Relatório Completo - Análise de Qualidade de Código

Projeto: sisVendas-BP - Sistema de Ponto de Vendas
Disciplina: Boas práticas de Programação - Unidade II
Equipe: - Brenda Gomes da Silva (mat. 20240021503) - Carlos Eduardo Miranda da Silva (mat. 20200118231) - Hugo Jose de Lima Nunes (mat. 20240062319)

Sumário

1. Introdução

Este relatório apresenta a análise completa de qualidade do código do projeto sisVendas-BP, um sistema de ponto de vendas desenvolvido em Java seguindo boas práticas de programação (SOLID, camadas claras, testes).

1.1 Ferramentas utilizadas

Ferramenta	Versão	Finalidade
PMD	6.55.0	Análise estática de código (parte I)
CK	0.7.0	Métricas OO - Chidamber e Kemerer (parte II)
Python + Matplotlib	3.x	Geração de gráficos e estatísticas

1.2 Estrutura do projeto analisado

```
src/main/java/com/sisvendas/
├── demo/                # Classe de demonstração
├── Exception/           # Exceções customizadas
├── model/               # Entidades de domínio
├── repository/          # Interfaces e implementações de repositório
├── service/             # Lógica de negócio
│   └── dto/             # Data transfer objects
├── util/                # Classes utilitárias
└── view/                # Apresentação (console)
```

Totais do projeto: - 16 classes analisadas - 66 métodos - 461 linhas de código (LOC)

---

2. Parte I - Análise estática de código

2.1 Resumo das violações encontradas

A ferramenta PMD identificou **213 violações** distribuidas em **30 categorias** diferentes. Abaixo está o resumo por categoria:

Categoria	Quantidade	Prioridade	Ruleset
MethodArgumentCouldBeFinal	46	3	Code Style
LawOfDemeter	46	3	Design
LocalVariableCouldBeFinal	38	3	Code Style
SystemPrintln	18	2	Best Practices
ShortVariable	13	3	Code Style
OnlyOneReturn	7	3	Code Style
UnnecessaryFullyQualifiedName	5	4	Code Style
IfStmtsMustUseBraces	5	3	Code Style
ControlStatementBraces	5	3	Code Style
AvoidFieldNameMatchingMethodName	5	3	Error Prone
LongVariable	4	3	Code Style
MissingSerialVersionUID	3	3	Error Prone
AvoidInstantiatingObjectsInLoops	3	3	Performance
AtLeastOneConstructor	3	3	Code Style
CyclomaticComplexity	1	3	Design
NPathComplexity	1	3	Design
CognitiveComplexity	1	3	Design
DataClass	1	3	Design
Outros	16	-	Diversos

2.2 Análise detalhada por categoria de problema

Categoria 1: Design - Complexidade excessiva

Regras Violadas: CyclomaticComplexity, NPathComplexity, CognitiveComplexity, ModifiedCyclomaticComplexity

Arquivo afetado: `VendaService.java` - Método `registrarVenda()`

Problemas identificados:

Metrica	Valor encontrado	Limite aceitável
Cyclomatic Complexity	28	10
NPath Complexity	16.848	200
Cognitive Complexity	23	15

Mensagens do PMD:

```
The method 'registrarVenda(TipoVenda, List, Optional)' has a cyclomatic c
The method 'registrarVenda(TipoVenda, List, Optional)' has an NPath compl
The method 'registrarVenda(TipoVenda, List, Optional)' has a cognitive co
```

Trecho problemático (linhas 34-99):

```
public Venda registrarVenda(TipoVenda tipo,
                            List<Par<String, Integer>> itensSolicitados,
                            Optional<EnderecoEntrega> enderecoEntrega) {
    if (tipo == null) {
        throw new ValidacaoVendaException("tipo de Venda e Obrigatório");
    }
    if (itensSolicitados == null || itensSolicitados.isEmpty()) {
        throw new ValidacaoVendaException("deve haver ao menos 1 item");
    }
    for (Par<String, Integer> par : itensSolicitados) {
        if (par == null || par.primeiro() == null || par.primeiro().isBla
            throw new ValidacaoVendaException("Código do produto e Obriga
        }
        if (par.segundo() == null || par.segundo() < 1) {
            throw new ValidacaoVendaException("quantidade deve ser >= 1")
        }
    }
    // ... mais 50 Linhas de Lógica complexa
}
```

Sugestão de correção: Extrair a lógica em métodos menores com responsabilidades únicas:

```
public Venda registrarVenda(TipoVenda tipo,
                            List<Par<String, Integer>> itensSolicitados,
                            Optional<EnderecoEntrega> enderecoEntrega) {
    validarParametrosEntrada(tipo, itensSolicitados, enderecoEntrega);
    Map<Produto, Integer> produtosValidados = validarDisponibilidadeEstoq
    List<ItemVenda> itens = processarItensVenda(produtosValidados);
    return criarERegistrarVenda(tipo, itens, enderecoEntrega);
}

private void validarParametrosEntrada(TipoVenda tipo,
                                      List<Par<String, Integer>> itens,
                                      Optional<EnderecoEntrega> endereco)
    Objects.requireNonNull(tipo, "tipo de Venda e Obrigatório");
    if (itens == null || itens.isEmpty()) {
        throw new ValidacaoVendaException("deve haver ao menos 1 item");
    }
    validarItens(itens);
    validarEnderecoParaVendaWeb(tipo, endereco);
}

private void validarItens(List<Par<String, Integer>> itens) {
    for (Par<String, Integer> par : itens) {
        if (par == null || par.primeiro() == null || par.primeiro().isBla
            throw new ValidacaoVendaException("Código do produto e Obriga
        }
        if (par.segundo() == null || par.segundo() < 1) {
            throw new ValidacaoVendaException("quantidade deve ser >= 1")
        }
    }
}
```

Categoria 2: Best Practices - Uso de System.out.println

Regra Violada: SystemPrintln

Quantidade: 18 ocorrências

Arquivos Afetados: `ConsoleVendasPresenter.java`

Problema: O uso de `System.out.println` e `System.out.printf` e considerado uma má prática porque: - Dificulta testes unitários - Não permite configuração de níveis de log - Não permite redirecionamento para arquivos - Performance inferior em produção

Mensagens do PMD:

```
System.out.println is used (linha 35)
System.out.printf is used (linha 57)
System.out.print is used (linha 30)
```

Trecho problemático:

```
public void imprimirListaVendas(List<Venda> vendas) {
    if (vendas.isEmpty()) {
        System.out.println("Nenhuma Venda registrada.");  // Violação
        return;
    }
    for (Venda v : vendas) {
        imprimirVenda(v);
        System.out.println("-----------------------------");  // Violação
    }
}
```

Sugestão de correção: Usar um framework de Logging ou injetar dependencia:

```
// Opcao 1: Usar SLF4J/Log4j
private static Final Logger LOGGER = LoggerFactory.getLogger(ConsoleVenda

public void imprimirListaVendas(List<Venda> vendas) {
    if (vendas.isEmpty()) {
        LOGGER.info("Nenhuma Venda registrada.");
        return;
    }
    vendas.forEach(v -> {
        imprimirVenda(v);
        LOGGER.info("-----------------------------");
    });
}

// Opcao 2: Injetar PrintStream para facilitar testes
public class ConsoleVendasPresenter {
    private Final PrintStream output;

    public ConsoleVendasPresenter() {
        this(System.out);
    }

    public ConsoleVendasPresenter(PrintStream output) {
        this.output = output;
    }

    public void imprimirListaVendas(List<Venda> vendas) {
        if (vendas.isEmpty()) {
            output.println("Nenhuma Venda registrada.");
            return;
        }
    }
}
```

Categoria 3: Code Style - Parâmetros e variáveis não-final

Regras Violadas: MethodArgumentCouldBeFinal, LocalVariableCouldBeFinal

Quantidade: 84 ocorrências (46 + 38)

Arquivos afetados: Praticamente todos os arquivos

Problema: Parâmetros e variáveis locais que não são reatribuídos devem ser declarados como `Final` para: - Indicar intenção de imutabilidade - Prevenir reatribuições acidentais - Melhorar legibilidade

Mensagem do PMD:

```
Parameter 'tipo' is not assigned and could be declared Final
Local variable 'vendas' could be declared Final
```

Trecho problemático:

```
public Venda registrarVenda(TipoVenda tipo,  // poderia ser final
                            List<Par<String, Integer>> itensSolicitados,
                            Optional<EnderecoEntrega> enderecoEntrega) {
    List<Venda> vendas = vendaRepository.listarTodas();  // poderia ser f
}
```

Sugestão de correção:

```
public Venda registrarVenda(Final TipoVenda tipo,
                            Final List<Par<String, Integer>> itensSolicit
                            Final Optional<EnderecoEntrega> enderecoEntre
    Final List<Venda> vendas = vendaRepository.listarTodas();
}
```

Categoria 4: Design - Violação da lei de Demeter

Regra Violada: LawOfDemeter

Quantidade: 46 ocorrências

Arquivos afetados: `VendaService.java` , `ConsoleVendasPresenter.java`

Problema: A lei de Demeter (princípio do menor conhecimento) diz que um objeto deve falar apenas com seus "amigos imediatos", evitando cadeias de chamadas como `a.getB().getC().doSomething()` .

Mensagens do PMD:

```
Potential violation of Law of Demeter (method chain calls)
Potential violation of Law of Demeter (object not created locally)
```

Trechos problemáticos:

```
// Cadeia de chamadas - violação
String Código = item.getProduto().getCodigo();
String nome = item.getProduto().getNome();

// Uso de streams com múltiplas operações
int totalItens = vendas.stream()
    .flatMap(v -> v.getItens().stream())
    .mapToInt(ItemVenda::getQuantidade)
    .sum();
```

Sugestão de correção:

```
// Opção 1: Método delegador em ItemVenda
public class ItemVenda {
    public String getCodigoProduto() {
        return produto.getCodigo();
    }
    public String getNomeProduto() {
        return produto.getNome();
    }
}

// Uso simplificado
String Código = item.getCodigoProduto();

// Opcao 2: Para streams, aceitar como idiomatico do Java moderno
// (a regra LawOfDemeter Pode ser desabilitada para streams)
```

Categoria 5: Code Style - Variáveis com nomes curtos

Regra Violada: ShortVariable

Quantidade: 13 ocorrências

Arquivos Afetados: Varios arquivos

Problema: Nomes de Variáveis muito Curtos (< 3 Caracteres) prejudicam a legibilidade do código.

Mensagens do PMD:

```
Avoid variables with short names like uf
Avoid variables with short names like v
Avoid variables with short names like o
Avoid variables with short names like p
Avoid variables with short names like id
```

Trechos Problematicos:

```
// EnderecoEntrega.java
private Final String uf;  // Muito curto

// Produto.java - metodo equals
public boolean equals(Object o) {  // 'o' muito curto

// VendaService.java
Produto p = entry.getKey();  // 'p' muito curto
int qtd = entry.getValue();  // 'qtd' aceitável mas poderia ser melhor
```

Sugestão de correção:

```
// EnderecoEntrega.java
private Final String Estado;  // ou 'unidadeFederativa'

// Produto.java
public boolean equals(Object other) {
    if (this == other) return true;
    if (other == null || getClass() != other.getClass()) return false;
    Produto produto = (Produto) other;
    return Objects.equals(Código, produto.Código);
}

// VendaService.java
Produto produto = entry.getKey();
int quantidade = entry.getValue();
```

Exceção: O nome `id` e amplamente aceito e pode ser mantido.

Categoria 6: Code Style - Estruturas de controle sem chaves

Regras Violadas: IfStmtsMustUseBraces, ControlStatementBraces

Quantidade: 10 ocorrências

Arquivos afetados: `Produto.java` , `Venda.java` , `InMemoryProdutoRepository.java`

Problema: Blocos if/else sem chaves são propensos a erros quando o código é modificado posteriormente.

Mensagens do PMD:

```
Avoid using if statements without curly braces
This statement should have braces
```

Trecho problemático:

```
// Produto.java - metodo equals
public boolean equals(Object o) {
    if (this == o) return true;  // Sem chaves
    if (o == null || getClass() != o.getClass()) return false;  // Sem ch
    Produto produto = (Produto) o;
    return Objects.equals(Código, produto.Código);
}

// InMemoryProdutoRepository.java
if (Código == null) return Optional.empty();  // Sem chaves
```

Sugestão de correção:

```
// Produto.java
public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (o == null || getClass() != o.getClass()) {
        return false;
    }
    Produto produto = (Produto) o;
    return Objects.equals(Código, produto.Código);
}

// InMemoryProdutoRepository.java
if (Código == null) {
    return Optional.empty();
}
```

Categoria 7: Error Prone - Falta de serialVersionUID

Regra Violada: MissingSerialVersionUID

Quantidade: 3 ocorrências

Arquivos afetados: Todas as classes de exceção

Problema: Classes que estendem Exception (que implementa Serializable) devem declarar um serialVersionUID para garantir compatibilidade de serialização.

Mensagens do PMD:

```
Classes implementing Serializable should set a serialVersionUID
```

Arquivos: - `EstoqueInsuficienteException.java` - `ProdutoNaoEncontradoException.java` - `ValidacaoVendaException.java`

Trecho problemático:

```
public class EstoqueInsuficienteException extends RuntimeException {
    // Falta serialVersionUID
    public EstoqueInsuficienteException(String Código, int solicitado, in
        super("Estoque insuficiente...");
    }
}
```

Sugestão de correção:

```
public class EstoqueInsuficienteException extends RuntimeException {
    private static Final long serialVersionUID = 1L;

    public EstoqueInsuficienteException(String Código, int solicitado, in
        super(String.format("Estoque insuficiente para produto %s: solici
                            Código, solicitado, disponivel));
    }
}
```

Categoria 8: Performance - Instanciação de objetos em loops

Regra Violada: AvoidInstantiatingObjectsInLoops

Quantidade: 3 ocorrências

Arquivos Afetados: `VendaService.java`

Problema: Criar novos objetos dentro de loops pode causar pressão no Garbage Collector e afetar performance.

Mensagens do PMD:

```
Avoid instantiating new objects inside Loops
```

Trecho problemático:

```
// VendaService.java - listarResumoVendas()
for (Venda v : vendas) {
    for (ItemVenda item : v.getItens()) {
        String Código = item.getProduto().getCodigo();
        String nome = item.getProduto().getNome();
        agregados.merge(Código,
            new ResumoPorProduto(Código, nome, item.getQuantidade(), item
            (oldVal, newVal) -> new ResumoPorProduto(  // Nova instancia
                oldVal.Código(),
                oldVal.nome(),
                oldVal.quantidadeVendida() + newVal.quantidadeVendida(),
                oldVal.valorTotalProduto() + newVal.valorTotalProduto()
            ));
    }
}
```

Sugestão de correção: Para este caso específico, a criação de objetos é necessária devido à imutabilidade dos records. Uma alternativa seria usar uma classe mutável internamente:

```
// Classe auxiliar mutável para agregação
private static class ResumoPorProdutoBuilder {
    private Final String Código;
    private Final String nome;
    private int quantidadeVendida;
    private double valorTotalProduto;

    void Adicionar(int quantidade, double valor) {
        this.quantidadeVendida += quantidade;
        this.valorTotalProduto += valor;
    }

    ResumoPorProduto build() {
        return new ResumoPorProduto(Código, nome, quantidadeVendida, valo
    }
}
```

Categoria 9: Design - Data Class

Regra Violada: DataClass

Quantidade: 1 ocorrência

Arquivo afetado: `EnderecoEntrega.java`

Problema: Classes que contem apenas dados (getters/setters) sem comportamento real violam princípios OO.

Mensagem do PMD:

```
The class 'EnderecoEntrega' is suspected to be a Data Class (WOC=0.000%,
```

Trecho problemático:

```
public class EnderecoEntrega {
    private Final String destinatario;
    private Final String logradouro;
    private Final String Número;
    private Final String bairro;
    private Final String cidade;
    private Final String uf;
    private Final String CEP;

    // Apenas getters...
    public String getDestinatario() { return destinatario; }
    public String getLogradouro() { return logradouro; }
    // ... mais getters
}
```

Sugestão de correção: Opcao 1 - Transformar em Record (Java 17+):

```
public record EnderecoEntrega(
    String destinatario,
    String logradouro,
    String Número,
    String bairro,
    String cidade,
    String uf,
    String CEP
) {
    public EnderecoEntrega {
        Objects.requireNonNull(destinatario, "destinatario e Obrigatório"
        Objects.requireNonNull(logradouro, "logradouro e Obrigatório");
        // ... Validações
    }

    public String formatarEndereco() {
        return String.format("%s, %s - %s, %s/%s - CEP: %s",
            logradouro, Número, bairro, cidade, uf, CEP);
    }
}
```

Opção 2 - Adicionar comportamentos:

```
public class EnderecoEntrega {
    // ... campos

    public String formatarEnderecoCompleto() {
        return String.format("%s\n%s, %s - %s\n%s/%s\nCEP: %s",
            destinatario, logradouro, Número, bairro, cidade, uf, CEP);
    }

    public boolean isEstadoValido() {
        return uf != null && uf.length() == 2;
    }

    public boolean isCepValido() {
        return CEP != null && CEP.matches("\\d{5}-?\\d{3}");
    }
}
```

2.3 Resumo de correções sugeridas por arquivo

Arquivo	Violações	Principais correções
VendaService.java	68	Extrair Métodos, Adicionar Final, Reduzir Complexidade
ConsoleVendasPresenter.java	32	Substituir System.out por Logging
Produto.java	22	Adicionar Chaves, Renomear Variáveis
Venda.java	21	Adicionar Chaves, Renomear Variáveis
EnderecoEntrega.java	16	Converter para record, Renomear uf
DemoVendas.java	10	Adicionar Final em Variáveis
InMemoryProdutoRepository.java	8	Adicionar Chaves, Final
Exceptions (3 arquivos)	9	Adicionar serialVersionUID
Outros	27	Correções Menores

---

3. Parte II - Análise das métricas de código

3.1 Métricas utilizadas

- **WMC (Weighted Methods per Class):** Soma da complexidade dos métodos
- **DIT (Depth of Inheritance Tree):** Profundidade da árvore de herança
- **NOC (Number of Children):** Número de subclasses diretas
- **CBO (Coupling Between Objects):** Acoplamento entre objetos
- **RFC (Response For a Class):** Métodos que podem ser invocados
- **LCOM (Lack of Cohesion of Methods):** Falta de coesão
- **LOC (Lines of Code):** Linhas de código

3.2 Análise I - Avaliação detalhada

3.2.1 Análise negativa - 5 casos de potenciais problemas

Caso negativo 1: VendaService.registrarVenda() - Alta complexidade

Arquivo: `src/main/java/com/sisvendas/service/VendaService.java`
Metodo: `registrarVenda(TipoVenda, List, Optional)`

Metrica	Valor	Limite	Status
Complexidade Ciclomática	28	< 10	CRÍTICO
NPath Complexity	16.848	< 200	CRÍTICO
Cognitive Complexity	23	< 15	ALTO
LOC do metodo	52	< 30	ALTO
WMC (Classe)	26	< 20	ALTO

Justificativa: A combinacao de multiplas métricas altas indica que o metodo Concentra Muitas Responsabilidades: Validação de Parâmetros (4 Verificações), Validação de Estoque (2 Loops), Processamento de itens e Criação da Venda. A Alta complexidade Ciclomática (28) Significa 28 Caminhos Independentes de Execução, Tornando Extremamente Difícil Testar todos os Cenários.

Caso negativo 2: VendaService - Alto acoplamento

Arquivo: `src/main/java/com/sisvendas/service/VendaService.java`

Metrica	Valor	Média projeto	Status
CBO	15	3.0	5x acima
RFC	68	9.88	7x acima
Fan-out	15	-	ALTO

Justificativa: O CBO de 15 combinado com RFC de 68 indica que a classe depende de muitas outras classes e pode invocar um número excessivo de métodos. Isso viola o princípio de baixo acoplamento e torna a classe sensível a mudanças em qualquer uma das 15 dependências.

Caso negativo 3: EnderecoEntrega - Baixa coesão

Arquivo: `src/main/java/com/sisvendas/model/EnderecoEntrega.java`

Metrica	Valor	Interpretacao
LCOM	22	MUITO ALTO
WOC	0.000%	SEM comportamento
TCC	0.25	Baixo
NOAM	7	MUITOS acessores

Justificativa: O LCOM de 22 Indica que os Métodos da Classe nao Colaboram Entre si (baixa Coesão). O WOC de 0% Confirma que nao ha Comportamento Real, Apenas getters. Uma Classe coesa Deveria ter Métodos que Trabalham Juntos Sobre os Mesmos Dados.

Caso Negativo 4: ConsoleVendasPresenter - Problemas estruturais

Arquivo: `src/main/java/com/sisvendas/view/ConsoleVendasPresenter.java`

Metrica	Valor	Status
WMC	15	MODERADO
RFC	43	ALTO
LCOM	6	ALTO
TCC	0.0	CRÍTICO

Justificativa: O TCC (Tight Class Cohesion) de 0.0 Indica que nenhum par de métodos públicos compartilha acesso a atributos. Isso sugere que a classe e uma coleção de métodos independentes que poderiam estar em classes separadas, ou que falta um estado interno adequado.

Caso Negativo 5: Venda - Classe grande com baixa coesão

Arquivo: `src/main/java/com/sisvendas/model/Venda.java`

Metrica	Valor	Status
WMC	17	MODERADO
LCOM	23	MUITO ALTO
LOC	84	MODERADO
TCC	0.083	MUITO baixo
LCC	0.083	MUITO baixo

Justificativa: A combinacao de LCOM alto (23) com TCC/LCC muito baixos (0.083) indica sérios problemas de coesão. A Classe Venda Inclui uma Inner class Builder que Contribui para Essas Métricas. Embora o Builder Seja um padrão válido, ele infla os indicadores de coesão da classe externa.

3.2.2 Análise positiva - 5 casos de boas situações

Caso positivo 1: InMemoryVendaRepository - Excelente simplicidade

Arquivo:
`src/main/java/com/sisvendas/repository/memory/InMemoryVendaRepository.java`

Metrica	Valor	Interpretacao
WMC	2	Excelente
CBO	2	Excelente
RFC	2	Excelente
LCOM	0	PERFEITO
LOC	9	ÓTIMO
TCC	1.0	PERFEITO

Justificativa: Todas as métricas indicam uma classe extremamente bem projetada. O LCOM de 0 Significa Coesão Total - todos os Métodos Trabalham com o Mesmo Atributo (Storage). O TCC de 1.0 Confirma que os Métodos Públicos Estão Intimamente Relacionados. Esta Classe Exemplifica o Single Responsibility Principle.

Caso positivo 2: ItemVenda - Modelo equilibrado

Arquivo: `src/main/java/com/sisvendas/model/ItemVenda.java`

Metrica	Valor	Interpretacao
WMC	5	BOM
CBO	1	Excelente
LCOM	0	PERFEITO
LOC	22	Ideal
TCC	0.5	BOM

Justificativa: A classe apresenta equilibrio entre tamanho (22 LOC), Complexidade (WMC=5) e coesão (LCOM=0). O metodo getSubtotal() adiciona comportamento de negócio, elevando a classe acima de uma simples data class. O CBO de 1 Indica Mínimo Acoplamento.

Caso positivo 3: TipoVenda - Enum minimalista

Arquivo: `src/main/java/com/sisvendas/model/TipoVenda.java`

Metrica	Valor	Interpretacao
WMC	0	PERFEITO
CBO	0	PERFEITO
RFC	0	PERFEITO
LCOM	0	PERFEITO
LOC	1	Minimalista

Justificativa: Todas as métricas zeradas indicam que o enum nao adiciona Complexidade ao sistema. E a Representação mais Simples Possível para tipos de Venda, Sendo Type-safe e Auto-documentado.

Caso positivo 4: ProdutoNaoEncontradoException - Exceção bem definida

Arquivo:
`src/main/java/com/sisvendas/Exception/ProdutoNaoEncontradoException.java`

Metrica	Valor	Interpretacao
WMC	1	PERFEITO
CBO	0	PERFEITO
RFC	0	PERFEITO
LCOM	0	PERFEITO
DIT	4	Esperado
LOC	5	Mínimo

Justificativa: A classe segue o princípio de fazer uma única coisa bem feita. O DIT de 4 e Esperado para Exceções Java (Hierarquia Exception). Nao ha Código Desnecessário, Apenas o Construtor que Formata a Mensagem de Erro.

Caso positivo 5: Venda.Builder - Padrão Builder exemplar

Arquivo: `src/main/java/com/sisvendas/model/Venda.java` (Classe Interna)

Metrica	Valor	Interpretacao
WMC	6	BOM
CBO	5	Adequado
RFC	1	Excelente
LOC	30	BOM

Justificativa: O WMC de 6 Corresponde Exatamente aos 6 Métodos do Builder (id, dataHora, tipo, itens, enderecoEntrega, build), indicando Complexidade minima por metodo. A fluent interface permite construção imutável e legível de objetos venda.

3.3 Análise II - Estatísticas gerais

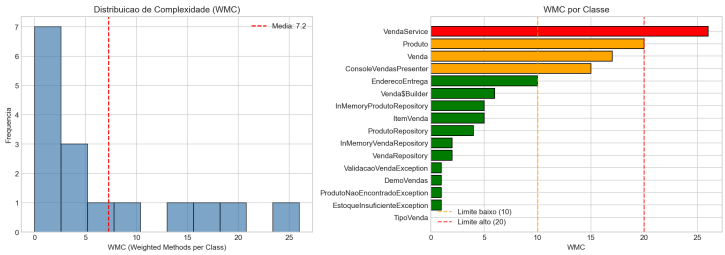
3.3.1 Tabela de estatísticas

Metrica	Total	Média	Mediana	Máximo	Mínimo	Desvio padrão
WMC	116	7.25	4.50	26	0	8.01
DIT	25	1.56	1.00	4	1	1.21
NOC	0	0.00	0.00	0	0	0.00
CBO	48	3.00	1.00	15	0	4.03
RFC	158	9.88	2.00	68	0	18.92
LCOM	81	5.06	0.00	23	0	8.02
LOC	461	28.81	18.00	84	1	29.40

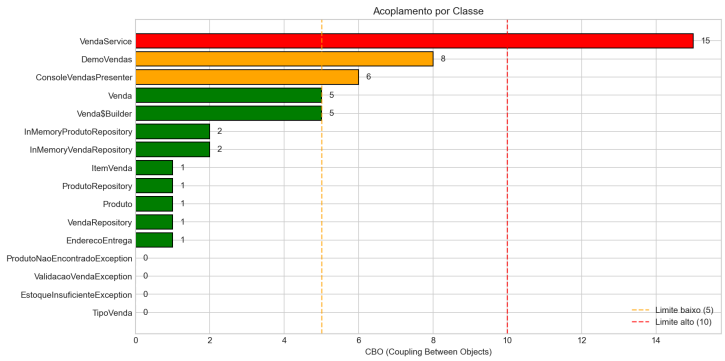
3.3.2 Gráficos gerados

Os gráficos a seguir foram gerados automaticamente e estão disponíveis em `analise-metricas/graficos/` :

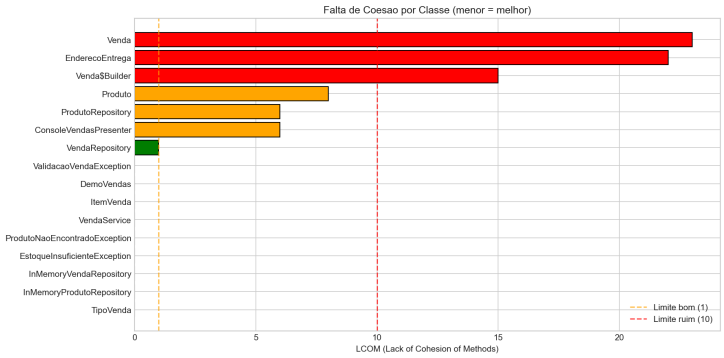
1. Distribuição de Complexidade (WMC)



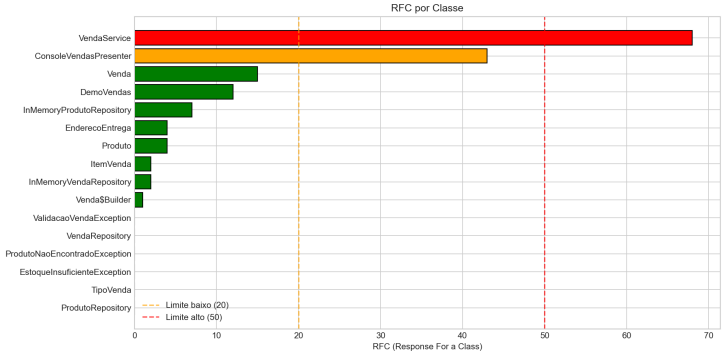
2. Acoplamento por Classe (CBO)



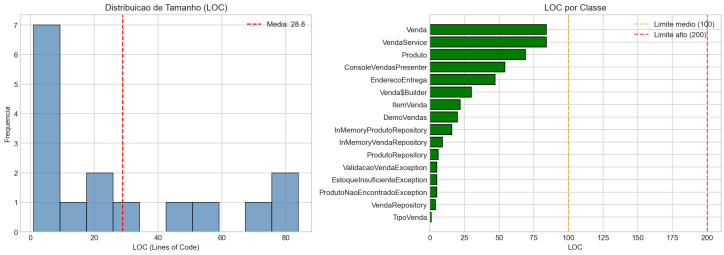
3. Falta de Coesão (LCOM)



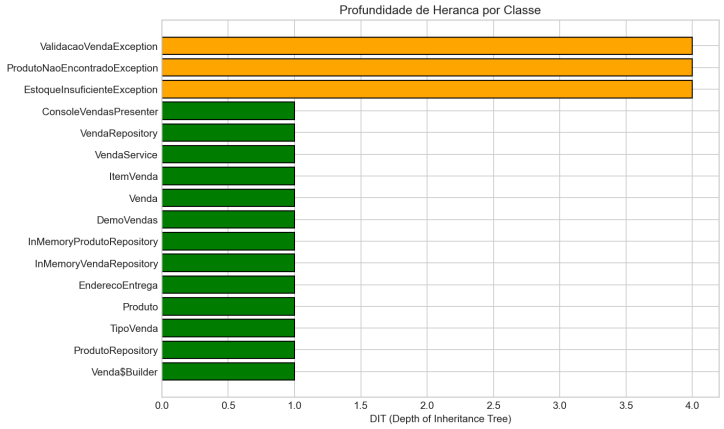
4. Response For Class (RFC)



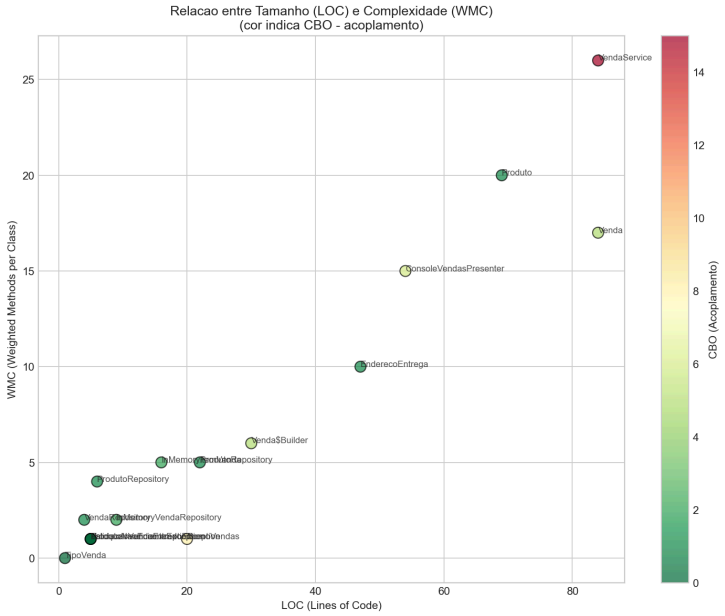
5. Distribuição de Linhas de Código (LOC)



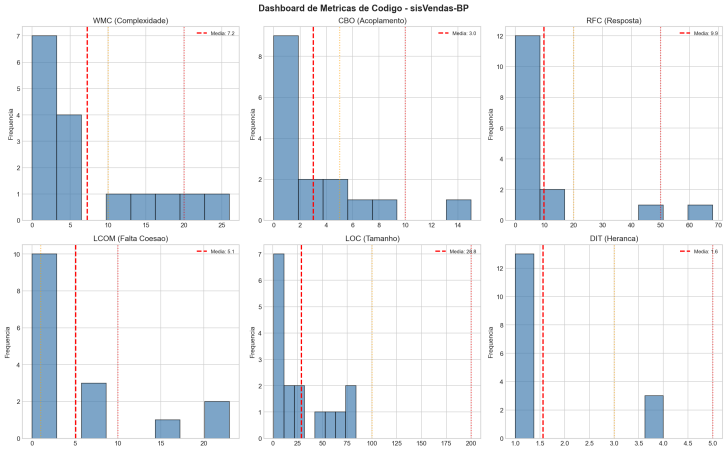
6. Profundidade de Herança (DIT)



7. Correlação WMC vs LOC



8. Dashboard Geral de Métricas



3.3.3 Análise por pacote

Pacote	Classes	WMC Total	WMC médio	CBO médio	LOC total
model	6	58	9.67	2.17	257
service	1	26	26.00	15.00	84
repository.memory	2	7	3.50	2.00	25
repository	2	6	3.00	1.50	10
Exception	3	3	1.00	0.00	15
view	1	15	15.00	6.00	54
demo	1	1	1.00	8.00	20

3.4 Parecer geral sobre o projeto

Complexidade

- **Avaliação:** Moderada
- **Justificativa:** Média WMC de 7.25 está aceitável. Porém, VendaService (WMC=26) requer refatoração urgente.

Tamanho

- **Avaliação:** Adequado
- **Justificativa:** 461 LOC em 16 classes indica projeto compacto e bem dimensionado.

Acoplamento

- **Avaliação:** Bom (com exceção)
- **Justificativa:** Média CBO de 3.0 é excelente. VendaService (CBO=15) é outlier problemático.

Coesão

- **Avaliação:** Variável
- **Justificativa:** Mediana LCOM de 0.0 indica boa coesão geral, mas Venda (23) e EnderecoEntrega (22) precisam atenção.

Qualidade Estrutural

- **Avaliação:** BOA
- **Justificativa:** Arquitetura em camadas clara, uso de Interfaces e padroes como Repository e Builder.

Facilidade de Teste

- **Avaliação:** Moderada
- **Justificativa:** Maioria das classes é testável, Exceto VendaService devido à alta complexidade.

---

4. Conclusão geral

4.1 Resumo dos principais problemas

Prioridade	Problema	Local	Métricas
Alta	Complexidade excessiva	VendaService.registrarVenda()	CC=28, NPath=16848
Alta	Alto acoplamento	VendaService	CBO=15, RFC=68
Média	Uso de System.out	ConsoleVendasPresenter	18 violações
Média	Data Class	EnderecoEntrega	WOC=0%, LCOM=22
BAIXA	Variáveis não-final	Todo o Projeto	84 violações
BAIXA	Falta de chaves	Produto, Venda	10 violações

4.2 Principais recomendações

1. **Refatorar VendaService** - Extrair métodos e reduzir complexidade
2. **Implementar logging** - Substituir System.out.println por SLF4J
3. **Adicionar serialVersionUID** - Em todas as exceções
4. **Padronizar estilo** - Usar chaves em todos os blocos if/else
5. **Converter para records** - EnderecoEntrega e DTOs

4.3 Nota final

Critério	Peso	Nota
Complexidade	20%	6.5
Acoplamento	20%	7.5
Coesão	20%	7.0
Qualidade Estrutural	20%	8.0
Boas práticas	20%	7.5
Nota final	**100%**	**7.3/10**

O projeto sisVendas-BP demonstra boa qualidade geral, com arquitetura bem definida e maioria das classes seguindo boas práticas. Os principais pontos de atenção são a classe VendaService e o uso de System.out.println.

---

5. Anexos

5.1 Arquivos de análise gerados

```
analise-metricas/
├── metricas-ck-Classes.csv      # Métricas de classes (CK)
├── metricas-ck-Métodos.csv      # Métricas de métodos (CK)
├── metricas-ck-campos.csv       # Métricas de campos
├── metricas-ck-Variáveis.csv    # Métricas de variáveis
├── pmd-report.xml               # Relatório PMD (213 Violações)
├── cpd-report.xml               # Relatório de duplicacao
├── estatisticas-metricas.csv    # Estatísticas calculadas
└── graficos/
    ├── wmc_distribuicao.png
    ├── cbo_acoplamento.png
    ├── lcom_coesao.png
    ├── rfc_resposta.png
    ├── loc_tamanho.png
    ├── dit_heranca.png
    ├── scatter_wmc_loc.png
    └── dashboard_metricas.png
```

5.2 Scripts de execução

```
# Executar análise CK
./scripts/executar-ck.sh

# Executar análise PMD
./scripts/executar-pmd.sh

# Gerar graficos
source venv/bin/activate
python scripts/gerar-graficos.py
```

5.3 URL do vídeo

🔗 **https://drive.google.com/drive/folders/1-vO1-ocfjQEjH1KEFBCdjNTlF_TZbDCZ?usp=sharing**

Link para a gravação do vídeo de apresentação do relatório

Todos os integrantes devem participar da apresentação (máximo 20 minutos).
