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

AMANDA VIVIAN ALVES DE LUNA E COSTA

UMA ANÁLISE SOBRE O ACOPLAMENTO EM ATIVIDADES DOS ALUNOS DA DISCIPLINA DE PROGRAMAÇÃO OO

CAMPINA GRANDE - PB

2021

AMANDA VIVIAN ALVES DE LUNA E COSTA

UMA ANÁLISE SOBRE O ACOPLAMENTO EM ATIVIDADES DOS ALUNOS DA DISCIPLINA 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 Gaudêncio do Rego.

CAMPINA GRANDE - PB

2021

C837a Costa, Amanda Vivian Alves de Luna.
Uma análise sobre o acoplamento em atividades dos alunos da Disciplina de Programação OO. / Amanda Vivian Alves de Luna Costa. – 2021.

12 f.

Orientador: Prof. Dr. Matheus Gaudencio do Rêgo.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Qualidade de código - software. 2. Paradigma orientado a objetos. 3. Programação OO. 4. Acoplamento - software. 5. Códigos. 6. Coupling Between Objects – CBO. 7. CodeMR. 8. Análise de código estático – ferramenta. 9. Design de código. 10. Chidamber and Kemerer – métrica. 11. Métrica CBO. 12. Código. 13. Ferramenta de análise de código. 14. Disciplina Programação OO – UFCG. I. Rêgo, Matheus Gaudencio do. II. Título.

CDU:004.415.3(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

AMANDA VIVIAN ALVES DE LUNA E COSTA

UMA ANÁLISE SOBRE O ACOPLAMENTO EM ATIVIDADES DOS ALUNOS DA DISCIPLINA 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 Gaudêncio do Rego

Orientador – UASC/CEEI/UFCG

Professora Dra. Melina Mongiovi Cunha Lima Sabino

Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 25 de Maio de 2021.

CAMPINA GRANDE - PB

ABSTRACT

Good quality code can be considered an essential property of the software. If the quality of the code is not good enough, there may be a financial loss or time loss due to maintenance, modifications or adjustments. In this study, we will use CodeMR, a plugin that performs code analysis to extract more information about the application, with a focus on design and coupling. With the CBO (Coupling Between Objects) metric, used to measure coupling, we evaluated the impact of grouping on the note and found that the CBO value does not influence the design notes and has a moderate influence on the overall note.

Uma Análise Sobre o Acoplamento em Atividades dos Alunos da Disciplina de Programação OO

Amanda Luna
amanda.costa@ccc.ufcg.edu.br
Federal University of Campina Grande
Campina Grande, Paraíba

Matheus Gaudencio
matheusgr@computacao.ufcg.edu.br
Federal University of Campina Grande
Campina Grande, Paraíba

RESUMO

Um código de boa qualidade pode ser considerado uma propriedade essencial do software. Se a qualidade do código não for boa o suficiente, pode haver perda financeira ou perda de tempo devido a manutenção, modificações ou ajustes. Neste estudo, usaremos o *CodeMR* Um plugin que realiza análise de código para extrair mais informações sobre a aplicação, com foco em design e acoplamento. Com a métrica CBO(Coupling Between Objects), utilizada para medir acoplamento, avaliamos o impacto do agrupamento na nota e encontramos que o valor de CBO não influencia nas notas de design e possui uma influência moderada na nota geral.

Palavras chave

Qualidade de código, CodeMR, Acoplamento, Design.

1. INTRODUÇÃO

Um código de boa qualidade pode ser considerado uma propriedade essencial do software. Se a qualidade do código não for boa o suficiente, isso pode levar a perdas financeiras ou perda de tempo devido a manutenção, modificações ou ajustes [1].

No cenário atual, o paradigma orientado a objetos (OOP) é o paradigma mais popular devido aos seus recursos como capacidade de reutilização, manutenção, etc. Grandes aplicativos de software são compostos por centenas de classes que, por sua vez, encapsulam um grande número de métodos. Portanto, o aplicativo é geralmente organizado em módulos usando o conceito de pacotes (*packages*). A característica desejada do OOP é alta coesão e baixo acoplamento. Isso permite alta compreensão e baixa sobrecarga de manutenção do aplicativo. Isso ocorre porque o alto grau de compreensibilidade reduz o tempo gasto para compreender o software e, portanto, sua testabilidade e manutenção [12].

Acoplamento pode ser entendido como a interdependência existente entre diferentes classes. Quanto mais uma classe conhece ou depende de outras classes, maior é o grau de acoplamento entre elas. Em sistemas com forte acoplamento, mudanças em uma classe forçam mudanças em classes relacionadas, fica mais difícil entender as classes isoladamente e é mais difícil reutilizar as classes, já que elas dependem da presença umas das outras [13].

Na disciplina de Programação 2, a nota das atividades é dividida em duas partes: a nota dos testes unitários automáticos, que avalia se as funcionalidades requisitadas naquele laboratório foram implementadas corretamente e a nota de design. O design é avaliado de maneira manual, com base em uma planilha disponibilizada pelos professores que possui orientações sobre como realizar a correção. Nela são descritos os critérios de avaliação, um detalhamento sobre as infrações e o valor que deve ser descontado da nota de acordo por cada uma delas.

Critérios	
Funcionalidade	-1 por infração
	7 O programa funciona para o fluxo normal de execução
	3 O programa funciona para fluxos alternativos de execução (exceções, alun
Design de classes	-1 por classe infratora
	4 Responsabilidade Única
	4 Encapsulamento
	2 Ocultação da Informação
Design entre classes	-0.5 por infração
	1 Expert
	1 Creators
	2 Controller
	1 Coesão
	1 Acoplamento
	2 Comparators
	2 Herança/Polimorfismo
Uso de Coleções	0 ou 2 por critério
	4 Uso adequado para armazenamento de Produto, Cliente, Fornecedor
	4 Uso adequado para armazenamento de Conta e Compras (itens que compões a Conta)
	2 hashcode/equals das entidades básicas
Critérios Não-funcionais	-0.5 por classe com várias infrações
	5 Legibilidade/Formatação/Constantes
	5 Documentação
Testes de unidade	-0.5 por cenário de teste ausente
	8 Testes das funcionalidades básicas

Figura 01:Planilha com as orientações de como proceder com a avaliação de design referente ao Laboratório 5.

Testes automáticos, neste contexto unitários, são necessários, pois eles dão um feedback imediato sobre as funcionalidades e promovem uma certa garantia de que o código realmente está funcionando conforme o esperado. Apesar disto, estes testes possuem limitações dado o escopo da disciplina. Por exemplo, um sistema feito em uma única classe, poderia passar em todos os testes automáticos mas apresentar sérios problemas de acoplamento. Isto é realidade também para um sistema com classes em excesso (e extremamente acopladas). Entretanto, o teste não observa nada a respeito da sua execução e, a avaliação do baixo ou alto coesão/acoplamento só acontece através de uma correção manual e demorada.

Dado isto, este estudo propõe tentar extrair outras informações, especificamente de acoplamento, utilizando um plugin chamado CodeMR, que faz análise de design de código utilizando as métricas de *Chidamber and Kemerer* [7].

Com estes dados, poderemos, através de conceitos de padrões de projeto e estudos referenciais, ter mais noção de coesão e coerência dos códigos avaliados, dado as análises providas pela ferramenta e assim, ter um panorama maior de possibilidades de avaliação de design e verificar se há relação entre as métricas e os pontos adotados em correções manuais.

2. FUNDAMENTAÇÃO TEÓRICA

Chidamber e Kemerer (CK) introduziram um conjunto de métricas para medir a testabilidade, manutenção e reutilização de uma classe, mas sem qualquer validação empírica. CK define Coupling Between Objects (CBO) para uma classe como a contagem do número de outras classes para as quais está diretamente acoplado. Este número representa o *fan-out* de um objeto para objetos externos. A base da métrica está no fato de que se um objeto for acoplado a outro, ele usa métodos ou variáveis de instância de outro [2]. Eles concluíram que "*...qualquer evidência de um método de um objeto usando métodos ou variáveis de instância de outro objeto constituem acoplamento.* " [3].

Hitz e Montazeri [4] nos definem que o acoplamento é como um atributo de pares de objetos, mas como uma métrica, é agregado ao número total de acoplamento que uma classe tem com outras classes, portanto, implicitamente assumindo que todos os acoplamentos básicos têm a mesma força.

Em geral, vemos acoplamento como algo ruim, pois ele pode trazer algumas das consequências abaixo [11]:

- Os desenvolvedores / programadores de manutenção precisam entender potencialmente todo o sistema para poder modificar com segurança um único componente.
- Mudar os requisitos que afetam a adequação de algum componente potencialmente exigirá uma ampla gama de mudanças para acomodar um componente de substituição mais adequado.
- É necessário pensar mais nas opções no início da vida útil de um sistema de software para tentar prever os requisitos de longo prazo do sistema porque as mudanças são mais caras.

Contudo, algumas vezes acoplamento se torna um "mal necessário", pois às vezes, um sistema é simplesmente tão grande e complexo que, mesmo que a maioria de seus componentes seja altamente coesa, você precisa dividi-los em partes e, possivelmente, precisa ser capaz de conectar outro código em algumas dessas partes, para tornar o sistema sustentável. Nesses casos, pode não haver escolha. Você pode precisar dimensionar um sistema além dos limites do servidor e dividi-lo em componentes específicos do servidor. Cada etapa de processamento pode precisar de acesso e conhecimento do estado completo para poder continuar o processamento, não importa como você tente dividir e fatiar as tarefas. Outro caso em que o aumento do acoplamento pode compensar o custo é a facilidade de uso [11].

3. METODOLOGIA

Esse trabalho fez uso de análises exploratórias e descritivas para criar informações e interpretações sobre métricas de design dos Laboratórios de Programação 2 do curso de Ciência da Computação da UFCG. Com o objetivo de verificar se há correlação entre a nota de design das atividades e a quantidade de acoplamento existente neles, bem como avaliar qualitativamente situações onde a métrica aponta uma baixa ou alta taxa de acoplamento.

3.1 Composição das amostras

As análises foram feitas com base nas submissões da 5ª atividade de laboratório da disciplina, no total tivemos 109 submissões de alunos, sendo 48 referentes às submissões do primeiro semestre de 2019 e 61 referentes às submissões do segundo semestre de 2019.

A atividade consiste em simular, de maneira simples, um sistema de gerenciamento de compras na qual, a proposta é implementar um sistema de comércio eletrônico para dar suporte à comercialização de lanches nos laboratórios. As necessidades de fornecedores e clientes motivou o surgimento do *SAGA - Sistema para Auto-Gestão de comércio de Alimentos*.

A avaliação de design deste laboratório baseia-se nos seguintes critérios:

- Padrões GRASP: ênfase no uso de expert da informação, creator, baixo acoplamento, alta coesão e controladores
- Reuso por meio de herança e composição
- Uso de interfaces: Comparable e Comparator
- Polimorfismo com herança e interfaces
- Uso de interface com composição como uma forma de ter menor acoplamento e polimorfismo

A nota de design se baseia no cumprimento destes critérios, mas neste estudo tivemos como foco a avaliação do acoplamento das submissões.

3.2 CodeMR

CodeMR é uma ferramenta de análise de código estático e qualidade da arquitetura do software [5], para realizar essas análises, ela utiliza como base as métricas *Chidamber and Kemerer* para gerar resultados detalhados sobre o design do código, qualidade e gráficos para fácil visualização destas métricas, como pode ser visto pela Figura 1.

Nela podemos ver a tela de dados gerais da métrica do projeto, as cores dos gráficos variam de acordo com as classificações destas métricas. A cor verde indica valores adequados para as métricas avaliadas.

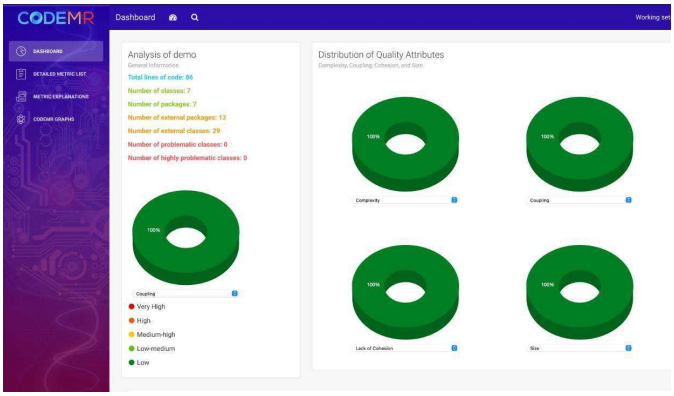

Figura 02: Exemplo de resultado decorrente da execução da ferramenta CodeMR.

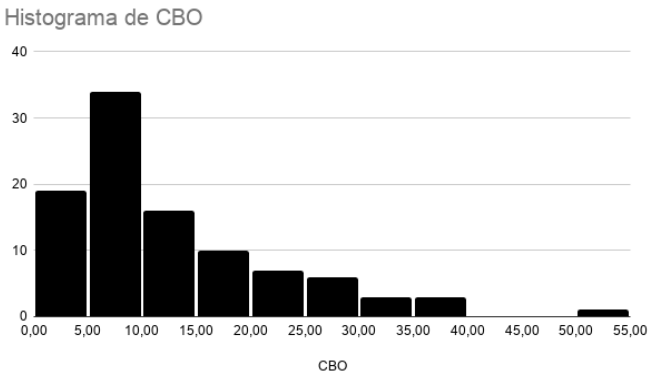

Figura 03: Histograma dos valores de CBO das atividades referentes ao laboratório 05.

3.3 Métrica CBO

Para realizar as análises, foi utilizada a métrica CBO, que avalia o acoplamento entre os objetos [6]. A ferramenta nos diz que, existe acoplamento entre duas classes X e Y se [7]:

- X tem um atributo que se refere a (ou é do tipo) Y;
- X tem um método que faz referência a Y (por meio do tipo de retorno ou parâmetro);
- X tem uma variável local cujo tipo é classe Y;
- X é uma subclasse da (ou implementa) a classe Y.

O cálculo do CBO de uma classe é feito contando outras classes cujos atributos ou métodos são usados por uma classe, mais aqueles que usam os atributos ou métodos da classe dada., em que as relações de herança são excluídas.

O CodeMR também nos provém parâmetros que podemos usar de base para classificar o nível de acoplamento de uma classe ou aplicação. Estes variam de *baixo* até *muito alto* e são dispostos da seguinte forma:

- Baixo: CBO <= 5
- Médio-baixo: CBO > 5 e <= 10
- Médio-alto: CBO > 10 e <= 20
- Alto: CBO > 20 e <= 30
- Muito alto: CBO > 30

4. RESULTADOS

4.1 CBO

Temos 100 submissões, nas quais os valores de acoplamento variam entre 0 e 50, sendo o primeiro o maior e o último o menor. Inicialmente buscamos conhecer as tendências centrais através da média, mediana e desvio padrão. Os valores foram 11,7, 9,0 e 9,7 respectivamente. Dado isto, podemos concluir, pelo alto valor do desvio padrão, que as observações não seguem uma distribuição normal, como podemos evidenciar no histograma da Figura 3.

De acordo com a Figura 03, podemos observar que a maioria dos laboratórios possuem CBO com valores entre 5 e 9,9, que são abaixo da média e, segundo as métricas do CodeMR, se encaixam na classificação acoplamento *Médio-baixo*.

Também segundo o gráfico, vemos que a maior parte dos dados estão abaixo da média, no caso, das 100 observações totais, **53** possuem CBO entre 0 e 10 e **47** entre 10 e 50, porém, não há registros entre 40 e 50, gerando um vazio neste intervalo.

Por fim, de acordo com a imagem, percebemos um comportamento de cauda, indicando que os dados não seguem uma distribuição normal.

4.2 Nota do Design de Classes

Esta métrica avalia se as classes do laboratório estão seguindo as boas práticas [8], tais como manter atributos de classes privados, nomenclatura de classes e métodos, atentar para a lei de Demeter [9], usar interfaces para se referir a collections, encapsulamento, entre outras.

Colhemos as métricas de média, mediana e desvio padrão, sendo estas 9,10 3 e 1,88, com isto, podemos inferir pelo valor do desvio padrão, que a média é significativa e que as amostras estão bem agrupadas.

Ainda de acordo com as métricas, podemos dizer que pelo menos 50% dos alunos tiraram nota máxima nesse quesito.

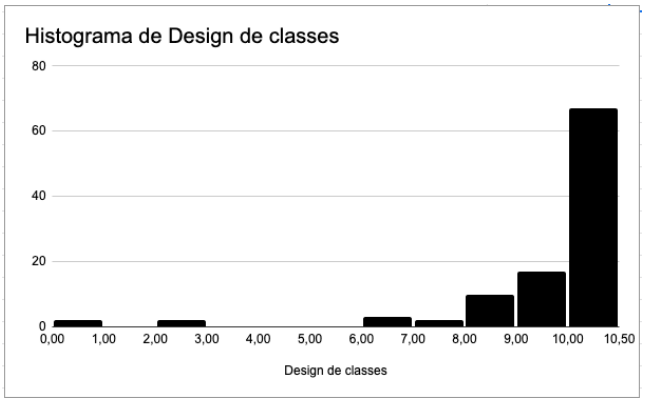

Figura 04: Histograma dos valores das notas de Design de Classe das atividades referentes ao laboratório 05.

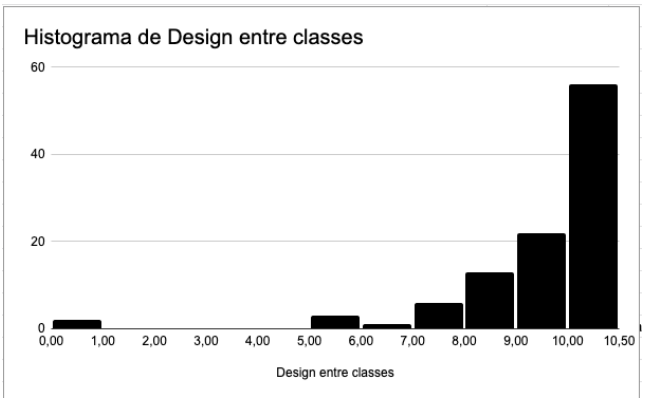

Figura 05: Histograma dos valores das notas de Design entre Classes das atividades referentes ao laboratório 05.

Na Figura 04, podemos perceber que os dados estão majoritariamente agrupados nos últimos setores, em que uma parte significativa dos alunos tiraram notas entre *9* e *9,9*, inclusive, e a maior parte dos alunos tirou 10. Com isso, podemos chegar a conclusão que grande parte dos alunos tiraram notas próximas do valor máximo, podendo ser um indicativo de bom desempenho.

4.3 Nota de Design entre as Classes

Aqui avaliamos, por exemplo, o princípio da responsabilidade única, que consiste basicamente em que uma classe deve fazer apenas uma coisa, deve fazê-la bem e deve fazer somente ela. Se uma classe tem mais de um motivo para ser alterada, ela não segue este princípio [10].

Além disso, as atividades devem seguir o padrão *MVC (Model View Controller)*, em que, *Repository* (ou classe equivalente) deve conhecer apenas o *model* a que faz referência. O *service/controller* deve referenciar apenas o *repository* da sua classe base e, caso precise de operações das outras classes, deve chamar os métodos do *service/controller* daquela classe e estes deve conhecem apenas o *service/controller* referente ao seu modelo. Dentre outros princípios e métricas. Com isso, vemos que esta é uma das principais métricas para avaliação de acoplamento nas atividades.

Colhemos a média, mediana e desvio padrão, com valores de 9, 10 e 1,74 respectivamente. Com isto, podemos afirmar que pelo menos metade dos alunos tirou nota máxima e que novamente não estamos tratando com um dado de natureza normal.

Na Figura 5 podemos observar que o histograma das notas de design entre classes possui comportamento semelhante ao apresentado na Figura 4, com uma grande concentração no final do gráfico, podendo também ser um indicativo de bom desempenho por parte dos alunos.

Além disso, percebemos que, mesmo com notas altas, em comparação com a nota de Design de Classes, tivemos notas menores, dando margem à possibilidade de que os alunos possuem ainda uma pequena dificuldade em quesito de acoplamento em comparação a estruturação de classes discutidas no critério anterior.

4.4 Nota Geral

Esta nota consiste em uma média ponderada entre todas as notas que compõem o lab, sendo estas:

- Funcionalidade (com peso 0,2)
- Design de classe (com peso 0,25)
- Design Entre Classes (com peso 0,.25)
- Uso de coleções (com peso 0,1)
- Critérios não Funcionais (com peso 0,1)
- Testes (com peso 0,1)

A soma de todos os pesos deve ser **1** e esta nota é decorrente do conjunto das avaliações dos monitores e professores.

Também foram geradas a média, mediana e desvio padrão, com valores 6,9, 7,9 e 3,0 respectivamente. Com isso, podemos observar que há uma grande variância dos valores, devido ao alto valor de desvio padrão..

Com o valor da mediana, podemos afirmar que pelo menos 50% dos alunos tiraram notas acima da média (7,0), e não só isso, mas pelo menos metade atingiu quase 80% da pontuação máxima.

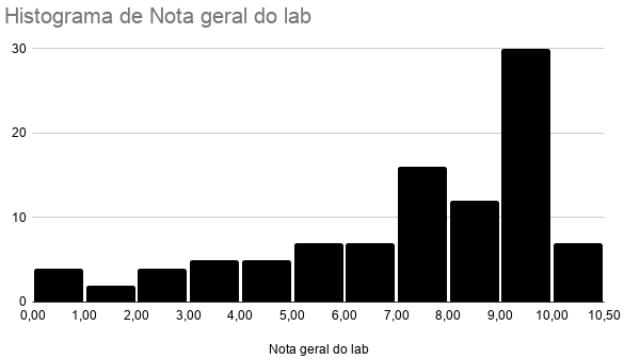

Figura 06: Histograma dos valores das notas gerais das atividades referentes ao laboratório 05.

De acordo com a Figura 06, do histograma das notas gerais, podemos ver que não temos uma distribuição normal, pois não há nenhum agrupamento bem definido.

Embora, a maior parte dos alunos tenha tirado entre 9 e 9,9, muitos discentes tiraram notas das mais variadas, inclusive uma parcela significativa conseguiu uma média entre 0 e 0,9, sendo este conjunto de amostras maior do que a quantidade de pessoas que tiraram a nota máxima e também do que a quantidade que tirou notas entre 1 e 6,9

Este comportamento pode ser explicado majoritariamente por alunos que não conseguiram completar todos os casos de uso da atividade e/ou implementaram e não conseguiram passar nos testes unitários feitos automaticamente por uma ferramenta interna da disciplina de Laboratório de Programação 2.

4.5 Análise das submissões com mais acoplamento

Dentre as atividades analisadas, temos seis com valores de CBO muito alto, ou seja, o valor é maior do que 30. Tomando esta amostra como base, destacamos trechos de códigos que aumentam de maneira considerável o acoplamento.

No Código 1, temos um trecho da classe *ContaController* que tem a finalidade de adicionar uma conta, que no contexto da atividade é uma simulação de caderneta de anotações de vendas, para um determinado cliente referenciando um fornecedor. Podemos observar os seguintes problemas:

- Só nesse método são instanciados todos os modelos do sistema diretamente, sem passar por outro controller (sublinhados).
- Realiza operações de adicionar dados em uma conta, que seria atribuição do repository

```
      public void adicionaConta(Cliente
cliente, Fornecedor fornecedor) {
      if (cliente == null)
throw new IllegalArgumentException("Erro
ao cadastrar conta: cliente nao pode vazio
ou nulo.");
         if (fornecedor == null)
throw new IllegalArgumentException("Erro
ao cadastrar conta: fornecedor nao pode
vazio ou nulo.");
         if (
this.encontraConta(cliente.getCPF(),
fornecedor.getNome()) )
       throw new
IllegalArgumentException("Erro ao
cadastrar conta: conta ja existe.");

ContaID id = new ContaID(cliente.getCPF(),
fornecedor.getNome());
Conta conta = new Conta(id, cliente,
fornecedor);
this.contas.put(id, conta);
}
```

Código 1: Código com instanciação de todos os modelos elevando o CBO para o valor 50.

Embora o valor de CBO seja muito alto, esta atividade obteve notas 10,9 e 9 para design de classe, design entre classe e nota geral respectivamente. O que nos leva a questionar se o acoplamento é realmente levado em conta na hora de avaliar as métricas de design.

No Código 2, temos um exemplo de um trecho do controller de conta, em que este método exibirá todas as contas existentes. O primeiro problema é que, na assinatura do método, este possui acesso ao objeto *fornecedor (*sublinhado) e, além disso, realiza operações no *repository* (sublinhado) de compra diretamente, enquanto a maneira correta seria o *service* de compra realizar esta operação e ser chamado no *controller* analisado.

```
String exibeContas(String cpf, Fornecedor
fornecedor) {
  String msg = "Erro ao exibir conta do
cliente";
  Fornecedor fornecedor =
this.fornecedorService.get(fornecedor.getN
ome(), msg);
  Cliente cliente =
this.checaClienteConta(fornecedor, cpf,
msg, msg + ": cliente nao tem nenhuma
conta com o fornecedor.",
        msg + ": cliente nao tem nenhuma
conta com o fornecedor.");
```

```
  List<String> results =
this.compraRepository.getCompras(fornecedo
r,
cliente).stream().map(x -> x.toString())
        .collect(Collectors.toList());


  return "Cliente: " + cliente.getNome() +
" | " + fornecedor.getNome() + " | " +
String.join(" | ", results);
}
```

Código 2: Código com instanciação de modelos fora de seus respectivos controllers e chamada de repository também quebrando padrão arquitetural, elevando o valor do CBO para 36.

Nesta atividade, as notas foram 10 para design de classe e 8 para design entre classes. Podemos ver que o design entre classes foi um pouco penalizado nesta correção, porém, não avaliamos o motivo da penalização e, ainda assim, é um nota relativamente alta.

No Código 3, temos o mesmo problema do trecho anterior, em que a classe *ContaController* conhece tanto o objeto Compra quanto o objeto Cliente e os recebe como parâmetro, ambos destacados com sublinhado.

```
void adicionaCompra(String cpf, Fornecedor
fornecedor, Date data, Cliente cliente,
String descricao) {
  Fornecedor fornecedor =
this.fornecedorService.get(fornecedor.getN
ome(), "Erro ao cadastrar compra");
 Cliente cliente =
this.checaCliente(cliente.getCpf(), "Erro
ao cadastrar compra");
```

Código 3: Código recebendo o objeto em si como parâmetro, elevando o valor do CBO para 31.

De forma similar aos outros laboratórios, as notas de design de classes e entre classes foram altas, 10 e 10 respectivamente. Podemos ver que não houve penalização nenhuma na avaliação de acoplamento desta atividade.

Em resumo, podemos observar que estes problemas se repetem na maioria dos casos e são bem comuns, porém não são penalizados, visto que as notas de design ainda são bem altas, independente do nível de acoplamento. Este problema pode se dar por alguns dos motivos abaixo:

1. A planilha guia de correção (Figura 01) não é clara o suficiente para ter uma correção assertiva, principalmente quando se trata de acoplamento, em que o único critério escrito é *"baixo acoplamento"*, sem nenhuma outra descrição ou detalhamento do que deveria ser realmente avaliado.
2. Monitores que corrigem os laboratórios podem deixar passar estes erros ou fazer vista grossa, dado que na disciplina os alunos estão iniciando com orientação a objeto e java.
3. Os corretores podem não ter boas noções de design em si e, em conjunto da planilha com pouco detalhamento, acabam não enxergando os códigos apresentados aqui como problemáticos.

Uma possível solução para estes problemas seria uma reformulação na planilha, para que ela ficasse o mais detalhada e clara possível, além de, caso necessite, os professores mostrarem, um exemplo real de correção, com exemplos e também apresentarem a planilha e darem um *overview*, mesmo que breve, de como as correções deveriam ocorrer.

4.6 Análise das submissões com baixo valor de acoplamento

Podemos ver abaixo no código 4 um exemplo de controller de conta que segue as regras do padrão MVC para o método *adicionaConta*, que é padrão para todas as atividades. Neste exemplo, as instâncias são bem utilizadas, pois services, controller e repository se comunicam da maneira correta, seguindo a arquitetura *MVC*,

```
void adicionaCompra(String cpf, String
fornecedorNome, Date data, String nome,
String descricao) {
  Fornecedor fornecedor =
this.fornecedorService.get(fornecedorNome,
"Erro ao cadastrar compra");
 Cliente cliente = this.checaCliente(cpf,
"Erro ao cadastrar compra");
  Produto produto =
this.checaProduto(fornecedor, nome,
descricao, "Erro ao cadastrar compra");
  this.compraRepository.add(fornecedor,
cliente, data, produto);
}
```

Código 4: Código seguindo os padrões MVC, com valor de CBO 10.

Neste outro trecho de código, temos uma implementação ainda mais limpa do *adicionaConta*, em que é instanciado o controller de Fornecedor e o verificador, mantendo o princípio da responsabilidade única e de código limpo.

```
public void adicionaCompra(String cpf,
String nomeFornecedor, String data, String
nomeProduto, String descricao) {
  verificador.verificaClienteCompra("ao
cadastrar compra",
clienteController.getKeys(), cpf);
```

```
  fornecedorController.adicionaCompra(cpf,
nomeFornecedor, data, nomeProduto,
descricao);
}
```

Código 5: Código limpo seguindo os padrões grasp, com CBO 7.

As notas para estes laboratórios foram 10 para design entre classes e design de classes. Podemos ver a diferença entre os códigos 4 e 5 e 1,2 e 3 e mesmo assim, as notas foram as mesmas, ressaltando que o acoplamento não influencia na avaliação das atividades.

4.7 Correlação entre acoplamento e nota de design

Primeiramente, plotamos gráficos de dispersão para visualizar se existe alguma correlação inicialmente. Foram gerados gráficos de dispersão entre CBO e as notas: design de classes, entre classes e nota geral.

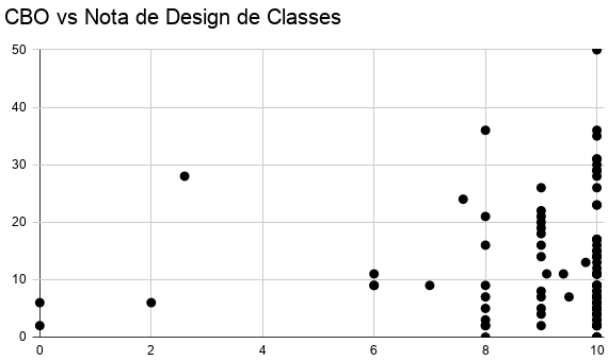

Figura 07: Gráfico de dispersão do CBO vs Nota de design de classes

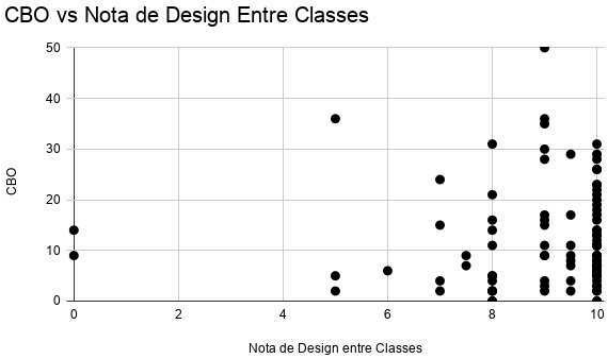

Figura 08: Gráfico de dispersão do CBO vs Nota de design entre classes

Como podemos ver nas Figuras 7 e 8, o valor de CBO pouco influencia nas notas de design, pois mesmo valores muito altos, que indicam níveis de acoplamento alto, são relacionados com notas boas. Na verdade, pelo gráfico podemos inferir que, independente da quantidade de acoplamento, a nota geralmente será alta, independente do critério de design escolhido entre os acima.

Figura 08: Gráfico de dispersão do CBO vs Nota geral da atividade

Já na Figura 8, podemos observar uma variação maior nos valores das notas, porém, mesmo com esta variação, se olharmos para a direita do eixo x, onde ocorre o maior agrupamento que refere às notas mais altas, o CBO parece não ser um fator que influencia nesta nota também, pois, da mesma forma que as métricas anteriores, as observações com maior valor de CBO continuam com notas mais altas.

Porém, não podemos afirmar se há ou não correlação entre CBO e as notas observadas apenas com os gráficos de dispersão. Por isso, decidimos realizar o cálculo de correlação.

Contudo, como foi frisado nas seções anteriores, as amostras não seguem o padrão de distribuição normal, e, portanto, a fórmula de correlação de Pearson não poderia ser aplicada corretamente de forma direta neste estudo. Por isso, optamos por utilizar a correlação de Spearman, em que classificamos as observações em rankings e, com os valores destes rankings, fazemos a correlação, agora utilizando Pearson para obter o valor da métrica desejada. Com isso, conseguimos os resultados indicados na Tabela 01, uma correlação moderada.

Correlação	**Valor**	**Classificação**
CBO e Nota geral da atividade	0,58	Correlação moderada

Tabela 01: Correlações entre CBO e notas gerais

Embora durante o artigo discutimos que o CBO não influenciava na nota de design, vemos que possui uma influência moderada na nota geral. O valor da correlação implica que há correlação moderada entre esta nota e a métrica observada, já nas notas de design podemos ver, inclusive pelos gráficos representados pelas figuras 7 e 8 que não possuem correlação.

É importante ressaltar que esse resultado é contra-intuitivo: quanto mais problemas no acoplamento entre as classes, maior a nota. Entretanto, este resultado parece sinalizar

que existem outras correlações ocorrendo simultaneamente. Por exemplo, quanto mais funcionalidades forem colocadas no laboratório proposto, mais classes e, por consequência, uma possibilidade maior de erros de acoplamento. Ao mesmo tempo, quanto mais funcionalidades codificadas, maior a nota geral por causa dos demais critérios avaliados.

5. Considerações Finais

Dado tudo que foi apresentado, temos que o valor do CBO, métrica utilizada para medir o acoplamento das atividades, não possui um impacto significativo nas notas de design, como foi mostrado nos gráficos, tanto de histograma quanto de dispersão. Apesar disso, ele possui uma correlação moderada na nota geral.

Contudo, precisamos pensar que, apesar de serem alunos iniciando em orientação à objeto e com pouca ou nenhuma noção deste paradigma e seus padrões, eles estão na disciplina justamente para aprendizado, e uma das melhores formas de aprender é corrigindo erros e não os repetindo. Os laboratórios são justamente um meio de avaliar este aprendizado e apresentar os erros cometidos pelos alunos para que ele possa melhorar dali em diante.

Deixando casos como os mostrados anteriormente passarem, corre o risco do discente crer que não há problema algum em realizar implementações de tal forma, prejudicando-o futuramente quando se deparar com exemplos e aplicações reais, em que o conhecimento adquirido deverá ser aplicado.

Por fim, tudo isto pode ser solucionado melhorando os guias de correção dos laboratórios e deixando mais claro as diretrizes e o que deve ser avaliado em cada atividade, para que a avaliação seja melhor executada e o aprendizado seja mais proveitoso.

6. BIBLIOGRAFIA

[1] Medium 2021. [online] Available at: <https://medium.com/emblatech/the-importance-of-code-quality-ac7afa598c0d> [Accessed 28 March 2021].

[2] Aloysius, A., and L. Arockiam. "Coupling complexity metric: A cognitive approach." *International Journal of Information Technology and Computer Science (IJITCS)* 4.9 (2012): 29-35.

[3] Shyam R. Chidamber, Chris F. Kemerer. Towards a Metrics Suite for object-oriented Design. In Proc. OOPSLA '91, ACM 1991, 197-211.

[4] Hitz, Martin & Montazeri, Behzad. (1995). Measuring coupling and cohesion in object-oriented systems.

[5] CodeMR. 2021. *CodeMR | Measure, visualise, and improve code quality | Better Code Better Quality!*. [online] Available at: <https://www.codemr.co.uk/> [Accessed 28 March 2021].

[6] Virtualmachinery.com. 2021. *Virtual Machinery - Sidebar 3 - WMC, CBO, RFC, LCOM, DIT, NOC - 'The Chidamber and Kemerer Metrics'*. [online] Available at: <http://www.virtualmachinery.com/sidebar3.htm> [Accessed 28 March 2021].

[7] CodeMR. 2021. *CodeMR | Documents*. [online] Available at: <https://www.codemr.co.uk/documents/> [Accessed 28 March 2021].

[8] Minh, N. and Minh, N., 2021. *10 Java Core Best Practices Every Java Programmer Should Know*. [online] Codejava.net. Available at: <https://www.codejava.net/coding/10-java-core-best-practices-every-java-programmer-should-know> [Accessed 29 March 2021].

[9] Medium. 2021. *Demeter's Law: Don't talk to strangers!*. [online] Available at: <https://betterprogramming.pub/demeters-law-don-t-talk-to-strangers-87bb4af11694> [Accessed 29 March 2021].

[10] DevMedia. 2021. *Arquitetura - O Princípio da responsabilidade única*. [online] Available at: <https://www.devmedia.com.br/arquitetura-o-principio-da-responsabilidade-unica/18700> [Accessed 29 March 2021].

[11] Hokstad, V., 2021. *Why coupling is always bad / Cohesion vs. coupling*. [online] Hokstad.com. Available at: <https://hokstad.com/why-coupling-is-always-bad-cohesion-vs-coupling> [Accessed 29 March 2021].

[12] Aprna Tripathi. 2018. An Analytical and Comparative Review of Cohesion Metrics. In *Proceedings of the 2018 International Conference on Software Engineering and Information Management* (*ICSIM2018*). Association for Computing Machinery, New York, NY, USA, 17–25. DOI:https://doi.org/10.1145/3178461.3178479

[13] Luque, Leandro. (2010). Coesão e Acoplamento em Sistemas OO. JavaMagazine. 77. 68-74.