UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO HENRIQUE DOS SANTOS SOARES

USO DE EXECUÇÃO DE CÓDIGO PARA DETECTAR PROBLEMAS DE DESIGN EM ATIVIDADES DE PROGRAMAÇÃO OO

CAMPINA GRANDE - PB

2022

JOÃO HENRIQUE DOS SANTOS SOARES

USO DE EXECUÇÃO DE CÓDIGO PARA DETECTAR PROBLEMAS DE DESIGN EM ATIVIDADES DE PROGRAMAÇÃO OO

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Matheus Gaudencio do Rêgo.

CAMPINA GRANDE - PB

2022

JOÃO HENRIQUE DOS SANTOS SOARES

USO DE EXECUÇÃO DE CÓDIGO PARA DETECTAR PROBLEMAS DE DESIGN EM ATIVIDADES DE PROGRAMAÇÃO OO

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Matheus Gaudencio do Rêgo

Orientador – UASC/CEEI/UFCG

Professora Dr. Eanes Torres Pereira

Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 11 de Abril de 2022.

CAMPINA GRANDE - PB

ABSTRACT

At the Federal University of Campina Grande, the Programming Laboratory 2 course is used for students to learn to program and understand code structuring. The practical evaluations are applied through laboratories, where the correction is made by tests and manual analysis of the code structure. In this process, feedback is essential for the formation of the student, however, the excess of laboratories makes quick feedback impossible. Automated tests and automatic analysis of code metrics are alternatives to quickly generate useful information for the student, but many of the existing tools are limited in their ability to evaluate a code structure. With the execution of the TraceAgent tool, an analysis will be carried out in order to identify problems related to the code design. In this context, we want to see if metrics extracted from code execution like X, Y, Z can be useful information for quick assessment of students.

USO DE EXECUÇÃO DE CÓDIGO PARA DETECTAR PROBLEMAS DE DESIGN EM ATIVIDADES DE PROGRAMAÇÃO OO

João Henrique dos Santos Soares
joao.soares@ccc.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil

Matheus Gaudencio do Rêgo
matheusgr@computacao.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil

RESUMO

Na Universidade Federal de Campina Grande, a disciplina Laboratório de Programação 2 é usada para os alunos aprenderem a programar e entenderem sobre a estruturação do código. As avaliações práticas são aplicadas através de laboratórios, onde a correção é feita por testes e uma análise manual da estrutura do código. Neste processo, o feedback é essencial para a formação do aluno, porém, o excesso de laboratórios impossibilita um feedback rápido. Testes automáticos e análise automática de métricas de código são alternativas para a geração rápida de informação útil para o aluno, entretanto muitas das ferramentas existentes são limitadas na sua capacidade de avaliar uma estrutura de código. Com a execução da ferramenta TraceAgent, uma análise será feita para conseguir identificar problemas relacionados ao design de código. Neste contexto, desejamos observar se métricas extraídas da execução de código como X, Y, Z podem ser informações úteis para avaliação rápida do aluno.

PALAVRAS CHAVE

Ensino de Programação, Análise de código, Programação Orientada a Objetos

1 INTRODUÇÃO

O Laboratório de Programação 2 é uma das disciplinas essenciais para os estudantes de computação. Nela é possível criar uma base sólida de Programação Orientada a Objetos (POO) por atividades práticas de 2 ou 3 semanas de duração. Porém, para alguns alunos a dificuldade no aprendizado se reflete na reprovação, desistência ou manutenção de falhas conceituais. Sabendo disso, é essencial que a equipe de ensino consiga dar um feedback o mais rápido possível para os alunos, assim sobrando mais tempo para que os alunos consigam tirar suas dúvidas com os tutores.

Entretanto, um feedback de qualidade demanda tempo. Uma maneira de contornar esse problema é conseguir um feedback de forma automática. Essa pesquisa pretende analisar o rastro de execução de código com uma base de testes automáticos e verificando a possibilidade que a ferramenta de trace possa ajudar a identificar problemas de código. O impacto dessa pesquisa é um feedback mais rápido em relação às atividades de laboratório de programação.

2 FUNDAMENTAÇÃO

Nesta Seção descreveremos a revisão de 4 artigos da literatura e comparamos com nosso trabalho.

2.1 COSMIC Functional Size Automation of Java Web Applications Using the Spring MVC Framework [5]

O artigo é voltado para análise de código de arquitetura MVC para uma aplicação web. Para realizar esta análise de forma automática o usuário deve deixar no padrão que o algoritmo deseja. Portanto, aborda a avaliação do design de código proposta a partir da análise estática de código de projetos com as seguintes características: Java versão 5 ou superior; aplicativos da Web usando a estrutura Spring Web. O nosso trabalho recorre à análise dinâmica, avaliando métricas de execução de código, ao contrário da proposta apresentada focando em padrões arquiteturais de desenvolvimento web.

2.2 Static Analysis of Students' Java Programs [4]

O estudo tem uma abordagem para realizar a verificação de forma automática de atividades de alunos, porém se limita ao tamanho do código. Essa metodologia pode ser aplicada em códigos iniciantes, mas em códigos maiores não há suporte à metodologia abordada. A estrutura é feita por similaridade então o tutor tem um código já desenvolvido e compara com o código do aluno, assim gerando algum tipo de feedback. Diferente do nosso trabalho que faz uso de análise dinâmica, avaliando métricas de execução de código, ao contrário da proposta apresentada que foca em uma análise por similaridade.

2.3 Uma Análise Sobre o Acoplamento em Atividades dos Alunos da Disciplina de Programação OO [3]

Foi aplicada uma ferramenta de análise de código estático e qualidade da arquitetura do software para realizar essas análises, o CodeMR. Ela utiliza como base as métricas Chidamber and Kemerer [1] para gerar resultados detalhados sobre o design do código, qualidade e gráficos para fácil visualização destas métricas. O teste foi realizado em uma base de dados com 100 alunos, chegando a conclusão que o valor do CBO, métrica utilizada para medir o acoplamento das atividades, não possui um impacto significativo nas notas de design. O nosso trabalho recorre à análise dinâmica, avaliando métricas de execução de código, ao contrário da proposta apresentada que foca em uma ferramenta CBO usando as métricas Chidamber and Kemerer [1] que necessita ser comparado as notas para ver sua veracidade.

2.4 Automatic assessment of Java code [2]

O PMD foi utilizado para analisar o código-fonte Java e procurar erros não funcionais e de qualidade de código. É composto por 22 regras dentre essas regras algumas foram aplicadas. Checkstyle é outra ferramenta de desenvolvimento gratuita e de código aberto que ajuda a garantir que o código Java esteja segundo as convenções de codificação estabelecidas. Checkstyle possui muitas regras de codificação prontas permitindo que o usuário crie suas próprias regras também. Por fim, a análise é feita em cima de regras criadas dessas ferramentas de modo a analisar se o código é bem feito e estar em uma convenção adotada. O nosso trabalho recorrerá à análise dinâmica, avaliando métricas de execução de código, ao contrário da proposta apresentada focando em regras de como um código deve ser escrito.

3 ANÁLISE DE TRACE

Para a detecção de problemas de design, abordamos a possibilidade do uso de uma ferramenta de trace.

3.1 Estratégia

Identificar automaticamente métricas sobre o rastro da execução de código. A partir dessas métricas, identificar respostas e submissões divergentes dos alunos para discutir, qualitativamente, sobre as soluções consideradas fora do comportamento padrão das métricas estudadas.

3.2 TraceAgent

É uma ferramenta de captura de dados sobre a execução de código, operando como um agente da JVM que intercepta a execução de código Java.

```
1 [TRACEAGENT] static [START] lab5.SAGA.main(java.lang.String[]) 1645331706375
2 [TRACEAGENT] 2101842856 [START] lab5.FornecedorController.getFornecedores() 1645331706403
3 [TRACEAGENT] 2101842856 [END] lab5.FornecedorController.getFornecedores() 1645331706403
4 [TRACEAGENT] 1514322932 [START] lab5.ClienteController.getClientes() 1645331706414
5 [TRACEAGENT] 1514322932 [END] lab5.ClienteController.getClientes() 1645331706414
6 [TRACEAGENT] 2101842856 [START] lab5.FornecedorController.getFornecedores() 1645331706414
7 [TRACEAGENT] 2101842856 [END] lab5.FornecedorController.getFornecedores() 1645331706414
```

Figura 1: Exemplo de resultado TraceAgent.

Na Figura 1, há um exemplo do registro gerado na execução do código com o TraceAgent e é possível verificar colunas sobre tal execução:

- Coluna 1: refere-se ao [TRACEAGENT] referência ao nome do próprio programa.
- Coluna 2: refere-se a chamada ser estática ou não estática, se não for estática um ID é gerado se for estática o retorno será static.
- Coluna 3: refere-se ao estado da chamada start/end.
- Coluna 4: refere-se a assinatura do método, sem identificar o retorno (nome do método e parâmetros).
- Coluna 5: refere-se a identifição do momento que o trace é registrado (System.currentTimeMillis()).

3.3 Métricas

As métricas são coletadas a partir do TraceAgent por meio dos seus resultados gerados. Para cada métrica, discutimos hipóteses do que representa um valor divergente elevado ou baixo que pode ser encontrado.

3.3.1 Número de classes exercitadas : Essa métrica indica o número de classes desenvolvidas e executadas durante a execução de um código, ignorando classes anônimas. Um grande número de classes pode representar responsabilidades fragmentadas. Um número baixo, pode significar classes com muitas responsabilidades.

3.3.2 Número de métodos estáticos totais exercitadas: Um sistema com muitos métodos estáticos pode apresentar um problema, se tais métodos representarem ações de negócio que poderiam ser implementadas por métodos com extensibilidade polimórfica. Os métodos estáticos também não são herdados, sobrescritos e não permitem extensão.

3.3.3 Número de invocações estáticos totais exercitados: O número de invocações é algo complementar à métrica anterior, na medida que um número de invocações é alto, pode representar potenciais riscos na lógica de negócios.

3.3.4 Número de métodos não estáticos totais exercitados: O número da quantidade de métodos não estáticos pode indicar um sistema flexível, com responsabilidades bem definidas, no entanto, um valor excessivo ou baixo, pode significar métodos com muitas responsabilidades, ou pouco bem definidas, respectivamente.

3.3.5 Número de invocações não estáticos totais exercitados: O número de invocações é algo complementar à métrica anterior, na medida que um número de invocações é alto, para um baixo número de métodos, pode representar um método com mais responsabilidade do que deveria.

3.3.6 Número de objetos invocados: Quantidade de instâncias de classes (objetos) que são invocadas, na medida que o número de invocações é alto, para um baixo número de classes, pode representar classes fortemente acopladas.

3.3.7 Número de troca de contexto (Objetos): Quanto maior é a troca de contexto existe um potencial da responsabilidade está espalhada em muitos objetos diferentes, ou compartilhada de forma que fere o encapsulamento.

4 METODOLOGIA

Esse trabalho usou o TraceAgent para gerar dados possíveis de análise exploratória e descritiva sobre os laboratórios de programação 2 do curso de Ciência da Computação da UFCG.

4.1 Base de dados

Foi coletada dos laboratórios de 4 períodos, incluindo de 2018.1 a 2019.2. Nos dois primeiros semestres avaliados, o laboratório consistia de um sistema de gerenciamento de apostas, já nos semestres de 2019.1 a 2019.2 esse sistema se torna um de compras, entre os semestres ocorreram algumas alterações na descrição dos laboratórios para os alunos, mas não são significativas em termo de design de código da solução.

Ambos os problemas apresentam domínios diferentes de solução, mas estruturas semelhantes, onde, idealmente, o aluno deve fazer uso de 2 ou 3 controllers, bem como uso dos conceitos

de interface e herança para reuso de código. Todos os laboratórios apresentam testes automáticos semelhantes nas atividades que exercitam de forma que os registros de execução possam ser comparáveis.

Em termos práticos, a avaliação manual realizada pela equipe de ensino, apresentava os mesmos critérios de correção, sendo estes:

- Testes unitarios automaticos.
- Funcionalidades Básicas.
- Design de Classes.
  - Responsabilidade Única.
  - Encapsulamento.
  - Ocultação da Informação.
- Design entre classes.
  - Expert.
  - Creators.
  - Controller.
  - Coesão.
  - Acoplamento.
  - Polimorfismo.
- Uso de Coleções.

Para o nosso estudo alguns critérios foram adotados para escolher a base de dados de teste:

- Ter passado em todos os testes automáticos.
- Ter uma estrutura de paginas que siga o padrão "nome da pasta/laboratorio de programação"que foi estabelecido para rodar o script de captura de métricas.

4.2 Estudo de caso

Para obter os resultados foi necessário seguir os seguintes passos:

- Coleta dos dados: Os laboratórios foram coletados e selecionados apenas os que passaram nos requisitos.
- Executar TraceAgent.
- Executar script de análise: Que foi criado em python para realizar o processamento dos dados gerados pelo trace.
- Fazer uma análise exploratória dos dados gerados pelo script de análise de dado.

5 RESULTADOS

5.1 Número de classes exercitadas

Ao avaliar o número de classes, destacou-se a mediana de 14 classes por laboratório. Como é possível observar na Tabela 1, temos um laboratório com 22 classes e outro laboratório com 8 classes. Os valores identificados nos laboratórios não estão tão distantes da mediana, fazendo parte ainda do intervalo de 1,5 vezes a distância inter-quartil (IQR), mas, destacamos esses dois laboratórios para análise de possíveis problemas de qualidade nestes códigos.

Os demais laboratórios se concentraram próximo da mediana, como mostra a Figura 2.

Tabela 1: Análise descritiva do número de classes exercitadas

Minimo	8
1 Quartil	12
2 Quartil	14
3 Quartil	17
Máximo	22
IQR	5
Limite inferior para outliers	4,5
Limite superior para outliers	24,5

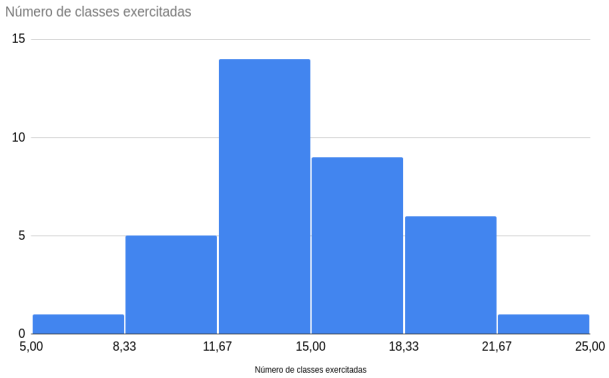

Figura 2: Histograma do número de classes.

O laboratório com 22 classes foi analisado e identificado a criação de várias classes de exceções, simbolizando um laboratório que especializou excessivamente suas classes.

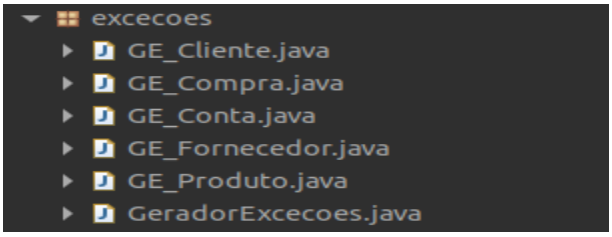

Figura 3: Classes para exceções.

O laboratório com 8 classes foi analisado e identificado que a classe ControllerCliente está exercendo dois papéis, o controlador está respondendo tanto pelo cliente quanto operações de compra.

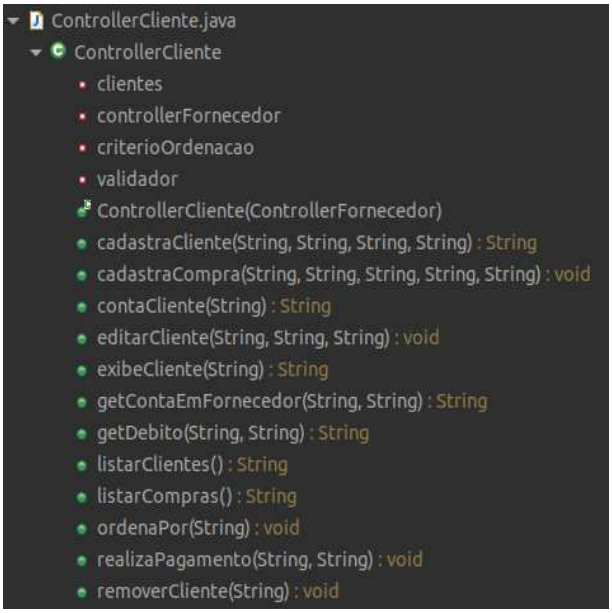

Figura 4: Classe ControllerCliente.

5.2 Número de métodos estáticos totais exercitadas

Antes da avaliação é válido salientar que todos os laboratórios têm no mínimo 1 método estático usado para executar os testes automáticos. Ao avaliar o número de métodos estáticos, destacou-se a mediana de 1 método estático por laboratório. Como é possível observar na Tabela 2, temos um laboratório com 27 métodos estáticos.

Tabela 2: Análise descritiva do número de métodos estáticos totais exercitadas

Minimo	8
1 Quartil	1
2 Quartil	1
3 Quartil	2,25
Máximo	27

Os demais laboratórios se concentraram dentro e próximo da mediana, como mostrado na Figura 5.

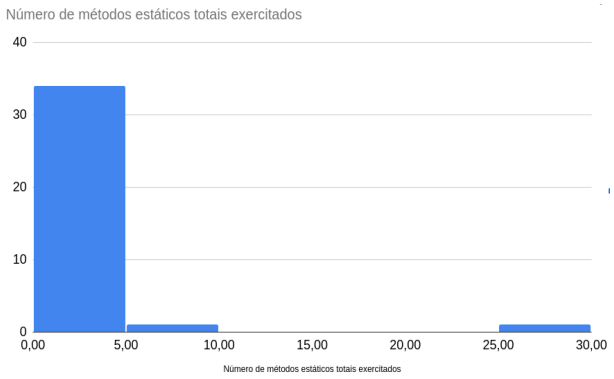

Figura 5: Número de métodos estáticos totais exercitadas

O laboratório com 27 métodos estáticos, após a análise foi detectado que os métodos estáticos estão divididos em duas classes a primeira é a classe Verificador com métodos com objetivos de verificar se valores são iguais a nulos ou vazios, já a outra classe que se chama Facade tem todos os seus métodos estáticos e esses métodos que delegam operação para os controleers do sistema. Códigos com métodos estáticos geram um aumento na complexidade do código por não ser extensível e não precisar de contexto para existir, por consequência uma manutenção em um método estático pode afetar todo o laboratório.

5.3 Número de invocações estáticos totais exercitados

Ao avaliar o número de invocações de métodos estáticos, destacou-se a mediana de 1 para invocações de métodos estáticos por laboratório. Como é possível observar a tabela a seguir, temos um laboratório com 2140 invocações de métodos estáticos e e vale salientar que esse laboratório não é o mesmo avaliado na regra anterior com 27 métodos estáticos totais exercitadas. O valor de 2140 está distante da mediana, não fazendo parte do intervalo de 1,5*IQR.

Tabela 3: Análise descritiva do número de invocações estáticos totais exercitados

Minimo	1
1 Quartil	1
2 Quartil	1
3 Quartil	254,5
Máximo	2140
IQR	253,5
Limite superior para outliers	634,75

Os demais laboratórios se concentraram dentro da mediana, como mostrado na Figura 6.

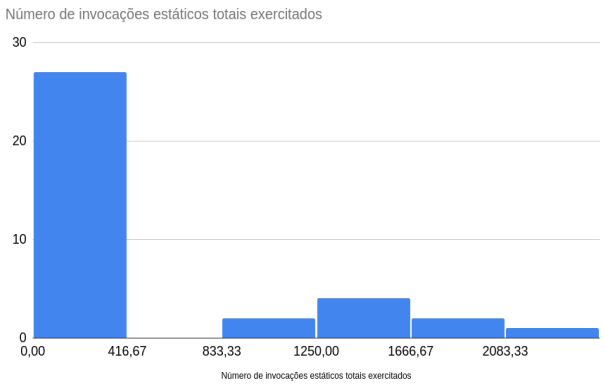

Figura 6: Número de invocações estáticos totais exercitados.

O laboratório com 2140 invocações de métodos estáticos, após a análise foi detectado que os métodos estáticos estão na classe Utilitários que é usada para verificar se valores são iguais a nulos ou vazios. Percebemos o motivo de ter tantas invocações porém o uso desses métodos estáticos é prejudicial gerando um código com grande complexidade por não ser extensível.

5.4 Número de métodos não estáticos totais exercitados

Ao avaliar o número de métodos não estáticos exercitados, destacou-se a mediana de 123 métodos não estáticos exercitados por laboratório. Como é possível observar na tabela a seguir, temos um laboratório com 168 e outro laboratório com 83 métodos não estáticos exercitados. Esses valores não estão tão distantes da mediana, fazendo parte ainda do intervalo de 1,5*IQR, mas, destacamos esses dois laboratórios para análise.

Tabela 4: Análise descritiva do número de métodos não estáticos totais exercitados

Minimo	83
1 Quartil	105
2 Quartil	123
3 Quartil	133,25
Máximo	168
IQR	28,625
Limite inferior para outliers	62,625
Limite superior para outliers	175,625

Os demais laboratórios se concentraram próximo da mediana, como mostrado na Figura 7.

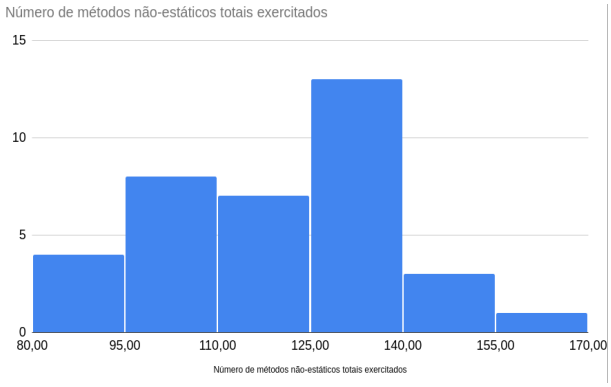

Figura 7 : Número de métodos não estáticos totais exercitados.

O laboratório com 83 métodos não estáticos totais exercitados, após a análise foi possível detectar métodos com mais de um papel, um exemplo seria na classe combo o método toString que está representando a entidade e também fazendo uma conversão dos dados para o formato correto. E também métodos faltosos o equals e o hashcode de Fornecedor, Compra e Conta. Que por meio dos testes automatizados não foram detectadas essas faltas mas com uma análise mais direcionada foi possível detectar.

O laboratório com 168 métodos não estáticos totais exercitados, foi o mesmo laboratório de 22 classes da regra Número de classes exercitadas e a conclusão se torna a mesma para os métodos existentes na aplicação, simbolizando um projeto que especializou excessivamente suas classes e também seus métodos.

5.5 Número de invocações não estáticos totais exercitados

Ao avaliar o número de métodos não estáticos exercitados, destacou-se a mediana de 3017,5 invocações de métodos não estáticos exercitados por laboratório. Como é possível observar na tabela a seguir, temos um laboratório com 7862 e outro laboratório com 1329 invocações de métodos não estáticos exercitados. O valor 1329 de não está tão distante da mediana, fazendo parte ainda do intervalo de IQR, mas 7862 está distante da mediante e não faz parte do intervalo de 1,5*IQR.

Tabela 5: Análise descritiva do número de invocações não estáticos totais exercitados.

Minimo	1329
1 Quartil	2573,5
2 Quartil	3017,5
3 Quartil	4012,5
Máximo	7862
IQR	1439
Limite inferior para outliers	415
Limite superior para outliers	6171

Os demais laboratórios se concentraram próximo da mediana, como mostrado na Figura 8, ocorrendo apenas um laboratório com 1329 invocações não estáticos totais exercitados e um laboratório com 7862 invocações não estáticos totais exercitados.

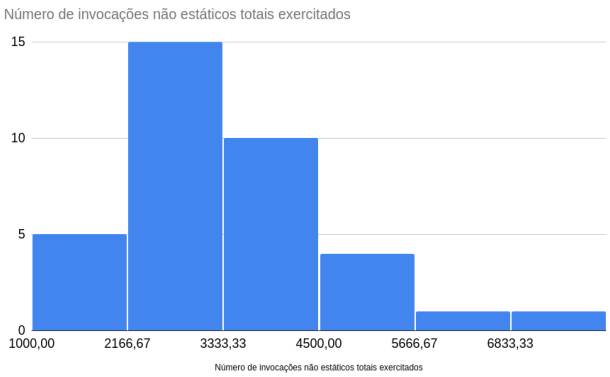

Figura 8 Número de invocações não estáticos totais exercitados.

O laboratório com 1329 invocações não estáticos totais exercitados, após a análise foi detectado métodos extensos que não exercem apenas uma função.

O laboratório com 7862 invocações não estáticos totais exercitados, após a análise foi detectado métodos que são excessivamente especializados.

5.6 Número de objetos invocados

Ao avaliar o número de objetos invocados, destacou-se a mediana de 115 número de objetos invocados por laboratório. Como é possível observar na tabela a seguir, temos um laboratório com 371 número de objetos invocados. Esse valor não está tão distante da mediana, fazendo parte ainda do intervalo de 1,5 * IQR, mas, destacamos esse laboratório para análise.

Tabela 6: Análise descritiva do número de objetos invocados

Minimo	44
1 Quartil	85,75
2 Quartil	115
3 Quartil	230,5
Máximo	371
IQR	144,75
Limite inferior para outliers	-131,375
Limite superior para outliers	447,625

Os demais laboratórios se concentraram próximo da mediana, como mostrado na Figura 9.

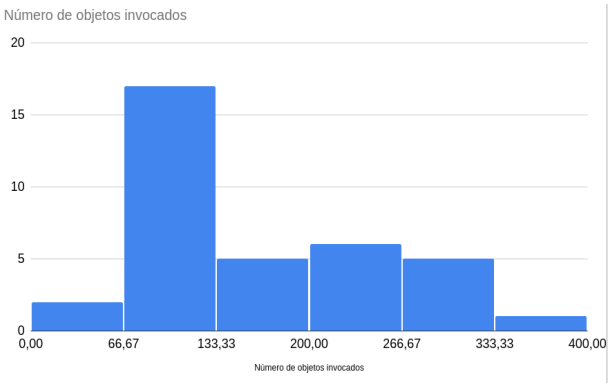

Figura 9: Número de objetos invocados.

O laboratório com o valor de 371 número de objetos invocados, após a análise foi possível detectar que não existe nenhum ponto negativo a ser levado apontado, o projeto foi bem desenvolvido.

5.7 Número de troca de contexto (Objetos)

Ao avaliar o número de troca de contexto, destacou-se a mediana de 2358 número de troca de contexto por laboratório. Como é possível observar na tabela a seguir, temos um laboratório com 5790 números de troca de contexto. Esse valor está distante da mediana, não fazendo parte do intervalo de 1,5 * IQR.

Tabela 7: Análise descritiva do número de troca de contexto (Objetos)

Minimo	957
1 Quartil	1908,5
2 Quartil	2358
3 Quartil	2974
Máximo	5790
IQR	1065,5
Limite inferior para outliers	310,25
Limite superior para outliers	4572,25

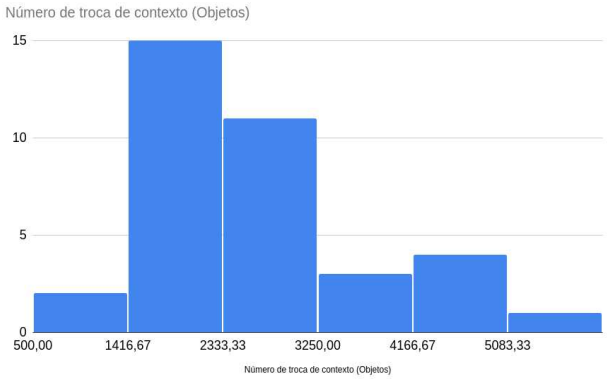

Figura 10: Número de troca de contexto (Objetos).

O laboratório com o valor de 5790, é o mesmo laboratório que foi avaliado com 7862 número de invocações não estáticos totais exercitados, com isso já levamos em consideração que o

laboratório tem métodos que são excessivamente especializados por consequência dessa excessiva especialização o número de troca de número de troca de contexto saiu elevado.

6 DISCUSSÃO

Avaliando os laboratórios destoantes, destacamos as métricas número de classes exercitadas, invocações estáticos totais exercitados, invocações não estáticos totais exercitados. Onde os outliers parecem apresentar erros de estrutura como uso de métodos estático, classes ou métodos tendo mais de um papel, classes ou métodos excessivamente especializados.

É importante destacar que os erros estruturais que encontramos podem existir nos demais laboratórios, mas que aqueles com métricas destoantes parecem ter um potencial para a presença dos problemas aqui citados.

Além disso, a execução do TraceAgent é bem simples e rápida, podendo executar todos os laboratórios de um período em segundos.

7 CONCLUSÕES

O desenvolvimento do presente estudo possibilitou a análise de como a ferramenta TraceAgent se comporta a ser executada em uma base de dados real, após a análise é possível observar que a ferramenta funciona na detecção das métricas propostas assim conseguindo detectar problemas no design de código.

Por fim, o TraceAgent tem a capacidade de melhorar as correções dos laboratórios, junto com o guia de correção as avaliações seriam melhor executada e o aprendizado seja mais proveitoso.

REFERÊNCIAS

[1] CodeMR. [n. d.]. CodeMR. Retrieved 1 de Março de 2022 from https://www.codemr.co.uk/documents

[2] AMANDA VIVIAN ALVES DE LUNA E COSTA. 2013. Automatic assessment of Java code. Retrieved 1 de Março de 2022 from http://saruna.mnu.edu.mv/jspui/bitstream/123456789/214/1/Automatic-assessment-mnjrv1n1-1.pdf

[3] AMANDA VIVIAN ALVES DE LUNA E COSTA. 2021. Uma Análise Sobre o Acoplamento em Atividades dos Alunos da Disciplina de Programação OO. Retrieved 1 de Março de 2022 from http://dspace.sti.ufcg.edu.br:8080/xmlui/bitstream/handle/riufcg/19812/AMANDA%20VIVAN%20ALVES%20DE%20LUNA%20E%20COSTA%20-%20TCC%20CI%c3%8aNCIA%20DA%20COMPUTA%c3%87%c3%83O%202021.pdf?sequence=1&isAllowed=y

[4] Peter Bancrof Nghi Truong, Paul Roe. 2004. Static Analysis of Students' Java Programs. Retrieved 1 de Março de 2022 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.503&rep=rep1&type=pdf

[5] Abdelaziz SAHABa and Sylvie TRUDEL. 2020. COSMIC Functional Size Automation of Java Web Applications Using the Spring MVC Framework. Retrieved 1 de Março de 2022 from http://ceur-ws.org/Vol-2725/paper7.pdf