Universidade Federal de Pernambuco
Centro de Informática
Graduação em Ciência da Computação

FERRAMENTA DE SUPORTE A UMA METODOLOGIA PARA TESTES EXPLORATÓRIOS

Trabalho de Graduação

TAÍSE DIAS DA SILVA

ORIENTADOR: ALEXANDRE VASCONCELOS (amlv@cin.ufpe.br)

Recife, junho de 2009

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TAÍSE DIAS DA SILVA

"FERRAMENTA DE SUPORTE A UMA METODOLOGIA PARA TESTES EXPLORATÓRIOS"

TRABALHO DE GRADUAÇÃO APRESENTADO À GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUESITO PARCIAL PARA OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO.

ORIENTADOR(A): DR. ALEXANDRE VASCONCELOS

RECIFE, JUNHO/2009

Agradecimentos

Agradeço a todos que contribuíram de todas as formas para o desenvolvimento deste trabalho de graduação. Ao meu orientador Alexandre Vasconcelos, a minha irmã Tarciana, aos meus pais, Fernanda e Luiz, ao meu irmão, minha família, Péricles, amigos, como Macarrão, ex-colegas de trabalho e colegas de faculdade, minha família anfitriã com quem morei durante a maior parte do período em que este trabalho foi desenvolvido, muito obrigada.

Resumo

Softwares fazem parte de nossas vidas e, muitas vezes, perdas significativas são causadas a pessoas por resultados inesperados de tais programas. Isso pode ser evitado ou minimizado desenvolvendo-se um produto de qualidade e, para isso, é indispensável a realização de testes durante o desenvolvimento do software, o que exige um grande esforço e tempo de projeto. Durante os ciclos de desenvolvimento de software de muitas empresas, testes exploratórios são realizados com o objetivo de conhecer o produto e encontrar falhas enquanto o produto é estudado, usando as informações adquiridas para projetar novos e melhores testes.

Testes exploratórios são muito úteis: quando se sabe pouco a respeito do produto, quando o projeto não contém documentação especificando o sistema ou esta é de baixa qualidade ou desatualizada, quando não se tem um processo de testes definido, para complementar um processo de testes definido, ou como parte da preparação de testes de roteiro. Para que esse tipo de teste seja realizado de forma estruturada, existem metodologias de testes exploratórios.

Um problema no caso de realizar testes exploratórios, é que os passos executados pelo testador enquanto ele navega pela aplicação não são previamente documentados, pois os testes são projetados e executados simultaneamente à medida que o testador estuda e aprende sobre o comportamento do software. Apenas o testador, portanto, possui as informações necessárias para a elaboração do relatório que reporta os resultados dos testes, como, por exemplo, que funcionalidades ou cenários foram testados e quais testes passaram. Muitas vezes, algumas dessas informações são esquecidas pelos testadores que realizam testes exploratórios de forma *ad hoc*, sem documentar o que foi feito, sem seguir uma metodologia ou procedimento estruturado para testar a aplicação e, por isso, esses testadores não conseguem descrever seus raciocínios por esquecimento e falta de documentação. Entretanto, por meio de utilização de metodologias existentes para guiar a prática de testes exploratórios, é possível realizar esses testes de forma estruturada e documentada.

Considerando o acima exposto, este trabalho de graduação tem como objetivo aperfeiçoar uma metodologia existente, chamada de Metodologia para Testes Exploratórios, que sugere passos a serem seguidos para a realização do teste. Além disso, é também proposta uma ferramenta para dar suporte a essa metodologia, auxiliando na reportagem dos resultados dos testes.

Palavras-Chave: teste de roteiro, cenários de testes, teste de caixa-preta, teste de caixa-branca, ciclo de vida de desenvolvimento de software, ciclo de vida de teste, *bug*, testes exploratórios.

Abstract

Software is involved in our lives and losses are caused to individuals by unexpected results of such programs. This can be avoided or minimized by developing a product with quality and, therefore, it is essential to test the product during software development, which requires a great effort and time of project. During cycles of software development in many companies, exploratory tests are carried out in order to know the product and finding faults while the product is studied, using the information gained to design new and better tests.

Exploratory tests are very useful: if you know little about the product, when the project contains no documentation specifying the system or it is of low quality or outdated, when you do not have a testing process, to complement a testing process, or as part of the preparation of scripted tests. In order to perform this type of test in a structured way, there are exploratory testing methodologies.

A disadvantage of exploratory testing is that the steps executed by the tester while he navigates through the application are not documented, because the tests are designed and implemented simultaneously as the tester studies and learns about the behavior of the software under test. Only the tester, therefore, knows the necessary information for preparing the report which reports the test results, such as features or scenarios that were tested and which tests passed.

Some of these details are often forgotten by testers who perform exploratory testing on an ad-hoc way, without documenting what was done and without following a structured methodology or procedure to test the application. Therefore, these testers cannot describe their reasoning for forgetfulness and lack of documentation. However, by using existing methodologies to guide the practice of exploratory testing, it is possible to perform these tests in a structured and documented way.

Considering the explanation above, this work aims to improve an existing methodology, called the Methodology for Exploratory Testing, suggesting steps to be followed for the test. Moreover, a tool to support this methodology is proposed, assisting in the reporting of test results.

***Keywords**: scripted test, test scenario, black-box testing, white-box testing, development life cycle, test life cycle, bugs, exploratpry testing.*

Lista de Figuras

Sumário

1 Introdução

Desenvolver software de qualidade é importante porque softwares fazem parte de nossas vidas. Utilizamos softwares em telefones celulares, carros e impressoras, entre outros produtos. Entretanto, a maioria das pessoas já teve uma experiência de um software não funcionar, como, por exemplo, um site da Internet que não foi carregado corretamente. Além disso, nem todos os sistemas de software possuem o mesmo risco e, portanto, nem todos os problemas causam o mesmo impacto. Falhas em sistemas de software podem causar danos significantes dependendo da criticidade da aplicação [1] .

Para o desenvolvimento de um software com qualidade, evitando que perdas significativas sejam causadas a pessoas por resultados inesperados da execução do software, é fundamental a realização de testes. De acordo com Müller, Thomas, et al. [1] , testar é um processo que consiste em todas as atividades do ciclo de vida (tanto estáticas como dinâmicas) voltadas para o planejamento, preparação e avaliação de produtos de software e produtos de trabalho relacionados, a fim de determinar se eles satisfazem os requisitos especificados e demonstrar que estão aptos para sua finalidade e para a detecção de defeitos. Entende-se por atividades estáticas as que não executam o código do programa e, por atividades dinâmicas, as que executam o código do programa. Preparação significa a seleção do que será testado, bem como o projeto dos casos de testes.

Entretanto, a realização de testes exige grande esforço e tempo de projeto [6] . Estudos demonstram que, apesar de os custos com testes representarem 45% do custo de desenvolvimento de um produto, o custo de não testar é muito maior, porque o custo da detecção de um erro após a entrega do produto é, no mínimo, 100 vezes maior do que se o mesmo tivesse sido detectado em tempo de definição do sistema [6] . Entretanto, não é possível testar completamente todas as possibilidades e situações a que uma aplicação pode ser submetida [7] , e, o fato de nenhum defeito ser encontrado não é prova de corretude, pois testes apenas mostram a presença de defeitos e reduzem a probabilidade de que defeitos não descobertos permaneçam no software. Portanto, para que testes possam ser realizados dentro do cronograma definido em um projeto, é necessário um planejamento efetivo para que os testes possam cobrir a maior área possível do software [6] .

Uma poderosa abordagem de testes para determinados tipos de software e projeto é o teste exploratório, que é definido como sendo, simultaneamente, aprendizagem, design de testes e execução de testes [8] ; isso significa que os testes não são definidos antecipadamente em um plano de tes-

tes, mas que são dinamicamente projetados, executados e modificados. Por isso, menos preparação (em relação a conhecimento dos requisitos e design prévio dos casos de testes) é exigida, e importantes defeitos são encontrados rapidamente, tornando-se mais estimulante intelectualmente para o testador do que a execução de testes de roteiro [4] . Além disso, testes exploratórios são muito úteis: quando se sabe pouco a respeito do produto, quando o projeto não contém documentação especificando o sistema ou esta é de baixa qualidade ou desatualizada, quando não se tem um processo de testes definido, para complementar um processo de testes definido, ou como parte da preparação de testes de roteiro [10] .

Entretanto, teste exploratório também possui desvantagens [4] , como por exemplo, os testes executados dependerem da habilidade, experiência e intuição de testadores, e, portanto, podem não ser revisados com antecedência, rastreados ou facilmente repetidos. Além disso, pelo fato de os testadores construírem mapas e modelos mentalmente em vez de no papel, as habilidades e conhecimento vão embora junto com eles quando eles deixam o projeto. Muitas vezes, algumas dessas informações são esquecidas pelos testadores [9] que realizam testes exploratórios de forma *ad hoc*, sem documentar o que foi feito, sem seguir uma metodologia ou procedimento estruturado para testar a aplicação e, por isso, esses testadores não conseguem descrever seus raciocínios por esquecimento e falta de documentação. Por outro lado, por meio de utilização de metodologias existentes para guiar a prática de testes exploratórios, é possível realizar esses testes de forma estruturada e documentada.

Teste Exploratório é muitas vezes associado a teste *ad hoc*, ou seja, muitos testadores encontram falhas acidentalmente, sem terem utilizado nenhum planejamento ou documentação para isso [5] . Apesar disso, ele pode ser e é realizado de maneira estruturada e formal por várias empresas, como, por exemplo, Microsoft e Nortel [8] , através de metodologias de suporte a essa técnica, cujo objetivo é encontrar falhas mais difíceis de serem descobertas, provenientes de cenários mais complexos, além de guiar testadores na realização de testes exploratórios de forma estratégica.

Com o objetivo de minimizar as desvantagens encontradas na utilização de testes exploratórios, o trabalho apresentado em [15] propõe a Metodologia para Testes Exploratórios, que consiste das seguintes fases: planejamento, elaboração de cenários, execução de testes e análise dos testes realizados. A vantagem dessa metodologia em relação às outras é a de que ela provê um guia de cenários de testes que apresenta cenários utilizados para testar várias funcionalidades, sendo esse guia dividido em testes de campo, usabilidade, negócio e segurança.

Considerando o acima exposto, este trabalho de graduação tem como objetivo aperfeiçoar a Metodologia para Testes Exploratórios [15] . Para tanto, foram estudadas e analisadas outras meto-

dologias existentes de testes exploratórios e, com essas informações, foi feito um estudo comparativo entre elas, de acordo com critérios adotados por este trabalho. Esses critérios utilizados na realização da análise e do estudo comparativo das metodologias foram escolhidos baseando-se nas definições de conceito de metodologia, apresentado na dissertação de mestrado de Pedro Silva [11] , de técnicas de testes e de classificação de técnicas de testes, apresentados no livro *Lessons learned in software testing: a context driven approach*, de Cem Kaner, et al. [13] . Esses conceitos são apresentados na próxima seção.

Além disso, o trabalho também propõe uma ferramenta, chamada de SMTE, para dar suporte à Metodologia para Testes Exploratórios [15] (após ela ser aperfeiçoada), auxiliando na reportagem dos resultados dos testes, entre outras atividades da metodologia. Para isso, são identificadas as necessidades de automação das atividades da metodologia para que a ferramenta SMTE seja definida, e são também apresentadas ferramentas existentes no mercado que servem como *benchmark* para que as funcionalidades da ferramenta SMTE sejam propostas.

A próxima seção introduz conceitos relacionados ao termo metodologia para que, durante a análise de metodologias de testes exploratórios e o estudo comparativo entre elas, a ser feita na subseção 2.2.2 e na seção 3.2, respectivamente, sejam levados em consideração os elementos definidos nesses conceitos. Pelo fato de o contexto em que essas metodologias estão inseridas ser o de testes de software, são introduzidos também conceitos sobre testes.

1.1 Conceitos Básicos para a Metodologia Proposta

O conceito de metodologia (ou método), no contexto de qualquer etapa do ciclo de vida de um projeto de software, consiste de uma agregação de pelo menos três elementos: técnicas, processo associado ao método e linguagem padrão associada ao método [11] . O processo, nesse caso, é suportado por técnicas com base nessa linguagem padrão.

Essa linguagem padrão associada ao método é aqui interpretada como sendo qualquer conceito relevante, uma definição teórica sobre qualquer termo técnico que é descrito na metodologia. Por exemplo, na metodologia apresentada na subseção 2.2.2, são definidos os conceitos de função e de estabilidade, que compõem a linguagem padrão na qual se baseia a técnica funcional, também apresentada nessa mesma metodologia.

O processo associado ao método é interpretado, neste trabalho, como sendo o procedimento seguido na metodologia. Tal procedimento pode ser apresentado em forma de con-

juntos de tarefas a serem executadas, como na metodologia apresentada na subseção 2.2.2, ou em fases, como na metodologia apresentada na subseção 2.2.1, em que cada fase é composta de atividades a serem realizadas.

Técnica, por sua vez, é interpretada como definida no dicionário [12] , ou seja, é o mesmo que prática. Entretanto, para contextualizar melhor esse conceito neste trabalho, o conceito de técnica será discutido na próxima subseção. A Figura 1.1 ilustra esses três principais elementos de uma metodologia. Na figura, o símbolo de agregação ilustrado entre a metodologia e seus três elementos representa o fato de uma metodologia consistir de uma agregação desses três elementos. A seta dupla entre técnicas e processo associado representa o fato de o processo ser suportado por técnicas, e a seta que liga técnicas à linguagem associada representa o fato de as técnicas estarem baseadas nessa linguagem.

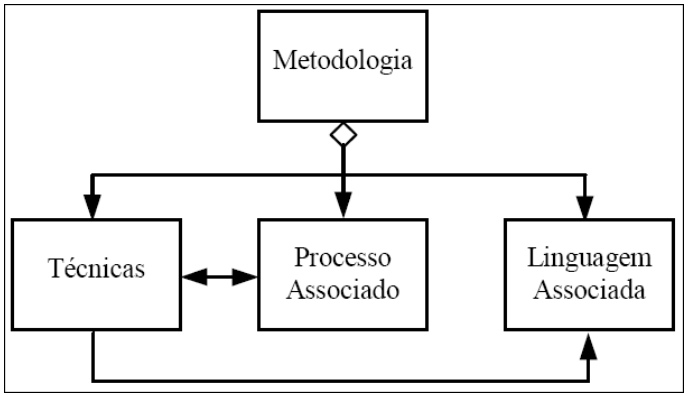

Figura 1.1 – Os Três Principais Elementos de uma Metodologia, **Fonte:** [11]

1.1.1 Técnicas de Testes

Nesta subseção, é apresentado o conceito de técnica de teste, a definição de algumas técnicas de testes e uma classificação de técnicas de testes.

Técnica de teste é um procedimento utilizado para derivar e/ou selecionar casos de teste [1] . Caso de teste, por sua vez, é um conjunto de valores de entrada (inputs), precondições de execução, resultados esperados e pós-condições de execução desenvolvidas para um determinado objetivo ou condição de teste, tais como para exercitar o caminho de um determinado programa ou verificar o atendimento a um requisito especifico [1] .

De acordo com Kaner, et al. [13] , existe um sistema de classificação de técnicas de testes, chamado *Five-fold Testing System*. Esse sistema afirma que qualquer teste que possa ser realizado pode ser

descrito em termos de cinco dimensões: testador, cobertura, problemas potenciais, atividades e avaliação. Testador é quem executa o teste. Por exemplo, o teste de aceitação é focado em teste por membros do mercado alvo, pessoas que normalmente irão utilizar o produto. Cobertura é o que é testado (exemplo, em teste funcional, cada função é testada). Problemas potenciais consistem no motivo pelo qual se está testando, ou seja, qual o risco pelo qual se está testando (exemplo, testar em busca de erros de valores extremos). A dimensão de atividades consiste em como testar (exemplo, teste exploratório). Avaliação consiste em como dizer se o teste passou ou falhou (exemplo, comparação para reconhecimento de um bom resultado). Dessa forma, técnicas de testes podem ser classificadas em técnicas baseadas em: testador, cobertura, problemas potenciais, atividades e avaliação.

Uma técnica de teste foca a atenção em uma ou mais dimensões [13] . É possível combinar uma técnica que foca em uma dimensão com outra que foque em outra dimensão para que seja alcançado o resultado que se deseja. A seguir são apresentadas algumas técnicas de testes.

Teste exploratório é definido por Kaner, et al. [13] como uma técnica baseada em atividade, em que o testador deve aprender, durante todo o projeto, sobre o produto, seu mercado, seus riscos, e de que formas o produto falhou em testes anteriores. Novos testes são constantemente criados e usados, e eles são mais poderosos do que os mais antigos porque são baseados no aumento do aprendizado contínuo do testador.

Teste funcional é uma técnica de teste baseada em cobertura, e consiste em testar a fundo cada função, uma por uma, até o ponto em que seja possível afirmar que essa função está funcionando [13] . Do ponto de vista de teste de caixa-preta, teste funcional foca em comandos e *features*, coisas que o usuário pode fazer enquanto usa a aplicação.

Consistência heurística é uma técnica de teste baseada na dimensão de avaliação do sistema de classificação de técnicas de testes *Five-fold Testing System*, pois consistência é um importante critério para avaliar um programa [13] . Inconsistência pode ser uma razão para reportar um *bug*, ou pode refletir em uma variação de projeto intencional. As sete principais consistências, de acordo com Kaner, et al. [13] , são:

- Consistência com história: comportamentos de funções atuais são consistentes com comportamentos passados dessas funções.
- Consistência com imagem da organização: comportamento da função é coerente com uma imagem que a organização quer projetar.

- Consistência com produtos comparáveis: comportamento de função é consistente com as funções similares em produtos comparáveis.

- Consistência com declarações: comportamento de função é consistente com o que as pessoas falam que ele deve ser.

- Consistência com expectativas do usuário: Comportamento de função é consistente com o que se espera que o usuário queira.

- Consistência dentro do produto: Comportamento de função é consistente com o comportamento de funções comparáveis ou padrões funcionais dentro do produto.

- Consistência com o propósito: Comportamento da função está consistente com o propósito aparente.

Outras técnicas de testes utilizadas por metodologias de teste exploratório apresentadas no próximo capítulo são:

Partição de Equivalência (ou análise de classe de equivalência) [13] : técnica que utiliza classes de equivalência, classes de equivalência é um conjunto de valores de uma variável que são considerados equivalentes. Casos de testes são equivalentes se eles testarem a mesma coisa, ou seja, se um deles encontrar um *bug*, o outro também encontrará esse mesmo *bug*. Se um deles não encontrar um *bug*, o outro provavelmente também não encontrará. Caso uma classe de equivalência tenha sido identificada, pode-se testar apenas um ou dois de seus elementos. Um exemplo da utilização dessa técnica é o seguinte [15] : em um sistema que contém um cadastro de produtos onde o campo descrição do produto deve conter entre 3 e 50 caracteres, as classes identificadas são a quantidade de caracteres abaixo de três, a quantidade de caracteres entre 3 e 50 e a quantidade de caracteres acima de 50. A figura a seguir ilustra esse exemplo.

Entrada	Valores Permitidos	Classes	Cenários
Descrição do Produto	De 3 a 50 Caracteres	< 3	Quantidade de caracteres = 1
		3 a 50	Quantidade de caracteres = 20
		> 50	Quantidade de caracteres = 60

Figura 1.2 – Partição de Equivalência, **Fonte**: [15]

Valores limites (ou teste de fronteira) [13] : técnica que complementa a técnica de partição de equivalência porque elas estão relacionadas da seguinte maneira: caso seja possível mapear os valores de uma classe de equivalência em uma linha numérica, os valores limites são o menor e o maior número da classe. Essa técnica testa esses valores limites, e os valores de fronteira das classes vizinhas que são menores do que o menor número da classe que está sendo testada, e maior do que o maior

número dessa classe. A figura a seguir apresenta a utilização da técnica de valores limites para exemplo apresentado anteriormente [15] .

Entrada	Valores Permitidos	Classes	Cenários
Descrição do Produto	De 3 a 50 Caracteres	< 3	Quantidade de caracteres = 2
		3 a 50	Quantidade de caracteres = 3 Quantidade de caracteres = 50
		> 50	Quantidade de caracteres = 51

Figura 1.3 – Valores Limites, **Fonte**: [15]

Valores Brancos ou Nulos [15] : consiste em não preencher entradas que são obrigatórias para o sistema.

Valores Inválidos e Negativos [15] : dados inválidos são utilizados nas entradas do software. Alguns exemplos disso são: digitar caracteres em campo numéricos e digitar números negativos em entradas que devem aceitar apenas números positivos. Algumas vezes os sistemas não permitem digitar valores inválidos e negativos, mas quando se cola tais valores que foram copiados de outro lugar nos campos de entrada, eles são aceitos.

Combinação de Dados [15] : são feitas várias combinações de dados de entrada. Cada combinação deve produzir um resultado específico. Nas combinações podem ser utilizados valores corretos, valores errados, valores inválidos, valor vazio.

Elizondo [5] apresenta técnicas utilizadas em testes exploratórios e as classifica em técnicas baseadas em entrada e técnicas baseadas em saída. Essas técnicas são listadas a seguir.

As técnicas baseadas em entrada são [5] :

- Mensagens de erros – o código fonte mostra todas as mensagens de erro disponíveis no produto. Um bug fácil de ser encontrado (que infelizmente acontece muito) é erro de escrita e de gramática na mensagem. Outro bug fácil de ser encontrado surge se for encontrado um cenário em que uma mensagem de erro não é apresentada quando deveria. Mas a idéia dessa técnica é exercitar todos os cenários que provocarem o aparecimento de uma mensagem de erro disponível. É importante que seja sempre apresentada a mensagem correta para cada cenário testado.

- Valores válidos e inválidos – A idéia é testar valores válidos e inválidos em todos os campos da aplicação. Por exemplo, se existir uma caixa de texto perguntando por idade, tente testar o campo com os valores -1, 0, 12, e 45686.

- Valores default – Se uma aplicação possui valores default em campos de entrada, delete esses valores e execute a aplicação. Depois, use valores diferentes, e tente usar os valores default novamente.

- Tipos de dados – Para uma caixa de texto que aceita idade, tente valores diferentes de inteiros; tente digitar uma letra para ver se a aplicação consegue lidar com tipos de dados diferentes corretamente.

- Overflow – Se uma caixa de texto aceita inteiros, digite o maior inteiro permitido e veja como a aplicação lida com ele.

- Mesma entrada várias vezes – Acredite ou não, se você digita alguma coisa em uma caixa de texto e executa a aplicação, e depois digita a mesma coisa e tenta executá-la novamente, ela pode travar. Muitas aplicações se comportam dessa forma.

- Refresh – Se existir um botão de refresh em uma aplicação, clique nele várias vezes.

- Preencha dados que são armazenados – Se uma aplicação armazena dados, preencha um campo cujo dado será armazenado e tente executar funcionalidades da aplicação que utilizam esses dados.

- Testar o software de interação – Se a sua aplicação faz uso de um servidor, faça com que o servidor pare de funcionar enquanto sua aplicação estiver se comunicando com ele.

- Corrompa o sistema de arquivos – Coisas esquisitas acontecem quando você corrompe arquivos que estão sendo utilizados pela sua aplicação.

As técnicas baseadas em saída são [5] :

- Saídas exatas – Verifique se computações matemáticas produzem saídas exatas.

- Forçar mudança de propriedades – Force mudanças nas propriedades dos componentes de interfaces com o usuário.

Pelo fato de técnicas de testes estarem relacionadas com estratégia de testes, a próxima subseção apresenta informações a respeito do conceito de estratégia, que também será útil para a compreensão da metodologia apresentada na subseção 2.2.3.

1.1.2 Estratégia de Testes

Técnicas de testes estão relacionadas com estratégia de testes, pois é durante a descrição da estratégia de testes que são definidas as técnicas que serão utilizadas nos testes. Segundo Kaner, et al. [13] , estratégia de teste refere-se ao conjunto de idéias que guiam o design de testes por todo o projeto, e é um componente importante de um bom plano de testes. Em uma estratégia de teste é definido como "cobrir" o produto para encontrar rapidamente problemas importantes, o que especificamente será testado, quais técnicas serão utilizadas para criar testes, e como as falhas são reconhecidas quando elas ocorrem. Para que falhas possam ser reconhecidas, é importante o uso de *oracles*, que são fontes que determinam qual é o resultado esperado de um software [20] . A estratégia de teste especifica a relação entre a missão de teste, ou seja, o que é testado ou quais problemas estão sendo procurados, e o projeto de teste.

Uma boa estratégia de testes é: específica para o produto e para a tecnologia usada, focada em risco (mostrando como o projeto vai analisar o que realmente importa, conectando o processo de teste com a missão no projeto), diversificada (incluindo uma variedade de técnicas e abordagens) e prática, ou seja, factível por quem irá executar essa estratégia, de forma que a estratégia sugerida não seja algo além das capacidades do projeto [13] . Para se garantir a diversidade de uma estratégia, podem ser selecionadas técnicas de testes de cada uma das cinco categorias de testes do sistema de classificação de testes aqui apresentado, o *Five-fold Testing System*.

1.2 Estrutura do Trabalho de Graduação

Além do capítulo introdutório, este documento é composto por mais quatro capítulos, cada um apresentado da seguinte forma:

Capitulo 2 – Apresenta o Estado da Arte e considerações a respeito de metodologias e ferramentas existentes de testes exploratórios.

Capítulo 3 – Apresenta um estudo comparativo das metodologias de testes exploratórios existente, o aperfeiçoamento da Metodologia para Testes Exploratórios, e a ferramenta SMTE proposta.

Capítulo 4 – Apresenta as conclusões sobre o trabalho, enfatizando as principais contribuições e sugestões para trabalhos futuros.

2 Estado da Arte

2.1 Introdução

Após a introdução dos conceitos relacionados ao termo metodologia no contexto da etapa de testes do ciclo de vida de um projeto de software no capítulo anterior, é essencial abordar o estado da arte, tanto de metodologias, como de ferramentas de testes exploratórios existentes hoje na comunidade científica e utilizadas por empresas no mercado de trabalho.

Este capítulo apresenta quatro metodologias. A primeira é a metodologia apresentada em um trabalho de graduação, chamada de Metodologia para Testes Exploratórios [15] , que será aperfeiçoada neste trabalho de graduação, e para a qual será especificada uma ferramenta de suporte. A segunda é a metodologia apresentada no artigo intitulado Testes Exploratórios: a próxima geração [5] , a terceira é o Procedimento de Teste de Funcionalidade e Estabilidade [14] , que na verdade não é considerada explicitamente pelo autor como sendo uma metodologia. Por isso, esse procedimento é analisado neste trabalho, que identifica aspectos desse procedimento que caracterizam uma metodologia [11] [13] , de maneira que este trabalho a considera uma metodologia. A quarta metodologia é a de Gerenciamento de Teste Baseado em Sessão [18] .

Além disso, este capítulo apresenta também as seguintes ferramentas de testes exploratórios existentes no mercado: Session-Based Test Management Scan Tool [18] , Exploratory Test Assistent, e Test Explorer Controller [16] . Essas ferramentas servem como *benchmark* para que as funcionalidades da nova ferramenta sejam propostas.

2.2 Metodologias

2.2.1 Metodologia para Testes Exploratórios

A metodologia de testes exploratórios desenvolvida por [15] , intitulada de Metodologia para Testes Exploratórios, tem como objetivo melhorar os resultados obtidos com a prática de execução de testes exploratórios, e sugere passos básicos para a realização dessa prática. Essa metodologia consiste de quatro fases: planejamento, elaboração de cenários básicos, execução e análise dos resultados. Tal metodologia deve ser utilizada e adequada de acordo com as necessidades de cada organização que irá utilizá-la, de forma que cada fase deva ser adaptada ou incluída, caso seja necessário.

Planejamento é a primeira fase da metodologia, em que o plano de teste é elaborado com base no plano de projeto. O plano de teste contém as seguintes seções: escopo de testes, cronograma, papéis e responsabilidades, estratégia e ambiente. Na seção de escopo de testes, é definido o que será testado, relatando-se os requisitos funcionais e não-funcionais do produto. No cronograma, é definido quando e quanto tempo cada fase deve durar. Na seção seguinte, os papéis e as responsabilidades dentro da execução de cada fase do processo são definidos, além de ser especificado quem da equipe irá assumir cada papel. Na seção de estratégia, são definidas a estratégia e as técnicas utilizadas para testar determinadas funcionalidades. Por fim, a última seção do plano de testes descreve o ambiente necessário para a execução dos testes. O plano de teste deve estar sempre consistente com o plano de projeto.

Elaboração de testes, a segunda fase da metodologia, baseia-se em um guia (Figura 2.1) que contém uma quantidade de cenários que se aplica a várias funcionalidades. Dessa forma, alguns desses cenários que se apliquem ao teste são selecionados e é desenvolvida uma planilha de execução (Figura 2.2), que é definida e apresentada na fase de execução. Esse guia é dividido em testes de campo, usabilidade, negócio e segurança, e contém cenários que foram elaborados baseando-se em técnicas tais como: partição de equivalência, valores-limite, valores brancos ou nulos, valores inválidos e negativos, e combinação de dados, explicadas na subseção 1.1.1. Dessa forma, os cenários devem ser escolhidos de acordo com as técnicas que forem mais adequadas para testar as funcionalidades do sistema. Esse guia é apresentado na Figura 2.1 a seguir.

	Guia de Cenários de Testes		
	Cenários	**Exemplo**	**Observações**
CAMPO	**VALIDAÇÃO DO PREENCHIMENTO DO CAMPO**	**VALOR PERMITIDO DE 3 A 50**	
	1. Valor maior que valor limite	**Quantidade de caracter igual a 51**	
	2. Valor igual ao valor limite	**Quantidade de caracter igual a 50**	
	3. Valor menor que o valor limite	**Quantidade de caracter igual a 25**	
	4. Valor igual ao mínimo	**Quantidade de caracter igual a 3**	
	5. Valor menor que o valor mínimo	**Quantidade de caracter igual a 2**	
	6. Em branco (não preenchido)		
	7. Valor maior que o valor limite (Ctrl + C, Ctrl + V)		**Copiar e colar uma quantidade de caracteres maior que a permitida**
	8. Espaçamento entre caracteres		
	VALIDAÇÃO DO CAMPO DO TIPO NÚMERO		
	1. Caracteres especiais	**#$%"&**	
	2. Números negatives	**-6**	
	3. Números decimais	**6,78**	
	4. Caracteres alfanuméricos	**5fg7**	
	VALIDAÇÃO DO CAMPO TIPO DATA		

	1. Mês inexistente	18/00/2007	
	2. Data inexistente	32/01/2007	
	3. Dia negative	-1/01/2007	
	4. Mês negative	18/-1/2007	
	5. Ano negative	18/1/-2007	
	6. Caracteres alfanuméricos	dd/mm/2007	
	7. Formato invalid	18/1/2007	
	8. Data maior que a atual	data de amanhã	Quando a maior data permitida é a atual
	9. Data menor que a atual	data de ontem	Quando a menor data permitida é a atual
	10. Data final menor que data inicial		Quando existe data de início de data de término
	11. Ano bissexto	29/02/2008	
	VALIDAÇÃO DO CAMPO DO TIPO HORA	Formato HH:MM	
	1. Hora inválida	25:25	
	2. Minuto invalid	12:67	
	3. Hora negative	-6:57	
	4. Minuto negative	13:-8	
	5. Hora vazia	:34	
	6. Minuto vazio	15:	
	7. Formato	12:34:23	
	VALIDAÇÃO DE OUTROS CAMPOS		
	1. Validação do campo email	exemplo#exemplo.com	sem @
	2. Validação do campo CPF	046.567.345-78	com valores inválidos
	VALIDAÇÃO DO CAMPO (OUTROS)		
	1. Somente leitura e/ou editáveis		
	2. Valor default dos campos		
	3. Barra de rolagem		Verificar a ativação da barra de rolagem em campo somente leitura
USABILIDADE	VALIDAÇÃO		
	1. Validação visual da tela e dos componentes		
	2. Texto de ajuda		
	3. Tecla ENTER		
	4. Tecla TAB		Troca de foco entre os campos
	5. Teclas de atalho		
	6. Texto tooltip		
	7. Links de navegação		
	8. Barra de rolagem		
	9. Resolução (800x600 e 1024x768)		
MENSAGEM			
	1. Sintaxe		
	2. Semântica		
	3. Padrão visual	Verificar componentes de tela	Verificar componentes de tela
NEGÓCIO	VALIDAÇÃO DE FLUXOS		
	1. Fluxo báscio		
	2. Fluxos alternatives		
	3. Fluxos de excessão		
	4. Regra de negócio		

	OPÇÕES DE PREENCHIMENTO DE CAMPOS		
	1. Todos os campos obrigatórios vazios		
	2. Todos os campos obrigatórios preenchidos		
	3. Primeiro e último campos obrigatórios		
	4. Apenas 1 campo obrigatório preenchido		
SEGURANÇA	**1. Base de dados indisponível**		
	2. Sessão web expirada		
	3. Permissão de acesso do Usuário		
	4. Garantir que o sistema não preserva dados na tela após operação	**Após incluir um registro**	
	5. Testes de F5/Refresh		
	6. Testes de SQL Injection		**Testar a inserção de comando SQL no sistema**

Figura 2.1 – Guia de Cenários de Teste, **Fonte**: [14]

Execução é a fase seguinte à elaboração de testes. Nela, os cenários de testes presentes na planilha de execução são executados pelo testador. Pelo fato de tais cenários serem básicos, é dada ao testador a liberdade para que sejam elaborados e executados outros testes durante essa fase. Os defeitos encontrados durante a execução são registrados.

Pode acontecer de alguns cenários não serem executados por restrições de tempo e recursos, o que requer que haja uma priorização dos cenários a serem testados. Dessa forma, aconselha-se testar cenários cujas funcionalidades envolvidas são mais críticas para o sistema, cenários que já apresentaram erros e testar também os aspectos vitais do sistema, que dependem das características e dos riscos do sistema. Esses aspectos vitais variam de um sistema para outro. Tais aspectos podem ser: funcional, segurança, desempenho e usabilidade.

O guia de testes apresentado anteriormente é instanciado para que seja criada uma planilha de execução, como a apresentada na Figura 2.2, que, além de conter informações apresentadas no guia, também contém campos a serem preenchidos com informações sobre a execução dos testes. Esses campos são: sistema, versão, tela, executor, data e, na área de resultado, é informada a quantidade de cenários que passaram no teste (passados), quantos cenários falharam (falhos), total de cenários (total) e o tempo total da execução (tempo total).

	Planilha de Execução				
	Sistema:	**Versão:**	**Tela:**	**Executor:**	**Data:**

	Resultado **Passados:** *Cenário(s)* **Falhos:** *Cenário(s)* **Total:** *Cenário(s)* **Tempo Total:**					
	Cenários	**Exemplo**	**Resultado**	**Solicitação de Mudança**		**Observações**
				ID	**Resumo**	
CAMPO	**VALIDAÇÃO DO PREENCHIMENTO DO CAMPO**					
	1. Valor maior que valor limite					
	2. Valor igual ao valor limite					
	3. Valor menor que o valor limite					
	4. Valor igual ao mínimo					
	5. Valor menor que o valor mínimo					
	6. Em branco (não preenchido)					
	7. Valor maior que o valor limite (Ctrl + C, Ctrl + V)					
	8. Espaçamento entre caracteres					
	VALIDAÇÃO DO CAMPO DO TIPO NÚMERO					
	1. Caracteres especiais	**#$%"&**				
	2. Números negatives	**-6**				
	3. Números decimais	**6,78**				
	4. Caracteres alfanuméricos	**5fg7**				
	VALIDAÇÃO DO CAMPO TIPO DATA					
	1. Mês inexistente	**18/00/2007**				
	2. Data inexistente	**32/01/2007**				
	3. Dia negative	**-1/01/2007**				
	4. Mês negative	**18/-1/2007**				
	5. Ano negative	**18/1/-2007**				
	6. Caracteres alfanuméricos	**dd/mm/2007**				
	7. Formato invalid	**18/1/2007**				
	8. Data maior que a atual	**data de amanhã**				
	9. Data menor que a atual	**data de ontem**				
	10. Data final menor que data inicial					
	11. Ano bissexto	**29/02/2008**				
	VALIDAÇÃO DO CAMPO DO TIPO HORA	**Formato HH:MM**				
	1. Hora inválida	**25:25**				
	2. Minuto invalid	**12:67**				
	3. Hora negative	**-6:57**				
	4. Minuto negative	**13:-8**				
	5. Hora vazia	**:34**				
	6. Minuto vazio	**15:**				
	7. Formato	**12:34:23**				
	VALIDAÇÃO DE OUTROS CAMPOS					
	1. Validação do campo email	**exemplo#exemplo.com**				
	2. Validação do campo CPF	**046.567.345-78**				
	VALIDAÇÃO DO CAMPO (OUTROS)					
	1. Somente leitura e/ou editáveis					

	2. Valor default dos campos					
	3. Barra de rolagem					
USABILIDADE	VALIDAÇÃO					
	1. Validação visual da tela e dos componentes					
	2. Texto de ajuda					
	3. Tecla ENTER					
	4. Tecla TAB					
	5. Teclas de atalho					
	6. Texto tooltip					
	7. Links de navegação					
	8. Barra de rolagem					
	9. Resolução (800x600 e 1024x768)					
MENSAGEM	1. Sintaxe					
	2. Semântica					
	3. Padrão visual					
NEGÓCIO	VALIDAÇÃO DE FLUXOS					
	1. Fluxo báscio					
	2. Fluxos alternatives					
	3. Fluxos de excessão					
	4. Regra de negócio					
	OPÇÕES DE PREENCHIMENTO DE CAMPOS					
	1. Todos os campos obrigatórios vazios					
	2. Todos os campos obrigatórios preenchidos					
	3. Primeiro e último campos obrigatórios					
	4. Apenas 1 campo obrigatório preenchido					
SEGURANÇA	1. Base de dados indisponível					
	2. Sessão web expirada					
	3. Permissão de acesso do Usuário					
	4. Garantir que o sistema não preserva dados na tela após operação	Após incluir um registro				
	5. Testes de F5/Refresh					
	6. Testes de SQL Injection					

Figura 2.2 – Planilha de Execução, **Fonte**: [14]

O campo sistema contém qual sistema está sendo testado; no campo versão, a versão do sistema, para que se saiba qual versão do sistema apresentou os erros reportados na planilha; no campo telas, é indicada a tela testada; em executor, o nome da pessoa que executou o teste; e no campo data, é indicada a data em que os testes foram executados.

Cada cenário da planilha contém os campos: resultado, ID, resumo e observações. O campo resultado informa se o teste passou ou falhou. ID e Resumo são preenchidos no caso de o teste falhar, com a identificação da solicitação de mudança e o resumo da falha, respectivamente. Já o campo de observações deve ser preenchido caso seja necessária alguma informação extra sobre o teste.

Análise de resultados é a fase seguinte nessa metodologia. Nela, os resultados obtidos são analisados e é elaborado um relatório reportando os resultados dos testes, contendo as seguintes seções: cenários dos testes, resultados e registros de defeitos abertos. A seção de cenários apresenta os cenários que foram executados para cada funcionalidade. Na seção de resultados são apresentados os resultados obtidos com a execução dos testes através de gráficos que informam: o status dos testes realizados, o percentual de testes que passaram por cenário de testes e o percentual de testes que falharam por cenário de teste. Já na seção de registro de defeitos, todos os defeitos reportados pelos executores são listados, bem como o número, a gravidade e a descrição dos defeitos.

2.2.2 Procedimento de Teste de Funcionalidade e Estabilidade

O Procedimento de Testes de Funcionalidade e Estabilidade [14] foi desenvolvido por James Bach e é intitulado *General Functionality and Stability Test Procedure*. Uma vez que esse Procedimento apresenta técnicas, processo associado ao método e linguagem associada ao método, este trabalho de graduação o considera como sendo, por definição [11] , uma metodologia.

Esse Procedimento de Testes de Funcionalidade e Estabilidade é utilizado pela Microsoft para efeito de certificação de compatibilidade de aplicações com o sistema operacional Windows 2000 [14] . Ele consiste de tarefas específicas, objetivos e entregas que fazem dele um processo sistemático. Esse conjunto de tarefas é interpretado neste trabalho de graduação como sendo o processo associado à metodologia, de acordo com o que foi apresentado na seção 1.1.

As tarefas específicas são: identificar o propósito do produto, identificar funções, identificar áreas de potencial instabilidade, testar cada função e registrar problemas, e, por fim, projetar e registrar um teste de verificação consistente. Essas tarefas não são seqüenciais, de forma que elas serão executadas concorrentemente até que todas sejam finalizadas.

As entregas a serem feitas após a realização das tarefas são: uma declaração do propósito do produto, um resumo das funções, uma lista de potenciais instabilidades e dados desafi-

adores (dados cujo uso possa provocar instabilidade no produto testado), falhas do produto e anotações, e um teste de verificação de consistência [14] . A Figura 2.3 a seguir ilustra essas tarefas e entregas.

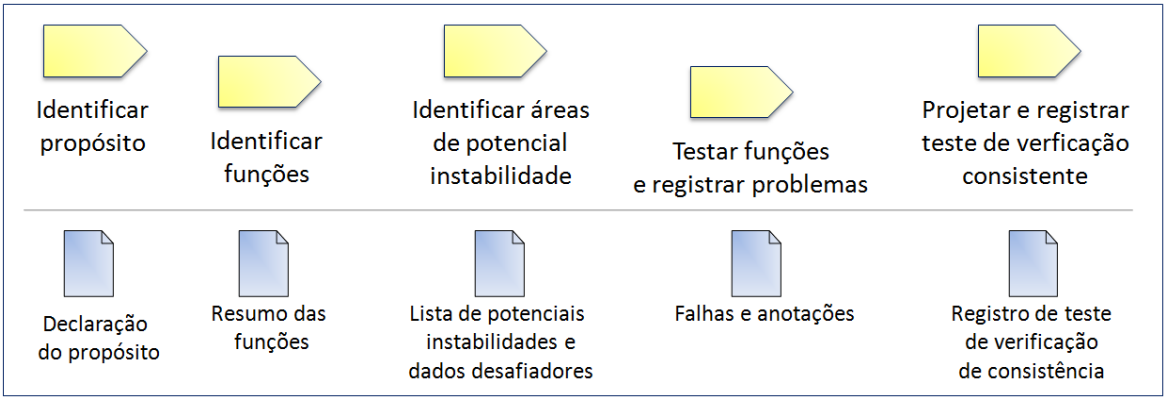

Figura 2.3 – Tarefas e entregas do Procedimento de Teste de Funcionalidade e Estabilidade

O Procedimento é finalizado uma vez que: cada tarefa tiver sido completada, cada dúvida ou assunto a ser tratado (problema) tiver sido resolvido ou aceito pelo gerente de teste, toda tarefa tiver sido aceita pelo gerente de teste, e o testador conhecer o suficiente sobre o produto para determinar se ele deveria ou não receber certificação de acordo com os critérios de funcionalidade e estabilidade [14] .

Esse Procedimento é suportado por técnicas, embora o termo "técnicas de teste" não tenha sido explicitado no mesmo. Por isso, ele foi analisado neste trabalho de graduação e as técnicas foram sendo identificadas de acordo com as definições e classificações de técnicas de testes apresentadas na seção 1.1.1. Foram identificadas duas técnicas de testes nesta metodologia: teste funcional e consistência heurística. Essa identificação foi inferida após estudar o background e conceitos envolvidos no Procedimento de Teste de Funcionalidade e Estabilidade, os quais são apresentados a seguir.

O background e conceitos envolvidos no Procedimento consistem de: definição de funções, forma de identificar tais funções no produto a ser testado, definição da cobertura de testes exigida e critérios de sucesso e de falha quando se está testando a funcionalidade e a estabilidade do produto. Os detalhes são descritos a seguir.

O Procedimento é organizado através de funções. O testador identifica as funções do produto e as classifica em primárias e contribuintes [14] . A prioridade é testar o máximo de funções primárias possíveis. Uma função é considerada primária se ela puder ser associada ao propósito do produto, e se ela for essencial para esse propósito. Uma função é classificada como contribuinte no caso de ela não ser primária e contribuir para a utilidade do produto [14] .

O primeiro ponto chave [14] para determinar se uma função é primária é conhecer o propósito do produto, o que requer que haja uma fonte autorizada de informação suficiente da qual se possa deduzir ou inferir esse propósito. O segundo ponto chave é saber que a função é essencial, o que depende de conhecimento sobre o usuário, de como a função trabalha e como outras funções do produto trabalham.

A cobertura de teste requerida nesse Procedimento [14] é: testar todas as funções que possam ser testadas de forma racional dentro do intervalo de tempo disponível, testar um conjunto interessante de funções contribuintes (com as quais provavelmente o testador irá se deparar ao testar funções primárias) e testar áreas selecionadas de potencial instabilidade. Com relação ao teste de funções primárias, o gerente de teste deve ser informado caso não seja possível testar alguma função primária, por falta de tempo ou habilidade para testar [14] . Com relação às áreas de instabilidade, que podem ser funções ou conjunto de funções, são escolhidas, em geral, de cinco a dez áreas do produto para serem testadas com dados que provavelmente provocarão instabilidade no produto.

O Procedimento [14] define critérios para avaliação do produto no que diz respeito às funcionalidades e estabilidade do mesmo, como é apresentado na Figura 2.4 a seguir.

Definição	Critério de Sucesso	Critério de Falha
Funcionalidade Habilidade do produto de funcionar.	1. Cada função primária testada opera de maneira aparentemente consistente com o seu propósito, sem levar em consideração a corretude das saídas da função	1. Pelo menos uma dessas funções parece incapaz de operar de maneira consistente com o propósito
	2. Qualquer comportamento incorreto que é observado no produto não prejudica seria-	2. O produto opera incorretamente de uma maneira que prejudica seriamente o seu uso

	mente o funcionamento do produto para uso normal	normal
Estabilidade Habilidade do produto de continuar funcionando, ao longo do tempo e sobre a sua escala de utilização, sem falhar ou causar falha.	3. O produto não atrapalha o funcionamento do Windows	1. O produto atrapalha o funcionamento do Windows
	4. O produto não trava, não tem que ser reiniciado e não provoca perda de dados	2. O produto travar, ter que ser reiniciado ou provoca perda de dados
	5. Nenhuma função primária se torna inoperável durante o teste	3. Pelo menos uma função primária se torne inoperável durante o teste

Figura 2.4 – Definições e Critérios de Avaliação de Funcionalidade e Estabilidade, **Fonte**: [14]

O padrão de funcionalidade é criado para ser o padrão mais exigente que possa ser racionalmente verificado por testadores, independentemente de terem familiaridade com o produto e de terem apenas alguns dias para completar o trabalho [14] . A palavra "aparentemente", mencionada no primeiro critério de sucesso de funcionalidade na Figura 2.4, significa "aparente para um testador com habilidades de computação ordinárias". O testador não será necessariamente capaz de dizer que o programa está funcionando "corretamente", mas se ele for capaz de dizer que o programa não está se comportando corretamente de uma maneira que seriamente danifica o produto, o produto falha no teste (certificação).

Para saber se o produto está seriamente danificado para uso normal, é preciso ter noção de como é o usuário normal e o que é o uso normal [14] . Em muitos casos, o usuário normal pode ser uma pessoa com habilidades de computação básicas. Em alguns casos, entretanto, o usuário normal será uma pessoa com atributos, habilidades, ou expectativas que são, de alguma forma, especializadas. Talvez seja preciso, portanto, estudar o domínio do produto ou consultar o vendedor a fim de pensar em um caso em que o produto deva falhar.

A fim de executar o teste de estabilidade, será preciso identificar e resumir os tipos de dados básicos que podem ser processados pelo produto [14] . Quando as áreas de potencial instabilidade forem testadas, será preciso utilizar esse conhecimento para projetar testes que utilizem entradas desafiadoras.

De acordo com o que foi apresentado até agora, este trabalho de graduação realiza algumas inferências como resultado da análise desta metodologia que está sendo apresentada nesta subseção. Primeiro, é inferido que o Procedimento de Funcionalidade e Estabilidade utiliza a técnica de teste funcional (subseção 1.1.1) porque ele é baseado em funções e tem como objetivo identificar e testar

as funções do produto para verificar se as mesmas estão funcionando. É inferida também a utilização da técnica de consistência heurística (subseção 1.1.1) porque esse Procedimento considera critérios de sucesso e de falha de funcionalidade e estabilidade do produto, verificando consistência de acordo com propósito, consistência com expectativas do usuário e consistência dentro do produto.

Baseando-se nos conceitos e background apresentados, as tarefas do Procedimento são executadas. Para isso, é necessário conhecer detalhes dessas tarefas. A documentação de cada tarefa é descrita por uma planilha que contém os seguintes elementos: descrição da tarefa, heurísticas, resultados, critério de saída e perguntas freqüentes (FAQs). A descrição da tarefa é uma descrição concisa de o que o testador tem que fazer. Heurísticas são uma ou mais listas de idéias que ajudam o testador a decidir o que fazer e servem para provocar ou focar em um raciocínio. Elas podem ser utilizadas de forma que cada idéia seja brevemente lida e considerar as implicações de tal idéia no produto que está sendo testado. Não é obrigatório o testador utilizar as heurísticas no caso em que nenhuma das idéias se aplica. Resultados é uma lista de entregas que devem ser feitas uma vez que a tarefa seja concluída, como resultados dessa tarefa. Critérios de saída é uma lista de coisas que devem ser verdadeiras para que a tarefa possa ser considerada como concluída, e o testador deve estar preparado para defender a veracidade dos elementos dessa lista. Alguns dos elementos da lista irão requerer algum julgamento subjetivo, mas nenhum deles é totalmente subjetivo. As perguntas freqüentes é uma lista de questões geralmente indagadas por testadores quando se deparam com uma tarefa do procedimento pela primeira vez. A seguir, as descrições das tarefas do procedimento em questão são apresentadas com mais detalhes.

A tarefa de identificar o propósito do produto consiste em:

1. Revisar o produto e determinar qual serviço fundamental ele deve prover. Na medida do possível, definir o público para o produto.

2. Escrever (ou editar) um parágrafo que explica sucintamente a finalidade do produto e o público destinado.

A tarefa de identificar funções do produto consiste em:

1. Navegar pelo produto e descobrir o que ele faz.

2. Fazer um resumo de todas as funções principais.

3. Registrar funções contribuintes que são interessantes ou relacionadas a funções primárias.

4. Encaminhar quaisquer funções que o testador não sabe como classificar ou que é incapaz de testar para o Gerente de Teste.

A tarefa de identificar áreas de potencial instabilidade do produto consiste em:

1. À medida que o produto é explorado, atentar para funções que parecem mais prováveis do que a maioria de violar os padrões de estabilidade.

2. Selecionar de cinco a dez funções ou grupos de funções para realizar teste de instabilidade. Você pode selecionar funções contribuintes, caso elas pareçam ter grandes chances de falhar, mas instabilidade em funções primárias é mais importante.

3. Determinar o que poderia ser feito com as funções que poderia potencialmente desestabilizá-las. Pensar em grandes e complexas, ou desafiadoras entradas.

4. Listar as áreas de instabilidade selecionadas, juntamente com o tipo de dados ou estratégias que serão utilizados para testar essas áreas.

A tarefa de testar cada função e registrar problemas encontrados no produto consiste em:

1. Testar todas as funções primárias possíveis dentro do intervalo de tempo disponível.

2. Testar todas as áreas de potencial instabilidade identificadas.

3. Testar um conjunto interessante de funções contribuintes.

4. Registrar qualquer falha encontrada.

5. Registrar quaisquer notas sobre comportamentos encontrados do produto. Notas são comentários sobre comportamentos preocupantes exibidos pelo produto, mas que não são falhas.

Por fim, a tarefa de projetar e registrar um teste de verificação de consistência tem por objetivo registrar um procedimento para exercitar as funções primárias mais importantes do produto para garantir que o produto se comporta consistentemente em outras plataformas e configurações do Windows.

2.2.3 Testes Exploratórios: a próxima geração

Essa metodologia apresenta técnicas de testes exploratórios pesquisadas e desenvolvidas pelo profissional da Microsoft, David G. Elizondo, apresentada no Starwest 2008, em seu artigo "Testes Exploratórios: a próxima geração" (*Exploratory Testing: The Next Generation*) [5] . Esse artigo descreve

novas abordagens para testes exploratórios que são suportadas por automação, bem como informações que os testadores precisam para explorar; explica também como levantar informações, como usar essas informações para encontrar mais *bugs* de maneira mais eficaz e demonstra uma metodologia exploratória rápida e direta de encontrar *bugs* de forma não-acidental [17] .

Segundo Elizondo [5] , o conceito de testes exploratórios compõe-se de quatro partes: *Freestyle testing*, *Recorded testing*, *Guided testing* e *Assisted testing*.

Freestyle testing é definido como sendo um teste *ad-hoc*, que pode ser executado por qualquer pessoa que dispõe de tempo e energia para testar o produto sem conhecer o código da aplicação (teste de caixa preta) ou ter um planejamento ou uma documentação para consultar ao testar o software [5] .

Recorded testing é apresentado como sendo um *freestyle testing*, com a diferença de que são gravados vídeos de como a aplicação se comporta ao ser testada [5] . Tais vídeos proporcionam vantagens interessantes em relação ao *freestyles testing*. Por exemplo, o vídeo gravado pode ser utilizado para elaboração de novos cenários de testes, criação de diagramas que podem ser utilizados para identificar novos caminhos de códigos a serem testados, ou até para ser usado como evidência de que o *bug* existe, de forma que o desenvolvedor analisará o cenário exato, com os passos que o testador executou para encontrar o *bug*.

Assisted testing é o que Elizondo [5] chama de o futuro do teste. Em diferentes técnicas, o testador interage com a aplicação, analisa a aplicação, aprende com ela, e segue em sua busca por novos *bugs*. Isso parece um processo que pode ser automatizado. Se a história se repete todo o tempo, e pesquisas provam que desenvolvedores cometem sempre os mesmos erros, então ferramentas, que vão ajudar o testador a encontrar *bugs* que são sempre criados em aplicações de todos os tipos, podem ser criadas. Imagine um testador navegando por uma aplicação que contém várias caixas de texto que aceitam diferentes tipos de dados. Um exemplo de *assisted testing* pode ser uma ferramenta que, em tempo real, detecta o tipo de dado suportado pela caixa de texto (e os limites que ela suporta) e, baseado nisso, sugere dados com os quais se pode testar essa aplicação. Isso irá mostrar que uma simples caixa de texto apresenta um número infinito de entradas (ou casos de testes) possíveis. Mas sabe-se também que existe uma teoria por trás de testar com certas entradas. Se isso for automatizado, é dado aos testadores o poder de exercitar código baseando-se na história aprendida por testadores do mundo inteiro.

Guided Testing, por sua vez, é definido como sendo um tipo de teste que foca em uma estratégia para encontrar *bugs* difíceis de serem descobertos de maneira *ad hoc*, e é onde está centrada a

idéia da metodologia [5] . *Guided Testing* possui três componentes que são utilizados para desenvolver a melhor estratégia (conceito de estratégia definido na subseção 1.1.1) a ser utilizada: **experiência com teste**, **experiência com projeto** e **documentação**.

Experiência com testes é a habilidade adquirida pelo profissional, proveniente de conhecimento obtido durante o período de tempo dedicado à prática de testar aplicações. Quanto mais novos softwares um profissional testa, mais idéias ele tem de como e de o que testar. O testador, segundo [5] , é capaz de acelerar seu processo de aquisição da experiência identificando e utilizando as técnicas certas. Essas técnicas são apresentadas separadamente em dois grupos, baseando-se no livro *How to Break Software,* de *James Whittaker*: técnicas baseadas em entradas e técnicas baseadas em saídas.

Experiência com projeto é o conhecimento documentado por profissionais enquanto trabalham em um projeto, como, por exemplo, *bugs* já encontrados e informações sobre código fonte. Essa experiência também inclui relacionamentos entre os profissionais, pois, quanto mais os testadores interagem com os desenvolvedores da aplicação testada, mais eles descobrem os erros que os desenvolvedores costumam cometer ao programar. Esses conhecimentos tornam-se, portanto, históricos que podem ser utilizados para fazer regressão e encontrar *bugs* em novos componentes.

Documentação refere-se a documentos utilizados durante o ciclo de desenvolvimento do software, tais como o de arquitetura e projeto e o código da aplicação. Isso ajuda a guiar o testador porque oferece informações ricas de como o software irá trabalhar, se interage com outras aplicações, entre outras que auxiliam na visualização de cenários de testes interessantes. Utilizar documentação para definir a estratégia do teste significa conhecer os detalhes do software, como, por exemplo, arquitetura, projeto e código da aplicação. Existem duas técnicas que podem ser utilizadas para descobrir *bugs* utilizando-se o código da aplicação: *code churning* e *code coverage*. *Code churning* consiste em partes do código que sofreram modificações, sejam elas linhas de código adicionadas, deletadas ou modificadas de uma versão do código fonte para outra. A técnica consiste em testar essas partes de código, as quais provavelmente apresentarão defeitos. *Code coverage* é a medida do grau de código fonte que está sendo testado. O interessante de saber essa informação é que estatísticas mostram quais partes do código é *dead code*, quais partes ainda não foram testadas e que não estão sendo cobertas, por exemplo, por testes automáticos e, portanto, devem ser exercitadas através de testes manuais.

A combinação dessas informações apresentadas nessa metodologia é feita pelo testador, de acordo com sua necessidade, para criar uma estratégia de teste e atacar o software que se deseja testar.

2.2.4 Gerenciamento de Teste Exploratório Baseado em Sessão

Testes exploratórios são utilizados para se encontrar *bugs* mais rapidamente e de forma otimizada e, para isso, o planejamento de testes precisa ser continuamente reajustado para que se possa focar nas áreas de risco do produto mais promissoras [18] . Isso inclui também minimizar o tempo disponível para documentação, o que causa problemas quanto a reportar o que foi feito durante a exploração do produto. Entretanto, é preciso conhecer essas informações para que se saiba o que foi testado, o que foi encontrado e quais são as prioridades para os próximos testes. Muitas vezes os testadores não conseguem se lembrar e expressar com detalhes essas informações, e também não documentam o que fazem em detalhes para não perder a flexibilidade de testar o produto de forma exploratória. Para que os testadores pudessem reportar e organizar o trabalho sem obstruir a flexibilidade e a capacidade de fazer descobertas importantes por acaso (que é o que faz um teste exploratório útil), Jonathan Bach desenvolveu um método suportado por uma ferramenta, chamado de Gerenciamento de Teste Baseado em Sessão [18] .

A primeira coisa que Jonathan Bach percebeu quando estava tentando reinventar o gerenciamento de teste exploratório foi que testadores fazem muitas coisas durante o dia além de testar. Para monitorar testes, seria preciso encontrar uma forma de distinguir teste de qualquer outra coisa. Foi quando nasceram as sessões.

Sessão, no caso, seria a unidade básica do trabalho de teste, um bloco ininterrupto, revisável e objetivo de um esforço de teste. Ser objetivo significa que cada sessão é associada a uma missão – o que é testado ou quais problemas estão sendo procurados. Ininterrupto significa ausência de emails, reuniões, conversas ou ligações telefônicas. Revisável significa que um relatório, chamado de planilha de sessão, é produzido e pode ser examinado por uma terceira parte, como o gerente de testes, e provê informações sobre o que aconteceu.

Cada sessão dura mais ou menos 90 minutos. O tempo não é marcado rigorosamente porque um bom teste é mais importante do que o tempo que foi gasto para se testar. Se uma sessão dura perto de 45 minutos, ela é chamada de sessão curta. Se ela demorar mais que du-

as horas, é chamada de sessão longa. Por causa de reuniões, emails, e outras atividades importantes, é esperado que cada testador realize não mais que três sessões em um dia normal.

O que especificamente acontece em cada sessão depende do testador e da missão da sessão. Por exemplo, o testador pode ser alocado para analisar uma função, ou procurar um problema particular, ou verificar um conjunto de *bugs* consertados.

Cada sessão é interrogada. Para novos testadores, o interrogatório acontece o mais cedo possível depois que a sessão termina. À medida que os testadores vão ganhando experiência e credibilidade no processo, essas reuniões demoram menos tempo, e é possível até cobrir várias sessões de uma vez só. O objetivo principal do líder de teste é entender e aceitar o relatório da sessão. Outro objetivo é prover *feedback* e treinamento ao testador.

Após desenvolver um entendimento estrutural através dos interrogatórios de o quanto pode ser feito em uma sessão de teste, e através do rastreamento de quantas sessões são realmente realizadas em um determinado período de tempo, é adquirida a habilidade de estimar a quantidade de trabalho envolvido em um ciclo de teste e prever quanto tempo o teste vai durar mesmo sem o trabalho ter sido planejado em detalhes.

Teste exploratório pode parecer uma tarefa grande e complexa, mas é na verdade um aglomerado de sub-tarefas que vão aparecendo e desaparecendo na medida em que elas são realizadas. É preciso saber que tarefas serão executadas em uma sessão sem que seja feito um relatório muito cheio de informações, porque coletar dados de testes de forma muito detalhada consome muita energia que deveria ser utilizada na realização dos testes.

Para saber sobre as tarefas realizadas no teste, é exigido que os testadores reportem essas tarefas de forma bem geral. As sessões de teste são divididas em três tipos de tarefas: design e execução de testes, investigação e reportagem de *bug*, e setup da sessão. Elas são chamadas de métricas "TBS" (Task Breakdown). Depois, é exigido que os testadores estimem uma proporção relativa de tempo que eles passam em cada tarefa. Design e execução de teste significa navegar pelo produto e procurar problemas. Investigação e reportagem de *bug* é o que acontece quando o testador se depara com um comportamento que parece ser incorreto. Setup da sessão é qualquer outra coisa que os testadores fazem para que as duas primeiras

tarefas sejam realizadas, incluindo tarefas como configuração de equipamento, localização de materiais, leitura de manuais, ou escrita de relatório da sessão.

É exigido também que os testadores reportem a porção do tempo de teste que eles passaram nas missões associadas às sessões, e a porção do tempo de teste que eles passaram em oportunidades, que é qualquer teste que não se adéqüe à missão da sessão. Uma vez que se está sendo realizado teste exploratório, é preciso lembrar e incentivar testadores a divergir um pouco da missão caso eles se depararem com um problema que não faz parte da missão, mas que pareça importante.

Além das métricas TBS, existem mais três importantes partes da planilha de sessão: *bugs*, *issues* e notas. *Bugs* são preocupações sobre a qualidade do produto. *Issues* são questões ou problemas relacionados ao processo de teste ou ao projeto de forma geral. Notas são registros de qualquer outra coisa, como, por exemplo, idéias para casos de teste, listas de funções, ou qualquer coisa relacionada ao teste que ocorre na sessão.

A sessão inteira consiste das seguintes seções: missão da sessão (incluindo uma declaração da missão e áreas a serem testadas), nome do testador, data e hora de início, métricas TBS (*Task Breakdown*), arquivos de dados, notas do teste, *issues* e *bugs*. A Figura 2.5 a seguir apresenta um exemplo de uma sessão:

MISSÃO

Analisar a funcionalidade Mapmaker do menu View e apresentar um relatório sobre as áreas de risco potencial.

#ÁREAS
OS | Windows 2000
Menu | View
Estratégia | Teste Funcional
Estratégia | Análise Funcional

START

5/30/00 03:20 pm

TESTADOR

Jonathan Bach

TASK BREAKDOWN

5
#DURAÇÃO
curta
DESIGN E EXECUÇÃO DE TESTE
65
INVESTIGAÇÃO E REPORTAGEM DE BUG
25
#SETUP DA SESSÃO
20
#MISSÃO VS. OPORTUNIDADE
100/0

ARQUIVOS DE DADOS

#N/A

NOTAS DE TESTE

Eu cliquei em cada item do menu abaixo, mas foquei mais no comportamento do zoom com várias combinações de elementos do mapa exibidos.
View: Tela de “Bem-Vindo”
Navegador
Mapa de Localização
Legenda
Elementos do Mapa
Níveis de Highway
Níveis de rua
Diagrama de Aeroportos
Zoom In
Zoom Out
Nível do Zoom
(Níveis 1-14)
Previous View
Riscos:
- Exibição incorreta de um elemento do mapa.
- Exibição incorreta devido a interrupção quando a tela está sendo pintada novamente.
- CD pode estar ilegível.
- Versão antiga do CD pode estar usada.
- Alguma função do produto pode não funcionar em um certo nível de zoom.
6

BUGS

#BUG 1321
Ampliar faz você colocar o CD 2 quando você chega a um certo nível de granularidade (nível de nome de rua) - mesmo que o CD 2 já se encontre no driver.
#BUG 1331
Ampliar rapidamente resulta em os nomes das ruas não serem renderizados.
#BUG <não_anotado>

Instabilidade com velocidade do CD baixa ou baixo vídeo RAM. Ainda está sendo investigado.

ISSUES

#ISSUE 1
Como saber quais detalhes devem aparecer nos níveis de zoom?
#ISSUE 2
Não tenho certeza de como o mapa de localização deve funcionar. Como os usuários devem interagir com ele?

Figura 2.5 – Exemplo de Planilha de Sessão, **Fonte**: [18]

As tabelas de *breakdown* (*Task Breakdown*), listadas na Figura 2.5, contêm os números de *breakdown* das tarefas para cada planilha de sessão. Essas tabelas são normalizadas de forma que os dados de todas as planilhas de sessões possam ser combinados. As métricas indicam qual parte do trabalho foi gasto com a missão em cada uma das tarefas TBS, qual parte de sessões foram gastas com oportunidades, e quanto tempo (em unidades de sessão) foi gasto em trabalho não-relacionado com sessão. A última métrica é obtida assumindo que um dia de trabalho normal inclui 5.333 sessões normais se testadores não fizerem nada além de sessões. Monitorando quem estava "em serviço" na equipe de teste, e em quais dias, é possível saber o número máximo teórico de sessões que podem ser executadas em um dado período de tempo do calendário. É subtraído o número de sessões que ocorrem atualmente, e é dada a quantidade total de trabalho que não é relacionado a sessões. Esse cálculo não é preciso, mas ele é preciso o suficiente para o propósito de ter uma idéia de quanto teste é efetivamente realizado durante o dia.

Essas métricas ajudam a estimar e prever melhor do que antes coisas como a curva de aprendizagem dos testadores, os efeitos de adicionar um novo testador a um time estabelecido, e o impacto na produtividade de testar um produto mais testável versus um menos testável. Por exemplo, digamos que uma equipe de teste esteja realizando três sessões de testes por dia, em média, e um novo testador é acrescentado à equipe. Uma vez que testadores serão interrompidos pela necessidade de proverem treinamento, eles farão menos (ou menores) sessões do que antes, ou eles gastariam mais tempo em teste de oportunidade (uma vez que dar assistência a um testador em outra sessão é uma forma de trabalho de testes não-relacionado à missão). No início, interrogar o novo testador demora mais tempo. Os efeitos aparecerão nas métricas. Outra coisa simples que pode ser feita com métricas é organizar em gráficos todas as sessões de teste ao longo do tempo até a data de entrega.

Embora essas métricas possam proporcionar uma melhor visibilidade e percepção sobre o que estamos fazendo em nosso processo de testes, é importante perceber que o processo de teste baseado em sessão e as métricas associadas a ele poderiam ser facilmente distorcidas por um gerente de teste confuso ou tendencioso, ou pó um testador que não relate o teste da forma que ele realmente foi feito. O uso eficaz de métricas e planilhas de sessão exige a permanente conscientização sobre o potencial para esses problemas.

2.2.5 Considerações

Esta seção apresentou metodologias utilizadas para dar suporte à prática de testes exploratórios. Essas metodologias serão comparadas e analisadas no próximo capítulo para que essa comparação seja utilizada como base para as modificações a serem feitas na metodologia para a qual a ferramenta SMTE, proposta neste trabalho de graduação, dará suporte. A seguir, é apresentado o estado da arte de ferramentas de apoio a testes exploratórios.

2.3 Ferramentas de Apoio a Testes Exploratórios

2.3.1 Session-Based Test Management Scan Tool

Session-Based Test Management Scan Tool é uma ferramenta que dá suporte à Metodologia de Gerenciamento de Testes Baseado em Sessão [18] . Um importante ingrediente nessa abordagem de gerenciamento de teste baseado em sessão é o formato da planilha da sessão: cada relatório é provido de um formato de texto em *tag*, e armazenado em um repositório com todos os outros relatórios. Esses relatórios são posteriormente escaneados por uma ferramenta que foi desenvolvida para quebrá-los em seus elementos básicos, normalizá-los e resumi-los em tabelas e métricas. Usando essas métricas, é possível monitorar de perto o progresso de um teste, e fazer relatórios instantâneos para a gerência, sem ter que requisitar uma reunião à equipe. De fato, colocando essas planilhas, tabelas e métricas de sessões online, os clientes do projeto podem até ter acesso instantâneo às informações que eles desejam. Por exemplo, o gráfico da Figura 2.6 a seguir mostra que os testadores estão usando apenas um terço do tempo deles realmente testando, o que corresponde, em média, a duas sessões por dia, em vez de três. Uma vez que o gráfico representa dois meses de trabalho, ele sugere que exista algum tipo de obstáculo que impede os testadores de trabalharem usando capacidade total.

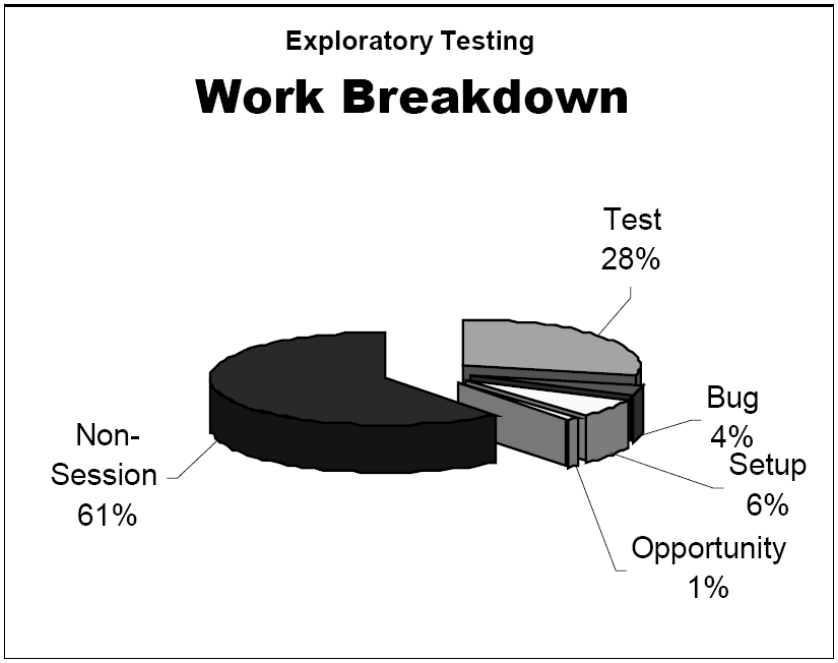

Figura 2.6 – Gráfico das Atividades do Testador, **Fonte**: [18]

A Figura 2.7 a seguir permite que se tenha uma idéia de quantas sessões se espera realizar durante o tempo que resta no projeto. Os dados da semana mais recente sugerem que a taxa de teste está acelerando.

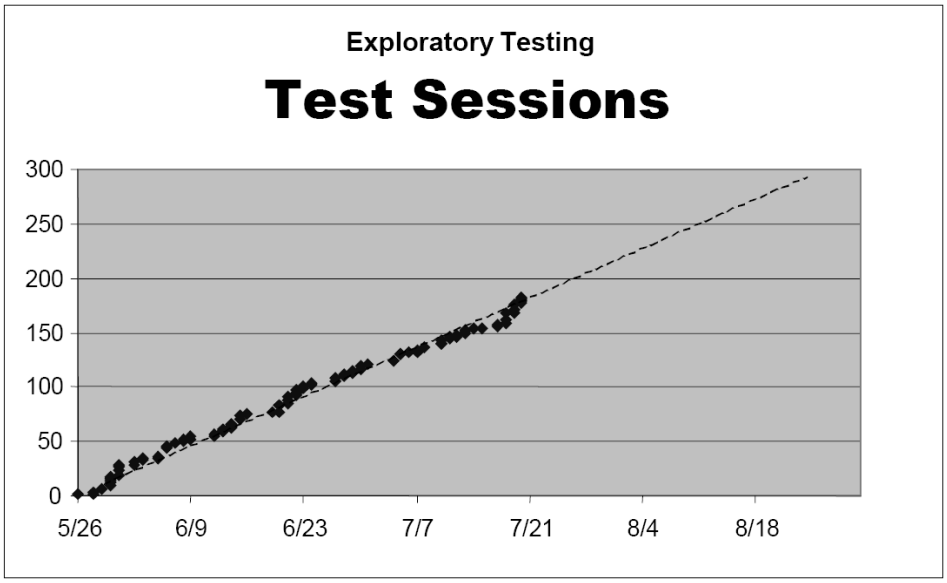

Figura 2.7 – Quantidade de Sessões por Intervalo de Tempo, **Fonte**: [18]

Descrição do Suporte Automatizado ao Método de Gerenciamento de Testes Baseado em Sessão

As planilhas de sessões são escaneadas por uma ferramenta desenvolvida em Perl. A ferramenta executa oitenta verificações de sintaxe e consistência. Por exemplo, se um arquivo de dados é referenciado em uma planilha de sessão, a ferramenta garante que o arquivo foi armazenado em um diretório de arquivos apropriado. A sessão da missão na planilha de sessão apresentada na Figura 2.5 permite a inclusão de *keyword* de área de teste específica (por exemplo, a *keyword* "Menu | View") de forma que cada sessão pode ser associada a elementos de uma matriz de teste utilizada na geração de tabelas descritas no próximo parágrafo. Os valores válidos dessas *keywords* são armazenados em um arquivo separado a fim de reduzir erros de digitação, pois em vez de as *keywords* serem digitadas ou criadas no momento em que são digitadas, elas já estarão pré-definidas nos arquivos e só precisarão ser utilizadas.

A saída do scanner é um conjunto de tabelas de texto que ajuda a contar a estória do projeto de teste. Cada tabela de texto está em um formato delimitado apropriado para importação para o Excel para formatação e análise. O scanner produz as seguintes tabelas:

- **Notas de teste**: contém sessões de notas de teste, por ID da sessão.
- **Bugs**: registro de bugs, por ID do bug e ID da sessão.
- **Issues**: registro de issue, por ID da issue e ID da sessão.
- **Misssão**: declaração da missão e keywords das áreas, por ID da sessão.
- **Arquivos de dados**: Nomes de arquivos de dados, por ID da sessão.
- **Breakdowns da sessão**: Métricas da sessão, por ID da sesão
- **Breakdowns da cobertura**: Métricas da sessão, por keyword da área.
- **Breakdowns do testador**: Métricas da sessão, por nome do testador.
- **Breakdowns do dia**: Métricas da sessão, por dia.

- **Sessões de ToDo**: Planilhas de sessões incompletas.

A tabela de sessões de ToDo é um método utilizado para agilizar o trabalho dos testadores. Uma sessão de ToDo é criada por um testador e contém apenas a missão do teste, estando todas as outras sessões em branco. Quando os testadores terminam uma sessão, eles olham a pasta de planilhas, escolhem uma planilha de ToDo e executam a sessão com aquela missão. A tabela de ToDo gerada pela ferramenta é uma lista de planilhas de ToDo que estão atualmente na pasta de planilhas de sessão. Essa lista é chama de "hopper".

A ferramenta possui também uma funcionalidade de busca, que permite rapidamente localizar, apresentar na tela e imprimir qualquer planilha de sessão que possui uma string específica. Essa ferramenta é importante porque o cliente pode pedir, a qualquer momento, para ver os dados por traz das métricas e o resumo dos relatórios.

2.3.2 Exploratory Test Assistant

Exploratory Test Assistent é uma ferramenta utilizada para dar suporte à execução de testes exploratórios de maneira que o testador possa realizar anotações enquanto testa [18] . As anotações podem ser registradas em um arquivo de texto e no quadro de edição (clipboard). Após esse registro, a ferramenta formata as anotações e gera um relatório final do teste, de acordo com os *templates* definidos pelo usuário.

A ferramenta não fica visível durante todo o tempo em que o teste está sendo realizado, pois são utilizados atalhos de teclado que fazem com que ela seja apresentada na tela na medida em que isso se fizer necessário. Após a realização das anotações, as mesmas são salvas e é utilizado mais um atalho para que a ferramenta não permaneça visível.

Por ainda ser um protótipo, a Exploratory Test Assistent ainda possui muitos *bugs*, problemas de usabilidade, poucos atalhos de teclado, e muitas funcionalidades ainda permanecem não-implementadas. O criador disponibiliza a ferramenta para quem quiser usá-la e/ou implementar mais funcionalidades. Ela possui *templates* para dar suporte aos relatórios de teste baseados em sessão de Jonathan Bach [18] .

Uma lista do tipo *drop down* mostra o *template* que está sendo utilizado em um determinado momento (Não-formatado, Bug, Nota, Risco, Task Breakdown, Issue, Missão).

2.3.3 Test Explorer Controller

Test Explorer Controller é uma suíte de quatro ferramentas de teste de software: TestExplorer (utilizada para criar e executar testes), Administrator (cria e gerencia projetos e usuários), Issue Manager (gerencia *issues)*, e Analyzer (gera relatórios e gráficos) [16] . Test Explorer Controller também dá suporte à Metodologia de Gerenciamento de Testes Baseado em Sessão [18] .

A aplicação TestExplorer permite que o usuário crie e execute testes manuais para as aplicações que estiverem sendo por eles testada através de teste exploratório de duas formas: com a ferramenta sendo completamente transparente, ou realizando gravação em um estilo interativo. Além disso, os testes podem também ser escritos antecipadamente e depois executados.

A aplicação Administrator permite que administradores de projetos controlem atividades de diferentes projetos, criem e deletem projetos, escrevam e designem missões, gerenciem os campos de usuário definidos e o conteúdo de todas as listas dropdown, e que gerenciem os usuários. Testes também podem ser deletados.

A aplicação Issue Manager permite que os usuários gerenciem todas as *issues* do projeto, sejam elas criadas através dos testes ou de outra forma. *Issues* podem ser criadas, deletadas, monitoradas, gerenciadas e reportadas.

A aplicação Analyzer permite que o usuário crie relatórios e gráficos a partir das várias métricas coletadas automaticamente durante o teste e no processo de gerenciamento de *issue*.

3 Metodologia e Ferramenta Propostas para Testes Exploratórios

3.1 Introdução

Este capítulo compara as metodologias apresentadas no capítulo anterior e, com base nessa comparação, apresenta as modificações propostas na metodologia para a qual a ferramenta proposta neste trabalho de graduação dará suporte. Além disso, a ferramenta proposta neste trabalho de graduação também é apresentada.

3.2 Estudo Comparativo das Metodologias

Todas as metodologias aqui apresentadas possuem como objetivo guiar o teste de produtos de software utilizando práticas de testes exploratórios e, para isso, as mesmas sugerem a utilização de técnicas de testes. Portanto, técnicas e processos serão utilizados como critérios de comparação das metodologias. No entanto, a metodologia apresentada no artigo "Testes Exploratórios: A próxima geração" [5] , na subseção 2.2.3, não apresenta de forma explícita um processo a ser seguido, e a metodologia de Gerenciamento de Teste Exploratório Baseado em Sessão [18] , apresentada na subseção 2.2.4, não apresenta de forma explícita as técnicas utilizadas. Assim, serão comparados os processos das três metodologias apresentadas nas sessões 2.2.1, 2.2.2 e 2.2.4, e as técnicas utilizadas pelas três metodologias apresentadas nas seções 2.2.1, 2.2.2 e 2.2.3. Portanto, técnicas e processos serão utilizados como critérios de comparação das metodologias.

3.2.1 Comparação entre os processos das Metodologias

O processo utilizado no Procedimento de Teste de Funcionalidade e Estabilidade consiste, como apresentado na subseção 2.2.2, na execução das seguintes tarefas: identificar o propósito do produto, identificar funções, identificar áreas de potencial instabilidade, testar cada função e registrar problemas, e, por fim, projetar e registrar um teste de verificação consistente. A Metodologia para Testes Exploratórios, apresentada na subseção 2.2.1, consiste na realização das seguintes fases: Planejamento, Elaboração de Cenários, Execução e Análise de resultados. Já a metodologia de Gerenciamento de Teste Exploratório Baseado

em Sessão, apresentada na subseção 2.2.4, consiste das seguintes tarefas: design e execução de testes, investigação e reportagem de *bug*, e setup da sessão.

De acordo com a definição de cada fase e tarefa apresentadas nas subseções 2.2.1, 2.2.2 e 2.2.4, pode-se observar que existem relações entre algumas delas. A tarefa de identificar funções e a de identificar áreas de potencial instabilidade do produto está relacionada com a fase de planejamento, pois essas tarefas listam justamente o que será testado, ou seja, o escopo dos testes. Na tarefa de design e execução, em 2.2.4, é realizado um estudo do produto, mas não são definidas especificamente que funções e áreas de instabilidade devem ser identificadas. A tarefa de testar cada função e registrar problemas está relacionada com a fase de execução, em 2.2.1, e também com a tarefa de design e execução, em 2.2.4, pois é onde os testes são executados e as falhas encontradas são registradas. A diferença é que, na tarefa descrita em 2.2.2, os testes são executados sem que sejam documentados projetos de teste ou cenários de testes elaborados para essa execução. Sabe-se apenas o que será testado e os critérios de sucesso e falha do teste. Já a fase de execução é seguida da de elaboração de cenários, de forma que o testador já tem uma idéia de como testar por causa do Guia de Cenários de Testes. Existe documentação também na fase de design e execução descrita em 2.2.4, na seção de descrição da missão do teste, bem como das áreas testadas do produto. A Figura 3.1 a seguir apresenta uma visualização da comparação entre os processos das metodologias.

<table>
<tr><th></th><th>Procedimento de Teste de Estabilidade e Funcionalidade</th><th>Metodologia para Testes Exploratórios</th><th>Gerenciamento de Teste Baseado em Sessão</th></tr>
<tr><td rowspan="6">Processo</td><td>Identificar o propósito do produto</td><td>Planejamento</td><td rowspan="2">Design e Execução de Teste</td></tr>
<tr><td>Identificar funções</td><td rowspan="2">Elaboração de Cenários</td></tr>
<tr><td>Identificar áreas de potencial instabilidade</td><td rowspan="2">Investigação e Reportagem de Bug</td></tr>
<tr><td>Testar cada função e registrar problemas</td><td>Execução</td></tr>
<tr><td>Projetar e registrar um teste de verificação consistente</td><td>Análise dos Resultados</td><td>Setup da Sessão</td></tr>
</table>

Figura 3.1 – Comparação entre processos das metodologias da subseções 2.2.2, 2.2.1 e 2.2.4

3.2.2 Comparação de Técnicas utilizadas nas Metodologias

Aparentemente, em nenhuma das três metodologias utiliza-se técnicas de testes em comum, com exceção das de combinação de dados e de valores inválidos que são utilizadas

tanto pela metodologia apresentada em "Testes Exploratórios: A próxima geração", (subseção 2.2.3), como pela Metodologia para Testes Exploratórios (subseção 2.2.1).

Durante a fase de execução da Metodologia para Testes Exploratórios, é sugerido que sejam testadas funcionalidades críticas, cenários que já apresentaram erros e garantir aspectos vitais do sistema. A prática de testar funcionalidades críticas e garantir aspectos vitais do sistema possui semelhança com o teste funcional que é realizado na metodologia desenvolvida por Bach – Procedimento de Teste de Funcionalidade e Estabilidade (subseção 2.2.2) – uma vez que esse teste funcional consiste em testar funções primárias do sistema que, pela definição apresentada na subseção 2.2.2, podem ser interpretadas como aspectos vitais do sistema e funcionalidades críticas. Já a sugestão de testar cenários que já apresentaram erros assemelha-se à questão de experiência com projeto apresentada na metodologia desenvolvida por Elizondo – Testes Exploratórios: A próxima geração (subseção 2.2.3) – uma vez que é sugerido que a estratégia de teste seja desenvolvida com base em conhecimentos documentados pelo profissional, como *bugs* já encontrados. A Figura 3.2 a seguir apresenta uma visualização da comparação entre as técnicas utilizadas nas metodologias.

<table>
<tr><th></th><th>Procedimento de Teste de Estabilidade e Funcionalidade</th><th>Testes Exploratórios: A próxima geração</th><th>Metodologia para Testes Exploratórios</th></tr>
<tr><td rowspan="14">Técnicas</td><td rowspan="9">Teste Funcional</td><td>Mensagens de erros</td><td rowspan="2">Partição de equivalência</td></tr>
<tr><td>Valores válidos e inválidos</td></tr>
<tr><td>Valores default</td><td rowspan="3">Valores-limite</td></tr>
<tr><td>Tipos de dados</td></tr>
<tr><td>Overflow</td></tr>
<tr><td>Combinações</td><td rowspan="4">Valores brancos ou nulos</td></tr>
<tr><td>Mesma entrada várias vezes</td></tr>
<tr><td>Refresh</td></tr>
<tr><td>Preencher armazenamento de dados</td></tr>
<tr><td rowspan="5">Consistência Heurística</td><td>Quebrar software de interação</td><td rowspan="4">Valores inválidos e negativos</td></tr>
<tr><td>Corromper o sistema de arquivo</td></tr>
<tr><td>Diferentes saídas para cada entrada</td></tr>
<tr><td>Saídas exatas</td></tr>
<tr><td>Forçar mudanças de propriedades</td><td rowspan="3">Combinação de dados</td></tr>
<tr><td>Code churning</td></tr>
<tr><td>Code coverage</td></tr>
</table>

Figura 3.2 – Comparação entre técnicas utilizadas nas metodologias da subseções 2.2.2, 2.2.3 e 2.2.1

3.2.3 Considerações

Durante essa seção, foram comparadas metodologias que puderam contribuir para a análise da Metodologia para Testes Exploratórios que está sendo utilizada nesse trabalho de

graduação. Para isso, foi necessário analisar todas essas metodologias e comparar, na medida do possível, seus aspectos. Como conseqüência desse estudo, foi aperfeiçoada a Metodologia para Testes Exploratórios de forma que as modificações foram feitas em aspectos concretos de metodologias já existentes utilizadas para o mesmo propósito. Essas modificações serão apresentadas na próxima seção.

3.3 Aperfeiçoamento da Metodologia

Com base nos conceitos e estado da arte apresentados neste trabalho de graduação, as fases da Metodologia para Testes Exploratórios [15] passaram a ser chamadas de atividades. Cada atividade da metodologia aperfeiçoada possui uma tarefa relacionada (à atividade), para a qual foram definidos explicitamente, de forma mais detalhada, os passos a serem executados a sua realização.

A fase de Elaboração de cenários passou a ser a atividade de Design de Teste, uma vez que, além da elaboração de cenários, é definida também a missão do teste.

Durante a execução de testes em geral, é importante que pelo menos três informações sejam registradas: o que foi testado, como foi testado e o que foi encontrado, para que seja possível identificar as prioridades para os próximos testes. Isto será incluído na nova metodologia apresentada neste trabalho e não é apresentado na Metodologia para Testes Exploratórios [15] , apesar de o guia, mencionado em 2.2.1, ajudar nessa documentação, uma vez que apresenta possíveis cenários. Entretanto, esses cenários são genéricos e para se ter as três informações citadas anteriormente é preciso que os registros sejam realizados em forma de *features* ou funcionalidades do produto que foram testadas.

A seguir é detalhada a nova versão da Metodologia para Testes Exploratórios.

3.3.1 Metodologia para Testes Exploratórios Modificada

A metodologia proposta neste trabalho é uma agregação de um processo de testes e técnicas de testes que dão suporte a esse processo. O propósito dessa metodologia é testar um produto de software utilizando-se práticas de testes exploratórios de forma estruturada. Essa metodologia consiste em um processo composto de quatro atividades: planejamento, design de testes, execução e análise. Essas atividades proporcionam a realização de testes ex-

ploratórios de forma estruturada, uma vez que são definidas tarefas específicas, objetivos e entregas.

Planejamento é a primeira atividade a ser realizada para que sejam estabelecidos cronograma, alocação da equipe, ambiente, escopo de testes e a estratégia de testes a ser utilizada. As atividades de design e execução são realizadas em paralelo, pelo fato de testes exploratórios serem, por definição, a realização simultânea de design de testes e execução. Em seguida, é realizada a atividade de análise para apurar os resultados dos testes e reportar métricas. A Figura 3.3 a seguir apresenta as atividades da Metodologia para Testes Exploratórios aperfeiçoada e como elas se relacionam.

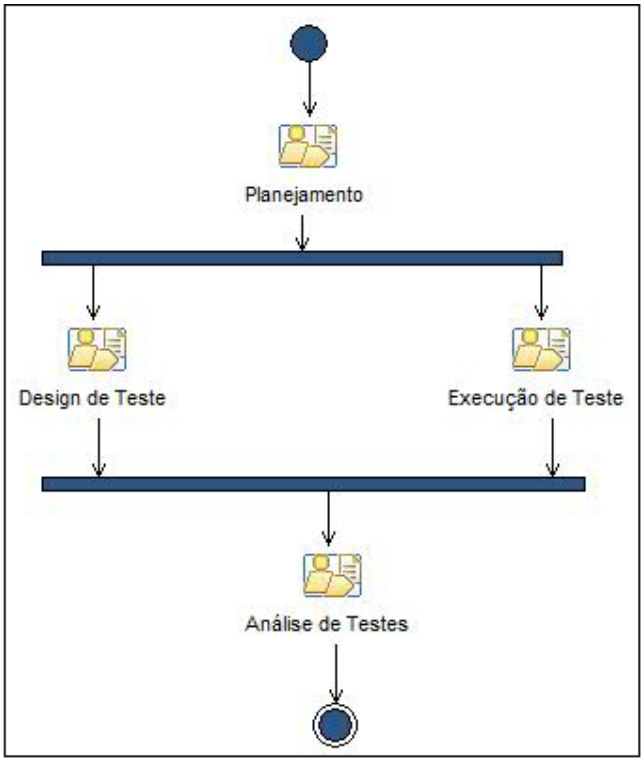

Figura 3.3 – Diagrama de atividades da Metodologia para Testes Exploratórios

A atividade de **Planejamento** consiste na tarefa de elaboração de um plano de testes com base no plano de projeto do produto. Para realizar essa tarefa, é necessário seguir os passos:

- **Estudar o produto**: consiste em navegar pelo produto a fim de conhecê-lo, identificar o propósito do produto, identificar funcionalidades, ou seja, o que o produto faz para atingir o seu propósito, e identificar quais dessas funcionalidades são críticas.

- **Definir escopo e estratégia**: procurar registro de bugs anteriores, erros que programadores costumam cometer, usar o conhecimento obtido pelo estudo do produto (propósito e funcionalidades críticas) e a experiência com testes de outros softwares para selecionar o escopo do teste. Baseando-se nessas informações, identificar as técnicas a serem utilizadas para testar as funcionalidades selecionadas. Para reconhecer falhas, é preciso que se utilize um oracle, que pode ser encontrado em documentos que descrevem o comportamento do produto, ou pode ser utilizada a experiência do testador.

- **Definir ambiente de teste**: define o ambiente necessário para a execução dos testes.

- **Definir cronograma**: define quanto tempo vai durar cada atividade do processo de teste.

- **Definir papéis e responsabilidades**: define quem vai executar quais atividades no processo de testes.

A atividade de **Design de Teste** consiste na tarefa de projetar testes, em que são definidas missões e cenários para a realização do teste. Para realizar essa tarefa, é necessário seguir os passos:

- **Definir missão do teste**: definir o que será testado ou quais problemas estão sendo procurados. Descrever as funcionalidades que são testadas nessa missão.

- **Definir cenários de testes**: para definir cenários de testes, pode-se utilizar o guia de cenários ou pode-se elaborar novos cenários de testes utilizando-se técnicas sugeridas na definição da estratégia de testes.

- **Elaborar planilha de execução de testes**: utilizando-se o guia de testes, cria-se a planilha de execução que é uma instância do guia. Caso novos cenários sejam criados, eles devem ser incluídos na planilha.

A atividade de **Execução de Teste** exercita e verifica funcionalidades do produto de acordo com a missão estabelecida e os cenários selecionados ou criados. Para realizar essa tarefa, é necessário seguir os passos:

- **Executar cenários da planilha**: os cenários da planilha de execução são executados para atingir a missão de teste estabelecida, testando-se as funcionalidades envolvidas e comparando os comportamentos observados com os comportamentos que o produto deve apresentar, de acordo com *oracles* utilizados, como documentos encontrados, experiência do testador ou heurísticas.

- **Reportar falhas ou *issues***: registrar falhas e *issues* encontradas durante os testes. Falhas são comportamentos não esperados do produto de acordo com o oracle utilizado para comparação com o comportamento esperado do produto. Issues são problemas que o testador acredita que possa ser uma falha, mas que não existe informação suficiente para comprovação de que o comportamento do produto não está correto.

A atividade **Análise de Teste** consiste na elaboração do relatório de testes, informando cenários executados por funcionalidade, resultados, e defeitos encontrados. Para realizar essa tarefa, é necessário seguir os passos:

- **Reportar cenários por funcionalidade das missões dos testes**: apresentar cenários que foram executados para cada funcionalidade das missões dos testes

- **Reportar resultados dos testes**: apresentar resultados dos testes através de porcentagens dos que passaram e falharam por cenário de teste.

- **Relatar defeitos encontrados**: apresentar defeitos que foram encontrados, informando a quantidade, gravidade e descrição.

3.4 Ferramenta Proposta

Esta seção apresenta a especificação da ferramenta SMTE para dar suporte à Metodologia para Testes Exploratórios. São apresentadas as atividades da metodologia envolvidas na automação, bem como tarefas e passos que são executados pelo engenheiro de testes que

receberão suporte da ferramenta. Além disso, a ferramenta é descrita em termos de casos de uso, interfaces gráficas com o usuário, e modelagem do banco de dados.

3.4.1 Identificação dos passos a serem automatizados

Para dar suporte à Metodologia para Testes Exploratórios proposta neste trabalho de graduação, foi especificada a ferramenta SMTE, que objetiva automatizar e auxiliar passos de tarefas das seguintes atividades da Metodologia: Design, Execução e Análise de Teste. A Figura 3.4 apresenta os passos das tarefas das atividades aos quais a ferramenta SMTE dá suporte. São eles: definir missão do teste, definir cenários de testes, elaborar planilha de execução de testes, reportar falhas ou *issues*, reportar cenários por funcionalidade das missões dos testes, reportar resultados dos testes e relatar defeitos encontrados.

Atividade	Tarefa	Passos
Design de Teste	Projetar Teste	- Definir missão do teste - Definir cenários de testes - Elaborar planilha de execução de testes
Execução de Teste	Executar Testes	- Reportar falhas ou issues
Análise de Teste	Elaborar relatório de testes	- Reportar cenários por funcionalidade das missões dos testes - Reportar resultados dos testes - Relatar defeitos encontrados

Figura 3.4 – Áreas da Metodologia para Testes Exploratórios suportadas pela ferramenta SMTE

3.4.2 Descrição da ferramenta SMTE

O usuário da ferramenta SMTE é o engenheiro de teste que, após realizar a atividade Planejamento da metodologia, precisa iniciar a atividade de Design e Execução de Teste e, em seguida, realizar a atividade de Análise de Teste.

Ao iniciar as atividades Design e Execução de Testes, o engenheiro precisa descrever a primeira missão do teste, as funcionalidades relacionadas a essa missão, e cenários apropriados para esse teste. Uma vez que a missão do teste, bem como cenários e funcionalidades a serem testados já estão definidos, pode-se iniciar a execução do teste, registrando os cenários

que passaram e falharam, à medida que o produto é explorado. Caso o produto apresente alguma inconsistência durante a execução de um cenário de teste, uma falha ou *issue* é reportada, de forma que o engenheiro deve descrever a inconsistência, classificar como falha ou *issue* e determinar a gravidade. Após o cumprimento da missão de teste, uma nova missão é definida, de acordo com a estratégia e o escopo do teste definido na atividade Planejamento.

Uma vez que as atividades Design e Execução de Testes estiverem concluídas, a atividade Análise de Teste é iniciada. O engenheiro de teste deve elaborar um relatório contendo o resultado dos testes, defeitos encontrados e os cenários testados por funcionalidade das missões dos testes.

Considerando-se essas necessidades do engenheiro de teste para realizar o seu trabalho durante a utilização da metodologia proposta, foram analisados requisitos para ferramenta SMTE. Essa análise resultou no diagrama de casos de uso apresentado na Figura 3.5. Foi identificada a necessidade de modelagem de onze casos de uso: definir missão, inserir, listar e remover funcionalidade, inserir, listar e remover cenário, registrar resultado do teste, reportar falhas e *issues*, gerar planilha de execução e gerar relatório.

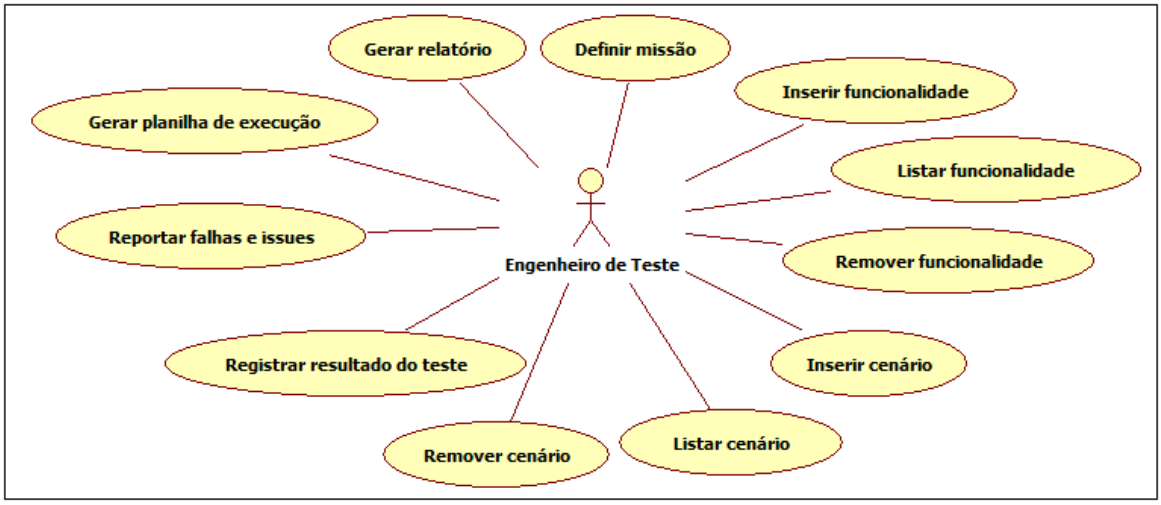

Figura 3.5 – Diagrama de Casos de Uso

As telas da ferramenta ilustram como a realização desses casos de uso é concretizada. A Figura 3.6 ilustra a interface gráfica inicial da ferramenta, exibindo o conteúdo da barra de

menu: definir missão, testar e registrar resultado, reportar falhas e *issues*, gerar planilha de resultados e gerar relatório.

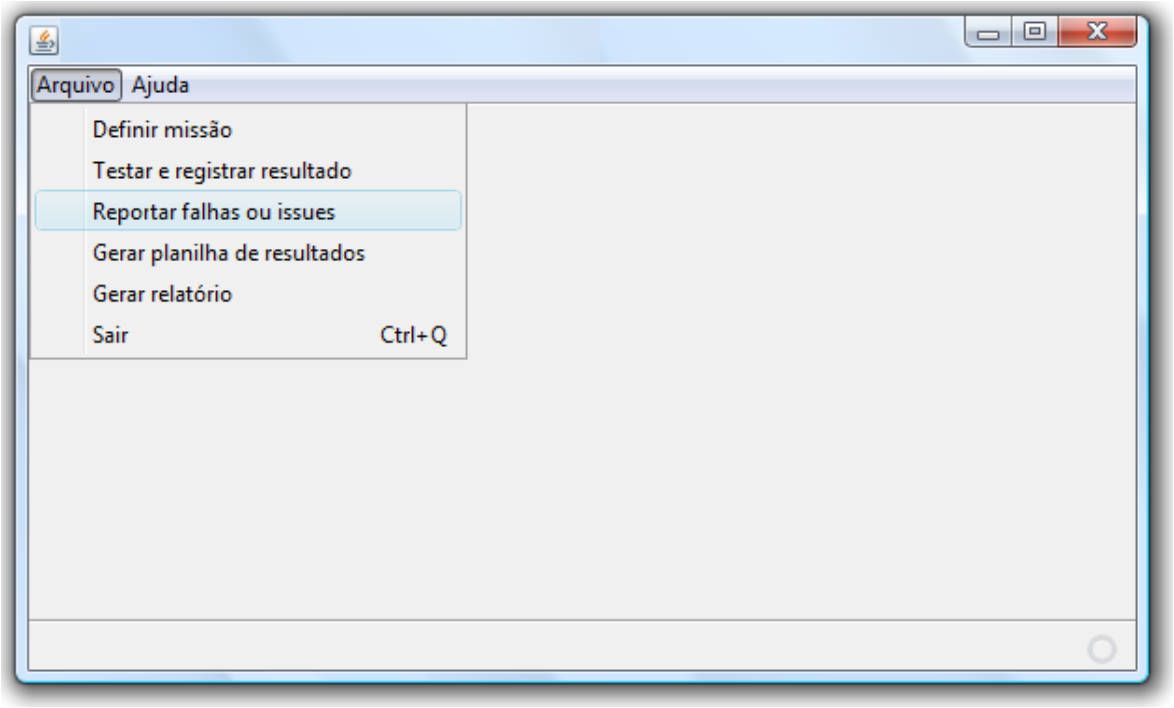

Figura 3.6 – Menu da ferramenta SMTE

A interface gráfica apresentada na Figura 3.7 ilustra como são realizados os seguintes casos de uso: definir missão, inserir, listar e remover funcionalidade, inserir, listar e remover cenário. Para definir uma missão, basta descrever o que será testado ou quais problemas estão sendo procurados no campo de texto indicado pelo *label* "Missão:". As funcionalidades testadas nessa missão podem ser registradas uma por uma, digitando-se a descrição da funcionalidade no campo de texto indicado pelo *label* "Funcionalidade:" e, em seguida, clicando-se no botão "Inserir". O botão "Remover" deve ser utilizado apenas no caso de uma funcionalidade ter sido inserida incorretamente. De forma análoga, um cenário é cadastrado digitando-se a descrição do cenário no campo de texto indicado pelo *label* "Cenário:" e, em seguida, clicando-se no botão "Inserir" ao lado desse campo. Os cenários inseridos podem ser os pré-definidos no Guia de Testes, bem como criados pelo engenheiro de teste, de acordo com o que for apropriado para cumprir a missão de teste. O botão "Remover" associado a cenário serve para o mesmo propósito do botão "Remover" associado à funcionalidade. A-

pós concluir a etapa de definição da missão de teste e dos cenários a serem utilizados, é importante salvar para que todos os dados sejam armazenados.

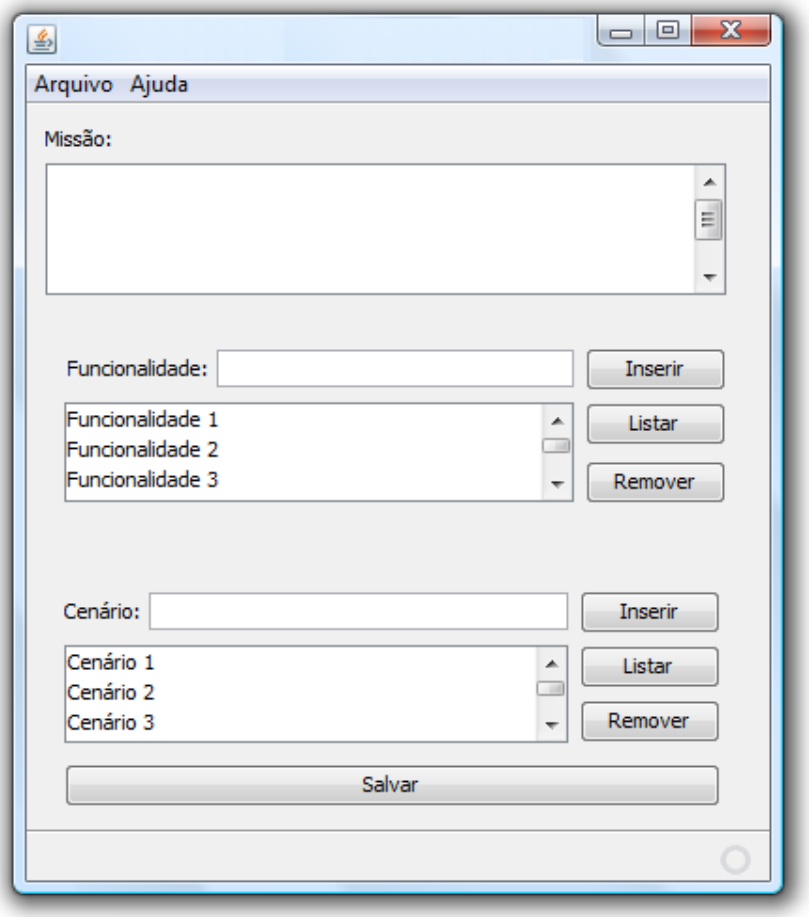

Figura 3.7 – Definir missão

Uma vez que se sabe qual a missão do teste, as funcionalidades e os cenários a serem testados, a execução é iniciada. A Figura 3.8 apresenta a interface gráfica de testar e registrar resultado, ilustrando como é realizado o caso de uso de registrar resultado do teste. Ao iniciar a execução, o engenheiro de teste seleciona o cenário que está testando e a funcionalidade associada, registrando o resultado que indica se o teste passou ou falhou, de acordo com o comportamento do produto testado.

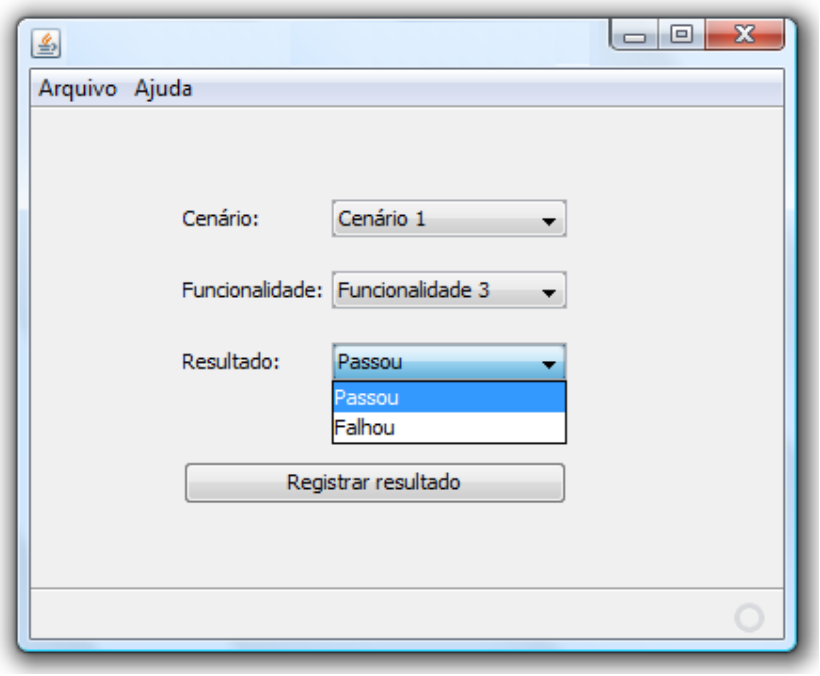

Figura 3.8 – Testar e registrar resultado

Na medida em que inconsistências forem sendo descobertas durante a execução dos testes, os cenários que falharam (ou que o engenheiro acha que falharam, mesmo sem ter certeza) devem ser reportados e a inconsistência deve ser descritas em detalhes. Além disso, é importante classificar a inconsistência como falha ou *issue*, de acordo com o que foi definido na metodologia. A Figura 3.9 ilustra a interface gráfica com a qual o usuário interage ao realizar esse caso de uso de reportar falhas e *issues*.

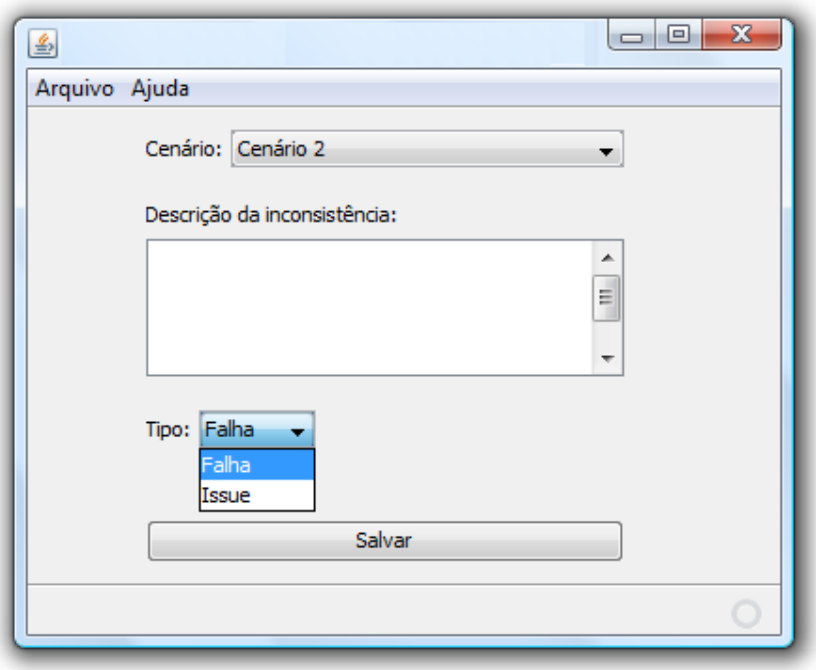

Figura 3.9 – Reportar falhas e issues

Os casos de uso de gerar planilha de execução e gerar relatório são ilustrados na Figura 3.6, uma vez que o que o usuário interage com a interface apenas selecionando a opção de gerar planilha e gerar relatório.

Descrição do minimundo

A ferramenta é composta por quatro entidades:

- Missão: representa as missões de testes e possui uma propriedade: descrição.
- Cenário: representa cenários de testes e possui duas propriedades: descrição e resultado.
- Funcionalidade: representa funcionalidades do software que está sendo testado e possui uma propriedade: descrição.
- Inconsistência: representa uma falha ou *issue* que foi encontrada durante o teste e possui três propriedades: descrição, tipo (se é uma falha ou se é uma *issue*) e gravidade.

As entidades relacionam-se da seguinte forma:

- Uma missão contém um ou mais cenários.
- Um ou mais cenários testam uma funcionalidade.

- Um cenário gera zero ou mais inconsistências.

Modelo ER

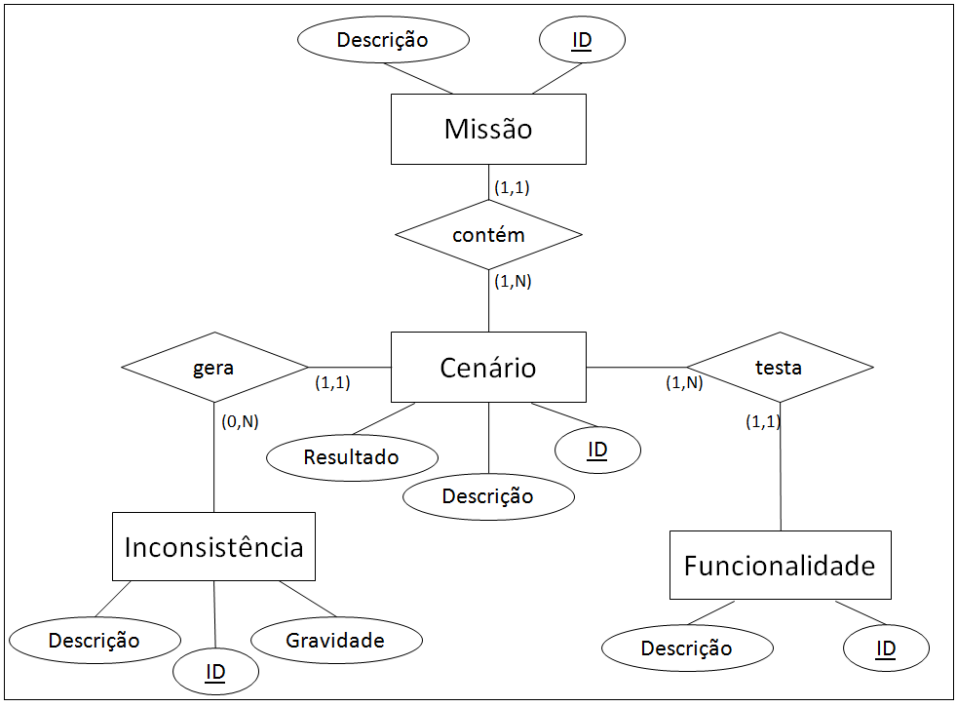

Figura 3.10 – Modelo Entidade-Relacionamento da ferramenta SMTE

Esquema conceitual em notação de diagrama de classe UML

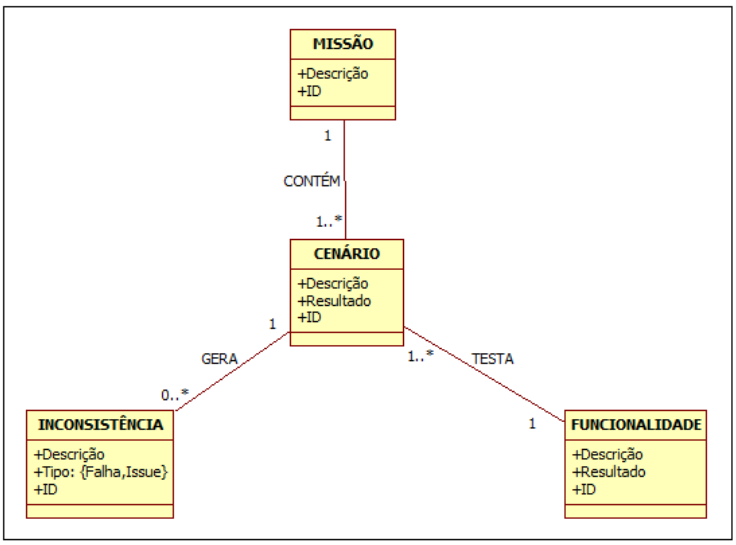

Figura 3.11 – Esquema conceitual da ferramenta SMTE

Esquema do banco de dados relacional da ferramenta SMTE

MISSÃO

ID_MISSÃO	DESCRIÇÃO

|-chave primária-|

CENÁRIO

ID_CENÁRIO	DESCRIÇÃO	RESULTADO	*ID_MISSÃO*	*ID_FUNCIONALIDADE*

|-chave primária-| |----------chave estrangeira ----------|

INCONSISTÊNCIA

ID_INCONSISTÊNCIA	DESCRIÇÃO	TIPO	*ID_CENÁRIO*

|----chave primária----| |--chave estrangeira--|

FUNCIONALIDADE

ID_FUNCIONALIDADE	DESCRIÇÃO	RESULTADO

|-chave primária-|

4 Conclusão e Trabalhos Futuros

Neste capítulo, serão apresentados as considerações finais, as principais contribuições e os possíveis trabalhos futuros.

4.1 Considerações Finais

Este trabalho de graduação abordou necessidades de testes de software para a obtenção de um produto de qualidade e, para isso, propôs uma metodologia e uma ferramenta para serem utilizadas no caso de os testes realizados em um produto serem do tipo testes exploratórios.

A fim de apresentar e definir a metodologia e a ferramenta propostas, foram introduzidos os conceitos básicos para o entendimento da metodologia proposta, e foi apresentado o estado da arte tanto de metodologias como de ferramentas existentes para dar suporte à realização de testes exploratórios.

Com base no estado da arte, a Metodologia para Testes Exploratórios foi aperfeiçoada, levando-se em consideração vários aspectos das metodologias existentes. Além disso, a ferramenta SMTE foi proposta neste trabalho, levando-se em consideração ferramentas desenvolvidas para dar suporte a testes exploratórios.

4.2 Contribuições

Este trabalho de graduação contribui com uma metodologia, disponível em http://www.cin.ufpe.br/~tds2/mte, e uma ferramenta de suporte a realização de testes exploratórios. A Metodologia para Testes Exploratórios proposta neste trabalho é resultado de um estudo profundo de metodologias já existentes. Dessa forma, os aspectos das metodologias estudadas que são considerados de grande utilidade para serem inseridos na metodologia aperfeiçoada, devem contribuir para uma melhor forma de execução de testes de forma exploratória. Além disso, para que essa forma seja ainda mais eficiente, foi especificada a ferramenta SMTE, que dá suporte à utilização dessa metodologia.

4.3 Trabalhos Futuros

Como trabalho futuro, é sugerido que o desenvolvimento da ferramenta SMTE seja concluído e que se realize um estudo de caso para que a ferramenta seja aperfeiçoada, objetivando adequar cada vez mais essa ferramenta às necessidades de seus usuários, os engenheiros de testes.

Referências

[1] **Müller, Thomas, et al.** Sobre o Quadro Internacional de Certificação de Teste de Software (ISTQB). *Site do Quadro Internacional de Certificação de Teste de Software (ISTQB).* [Online] 2007. http://www.istqb.org/downloads/syllabi/SyllabusFoundation.pdf.

[2] **Bach, James.** Sobre a Empresa: Satisface, INC. *Site da Satisface, INC.* [Online] 16 de Abril de 2003. http://www.satisfice.com/articles/et-article.pdf.

[3] **Bourque, P. e Dupuis, R.** *Guide to the Software Engineering Body of Knowledge (SWEBOK).* Los Alamitos, CA, Estados Unidos da América : s.n., 2004.

[4] **Santamaria, Marina Gil.** Sobre o site de recursos on-line StickyMinds.com. *StickyMinds.com.* [Online] 2007. http://www.stickyminds.com/getfile.asp?.

[5] **Elizondo, David Gorena.** Sobre site de recursos on-line StickyMinds.com. *StickyMind.com.* [Online] 2 de Outubro de 2008. [Citado em: 20 de Maio de 2009.] https://www.stickyminds.com/sitewide.asp?ObjectId=14514&Function=DETAILBROWSE&ObjectType=CP&sqry=*Z(SM)*J(MIXED)*R(relevance)*K(simplesite)*F(exploratory+testing%3A+next+generation)*&sidx=0&sopp=10&sitewide.asp?sid=1&sqry=*Z(SM)*J(MIXED)*R(relevance)*K(si

[6] **Viana, Virginia**. Um Método para Seleção de Testes de Regressão para Automação. Dissertação de Mestrado pelo Centro de Informática da UFPE. 2006.

[7] **Graham, Doroty, et al.** *Foundations of Software Testing: ISTQB Certification.* London : Thomson Learning, 2007. ISBN.

[8] **Bach, James.** Sobre a Empresa: Satisface, INC. *Site da Satisface, INC.* [Online] 16 de Abril de 2003. http://www.satisfice.com/articles/et-article.pdf.

[9] Sobre a Empresa: Satisfice Inc. *Site da Stisfice Inc.* [Online] [Citado em: 23 de Maio de 2009.] http://www.satisfice.com/sbtm/index.shtml.

[10] Sobre a Empresa: Satisfice Inc. *Site da Satisfice Inc.* [Online] [Citado em: 23 de Maio de 2009.] http://www.satisfice.com/articles/what_is_et.shtml.

[11] **Silva, Pedro Luciano Leite**. Um Processo para Seleção de Metodologias de Desenvolvimento de Software. Dissertação de Mestrado pelo Centro de Informática da UFPE. 2003.

[12] *Dicionário Priberam da Língua Portuguesa.* [Online] [Citado em: 23 de Maio de 2009.] http://www.priberam.pt/dlpo/dlpo.aspx.

[13] **Kaner, Cem, Bach, James e Pettichord, Bret.** *Lessons learned in software testing: a context driven approach.* New York : John Wiley & Sons, Inc., 2002. ISBN.

[14] **Bach, James**. *General Functionality and Stability Test Procedure*, 1999, http://www.satisfice.com/tools/procedure.pdf, [Online] [Citado em: 9 de Março de 2009.]

[15] **Rúbia, Diana**. Desenvolvendo uma Metodologia para Testes Exploratórios. Trabalho de Graduação pelo Centro de Informática da UFPE. 2007

[16] **TestExplorer**. http://testexplorer.com/ [Online] [Citado em: 17 de Maio de 2009.]

[17] **StickyMinds.com.** http://www.stickyminds.com/sitewide.asp?function=search&kind=simplesite&tt=SRCHBOX&tth=Y&freetext=exploratory+testing%3A+next+generation&submit.x=0&submit.y=0, [Online] [Citado em: 18 de Maio de 2009.]

[18] **Bach, Jonathan.** Sobre a Empresa: Satisfice Inc. *Site da Satisfice Inc.* [Online] Novembro de 2000. [Citado em: 27 de Maio de 2009.] http://www.satisfice.com/articles/sbtm.pdf.

[19] **Richardson, Alan**. Sobre a Empresa: Compendium Developments. *Blog Evil Tester*. [Online] Janeiro de 2009. [Citado em: 13 de Maio de 2009.] http://www.eviltester.com/index.php/2009/01/15/exploratory-test-assistant-a-tool-for-recording-your-exploratory-testing-notes/

[20] **D'Amorim, Marcelo**. Disciplina de Graduação do Centro de Informática da UFPE. Teste e Depuração de Software. Aula [Citado em: Junho de 2009.] http://www.cin.ufpe.br/~damorim/teaching/testing/2008.1/slides/01-o-que-eh-testes.ppt

[21] **Silva, Taíse**. Trabalho de Graduação em Ciência da Computação pela Universidade Federal de Pernambuco. http://www.cin.ufpe.br/~tds2/mte