



Preparing Data

# Qualidade e Teste de Software

## **Grupo BIC**

Daniel Fontoura

Daniele Pimenta

Irhael Chagas

João Gabriel Otogali

# Recapitulando...

- Projeto escolhido: Banco BIC
- Aplicação orientada a objetos Java
- Trata-se de uma aplicação Java de linha de comando (CLI), e sua interação com o usuário ocorre exclusivamente por meio de impressão e leitura no console.

```
=====
[0] - Encerrar programa
[1] - Acessar conta
[2] - Criar conta
=====

> 2
=====

Tipo de cliente:
[0] - Cancelar
[1] - Pessoa física
[2] - Pessoa jurídica
>
1
[DIGITE CORRETAMENTE AS INFORMACOES A SEGUIR, NUMERO MAX DE CARACTERES ACEITO >> 60
CEP:
> 12345678910
Numero da Residencia:
> 123
Complemento (Opcional):
>
```

## Objetivos desta entrega:

- Expandir a cobertura de testes (unitários, integração, sistema)
- Aplicar técnicas de Caixa Branca (Estrutural), Caixa Preta (Funcional)
- Validação da eficácia via Mutação

# Ferramentas utilizadas



**System  
Lambda**

**GitHub**

sonarqube



# Testes Unitários

- Complexidade ciclomática  $\geq 10$
- Não CRUD

Para a implementação dos testes, as seguintes classes foram utilizadas:

- VerificadorTransacao.java (Daniel Fontoura)
- Conta.java (Daniele Pimenta)
- VerificadorClientes.java (Irhael Chagas)
- VerificadorPix.java (João Gabriel Otogali)
- VerificadorData.java (João Gabriel Otogali)

# Análise estatística

**sonarqube** 

# Análise estatística (SonarQube)

- **26 issues** apontadas nas classes testadas
- A análise foi realizada em cada classe de responsabilidade de cada integrante do grupo
- Principais apontamentos:
  - Renomear variáveis, métodos e pacotes;
  - Refatoração de código para reduzir complexidade;
  - Tornar construtores ou parâmetros privados;
  - Etc.

# Análise estatística (SonarQube)

Exemplo com a classe VerificadorTransacao.java

The screenshot displays the SonarQube IDE interface. The top bar shows the project name 'PrintUtils' and the current report for 'Report - dez. 02, 15:40'. The left sidebar indicates 'Found 7 Issues' for the file 'VerificadorTransacao.java'. The main panel lists seven issues, with the fifth issue, '(17, 26) Refactor this method to reduce its Cognitive Complexity from 23 to the 15 allowed.', highlighted. The right sidebar provides a detailed view of this issue, titled 'Cognitive Complexity of methods should not be too high'. It categorizes the issue as a 'Maintainability' problem and explains that the rule triggers when a function's cognitive complexity exceeds a threshold of 15. The sidebar also includes a 'Parameters' section showing the 'Threshold' is set to 15.

SonarQube for IDE Findings Log Help & Feedback Report - dez. 02, 15:40

Found 7 Issues

VerificadorTransacao.java (7 Issues)

- (1, 8) Rename this package name to match the regular expression `^[a-z_]+(\.[a-z_][a-z0-9_]*)*$`
- (28, 12) Merge this if statement with the enclosing one. (+1 location) 3 years ago java:S1066
- (54, 12) Merge this if statement with the enclosing one. (+1 location) 3 years ago java:S1066
- (15, 13) Add a private constructor to hide the implicit public one. 3 years ago java:S1118
- (17, 26) Refactor this method to reduce its Cognitive Complexity from 23 to the 15 allowed. (+11)
- (22, 12) Replace this use of System.out by a logger. 3 years ago java:S106
- (30, 16) Add a default case to this switch. 3 years ago java:S131

No Security Hotspots to display

No Taint Vulnerabilities to display

Rule Locations

### Cognitive Complexity of methods should not be too high

Adaptability issue | Not focused Maintainability java:S3776 [Learn more](#)

This rule raises an issue when the code cognitive complexity of a function is above a certain threshold.

Why is this an issue? How can I fix it? More Info

Cognitive Complexity is a measure of how hard it is to understand the control flow of a unit of code. Code with high cognitive complexity is hard to read, understand, test, and modify. As a rule of thumb, high cognitive complexity is a sign that the code should be refactored into smaller, easier-to-manage pieces.

Which syntax in code does impact cognitive complexity score?

Here are the core concepts:

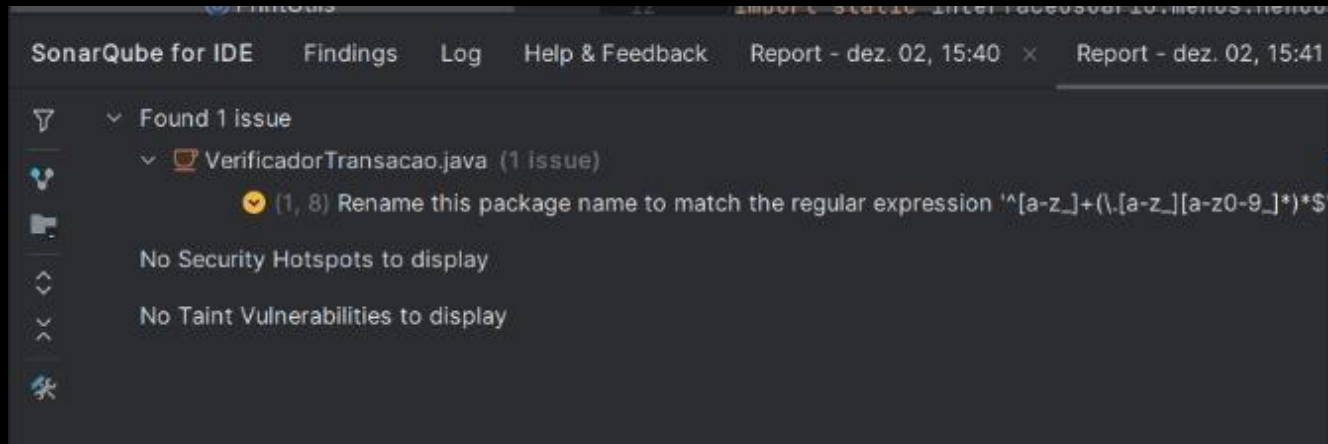
Parameters

Threshold 15

Parameter values can be set in [Rule Settings](#). In connected mode, server side configuration overrides local settings.

# Análise estatística (SonarQube)

- Após análise e implementação de algumas correções ficamos com **13 issues** apontadas nas classes testadas
- Principais apontamentos não ajustados:
  - Renomear variáveis, métodos e pacotes;
    - Evitar quebrar a estrutura de importações de todo o projeto legado









# Técnica de Teste Estrutural

# Teste Estrutural

- Técnica de Teste Caixa-Branca
- Critério Utilizado: **Todas-Arestas** - Cada desvio true/false de if, switch case ou try/catch foi percorrido
- Métrica alvo: > 80% de cobertura

Element ^	Class, %	Method, %	Line, %	Branch, %
▼  interfaceUsuario.verificadores	100% (1/1)	100% (6/6)	90% (37/41)	80% (33/41)
 VerificadorTransacao	100% (1/1)	100% (6/6)	90% (37/41)	80% (33/41)
 VerificadorPix	100% (1/1)	100% (3/3)	92% (13/14)	100% (12/12)
 VerificadorClientes	100% (1/1)	100% (11/11)	92% (47/51)	90% (40/44)

# Técnica de Teste Funcional

# Teste Funcional

- Técnica de Teste Caixa-Preta
- Critério Utilizado:
  - **Particionamento por equivalência (PPE)**
  - **Análise de valor limite (AVL)**
- 29 Teste Funcionais criados:
  - PPE: 14
  - AVL: 7
  - Outros (Fluxo, Segurança, Stress, etc): 7

Exemplo: VerificadorPixFuncionalTest.java

✓ ✕ VerificadorPixFuncionalTest (InterfaceUsuario.verificadores.dados)	4 sec 99 ms
✓ Funcional: Bloqueio de Telefone Inválido (Excedendo caracteres)	3 sec 990 ms
✓ Robustez: Chave Vazia ou Apenas Espaços	12 ms
✓ Robustez: Uso de Emojis e Caracteres Especiais	20 ms
✕ Funcional: Validação de consistência de CPF (Deve falhar se aceitar texto)	36 ms
✓ Segurança: Tentativa de SQL Injection no Email	10 ms
✓ Segurança: Teste de Carga/Stress com String Gigante	15 ms
✓ Funcional: Fluxo de Desistência/Correção pelo Usuário	8 ms
✓ Funcional: Fluxo de Cadastro de Email Válido	8 ms

# Técnica baseada em defeitos (Mutaç o)

# Técnica baseada em defeitos (Mutações)

- Nesta etapa foi provavelmente a maior dificuldade encontrada pelo grupo, pois simplesmente o PIT não encontrada de jeito nenhum a classe a ser mutada
  - A organização do diretório de pastas do projeto bic-poo estava confusa

**Plano de ação:** **Manualmente** realizamos as mutações nas classes testadas e rodamos os testes antes e depois da mutação

# Técnica baseada em defeitos (Mutações)

- Exemplo das mutações geradas manualmente

ID do Mutante	Alteração Realizada	Classificação
MT-01	Alterado if (valor > 0) para if (valor >= 0)	ROR
MT-02	Alterado if (valor <= LIMITE) para if (valor < LIMITE)	ROR
MT-03	Alterado if (A && B) para `if (A    B)`	COR
MT-04	Removido o bloco if (valor > fatura)	SDL
MT-05	Alterado if (!verificarData) para if (verificarData)	COR

## Antes:

- ✓ Cenário Estrutural: Instanciação da Classe
- ✗ Cenário 1.1.1: Deve lançar exceção para entrada não numérica
- ✓ Cenário 1.1.2: Deve lançar ValorInvalido para entrada negativa
- ✗ Cenário 1.2.1 (Falha): Deve lançar exceção para transferência com saldo insuficiente
- ✓ Cenário 1.3.1: Depósito Standard - Deve lançar exceção para valor acima do limite
- ✓ Cenário 1.3.1: Depósito Standard - Deve retornar true para valor válido
- ✓ Cenário 1.3.1-L: Depósito Standard - Deve retornar true para valor no limite
- ✓ Cenário 1.3.1-S: Depósito Standard - Deve lançar exceção se soma de depósitos exceder o limite
- ✓ Cenário 1.3.2: Depósito Premium - Deve lançar exceção para valor acima do limite
- ✓ Cenário 1.3.2-S: Depósito Premium deve falhar se soma exceder limite
- ✓ Cenário 1.3.3: Depósito Diamond - Deve lançar exceção para valor acima do limite
- ✓ Cenário 1.3.3-S: Depósito Diamond deve falhar se soma exceder limite
- ✓ Cenário 1.3.4: Depósito - Deve retornar false para tipo de conta inválido
- ✓ Cenário 1.4.1: Depósito com valor zero deve retornar false
- ✓ Cenário 1.4.2: Depósito com valor negativo deve retornar false
- ✓ Cenário 1.4.3: Transferência com valor zero deve retornar true
- ✓ Cenário 2.1.1: Deve retornar true para pagamento de fatura válido
- ✓ Cenário 2.1.2: Deve lançar ValorInvalido para pagamento de fatura com saldo insuficiente
- ✓ Cenário 2.1.3: Deve lançar ValorInvalido para pagamento maior que a fatura
- ✓ Cenário 2.2.1: Deve retornar false para aumento de fatura válido
- ✓ Cenário 2.2.2: Deve lançar ValorInvalido para aumento de fatura maior que o limite
- ✓ Cenário 2.3.1: Pagar Fatura com valor zero deve retornar true
- ✓ Cenário 2.3.2: Pagar Fatura com valor exato da fatura deve retornar true
- ✓ Cenário 2.3.3: Aumentar Fatura com valor exato do limite restante
- ✓ Cenário 2.4: Valor Fatura com operação desconhecida deve retornar true
- ✓ Cenário 3.1: Deve retornar true para agendamento de transação válido
- ✗ Cenário 3.2: Deve retornar false para agendamento com valor inválido
- ✗ Cenário 3.3: Deve lançar ValorInvalido para agendamento com data inválida
- ✓ Cenário 4.1: Deve retornar true para dados de boleto válidos
- ✓ Cenário 4.2: Deve retornar false para valor do boleto negativo

## Depois:

- ✓ Cenário Estrutural: Instanciação da Classe
- ✗ Cenário 1.1.1: Deve lançar exceção para entrada não numérica
- ✓ Cenário 1.1.2: Deve lançar ValorInvalido para entrada negativa
- ✗ Cenário 1.2.1 (Falha): Deve lançar exceção para transferência com saldo insuficiente
- ✓ Cenário 1.3.1: Depósito Standard - Deve lançar exceção para valor acima do limite
- ✓ Cenário 1.3.1: Depósito Standard - Deve retornar true para valor válido
- ✓ Cenário 1.3.1-L: Depósito Standard - Deve retornar true para valor no limite
- ✗ Cenário 1.3.1-S: Depósito Standard - Deve lançar exceção se soma de depósitos exceder o limite
- ✓ Cenário 1.3.2: Depósito Premium - Deve lançar exceção para valor acima do limite
- ✓ Cenário 1.3.2-S: Depósito Premium deve falhar se soma exceder limite
- ✓ Cenário 1.3.3: Depósito Diamond - Deve lançar exceção para valor acima do limite
- ✓ Cenário 1.3.3-S: Depósito Diamond deve falhar se soma exceder limite
- ✓ Cenário 1.3.4: Depósito - Deve retornar false para tipo de conta inválido
- ✗ Cenário 1.4.1: Depósito com valor zero deve retornar false
- ✓ Cenário 1.4.2: Depósito com valor negativo deve retornar false
- ✓ Cenário 1.4.3: Transferência com valor zero deve retornar true
- ! Cenário 2.1.1: Deve retornar true para pagamento de fatura válido
- ✓ Cenário 2.1.2: Deve lançar ValorInvalido para pagamento de fatura com saldo insuficiente
- ✗ Cenário 2.1.3: Deve lançar ValorInvalido para pagamento maior que a fatura
- ! Cenário 2.2.1: Deve retornar false para aumento de fatura válido
- ✗ Cenário 2.2.2: Deve lançar ValorInvalido para aumento de fatura maior que o limite
- ! Cenário 2.3.1: Pagar Fatura com valor zero deve retornar true
- ! Cenário 2.3.2: Pagar Fatura com valor exato da fatura deve retornar true
- ✓ Cenário 2.3.3: Aumentar Fatura com valor exato do limite restante
- ✓ Cenário 2.4: Valor Fatura com operação desconhecida deve retornar true
- ✓ Cenário 3.1: Deve retornar true para agendamento de transação válido
- ✗ Cenário 3.2: Deve retornar false para agendamento com valor inválido
- ✗ Cenário 3.3: Deve lançar ValorInvalido para agendamento com data inválida
- ✗ Cenário 4.1: Deve retornar true para dados de boleto válidos
- ✓ Cenário 4.2: Deve retornar false para valor do boleto negativo



# Técnica baseada em defeitos (Mutação)

## Conclusão do teste de mutação

- A suíte de testes demonstrou eficácia, mesmo que tendo expressivamente menos mutantes gerados, sendo capaz de detectar alterações sutis na lógica de negócio, como mudanças em operadores relacionais e remoção de blocos de validação.
- O fato de múltiplos testes falharem para um único mutante indica uma redundância positiva na cobertura de testes.

### Manual:

Mutantes Gerados: 5  
Mutantes Mortos: 5  
Mutantes Vivos: 0  
Escore Final: 100% (5/5)

### Automatizado:

Mutantes Gerados pelo PIT: 141  
Mutantes Mortos: 0  
Mutantes Vivos: 141  
Escore Final: 0% (0/141)

Teste de sistema

# Teste de sistema

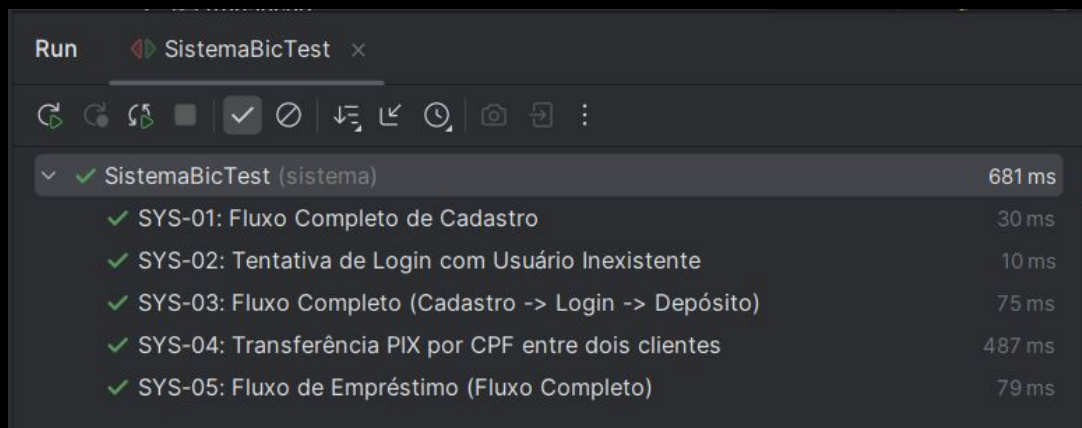
Como nosso sistema é baseado em console e não era possível usar Selenium, então adotamos a biblioteca System Lambda para executar testes de sistema, simulando entradas e capturando saídas como se fosse um usuário real interagindo pelo terminal

O teste envia entradas ( '1' , '0' , valores numéricos, texto, etc) e verifica se o sistema responde corretamente ( 'Operação cancelada' , 'Valor inválido' , etc.)

# Teste de sistema

- No total, 11 cenários de teste de sistema foram criados simulando o comportamento do usuário, sendo todos aprovados

## Exemplo



The screenshot shows a test runner window titled 'Run' with a tab for 'SistemaBicTest'. Below the title bar is a toolbar with various icons. The main area displays a list of test results, all marked with a green checkmark, indicating they passed. The first item is 'SistemaBicTest (sistema)' with a total duration of 681 ms. It is expanded to show five sub-items: 'SYS-01: Fluxo Completo de Cadastro' (30 ms), 'SYS-02: Tentativa de Login com Usuário Inexistente' (10 ms), 'SYS-03: Fluxo Completo (Cadastro -> Login -> Depósito)' (75 ms), 'SYS-04: Transferência PIX por CPF entre dois clientes' (487 ms), and 'SYS-05: Fluxo de Empréstimo (Fluxo Completo)' (79 ms).

Run	SistemaBicTest	x
✓ SistemaBicTest (sistema)		681 ms
✓ SYS-01: Fluxo Completo de Cadastro		30 ms
✓ SYS-02: Tentativa de Login com Usuário Inexistente		10 ms
✓ SYS-03: Fluxo Completo (Cadastro -> Login -> Depósito)		75 ms
✓ SYS-04: Transferência PIX por CPF entre dois clientes		487 ms
✓ SYS-05: Fluxo de Empréstimo (Fluxo Completo)		79 ms

# Teste de sistema

## Exemplo:

- SYS-04:

Cadastrar Pagador → Cadastrar Recebedor  
→ Logar Pagador e Depositar → Transferir  
via Chave PIX → Consultar Saldo

## Validação:

Verificar se o comprovante foi  
impresso e se o saldo foi debitado

```
@Test  @ Daniel Fontoura *
@DisplayName("SYS-04: Transferência PIX por CPF entre dois clientes")
void fluxoTransferenciaPix() throws Exception {
    String textoConsole = SystemLambda.tapSystemOut(() -> {
        SystemLambda.withTextFromSystemIn(
            // --- 1. CADASTRO DO PAGADOR ---
            ...lines: "2", "1", "24000000", "10", "Casa",
                    "Pagador", "pagador@email.com", "21999999999", "30",
                    "1111111111", "123", "5000", "0", "CardPagador", "0",
            // --- 2. CADASTRO DO RECEBEDOR (CPF: 2222222222) ---
                    "2", "1", "24000000", "20", "Apto",
                    "Recebedor", "recebedor@email.com", "21888888888", "30",
                    "2222222222", "123", "5000", "0", "CardRecebedor", "0",
            // --- 3. LOGIN DO PAGADOR E DEPÓSITO ---
                    "1", "1111111111", "123",
                    "5", "1000", // Depositar 1000 para ter saldo
            // --- 4. TRANSFERÊNCIA VIA CPF ---
                    "3", // [3] Transferir
                    "200", // Valor
                    "identificacao", // Tipo Chave: identificacao (CPF/CNPJ)
                    "2222222222", // Chave: 0 CPF do recebedor
                    "1", // Confirmação do PIX: [1] Sim, está correta
                    "1", // [1] Verificar Saldo

                    "0", "0"
        ).execute() -> {
            MenuUsuario.TECLADO = new java.util.Scanner(System.in);
            MenuUsuario.iniciar();
        });
    });
};
```

# Atributos de Qualidade - ISO 25010

Atributo ISO 25010	Nota (1-5)
Adequação Funcional	4
Confiabilidade	3
Usabilidade	2
Eficiência de Desempenho	4
Segurança	3
Manutenibilidade	4
Compatibilidade	4
Portabilidade	5

Considerar escala:

1 - Ruim

2 - Fraco

3 - Razoável / aceitável

4 - Bom

5 - Excelente

Obrigado!