UNIVERSIDADE FEDERAL DO PERNAMBUCO - UFPE
CENTRO DE INFORMÁTICA - CIn

Relatório sobre o Projeto Piloto de Inspeção do Código-Fonte da Urna Eletrônica na UFPE

PROF.ANDRÉ LUÍS DE MEDEIROS SANTOS
PROF. BRENO MIRANDA
PROF. ROBERTO SOUTO MAIOR DE BARROS
ELVYS SOARES
EMERSON PAULO SOARES DE SOUZA
DAVI SIMÕES FREITAS

COORDENADOR: PROF. ANDRÉ LUÍS DE MEDEIROS SANTOS

Recife, 22 de julho de 2022

Resumo

Este relatório descreve o trabalho realizado conforme o Plano de Trabalho referente ao Termo de Adesão para Inspeção do Código Fonte assinado entre a UFPE e o TSE (SEI 2019.00.000003122-3).

Este projeto é particularmente importante por se tratar de uma iniciativa pioneira de abertura do código-fonte da urna fora das instalações do TSE. Este acesso permitiu que os pesquisadores participantes tivessem um prazo maior para o estudo e análise do código, além de um acesso irrestrito, sem precedentes, ao código-fonte dos sistemas utilizados na urna eletrônica e nas eleições.

A execução do plano de trabalho compreendeu a inspeção de código em geral dos sistemas utilizados na Urna Eletrônica, bem como uma análise de qualidade dos testes automatizados e manuais. Esta análise dos testes envolveram a análise de cobertura dos testes e também a avaliação da qualidade dos testes automáticos e dos testes manuais a partir da abordagem de *test smells*.

Nenhum dos trabalhos realizados identificou problemas que comprometam o funcionamento dos softwares analisados ou que demandem correções ou alterações na versão do software prevista para uso no ciclo eleitoral corrente.

Apesar de não haver necessidade de alterações urgentes, as sugestões apresentadas neste documento e nas apresentações realizadas visam o aumento da qualidade dos testes automatizados, da especificação dos testes manuais e do código fonte. Essas melhorias, uma vez realizadas, possibilitam testes ainda mais robustos, legíveis e fáceis de manter. A equipe do TSE poderá avaliar as sugestões e o melhor momento para que sejam adotadas.

Acreditamos que este trabalho, através das análises e sugestões realizadas, contribui para a avaliação e comprovação da qualidade do software usado nas eleições no Brasil, para termos eleições rápidas, seguras e confiáveis.

Índice

Introdução

Este relatório apresenta o trabalho realizado conforme o Plano de Trabalho referente ao Termo de Adesão para Inspeção do Código Fonte assinado entre a UFPE e o TSE (SEI 2019.00.000003122-3). A execução do plano de trabalho compreendeu a inspeção de código em geral dos sistemas utilizados na Urna Eletrônica, bem como uma análise de qualidade dos testes, automatizados e manuais. Esta análise dos testes envolveram a análise de cobertura dos testes e também a avaliação da qualidade dos testes automáticos e manuais a partir da abordagem de *test smells*.

O plano foi executado sob o seguinte cronograma de atividades:

a) 29/03 a 27/04/22 – SEVIN - Disponibilização para UFPE do código fonte e ambiente para compilação e execução dos testes unitários, com o respectivo suporte pela equipe do TSE;

b) 22/04/22 - SEVIN - Disponibilização para UFPE de amostras de especificações de testes manuais, de relatório de testes exploratórios e testes de sanidade;

c) 26/05/22 – UFPE - Apresentação da UFPE dos resultados iniciais obtidos, bem como detalhamento da abordagem de trabalho adotada;

d) 03/06/22 - SEVIN - Disponibilização para UFPE do código fonte mais atualizado;

e) Reuniões periódicas de acompanhamento, bem como suporte técnico sob demanda;

O trabalho foi desenvolvido nas instalações do Centro de Informática da UFPE e foi divido em três atividades:

a) avaliação da qualidade dos testes presentes no código e da especificação dos testes manuais, a partir da abordagem de test smells, realizada por Elvys Soares e Emerson Souza, sob a supervisão do prof. André Santos;

b) avaliação da cobertura dos testes presentes no código, visando a proposta de formas de aumentar a cobertura dos testes no código, realizada por Davi Freitas, sob a supervisão do prof. Breno Miranda; e

c) avaliação da qualidade do código fonte, através da inspeção do código, realizada pelo prof. Roberto Barros;

Nas seções a seguir são descritas essas atividades e as principais observações realizadas.

Análise de Test Smells

No contexto de testes automatizados de software, muitas vezes é possível encontrar oportunidades de melhoria no projeto e implementação do código de teste, de forma análoga ao que acontece para o código-fonte da aplicação testada. Estas oportunidades de melhoria são conhecidas por *tests smells,* também de forma análoga aos *code smells* – estes voltados para o código da aplicação. Tais oportunidades são identificadas através de sintomas que podem representar comportamentos incorretos na execução dos testes, tais como falsos positivos e/ou falsos negativos, comprometendo assim a qualidade do software devido à capacidade limitada de detecção de defeitos por parte dos testes (SOARES, 2022).

Embora o conceito de *Tests Smells* não seja recente (VAN DEURSEN, 2001), estudos anteriores demonstraram que são frequentes na prática, tanto em projetos de código aberto como na indústria, se aplicando não somente para testes automatizados, como também para testes manuais, e que sua presença é danosa para as atividades de manutenção e compreensão de código (BAVOTA, 2015).

Este projeto torna-se pioneiro ao analisar os testes da Urna em busca de *smells* em ambos os contextos. Ressalta-se que, até então, não há pesquisas relatadas que investiguem *test smells* nas tecnologias escolhidas para implementar o ambiente de testes da Urna no contexto automatizado. Quanto aos *test smells* em testes manuais, o estado da produção acadêmica atual ainda é incipiente (HAUPTMANN, 2013).

Desta forma, este trabalho tem a dupla motivação de contribuir para a garantia da qualidade de um dos softwares mais importantes para a democracia do Brasil e de contribuir para a pesquisa de *test smells* em tecnologias até então não exploradas neste viés.

Possui como objetivo geral:

- Inspecionar os testes manuais e automatizados do ambiente de software da Urna, com o apoio do TSE, na busca por *test smells*.

Como objetivos específicos:

1. Estudar as tecnologias usadas no projeto da Urna Eletrônica;
2. Realizar amostragem das funções do Google Test nos testes automatizados;
3. Definir a estratégia de análise das funções do Google Test;

4. Realizar amostragem nos testes manuais escritos em Linguagem Natural.

Atividades Desenvolvidas

As atividades desenvolvidas no presente projeto estão divididas em 4 (quatro) etapas:

A primeira etapa concentrou-se na instalação do projeto de código fonte da urna eletrônica, em um computador exclusivo, com Sistema Operacional Linux, versão Ubuntu 20.04, sem acesso à internet, exceto em momentos de atualização de bibliotecas necessárias para a correta compilação, disponibilizado pelo Centro de Informática da UFPE, em sala segura com câmeras e acesso restrito a pessoas autorizadas, participantes do projeto.

Na segunda etapa, após o código devidamente **atualizado e compilado**, os pesquisadores trabalharam na amostragem do código fonte, quantificando as funções dos testes desenvolvidos usando Google Testes no projeto VOTA. As funções vistas no código foram: ASSERT_NE, EXPECT_CALL, EXPECT_CALL, EXPECT_EQ, EXPECT_EQ_THROW, EXPECT_FALSE, EXCPECT, GE, EXPECT_GT, EXPECT_LT, EXPECT_NE, EXPECT_NO_THROW, EXPECT_THROW, EXPECT_TRUE, FAIL. Suas respectivas quantidades estão disponibilizadas no Quadro 01 abaixo.

Quadro 01: Quantidade de funções GoogleTests encontradas no projeto VOTA

Projetos/uenux/test/app/vota	
Propositions	**Ocorrências**
ASSERT_NE	15
EXPECT_CALL	31
EXPECT_EQ	2635
EXPECT_EQ_THROW	38
EXPECT_FALSE	672
EXPECT_GE	1
EXPECT_GT	3
EXPECT_LE	1
EXPECT_LT	2
EXPECT_NE	131
EXPECT_NO_THROW	88
EXPECT_THROW	10

EXPECT_TRUE	646
FAIL	4

Fonte: Elaborado pelos autores, 2022.

É preciso esclarecer que a escolha pelo projeto Vota deu-se por ser o projeto central responsável pelo registro dos votos. A soma dos valores das quantidades das funções é de 4277. Sendo considerado um número bastante expressivo para **analisar e avaliar** se a maneira que o código está escrito está de acordo com a literatura do Google Test.

A terceira etapa deste projeto, chamada de Amostragem, caracteriza a primeira frente de trabalho e refere-se a identificação de Smells nos testes automatizados, identificados anteriormente, fazendo o somatório da quantidade e a nomeação destes *test smells* com base na literatura disponível até o momento. Foram identificados *test smells,* tais como ***Disorder***, ***Redundant Assert***, ***Bad naming***, ***Test body is somewhere else***, ***No assertions***, ***Conditional Test Execution***, ***Assertion Roulette***, ***Not Idempont***, ***Magic Values***, ***Asserting Pre-condition and Invariants***, ***Test Code Duplication***, ***Eager Test***.

No Quadro 02, estão descritos a classe onde o teste está inserido, o nome do teste analisado, observações de projeto e implementação que caracterizam os *test smells* e, por fim, o nome dos *test smells* identificados.

Quadro 02: *Test smells* identificados no código fonte do projeto Vota

Classe	Teste	Observações	Test Smell
operador/aguardaoperador		Elementos desordenados (declaração da fixture, implementação e testes)	Disorder
	testCriacao	comparando dois valores calculados	Redundant Assert
		Vários métodos com conteúdo duplicado: testaOficialPreparaTeste e testaEleitorPreparaTeste, e testaOficialStartState e testaEleitorStartState	
	testaOficial	O nome do teste não indica o propósito	Bad naming
		Chama dois métodos que não se sabe se testam algo	Test body is somewhere else
		não faz verificação alguma	No Assertions
	testaOficialPreparaTeste	Chama método externo	Test body is somewhere else
		não faz verificação alguma	No Assertions
	testaOficialStartState	Chama método externo	Test body is

			somewhere else
	testaOficialProcessMsgETick	Chama método externo	Test body is somewhere else
		não faz verificação alguma	No Assertions
	testaOficialProcessInput	Chama método externo	Test body is somewhere else
		não faz verificação alguma	No Assertions
	testaEleitor	Chama método externo	Test body is somewhere else
		não faz verificação alguma	No Assertions
	testaEleitorPreparaTeste	Chama método externo	Test body is somewhere else
		não faz verificação alguma	No Assertions
	testaEleitorStartState	Chama método externo	Test body is somewhere else
	testaEleitorStartStateAudio	Chama método externo	Test body is somewhere else
	startState	Possui condicionais, inclusive aninhados. Deveriam ser testes/métodos separados	Conditional Test Execution
		Testes (asserções com ou sem falha) sem mensagem explicando o erro	Assertion Roulette
		Precondição testada com ASSERT_NE, quando deveria ser utilizada a macro GTEST_SKIP() para evitar falsos negativos	Asserting Pre-condition and Invariants
		O método altera o estado do sistema para habilitar o audio mesmo que o audio não esteja no escopo do teste	Not Idempotent
		Há valores textuais e inteiros sem explicação do seu significado	Magic Values
	testaEleitor_semNomeSocial	Há valores textuais e inteiros sem explicação do seu significado	Magic Values
		Precondição testada com ASSERT_NE, quando deveria ser utilizada a macro GTEST_SKIP() para evitar falsos negativos	Asserting Pre-condition and Invariants
	testaEleitor_comNomeSocial	O eleitor testado não tem nome social cadastrado no teste, porém o nome do teste é com Nome Social	
		Há valores textuais e inteiros sem explicação do seu significado	Magic Values

		Precondição testada com ASSERT_NE, quando deveria ser utilizada a macro GTEST_SKIP() para evitar falsos negativos	Asserting Pre-condition and Invariants
		Duplicação do teste anterior	Test Code Duplication
	testaNavegacaoSemMarcaPULocalBiometrico	Há valores textuais e inteiros sem explicação do seu significado	Magic Values
		Precondição testada com ASSERT_NE, quando deveria ser utilizada a macro GTEST_SKIP() para evitar falsos negativos	Asserting Pre-condition and Invariants
		Testes (asserções com ou sem falha) sem mensagem explicando o erro	Assertion Roulette
		comparando dois valores calculados	Redundant Assert
		Testando dois objetos diferentes no mesmo método de teste	Eager Test
	testaNavegacaoSemMarcaPULocalNaoBiometrico	Há valores textuais e inteiros sem explicação do seu significado	Magic Values
		Precondição testada com ASSERT_NE, quando deveria ser utilizada a macro GTEST_SKIP() para evitar falsos negativos	Asserting Pre-condition and Invariants
		Testes (asserções com ou sem falha) sem mensagem explicando o erro	Assertion Roulette
		comparando dois valores calculados	Redundant Assert
		Testando dois objetos diferentes no mesmo método de teste	Eager Test
		Duplicação do teste anterior	Test Code Duplication
	testaNavegacaoSemMarcaPULocalNaoBiometricoComAudio	Há valores textuais e inteiros sem explicação do seu significado	Magic Values
		Precondição testada com ASSERT_NE, quando deveria ser utilizada a macro GTEST_SKIP() para evitar falsos negativos	Asserting Pre-condition and Invariants
		Testes (asserções com ou sem falha) sem mensagem explicando o erro	Assertion Roulette
		comparando dois valores calculados	Redundant Assert
		Testando dois objetos diferentes no	Eager Test

		mesmo método de teste	
		Duplicação do teste anterior	Test Code Duplication
	testaNavegacaoComMarcaPULocalBiometrico	Há valores textuais e inteiros sem explicação do seu significado	Magic Values
		Precondição testada com ASSERT_NE, quando deveria ser utilizada a macro GTEST_SKIP() para evitar falsos negativos	Asserting Pre-condition and Invariants
		Testes (asserções com ou sem falha) sem mensagem explicando o erro	Assertion Roulette
		comparando dois valores calculados	Redundant Assert
		Testando dois objetos diferentes no mesmo método de teste	Eager Test
		Duplicação do teste anterior	Test Code Duplication
	testaNavegacaoComMarcaPULocalBiometricoEleitorSemBiometria	Há valores textuais e inteiros sem explicação do seu significado	Magic Values
		Precondição testada com ASSERT_NE, quando deveria ser utilizada a macro GTEST_SKIP() para evitar falsos negativos	Asserting Pre-condition and Invariants
		Testes (asserções com ou sem falha) sem mensagem explicando o erro	Assertion Roulette
		comparando dois valores calculados	Redundant Assert
		Testando dois objetos diferentes no mesmo método de teste	Eager Test
		Duplicação do teste anterior	Test Code Duplication
	testaNavegacaoComMarcaPULocalNaoBiometrico	Há valores textuais e inteiros sem explicação do seu significado	Magic Values
		Precondição testada com ASSERT_NE, quando deveria ser utilizada a macro GTEST_SKIP() para evitar falsos negativos	Asserting Pre-condition and Invariants
		Testes (asserções com ou sem falha) sem mensagem explicando o erro	Assertion Roulette
		comparando dois valores calculados	Redundant Assert
		Testando dois objetos diferentes no mesmo método de teste	Eager Test

		Duplicação do teste anterior	Test Code Duplication
	testaEleitorProcessInputModoDemo	Precondição testada com ASSERT_NE, quando deveria ser utilizada a macro GTEST_SKIP() para evitar falsos negativos	Asserting Pre-condition and Invariants
		Testes (asserções com ou sem falha) sem mensagem explicando o erro	Assertion Roulette
		Testando dois objetos diferentes no mesmo método de teste	Eager Test
	método processInput()	Há valores textuais e inteiros sem explicação do seu significado	Magic Values
		Precondição testada com ASSERT_NE, quando deveria ser utilizada a macro GTEST_SKIP() para evitar falsos negativos	Asserting Pre-condition and Invariants
		Testes (asserções com ou sem falha) sem mensagem explicando o erro	Assertion Roulette
		Testando dois objetos diferentes no mesmo método de teste	Eager Test
		comparando dois valores calculados	Redundant Assert

Classe	**Teste**	**Observações**	**Test Smell**
operador/reconhecimentobiometrico	nomesReservados	const bool nomesReservados = (name== "." || name == "..") - Valores indefinidos.	Magic Values
	EXPECT_CALL(matcher, GetScore).Times(1).WillOnce(testing::Return(38)	EXPECT_CALL(matcher, Match) .Times(**3**) / EXPECT_CALL(matcher, GetScore).Times(**1**).WillOnce(testing::Return(**38**) O que seriam esses valores?	
	ControlaReconhecimento(**digitalCapturadaMesarioSecaoSemBiometriaCadastrada**	EXPECT_EQ(true, (CControlaReconhecimento::GetInst().GetTituloMesario() == "000000040132"))	Magic Values
	digitalCapturadaMesarioBiometriaInexistente(scanner.GetImageSize(). '**x**')		Magic Values
	digitalCapturadaMesarioBiometriaInexistente(scanner.GetImageSize(). '**z**')		Magic Values

	EXPECT_CALL(matcher, Match) .Times(6) .WillOnce(testing::Return **(false)**)5x .WillOnce(testing::Return **(true)**);	Por quê o parâmetro muda de false para true na 6ª e última chamada?	?
	EXPECT_CALL(matcher, GetScore) .Times(4) .WillOnce(testing::Return (**19**))(3x) .WillOnce(testing::Return (**35**))	O que significam os valores 19 e 35, e porque deve mudar na última chamada?	
	EXPECT_EQ(true, (CControlaReconhecimento::GetInst().GetTituloMesario() == "**000000040132**"))		Magic Values
	EXPECT_EQ(true, (CControlaReconhecimento::GetInst().GetTituloMesario() == "**014296172097**"))		
	ueint32 qtdArquivosBiometria = QtdArquivosDiretorioBiometrias();		

	TESTE_F(TesteCRegistraDigitalOperador, testeProcessInput){
	auto state = &RegistraDigitalOperador::GetInst();
	state->StartState();
	setInputMT("**C**");
	state->ProcessInput();
	EXPECT_FALSE(state->NeedChangeState());
	
	setInputMT("D")
	state->ProcessInput();
	EXPECT_TRUE(state->NeedChangeState());
	}		

	const std::vector<uebyte> digitalCapturadaMesarioForaSecao(scanner.GetImageSize(), '**k**');
	const std::vector<uebyte> digitalCapturadaMesarioForaSecao(scanner.GetImageSize(), '**l**');
	const std::vector<uebyte> digitalCapturadaMesarioForaSecao(scanner.GetImageSize(), '**x**');		

Fonte: Elaborado pelos autores, 2022.

Os *test smells* identificados nos testes serão explicados com base na literatura na apresentação final deste projeto.

A quarta e última etapa deste projeto desenvolvida até o momento refere-se à segunda frente de trabalho, sobre a identificação de *test smells* em testes manuais da Urna Eletrônica. Nesta iniciativa, foram identificados os seguintes *test smells*, vistos com mais detalhes no Quadro 03: ***Random Data, Tacit Knowledge, Misplaced Step, Conditional Result Verification, Magic Value, Eager Step, Calculating expected results on the fly***.

Quadro 03: *Test smells* identificados em testes manuais

Caso de Teste SEVIN-986: VOTA - Encerramento - UESEÇÃO - Validar com sucesso os arquivos de saida do VOTA - BU com 01 voto. [Versão : 2]
As pré-condições não especificam que sessão, especificamente, utilizar. Os cálculos realizados são todos relativos à sessão aleatoriamente escolhida. **(Random Data - new)** No passo 1, não foi explicado o que significa votação nominal **(Tacit Knowledge - new)** No passo 2, o resultado do passo 2 é mais um passo **(Misplaced Step)** No passo 3, existe uma lógica de branching nos resultados esperados. É exibido como verificar em sessão normal, mas não como verificar em qualquer outro tipo de sessão. **(Conditional Result Verification - new)**
Caso de Teste SEVIN-3834: Verificar a integridade da assinatura dos dados de entrada alterados [Versão : 6]
Nas pré-condições, é especificado o horário de 8h para a urna, sem explicação do motivo **(Magic Value - new)**

O passo 2, na verdade, é uma coleção de passos verificados somente ao final **(Eager Step)**
O passo 3 assume que a urna está desligada, sem ter comandado seu desligamento em passos anteriores **(Tacit Knowledge - new)**

Caso de Teste SEVIN-985: VOTA-SEÇÃOBIMÉTRICA - Validar os arquivos de saída, considerando um dos mod. 2009, 2010, 2011,2013. [Versão : 1]

As pré-condições não especificam que sessão, especificamente, utilizar. Os cálculos realizados são todos relativos à sessão aleatoriamente escolhida. **(Random Data - new)**

As verificações realizadas nos resultados esperados do passo 3 não são previstas pelo caso de teste e devem ser computadas em tempo de execução, de acordo com os dados escolhidos aleatoriamente **(calculating expected results on the fly - new)**

Caso de Teste SEVIN-988: RED - Encerrar BU para o tot. UESeção - Validar com sucesso os arquivos de saída. BU 00 votos. [Versão : 1]

Nas pré-condições, é especificado o horário de 16:10 para a urna, sem explicação do motivo **(Magic Value - new)**
As pré-condições não especificam que sessão, especificamente, utilizar. Os cálculos realizados são todos relativos à sessão aleatoriamente escolhida. **(Random Data - new)**
No passo 1, há uma espera indeterminada **(Undefined Wait)**

Fonte: Elaborado pelos autores, 2022.

Após alguns resultados das duas iniciativas de trabalhos, é possível observar que há um número expressivo de testes, tanto manuais quanto automatizados e que a ausência de um padrão na escrita destes, torna o desafio para sua análise ainda maior. Os testes manuais escritos em Linguagem Natural são bem extensos, e quem não é da área eleitoral pode sentir dificuldades no entendimento e consequentemente na validação, devido à falta de clareza nos resultados esperados. O mesmo aplica-se para a maioria dos testes automatizados, observados até o momento.

Resultados

Os resultados obtidos até o presente momento mostram o quanto o sistema da Urna Eletrônica é robusto, bem planejado e testado. No entanto, esforços em busca de uma maior padronização na nomenclatura e da uniformização na escrita dos testes, independentemente de sua modalidade (manual ou automatizado), são recomendáveis. Desta forma, e como já apontado na literatura relacionada, facilitam-se a manutenção e o entendimento dos testes por parte dos profissionais envolvidos no desenvolvimento e na execução destes.

Contribuições da Proposta

A proposta deste trabalho é identificar Smells no código fonte de testes automatizados e na documentação que define os testes manuais da Urna Eletrônica. Com a intenção de sugerir melhorias no desenvolvimento e padronização do código e dos testes manuais, tornando-os mais objetivos, claros e autoexplicativos. Até o presente momento, foram propostas melhorias na padronização do código-fonte, além do desenvolvimento de um estudo para geração automática de testes, visando aumentar a cobertura, e análises preliminares da documentação dos testes manuais, do código de testes automatizados e da cobertura dos testes.

Ressalta-se que, apesar da possibilidade de encontrar alguma situação crítica no código da aplicação ou dos testes durante este trabalho, isto não ocorreu. Tal fato somente reforça a robustez do software do ambiente da Urna, já publicamente demonstrada e notoriamente reconhecida.

Neste sentido, as contribuições desta proposta se concentram em melhorias para a qualidade, legibilidade e manutenção do código. Estes atributos são desejáveis para a manutenção de sua robustez a curto e longo prazo, bem como para minimizar a inserção de falhas.

Trabalhos Futuros

Espera-se, como trabalhos futuros, a ampliação do tempo de análise disponível e o número de pesquisadores e especialistas envolvidos, bem como as técnicas e ferramentas disponíveis, contribuindo para o desenvolvimento de soluções ainda inéditas, aplicadas ao contexto do ambiente da urna.

Com a continuidade do estudo da especificação dos testes manuais pode-se buscar uma padronização em sua descrição que viabilize a criação de uma

ferramenta que automatize sua análise. O desenvolvimento de uma ferramenta de análise automática poderá otimizar o trabalho manual feito inicialmente e contribuir para a melhoria de sua qualidade, reduzindo a quantidade de testes manuais a partir de uma proposta de arquitetura para execução de testes automatizados na própria urna, bem como tornando mais eficiente a especificação de testes manuais.

Análise de Cobertura e Geração Automática de Testes

Na primeira etapa do projeto, optamos por realizar uma análise em largura, explorando a viabilidade de aplicar diferentes estratégias de teste sem, entretanto, realizar uma análise aprofundada das estratégias candidatas. Nosso grupo ficou encarregado de investigar estratégias de teste dinâmicas, ou seja, estratégias que assumem a execução do software sendo testado. Concentramos os nossos esforços em três frentes principais: i) cobertura de testes; ii) geração automática de casos de teste; e iii) teste de mutação. Estas três frentes estão diretamente relacionadas e focam na melhoria da efetividade da suíte de testes.

Começamos com uma análise da cobertura de testes [5] porque tal análise serve como termômetro: se os índices de cobertura já são altos para todos os módulos, investir em técnicas de geração automática de casos de teste ou em teste de mutação teriam um efeito limitado.

A análise preliminar de teste de cobertura do módulo *libecourna*, por exemplo, demonstrou que, em geral, muitos módulos apresentam altos índices de cobertura (~90% dos módulos listados na Tabela 1 possuem cobertura superior a 75% das linhas de código). Apesar disso, foi possível identificar a presença de módulos que não alcançam 100% de cobertura do código de produção, o que representa uma oportunidade de melhoria da suíte de testes (7 dos módulos listados na Tabela 1 possuem cobertura de linhas inferior a 75%). Cabe destacar que definir 100% de cobertura como meta, o que é impraticável em muitos casos, não significa, necessariamente, garantia de qualidade dos testes gerados — se os casos de teste são criados apenas para aumentar a taxa de cobertura sem o objetivo de exercitar áreas propensas a erros e possivelmente revelar falhas, alcançar altos níveis de cobertura não ajuda a ganhar confiança sobre o nível de qualidade do software que está sendo testado. Por outro lado, a existência de trechos de código que não são cobertos por nenhum caso de teste podem representar um risco, já que eventuais falhas relacionadas com aqueles trechos não seriam reveladas durante a fase de testes.

Directory	Line Coverage	Directory	Line Coverage
src/api	90,00%	src/app/dados/asn/rdv	99,30%
src/api/compression	85,90%	src/app/dados/asn/respostaversaopacotes	77,80%
src/api/io	78,90%	src/app/dados/asn/resultadournacadastro	74,90%
src/api/io/asn	91,20%	src/app/dados/asn/scueconf	100,00%

src/api/log	93,90%	src/app/dados/asn/secoes	96,00%
src/api/pattern	90,60%	src/app/dados/asn/solicitacaoversaopacotes	100,00%
src/api/security	90,10%	src/app/dados/asn/urna	74,50%
src/api/security/asn1	92,50%	src/app/dados/boletimurna	93,90%
src/api/security/cepesc	79,80%	src/app/dados/candidatos	99,50%
src/api/security/certchain	84,80%	src/app/dados/certificacao	100,00%
src/api/security/rsa	72,90%	src/app/dados/coligacoes	100,00%
src/api/sql	100,00%	src/app/dados/correspondencia	89,30%
src/api/sql/sqlite	74,70%	src/app/dados/correspondenciaurna	99,20%
src/api/util	98,20%	src/app/dados/eleitores	92,90%
src/app/dados	90,30%	src/app/dados/flash	100,00%
src/app/dados/asn	68,20%	src/app/dados/fotoscandidatos	82,20%
src/app/dados/asn/boletimurna	98,30%	src/app/dados/impedidos	79,40%
src/app/dados/asn/candidatos	66,50%	src/app/dados/local	100,00%
src/app/dados/asn/certificacao	100,00%	src/app/dados/midias	91,20%
src/app/dados/asn/coligacoes	100,00%	src/app/dados/migracaomunicipioeleicao	100,00%
src/app/dados/asn/correspondencia	99,00%	src/app/dados/municipio	99,40%
src/app/dados/asn/correspondenciaurna	100,00%	src/app/dados/municipiozona	100,00%
src/app/dados/asn/eleitores	91,60%	src/app/dados/notificacaoimportacao	100,00%
src/app/dados/asn/fotoscandidatos	100,00%	src/app/dados/parametrizacaourna	100,00%
src/app/dados/asn/impedidos	56,40%	src/app/dados/partidos	100,00%
src/app/dados/asn/local	100,00%	src/app/dados/processoeleitoral	99,30%
src/app/dados/asn/midias	83,30%	src/app/dados/rdv	78,10%
src/app/dados/asn/migracaomunicipioeleicao	98,60%	src/app/dados/respostaversaopacotes	100,00%
src/app/dados/asn/municipio	100,00%	src/app/dados/resultadournacadastro	93,10%
src/app/dados/asn/municipiozona	100,00%	src/app/dados/scueconf	100,00%
src/app/dados/asn/notificacaoimportacao	45,30%	src/app/dados/secoes	98,40%
src/app/dados/asn/parametrizacaourna	92,70%	src/app/dados/solicitacaoversaopacotes	100,00%
src/app/dados/asn/partidos	100,00%	src/app/dados/urna	100,00%
src/app/dados/asn/processoeleitoral	95,60%	src/app/rdv/servico	100,00%

Tabela 1 - Relatório de cobertura de linhas do módulo *libecourna*

Na análise de teste de cobertura também foi possível identificar que, em geral, os módulos externos/3rd-party (módulos em "/include/extern") possuem índices de cobertura bem mais baixos que aqueles observados para os módulos principais — provavelmente desenvolvidos *in-house* (Tabela 2). Por um lado, vale destacar que é comum assumir que a responsabilidade de testar módulos externos é do agente (desenvolvedor/empresa) que fornece tal módulo. Por outro lado, caso existam recursos disponíveis para a melhoria da efetividade dos testes, vale a pena considerar melhorar os índices de cobertura, também, dos módulos externos.

Directory	Line Coverage	Directory	Line Coverage
src/include/asn1	99,20%	src/include/extern/boost/random	93,80%
src/include/extern/boost	81,20%	src/include/extern/boost/random/detail	71,40%
src/include/extern/boost/algorithm/string	82,90%	src/include/extern/boost/range	100,00%
src/include/extern/boost/algorithm/string/detail	91,70%	src/include/extern/boost/range/detail	100,00%
src/include/extern/boost/bind	100,00%	src/include/extern/boost/regex/v5	28,90%
src/include/extern/boost/core	91,70%	src/include/extern/boost/signals2	8,50%
src/include/extern/boost/date_time	42,30%	src/include/extern/boost/signals2/detail	39,10%
src/include/extern/boost/date_time/gregorian	38,50%	src/include/extern/boost/smart_ptr	66,70%
src/include/extern/boost/date_time/posix_time	45,90%	src/include/extern/boost/smart_ptr/detail	61,40%
src/include/extern/boost/detail	11,40%	src/include/extern/boost/system	0,00%
src/include/extern/boost/dynamic_bitset	92,90%	src/include/extern/boost/system/detail	0,00%
src/include/extern/boost/dynamic_bitset/detail	25,00%	src/include/extern/boost/thread	49,10%
src/include/extern/boost/exception	36,60%	src/include/extern/boost/thread/detail	93,10%
src/include/extern/boost/filesystem	100,00%	src/include/extern/boost/thread/pthread	78,80%
src/include/extern/boost/format	56,90%	src/include/extern/boost/type_traits	75,00%
src/include/extern/boost/function	0,00%	src/include/extern/boost/utility	75,00%
src/include/extern/boost/io	68,50%	src/include/extern/boost/variant	0,00%
src/include/extern/boost/io/detail	53,80%	src/include/extern/boost/variant/detail	0,00%
src/include/extern/boost/iterator	36,10%	src/include/extern/gtest	15,20%
src/include/extern/boost/lexical_cast	18,20%	src/include/extern/gtest/internal	23,20%
src/include/extern/boost/lexical_cast/detail	70,20%	src/include/extern/iiiasn1	50,00%
src/include/extern/boost/move	100,00%	src/include/extern/iiiasn1/codecs	48,40%
src/include/extern/boost/numeric/conversion	0,00%	src/include/extern/iiiasn1/types	95,60%
src/include/extern/boost/optional	23,10%	src/include/extern/liblzma/CPP/7zip/UI/Console	100,00%
src/include/extern/boost/optional/detail	0,00%		

Tabela 2 - Relatório de cobertura de linhas do módulo *libecourna* (módulos externos/3rd-party)

Dado que a análise preliminar de cobertura evidenciou oportunidades de melhoria na qualidade das suítes de testes, o próximo passo natural foi o de investigar estratégias para a geração de casos de teste. Como a geração manual de casos de teste é bastante custosa, optamos por investigar a viabilidade de integrar ferramentas e/ou estratégias de geração automática de testes. Dentre as várias possibilidades para a geração automática de testes, iniciamos a nossa investigação com ferramentas de *fuzzing*, uma vez que tal estratégia foca em testes de robustez e segurança [6]. Nossa análise preliminar indica que seria viável integrar ferramentas de *fuzzing* existentes (por exemplo, AFLplusplus[1], libFuzzer[2], honggfuzz[3] e radamsa[4]).

Um desafio associado com a geração automática de casos de teste é a definição dos *oráculos* [7], ou seja, a definição do mecanismo que produz o veredito (passa ou falha). Por isso, como trabalho futuro, pretendemos investigar estratégias de *test amplification* [8] — ferramentas de *test amplification* recebem, como entrada, um caso de teste existente e produzem uma versão mais robusta do teste através da inserção automática de *oráculos*. Outra oportunidade de trabalho futuro é a investigação da viabilidade de adotar estratégias de geração de casos de teste baseadas em algoritmos de busca [9]. A efetividade dos casos de teste gerados automaticamente pode ser verificada utilizando a estratégia de teste de mutação [10].

As três estratégias que foram consideradas nessa análise preliminar — cobertura de testes, geração automática de casos de teste e teste de mutação — estão intimamente ligadas e compartilham um objetivo final comum: o de melhorar a efetividade da suíte de testes. Caso este projeto seja continuado, pretendemos investigar tais estratégias de maneira mais aprofundada.

[1] https://github.com/AFLplusplus/AFLplusplus
[2] https://llvm.org/docs/LibFuzzer.html
[3] https://github.com/google/honggfuzz
[4] https://gitlab.com/akihe/radamsa

Qualidade do Código Fonte

Foram realizadas observações gerais em relação ao código fonte, particularmente no sistema savp-votacao:

- o código pareceu excessivamente modularizado, com muitas subrotinas e/ou métodos com códigos bem pequenos, alguns até com 10 linhas ou menos;

- verificamos o uso de estilos de programação diferentes em situações semelhantes.

Uma sugestão seria considerar padronizar o estilo do código, quando possível. Por exemplo, em situações onde o comando return será usado mais de uma vez. Em uma parte dos códigos, usa-se ninhos de IFs com ELSE, e em outros só os IFs com os return, sem usar ELSE.

Algumas observações específicas sobre os códigos são listadas abaixo:

- Em chabilitadorEleitores :: EleitorExiste
  Aparentemente o código faz uma pesquisa sequencial na lista de eleitores da seção inteira (título como chave).
  Seria importante analisar a viabilidade de usar alguma estrutura de acesso direto para melhorar a eficiência.

  Isso também acontece em outros códigos.
  Por exemplo:
  - em czeResumoController :: montaCandidatosConsultasZerestima, no segundo e terceiro FOR;

- Em cVotoEleitorService
  - Em app :: dados :: cVoto

    cVotoEleitorService :: verificarVotoProporcional (

Há um ninho de IFs no final com mais de uma opção com return iguais: três opções para votos nulos e duas para opções de legenda.

Acho que seria mais claro ligar esses casos iguais com OR/AND, conforme a situação.

- Isso também acontece em outros códigos.

Por exemplo, em cfunctorZerestima :: PodeEmitirZerestima.

Vários IFs com NOT mas o retorno é só TRUE ou FALSE.

O código poderia ser mais simples e eficiente, sem usar os NOT nas condições, algo como:

```
IF ... AND ... AND ...
   return TRUE
ELSE
   return FALSE
```

Esse é um exemplo do caso citado acima - ELSE opcional: poderia ser usado ou não, dependendo do estilo preferido.

- Em app :: dados :: cVoto

cVotoEleitorService :: verificarVotoConsulta (

Há um comando FOR com consulta repetida a listaRespostas.size().

Idealmente fazer consulta única e guardar o resultado em uma variável para depois usá-la no FOR. Isso inclusive já foi feito em outros códigos em situações semelhantes.

- Em cCedulaControlador :: formataVotoParaExibicao

- Vários comandos IF só para converter tipo de voto para string. Seria interessante já ter isso disponível diretamente, de alguma forma, já que é executado muitas vezes.

- Mesmo ponto de consultar size em comando FOR citado acima.

Conclusões

Nenhum dos estudos desenvolvidos identificou problemas que comprometam o funcionamento do software analisado ou que demandem correções ou alterações na versão do software prevista para uso no ciclo eleitoral corrente.

Apesar de não haver necessidade de alterações urgentes, as sugestões apresentadas neste documento e nas apresentações realizadas visam o aumento da qualidade do código das aplicações e de seus referidos testes manuais e automatizados. Essas melhorias, uma vez realizadas, tornariam o código e seus testes mais robustos, legíveis e mais fáceis de manter, aumentando, assim, a sua qualidade.

Acreditamos que as sugestões propostas podem ser avaliadas pela equipe do TSE para possível adoção em um próximo ciclo, sem que haja prejuízo para o funcionamento do ciclo em curso.

Ressaltamos também que os pesquisadores envolvidos no projeto tem interesse, se possível, em dar continuidade a este trabalho até o prazo final do Termo de Adesão vigente e, se for o caso, vir a estende-lo, possibilidade também prevista no Termo de Adesão.

Acreditamos que este trabalho, através das análises e sugestões realizadas, contribuiu para a avaliação e validação da qualidade do software usado nas eleições no Brasil, para termos eleições rápidas, seguras e confiáveis.

Referências Bibliográficas

[1] Elvys Soares, Márcio Ribeiro, Rohit Gheyi, Guilherme Amaral, and André Medeiros Santos, “Refactoring test smells with JUnit 5: Why should developers keep up-to-date?”, IEEE Transactions on Software Engineering, May 2022.

[2] Arie Van Deursen, Leon Moonen, Alex Van Den Bergh, and Gerard Kok. 2001. Refactoring test code. In Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP). 92–95.

[3] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and Dave Binkley. 2015. Are test smells really harmful? An empirical study. Empirical Software Engineering 20, 4 (2015), 1052–1094.

[4] HAUPTMANN, Benedikt et al. Hunting for smells in natural language tests. In: 2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013. p. 1217-1220.

[5] Zhu, Hong, Patrick AV Hall, and John HR May. "Software unit test coverage and adequacy." Acm computing surveys (csur) 29, no. 4 (1997): 366-427.

[6] Li, Jun, Bodong Zhao, and Chao Zhang. "Fuzzing: a survey." Cybersecurity 1, no. 1 (2018): 1-13.

[7] Barr, Earl T., Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. "The oracle problem in software testing: A survey." IEEE transactions on software engineering 41, no. 5 (2014): 507-525.

[8] Danglot, Benjamin, Oscar Vera-Perez, Zhongxing Yu, Andy Zaidman, Martin Monperrus, and Benoit Baudry. "A snowballing literature study on test amplification." Journal of Systems and Software 157 (2019): 110398.

[9] Ali, Shaukat, Lionel C. Briand, Hadi Hemmati, and Rajwinder Kaur Panesar-Walawege. "A systematic review of the application and empirical investigation of search-based test case generation." IEEE Transactions on Software Engineering 36, no. 6 (2009): 742-762.

[10] Jia, Yue, and Mark Harman. "An analysis and survey of the development of mutation testing." IEEE transactions on software engineering 37, no. 5 (2010): 649-678.