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 12):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m1 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 13):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m1 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 14):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 15):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 16):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 17):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 18):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 19):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 20):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 21):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens:** [0m14787 **[1mGPT Generated Tokens:** [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 22):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens:** [0m14787 **[1mGPT Generated Tokens:** [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 23):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens:** [0m14787 **[1mGPT Generated Tokens:** [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 24):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens:** [0m14787 **[1mGPT Generated Tokens:** [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 25):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 26):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 27):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 28):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 29):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1000 milliseconds

GPT Insight(Test 30):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 31):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 32):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 33):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 34):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 35):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 36):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 37):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 38):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m1 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 39):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m1 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 40):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m1 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 41):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m1 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 42):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 43):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m3 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 44):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m4 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 45):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 46):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 47):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 48):
1. type1
2. type10
3. type11
4. type22
5. type59

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 49):
1. type1
2. type10
3. type11
4. type22
5. type59

=====================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

[1mWaited time: [0m1 seconds

GPT Insight(Test 50):
1. type1
2. type10
3. type11
4. type22
5. type59

=====================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m2 seconds **[1mGPT Context Tokens**: [0m14787 **[1mGPT Generated Tokens**: [0m": 24

3.2.3 PROMPT ENGINEERING STRUCTURE

Experiment without answer limitation, which showed prolix answer

Figure A.3.49 – Extract from experiment on DR-Tools data on prompt that shows need to limit the answer

```
------------------
SUMMARY OF METRICS
------------------
           Total of Namespaces: 32
                Total of Types: 128 - 4.00 (number of types/namespaces - median: 3.00 - std dev: 2.89)
                 Total of SLOC: 8185 - 63.95 (number of SLOC/types - median: 38.00 - std dev: 77.80)
              Total of Methods: 1182 - 9.23 (number of methods/types - median: 3.00 - std dev: 7.38)
                Total of CYCLO: 1544 - 12.06 (number of CYCLO/types)
        Processing time: 4 seconds

======================ChatGPT Insight (built prompt):


Prompt used (first 600 characters of 907):
        Act as a Software Architect. You are giving advise for refactoring for a code with the following metrics begining after the question, what is higher the average of
line of code per type or its median? <BEGIN DATA TOKEN>SUMMARY OF METRICS:  <BEGIN DATA TOKEN>Total of Namespaces: 32<END DATA TOKEN>  <BEGIN DATA TOKEN>Total of Types:
128<END DATA TOKEN>  <BEGIN DATA TOKEN>Number of types/namespaces: 4.00 (median: 3.00 - std dev: 2.89')<END DATA TOKEN>  <BEGIN DATA TOKEN>'Total of SLOC: 8185'<END DATA
TOKEN>  <BEGIN DATA TOKEN>'Number of SLOC/types: 63.95 (median: 38.00 - std dev: 77.80)')<END DAT ...
GPT Insight:
        Based on the provided metrics, let's answer your question before moving on to the advice for refactoring.\n\n### Calculating the Average vs. Median of Lines of Code
per Type\n\n- **Total Number of Lines of Code (SLOC):** 8,185\n- **Total Number of Types:** 128\n\n**Average lines of code (LOC) per type** can be calculated as total SLOC
divided by total types:\n\\[ \\text{Average} = \\frac{\\text{Total SLOC}}{\\text{Total Types}} = \\frac{8185}{128} \\approx 63.95 \\]\n\n**Median of lines of code per type**
is given as 38.00.\n\nBetween the **average** (63.95) and the **median** (38.00), the **average is higher**.\n\n### Advice for Refactoring\n\nGiven the statistics and the
disparity between the average and the median LOC per type, it's clear that while many types have a relatively low LOC, there are outliers with significantly higher LOC
contributing to the average being much higher. This indicates a skewed distribution, where a few types contain a lot more code than the rest. Here's how to proceed with
refactoring:\n\n- **Identify Large Classes/Types:** Start by identifying the outliers—types with an exceptionally high number of LOC. These are likely candidates for
decomposition.\n\n- **Apply the Single Responsibility Principle (SRP):** Ensure that each type has a single responsibility and split those that handle more than one
responsibility. This often involves breaking down large classes into smaller, more manageable, and more cohesive classes.\n\n- **Increase Modularity:** Aim to increase
modularity by separating concerns and improving encapsulation. This will not only simplify the codebase but also enhance maintainability and scalability.\n\n- **Refactor
Methods:** Just as with types, identify methods with a high Cyclomatic Complexity (CYCLO) and a high number of lines. Refactor these methods by breaking them down into
smaller, more focused methods, and apply the SRP.\n\n- **Improve Code Reuse:** Check for duplicated code across types. Refactor these by pulling the common code into separate
methods or types, which can promote reuse and reduce the total LOC.\n\n- **Automated Testing:** With any refactoring, it's essential to have a comprehensive suite of
automated tests to ensure that changes don't introduce regressions. If the test coverage is lacking, prioritize increasing it before substantial refactoring.\n\n- **Regular
Code Reviews:** Implement or enhance code review processes to ensure adherence to clean coding standards and architectural guidelines. Peer reviews can catch potential design
flaws early and foster a culture of continuous improvement.\n\nRemember, refactoring is an iterative process. Make incremental changes and keep measuring the impact on your
code metrics. Aim for improvements in readability, maintainability, and scalability without sacrificing functionality.

======================ChatGPT Insight end(built prompt)

        GPT processing time: 40 seconds GPT Context Tokens: 289 GPT Generated Tokens: ": 576
```

Source: Elaborated by the author

When limited the size of the answer, it gives the answer directly as in the extract “The average of lines of code per type is higher.” Therefore, it will always be specified a size of an answer that is consistent with the size expected of the advice that would be considered the best.

First experiment with request advice using summary metrics

The first experiment with request for refactoring advise was simply to ask repeated time refactoring advice based on summary metrics:

Figure A.3.50 – Experiment request areas of focus for the refactoring based on summary metrics

```
======================ChatGPT Insight:


Prompt used (first 700 characters of 840):
        Act as a Software Architect, providing advice to a new developer on a refactoring project, based on the summary metrics provided after <D>. Provide which activity
should be the focus of the refactoringWhat should be the areas of focus to the developer to refactor this code based on the Metrics? Answer breaked into 2 linesMetrics
data:<D>SUMMARY OF METRICS: <D>Total of Namespaces: 32</D><D>Total of Types: 128</D><D>Number of types/namespaces<D>Average: 4.00</D><D>Median: 3.00</D><D>Standard Deviation:
2.89</D></D><D>Total of SLOC: 8233</D><D>Number of SLOC/types<D>Average: 64.32</D><D>Median: 38.00</D><D>Standard deviation: 78.08</D></D><D>Total of Methods:
1185</D><D>Number of methods/types: ...

GPT Insight:
        Based on the provided summary metrics, there are several insights that can guide the refactoring focus for the developer. Breaking them down:

1. **Code Organization and Modularity**:
   - With an **average of 4 types per namespace** but a median of 3, there's a hint that while most namespaces contain a manageable number of types, there are potentially a
few namespaces with a significantly higher count. This disparity can indicate uneven distribution and potential issues with modularity and separation of concerns.
   - **Action**: Evaluate the namespaces with a higher than average number of types for potential refactoring into more cohesive and smaller namespaces, enhancing modularity
and maintainability.

2. **Source Lines of Code (SLOC) Concerns**:
   - The **average SLOC per type is 64.32**, with a median of 38 but a large standard deviation (78.08). This suggests a wide variation in the size of types, with some types
possibly being too large and complex, potentially violating the Single Responsibility Principle (SRP).
   - **Action**: Focus on types with significantly higher SLOC than the median, breaking down complex types into more manageable and focused components, potentially improving
readability, testability, and maintainability.

3. **Methodological Refinement**:
   - The metrics suggest an **average of 9.26 methods per type**, but with a median of just 3 and a standard deviation of 6.57. This discrepancy implies that while most types
have a few methods, some have a greatly inflated number, which can be a red flag for types taking on too many responsibilities.
   - **Action**: Identify types with a high number of methods to evaluate their responsibilities. Consider applying principles like SRP to refactor types, possibly by
splitting them into smaller, well-defined types or by extracting methods into new types.

4. **Cyclomatic Complexity (CYCLO)**:
   - An average of **12.19 CYCLO per type** might indicate that some methods or types are too complex, making them hard to understand, test, and maintain.
   - **Action**: Prioritize examining types and methods with high cyclomatic complexity scores for simplification. Refactor complex conditional logic into smaller, more
manageable methods, utilize design patterns where appropriate to handle complex behavior, and aim to reduce branching as much as possible.

**Overall Refactoring Strategy**:
A balanced approach targeting both organizational structure (namespaces and types) and code quality (SLOC, methods per type, and cyclomatic complexity) should be the
developer's focus during the refactoring process. Special attention should be given to identifying outliers that significantly deviate from the median values, as these are
likely candidates for immediate improvement.


======================ChatGPT Insight end(built prompt)

        GPT processing time: 31 seconds GPT Context Tokens: 290 GPT Generated Tokens: ": 550
```

Source: Elaborated by the author

Below follows the text version of the extract:

```
------------------
SUMMARY OF METRICS
------------------
Total of Namespaces: 32
Total of Types: 128 - 4.00 (number of types/namespaces - median: 3.00 - std
dev: 2.89)
Total of SLOC: 8233 - 64.32 (number of SLOC/types - median: 38.00 - std
dev: 78.08)
Total of Methods: 1185 - 9.26 (number of methods/types - median: 3.00 - std
dev: 6.57)
Total of CYCLO: 1560 - 12.19 (number of CYCLO/types)
Processing time: 10 seconds

======================ChatGPT Insight:


Prompt used (first 700 characters of 840):
Act as a Software Architect, providing advice to a new developer on a
refactoring project, based on the summary metrics provided after <D>.
Provide which activity should be the focus of the refactoringWhat should be
the areas of focus to the developer to refactor this code based on the
Metrics? Answer breaked into 2 linesMetrics data:<D>SUMMARY OF METRICS:
<D>Total of Namespaces: 32</D><D>Total of Types: 128</D><D>Number of
types/namespaces<D>Average: 4.00</D><D>Median: 3.00</D><D>Standard
Deviation: 2.89</D></D><D>Total of SLOC: 8233</D><D>Number of
SLOC/types<D>Average: 64.32</D><D>Median: 38.00</D><D>Standard deviation:
78.08</D></D><D>Total of Methods: 1185</D><D>Number of methods/types: ...

GPT Insight:
Based on the provided summary metrics, there are several insights that can
guide the refactoring focus for the developer. Breaking them down:
```

1. **Code Organization and Modularity**:
- With an **average of 4 types per namespace** but a median of 3, there's a hint that while most namespaces contain a manageable number of types, there are potentially a few namespaces with a significantly higher count. This disparity can indicate uneven distribution and potential issues with modularity and separation of concerns.
- **Action**: Evaluate the namespaces with a higher than average number of types for potential refactoring into more cohesive and smaller namespaces, enhancing modularity and maintainability.

2. **Source Lines of Code (SLOC) Concerns**:
- The **average SLOC per type is 64.32**, with a median of 38 but a large standard deviation (78.08). This suggests a wide variation in the size of types, with some types possibly being too large and complex, potentially violating the Single Responsibility Principle (SRP).
- **Action**: Focus on types with significantly higher SLOC than the median, breaking down complex types into more manageable and focused components, potentially improving readability, testability, and maintainability.

3. **Methodological Refinement**:
- The metrics suggest an **average of 9.26 methods per type**, but with a median of just 3 and a standard deviation of 6.57. This discrepancy implies that while most types have a few methods, some have a greatly inflated number, which can be a red flag for types taking on too many responsibilities.
- **Action**: Identify types with a high number of methods to evaluate their responsibilities. Consider applying principles like SRP to refactor types, possibly by splitting them into smaller, well-defined types or by extracting methods into new types.

4. **Cyclomatic Complexity (CYCLO)**:
- An average of **12.19 CYCLO per type** might indicate that some methods or types are too complex, making them hard to understand, test, and maintain.
- **Action**: Prioritize examining types and methods with high cyclomatic complexity scores for simplification. Refactor complex conditional logic into smaller, more manageable methods, utilize design patterns where appropriate to handle complex behavior, and aim to reduce branching as much as possible.

Overall Refactoring Strategy:
A balanced approach targeting both organizational structure (namespaces and types) and code quality (SLOC, methods per type, and cyclomatic complexity) should be the developer's focus during the refactoring process. Special attention should be given to identifying outliers that significantly deviate from the median values, as these are likely candidates for immediate improvement.

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m31 seconds **[1mGPT Context Tokens**: [0m290 **[1mGPT Generated Tokens**: [0m": 550

Initial experiment with type metrics analysis:

Then it was tested the same with type metrics, below follows the console answer to type metrics.

```
======================ChatGPT Insight:


Prompt used (first 700 characters of 26346):
Act as a Software Architect, providing advice to a new developer on what to
do on a refactoring, based on the Data provided after the requestsWhat
should be the areas of focus to the developer to refactor this code based
on the Metrics? Answer breaked into 2 linesMetrics data:<D>Types
metrics:<D><D>Type: output.MetricResultJSON</D><D>SLOC: 374</D><D>NOM:
42</D><D>NPM: 38</D><D>WMC: 63</D><D>DEP: 25</D><D>I-DEP: 16</D><D>FAN-IN:
4</D><D>FAN-OUT: 22</D><D>NOA: 5</D><D>LCOM3: 0.95</D></D><D><D>Type:
output.MetricResultGPT</D><D>SLOC: 366</D><D>NOM: 24</D><D>NPM:
23</D><D>WMC: 40</D><D>DEP: 17</D><D>I-DEP: 15</D><D>FAN-IN: 1</D><D>FAN-
OUT: 20</D><D>NOA: 8</D><D>LCOM3: 0.85</D></D><D><D>Type: str ...

GPT Insight(Test 1):
Given the comprehensive metrics data provided, the refactoring efforts
should target the following areas to enhance code quality, maintainability,
and performance:

1. **High Lack of Cohesion (LCOM3 values close to 1)**: Focus on types such
as `fixtures.output.CSVDataFixture`, `TypeMetricFixture`,
`NamespaceVisitor`, and several types with LCOM3 = 1.00. This suggests that
the methods within these types are not working together, which may indicate
that the type is doing too much or its responsibilities are not well
aligned. Consider splitting these types into more cohesive units or
reevaluating their responsibilities.

2. **High Number of Methods (NOM) and Weighted Method Count (WMC)**:
Inspect types such as `structures.results.TypeMetricResult` and
`output.MetricResultFile` which have high NOM and WMC values. This could
indicate complex classes that are trying to do too much. Applying the
Single Responsibility Principle (SRP) can help in breaking down these
classes into smaller, more focused classes.

Refactoring these areas will likely have a significant positive impact on
the maintainability and understandability of the codebase.


======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m12 seconds **[1mGPT Context Tokens**: [0m15174
[1mGPT Generated Tokens: [0m": 238

[1mWaited time: [0m80 seconds

```
GPT Insight(Test 2):
Based on the provided metrics data from various types, here are the key
takeaways and areas of focus for refactoring:

1. **Reduce High Cyclomatic Complexity (WMC):** Look for types with high
Weighted Methods per Class (WMC) values, indicating complex code that could
```

benefit from simplification. Examples include `Type: structures.results.TypeMetricResult` with `WMC: 99` and `Type: javaProject.com.controller.Type` with `WMC: 58`. Reducing complexity makes the code more maintainable and easier to understand.

2. **Address High Lack of Cohesion of Methods (LCOM3):** High LCOM3 values suggest that classes could be doing too much or the class’s methods and fields are not related closely enough, which can lead to poor class cohesion. For example, `Type: structures.results.TypeMetricResultTest` and `Type: output.MetricResultFake` both have `LCOM3: 0.95`. Consider refactoring these classes by segregating unrelated functionalities into separate classes or by enhancing the existing methods and fields to work more harmoniously.

---

3. **Optimize Fan-Out and Dependency Metrics:** High Fan-Out values, such as those seen in `Type: output.MetricResultJSON` (`FAN-OUT: 22`) and `Type: output.MetricResultCSV` (`FAN-OUT: 22`), indicate that a type depends on many other types. This could signify overly tight coupling. Look into possibilities to introduce interfaces, or perhaps dependency inversion principles, to reduce direct dependencies.

4. **Improve Method and Class Design:** Examine types with a high Number of Methods (NOM) but low usage of class fields (NOA), e.g., `Type: output.MetricResultFile` (`NOM: 41, NOA: 21`) and `Type: structures.metrics.TypeMetric` (`NOM: 36, NOA: 15`). It may indicate methods could be refactored into more cohesive, smaller classes, or there are opportunities to make better use of class-level fields, reducing parameter passing and improving encapsulation.

Refactoring with these focus areas not only aims at enhancing code readability and maintainability but also at improving the overall architecture's scalability and robustness.

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m34 seconds **[1mGPT Context Tokens**: [0m15174
[1mGPT Generated Tokens: [0m": 468

[1mWaited time: [0m80 seconds

GPT Insight(Test 3):
Given the complexity and volume of metrics provided for various types, the first line of advice would revolve around understanding and identifying key areas where refactoring could yield significant improvements in maintainability, performance, and code quality. With this extensive data, certain metrics immediately stand out and should guide the initial focus of refactoring efforts:

1. **Cohesion (LCOM3):** Types with high LCOM3 values close to 1 (e.g., `fixtures.output.data.CSVDataFixture` with LCOM3 of 1.00, and several others) indicate poor cohesion, meaning that the methods within these types are not well related. Focusing on increasing cohesion by ensuring that classes have a single responsibility can significantly improve the codebase. Refactoring suggestions include splitting large classes into smaller, more focused ones or moving methods to classes where they are more relevant.

2. **Cyclomatic Complexity (WMC):** Types with high Weighted Methods per Class (WMC) indicate complex logic that can be hard to understand and test (e.g., `structures.results.TypeMetricResult` with a WMC of 99). Refactoring efforts should aim to simplify complex methods, possibly by breaking them down into smaller methods that are easier to understand and test. Techniques such as applying the Extract Method refactoring or simplifying conditional logic can be particularly useful here.

For the second line of advice, attention should be shifted towards dependencies and method metrics:

1. **Dependencies (DEP, I-DEP, FAN-IN, FAN-OUT):** High dependency metrics, especially FAN-OUT values, suggest that the type is overly dependent on many other types or components (e.g., `output.MetricResultJSON` with FAN-OUT of 22). This can make the system brittle and difficult to change. To minimize dependency issues, consider applying the Dependency Inversion Principle by refactoring towards depending on abstractions rather than concrete implementations. Additionally, look into partitioning the system into more decoupled, independently deployable units or modules.

2. **Method Metrics (NOM, NPM):** Types with a high Number of Methods (NOM) or Number of Public Methods (NPM) are indicative of potential \

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m49 seconds **[1mGPT Context Tokens**: [0m15174
[1mGPT Generated Tokens: [0m": 564

[1mWaited time: [0m80 seconds

GPT Insight(Test 4):
Given the comprehensive metrics data provided, the focus of refactoring should be on the areas of high Lack of Cohesion of Methods (LCOM3), high Source Lines of Code (SLOC), high Weighted Method Count (WMC), and excessive dependencies (DEP and I-DEP). To start, areas with a high LCOM3 value close to 1 indicate poor cohesion within a class, suggesting that the class is doing too much or the class's methods are not well related. High SLOC indicates potential complexity and difficulty in maintenance, while high WMC points towards complex classes that may be trying to perform too many actions, increasing the risk of bugs.

Firstly, attention should be directed towards reducing the complexity and enhancing the maintainability of the classes with high SLOC counted above. Refactoring strategies might include the extraction of methods or classes (if a class is performing too many functions), modularizing the code better, and identifying common patterns or behaviors that can be abstracted or unified.

Secondly, reducing the Lack of Cohesion of Methods (LCOM3) by ensuring that methods within a class are more tightly related to each other should be a priority. This often involves rethinking the responsibilities assigned to each class and possibly redistributing methods among classes or introducing new classes to group related functionalities together more logically.

In refactoring efforts targeting these areas, unit tests play a crucial role in ensuring that behavior is preserved. Automated tests should be in place before starting the refactoring process to detect any inadvertent changes or regressions in functionality. It's also prudent to tackle these refactoring efforts incrementally, applying small, manageable changes and validating through testing after each modification.

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m30 seconds **[1mGPT Context Tokens**: [0m15174
[1mGPT Generated Tokens: [0m": 339

[1mWaited time: [0m80 seconds

GPT Insight(Test 5):
Given the comprehensive list of metric data for various types, here are key recommendations for areas of focus during refactoring:

1. **High Lack of Cohesion (LCOM3):** Several types exhibit a high Lack of Cohesion of Methods (LCOM3) score, close to 1.0, such as `fixtures.output.CSVDataFixture`, `structures.results.TypeMetricResultTest`, and `output.MetricResultFake`. These types should be refactored to improve cohesion. Aim to modularize functionality such that each class focuses on a single purpose or closely related purposes.

2. **High Fan-Out:** Certain types have a high Fan-Out, such as `output.MetricResultJSON` and `output.MetricResultCSV` with Fan-Outs of 22. High Fan-Out indicates a type's high dependency on other types, suggesting a potential need for decoupling and encapsulation improvements. Extract interfaces, use design patterns like Dependency Injection, or apply the Facade Pattern to reduce direct dependencies.

Continued areas of focus:

3. **Complexity Metrics (WMC) and Size Metrics (SLOC, NOM):** Types like `structures.results.TypeMetricResult` with WMC of 99, SLOC of 328, and NOM of 45 are likely to be complex and large, suggesting potential refactoring to break down large classes into smaller, more manageable ones. Aim for single responsibility and lower complexity to improve maintainability.

4. **High Internal Dependency (DEP & I-DEP):** Consider types with high external and internal dependencies, such as `parser.java.visitors.MethodVisitor` with DEP of 23 and `selection.options.Options` with I-DEP of 19. It's vital to assess these dependencies for possible reduction through architectural redesign, possibly by introducing more abstract layers or utilizing patterns that favor loose coupling.

By addressing these areas, the software architecture can evolve towards a more maintainable, scalable, and loosely coupled system. Refactoring efforts should prioritize readability, simplicity, and encapsulation, applying solid principles and design patterns where appropriate.

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m31 seconds **[1mGPT Context Tokens**: [0m15174
[1mGPT Generated Tokens: [0m": 424

[1mWaited time: [0m80 seconds

GPT Insight(Test 6):
Given the comprehensive set of metrics provided for a variety of types, the key focus areas for refactoring should prioritize addressing high complexity, reducing tight coupling, enhancing cohesion, and reducing code duplication. Specifically:

1. **High Complexity & Tight Coupling**: Target types with high WMC (Weighted Method Count) and high FAN-OUT values as they indicate complexity and tight coupling. For example, the `structures.results.TypeMetricResult` has a WMC of 99 and a FAN-OUT of 9, suggesting it could benefit from simplification and breaking down into smaller, more focused components.

2. **Low Cohesion**: Look for types with high LCOM3 (Lack of Cohesion of Methods) values, as this indicates that the methods within a class do not share much in common. Types like `output.MetricResultFake` with LCOM3 of 1.00 and `fixtures.TypeMetricFixture` with LCOM3 of 1.00 are prime candidates for refactoring to improve cohesion.

Continuing with guidance based on the provided metrics:

1. **Reduce Dependencies**: Types with high DEP (Direct dependencies) and I-DEP (Indirect dependencies) like `selection.options.Options` with DEP of 21 and I-DEP of 19, indicate a high reliance on other components, making the system more fragile and harder to maintain. Work on reducing these dependencies, possibly through interface abstraction or by applying the Dependency Inversion Principle.

2. **Improve Encapsulation**: NOA (Number of Attributes) figures suggest how many state variables a type is managing. For better encapsulation and to facilitate easier unit testing, consider refactoring types with a high NOA, like `output.MetricResultFile` with NOA of 21, by perhaps breaking them down into smaller classes each responsible for a more specific subset of functionality.

This strategic approach to refactoring should lead to improved maintainability, readability, and scalability of your codebase.

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m29 seconds **[1mGPT Context Tokens**: [0m15174 **[1mGPT Generated Tokens**: [0m": 404

[1mWaited time: [0m80 seconds

GPT Insight(Test 7):
Based on the provided metrics data, here are the key areas you should focus on for refactoring:

1. **Reduce High Complexity and Interdependence**: Look into types with high Weighted Method Count (WMC) and high dependence (DEP and I-DEP). For example, refactoring the types with the highest WMC like `structures.results.TypeMetricResult` (WMC: 99) and

`structures.metrics.TypeMetric` (WMC: 37) could improve maintainability. Aim to simplify complex methods and reduce coupling where possible.

2. **Address High Lack of Cohesion (LCOM3)**: Types with high Lack of Cohesion of Methods version 3 (LCOM3) values, near or equal to 1.00, indicate that the class methods and fields are not well related. Examples are `fixtures.output.data.CSVDataFixture` and `fixtures.TypeMetricFixture`, both with LCOM3 of 1.00, and several others. Consider breaking these types into smaller, more cohesive ones.

Continued in the next part...

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m20 seconds **[1mGPT Context Tokens**: [0m15174 **[1mGPT Generated Tokens**: [0m": 218

[1mWaited time: [0m80 seconds

GPT Insight(Test 8):
Given the substantial amount of metrics data for various types, the areas for refactoring can be effectively identified by focusing on certain key metrics indicative of code quality and maintainability. These metrics include SLOC (Source Lines of Code), NOM (Number of Methods), WMC (Weighted Methods per Class), DEP (Dependencies), FAN-IN/FAN-OUT, NOA (Number of Attributes), and notably LCOM3 (Lack of Cohesion of Methods version 3). Identifying these metrics helps in determining the complexity, coupling, and cohesion of the code, which are significant factors in software maintainability.

To start, the primary areas of focus for the developer to refactor, based on the metrics are:

1. **Reduce Complexity and Size:** Look for types with high SLOC and WMC values, as these are indicators of complexity. For example, `structures.results.TypeMetricResult` and `output.MetricResultJSON` have high SLOC and WMC values, meaning they could be made more maintainable by breaking them down into smaller, more focused units, thus reducing complexity and potentially duplicating code.

2. **Improve Cohesion:** Types with high LCOM3 values, like `fixtures.output.CSVDataFixture`, `structures.results.TypeMetricResultTest`, and `structures.results.NamespaceMetricResultTest`, indicate low cohesion, meaning the methods and attributes of these types are not working well together. Refactoring towards more cohesive designs will likely involve segregating functionality into well-defined classes that have clear responsibilities, which can improve understandability and reusability.

3. **Reduce Coupling:** High values of DEP (Dependencies), FAN-IN, and especially FAN-OUT suggest a high level of coupling. For instance, `javaProject.com.controller.Type` has a high FAN-OUT, making it heavily dependent on many other types. Lowering these metrics through the use of interfaces, dependency injection, or event patterns can help decrease coupling, making the system more modular and easier to modify or extend.

4. **Consolidate Data Structures:** Observing the NOA metric can reveal classes that might be acting more as data structures rather than providing

behavior, such as `output.MetricResultFile` with high NOA. Evaluating whether these attributes can be enclosed within more functional objects or whether such classes can be simplified or merged with others might result in a more concise and maintainable codebase.

By targeting these areas based on the provided metrics, the developer can systematically address issues of complexity, cohesion, and coupling in the codebase, leading towards an overall improvement in code quality and maintainability. These refactorings should be done iteratively and tested thoroughly to ensure system integrity is maintained throughout the process.

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m40 seconds **[1mGPT Context Tokens**: [0m15174 **[1mGPT Generated Tokens**: [0m": 551

[1mWaited time: [0m80 seconds

GPT Insight(Test 9):
Given the provided metrics data, here are key areas the developer should focus on for refactoring:

1. **Cohesion and Coupling**: High LCOM (Lack of Cohesion of Methods) values (close to 1) suggest poor cohesion within types, meaning the class methods are not well related in their purpose. For instance, types with LCOM3 values of 0.95 or higher should be carefully reviewed to see if they can be decomposed into more cohesive classes. Coupling metrics, including both FAN-IN and FAN-OUT, offer insight into how interdependent the classes are. Types with high FAN-OUT values, such as 22, suggest a class is highly dependent on many other classes and might benefit from refactoring to reduce dependencies.

2. **Size Metrics**: SLOC (Source Lines of Code) provides a quick way of identifying large classes that might be doing too much. For instance, types with the highest SLOC such as `374` might be good candidates for decomposition. Breaking down large classes into smaller, more manageable pieces can make the codebase more understandable and easier to maintain.

Given these two areas of focus, the developer should aim to:
- Improve cohesion within classes by ensuring that methods within a class are closely related in their functionality. If a class has methods that seem unrelated, consider splitting these into separate classes.
- Reduce coupling between classes by minimizing the number of dependencies a class has on other classes. This might involve using design patterns such as Facade to simplify interactions or adopting Dependency Injection to manage class dependencies better.
- Break down large classes into smaller ones. If a class has an unusually high SLOC, it's likely doing too much and should be decomposed into smaller, more focused classes that each handle a single responsibility.

By tackling these areas, the developer can work towards a more modular, maintainable, and understandable codebase.

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m31 seconds **[1mGPT Context Tokens**: [0m15174 **[1mGPT Generated Tokens**: [0m": 384

[1mWaited time: [0m80 seconds

GPT Insight(Test 10):
Given the extensive metric data provided, focusing on key areas can significantly improve code quality during the refactoring process. Here are the primary focus areas for refactoring:

1. **Reduce High Cyclomatic Complexity (WMC):** Look for types with high Weighted Methods per Class (WMC) values as these are indicators of complex classes. High complexity may make maintenance and understanding difficult. `structures.results.TypeMetricResult` and `main.Bootstrapper` are examples with high WMC values, suggesting that the methods in these classes are complex and potentially doing too much. Consider breaking down complex methods into smaller, more manageable ones.

2. **Address High Lack of Cohesion of Methods (LCOM3):** High LCOM3 values (approaching 1) indicate that the class’s methods are not related to each other in terms of accessing the same class attributes. In general, a high LCOM3 suggests that the class could be split into multiple, more cohesive classes. Types like `output.MetricResultJSON`, `structures.results.TypeMetricResultTest`, and `fixtures.output.CSVDataFixture` demonstrate high LCOM3 values, suggesting potential refactoring into more cohesive units.

---

3. **Minimize Fan-out Complexity:** Types with high Fan-Out, such as `output.MetricResultJSON` and `structures.results.TypeMetricResult`, are highly dependent on other classes or modules. Reducing these dependencies can make the system easier to understand, test, and maintain. Consider applying the Dependency Inversion Principle or breaking down classes to reduce their responsibilities.

4. **Improve Method Responsiveness by Reducing High NOM and NPM Values:** High Numbers of Methods (NOM) and Numbers of Public Methods (NPM) within a class might indicate classes with too many responsibilities. Types like `structures.results.TypeMetricResult` and `output.MetricResultFile` demonstrate high values, suggesting a violation of the Single Responsibility Principle. Try to segregate responsibilities into separate, focused classes.
Refocusing on these areas during the refactoring process should contribute to a more maintainable, understandable, and cohesive codebase. Always remember to unit test your changes to ensure that refactoring does not affect the existing functionality.

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m32 seconds **[1mGPT Context Tokens**: [0m15174
[1mGPT Generated Tokens: [0m": 447

Second experiment with type metrics analysis

The same experiment from the previous one was executed, to further analyze the response.

Prompt used (first 700 characters of 26346):
Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat should be the areas of focus to the developer to refactor this code based on the Metrics? Answer breaked into 2 linesMetrics data:<D>Types metrics:<D><D>Type: output.MetricResultJSON</D><D>SLOC: 374</D><D>NOM: 42</D><D>NPM: 38</D><D>WMC: 63</D><D>DEP: 25</D><D>I-DEP: 16</D><D>FAN-IN: 4</D><D>FAN-OUT: 22</D><D>NOA: 5</D><D>LCOM3: 0.95</D></D><D><D>Type: output.MetricResultGPT</D><D>SLOC: 366</D><D>NOM: 24</D><D>NPM: 23</D><D>WMC: 40</D><D>DEP: 17</D><D>I-DEP: 15</D><D>FAN-IN: 1</D><D>FAN-OUT: 20</D><D>NOA: 8</D><D>LCOM3: 0.85</D></D><D><D>Type: str ...

GPT Insight(Test 1):
Given the comprehensive metrics data provided, the refactoring efforts should target the following areas to enhance code quality, maintainability, and performance:

1. **High Lack of Cohesion (LCOM3 values close to 1)**: Focus on types such as `fixtures.output.CSVDataFixture`, `TypeMetricFixture`, `NamespaceVisitor`, and several types with LCOM3 = 1.00. This suggests that the methods within these types are not working together, which may indicate that the type is doing too much or its responsibilities are not well aligned. Consider splitting these types into more cohesive units or reevaluating their responsibilities.

2. **High Number of Methods (NOM) and Weighted Method Count (WMC)**: Inspect types such as `structures.results.TypeMetricResult` and `output.MetricResultFile` which have high NOM and WMC values. This could indicate complex classes that are trying to do too much. Applying the Single Responsibility Principle (SRP) can help in breaking down these classes into smaller, more focused classes.

Refactoring these areas will likely have a significant positive impact on the maintainability and understandability of the codebase.

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m12 seconds **[1mGPT Context Tokens**: [0m15174
[1mGPT Generated Tokens: [0m": 238

[1mWaited time: [0m80 seconds

GPT Insight(Test 2):
Based on the provided metrics data from various types, here are the key takeaways and areas of focus for refactoring:

1. **Reduce High Cyclomatic Complexity (WMC):** Look for types with high Weighted Methods per Class (WMC) values, indicating complex code that could benefit from simplification. Examples include `Type: structures.results.TypeMetricResult` with `WMC: 99` and `Type: javaProject.com.controller.Type` with `WMC: 58`. Reducing complexity makes the code more maintainable and easier to understand.

2. **Address High Lack of Cohesion of Methods (LCOM3):** High LCOM3 values suggest that classes could be doing too much or the class’s methods and fields are not related closely enough, which can lead to poor class cohesion. For example, `Type: structures.results.TypeMetricResultTest` and

`Type: output.MetricResultFake` both have `LCOM3: 0.95`. Consider refactoring these classes by segregating unrelated functionalities into separate classes or by enhancing the existing methods and fields to work more harmoniously.

---

3. **Optimize Fan-Out and Dependency Metrics:** High Fan-Out values, such as those seen in `Type: output.MetricResultJSON` (`FAN-OUT: 22`) and `Type: output.MetricResultCSV` (`FAN-OUT: 22`), indicate that a type depends on many other types. This could signify overly tight coupling. Look into possibilities to introduce interfaces, or perhaps dependency inversion principles, to reduce direct dependencies.

4. **Improve Method and Class Design:** Examine types with a high Number of Methods (NOM) but low usage of class fields (NOA), e.g., `Type: output.MetricResultFile` (`NOM: 41, NOA: 21`) and `Type: structures.metrics.TypeMetric` (`NOM: 36, NOA: 15`). It may indicate methods could be refactored into more cohesive, smaller classes, or there are opportunities to make better use of class-level fields, reducing parameter passing and improving encapsulation.

Refactoring with these focus areas not only aims at enhancing code readability and maintainability but also at improving the overall architecture's scalability and robustness.

=====================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m34 seconds **[1mGPT Context Tokens**: [0m15174
[1mGPT Generated Tokens: [0m": 468

[1mWaited time: [0m80 seconds

GPT Insight(Test 3):
Given the complexity and volume of metrics provided for various types, the first line of advice would revolve around understanding and identifying key areas where refactoring could yield significant improvements in maintainability, performance, and code quality. With this extensive data, certain metrics immediately stand out and should guide the initial focus of refactoring efforts:

1. **Cohesion (LCOM3):** Types with high LCOM3 values close to 1 (e.g., `fixtures.output.data.CSVDataFixture` with LCOM3 of 1.00, and several others) indicate poor cohesion, meaning that the methods within these types are not well related. Focusing on increasing cohesion by ensuring that classes have a single responsibility can significantly improve the codebase. Refactoring suggestions include splitting large classes into smaller, more focused ones or moving methods to classes where they are more relevant.

2. **Cyclomatic Complexity (WMC):** Types with high Weighted Methods per Class (WMC) indicate complex logic that can be hard to understand and test (e.g., `structures.results.TypeMetricResult` with a WMC of 99). Refactoring efforts should aim to simplify complex methods, possibly by breaking them down into smaller methods that are easier to understand and test. Techniques such as applying the Extract Method refactoring or simplifying conditional logic can be particularly useful here.

For the second line of advice, attention should be shifted towards dependencies and method metrics:

1. **Dependencies (DEP, I-DEP, FAN-IN, FAN-OUT):** High dependency metrics, especially FAN-OUT values, suggest that the type is overly dependent on many other types or components (e.g., `output.MetricResultJSON` with FAN-OUT of 22). This can make the system brittle and difficult to change. To minimize dependency issues, consider applying the Dependency Inversion Principle by refactoring towards depending on abstractions rather than concrete implementations. Additionally, look into partitioning the system into more decoupled, independently deployable units or modules.

2. **Method Metrics (NOM, NPM):** Types with a high Number of Methods (NOM) or Number of Public Methods (NPM) are indicative of potential \

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m49 seconds **[1mGPT Context Tokens**: [0m15174
[1mGPT Generated Tokens: [0m": 564

[1mWaited time: [0m80 seconds

GPT Insight(Test 4):
Given the comprehensive metrics data provided, the focus of refactoring should be on the areas of high Lack of Cohesion of Methods (LCOM3), high Source Lines of Code (SLOC), high Weighted Method Count (WMC), and excessive dependencies (DEP and I-DEP). To start, areas with a high LCOM3 value close to 1 indicate poor cohesion within a class, suggesting that the class is doing too much or the class's methods are not well related. High SLOC indicates potential complexity and difficulty in maintenance, while high WMC points towards complex classes that may be trying to perform too many actions, increasing the risk of bugs.

Firstly, attention should be directed towards reducing the complexity and enhancing the maintainability of the classes with high SLOC counted above. Refactoring strategies might include the extraction of methods or classes (if a class is performing too many functions), modularizing the code better, and identifying common patterns or behaviors that can be abstracted or unified.

Secondly, reducing the Lack of Cohesion of Methods (LCOM3) by ensuring that methods within a class are more tightly related to each other should be a priority. This often involves rethinking the responsibilities assigned to each class and possibly redistributing methods among classes or introducing new classes to group related functionalities together more logically.

In refactoring efforts targeting these areas, unit tests play a crucial role in ensuring that behavior is preserved. Automated tests should be in place before starting the refactoring process to detect any inadvertent changes or regressions in functionality. It's also prudent to tackle these refactoring efforts incrementally, applying small, manageable changes and validating through testing after each modification.

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m30 seconds **[1mGPT Context Tokens**: [0m15174
[1mGPT Generated Tokens: [0m": 339

[1mWaited time: [0m80 seconds

GPT Insight(Test 5):
Given the comprehensive list of metric data for various types, here are key recommendations for areas of focus during refactoring:

1. **High Lack of Cohesion (LCOM3):** Several types exhibit a high Lack of Cohesion of Methods (LCOM3) score, close to 1.0, such as `fixtures.output.CSVDataFixture`, `structures.results.TypeMetricResultTest`, and `output.MetricResultFake`. These types should be refactored to improve cohesion. Aim to modularize functionality such that each class focuses on a single purpose or closely related purposes.

2. **High Fan-Out:** Certain types have a high Fan-Out, such as `output.MetricResultJSON` and `output.MetricResultCSV` with Fan-Outs of 22. High Fan-Out indicates a type's high dependency on other types, suggesting a potential need for decoupling and encapsulation improvements. Extract interfaces, use design patterns like Dependency Injection, or apply the Facade Pattern to reduce direct dependencies.

Continued areas of focus:

3. **Complexity Metrics (WMC) and Size Metrics (SLOC, NOM):** Types like `structures.results.TypeMetricResult` with WMC of 99, SLOC of 328, and NOM of 45 are likely to be complex and large, suggesting potential refactoring to break down large classes into smaller, more manageable ones. Aim for single responsibility and lower complexity to improve maintainability.

4. **High Internal Dependency (DEP & I-DEP):** Consider types with high external and internal dependencies, such as `parser.java.visitors.MethodVisitor` with DEP of 23 and `selection.options.Options` with I-DEP of 19. It's vital to assess these dependencies for possible reduction through architectural redesign, possibly by introducing more abstract layers or utilizing patterns that favor loose coupling.

By addressing these areas, the software architecture can evolve towards a more maintainable, scalable, and loosely coupled system. Refactoring efforts should prioritize readability, simplicity, and encapsulation, applying solid principles and design patterns where appropriate.

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m31 seconds **[1mGPT Context Tokens**: [0m15174
[1mGPT Generated Tokens: [0m": 424

[1mWaited time: [0m80 seconds

GPT Insight(Test 6):
Given the comprehensive set of metrics provided for a variety of types, the key focus areas for refactoring should prioritize addressing high complexity, reducing tight coupling, enhancing cohesion, and reducing code duplication. Specifically:

1. **High Complexity & Tight Coupling**: Target types with high WMC (Weighted Method Count) and high FAN-OUT values as they indicate complexity and tight coupling. For example, the `structures.results.TypeMetricResult` has a WMC of 99 and a FAN-OUT of 9, suggesting it could benefit from simplification and breaking down into smaller, more focused components.

2. **Low Cohesion**: Look for types with high LCOM3 (Lack of Cohesion of Methods) values, as this indicates that the methods within a class do not share much in common. Types like `output.MetricResultFake` with LCOM3 of 1.00 and `fixtures.TypeMetricFixture` with LCOM3 of 1.00 are prime candidates for refactoring to improve cohesion.

Continuing with guidance based on the provided metrics:

1. **Reduce Dependencies**: Types with high DEP (Direct dependencies) and I-DEP (Indirect dependencies) like `selection.options.Options` with DEP of 21 and I-DEP of 19, indicate a high reliance on other components, making the system more fragile and harder to maintain. Work on reducing these dependencies, possibly through interface abstraction or by applying the Dependency Inversion Principle.

2. **Improve Encapsulation**: NOA (Number of Attributes) figures suggest how many state variables a type is managing. For better encapsulation and to facilitate easier unit testing, consider refactoring types with a high NOA, like `output.MetricResultFile` with NOA of 21, by perhaps breaking them down into smaller classes each responsible for a more specific subset of functionality.

This strategic approach to refactoring should lead to improved maintainability, readability, and scalability of your codebase.

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m29 seconds **[1mGPT Context Tokens**: [0m15174
[1mGPT Generated Tokens: [0m": 404

[1mWaited time: [0m80 seconds

GPT Insight(Test 7):
Based on the provided metrics data, here are the key areas you should focus on for refactoring:

1. **Reduce High Complexity and Interdependence**: Look into types with high Weighted Method Count (WMC) and high dependence (DEP and I-DEP). For example, refactoring the types with the highest WMC like `structures.results.TypeMetricResult` (WMC: 99) and `structures.metrics.TypeMetric` (WMC: 37) could improve maintainability. Aim to simplify complex methods and reduce coupling where possible.

2. **Address High Lack of Cohesion (LCOM3)**: Types with high Lack of Cohesion of Methods version 3 (LCOM3) values, near or equal to 1.00, indicate that the class methods and fields are not well related. Examples are `fixtures.output.data.CSVDataFixture` and `fixtures.TypeMetricFixture`, both with LCOM3 of 1.00, and several others. Consider breaking these types into smaller, more cohesive ones.

Continued in the next part...

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m20 seconds **[1mGPT Context Tokens**: [0m15174
[1mGPT Generated Tokens: [0m": 218

[1mWaited time: [0m80 seconds

GPT Insight(Test 8):
Given the substantial amount of metrics data for various types, the areas for refactoring can be effectively identified by focusing on certain key metrics indicative of code quality and maintainability. These metrics include SLOC (Source Lines of Code), NOM (Number of Methods), WMC (Weighted Methods per Class), DEP (Dependencies), FAN-IN/FAN-OUT, NOA (Number of Attributes), and notably LCOM3 (Lack of Cohesion of Methods version 3). Identifying these metrics helps in determining the complexity, coupling, and cohesion of the code, which are significant factors in software maintainability.

To start, the primary areas of focus for the developer to refactor, based on the metrics are:

1. **Reduce Complexity and Size:** Look for types with high SLOC and WMC values, as these are indicators of complexity. For example, `structures.results.TypeMetricResult` and `output.MetricResultJSON` have high SLOC and WMC values, meaning they could be made more maintainable by breaking them down into smaller, more focused units, thus reducing complexity and potentially duplicating code.

2. **Improve Cohesion:** Types with high LCOM3 values, like `fixtures.output.CSVDataFixture`, `structures.results.TypeMetricResultTest`, and `structures.results.NamespaceMetricResultTest`, indicate low cohesion, meaning the methods and attributes of these types are not working well together. Refactoring towards more cohesive designs will likely involve segregating functionality into well-defined classes that have clear responsibilities, which can improve understandability and reusability.

3. **Reduce Coupling:** High values of DEP (Dependencies), FAN-IN, and especially FAN-OUT suggest a high level of coupling. For instance, `javaProject.com.controller.Type` has a high FAN-OUT, making it heavily dependent on many other types. Lowering these metrics through the use of interfaces, dependency injection, or event patterns can help decrease coupling, making the system more modular and easier to modify or extend.

4. **Consolidate Data Structures:** Observing the NOA metric can reveal classes that might be acting more as data structures rather than providing behavior, such as `output.MetricResultFile` with high NOA. Evaluating whether these attributes can be enclosed within more functional objects or whether such classes can be simplified or merged with others might result in a more concise and maintainable codebase.

By targeting these areas based on the provided metrics, the developer can systematically address issues of complexity, cohesion, and coupling in the codebase, leading towards an overall improvement in code quality and maintainability. These refactorings should be done iteratively and tested thoroughly to ensure system integrity is maintained throughout the process.

=====================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m40 seconds **[1mGPT Context Tokens**: [0m15174 **[1mGPT Generated Tokens**: [0m": 551

[1mWaited time: [0m80 seconds

GPT Insight(Test 9):
Given the provided metrics data, here are key areas the developer should focus on for refactoring:

1. **Cohesion and Coupling**: High LCOM (Lack of Cohesion of Methods) values (close to 1) suggest poor cohesion within types, meaning the class methods are not well related in their purpose. For instance, types with LCOM3 values of 0.95 or higher should be carefully reviewed to see if they can be decomposed into more cohesive classes. Coupling metrics, including both FAN-IN and FAN-OUT, offer insight into how interdependent the classes are. Types with high FAN-OUT values, such as 22, suggest a class is highly dependent on many other classes and might benefit from refactoring to reduce dependencies.

2. **Size Metrics**: SLOC (Source Lines of Code) provides a quick way of identifying large classes that might be doing too much. For instance, types with the highest SLOC such as `374` might be good candidates for decomposition. Breaking down large classes into smaller, more manageable pieces can make the codebase more understandable and easier to maintain.

Given these two areas of focus, the developer should aim to:
- Improve cohesion within classes by ensuring that methods within a class are closely related in their functionality. If a class has methods that seem unrelated, consider splitting these into separate classes.
- Reduce coupling between classes by minimizing the number of dependencies a class has on other classes. This might involve using design patterns such as Facade to simplify interactions or adopting Dependency Injection to manage class dependencies better.
- Break down large classes into smaller ones. If a class has an unusually high SLOC, it's likely doing too much and should be decomposed into smaller, more focused classes that each handle a single responsibility.

By tackling these areas, the developer can work towards a more modular, maintainable, and understandable codebase.

=====================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m31 seconds **[1mGPT Context Tokens**: [0m15174 **[1mGPT Generated Tokens**: [0m": 384

[1mWaited time: [0m80 seconds

GPT Insight(Test 10):
Given the extensive metric data provided, focusing on key areas can significantly improve code quality during the refactoring process. Here are the primary focus areas for refactoring:

```
1. **Reduce High Cyclomatic Complexity (WMC):** Look for types with high
Weighted Methods per Class (WMC) values as these are indicators of complex
classes. High complexity may make maintenance and understanding difficult.
`structures.results.TypeMetricResult` and `main.Bootstrapper` are examples
with high WMC values, suggesting that the methods in these classes are
complex and potentially doing too much. Consider breaking down complex
methods into smaller, more manageable ones.

2. **Address High Lack of Cohesion of Methods (LCOM3):** High LCOM3 values
(approaching 1) indicate that the class’s methods are not related to each
other in terms of accessing the same class attributes. In general, a high
LCOM3 suggests that the class could be split into multiple, more cohesive
classes. Types like `output.MetricResultJSON`,
`structures.results.TypeMetricResultTest`, and
`fixtures.output.CSVDataFixture` demonstrate high LCOM3 values, suggesting
potential refactoring into more cohesive units.

---

3. **Minimize Fan-out Complexity:** Types with high Fan-Out, such as
`output.MetricResultJSON` and `structures.results.TypeMetricResult`, are
highly dependent on other classes or modules. Reducing these dependencies
can make the system easier to understand, test, and maintain. Consider
applying the Dependency Inversion Principle or breaking down classes to
reduce their responsibilities.

4. **Improve Method Responsiveness by Reducing High NOM and NPM Values:**
High Numbers of Methods (NOM) and Numbers of Public Methods (NPM) within a
class might indicate classes with too many responsibilities. Types like
`structures.results.TypeMetricResult` and `output.MetricResultFile`
demonstrate high values, suggesting a violation of the Single
Responsibility Principle. Try to segregate responsibilities into separate,
focused classes.
Refocusing on these areas during the refactoring process should contribute
to a more maintainable, understandable, and cohesive codebase. Always
remember to unit test your changes to ensure that refactoring does not
affect the existing functionality.


======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m32 seconds [1mGPT Context Tokens: [0m15174
[1mGPT Generated Tokens: [0m": 447
```

Experiment with type metrics request Top 3 types needing refactoring

This experiment was to provide a more valuable insight, so that integration provides which classes on our project should be refactored.

Figure A.3.51 – Result from insight request based on type metrics

```
GPT Insight(Test 1):
	Top 3 types that need refactoring:

	1. structures.results.TypeMetricResult
	2. output.MetricResultJSON
	3. output.MetricResultCSV

	For **structures.results.TypeMetricResult**, the primary concern is the extremely high Weighted Methods per Class (WMC) of 99, indicating a high complexity and potentially many responsibilities within a single type. This, coupled with a high Lack of Cohesion of Methods (LCOM3) of 0.94, suggests that the type is doing too much, and its methods are not closely related in functionality. **Refactoring Techniques**: The type should undergo **Extract Class or Extract Subclass refactoring** to divide its responsibilities into smaller, cohesive classes. The high complexity (WMC) could be addressed through **Extract Method refactoring** to break down complex methods into simpler ones, making the codebase easier to understand, maintain, and extend. For more details, the \"Refactoring: Improving the Design of Existing Code\" by Martin Fowler provides comprehensive strategies for such refactorings.

	**output.MetricResultJSON** shows signs of significant complexity (WMC: 64) and a high number of methods (NOM: 43), indicating that it might be overloaded with responsibilities. Its high LCOM3 value of 0.95 suggests poor cohesion among methods. **Refactoring Techniques**: **Extract Class or Interface** would be beneficial here, splitting the type into more manageable parts that are easier to understand and test. Using **Extract Method** can help in breaking down large methods that do too much. These changes can improve maintainability and readability. Fowler's book provides practical guidance on performing these refactorings effectively.

	**output.MetricResultCSV** has a very high LCOM3 of 0.96, approaching the least cohesive score possible, coupled with a high number of dependencies (DEP: 17) and fan-out complexity indicating that it's too intertwined with other parts of the system. **Refactoring Techniques**: **Extract Class** refactoring can help by splitting the class into smaller, more cohesive ones. **Modularization** can also be employed to better encapsulate its functionality, making the system more flexible and reducing dependencies. The **Interface Segregation Principle** from SOLID principles can guide reducing the fan-out complexity by ensuring that classes do not depend on interfaces they do not use. For detailed approaches, Robert C. Martin's \"Clean Code: A Handbook of Agile Software Craftsmanship\" offers insights into writing cleaner, more maintainable code through effective refactoring practices.
```

Source: Elaborated by the author

Figure A.3.52– Result from insight request based on type metrics (2nd Request)

```
GPT Insight(Test 2):
	The top 3 types that need refactoring based on the provided metrics data are:

	1. output.MetricResultJSON
	2. structures.results.TypeMetricResult
	3. output.MetricResultCSV

	For output.MetricResultJSON, with a high Lack of Cohesion of Methods (LCOM3) value of 0.95, and a high Weighted Method Count (WMC) of 64, it indicates that the class is doing too many things, leading to high complexity and low cohesion. Refactoring techniques such as Extract Class or Extract Method can be used to split the class into smaller, more cohesive pieces or to pull out methods into new classes. Martin Fowler's \"Refactoring: Improving the Design of Existing Code\" offers detailed insights on when and how to apply these techniques effectively.

	structures.results.TypeMetricResult shows a similar pattern with an even higher WMC of 99 and an LCOM3 of 0.94. This suggests that the type suffers from bloated class syndrome and possibly manages too many responsibilities. Applying the Single Responsibility Principle (SRP) from the SOLID principles can guide the refactoring process. Techniques like Extract Class, along with moving methods to the classes where they are most closely related (e.g., using the Move Method technique), can significantly reduce the class's complexity and increase its maintainability.

	output.MetricResultCSV exhibits a very high LCOM3 of 0.96, alongside a significant WMC of 46, indicating poor cohesion and high complexity. This type would benefit greatly from refactoring aimed at reducing its responsibilities and improving cohesion. Decomposing the class into smaller, purpose-focused classes through the Extract Class refactoring technique would help. Moreover, careful application of the Interface Segregation Principle (part of the SOLID principles) could ensure that clients of this type only depend on the interfaces they use, further enhancing modularity and maintainability.

	Reference documentation includes Martin Fowler's \"Refactoring: Improving the Design of Existing Code,\" which provides comprehensive guidance on implementing these refactoring techniques. Additionally, Robert C. Martin's \"Clean Code: A Handbook of Agile Software Craftsmanship\" and his writings on SOLID principles are invaluable resources for understanding the theoretical underpinnings of these recommendations and their practical applications.


======================ChatGPT Insight end(built prompt)

	GPT processing time: 43 seconds GPT Context Tokens: 14677	GPT Generated Tokens: ": 441
```

Source: Elaborated by the author

Figure A.3.53 – Result from insight request based on type metrics (3rd Request)

```
GPT Insight(Test 3):
	The top 3 types that need refactoring based on the provided metrics data are:

	1. structures.results.TypeMetricResult
	2. output.MetricResultJSON
	3. output.MetricResultCSV

	For structures.results.TypeMetricResult, the key metrics indicating the need for refactoring include a high Weighted Method per Class (WMC) at 99, a high Lack of Cohesion of Methods (LCOM3) at 0.94, and a relatively high Number of Methods (NOM) at 45. This suggests the type is doing too much and its methods are not closely related in terms of the data they use. The refactoring techniques recommended here would include Extract Class to move methods and fields to new or existing classes where they fit more logically, and Extract Method to break down large methods into smaller, more cohesive methods. Additionally, applying the Interface Segregation Principle may help in reducing the class's responsibilities.

	For output.MetricResultJSON, similar to structures.results.TypeMetricResult, the indicators of concern include a high WMC at 64, high NOM at 43, and a high LCOM3 at 0.95. This suggests that output.MetricResultJSON suffers from high complexity and low cohesion, making it hard to maintain and extend. Refactoring techniques such as Extract Class, Extract Method, and possibly Introduce Parameter Object for methods with many parameters can be used here. These techniques help in decomposing the class into more manageable and cohesive components.

	output.MetricResultCSV also shows signs of needing refactoring, with a high LCOM3 score of 0.96 indicating low cohesion, and high WMC and NOM suggesting that the class is overly complex. Similar to output.MetricResultJSON and structures.results.TypeMetricResult, applying methods like Extract Class and Extract Method would help in making this class more cohesive and its behavior more localized.

	For detailed guidance on applying these refactoring techniques, one can refer to the book \"Refactoring: Improving the Design of Existing Code\" by Martin Fowler, which provides in-depth explanations and examples of these and many other refactoring strategies.


======================ChatGPT Insight end(built prompt)

	GPT processing time: 36 seconds GPT Context Tokens: 14677	GPT Generated Tokens: ": 397
```

Source: Elaborated by the author

Figure A.3.54 – Result from insight request based on type metrics (4th Request)

```
GPT Insight(Test 4):
        Top 3 Types that need refactoring: output.MetricResultCSV, output.MetricResultJSON, structures.results.TypeMetricResult.

        For **structures.results.TypeMetricResult**, with high WMC (Weighted Methods per Class) at 99, indicating complex class structure, and a high LCOM3 (Lack of Cohesion of Methods) at 0.94, suggesting methods within the class do not share data often. Refactoring techniques such as **Extract Method** for overly complex methods and **Extract Class** for functionalities not closely related to the class's primary responsibility should be considered. The goal is to lower the WMC by breaking down complex methods or class functionalities into smaller, more manageable pieces, thereby also decreasing LCOM3 by increasing method cohesion. Martin Fowler's \"Refactoring: Improving the Design of Existing Code\" provides excellent guidance on applying these techniques.

        For **output.MetricResultJSON**, notable metrics are a high NOM (Number of Methods) and NPM (Number of Public Methods), both suggesting the class might be doing too much or has overly complex methods. Alongside, a very high LCOM3 value of 0.95 indicates low cohesion. Utilizing the **Extract Class** refactoring method can help by separating concerns within the class, thus adhering to the Single Responsibility Principle. Another useful technique could be **Extract Method** to break down complex methods into simpler ones, as discussed in Robert C. Martin's \"Clean Code: A Handbook of Agile Software Craftsmanship.\"

        **output.MetricResultCSV** has one of the highest LCOM3 values at 0.96, which is a strong indicator of low cohesion within the class. Coupled with a relatively high value of FAN-OUT at 22, indicating many outbound dependencies, this class would benefit from **Extract Class** refactoring to separate concerns and reduce dependencies. Additionally, implementing **Module Extract** patterns could be advantageous to isolate responsibility and enhance modularity, thereby reducing the class's complexity and enhancing maintainability. References for these techniques can be found in Steve McConnell's \"Code Complete.\"


======================ChatGPT Insight end(built prompt)

        GPT processing time: 41 seconds GPT Context Tokens: 14677       GPT Generated Tokens: ": 406
```

Source: Elaborated by the author

Figure A.3.55 – Prompt used for the results above

```
Prompt used (first 500 characters of 22630):
        Act as a Software Architect, providing advice to a new developer on what to do on a refactoring, based on the Data provided after the requestsWhat is the top 3 types that need refactoring? Answer only the type name, then one paragraph explain what should be refactored and which technique to be used, providing reference documentation.Metrics data:<D>Types metrics:<D><D>Type: type1</D><D>SLOC: 377</D><D>NOM: 43</D><D>NPM: 39</D><D>WMC: 64</D><D>DEP: 25</D><D>I-DEP: 16</D><D>FAN-IN: 4</D><D>FAN-OUT ...
```

Source: Elaborated by the author

Text version of the results (with all 10 executions):

```
======================ChatGPT Insight:


Prompt used (first 700 characters of 26418):
Act as a Software Architect, providing advice to a new developer on what to
do on a refactoring, based on the Data provided after the requestsWhat is
the top 3 types that need refactoring? Answer only the type name, then one
paragraph explain what should be refactored and which technique to be used,
providing reference documentation.Metrics data:<D>Types metrics:<D><D>Type:
output.MetricResultJSON</D><D>SLOC: 374</D><D>NOM: 42</D><D>NPM:
38</D><D>WMC: 63</D><D>DEP: 25</D><D>I-DEP: 16</D><D>FAN-IN: 4</D><D>FAN-
OUT: 22</D><D>NOA: 5</D><D>LCOM3: 0.95</D></D><D><D>Type:
output.MetricResultGPT</D><D>SLOC: 366</D><D>NOM: 24</D><D>NPM:
23</D><D>WMC: 40</D><D>DEP: 17</D><D>I-DEP: 15</D><D>FAN-IN: 1< ...

GPT Insight(Test 1):
The top 3 types that need refactoring based on the provided metrics data
are:

1. structures.results.TypeMetricResult
2. output.MetricResultJSON
3. output.MetricResultGPT

The `structures.results.TypeMetricResult` type has a high Weighted Method
Count (WMC) of 99, which implies complex methods possibly doing too much
work or having too many responsibilities. A high LCOM3 value of 0.94
suggests low cohesion, indicating that the methods are not well-related in
functionality, leading to difficulties in maintenance and understanding. To
refactor, extract methods or even classes from this type to reduce
complexity and increase cohesion. Techniques such as Extract Class or
Extract Method, as described in Fowler's \


======================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m30 seconds **[1mGPT Context Tokens**: [0m15187
[1mGPT Generated Tokens: [0m": 422

[1mWaited time: [0m80 seconds

GPT Insight(Test 2):
The top 3 types that need refactoring based on the provided metrics data are:

1. structures.results.TypeMetricResult
2. output.MetricResultJSON
3. output.MetricResultCSV

The `structures.results.TypeMetricResult` type shows a high Weighted Method Count (WMC) at 99, indicating a high complexity and potentially multiple responsibilities violating the Single Responsibility Principle. This type also has a high Lack of Cohesion in Methods 3 (LCOM3) at 0.94, suggesting that the methods of the class are not cohesive and could probably be separated into multiple, more focused classes. Refactoring strategies could include applying the Extract Class refactoring technique to create more cohesive classes, and breaking down complex methods into smaller ones, potentially using the Extract Method refactoring technique. For reference and further reading on these refactoring techniques, see Martin Fowler's book \

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m32 seconds **[1mGPT Context Tokens**: [0m15187
[1mGPT Generated Tokens: [0m": 453

[1mWaited time: [0m80 seconds

GPT Insight(Test 3):
Top 3 Types that need refactoring:
1. ```structures.results.TypeMetricResult```
2. ```output.MetricResultJSON```
3. ```output.MetricResultGPT```

For the `structures.results.TypeMetricResult` type, the high Weighted Method Count (WMC) of 99 and the high Lack of Cohesion in Methods 3 (LCOM3) of 0.94 suggest that this class has too many responsibilities and its methods do not share enough common data. Extract Class and Extract Method refactorings [Fowler] should be considered to reduce the class size, improve cohesion, and separate concerns. Splitting this class into smaller, more focused classes can help manage complexity and improve maintainability. Additionally, reviewing the use of instance variables across methods to identify opportunities for Extract Method refactoring can lead to more cohesive and modular code.

For `output.MetricResultJSON`, a Source Lines of Code (SLOC) of 374 and a high method count (NOM of 42) indicate that this class might be doing too much and could benefit from decomposition. Applying the Extract Class refactoring technique can help in separating concerns, simplifying future maintenance, and improving readability. For instance, if this class is handling both JSON formatting and data manipulation, these responsibilities could be divided into separate classes.

With `output.MetricResultGPT`, similar to `output.MetricResultJSON`, the SLOC of 366 and the NOM of 24 suggest that refactoring techniques such as Extract Class or Method should be applied to break down responsibilities into smaller, more manageable pieces. The aim should be to enhance the Single Responsibility Principle (SRP) and make the codebase more maintainable.

Reference documentation for these refactorings can be found in Martin Fowler's refactoring book and catalog, which provides comprehensive guidance on how to apply these techniques effectively [Fowler, Martin. \

=====================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m40 seconds **[1mGPT Context Tokens**: [0m15187
[1mGPT Generated Tokens: [0m": 406

[1mWaited time: [0m80 seconds

GPT Insight(Test 4):
The top 3 types that need refactoring are:

1. structures.results.TypeMetricResult
2. output.MetricResultJSON
3. output.MetricResultGPT

For **structures.results.TypeMetricResult**, the high Weighted Method Count (WMC) of 99 and Number of Methods (NOM) of 45 suggest a complex class with too much functionality crammed into a single class. This class should be refactored to adhere to the Single Responsibility Principle (SRP), possibly by splitting it into smaller, more cohesive classes. The Extract Class refactoring method can be applied here. Moreover, a high Lack of Cohesion in Methods 3 (LCOM3) value of 0.94 indicates that the methods within the class do not share much data, further suggesting the need for class splitting. The Extract Method refactoring can also be applied to reduce the complexity of methods, making them do one thing only as suggested by SRP. Reference: Martin Fowler's \

=====================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m42 seconds **[1mGPT Context Tokens**: [0m15187
[1mGPT Generated Tokens: [0m": 485

[1mWaited time: [0m80 seconds

GPT Insight(Test 5):
The top 3 types that need refactoring are:

1. structures.results.TypeMetricResult
2. output.MetricResultJSON
3. output.MetricResultGPT

These types show indications of high complexity and potential design issues, highlighted by metrics like Source Lines of Code (SLOC), Weighted Methods per Class (WMC), and Lack of Cohesion of Methods (LCOM3).

For the `structures.results.TypeMetricResult` type, with the highest WMC (99) suggesting complex class methods, and a high LCOM3 (0.94) indicating low cohesion, refactoring towards simplifying these methods and improving cohesion is advised. Techniques could include Extract Method to reduce complexity of individual methods, and Extract Class to improve cohesion by grouping related functionalities together.

The `output.MetricResultJSON` type has a relatively high SLOC (374) and a high WMC (63), pointing to complex and potentially uncohesive methods. This type could benefit from refactoring techniques like Extract Method to break down complex methods into simpler ones, making it easier to understand and modify.

The `output.MetricResultGPT` type has a high LCOM3 value (0.85) alongside substantial SLOC (366) and WMC (40), indicating a need for better cohesion and possible method complexity issues. Implementing refactoring strategies such as Extract Class can help in dividing responsibilities into more cohesive units, while Extract Method can address complex methods by dividing them into more manageable segments.

Reference documentation for these techniques can be found in Martin Fowler's \

=====================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m25 seconds **[1mGPT Context Tokens**: [0m15187
[1mGPT Generated Tokens: [0m": 347

[1mWaited time: [0m80 seconds

GPT Insight(Test 6):
Top 3 types that need refactoring:

1. `structures.results.TypeMetricResult`
2. `output.MetricResultJSON`
3. `output.MetricResultCSV`

The `structures.results.TypeMetricResult` type shows a high complexity with a Weighted Method Count (WMC) of 99 and a high Lack of Cohesion of Methods 3 (LCOM3) score of 0.94, alongside a significant number of methods (NOM) at 45. This indicates that the class likely has multiple responsibilities that could be broken down into smaller, more cohesive classes. Refactoring techniques such as Extract Class or Extract Method could be beneficial in reducing complexity and increasing cohesion.

For `output.MetricResultJSON`, it also suffers from high complexity with a WMC of 63, a high LCOM3 score of 0.95, and a rather large number of methods (NOM of 42). Similar to `structures.results.TypeMetricResult`, applying Extract Class and Extract Method could help. Additionally, considering its function, applying patterns like Data Transfer Objects (DTO) could encapsulate the data better.

Lastly, `output.MetricResultCSV` has a high LCOM3 of 0.95 and a considerable number of non-private methods (NPM of 34) with a WMC of 45. This suggests a lack of method cohesion and potential for method-level refactoring. Extract Method could be used here too for better modularization of the code.

```
For reference documentation on these refactoring techniques, \


=====================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m32 seconds **[1mGPT Context Tokens**: [0m15187
[1mGPT Generated Tokens: [0m": 371

[1mWaited time: [0m80 seconds

```
GPT Insight(Test 7):
The top 3 types that need refactoring based on the provided metrics are:

1. structures.results.TypeMetricResult
2. output.MetricResultJSON
3. output.MetricResultGPT

For the type `structures.results.TypeMetricResult`, it has a very high
Weighted Method Count (WMC) of 99, indicating that the methods in this type
are quite complex. To tackle this, methods should be broken down into
smaller, more manageable pieces. Applying the Extract Method refactoring
technique could be a good starting point, as described in Martin Fowler's \


=====================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m33 seconds **[1mGPT Context Tokens**: [0m15187
[1mGPT Generated Tokens: [0m": 449

[1mWaited time: [0m80 seconds

```
GPT Insight(Test 8):
Top 3 types that need refactoring:

1. structures.results.TypeMetricResult
2. output.MetricResultJSON
3. output.MetricResultConsole

The class `structures.results.TypeMetricResult` stands out primarily due to
its very high Weighted Methods per Class (WMC) value of 99, which typically
indicates a high level of complexity and possibly a large number of
responsibilities. Such complexity can often lead to difficulties in
maintenance and understanding the class. Refactoring techniques that could
be applied here include **Extract Class** to create more cohesive classes
with fewer responsibilities, and **Extract Method** to reduce complexity
within methods. Focusing on reducing its high complexity will likely
improve maintainability and readability. Reference for these techniques can
be found in Martin Fowler's \


=====================ChatGPT Insight end(built prompt)
```

[1mGPT processing time: [0m24 seconds **[1mGPT Context Tokens**: [0m15187
[1mGPT Generated Tokens: [0m": 367

[1mWaited time: [0m80 seconds

GPT Insight(Test 9):
The top 3 types that need refactoring are:

1. structures.results.TypeMetricResult
2. output.MetricResultJSON
3. output.MetricResultGPT

For **structures.results.TypeMetricResult**, the Weighted Method Count (WMC) stands out as significantly high at 99, indicating complex methods which might be doing too much or are too complicated. Refactoring this would likely involve breaking these methods into smaller, more manageable pieces. Applying the Extract Method refactoring technique, as described in Martin Fowler's "Refactoring: Improving the Design of Existing Code", could be quite effective here. This would not only help in reducing the complexity but might also improve the modularity and reusability of the code.

For **output.MetricResultJSON**, the class has a high Source Lines of Code (SLOC) and Number of Methods (NOM), which suggests that it might be handling too many responsibilities. A potential refactoring strategy could involve applying the Single Responsibility Principle (SRP). This principle, part of the SOLID principles detailed by Robert C. Martin, advises that a class should have only one reason to change. Segregating responsibilities into different classes could improve coherence and maintainability.

Finally, **output.MetricResultGPT** shows a combination of high SLOC and high Number of Public Methods (NPM). This might indicate that the class is offering too many services or functionalities, potentially complicating the use and maintenance of the class. Refactoring by applying the Interface Segregation Principle (ISP), another of the SOLID principles, by splitting the functionality into multiple, more specific interfaces, could make the class more manageable and adaptable to change. Reducing the class size and separating concerns would likely make the codebase more flexible and easier to understand.

Reference documentation for these techniques and principles can be found in Martin Fowler's "Refactoring: Improving the Design of Existing Code" for practical refactoring techniques and Robert C. Martin's "Clean Architecture" and "Agile Software Development, Principles, Patterns, and Practices" for SOLID principles application.

======================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m32 seconds **[1mGPT Context Tokens**: [0m15187
[1mGPT Generated Tokens: [0m": 421

[1mWaited time: [0m80 seconds

GPT Insight(Test 10):
Top 3 types that need refactoring:

1. structures.results.TypeMetricResult
2. output.MetricResultJSON
3. output.MetricResultGPT

For the `structures.results.TypeMetricResult`, given its high Weighted Method Count (WMC) of 99 and its high Lack of Cohesion in Methods (LCOM3) of 0.94, it suggests that the class is doing too much, with methods not

```
closely related in functionality. Refactoring strategies for this class
should include **Extract Class** to separate responsibilities into other
classes, improving cohesion, and potentially **Extract Method** to break
down large methods into smaller, cohesive ones. Additionally, evaluating
the necessity and usage of class-level variables (with a NOA of 8) could
identify opportunities for **Move Field** to relocate some fields to more
relevant classes. The **Martin's Principles of Package Design** can provide
further guidance on modularizing the system more effectively.

The `output.MetricResultJSON` and `output.MetricResultGPT` share similar
issues, with high SLOC (374 and 366 respectively), high NOM (42 and 24
respectively), and very high LCOM3 scores (0.95 and 0.85, indicating low
cohesion). For both types, applying **Extract Method** for lengthy methods
and **Extract Class** for unrelated functionalities within these types
could greatly improve maintainability. For example, handling JSON in
multiple formats or contexts might be spread across various methods that
could be better organized in separate classes. Considering the **Single
Responsibility Principle** from SOLID principles could be beneficial in
guiding these extractions, ensuring each class or method has one reason to
change.

Reference documentation includes:
- Fowler, M. (2018). Refactoring: Improving the Design of Existing Code.
Addison-Wesley Professional. This book covers a wide range of refactoring
techniques, including Extract Method and Extract Class.
- Martin, R. (2009). Clean Code: A Handbook of Agile Software
Craftsmanship. Prentice Hall. This provides an in-depth look at writing
clean, maintainable code which is vital during and after refactoring.
- Martin, R. C. (2003). Agile Software Development, Principles, Patterns,
and Practices. Prentice Hall. Offers insights into principles like Single
Responsibility to guide the refactoring process.


=====================ChatGPT Insight end(built prompt)

[1mGPT processing time: [0m38 seconds [1mGPT Context Tokens: [0m15187
[1mGPT Generated Tokens: [0m": 476
```

3.3 DR-TOOLS CODE HEALTH EXPERIMENT

3.3.2 PROMPT ENGINEERING FOR DR-TOOLS CODE HEALTH DATA

This section presents the full data in its raw format from the information presented in section 3.3.2.

Full data on prompt format:

“*<D>Code Smell Metric:<D><D>TYPE: structures.metrics.TypeMetric<\D><D>SLOC: 151 <\D><D>NOM: 36 <\D><D>NPM: 36 <\D><D>WMC: 38<\D><D>DEP: 9 <\D><D>I-DEP: 2 <\D><D>FAN-IN: 11 <\D><D>FAN-OUT: 5 <\D><D>NOA: 15 <\D><D>LCOM3: 0.80 <\D><D>DIT: 1.0*

<\D><D>CHILD: 0.0 <\D><D>NPA: 0.0<\D><D>List of smells detected:<D>Insufficient Modularization<\D><D>Multifaceted Abstraction<\D><\D><D>TOTAL OF SMELLS DETECTED: 2<\D><\D><D><D>TYPE: output.MetricResultGPT<\D><D>SLOC: 377 <\D><D>NOM: 24 <\D><D>NPM: 23 <\D><D>WMC: 41<\D><D>DEP: 17 <\D><D>I-DEP: 15 <\D><D>FAN-IN: 1 <\D><D>FAN-OUT: 20 <\D><D>NOA: 8 <\D><D>LCOM3: 0.85 <\D><D>DIT: 1.0 <\D><D>CHILD: 0.0 <\D><D>NPA: 2.0<\D><D>List of smells detected:<D>Insufficient Modularization<\D><D>Multifaceted Abstraction<\D><D>Deficient Encapsulation<\D><D>TOTAL OF SMELLS DETECTED: 3<\D><\D><D><D>TYPE: structures.results.TypeMetricResult<\D><D>SLOC: 328 <\D><D>NOM: 45 <\D><D>NPM: 31 <\D><D>WMC: 99<\D><D>DEP: 12 <\D><D>I-DEP: 3 <\D><D>FAN-IN: 16 <\D><D>FAN-OUT: 9 <\D><D>NOA: 8 <\D><D>LCOM3: 0.94 <\D><D>DIT: 1.0 <\D><D>CHILD: 0.0 <\D><D>NPA: 0.0<\D><D>List of smells detected:<D>Insufficient Modularization<\D><D>Multifaceted Abstraction<\D><\D><D>TOTAL OF SMELLS DETECTED: 2<\D><\D><D><D>TYPE: javaProject.one.A<\D><D>SLOC: 5 <\D><D>NOM: 0 <\D><D>NPM: 0 <\D><D>WMC: 1<\D><D>DEP: 1 <\D><D>I-DEP: 1 <\D><D>FAN-IN: 1 <\D><D>FAN-OUT: 1 <\D><D>NOA: 1 <\D><D>LCOM3: 0.00 <\D><D>DIT: 1.0 <\D><D>CHILD: 0.0 <\D><D>NPA: 0.0<\D><D>List of smells detected:<D>Cyclically-dependent Modularization<\D><\D><D>TOTAL OF SMELLS DETECTED: 1<\D><\D><D><D>TYPE: javaProject.com.controller.Type<\D><D>SLOC: 245 <\D><D>NOM: 35 <\D><D>NPM: 25 <\D><D>WMC: 58<\D><D>DEP: 7 <\D><D>I-DEP: 2 <\D><D>FAN-IN: 0 <\D><D>FAN-OUT: 9 <\D><D>NOA: 13 <\D><D>LCOM3: 0.85 <\D><D>DIT: 1.0 <\D><D>CHILD: 0.0 <\D><D>NPA: 0.0<\D><D>List of smells detected:<D>Insufficient Modularization<\D><D>Multifaceted Abstraction<\D><\D><D>TOTAL OF SMELLS DETECTED: 2<\D><\D><D>TOTAL OF TYPES WITH SMELLS: 5<\D><\D>”

4. QUALITATIVE ANALYSIS OF USE CASES

4.3 USE CASE 3: USING METHOD METRICS TO PROVIDE INSIGHTS

Just for reader information, below are the metrics of the code analyzed in section 4.3, once in the main text was just provided a small part of the methods. Here as well is just presented the methods with over 15 LOC as total number would be unnecessary.

Table A.4.1 – Method metrics from the methods used on subsection 4.3's analysis (only methods over 15 LOC, just 79 of 1185 methods)

method	**LOC**	**CYCLO**	**CALLS**	**NBD**	**PARAM**
chatGPTIntegration.GPTintegration.GPTinsight()	94	5	12	3	0
fixtures.output.DataFixture.getMethodData()	59	1	56	0	0
output.MetricResultGPT.showSummary()	57	1	70	3	0
fixtures.output.DataFixture.getThresholdData()	51	1	25	0	0
output.MetricResultJSON.generateSummary()	49	1	57	2	0
structures.metrics.MetricThreshold.MetricThreshold()	49	1	25	1	0
javaProject.com.controller.Type.isSubtype(String dottedSubtype, String collectionType)	45	7	3	2	2
utils.files.SourceCodeLineCounter.isSourceCodeLine(String line)	42	10	13	4	1
output.MetricResultGPT.showTypes()	42	3	46	3	0
fixtures.output.DataFixture.getTypesResonance()	37	1	23	0	0
javaProject.com.controller.Type.addClassAndGetClassVertex(XClass xclass)	32	6	5	2	1
output.MetricResultGPT.showDependencies()	31	4	16	3	0
output.MetricResultCSV.generateSummary()	31	1	53	1	0
structures.results.TypeMetricResultTest.createTypes()	31	1	46	1	0
fixtures.statistics.StatisticOfTypeFixture.createTypes()	31	1	46	0	0
output.MetricResultGPT.showInternalDependencies()	30	4	19	3	0
output.MetricResultGPT.showMethods()	30	3	22	3	0
output.utils.InfoConsole.printCommands()	30	1	28	0	0
utils.files.SourceCodeLineCounter.getNumberOfLines(BufferedReader bReader)	28	8	10	4	1
output.MetricResultConsole.showSummary()	28	1	38	2	0
output.MetricResultGPT.showNamespaceCoupling()	27	2	21	3	0
utils.files.SourceCodeLineCounter.commentBegan(String line)	25	5	9	4	1
chatGPTIntegration.ChatGPTAPI.requesttoGPT(String body, HttpURLConnection connection)	25	2	12	2	2
chatGPTIntegration.ChatGPTAPI.createChatGPTAPIConnection()	25	1	8	2	0
javaProject.com.controller.Type.traverseSupertypes(ClassDescriptor start)	24	2	5	2	1
output.utils.InfoConsole.printMetrics()	24	1	23	0	0
output.MetricResultConsole.showInternalDependencies()	23	4	15	2	0
chatGPTIntegration.ChatGPTAPI.requestGPTClosure(String body, HttpURLConnection connection)	23	4	10	2	2

chatGPTIntegration.ChatGPTAPI.requestGPTKeepOpen(String body, HttpURLConnection connection)	23	4	8	2	2
utils.files.SourceCodeLineCounter.commentEnded(String line)	22	4	5	4	1
main.Bootstrapper.verifyInvalidOptions()	21	4	4	2	0
javaProject.com.controller.Type.computeKnownSubtypes(ClassDescriptor classDescriptor)	21	2	3	2	1
selection.options.Options.Options()	21	1	19	0	0
output.MetricResultGPT.showCyclicDependencies()	20	3	7	3	0
fixtures.output.JSONDataFixture.generateTypes()	20	2	27	1	0
fixtures.output.JSONDataFixture.getDependencies()	20	1	18	1	0
fixtures.output.JSONDataFixture.getInternalDependencies()	20	1	18	1	0
fixtures.output.DataFixture.getTypeData()	20	1	17	0	0
main.Bootstrapper.getOutputFormat(long startTime)	19	5	8	2	1
output.MetricResultConsole.showNamespaceCoupling()	19	2	18	2	0
fixtures.output.CSVDataFixture.generateStatisticalMethod()	19	2	35	1	0
fixtures.output.CSVDataFixture.generateStatisticalNamespace()	19	2	35	1	0
fixtures.output.CSVDataFixture.generateStatisticalType()	19	2	35	1	0
fixtures.output.JSONDataFixture.generateStatisticalMethod()	19	2	24	1	0
fixtures.output.JSONDataFixture.generateStatisticalNamespace()	19	2	24	1	0
fixtures.output.JSONDataFixture.generateStatisticalType()	19	2	24	1	0
structures.results.TypeMetricResult.cleanListOf(Set<String> typesWithCyclos)	18	6	5	3	1
chatGPTIntegration.GPTintegration.printGPTResponse(String prompt)	18	4	10	3	1
structures.results.TypeMetricResult.getCyclicDependencies()	18	3	7	3	0
output.MetricResultGPT.showNamespaces()	18	2	18	3	0
output.MetricResultJSON.generateNamespaceCoupling()	18	2	22	2	0
output.MetricResultCSV.generateNamespaceCoupling()	18	2	17	1	0
output.MetricResultFile.show()	18	1	16	0	0
structures.results.TypeMetricResult.getTotalOfVariablesUsedInMethods(TypeMetric type)	17	5	5	3	1
chatGPTIntegration.ChatGPTAPI.chatGPTConversation(String[] prompt)	17	4	4	2	1
output.MetricResultCSV.getStatisticalMetrics(List<StatisticMetricResult> list)	17	2	34	1	1
structures.statistics.StatisticOfType.compute()	17	2	14	1	0
parser.java.visitors.TypeVisitor.visit(TypeDeclaration node)	16	3	11	2	1
output.MetricResultJSON.getStatisticalMetrics(List<StatisticMetricResult> list)	16	2	23	2	1

output.MetricResultConsole.showTypes()	16	2	20	2	0
fixtures.output.JSONDataFixture.generateMethods()	16	2	16	1	0
fixtures.output.JSONDataFixture.generateNamespaceCoupling()	16	2	16	1	0
fixtures.output.JSONDataFixture.generateThresholds()	16	2	14	1	0
structures.results.StatisticMetricResult.StatisticMetricResult(String acronym, double average, double median, double amplitude, double firstQuartile, double thirdQuartile, double standardDeviation, double lowerFence, double upperFence, double interQuartileRange, double minValue, double maxValue, double threshold)	16	1	0	0	13
structures.results.TypeMetricResult.getInternalImportsBy(String namespace)	15	6	8	3	1
structures.results.TypeMetricResult.getTotalOfAbstractTypesIn(String namespace)	15	6	6	3	1
javaProject.com.model.Man.foo()	15	5	1	1	0
chatGPTIntegration.promptEngineer.promptContext()	15	5	0	0	0
chatGPTIntegration.promptEngineer.promptInstruction()	15	5	0	0	0
output.MetricResultConsole.showDependencies()	15	4	12	2	0
chatGPTIntegration.ChatGPTAPI.printExperimentMultipleRequest(String prompt, String[] prompts)	15	4	10	2	2
parser.java.visitors.MethodVisitor.defineParameters()	15	3	11	2	0
fixtures.output.JSONDataFixture.generateSummary()	15	2	14	1	0
output.MetricResultFileTest.deleteFiles()	15	1	14	1	0
output.MetricResultFileTest.setUp()	15	1	14	1	0

Source: Elaborated by the author

APPENDIX B – TRABALHO DE GRADUAÇÃO 1

Integration of Chat GPT to Software Engineering Smell detection tool

Glauber de Souza Rosa[1], **Guilherme Lacerda**[1], **Marcelo Pimenta**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{glauberosadesouza@gmail.com, guilhermeslacerda@gmail.com, mpimenta@inf.ufrgs.br

Abstract. *Develop and analyze the efficiency of integrating ChatGPT (and how to use it efficiently) to a software analyzing suite for code "Smells" detection and code quality evaluation, so that the integration provides guidance to software engineers or even some level of automation on its refactoring work.*

Resumo. *Desenvolver e analisar a eficiência de integrar o Chat GPT (e como usá-lo de maneira mais eficaz) à uma suíte de ferramentas de analise de código (métricas, indicadores) sobre code smells e qualidade de código, de maneira que tal integração produza orientações para o engenheiro de software ou até mesmo algum nível de automação no seu trabalho de refactoring.*

1. Introduction

Software maintenance and constant updates are currently a big part of the software development, as it is needed to comply with new regulatory requirements or corrections to adapt to new needs. Currently we are at a point that much of our codes used through out applications are over 10 years old and on some cases more than 25 years old, so that software maintenance becomes by each day a more fundamental part of our society.

On the near future, there will be needed some major maintenance such as add support to addition of digits to US phone number or US Social Security numbers. We already have similar situations in the past, like the **Year 2000 software bug**, which is estimated that over 75 percent of all software applications were affected by the issue.

This highlights the importance of keeping the software easy to maintain and have tools and automation to help to keep legacy code with quality. A study from 2014 [Jones 2006] estimates that soon the number of professionals working on maintenance comparing to new developments would top 75 percent of all IT professionals working with software engineering.

By mid-21st century, maintenance costs could tops five trillion dollars in overall, which makes clear the need for better maintenance tools and technologies to support these activities.

The predecessor work explored on this project covered analytical part of software maintenance through smells and refactoring metrics, which lead to Dr. Tools Suite presented on [Guilherme Lacerda 2023], where the tool presents data that provides statistics about software Smells and refactoring opportunities for more efficient code maintenance.

At the same time that we have this growing need for software maintenance, we see new advances on a different area which can be brought to assist on it, AI and Large Language models, like ChatGPT.

ChatGPT is a interactive AI released on 2022 and that in a few months proved to be a powerful tool to problem solving and creative creation through a precise prompt engineer to guide it. Such tool were already used on multiple creative activities automation with different success rate, though we see potentially to assist on the software maintenance through provide insights on maintenance or even provide automation.

On this work, we will integrate the two technologies referred above to provide with the software engineering community with a guidance to how to leverage both and take the best synergy possible from them, either by using ChatGPT with the provided statics from DR-Tool to provide insights to what software engineer should look into and what could be done, to even scenarios in which some automation could be implemented.

2. Biographical Review

Through this section, we will review similar and predecessor researches to develop the tools that this project will be studying on how to best integrating the two technologies, DR-Tools and ChatGPT.

2.1. Code Smells and Refactoring

2.1.1. Code Smells

Smell is a concept that is used on software engineering for a software problem that is not the same as a bug that would generate a failure, but is a problem that can impact the software maintenance and future enhancements through increased complexity, for example [Guilherme Lacerda 2020].

The term "smells" became popular initially with the agile software development and was popularized due to the original work of [Fowler 1999], which was pioneer in the come smell identification and provided techniques to solve them.

Smells can be divided on lower level, known as code level [Fowler 1999], or higher level, known as design level [Brown 1998].

[Fowler 1999] has presented originally 22 code smells with proposed ways to have it refactored, later the list was extended by researches like [Fowler 2018]. On Table 1, are presented the referred smells from the original work and the 6 additions proposed.

[Brown 1998] on his work presented anti-patterns that could be divided in on development, architecture and project management design smells. The anti-patterns describe the common occurrences that could result in negative consequences through out the code life-cycle. On table 2 is presented a list of the main design smells according to [Brown 1998].

There are sub-sequential works, for example, [Wake 2003] and [Kerievsky 2004] that expanded with addition of other smells and different perspectives, though this will not be further detailed as the main smells and perspective were covered already and we will initially focus on them.

Table 1. List of code smells presented by [Fowler 1999] and [Fowler 2018].

Smell	Description
Duplicated Code	Consists of equal or very similar passages in different fragments of the same code base
Long Method/Long Function	Very large method/function and, therefore, difficult to understand, extend and modify. It is very likely that this method has too many responsibilities, hurting one of the principles of a good OO design (*SRP: Single Responsibility Principle* (Martin and Martin, 2007))
Large Class	Class that has many responsibilities and therefore contains many variables and methods. The same *SRP* also applies in this case
Long Parameter List	Extensive parameter list, which makes it difficult to understand and is usually an indication that the method has too many responsibilities. This smell has a strong relationship with *Long Method*
Divergent Change	A single class needs to be changed for many reasons. This is a clear indication that it is not sufficiently cohesive and must be divided
Shotgun Surgery	Opposite to *Divergent Change*, because when it happens a modification, several different classes have to be changed
Feature Envy	When a method is more interested in members of other classes than its own, is a clear sign that it is in the wrong class
Data Clumps	Data structures that always appear together, and when one of the items is not present, the whole set loses its meaning
Primitive Obsession	It represents the situation where primitive types are used in place of light classes
Switch Statements/Repeated Switches	It is not necessarily smells by definition, but when they are widely used, they are usually a sign of problems, especially when used to identify the behavior of an object based on its type
Parallel Inheritance Hierarchies	Existence of two hierarchies of classes that are fully connected, that is, when adding a subclass in one of the hierarchies, it is required that a similar subclass be created in the other
Lazy Class	Classes that do not have sufficient responsibilities and therefore should not exist
Speculative Generality	Code snippets are designed to support future software behavior that is not yet required
Temporary Field	Nember-only used in specific situations, and that outside of it has no meaning
Message Chains	One object accesses another, to then access another object belonging to this second, and so on, causing a high coupling between classes
Middle Man	Identified how much a class has almost no logic, as it delegates almost everything to another class
Inappropriate Intimacy	A case where two classes are known too, characterizing a high level of coupling
Alternative Classes with Different Interfaces	One class supports different classes, but their interface is different
Incomplete Class Library	The software uses a library that is not complete, and therefore extensions to that library are required
Data Class	The class that serves only as a container of data, without any behavior. Generally, other classes are responsible for manipulating their data, which is a case of *Feature Envy*
Refused Bequest	It indicates that a subclass does not use inherited data or behaviors
Comments	It cannot be considered a smell by definition but should be used with care as they are generally not required. Whenever it is necessary to insert a comment, it is worth checking if the code cannot be more expressive
Mysterious Name	Non-significant names that do not represent the software elements
Global Data	It can be modified from anywhere in the code base, and there's no mechanism to discover which bit of code touched it
Mutable Data	It changes to data can often lead to unexpected consequences and tricky bugs
Lazy Element	Software elements designed to grow, but do not conform with software evolution
Insider Trading	Coupling problems caused by trade data between modules

Table 2. List of design smells presented by [Brown 1998].

Smell	Description
Blob	As know as *God Class*, is a style of procedural design procedural which brings an object to have too many responsibilities (*Controller*) and attributes with low cohesion, while others only save data or execute simple processes
Lava Flow	Dead code and forgot information frozen with design
Functional Decomposition	A procedural code in a technology that implements the OO paradigm (usually the main function that calls many others), caused by the previous expertise of the developers in a procedural language and little experience in OO
Poltergeist	Classes that have a role and life cycle very limited, frequently starting a process for other objects
Spaghetti Code	Use of classes without structures, long methods without parameters, use of global variables, in addition to not exploiting and preventing the application of OO principles such as inheritance and polymorphism
Cut and Paste Programming	Reused code by a copy of code fragments, generating maintenance problems
Swiss Army Knife	Exposes the high complexity to meet the predictable needs of a part of the system (usually utility classes with many responsibilities)

2.1.2. Refactoring

Refactoring, as defined by [Guilherme Lacerda 2020], is the primary approach to remove smells. Refactoring is the reorganization strategies to support change in software to help to improve code quality by making it more readable, efficient and/or eliminating possible problems, these strategies were introduced by [Opdyke 1992].

Refactoring can be done on different levels of abstraction and on different software entities. For example, it can be done a refactoring on the UML models, database schemes, software architecture, requirements and language structure [Mens 2003]. As the refactoring does not change the purpose or the behavior of the software, it can be done on different levels to achieve the best results to have the code supported in the future, which means that different techniques can be used and often be used in a sequence to improve the quality, though its sequence is arbitrary.

Refactoring is usually divided at two levels as smells: high-level (composite refactoring) and low-level (primitive refactoring). High-level refactoring consist on significant and structural design changes at a macro or architectural level, while low-level are small and specific code changes. [Opdyke 1992] work defined that to do a high-level refactoring a low-level refactoring will be required, as well as for both it was introduced the fundamental element for the refactoring that is precondition. The concept of precondition is that you need to establish preconditions that are checked before applying the transformations and after applied, these conditions are rechecked to guarantee that the behavior of the code is not altered by the refactoring changes, having the same preconditions.

The key importance of performing refactoring on code that do not present bugs, is that 40 percent of the time invested in software maintenance is the cost to understand the code and its architecture [Telea 2011]. One key strategy is to invest on automation and provide tools for developers to detect refactoring opportunities (or smells), so that the process can be optimized.

2.2. DR-Tools Suite

DR-Tools Suite is a set of lightweight open-source tools that provide resources and information to improve source code quality, supporting the developer in his daily work. DR-Tools Suite was inspired by the medicine metaphor [Guilherme Lacerda 2023].

DR-Tools Suite consist of 2 tools: DR-Tools Metric, which is a command-line Interface (CLI) tool that collects and shows different source code metrics, and DR-Tools Metric Visualization, which is a tool to provide a visual feedback through different graphical formats from the data generated by DR-Tools Metric. DR-Tools Metric will be the tool that will be the focus of this work and for that reason we will only further develop on it.

2.2.1. DR-Tools Metric

DR-Tools Metric is designed to provide a well-known set of combined metrics from software metrics research as well as define a set of heuristics for the combination of metric based on relationships and thresholds. With this the tool calculates metrics and provides insights from the source code, so that this can help developers to learn about software complexity, smells and refactoring opportunities.

To better understand the source code, a key point is to have the code metrics and its correlations clear, as shown on studies like [D. Radjenovic 2013]. For example, the relation between size metrics and Object Orientation metrics help analyze aspects of code maintainability. Also according to [M. A. Bigonha 2019], when a metric is associated with some threshold, it facilitates its use and understanding.

DR-Tools Metric analyzes the source code and provides the results in different formats (line command, CSV, and JSON) to be used in different contexts. There is no necessary any configuration or installation of any complementary software or plug-in to use the tool.

DR-Tools Metric provides 33 metrics contextualized by project summary, namespaces (packages), types (classes), methods, dependencies, and coupling (namespace and type). The following is the list of metrics by context:

- **Summary (9):** Total of namespaces, total of types, mean number of types/namespaces, total of lines of code (SLOC), mean number of SLOC/types, total of methods, mean number of methods/types, total of complexity (CYCLO), and mean number of complexity/types;
- **Namespaces (2):** Number of classes/types (NOC) and number of abstract classes (NAC);
- **Types (9):** Lines of code (SLOC), number of methods (NOM), number of public methods (NPM), class complexity (WMC), number of dependencies(DEP), number of internal dependencies (I-DEP), number of other types that depend on a given type (FAN-IN), number of other types referenced by a type (FAN-OUT), and number of fields/attributes (NOA);
- **Methods (5):** Lines of code (MLOC), cyclomatic complexity (CYCLO), number of invocations (CALLS), nested block depth (NBD), and number of parameters (PARAM);
- **Namespace Coupling (5):** Afferent coupling (CA), efferent coupling (CE), instability (I), abstractness degree (A), and normalized distance (D);
- **Type Coupling (4):** number of dependencies (DEP), number of internal dependencies (I-DEP), number of other types that depend on a given type (FAN-IN), and number of other types referenced by a type (FAN-OUT);
- **Dependencies (3):** General dependencies (DEP), internal dependencies (I-DEP), and cyclic dependencies.

The tool also provides its users with the flexibility to combine and query contextual information, from general information (summary), information about packages, classes, methods, dependency types, couplings, and reference thresholds of metrics. When presenting the results, the data are sorted according to the context. For example, when presenting information about classes, data is sorted by lines of code, complexity, and number of methods or when presenting about methods, the combination is cyclomatic complexity, nested blocks, lines code, and invocations.

It is also possible to filter contextualized results using the **–top** option. Like this, it is easier for developers to analyze the source code and filter out the most problematic elements. As presented in Figure 1, it is possible to have a view on summary and packages, more complex classes, and methods (showing the first 5), in a single option.

The tool is currently only developed to analyze Java code, but its architecture is designed to allow simple enhancement to other languages by developing a parser and corresponding visitor to the new language.

DR-Tools Metric is designed to be independent of environments and platforms, facilitating interoperability. Its open architecture allows both the functionalities and the resulting data in known standardized formats being integrated with other tools, without additional installation or configuration. The tool research is a work in progress and is intended to be expanded with new tools, like refactoring recommendation and other tools to support code review. At this place that this work will be connected.

2.3. Chat GPT

ChatGPT, released by OpenAI in November 2022, is a large-scale language model that once made available reached 100 milion users in 3 months and have over 25 milion daily users.

OpenAI's GPT (generative pre-trained transformer) models, the tool behind ChatGPT, have been trained to understand natural language and code, in a way that when provided with a text input, it provides a text output in response. These inputs are referred to as ”prompts” and its designing is essentially how you extract the answers from a GPT model, which will directly influence on its answer and accuracy.

According to OpenAI, GPTs can be used across a great variety of tasks including content or code generation, summarization, conversation, creative writing, and more.

2.3.1. Chat GPT APIs

OpenAI, through its site makes available to developers an public API (application programming interface) that can be used to access the GPT resources. It works by sending a request containing the inputs and the developer API key, and receive a response containing the model's output. The latest models, gpt-4 and gpt-3.5-turbo, are accessed through the chat completions API endpoint.

With the key generated at https://platform.openai.com (Figure 1 below), it possible to can send requests using https://api.openai.com/v1/chat/completions endpoint through HTTP requests from any language.

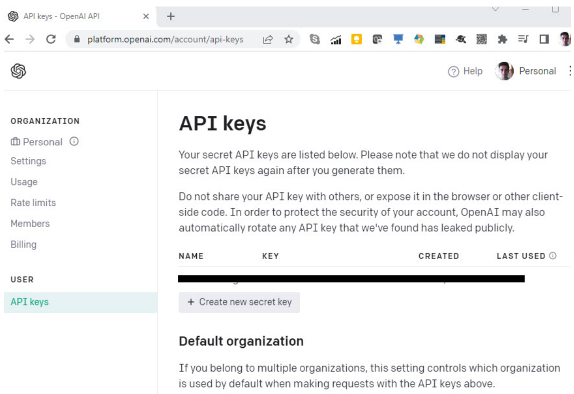


Figure 1. ChatGPT API Key

2.3.2. Chat GPT on Development

Currently is being studied the implications and the applicability of ChatGPT to code development and to support software activities. According to studies and recent results of the ChatGPT, increased interest is on the area of automation of software development tasks in a way to improve the developers to make their tasks more efficiently [Tyna Eloundou and Rock 2023].

Tools like ChatGPT are leading impressive results, on both quantity and quality, producing outcomes (e.g., code) that are in some cases comparable levels to what humans produce. For example, [Mehdi Golzadeh and Chidambaram 2023] investigations in large open-source projects on GitHub concluded that bots are among the biggest and most active contributors, without being labeled as bots.

On the empirical study done over quality of code between developers and tools like ChatGPT performed by [Nathalia Nascimento 2023], the result was that in certain scenarios the performance of ChatGPT has outperformed new software engineers in specific tasks, though this was more specific on solving easy to medium-level tasks/problems, when the ChatGPT consistently outperformed the new software engineer.

On the other hand, the same study has concluded that there is decisive evidence to support the theory that ChatGPT would outperform an experienced developer in terms of solution performance. In summary, the study reveal a dynamic interplay between human and AI performance and the need to a collaborative approach to fine-tone the AI inputs based on the developer expertise at the same time that improve efficiency via automatons through the AI.

2.3.3. ChatGPT Prompt Patterns for Improving Code Quality

As already stated by OpenAI itself, a key point of good usage of Large Language Models (LLM), like ChatGPT, is to provide a good prompt and context to the request. So this is an area of study that is being developed and we will explore in this research.

Though there is several patterns and they can take various forms, to perform software engineering tasks, according to [Jules White 2023], it is typically better to start with a scoping statement, like ”from now on”, act as a X”, ”for the next four prompts”. On [Jules White 2023], it was investigated and proposed 13 prompt patterns for different software engineering tasks. They were documented, tested and analysed with the below format:

- **A name and classification:** provides a clear name to be identified and classified the pattern based on the type of problem to be solved. The prompt patterns proposed can be view on Table 3;
- **The intent and context:** summarises of the problem to be solved and its goal;
- **Motivation:** explains the importance of the problem to be solved;
- **The structure and key ideas:** describes the fundamentals of the pattern and the context that need to be provided to the LLM to achieve the expected resolution;
- **Example implementation:** shows an example of the patter implemented and discusses it;
- **Consequences:** evaluates the pros and cons of using the pattern and how to adapt the pattern to other scenarios.

On this research our focus will be more specifically on the Refactoring piece as the goal is to integrate ChatGPT to provide the refactoring technique to be followed and study what can be provided by ChatGPT. The research already done on the topic by [Jules White 2023] supports the intention of our project as according to it, tools like ChatGPT have a surprisingly powerful understanding of abstract coding constructs and can delivery innovative approaches to code refactoring.

- **The Pseudo-code Refactoring Pattern:**

Table 3. Classifying Prompt Patterns for Automating Software Engineering Tasks by [Jules White 2023].

Requirements Elicitation	Requirements Simulator Specification Disambiguation Change Request Simulation
System Design and Simulation	API Generator API Simulator Few-shot Example Generator Domain-Specific Language (DSL) Creation Architectural Possibilities
Code Quality	Code Clustering Intermediate Abstraction Principled Code Hidden Assumptions
Refactoring	Pseudo-code Refactoring Data-guided Refactoring

This method consists in basically provide the LLM (ChatGPT) with the pseudo-code structure desired and have the AI refactor to adapt to the specific situation. The pattern will follow as the below Figure 2.

	Pseudo-code Refactoring Pattern
1.	Refactor the code
2.	So that it matches this pseudo-code
3.	Match the structure of the pseudo-code as closely as possible

Figure 2. Pseudo-code Refactoring Patter by [Jules White 2023]

One important considerations is that in case the pseudo-code requires an extensive description and precising code, the usage of the LLM could be not advantageous as its benefits will be reduced by the required coding of the pseudo-code to specify the refactoring. This pattern can also lead to substantial refactoring and because of that requires to the code to be split or have functions removed, which could impact on its public interface be changed and require further refactoring.

Important to note that this methodology would not be the best fit for this research, as we will be integrating with a data generating tool and not a pseudo-code tool.

- **The Data-guided Refactoring Pattern:**

On this pattern, the idea is that to provide the data that needs to be changed and the LLM will do the refactoring to have the data the closest to the requested. Though it is not exactly to have it matching the smells and source code metrics, but code results and formats, we believe this might be the best pattern to fit our scenario. Figure 3 shows how this pattern would be structured.

The results from the [Jules White 2023], indicate that this pattern reduces the manual effort to refactor many types of code changes. On many cases the refactor can

	Data-guided Refactoring Pattern
1.	Refactor the code
2.	So that its input, output, or stored data format is X
3.	Provide one or more examples of X

Figure 3. Data-guided Refactoring Patter by [Jules White 2023]

be completed automatically or at least be a booster and speed up the refactor causing potentially a cost reduction on the change of data formats for example.

In general this last study concluded that the depth of the capabilities of LLMs, like ChatGPT, are not fully understood or appreciated as the tool holds a lot of potential to software engineering automation through out the software life-cycle. The conclusion is that the key to leverage all this capabilities is to codify an effective catalog of prompts and guidance on how to combine this patterns to improve the software engineering through automation. At the same time, it is highlighted the significance of human involvement and expertise as currently ChatGPT has a tendency to ”hallucinate” confidently, so guidance and scrutiny is required to mitigate these possible issues.

In conclusion, the tools have a lot of potential, but it is required much research and development on prompt pattern engineering to have the best results provided and all potential be fulfilled.

3. Proposed Work

As highlighted through the Introduction and the literature review, the goal of this project is to investigate and develop a integration of the DR-Tools with ChatGPT, both introduced and explained earlier. Therefore it will be studied how to use the data provided from DR-Tools and how to better deliver value to the software engineering on the process of refactoring through these technologies.

The simple integration and asking to ChatGPT is not the solution as studies reviewed show that the prompt engineering plays an important role on the quality of the ChatGPT responses and problem solving capability. With this in mind, we will design how that integration is going to be done and evaluate the results of it through out this work, to at the end provide a case study and a possible best practice for similar future works.

To follow best practices on the tool integration development, we already are starting from a high-level idea of architecture, which is presented on Figure 4.

3.1. Schedule of Activities

To do so, we will divide this work on the 8 parts below (which might be further change through the research process):

1. Developing the Class/Component to handle the API calls to ChatGPT;
2. Developing the Prompt engineering Class/Component to create the prompts that will be used to call ChatGPT;
3. Developing the integration Class/Component to retrieve data from Dr. Tool and provide it to the prompt engineering;

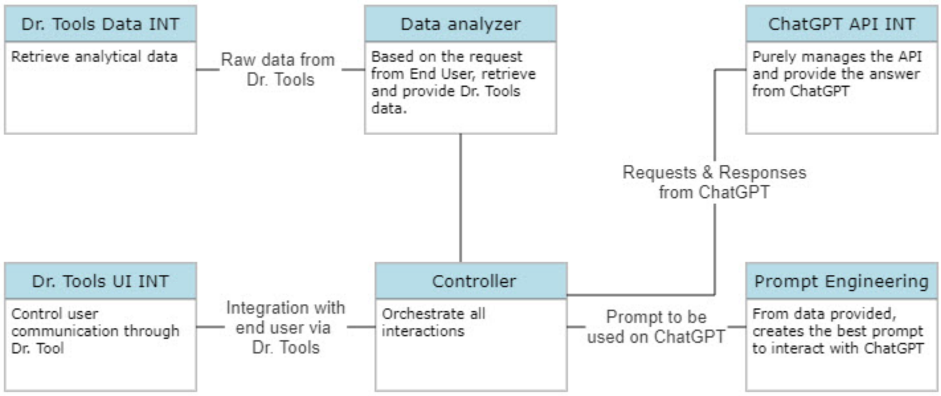


Figure 4. Initial proposed high-level architecture for ChatGPT Integration

4. Developing the UI integration Class/Component;
5. Testing and interactively improving on the different elements of this project;
6. Describing and detailing design and architecture with explanation on what could be achieved and the reasons for the decision;
7. Analyzing the end results and provide a overview of the success and leanings;
8. Thesis writing;
9. Presentation preparation;

For development parts (1 through 5), we will approach on the following steps:

1. Reviewing previous works and established best practices;
2. Defining high-level architecture and concepts to be used on the activity;
3. Code Development and testing;
4. Validating results;

We will consider the implementation phase starting on second week of August, calendar week 31 (CW31), taking 16 weeks and ending on calendar week 48 (CW45), as planned schedule on figure 5.

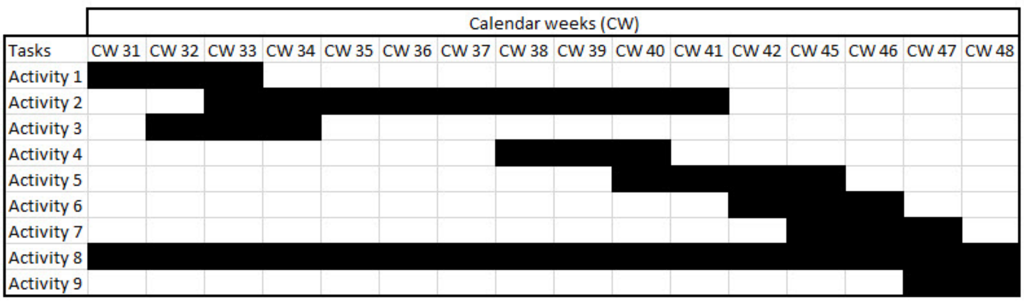


Figure 5. Proposed work schedule

4. Conclusion

On this report, we introduced the problem/opportunity, reviewed literature on the relevant subjects of this study and presented the proposed solution and its methodology. On our review we started by reviewing and introducing the concepts of Smells and Refactoring, to later connect with the tool that we will be working on, Dr-Tools.

We introduced and reviewed the concept of Large Language Models (LLM) and ChatGPT, also contextualized the importance and the concept of prompt engineering, which will be the biggest challenge of this project and as such we expect to have it taking majority of the project time. We tied together the two areas by the proposal work of integrating the existing Dr-Tools and its data to provide refactoring though use ChatGPT with the correct prompt engineering, which our review presented as key to effectiveness of the ChatGPT.

We expect that at the end of this work, we will be able to:

- Provide a best practice to prompt engineering for refactoring and smell removal;
- Improve DR-Tools capability with a new automation with ChatGPT;
- Evaluate the effectiveness of ChatGPT and if the tool is able to accomplish this proposed task;
- Provide a forward view on how ChatGPT can be further explored;

References

Brown, W.H., M. R. I. H. M. T. (1998). Antipatterns: Refactoring software, architectures, and projects in crisis. John Wiley and Sons, Inc.

D. Radjenovic, M. Hericko, R. T. A. (2013). Software fault prediction metrics: A systematic literature review. Information and Software Technology 55.

Fowler, M., B. K. B. J. (2018). Refactoring: Improving the design of existing code - second edition. Pearson.

Fowler, M., B. K. B. J. O. W.-R. D. (1999). Refactoring: Improving the design of existing code. Addison-Wesley.

Guilherme Lacerda, Fabio Petrilloc, M. P. (2023). Dr-tools: a suite of lightweight open-source tools to measure and visualize java source code. Publishing Press.

Guilherme Lacerda, Fabio Petrilloc, M. P. Y. G. G. (2020). Code smells and refactoring: A tertiary systematic review of challenges and observations. In *The Journal of Systems and Software*. Publishing Press.

Jones, C. (2006). The economics of software maintenance in the twenty first century. Software Productivity Research, Inc.

Jules White, Sam Hays, Q. F. J. S.-S. D. C. S. (2023). Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. arXiv, Cornell University.

Kerievsky, J. (2004). Refactoring to patterns. Addison-Wesley.

M. A. Bigonha, K. Ferreira, P. S. B. S. M. J. D. L. (2019). The usefulness of software metric thresholds for detection of bad smells and fault prediction. Information and Software Technology 115.

Mehdi Golzadeh, Tom Mens, A. D. E. C. and Chidambaram, N. (2023). Recognizing bot activity in collaborative software development. IEEE Software.

Mens, T., D. S. B. B. S. H. G. P. (2003). Refactoring: Current research and future trends. In *Electronic Notes in Theoretical Computer Science 82*. LDTA'2003 - Language descriptions, Tools and Applications.

Nathalia Nascimento, Paulo Alencar, D. C. (2023). Comparing software developers with chatgpt: An empirical investigation. arXiv, Cornell University.

Opdyke, W. (1992). Refactoring object-oriented frameworks. University of Illinois at Urbana-Champaign.

Telea, A., V. L. (2011). Visual software analytics for the build optimization of large-scale software systems. In *Comput. Stat. 26*. Comput. Stat. 26.

Tyna Eloundou, Sam Manning, P. M. and Rock, D. (2023). Gpts are gpts: An early look at the labor market impact potential of large language models. arXiv, Cornell University.

Wake, W. (2003). Refactoring workbook. Addison-Wesley.