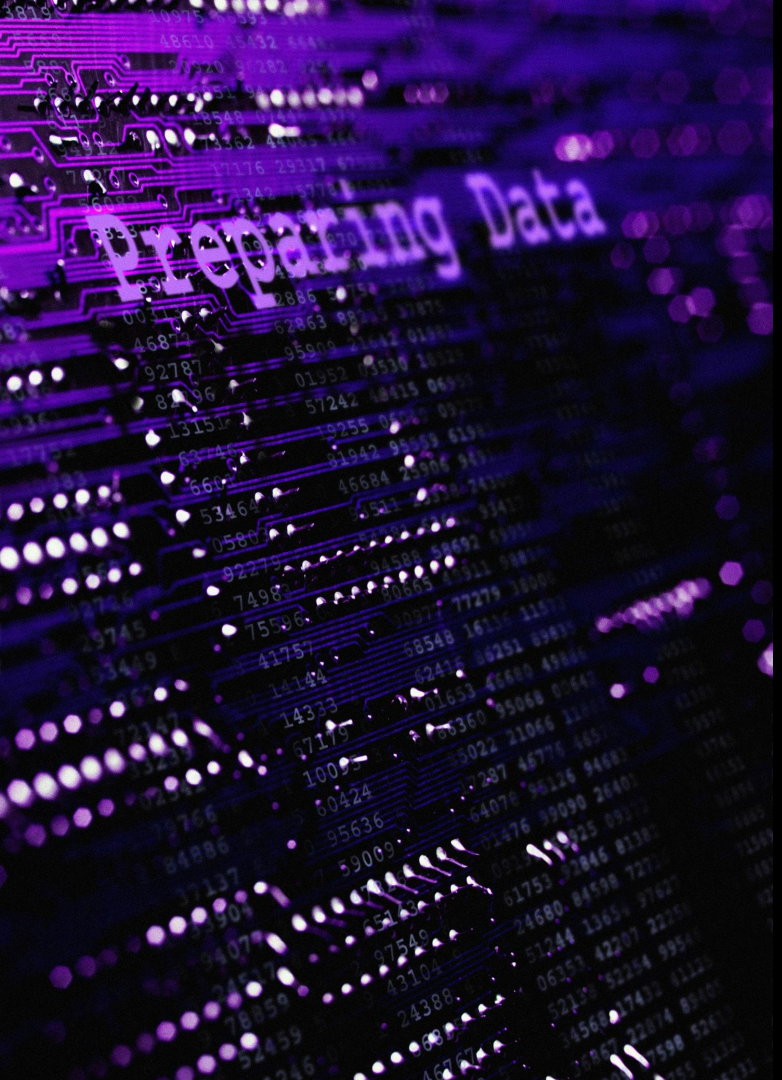


Qualidade e Teste de Software

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 fisica
[2] - Pessoa juridica
>
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

mock

Testes Unitários

- Complexidade ciclomática >= 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)

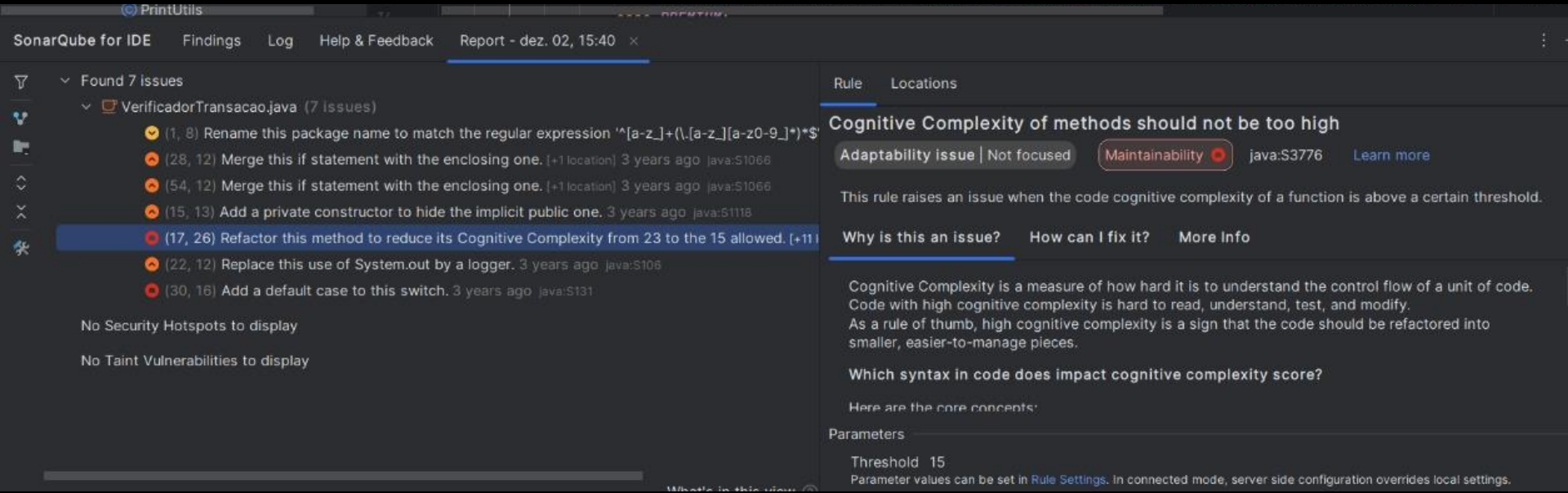
Análise estatística

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.

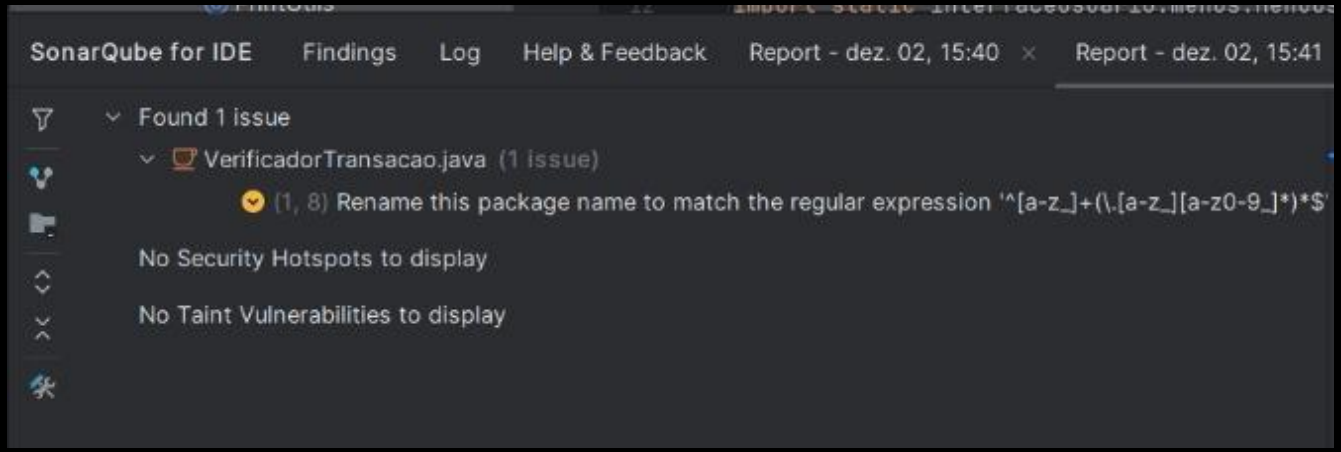
Análise estatística (SonarQube)

Exemplo com a classe VerificadorTransacao.java



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)

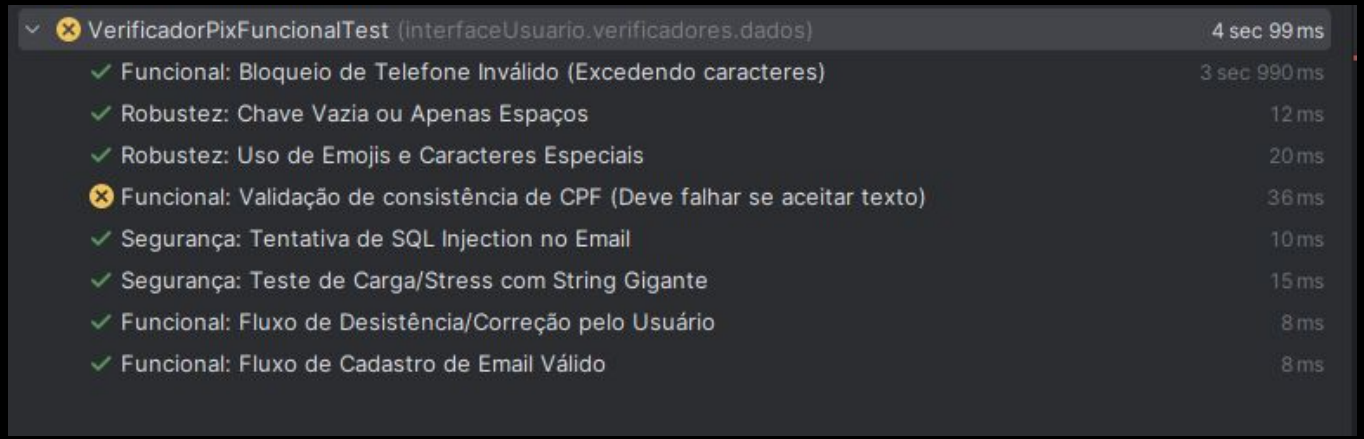
Técnica de Teste Funcional

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



Técnica baseada em defeitos (Mutação)

Técnica baseada em defeitos (Mutação)

- 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

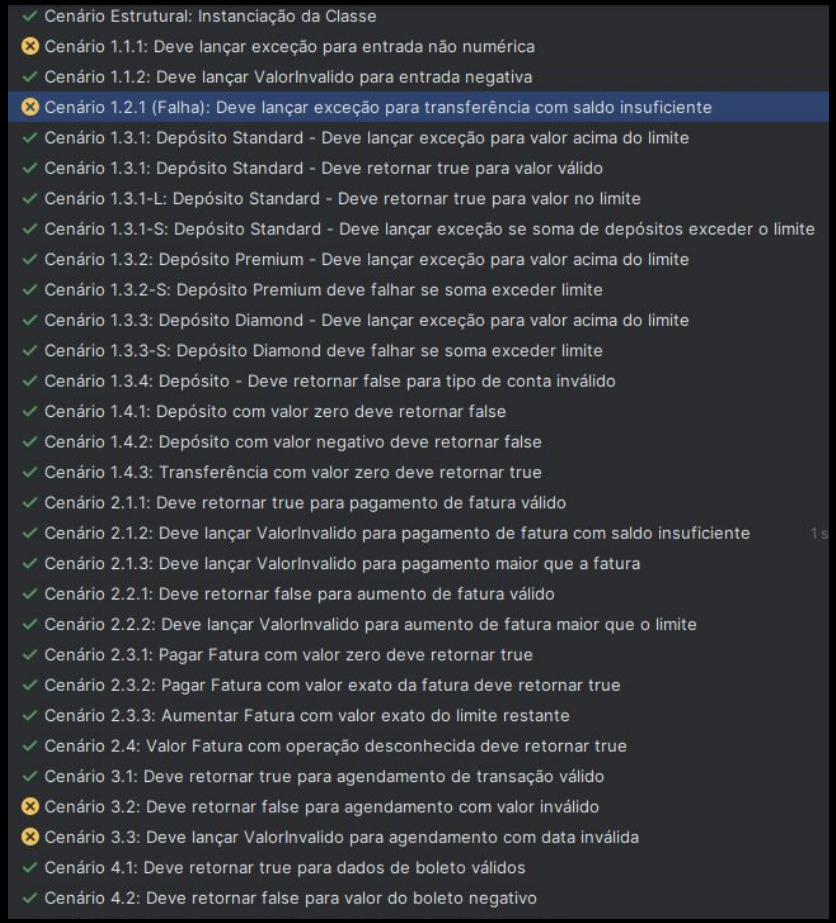
Plano de ação: **Manualmente** realizamos as mutações nas classes testadas e rodamos os testes antes e depois da mutação

Técnica baseada em defeitos (Mutação)

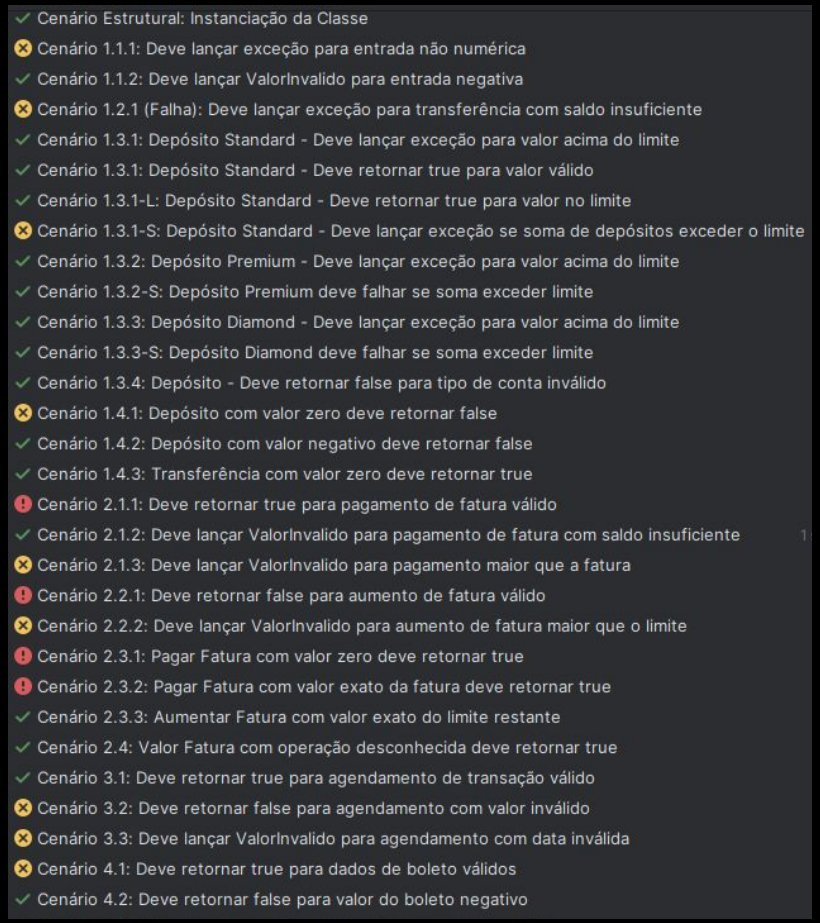
- Exemplo das mutações geradas manualmente

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:



Depois:



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

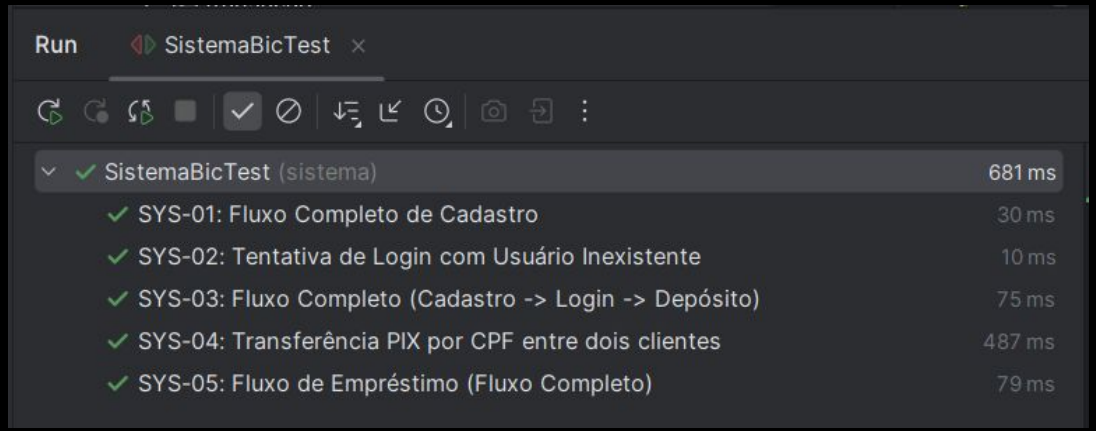
O teste envia entradas (‘1’, ‘0’, valores numéricos, texto, etc) e verifica se o sistema responde corretamente (‘Operação cancelada’, ‘Valor inválido’, etc.)

https://github.com/stefanbirkner/system-lambda

Teste de sistema

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

Exemplo



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",
                "11111111111", "123", "5000", "0", "CardPagador", "0",
                // --- 2. CADASTRO DO RECEBEDOR (CPF: 22222222222) ---
                "2", "1", "24000000", "20", "Apto",
                "Recebedor", "recebedor@email.com", "21888888888", "30",
                "22222222222", "123", "5000", "0", "CardRecebedor", "0",
                // --- 3. LOGIN DO PAGADOR E DEPÓSITO ---
                "1", "11111111111", "123",
                "5", "1000", // Depositar 1000 para ter saldo
                // --- 4. TRANSFERÊNCIA VIA CPF ---
                "3",                    // [3] Transferir
                "200",                  // Valor
                "identificacao",        // Tipo Chave: identificacao (CPF/CNPJ)
                "22222222222",          // Chave: O 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!