ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

UMA NOVA ABORDAGEM PARA TESTES BASEADOS EM RISCOS NOS REQUISITOS DE SOFTWARE

Trabalho de Conclusão de Curso

Engenharia da Computação

Nome do Aluno: Andreza Maria Diniz Morais Soares
Orientador: Prof. Cristine Martins Gomes de Gusmão

Recife, novembro de 2007

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

UMA NOVA ABORDAGEM PARA TESTES BASEADOS EM RISCOS NOS REQUISITOS DE SOFTWARE

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Nome do Aluno: Andreza Maria Diniz Morais Soares
Orientador: Prof. Cristine Martins Gomes de Gusmão

Recife, novembro de 2007

UNIVERSIDADE
DE PERNAMBUCO

Andreza Maria Diniz Morais Soares

UMA NOVA ABORDAGEM PARA TESTES BASEADOS EM RISCOS NOS REQUISITOS DE SOFTWARE

Resumo

Com o mercado cada vez mais competitivo, as empresas têm que desenvolver produtos cada vez melhores, com alto nível de qualidade. No processo de desenvolvimento de qualquer produto, muitos são os riscos que ameaçam o bom andamento do projeto. Risco, para a Engenharia de Software, se refere à probabilidade de ocorrência de um evento adverso que, se ocorrer, pode trazer conseqüências drásticas ao projeto: falhas no sistema, por exemplo. O teste de software é uma prática indispensável, pois, grande é sua contribuição para a melhoria da qualidade do software, além de minimizar a ocorrência das falhas, detectando-as previamente. O objetivo dos testes de software é encontrar falhas no sistema, assim, já que os riscos podem gerar inúmeras falhas, é interessante focar os testes naquelas partes do sistema onde os impactos que os riscos podem trazer sejam mais danosos. Foi através dessa percepção que surgiu o *Risk-based Testing* – teste baseado em risco. Na Engenharia de Software tradicional, a fase de testes só é iniciada após toda a construção do sistema, no entanto, é fato que quanto mais próximos da sua origem os erros forem detectados, menor será o custo e a dificuldade em efetuar correções. Nesse contexto, esse trabalho propõe uma abordagem de testes baseados em riscos encontrados ainda no documento de requisitos, ou seja, dadas funcionalidades comuns a grande parte dos documentos de requisitos, é feita uma análise de riscos nos requisitos e casos de testes são criados para tentar mitigar, encontrar e solucionar possíveis falhas. Com essa nova abordagem, busca-se otimizar a realização de testes, pois se espera grandes benefícios relacionados a questões de custo, prazo e até mesmo, cultura organizacional.

Abstract

With the increasingly competitive market, companies must develop better products, with high quality level. In the process of development of any product, there are many risks that threaten the smooth progress of the project. Risk for Software Engineering, refers to the probability of occurrence of an adverse event, which, if happens, could bring drastic consequences to the project: failures in a system, for example. Software test is a vital activity, therefore, is its great contribution to improving software quality, and minimizes the occurrence of failures, detecting them previously. The purpose of testing is to find software flaws in a system. As risks may generate numerous flaws; it is interesting focus on testing those parts of a system in which the impacts of risks can be more harmful. Did to perception Risk-based Testing has emerged. In traditional Software Engineering, the testing phase begins after the whole construction of a system, however, this the fact is an origin of errors that are detected, lowering cost and difficulty of making corrections. In this context, this work suggests an approach for testing based on risks present in a document of requirements, that is, given common features a lot of the documentation of requirements, is done an analysis of risks in requirements and cases of tests are designed to trying to mitigate, find and resolve potential failures. With this new approach, is hoped great benefits related to issues of cost, time and even organizational culture.

Sumário

Índice de Figuras

Índice de Tabelas

Índice de Equações

Tabela de Símbolos e Siglas

ABGP – Associação Brasileira de Gerência de Projetos

APM – *Association to project management*

CBO - *Coupling Between Objects*

DIT - *Depth in Tree*

GUI – *Grafic Users Interface*

NASA – *Goddard Space Flight Center*

NOC - *Number of Children*

NOM – *Numbers of Methods*

PMBOK – *Project Management Body of Knowledge*

PMI – *Project Management Institute*

RFC - *The Response for a Class*

SATC – *Software Assurance Technology Center*

SEI – *Software Engineering Institute*

TDD – *Test Driven Development*

WMC – *The Weighted Methods per Class*

Agradecimentos

Gostaria de agradecer primeiramente a Deus, por ter colocado em minha vida pessoas tão especiais que me ajudaram a crescer e a concluir minha segunda graduação.

Dentre essas pessoas, gostaria de iniciar agradecendo aos meus pais, Aldemir e Auxiliadôra, por tudo que fazem por mim, por todo investimento na minha vida pessoal e profissional, por toda confiança que sempre depositam em mim e por todo amor e compreensão sem limites.

Queria agradecer também, a minhas irmãs, Andréa, Adriana e Amanda, por cada concessão que fizeram para me ajudar na conclusão desse trabalho, bem como pela torcida para que eu vença.

Ao meu sobrinho, Arthur, que mesmo sem entender, fez cada sorriso ser muito importante nos momentos difíceis.

Ao meu namorado, amigo, companheiro, Aldhir, por todo carinho, toda a força que me dá e por não esconder de ninguém o orgulho que tem de mim.

A minha orientadora, Cristine Gusmão, que me mostra a cada dia o tipo de profissional que eu quero ser.

Aos meus amigos, Thiago Trigo e Luciana Guerra, que tiveram uma participação fundamental em toda minha graduação, sempre dispostos a me ajudar.

Aos demais amigos da graduação, que muito me ensinaram e tornaram os últimos anos mais divertidos.

A todos os professores do curso de Engenharia da Computação da Universidade de Pernambuco, responsáveis pela minha formação como Engenheira, bem como pelo empenho em melhorar e dar visão ao nosso curso.

Por fim, a todos os outros familiares e amigos que torcem pelo meu sucesso.

Muito obrigada a todos.

Capítulo 1

Introdução

Durante as últimas décadas, a globalização da economia e os avanços nas tecnologias exigem a adaptação das empresas a novas realidades. Para encarar esses desafios, novos modelos de desenvolvimento têm sido projetados para criação de produtos. Essas abordagens são utilizadas como meios de obter redução de custos, melhor aproveitamento de tempo e de recursos e a melhoria da qualidade.

O desenvolvimento de software envolve uma série de atividades de produção nas quais a probabilidade de ocorrerem erros de origem humana é grande. Falhas podem acontecer em qualquer etapa do projeto. Por esta razão, o desenvolvimento de software deve ser acompanhado por atividades de testes que têm por objetivo garantir a Qualidade do Software.

O teste de software pode ser definido como o processo de executar um sistema de forma controlada com o objetivo de verificar se o mesmo se comporta conforme foi inicialmente especificado[1]. Os testes são indispensáveis para encontrar e remover defeitos e para avaliar o grau de qualidade de um produto e dos seus componentes**.**

Geralmente, devido à pressão para o cumprimento de prazos, os desenvolvedores não se sentem motivados para testar os seus desenvolvimentos e acabam por pular esta fase do processo de desenvolvimento.

É então, necessário utilizar metodologias para garantir uma maior abrangência, analisar e planejar os testes, visando otimizar o tempo gasto, possibilitando que este seja mais eficaz e de melhor qualidade**.**

Este capítulo introdutório tem a finalidade de apresentar a motivação para o desenvolvimento deste trabalho, na Seção 1.1; os objetivos que deverão ser alcançados, na Seção 1.2, a metodologia utilizada durante todo o desenrolar da pesquisa, na Seção 1.3; e a estrutura do documento na Seção 1.4.

1.1 Motivação

Risco, em um projeto de software, pode ser representado como uma medida da probabilidade e da perda relacionada à ocorrência de um evento negativo (desvio com relação aos requisitos inicialmente solicitados) que afete o próprio projeto, seu processo ou o seu produto [2], em outras palavras, qualquer coisa que possa acontecer e ameaçar o bom andamento do projeto é um risco.

Nesse contexto, é importante tomar ações preventivas que permitam um maior controle destes eventos considerados adversos. Assim, **por que não focar testes nas áreas onde o risco de ocorrer uma falha seja grande?**

A utilização de testes no apoio ao desenvolvimento de produtos de software é uma área bastante promissora, onde é utilizado um tipo de teste de software que toma por base a prioridade de componentes ou parte do sistema, que apresentam maior probabilidade de falhas. Assim surgiu o planejamento de testes de software baseados na análise de riscos – *Risk-based Testing.*

O *Risk-based Testing* pode ser considerado uma técnica que tem como foco analisar o software e realizar o planejamento dos testes a partir das áreas cuja probabilidade de ocorrer um problema ou uma falha, que causem grandes conseqüências, seja maior. Uma definição simplificada do processo de testes baseados em riscos, apresentada por Bach [3], pode ser encontrada abaixo:

- Fazer uma Lista dos Riscos por prioridade;
- Realizar testes que exploram os riscos;
- Controlar e acompanhar o estado dos riscos identificados, para ajustes do esforço de teste focando em tarefas atuais.

Uma pesquisa do Departamento de Comércio dos Estados Unidos da América, publicada em 2002, revelou que falhas de software são tão comuns e tão danosas que se estima que causem um prejuízo anual de mais de 60 bilhões de dólares para a economia americana. O estudo também alega que, embora não seja possível remover todos os erros, mais de um terço destes custos poderia ser eliminado caso se utilizasse uma infra-estrutura de testes melhor, que permitisse identificar e remover defeitos mais cedo e de forma mais eficaz. Atualmente, calcula-se que cerca de 50% dos defeitos são encontrados apenas nas fases finais dos projetos, ou após os sistemas começarem a ser utilizados em produção[4].

Nesse sentido, **por que não aplicar o *Risk-based Testing* nas fases iniciais do projeto?** A proposta desse estudo é desenvolver uma análise de riscos, no documento de requisitos do software, e em seguida, criar casos de testes que proporcionem a identificação, e

conseqüentemente a solução, de algumas falhas originadas pelos riscos, antes e durante o desenvolvimento do sistema.

1.2 Objetivos

Esse estudo tem como objetivo assessorar os planejadores de testes dando a eles uma nova maneira, mais econômica e eficiente, de efetuar seus testes, mais econômica, porque o esforço de teste inicia-se antes mesmo das primeiras linhas de código serem criadas; e mais eficiente porque foca os testes naquelas áreas do sistema onde é alta a probabilidade de ocorrerem falhas no sistema.

Assim, nosso outro objetivo, complementar ao primeiro é o desenvolvimento de uma abordagem que permita a criação de casos de teste tendo como base à análise de riscos no documento de requisitos.

Como é sabido, os testes têm como objetivo encontrar os erros, para que o produto desenvolvido seja de qualidade e com isso obter a satisfação do cliente. Além disso, os riscos, grosso modo, são tidos como qualquer evento que possa causar uma falha no software. Por isso a motivação de criar uma forma dos testes serem focados nas áreas do software onde a incidência de graves riscos seja forte.

1.3 Metodologia

Esse estudo pretende se enquadrar nas categorias de pesquisa estabelecidas pelo método científico. Uma pesquisa pode ser classificada em categorias, de modo que o pesquisador possa ajustar seus problemas específicos a cada classe e desenvolver seu trabalho com maior corretude. São elas: exploratória, descritiva e experimental.

Sendo assim, podemos classificar o presente estudo como sendo de caráter exploratório, na medida em que busca aprofundar e ampliar certos conhecimentos tidos como imprescindíveis à consecução dos objetivos enunciados nesse trabalho. Será realizado em várias fases, buscando em cada uma delas estabelecer estratégias que ajudem o desenvolvimento dessa proposta bem como o alcance dos objetivos elencados.

Fase 1 – Revisão Bibliográfica e Fundamentação Teórica

Definidos os objetivos, será realizada à revisão da literatura que buscará sintetizar os principais conceitos, tendências e abordagens referentes ao tema escolhido, ou seja, será feito um estudo minucioso a respeito dos assuntos tratados aqui e com isso será estabelecido todo o referencial teórico dessa dissertação, o que caracteriza a pesquisa exploratória. Os assuntos são: testes de

software, riscos de software e *Risk-based Testing* (testes baseados em riscos). Procurou-se realizar tal síntese, de maneira crítica, a partir dos estudos feitos em obras consideradas relevantes e representativas sobre o tema.

Em seguida, partiu-se para uma fundamentação teórica, que, basicamente, foi uma síntese final dos conceitos utilizados no decorrer do trabalho, subdivididos pelos principais temas abordados. Logicamente, a principal fonte desta etapa é a revisão de literatura.

Fase 2 – Definição do Escopo da pesquisa

De posse do conhecimento teórico podemos passar para a fase seguinte que tratará de definir o escopo do projeto. A primeira parte dessa definição consiste em estabelecer quais os requisitos genéricos serão utilizados na pesquisa, genéricos porque deve ser comum a grande parte dos documentos de requisitos. A segunda parte visa determinar um escopo de riscos para o estudo, ou seja, quais os riscos que podem ameaçar os requisitos.

Fase 3 – Criação e aplicação dos Casos de Teste

Estabelecido o escopo da pesquisa, parte-se para a fase prática. A primeira fase é fazer a análise dos riscos. Essa fase consiste em estabelecer para cada requisito, quais riscos o ameaçam. De posse desse estudo, devemos utilizar o modelo *inside-out* [3]. Esse modelo analisa os riscos e verifica as vulnerabilidades do sistema, as ameaças que podem disparar falhas e as vítimas das possíveis falhas. Para cada vulnerabilidade, casos de teste são criados. Em seguida, realizaremos a análise em um documento de requisitos para dar mais consistência ao método.

Fase 4 – Análise dos resultados

Os resultados, ou seja, os casos de teste obtidos nessa fase irão gerar a conclusão da pesquisa, que estabelecerá qual a relevância e contribuição da mesma para o assunto bem como indicará a viabilidade do trabalho.

1.4 Estrutura do Documento

Após este capítulo introdutório e para uma melhor explanação das ações realizadas para alcançar os objetivos, esse documento está disposto da seguinte maneira:

- Capítulo 2 – **Testes de Software** – este capítulo apresenta o conceito de teste de software bem como os níveis e as técnicas existentes. Além disso, ele procura esclarecer a importância de realizar testes e quais as fases que compõe a engenharia de testes.

- Capítulo 3 – **Testes *versus* Riscos** – este capítulo tem a finalidade de apresentar o conceito de risco bem como as principais classificações existentes. Em seguida, as várias técnicas de teste baseado em riscos são apresentadas.
- Capítulo 4 – **Uma nova abordagem para aplicação de *Risk-based Testing*** – este capítulo apresenta uma nova proposta para a utilização de testes baseados em riscos utilizando o documento de requisitos com a fonte de riscos e criando casos de testes para minimizar os efeitos das possíveis falhas.
- Capítulo 5 – **Conclusão e Trabalhos Futuros** – este capítulo tem como objetivo apresentar resultados e considerações finais sobre o modelo proposto, além de apresentar sugestões de trabalhos futuros na área.

Capítulo 2

Testes versus Riscos

Quando adquirimos um determinado produto o que mais exigimos é que ele realize sua função corretamente, satisfazendo nossas necessidades com qualidade. Na Engenharia de Software não é diferente. Quando um cliente solicita um software, ele quer um produto de qualidade, que atenda todas as suas expectativas, ou seja, ele quer um sistema que realize o que ele requisitou sem falhas. Para isso, é importante se dedicar um esforço maior no desenvolvimento para encontrar falhas e removê-las. É por isso que a fase de testes de software se torna indispensável.

Risco, em um projeto de software, pode ser representado como uma medida da probabilidade e da perda relacionada à ocorrência de um evento negativo que afete o próprio projeto, seu processo ou o seu produto [2], em outras palavras, qualquer coisa que possa acontecer e ameaçar o bom andamento do projeto é um risco.

Dentro desse contexto, esse capítulo destina-se a introduzir os conceitos de teste e de risco de software, mostrando os principais aspectos referentes à atividade de testes e as principais classificações de risco de software.

Para melhor contextualizar esse tema, a Seção 2.1 trará o conceito de teste de software, a mostrando as fases da engenharia de teste, os níveis de teste de software e as técnicas de testes existentes; a Seção 2.2 conceitua Risco de software e explana as principais classificações existentes e Seção 2.3 faz um breve resumo dos assuntos tratados nesse capítulo.

2.1 Teste de Software

Teste de software é uma das atividades de verificação e validação do produto desenvolvido. Para Hetzel [5], o objetivo principal dessa atividade é avaliar um componente, ou um recurso de um sistema.

Segundo Roger Pressman [6], a fase de testes, é aquela onde é executada uma série de atividades que consistem em avaliar o produto de software criado, através do código

desenvolvido e/ou das funcionalidades disponibilizadas, a procura de qualquer tipo de erro ou falha, que possa vir a interferir no correto funcionamento do aplicativo. O teste deve detectar o maior número de erros possível para que eles possam ser corrigidos antes da entrega final do produto.

Para realizar um teste, devemos inserir no sistema dados de entradas reais e comparar se os resultados obtidos são semelhantes aos esperados. Normalmente essas situações de verificação são chamadas “Casos de Testes” e são desenvolvidas na fase de análise dos requisitos do novo produto. Sendo verdade, podemos afirmar que essa parte do software não possui erros e que as respostas geradas são confiáveis. No entanto, o que realmente se deseja é que os testes encontrem falhas, caso isso não ocorra, pode ser que a elaboração dos casos de teste não tenha sido feita corretamente. Roger Pressman defende que os casos de testes não serão bem sucedidos se os mesmos não elevam a probabilidade de detecção de erros, bem como que um teste bem sucedido é aquele que aponta falhas ainda não constatadas com o mínimo de tempo e esforço[6].

2.1.1 A Engenharia de Teste

Na engenharia de software tradicional, a fase de teste é realizada após a fase de construção do sistema. No entanto, essa prática vem sendo modificada, pois se percebeu que quanto mais tarde os testes forem realizados, maiores os custos de uma mudança no código desenvolvido[7], como podemos observar na Figura 2.1.

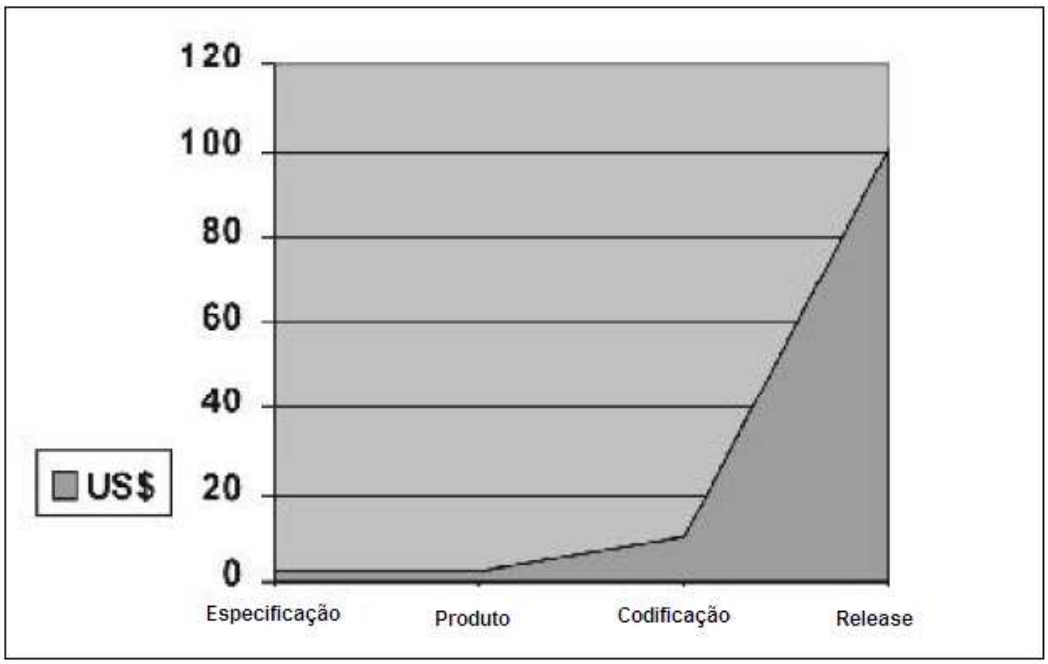

Figura 2.1 – Custo de solução de defeitos por fase do projeto [8]

Com isso, cada vez mais cedo se deve iniciar a fase de testes de um projeto.

Como definido anteriormente, teste de software é o processo de executar o software de uma maneira controlada com o objetivo de avaliar se o mesmo se comporta conforme o especificado. Assim, para esse controle ser eficiente, a fase de teste requer uma preparação cuidadosa que antecede sua execução, bem como uma análise eficiente dos resultados obtidos. Para isso, de uma forma geral, a engenharia de testes está dividida em fases, descritas a seguir:

- **Planejamento do teste -**

  O objetivo é a criação de um plano de teste. Nesse plano, as etapas e categorias de teste a serem conduzidos são definidas. No planejamento são identificadas as informações existentes no projeto e o escopo do software a ser testado, e contém: uma lista de requisitos a serem testados; regras e responsabilidades; estratégias, ferramentas e técnicas a serem utilizadas, os recursos necessários e um cronograma para as atividades de teste.

- **Projeto de Caso de Teste -**

  Essa etapa consiste em transformar os requisitos do sistema, bem como o comportamento esperado na aplicação de cada um deles, em casos de testes especificados e documentados.

- **Desenvolvimento do Teste -**

  Após a documentação dos testes, devemos “tirá-los do papel” e transformá-los em programas, “scripts”, que irão testar exaustivamente o sistema em desenvolvimento. Existem ferramentas próprias para isso que devem ser usadas.

- **Execução do Teste -**

  Consiste simplesmente na execução dos scripts e envia os resultados para análise.

- **Análise dos Resultados -**

  Os resultados obtidos na execução dos testes são confrontados com os valores esperados. Os resultados em não conformidade deverão ser confirmados e detalhados para que as correções necessárias sejam realizadas.

- **Encerramento do Processo -**

  Essa etapa é caracterizada pela avaliação de todo Processo de Teste, comparando os resultados alcançados com o que foi inicialmente planejado. Neste momento, indicadores são extraídos, com o objetivo de avaliar qualitativamente e quantitativamente o desempenho dos testes, utilizando projetos anteriores como base para a comparação.

A engenharia de teste contribui para melhorar a efetividade e eficiência das atividades de teste em um projeto. A melhoria é alcançada através de etapas bem formuladas e implantadas.

2.1.2 Níveis de Testes de Software

O teste de software pode ser realizado em diferentes níveis durante todo o processo de desenvolvimento do software. Como podemos observar na Figura 2.2, cada teste está ligado a um nível arquitetural do sistema.

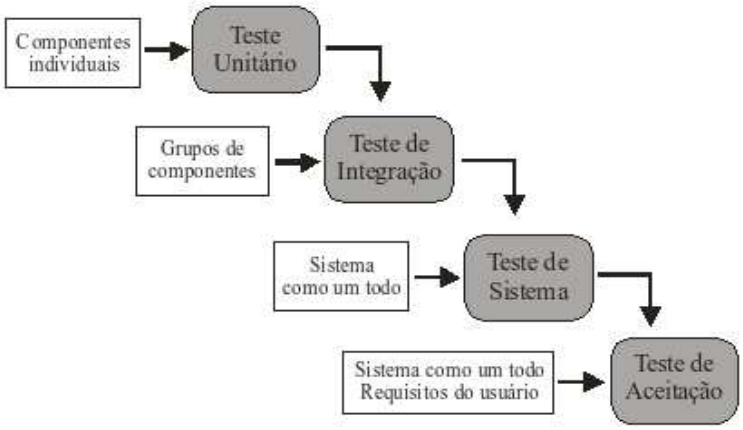

Figura 2.2 – Níveis de Teste [9]

De acordo com a Figura 2.2, pode-se notar que:

- **Teste Unitário -**

  Essa etapa do processo de teste é responsável por verificar as pequenas partes do software desenvolvido, ou seja, se concentra na verificação da menor unidade do projeto de software. Tem como alvo: uma função ou um *procedure*, em um sistema procedural; uma classe ou um método, em um sistema orientado a objeto; ou qualquer trecho pequeno de código.

- **Teste de Integração -**

  É uma técnica sistemática para formar a estrutura do programa, realizando testes para descobrir falhas provenientes da integração interna dos componentes de um sistema. A integração entre sistemas distintos não faz parte dessa etapa de teste. Essas interfaces são testadas na fase seguinte, a de teste de sistema.

- **Teste de Sistema -**

  Essa fase de teste busca testar o sistema de um modo geral, a partir da visão do usuário final. Todas as funcionalidades são testadas em busca de falhas. Assim, os testes são executados em condições reais e semelhantes às que o usuário final utilizará em seu cotidiano ao manipular o sistema.

- **Teste de Aceitação** -

  É a fase de teste realizada pelo usuário final, que simula operações rotineiras do sistema para verificar se as funcionalidades do software estão de acordo com o que fora solicitado. A validação é bem sucedida quando o software funciona de uma maneira razoavelmente esperada pelo cliente [6].

Além desses quatro níveis mais comuns de teste de software, temos ainda mais dois níveis que podem ser inseridos no processo de testes.

- **Teste de Operação** -

  Fase do teste conduzida pelos administradores do ambiente onde o sistema será utilizado. São feitas simulações para garantir que o sistema entrará em produção de maneira bem sucedida. Essa etapa de testes é útil quando um sistema irá substituir outro, para garantir que o novo sistema dará suporte às atividades do anterior.

- **Teste de Regressão -**

  Essa fase de teste é aplicada quando algum novo componente é inserido no sistema, ou quando alguma modificação nos componentes existentes é realizada e apresentam defeitos ao se juntar com o restante do sistema já testado. Quando isso ocorre, diz-se que o sistema regrediu. [10]

2.1.3 Técnicas de Teste de Software

Como já visto na seção 2.2, os testes devem ser cuidadosamente desenhados e planejados. Para isso, existem algumas técnicas de software que auxiliam a construção dos testes. As duas técnicas padrão são o teste de Caixa-preta (funcional) e o teste de Caixa-branca (Estrutural), descritos a

seguir. Além desses, Bartié [11] enumera mais algumas técnicas importantes, analisadas nas próximas subseções.

- **Teste Funcional**

O teste funcional, também chamado de teste de Caixa-Preta, busca verificar o sistema e seu processo interno pela sua interação através da Interface Gráfica do Usuário (GUI – *Graphical User Interface*) e da análise das saídas ou resultados, ou seja, não se considera o comportamento interno do sistema. [6]. Como podemos observar na Figura 2.3, dados de entrada são fornecidos ao sistema, o teste é executado e o resultado obtido é comparado a um resultado esperado previamente conhecido.

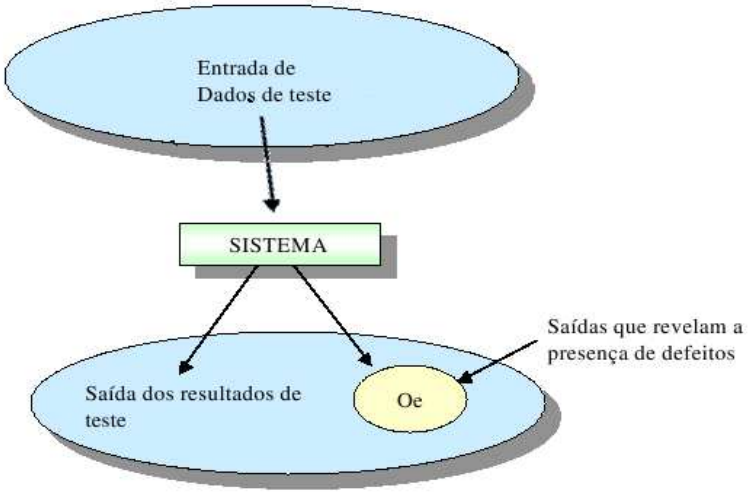

Figura 2.3 – Teste Caixa-Preta

A técnica de teste de Caixa-Preta é aplicável a todas as fases de teste – fase de teste de unidade (ou teste unitário), de integração, de sistema e de aceitação.

- **Teste Estrutural**

O teste estrutural, também conhecido como teste de Caixa-Branca, tem por objetivo determinar defeitos na estrutura interna do produto por meio do desenho de testes que exercitem suficientemente os possíveis caminhos de execução. [8]. Como o testador tem acesso ao código fonte da aplicação, esse tipo de teste é desenvolvido elaborando-se casos de teste que cubram todas as possibilidades de um componente de software, ou seja, dado o código de um determinado componente, casos de teste são derivados de acordo com a estrutura do programa, e aplicados diretamente ao código, como podemos observar na Figura 2.4.

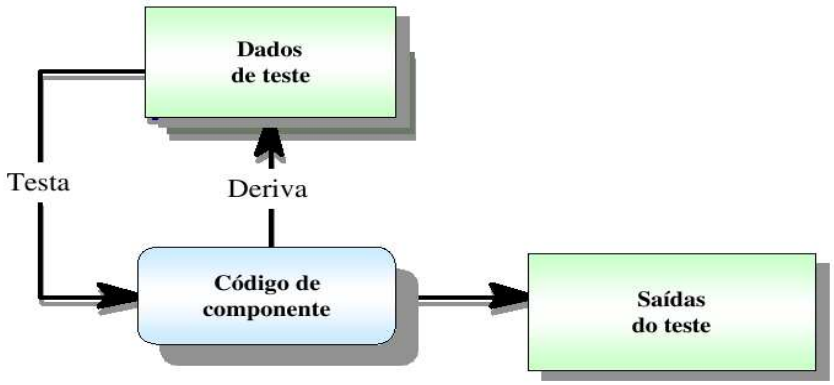

Figura 2.4 – Teste Caixa-Branca

- **Teste de Usabilidade**

Essa técnica visa detectar problemas de interface ou problemas que tornam o software pouco intuitivo. Nesse tipo de teste, os usuários reais do sistema são incentivados a utilizar o sistema em um ambiente monitorado, para verificar a facilidade de uso da interface em questão.

- **Teste de Volume e Carga**

O teste de Volume e Carga é realizado com o intuito de avaliar como o software reage a uma grande quantidade de transações. Então, o teste submete excessivas quantidades de dados ao sistema para determinar se limites que causam a falha do software são alcançados. Ele também identifica a carga ou volume máximo persistente que o sistema pode suportar por um dado período.

- **Teste de Ambiente**

Esse teste é realizado para avaliar como o software reage a diversos hardwares e sistemas operacionais diferentes, com o intuito de determinar as configurações de hardware e software necessárias.

- **Teste de Segurança**

O teste de segurança tenta verificar se todos os mecanismos de proteção embutidos num sistema o protegerão, de verdade, de acessos indevidos. Para realizar esse teste, o testador deve tentar penetrar no sistema de qualquer maneira, descobrindo, assim, falhas de segurança que possam comprometer o sigilo e a fidelidade das informações.

- **Teste de Compatibilidade**

Utilizado para assegurar a compatibilidade entre outros componentes do software, entre sistemas e aplicações. Por exemplo, o teste para assegurar compatibilidade de um sistema *web* que rode em diferentes *browsers*, em diferentes sistemas operacionais e diferentes plataformas de hardware[12].

- **Teste de Performance**

Este tipo de teste determina se o desempenho atende os requisitos. Ele identifica onde residem os gargalos dos sistemas, se é na própria aplicação ou na infra-estrutura em que ela está implementada. Geralmente envolve teste de automação, facilitando a simulação de uma série de cenários como normal, picos e condições de carregamentos extraordinários.

- **Teste de Instalação**

O teste de instalação é utilizado para garantir que o software possa ser instalado sob condições apropriadas e, verificar que, uma vez instalado, o software funciona corretamente, ou seja, serve para validar o procedimento de instalação da aplicação.

- **Teste de Confiabilidade e Disponibilidade**

Essa técnica de teste busca avaliar se o software opera satisfatoriamente, sem ocorrência de falhas, durante um período especificado de tempo em um determinado ambiente, identificando interrupções no uso do software e o tempo para a resolução do problema.

- **Teste de Recuperação**

O teste de recuperação é realizado forçando o sistema a falhar de diversas maneiras e verificando se a recuperação é adequadamente executada. Busca avaliar o comportamento do software após uma condição anormal.

- **Teste de Contingência**

Esta categoria de testes visa validar os procedimentos de contingência a serem aplicados à determinada situação prevista no planejamento do software[11]. Seu objetivo é simular os cenários de contingência e avaliar a precisão dos procedimentos.

2.2Riscos: Conceito e Classificações

Na literatura, encontramos a palavra risco utilizada pelas mais diversas áreas de conhecimento: nos negócios (riscos de negócios), na sociedade (riscos sociais), na economia (riscos econômicos), na segurança (riscos de segurança), na política (riscos políticos), na saúde (risco de morte, ou de epidemias), entre outros.

Para a Gerência de Projetos, o risco está diretamente ligado aos impactos negativos ou positivos que ele pode causar. Podemos verificar isso nos diversos conceitos encontrados na bibliografia da área.

Segundo o *Project Management Institute* – PMI, "Risco é um evento ou condição incerta que, se ocorrer, tem um efeito positivo ou negativo sobre ao menos um dos objetivos do projeto"; a *Association for Project Management* – APM, define risco como "a combinação da probabilidade ou freqüência de ocorrência de uma ameaça ou oportunidade definida e a magnitude das conseqüências de sua ocorrência"; e, para a Associação Brasileira de Gerenciamento de Projetos – ABGP, "riscos são acontecimentos com impacto negativo (prejuízos ou danos) ao sucesso geral do projeto, ou são eventos que podem causar prejuízos que não puderam ser previstos".

Uma definição mais completa é dada por Aguiar, onde o risco em um projeto de software pode ser representado como uma medida da probabilidade e da perda relacionadas à ocorrência de um evento negativo que afete o próprio projeto, seu processo ou o seu produto[2]. Em outras palavras, qualquer coisa que possa acontecer e ameaçar o bom andamento do projeto - um problema em potencial, o fracasso de um ou mais componentes: prazo, custo, atividades - é um risco. Não é certo que ele ocorrerá, mas se ocorrer, poderá trazer prejuízos.

Por se tratar de um evento inesperado, normalmente visto negativamente, a mitigação de riscos passou a ser objetivo da gerência de projetos e a gestão dos riscos uma disciplina fundamental para conseguir tal objetivo.

De acordo com o Guia PMBOK [13] a Gerência de Riscos é tida como o processo para identificar, analisar, planejar ações que venham a minimizar os efeitos que os riscos podem causar, controlar e monitorar os riscos em um projeto. Sendo assim, a Gerência de Riscos visa aumentar a probabilidade de ocorrência e os impactos de eventos positivos e diminuir a probabilidade e os impactos dos eventos adversos aos objetivos do projeto [14], ou seja, um risco tem uma causa, e se ele ocorrer, gera uma conseqüência que pode ser positiva ou não. E não

sendo positivo, os passos da Gerência de Riscos são importantes por dar um tratamento adequado às incertezas e promover a melhoria dos processos de software.

2.2.1 Classificações

Os riscos podem ser classificados quanto a vários aspectos. Sua classificação permite estabelecer técnicas para identificar os riscos de forma sistemática, mas não garante resultados, pois nem todos os riscos podem ser identificados. As classificações mais conhecidas estão descritas a seguir.

1. Quanto à natureza

A primeira classificação, senão a mais utilizada, diz respeito à essência dos riscos, ou seja, em qual área o risco está inserido, ou melhor, qual área ele afeta diretamente. Nesta classificação encontramos as seguintes categorias:

- **Riscos de Projeto**

Essa categoria de risco engloba os eventos que, caso venham a ocorrer, comprometem ou impedem a realização de um dado projeto, ou seja, problemas com orçamento, cronograma, pessoal, recursos, cliente e requisitos.

Como exemplos de riscos de projeto podemos citar:

- Má definição dos requisitos do projeto;
- Não entendimento do problema do cliente;
- Doença em algum membro da equipe que impeça a participação do mesmo no projeto;
- Afastamento definitivo, ou até mesmo falta, de pessoal com conhecimentos necessários para o projeto, dentre outros.

- **Riscos de Produto (ou Riscos Técnicos)**

São os riscos que afetam a qualidade ou desempenho do software que é desenvolvido, ou seja, problemas com implementação, interface, ambigüidade de especificação, tecnologias de ponta, implantação, verificação e manutenção.

Para exemplificar, segue abaixo alguns riscos técnicos:

- Modificações nas funcionalidades do projeto pelo usuário, causando atrasos no cronograma;

- Falta de atualizações no sistema por parte dos desenvolvedores;
- Impossibilidade ou dificuldade na implementação do projeto por falta de conhecimento com as tecnologias utilizadas;
- Problemas nas máquinas (hardware) utilizadas no projeto, dentre outros.

➢ **Riscos de Negócios**

São aqueles que afetam a organização que está desenvolvendo ou adquirindo o produto, ou seja, problemas com o mercado, com as estratégias da empresa, com a gerência e com orçamentos.

A seguir, alguns exemplos:

- Perda do compromisso orçamentário;
- Capacitação não suficiente aos usuários do produto;
- Um mau gerente de projetos;
- Perda do compromisso do pessoal;
- Possibilidade de fazer um produto não adequado ao mercado, dentre outros.

2. Quanto à probabilidade do evento

Outra classificação encontrada se refere à probabilidade de ocorrência de um evento pertinente a um determinado risco. Nesse contexto, os riscos podem ser classificados nas categorias descritas nas seções subseqüentes.

➢ **Riscos Conhecidos**

São aqueles riscos identificados diretamente após avaliações no plano de negócio, nas condições técnicas e comerciais, por exemplo, escopo mal definido ou prazos não condizentes com a realidade.

➢ **Riscos Previsíveis**

Constituem os riscos determinados pela experiência com outros projetos passados, ou seja, graças à experiência do gerente de projetos (riscos), ele é capaz de prever possíveis riscos sem que eles sejam encontrados diretamente. Como exemplo desses riscos, temos: ruídos na comunicação com os usuários; mudanças nas equipes de trabalho, seja por doença ou por motivos particulares a cada membro; dentre outros.

➢ **Riscos Imprevisíveis**

Existem riscos que são praticamente impossíveis de serem determinados antecipadamente. Esses riscos são incluídos nessa categoria.

2.2.2 Técnicas e Métodos de Identificação de Riscos

De uma forma geral, observa-se na literatura que os autores consideram a fase de identificação de risco como uma das mais importantes em todo processo do gerenciamento de risco, pois apresenta um impacto maior na acuracidade das avaliações de risco, já que a forma como os riscos são identificados e coletados constituem-se na questão central para a efetividade de todo este processo[15]. Segundo o Guia PMBOK [13], a fase de identificação de risco compreende a determinação de quais riscos podem afetar o projeto e em documentar as suas características.
Observa-se através do levantamento bibliográfico realizado a existência de várias técnicas propostas para a identificação de riscos em projetos. A seguir, serão especificadas de forma resumida, as técnicas de identificação de risco mais usuais em projetos:

1.***Brainstorming***

Técnica de geração de idéias em grupo dividida em duas fases: a primeira, onde os participantes apresentam o maior número possível de idéias; e a segunda, onde cada participante defende sua idéia com o objetivo de convencer os demais membros do grupo. Na segunda fase são filtradas as melhores idéias, permanecendo somente aquelas aprovadas pelo grupo.

2.Técnica de *Delphi*

Delphi é uma técnica para a busca de um consenso de opinião de um grupo de especialistas a respeito de eventos futuros. Baseia-se no uso estruturado do conhecimento, da experiência e da criatividade de um painel de especialistas, pressupondo-se que o julgamento coletivo, quando organizado adequadamente, é melhor que a opinião de um só indivíduo. Esta técnica de criação de consenso utiliza respostas escritas ao invés de reunir pessoalmente os membros do grupo, ou ainda método para a sistemática coleta e comparação crítica de julgamentos, de participantes anonimamente isolados, sobre um tópico, através de um conjunto de questionários cuidadosamente desenvolvidos, intercalados com informações sumarizadas e “feedback” das opiniões, derivadas das respostas anteriores.

3.*Checklists*

Consiste em uma lista de itens, que vão sendo marcados como sim ou não, podendo ser utilizada por um membro da equipe, em grupo ou em uma entrevista.

4. Entrevistas

Entrevistas livres, semi-estruturadas ou estruturadas, conduzidas individualmente ou em grupo com membros experientes do projeto, envolvidos ou especialistas.

5. Técnica de Grupo Nominal

A técnica de grupo nominal foi elaborada para ser utilizada na área de planejamento, com o objetivo de ampliar a produção criativa do grupo, facilitar as decisões em equipe, estimular a geração de idéias críticas e servir como instrumento de agrupamento de idéias. Assim sendo, esta técnica corresponde à geração silenciosa de idéias escritas; exposição das idéias geradas ao grupo na forma de frases simples em cartões ou tiras de papel; discussão de cada idéia registrada para esclarecimento e avaliação; votação individual das idéias em ordem de prioridade, com a decisão do grupo sendo trabalhada matematicamente através da classificação por quantidade de votos obtidos ou ordenação por ordem de prioridade.

2.2.3 Taxonomia de Riscos de Projetos de Software

Antes de apresentar uma taxonomia de riscos é importante diferenciar taxonomia de classificação. Taxonomia é a ciência responsável por descrever, nomear e classificar coisas em categorias mutuamente exclusivas; e a classificação é a ação de atribuição de entidades às categorias definidas na taxonomia, ou seja, agrupamento de itens semelhantes, tomando por base critérios estabelecidos[16]. Para a taxonomia de riscos, a classificação irá atribuir riscos dentro das categorias propostas.

Dentre as várias taxonomias de riscos já criadas, será utilizada, nesse trabalho, aquela elaborada pelo *Software Engineering Institute* – SEI. Segundo o SEI, os riscos são originados por problemas que acontecem no projeto e no processo devido a ações da gerência; e por problemas no processo (desenvolvimento) produtivo do software e no produto graças a ações técnicas, como mostra a Figura 3.1.

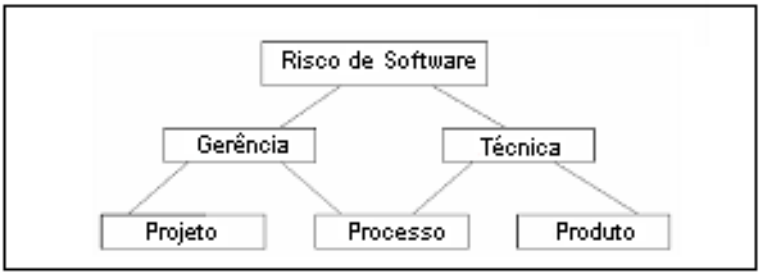

Figura 3.1 – Origem dos Riscos de Software[17]

A taxonomia de riscos da SEI organiza os riscos de software em classes, cada classe está subdividida em elementos que por sua vez são divididos em atributos, como podemos ver na Figura 3.2:

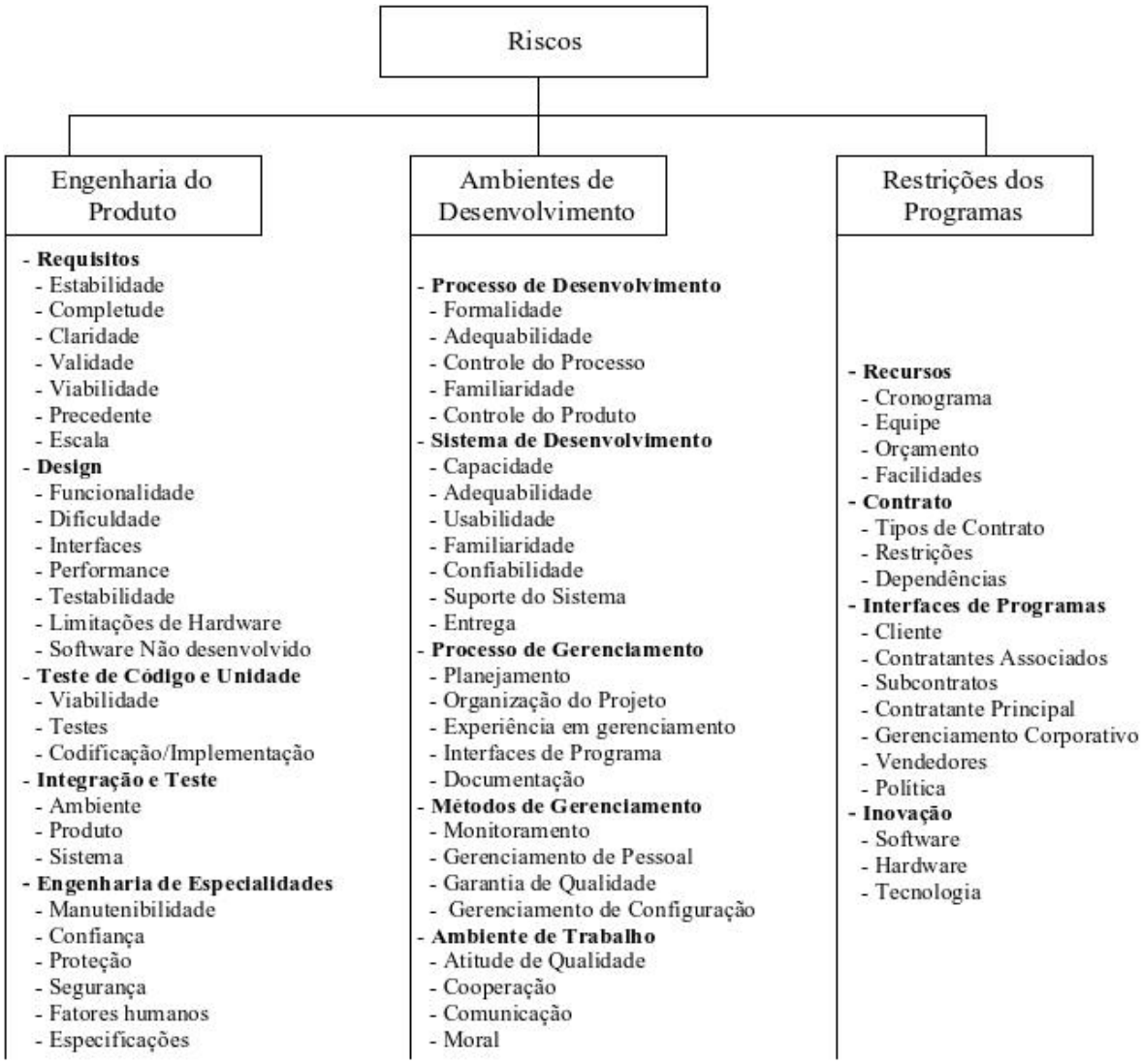

Figura 3.2 – Taxonomia de Riscos [18]

As três classes que compõem a taxonomia são:

- **Engenharia do produto** – Esta classe engloba todos os riscos encontrados no processo de desenvolvimento de um sistema, levando-se em consideração aspectos técnicos e físicos que satisfaçam as expectativas dos clientes. Nessa classe encontramos os riscos pertinentes aos seguintes elementos: requisitos, design, teste de código e de unidade, integração e teste e a engenharia de especialidades.
- **Ambientes de Desenvolvimento** – Essa classe contempla os riscos relacionados com as estratégias, os métodos, os procedimentos e as ferramentas utilizados na produção do software, ou seja, se refere ao ambiente onde o software será desenvolvido.

  Os riscos analisados aqui pertencem aos seguintes elementos: processo de desenvolvimento, sistema de desenvolvimento, processo de gerenciamento, métodos de gerenciamento e ambiente de trabalho.
- **Restrições do Programas –** Esta classe se refere aos riscos relativos a fatores externos do projeto. São fatores que mesmo fora de controle do projeto, influenciam o sucesso do mesmo. Podemos considerar aspectos contratuais, operacionais e organizacionais como fontes desse tipo de risco. Os elementos analisados nessa classe são: recursos, contratos, interface dos programas e inovação.

A partir dessa refinada classificação, fica muito mais fácil administrar os riscos, prevendo as possíveis falhas e propondo soluções que amenizem seus efeitos.

Existe uma técnica de teste, ainda pouco diversificada, denominada teste baseado em riscos. Esse estudo tem como alvo principal essa técnica e o próximo capítulo se destina a explanar as teorias já existentes sobre esse assunto.

2.3 Resumo do Capítulo

Este capítulo tratou de introduzir o conceito de teste de software, ressaltando sua importância para a engenharia de software bem como sua ligação direta com a qualidade do produto desenvolvido. Vimos também que podemos realizar testes em qualquer magnitude arquitetural, desde componentes individuais até sistemas inteiros. Outro assunto tratado é a apresentação das técnicas de testes de software existentes, que unidas, têm a capacidade de encontrar falhas ligadas a qualquer aspecto do sistema. Além de testes, o capítulo tratou de conceituar riscos, mostrando

as principais classificações existentes, principalmente a taxonomia de riscos, que subsidiou esse estudo.

Capítulo 3

Testes Baseado em Riscos

Ao se iniciar qualquer projeto, é muito importante a preocupação com eventos futuros. Vários são os problemas que podem ocorrer: as necessidades dos usuários não serem atendidas; problemas técnicos com as máquinas; mudanças indesejadas nas equipes de trabalho; problemas com o prazo de entrega dos artefatos, que podem vir a causar grandes perdas e danos ao projeto. Porém, junto com essas possíveis ocorrências, temos um produto final, como objetivo, que precisa ser concluído e com isso, vários benefícios estão associados.

Assim, é importante tomar ações preventivas que permitam um maior controle destes eventos considerados adversos. A utilização de testes no apoio ao desenvolvimento de produtos de software é uma área bastante promissora, onde é utilizado um tipo de teste de software que toma por base a prioridade de componentes, ou parte do sistema, que apresentam maior probabilidade de falhas.

Esse capítulo destina-se a relacionar testes de software com riscos. Essa junção dá origem ao *Risk-based Testing* - testes baseados em riscos, objeto principal do presente estudo. Para melhor contextualizar esse tema, a Seção 3.1, tratará de definir teste baseado em risco, mostrando e mostrará um pouco dos diversos estudos realizados nessa área que estimularam o desenvolvimento dessa pesquisa.

3.1 Introdução

É fato que quando você implementa um sistema o importante é que ele funcione, mas como garantir isso? Testar é fundamental para desenvolver software de alta qualidade e garantir a tranqüilidade das operações, pois, as conseqüências que algumas falhas podem causar são terríveis. Além disso, testes trazem benefícios como a economia de custos e de tempo. E por que não priorizar os testes nos componentes ou nas partes do sistema que apresentam maior probabilidade de falha? Essa é a idéia do *Risk-based Testing* – teste baseado em riscos.

O *Risk-based Testing* pode ser considerado como uma técnica de teste que tem como foco analisar o software e realizar o planejamento dos testes a partir das áreas cuja probabilidade de ocorrer um problema ou uma falha, que causem grandes conseqüências, seja maior. Uma definição simplificada do processo de testes baseados em riscos pode ser encontrada abaixo[3]:

1. Fazer uma Lista dos Riscos por prioridade;
2. Realizar testes que exploram os riscos;
3. Controlar e acompanhar o estado dos riscos identificados, para ajustes do esforço de teste focando em tarefas atuais.

3.2 Principais Abordagens de Testes Baseados em Riscos

Para esta técnica, vários autores desenvolveram estudos, criando sua própria abordagem de teste baseado em risco. A seguir, descreveremos as principais técnicas existentes.

3.2.1 Análise de Riscos Baseada em Heurística

Bach [3] criou duas abordagens distintas para identificação dos riscos a serem explorados pelo teste, baseadas em heurísticas, ou seja, baseada em experiências anteriores. Uma delas é chamada *Inside-out* (de dentro pra fora) e a outra é *Outside-in* (de fora pra dentro). É importante ressaltar que dentre todas as abordagens de *Risk-based Testing* desenvolvidas, o estudo de Bach é o único que leva em consideração não só a análise de riscos que será alvo dos testes, mas também, a maneira como o esforço de teste deve se organizar.

⇒ ***Inside-out***

Essa abordagem consiste em analisar o produto do software de dentro para fora, levantando para cada componente ou funcionalidade as seguintes questões:

- Vulnerabilidades – quais fraquezas ou possíveis falhas podem ocorrer?
- Ameaças - Que entradas ou situações poderiam existir que podem explorar uma vulnerabilidade e disparar uma falha neste componente?
- Vítimas: Quem ou o que poderia ser impactado por uma possível falha e quão ruim isto seria?

A partir destas informações, os casos de teste são gerados. O levantamento das vítimas não auxilia a criação dos casos de teste, ele só ajuda a entender o quão grave foi a falha para uma determinada vítima e assim julgar se é necessário ou não escrever um caso de teste para aquela situação.

⇒ ***Outside –in***

Essa abordagem consiste em, a partir de um conjunto de riscos em potencial, relacioná-los aos detalhes da situação. Uma lista pré-definida de riscos é consultada e em seguida determinam-se quais riscos são aplicáveis naquele momento, naquela situação. Essa lista pode ser baseada em experiência própria ou em listas já existentes. Bach sugere três tipos de listas:

- **Lista de Categorias de Critérios de Qualidade** – essa lista possui categorias desenvolvidas para atenderem a diferentes tipos de requisitos: capacitação, confiabilidade, usabilidade, desempenho, instabilidade, compatibilidade, suportabilidade, testabilidade, manutenibilidade, portabilidade e localizabilidade.
- **Lista Genérica de Riscos** – essa lista possui riscos universais a qualquer sistema: complexo, novo, modificado, dependência em componente superior, dependência em componente inferior, crítico, preciso, popular, estratégico, terceirizado, distribuído, defeituoso e falhou recentemente.
- **Catálogos de Risco** – são listas de riscos que pertencem a um domínio em particular.

⇒ **Como organizar Testes baseados em Riscos**

Bach [3] propõe também, três maneiras de organizar os testes tendo por base os riscos, são elas:

- A lista de observação de riscos é a mais simples. Consiste em uma lista de riscos que você periodicamente revê e se pergunta o que os testes revelaram sobre os mesmos.
- A matriz risco/tarefa consiste em uma tabela cuja primeira coluna possui uma lista de riscos, ordenados por importância, e a segunda coluna possui uma lista de tarefas de mitigação associada a cada risco.
- A matriz componente/risco é uma tabela com três colunas: primeira indica o componente do sistema, a central o grau de severidade do risco e a ultima a(s) heurística(s) que expõem o risco.

3.2.2 Testes Baseados em Riscos para Sistemas Orientados a Objetos

Essa técnica propõe a utilização de uma metodologia para identificação das classes de um sistema orientado a objeto, cuja probabilidade de falha seja maior, ou seja, aquelas que apresentam maior risco de falha. Rosenberg, Stapko e Gallo [19], dizem que foi comprovado que o código mais complexo é o que apresenta uma maior incidência de erros. Assim, métricas de complexidade de softwares orientados a objetos foram utilizadas para se chegar aquelas classes mais propensas a falhas.

Os autores defendem a utilização de seis métricas de medição, propostas e aplicadas pelo *Software Assurance Technology Center* (SATC) da NASA *Goddard Space Flight Center*. São elas:

- **Número de métodos** (NOM – *Numbers of Methods*): contagem dos métodos distintos existentes em uma classe.
- **Número Ponderado de Métodos por Classe** (WMC - *The Weighted Methods per Class*): soma ponderada dos métodos em uma classe. Os pesos são determinados pela complexidade de cada método.
- **Acoplamento entre objetos** (CBO - *Coupling Between Objects*): contagem do número de outras classes para o qual uma classe está acoplada.
- **Resposta a uma classe** (RFC - *The Response for a Class*): é o número total de métodos que um objeto de uma determinada classe pode chamar tanto dela mesma quanto de outras classes, ou seja, é o número de métodos que podem ser executados em resposta a uma mensagem recebida por um objeto de uma classe.
- **Profundidade na árvore** (DIT – *Depth in Tree*): é o número de saltos saindo de uma classe até a raiz da hierarquia de classes e é medida pelo número de ancestrais na classe. Quando existe mais de uma herança, deve-se utilizar a maior DIT.
- **Número de Filhos** (NOC – *Number of Children*): Número de subclasses (filhos) que herdam diretamente da classe na hierarquia.

Deve-se utilizar, pelo menos, duas ou três métricas para dar uma indicação clara de problemas em potencial. Portanto, para cada projeto, o SATC cria uma tabela de classes que possuem alto risco. Alto risco é identificado com uma classe que tem ao menos duas métricas que excedem os limites recomendados[19]. Os valores limites foram derivados do estudo da distribuição das métricas coletadas pelo SATC, que há três anos coleta e analisa código orientado a objeto.

3.2.3 Teste Baseado em Riscos para E-Business

Esse estudo, realizado por Gerrard [20], propõe um modelo de teste baseado nos cinco principais riscos para negócios eletrônicos: usabilidade, desempenho, segurança, disponibilidade e funcionalidade.

Seu método de teste se diferencia dos demais pela execução de testes em todas as fases do projeto, como podemos ver na Figura 3.3, proposta pelo autor. O modelo W promove a idéia de que para toda atividade que gera um artefato que pode ser entregue, cada um destes artefatos deve ter uma atividade de teste associada[20].

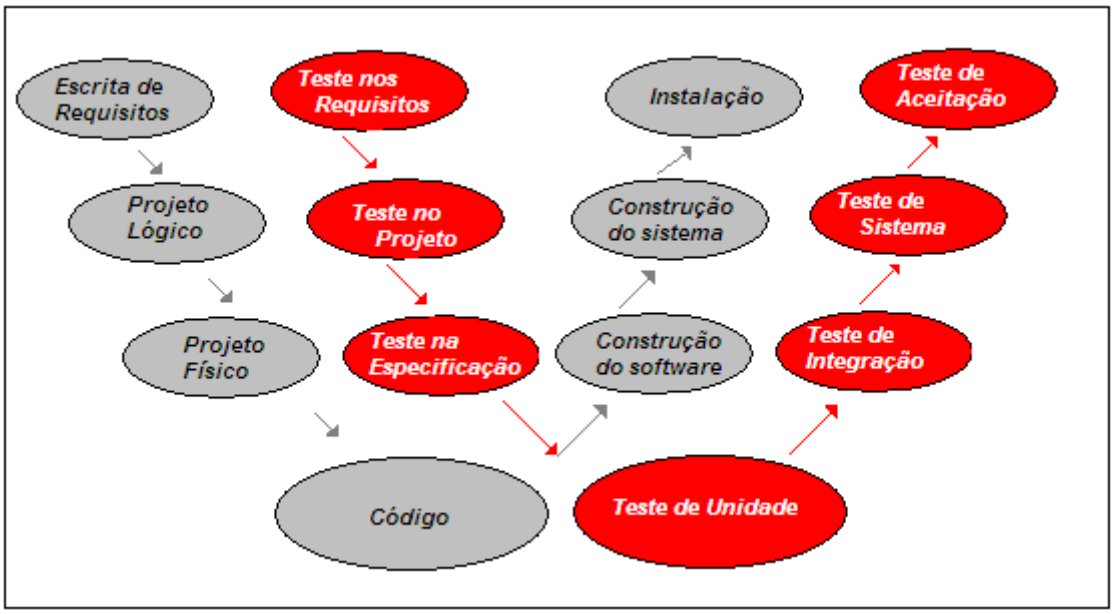

Figura 3.3 – Modelo W [20]

O autor propõe algumas características para a formulação dos testes:

➔ os testes devem ser automatizados desde o início, devem-se evitar testes manuais, ao menos que seja realmente necessário;

➔ deve-se ter a opção de amarrar os testes realizados no código do desenvolvedor aos procedimentos de colocação e extração de códigos fonte em softwares, pois, isso garantirá que eles realizarão alguma espécie de teste e que um conjunto confiável de teste de regressão seja mantido durante o desenvolvimento;

➔ considere o uso de *test drivers*, programas simples que aceitam dados de teste, constroem as chamadas ao código, executam as transações e guardam os resultados para avaliação.

Gerrard [20] identificou 20 (vinte) tipos distintos de testes, cada relacionado a uma área específica de risco, agrupados em cinco categorias:

- ➔ Teste estático
- ➔ Teste de navegação
- ➔ Teste funcional
- ➔ Teste não funcional
- ➔ Integração em larga escala

Os testes devem ser priorizados segundo a seguinte ordem e área de risco:

1. Teste de fumaça (o sistema suporta pelo menos uma sessão de uso sem apresentar falha)
2. Usabilidade
3. Desempenho
4. Funcionalidade

O autor classifica também os testes quanto ao estágio:

1. Teste no computador de desenvolvimento
2. Teste de infra-estrutura
3. Teste de sistema
4. Alta escala de integração
5. Monitoramento após entrada em produção

Os testes podem ser ainda manuais(M) ou automatizados(A); e estáticos(E) – não precisa executar o programa – ou dinâmicos(D) – precisa executar o programa.

A partir dessas classificações, Gerrard criou o arcabouço para testes baseados em riscos. Podemos verificar um modelo desse arcabouço na Tabela 3.1. Analisando a tabela, podemos observar, na segunda linha, que a atividade de “teste de HTML” pertence aos riscos relacionados à fumaça, e o teste deve ser estático, com uma parte automatizada e uma parte manual, na fase de desenvolvimento na estação de trabalho. Outro detalhe importante é que, por prioridade, deve-se realizar todos os testes que possuem um “S” na coluna “Fumaça”, em seguida na coluna “Usabilidade”, depois “Performance” e por fim “Funcionalidade”.

Tabela 3.1 – Arcabouço do processo de teste [20]

	Prioridades de teste					Tipo de Teste Mapeados para Estágios de Teste				
Tipo de Teste	Fumaça	Usabilidade	performance	Funcionalidade	Estático/Dinâmico	Desenvolvimento na Estação de Trabalho	Teste de infra-estrutura	Teste de Sistema	Teste de integração	Monitoramento em Produção
Teste estático										
Checagem de Conteúdo		S			E	A/M				
Teste de HTML	S				E	A/M				
Compabilidade com a Sintaxe do Navegador Internet	S				E	A				
Validação Visual no Navegador		S			D	M		M		M
Teste de Navegação										
Checagem dos Links	S				D			A		A
Tempo de Carga de Objetos		S	S		D			A		A
Verificação de Transação	S				E	A/M		A/M		
Teste Funcional										
Teste de Navegação da Página	S				D	A/M				
Teste de Componentes de CGI	S				D		A/M			
Teste de Transação				S	D			A/M		
Teste de Aplicações de Sistema				S	D			A/M		
Internacionalização		S			D	A/M		A/M		
Teste Não-funcional										
Teste de Configuração	S				D	M		A/M	M	
Teste de Performance			S		D		A	A		A
Teste de Resistência/ confiabilidade	S				D	A	A	A	A	
Disponibilidade					D					A
Usabilidade		S			E/D			M		
Segurança				S	D		A/M	A/M	A/M	A
Integração em Larga Escala										
Links externos/ integração com sistemas legados				S	D				A/M	
Funcionalidade Ponta-à-ponta	S				D		A/M		A/M	A

3.2.4 Estratégia de Seleção de Casos de Teste

Em Chen e Prober [21] encontramos a utilização de métricas de riscos para selecionar casos de teste a serem aplicados. A abordagem dos autores, baseada em risco, foca em casos de teste que verificam as áreas que possuem maior risco, ajudando o sistema a atingir um nível de confiabilidade adequado na qualidade do software.

A metodologia dessa proposta consiste de duas etapas: seleção de casos de teste e seleção dos cenários de teste ponta-a-ponta. Ela utiliza o modelo de risco de Amland[22] apresentado na Seção 3.2.5.

A fase de seleção de casos de teste engloba as seguintes atividades:

1. estimar o custo de cada caso de teste, ou seja, o custo que uma falha ocorrida naquela área pode causar;
2. derivar a severidade para cada caso de teste;
3. calcular o grau de exposição de risco para cada caso de teste.

Selecione os casos de teste que têm os maiores valores de exposição ao Risco como Casos Seguros (checam que falhas graves não ocorrem). A fase de seleção de cenários de teste baseados em risco deve obedecer duas regras:

1. Selecione os cenários para cobrir os casos de teste mais críticos
2. Garanta que os cenários cubram tantos casos de teste quanto possível

A fase de seleção segue os seguintes passos:

1. calcule a exposição de riscos para cada cenário;
2. selecione os cenários com maior exposição ao risco;
3. remova os cenários selecionados e os testes já cobertos e recalcule a exposição ao risco;
4. repita os passos 1 e 2 quanto desejar.

3.2.5 Seleção de Áreas de Testes

Amland [22] realizou um estudo de caso de uma aplicação em uma instituição financeira. Esse estudo consistiu em desenvolver um conjunto de métricas para a análise de risco com objetivo de subsidiar um processo de teste.

Antes do início dos testes no sistema, foi desenvolvida a análise do risco, todavia, essa continuou sendo realizada durante a execução dos testes, para acompanhamento e controle da situação de risco. A aplicação possui dois módulos, um *on-line* e outro que processa em lote (*batch*). Para exemplificar a técnica de Amland, utilizaremos o modelo de análise utilizado no módulo de processamento em lote, descrito a seguir.

A instituição financeira desenvolveu um modelo para o cálculo da exposição ao risco baseado em:

- Probabilidade de um erro
- A conseqüência (custo) de um erro

As principais fontes de análise de risco são:

- Qualidade da função (módulo ou programa) a ser testado. Isto foi utilizado como indicação da probabilidade de uma falha – P(f).
- As conseqüências de uma falha em uma função do ponto de vista de um cliente, representando um custo para o consumidor C(c).
- As conseqüências de uma falha em uma função do ponto de vista de um vendedor, representando um custo para o vendedor C(v).

Desta forma podemos analisar a equação 3.1.

$$\mathrm{Re}(f) = P(f) * \frac{C(c) + C(v)}{2}$$

Equação 3.1 – Cálculo da exposição de risco

A partir dos graus de Exposição ao Risco, as áreas com maior risco possuem prioridade no teste. Essa prioridade pode ser modificada a medida que falhas vão ocorrendo e novas prioridades podem ser adicionadas, assim, os graus de exposição ao risco vão sendo atualizados.

As áreas de processamento em lote da aplicação foram divididas em funções que foram utilizadas para a análise de risco:

- Cálculo de juros
- Cálculo de multas
- Análise de lucratividade
- Exclusão de dados
- Geração de relatórios
- Capitalização

Amland propôs os seguintes indicadores:

- Custo de falha: 1 a 3 com 1 representando o custo mínimo. Os elementos a serem considerados são:
  - Recurso de manutenção a serem alocados no caso de uma falha;
  - Conseqüências legais no caso do vendedor não cumprir com requisitos governamentais;
  - Conseqüências de uma má reputação.

A probabilidade de uma falha foi indicada dando a 4 indicadores um número que varia de 1 a 3, onde 1 indica probabilidade de falha baixa. Os indicadores são:

- Mudanças ou novas funcionalidades
- Qualidade do projeto
- Tamanho
- Complexidade

Para o custo relacionado a cada função em lote, utiliza-se a média do custo do cliente – C(c) e o custo do vendedor C(v). Os indicadores usados para calcular a probabilidade de falhas para uma função particular, P(f), foram ponderados, com os pesos variando de 1 a 5, com o número 5 indicando a maior probabilidade de falha.

No exemplo, os pesos utilizados pelo vendedor foram:

- Mudanças ou novas funcionalidades - 5
- Qualidade do projeto - 5
- Tamanho - 1
- Complexidade – 3

Podemos observar um exemplo do grau de exposição ao risco para a função “Fechar contas” na Tabela 3.2. O grau de exposição ao risco – Re(f) – foi calculado para todas as funções, não apresentadas aqui, e a lista foi ordenada para identificar quais áreas deveriam ser focadas durante o teste. A probabilidade é calculada como a média ponderada de uma função em particular dividida pela maior média ponderada de todas as funções, dando uma probabilidade na faixa [0,1].

Tabela 3.2 – Cálculo do grau de exposição ao risco [22]

	Custo			Probabilidade						
Função	C(v)	C(c)	Média C	Nova Fun-ção (5)	Quali-dade (5)	Tama-nho (1)	Com-plexi-dade (3)	Média Pon-de-rada	Proba-bilida-de P(f)	Expo-si-çao ao risco Re(f)
Fechar Conta	1	3	2	2	2	2	3	7,75	0,74	1,48

3.2.6 Planejamento de Testes a partir do Conceito de Risco

Redmill [23] define risco como sendo função da probabilidade da ocorrência de um evento indesejado e das conseqüências potenciais que esse evento possa causar. Assim, ele propõe o planejamento de testes a partir de três diferentes análises de riscos: fator único de análise – conseqüência; fator único de análise – probabilidade; análise de dois fatores de análise combinados.

1. Fator único de análise: conseqüência

Essa técnica utiliza a conseqüência de uma falha como base para analisar os riscos e em seguida planejar os testes. Para isso, os seguintes passos são propostos:

a. identificar conseqüências;
b. focar os testes de acordo com a severidade das conseqüências;
c. encontrar falhas;
d. fazer correções;
e. testar novamente e assim reduzir probabilidade de falhas no risco.

A análise por conseqüência pode fornecer três níveis de refinamento: tratar o sistema como um todo, distinguir dentre os serviços fornecidos os mais importantes (menor conseqüência estimada) e identificar a importância das funções individuais para a provisão dos serviços. O próximo passo é determinar como o valor da conseqüência é interpretado pelos planos de teste para os itens de software. Esse “valor da conseqüência” é traduzido em outro valor denominado “nível de integridade”. Assim, quanto maior a conseqüência, maior o valor do nível de integridade. O princípio é planejar os testes para cada nível de integridade sendo que os testes mais rigorosos são aplicados aos maiores níveis de integridade.

2. Fator único de análise: probabilidade

As estimativas da probabilidade de falhas do sistema e do serviço não são possíveis antes da produção do software. Assim, em primeiro lugar, o planejamento de teste baseado em risco, deve ser informado pela análise da conseqüência, que produz os alvos da probabilidade que são aplicados ao projeto de planejamento de testes.

A base dessa análise é a qualidade de dois fatores: produto e processo de criação do software. A idéia é que, a partir da análise da probabilidade de ocorrência de uma falha, em um determinado item de software, alguns atributos sejam avaliados, e pontuações sejam estabelecidas, como podemos observar na Tabela 3.3. Para usar os atributos como estimativa de qualidade é preciso fazer uma categorização de quatro níveis para cada atributo. Uma escala numérica pode ser utilizada, onde o quatro representa o melhor (excelente, muito alta) e o um representaria o pior (mal, muito baixa). Ou seja, se a probabilidade de ocorrer uma falha em um determinado atributo for alta, então o fator de qualidade é baixo. A partir dessa avaliação, calcula-se o fator de qualidade que subsidia o planejamento de testes. Assim, os itens cujo fator de qualidade forem menores, terão testes mais rigorosos.

Tabela 3.3 – Cálculo do fator de qualidade [23]

	Item de Software A	Item de software
Software:		
Estrutura		
Comentários		
Complexidade		
Avaliação total do software		
Documentação:		
Estrutura		
Legibilidade		
Exatidão		
Integralidade.		
Avaliação total da documentação		
Desenvolvimento:		
Processo de Especificação		
Processo de Projeto		
Processo de Codificação		
Avaliação total do desenvolvimento		
Desenvolvimento Pessoal		
Gerente		
Projetista		
Programador		
Avaliação total do pessoal		
Fator de Qualidade		

3. Análise de dois fatores combinados

A idéia de analisar o fator conseqüência junto com o fator probabilidade, gera uma tabela, denominada matriz de confiança. É a partir da análise dessa tabela que as atividades de teste são planejadas. Essa matriz é formada por dois eixos: o eixo que indica os níveis de integridade, derivados da análise das conseqüências; e o eixo que indica os fatores de qualidade, derivados da análise da probabilidade, como podemos observar na Tabela 3.4. Cada item de software é localizado na tabela. Por exemplo, o item 1 que possui nível de integridade 2 e fator de qualidade 4, será alocado na terceira linha e quarta coluna da matriz.

Tabela 3.4 – Matriz de confiança [23]

Níveis de Integridade

4		Item 3		
3	Item 2			
2	Item 4			Item 1
1				
	1	**2**	**3**	**4**

Fatores de Qualidade

A partir dessa alocação dos itens, as atividades de teste são realizadas. A tabela 3.5, dividida em três classes de risco, é utilizada como base para determinar a atividade mais adequada a cada item de software, de acordo com a sua localização. As classes, e as respectivas atividades que cada uma determina, são as seguintes:

A: O sistema deve retornar para os desenvolvedores para ser reescrito, ou para melhoria dos atributos de qualidade;

B: Aplique o programa planejado de teste por completo;

C: O programa planejado de teste pode ser reduzido sob circunstâncias excepcionais (estouro de prazo, por exemplo).

Tabela 3.5 – Matriz de confiança: classes de risco [23]

Níveis de Integridade

4	A	A	B	B
3	A	B	B	C
2	A	B	C	C
1	A	C	C	C
	1	**2**	**3**	**4**

Fatores de Qualidade

Observando as duas matrizes, vimos, por exemplo, que o “item 1” pertence à classe de risco “C” e que deve executar um programa reduzido de teste; e que o “item 2” pertence à classe “A” e deve retornar para os programadores.

3.3 Resumo do Capítulo

Este capítulo tratou de relacionar os riscos aos testes de software. Assim, introduzimos o conceito de teste baseado em risco, analisando diversas abordagens de aplicação dessa interessante técnica.

Capítulo 4

Uma Nova Abordagem para Utilização de *Risk-based Testing*

Este capítulo irá explicar uma nova proposta para realização de testes baseados em riscos. Assim, a maior contribuição desse trabalho é a criação de uma nova abordagem de definição de casos de testes, que são obtidos a partir dos riscos encontrados no documento de requisitos. Para isso, é importante a utilização de técnicas de identificação de riscos, como o uso da taxonomia de riscos.

Através do uso de taxonomia, os riscos identificados são aqueles pertinentes à classe de engenharia de produto, encontrados no elemento denominado requisitos. Sendo assim, dada as funcionalidades comuns à grande maioria dos softwares, são identificados os riscos que as afetam e assim, casos de testes são propostos para minimizar e remover os riscos.

Assim, nas seções subseqüentes, a idéia proposta será demonstrada. Na Seção 4.1, será feita uma explanação sobre documentos de requisitos, mostrando quais funcionalidades são consideradas genéricas e estabelecendo quais serão consideradas nesse trabalho. A Seção 4.2 tratará de explicar quais riscos serão considerados para o presente estudo. Um relacionamento entre os riscos e as funcionalidades de um software é observado na Seção 4.3. Finalmente, na Seção seguinte 4.4, serão determinados os testes que serão utilizados visando minimizar e até remover as falhas provenientes dos riscos encontrados no documento de requisitos.

4.1 Requisitos de Software

Os requisitos de software expressam as características e limitações de um produto de software, tendo como principal objetivo garantir que as necessidades exigidas pelo cliente sejam satisfeitas.

Para isso, dentro da Engenharia de Software, existe uma disciplina, denominada Engenharia de Requisitos, que se preocupa com a identificação, análise e gerência desses requisitos, negociando com o cliente quais os requisitos são viáveis ou não, e formalizando-os em um documento de requisitos. Podemos observar essas fases na Figura 4.1.

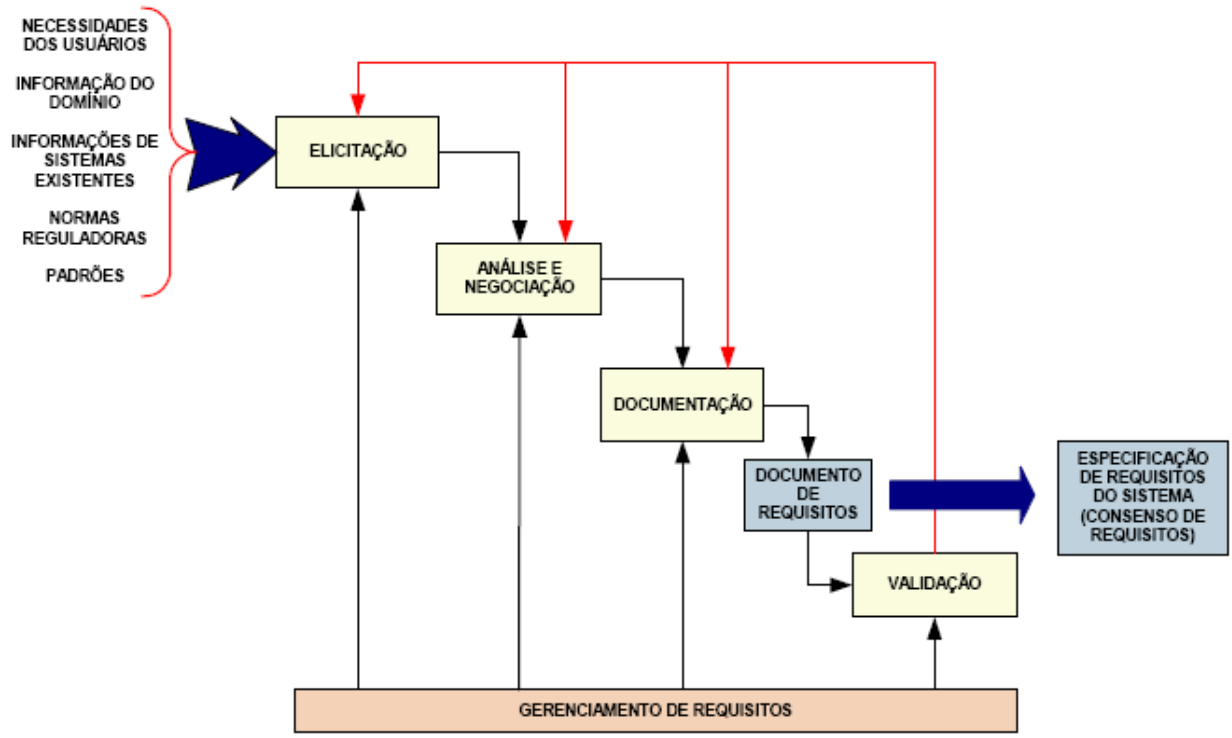

Figura 4.1 – Um modelo para o processo de Engenharia de Requisitos [24]

De acordo com Gilb[25], os requisitos de software podem ser classificados em duas categorias:

- **Requisitos Funcionais**, que declaram quais serviços que o sistema deve fornecer, como o sistema deve reagir a entradas específicas e como deve se comportar em determinadas situações; e
- **Requisitos Não Funcionais**, caracterizados pelos quesitos de qualidade e restrições operacionais ou do desenvolvimento do software[25].

Na documentação, os requisitos são diferenciados através dessas duas classes. Para esse estudo, teremos como recurso único utilizado o documento de requisitos. Neste documento, focaremos nos requisitos funcionais.

Observando os requisitos funcionais, temos algumas atividades que são encontrados em grande parte dos sistemas. Essas funcionalidades, comuns aos sistemas, são elencadas abaixo:

⇒ Cadastro – Grande parte dos sistemas necessita de um banco de dados para armazenar as informações e gerenciá-las. Para uma atividade de cadastro eficiente, algumas operações são requeridas:

➔ Inclusão – permite inserir alguma informação ao banco de dados;

➔ Remoção – permite remover algum cadastro do banco de dados;

➔ Alteração – permite editar algum cadastro feito anteriormente;

➔ Consultas – permite o detalhamento dos cadastros realizados, bem como o acesso a outras informações do banco de dados.

A partir dessas funcionalidades, estabeleceremos quais riscos podem ameaçar o desenvolvimento e a partir disso, propor casos de testes que, aplicados ao documento de requisitos, possam diminuir o esforço de teste ao longo do desenvolvimento.

4.2 Riscos de Requisitos

Como visto na Seção 3.1.3, a taxonomia de riscos é formada por classes subdivididas em elementos. Esse trabalho irá focar nos riscos de requisitos, encontrados na classe de Engenharia do produto. Os riscos de requisitos estão associados à qualidade da especificação dos requisitos e na dificuldade de implementar um software que satisfaça às necessidades requeridas.

Os requisitos técnicos, difíceis de implementar ou imprecisos, juntamente com a falta de habilidade para negociar requisitos, ou ainda orçamentos e prazos, são uma fonte bem reconhecida da engenharia de riscos do software. Os atributos, caracterizados nas próximas subseções, serão empregados para iluminar a natureza dos riscos que são associados aos requisitos.

4.2.1 Estabilidade

A estabilidade se refere aqueles projetos onde os requisitos não mudam. Se existe algum projeto no qual o requisito ou a interface externa muda durante o processo de desenvolvimento, então existem riscos de estabilidade de requisitos. A evolução é uma característica intrínseca a qualquer sistema e assim podemos considerar que alterações em requisitos são inevitáveis. Tais projetos às vezes falham por não haver uma gerência efetiva na evolução dos requisitos. Com isso, mudanças nos requisitos refletem alterações nos orçamentos, prazos e recursos do projeto, que se não forem bem controladas, ameaçam a qualidade do software.

Assim, o controle de mudanças permanece como uma importante ferramenta para o gerenciamento de riscos e qualquer problema inesperado que ocorrer é considerado uma falha na gerência do risco de estabilidade.

4.2.2. Completude

Requisitos incompletos não conseguem descrever todas as intenções, muito menos as reais intenções dos clientes. A principal conseqüência desta fonte de riscos é que o escopo não pode ser alinhado com prazos, orçamentos e recursos.

A especificação dos requisitos visa uma melhor codificação dos requisitos de modo que prazos, orçamentos e recursos possam ser mais plenamente gerenciados. Além disso, testes e verificação dos códigos, não podem ser muito rigorosos sem requisitos completos.

Sendo assim, se existe algum projeto que possui requisitos ou interface externa que não estão completamente definidas ou especificadas, então existe risco de completude.

4.2.3. Clareza

A Clareza dos requisitos está relacionada com o entendimento, ou seja, se um cliente expressa a necessidade de alguma funcionalidade e os desenvolvedores traduzem-na em requisitos ou em uma especificação completa sem que a tenham entendido corretamente, a probabilidade de falhas virem a ocorrer é grande. A conseqüência da falta de clareza é que a verdadeira intenção dos requisitos pode não ser descoberta até que impactos negativos ocorram nos prazos e orçamentos do projeto.

A falta de clareza nos requisitos tem sido citada como a terceira maior fonte de riscos encarados por todos os projetos de desenvolvimento de software[26]. Se existem projetos cujos requisitos não estão claros ou precisam de interpretação, então existe risco de clareza de requisitos.

4.2.4. Validade

A exatidão se refere às expectativas de que os requisitos refletem as intenções dos clientes para as aplicações, ou seja, se eles estão satisfazendo as reais necessidades dos clientes. Se os requisitos não capturam as expectativas dos clientes, o comprometimento do cliente para com o projeto pode ser posto em perigo.

O mau comprometimento do usuário tem sido citado como a segunda maior fonte de riscos de software [26]. Se existe algum projeto cujos requisitos não representam a necessidade do usuário, ou ainda se esses requisitos não tenham o mesmo entendimento para o cliente e desenvolvedor, então existe risco de validade de requisitos.

4.2.5. Precedência

Essa fonte de risco ameaça qualquer projeto de desenvolvimento do software que propõe algo que não foi demonstrado em softwares existentes ou que são além da experiência da equipe ou da instituição. A conseqüência é que os gerentes e a equipe de projeto podem não conseguir

identificar se os objetivos são inviáveis ou até ter dificuldade em implementar as funcionalidades sem precedentes.

4.2.6. Escala

Esse risco se refere à escalabilidade do sistema perante a organização, ou seja, se o produto a ser desenvolvido é considerado complexo, ou grande demais, ou ainda se a organização desenvolvedora não tem experiência com esse tipo de produto, existe um risco de escala nesses requisitos.

Como podemos ver, o risco de escala se refere a todo o documento de requisitos, ao produto formado como um todo, e não a um requisito específico e como tal não será analisado para funcionalidades específicas.

4.2.7. Viabilidade

Todos os requisitos precisam ser analisados e negociados com o cliente. No entanto, podem ocorrer requisitos inviáveis ao projeto ou até difíceis de implementar. Se ocorrerem requisitos desse tipo têm-se riscos de viabilidade.

De posse dos riscos que ameaçam os requisitos de software, podemos identificar quais afetam cada funcionalidade apresentada na Seção 4.1.

4.3 Riscos *versus* Requisitos

Esta seção tratará de relacionar os requisitos de software com os riscos que os ameaçam, mostrando onde ocorrem essas ameaças e o que pode acontecer caso os riscos se transformem em problemas potenciais.

4.3.1 Inclusão

O primeiro requisito que iremos analisar é a inclusão. Como já dito anteriormente, esse caso de uso, comum à maioria dos documentos de requisitos, se refere à inserção de qualquer dado ao banco de dados do sistema. Vamos analisar quais riscos afetam essa funcionalidade e quais impactos a falta de controle desses riscos podem causar.

Estabilidade: esse risco afeta este requisito por estar diretamente ligado a mudanças no sistema, que como já vimos anteriormente, são inevitáveis. Especificamente, as mudanças

na inclusão podem ocorrer devido à má especificação dos requisitos, ou na fase de elicitação dos requisitos, ou ainda por vontade dos clientes, enfim, dentre essas mudanças, podemos prever as seguintes:

- se todos os dados de entrada (atributos) para o cadastro condizem com o solicitado;
- se os campos reservados para os dados de entrada possuem o tamanho requerido (por exemplo o CPF que necessita de 11 caracteres no mínimo);
- quais campos devem ser preenchidos para um cadastro ser efetivado;
- quem poderá realizar um cadastro;
- se os dados estão sendo armazenados corretamente no banco de dados;

Completude: na inclusão, para verificar risco de completude, temos que analisar os dados de entrada, ou seja, se todas as entidades e atributos são suficientes e necessários.

Clareza: o risco de clareza só ocorre na inclusão se as intenções do cliente não foram bem interpretadas ou entendidas, se o documento não consegue esclarecer todas as entidades e respectivos atributos que deverão ser cadastrados, bem como a relação de cada entidade com as demais, esse documento possui risco de clareza para a inclusão.

Validade: a validade se refere à exatidão, ou seja, se a inclusão está condizendo com o que o cliente solicitou. Deve-se verificar aqui se o documento interpreta corretamente as intenções do cliente, ou seja, se todas as entidades solicitadas estão presentes no documento, se os respectivos atributos estão corretos, se as entidades estão relacionadas corretamente para criação de alguma funcionalidade específica, enfim, se tudo o que foi negociado se encontra no documento.

Precedência: como a inclusão é uma funcionalidade comum a qualquer documento de requisito, o risco de precedência é praticamente nulo, podendo ocorrer em algum sistema específico. Sendo assim, a precedência não será considerada como risco a inclusão.

Viabilidade: a inclusão é viável a qualquer sistema, mas com relação à viabilidade o que temos que analisar é se as entidades a serem cadastradas, bem como os atributos ligados a ela, são viáveis.

4.3.2 Remoção

Iremos verificar agora, como os riscos podem afetar a ação de remover algum cadastro do sistema. Para isso, analisaremos cada risco separadamente.

Estabilidade: como vimos na inclusão, o risco de estabilidade só ameaça requisitos passíveis de mudanças. As mudanças referentes à remoção são muito sutis. O que temos que verificar é se, após a remoção, o banco de dados realmente removeu todo registro, e se, ao se optar pela remoção de um registro, essa ação é feita diretamente, sem nenhuma confirmação anterior. Se isso ocorrer, não haverá risco de estabilidade para a remoção, caso não ocorra, mudanças no software serão necessárias.

Completude: só haverá risco de completude para a remoção se o requisito não possuir a opção de confirmar remoção. No entanto, isso é um detalhe do requisito e não uma exigência que comprometa a ação de exclusão. Assim, como essa funcionalidade apenas solicita a remoção de algo do banco de dados, podemos considerá-la um requisito completo.

Clareza: não há fator de risco associado à remoção.

Validade: para haver risco de validade algum aspecto do requisito deve não estar correto e para esse requisito, a única inexatidão possível é, ao solicitar a remoção, o banco de dados não apagar o registro. Se isso ocorrer, haverá risco de validade.

Precedência: como ocorre na inclusão, não há fator de risco associado à remoção.

Viabilidade: Para a remoção ser viável, temos que verificar quais cadastros podem ou não ser removidos, bem como, quanto tempo um cadastro, mesmo sem uso, deve durar no sistema. Num sistema de um hospital, por exemplo, pode ser inviável remover qualquer registro (prontuário) de um paciente, mesmo em caso de óbito, pois é útil para análises estatísticas.

4.3.3 Alteração (Edição)

Agora os riscos serão analisados sob a visão da alteração dos cadastros.

Estabilidade: para que o risco da estabilidade não ameace essa funcionalidade, o sistema deve retornar o cadastro solicitado corretamente e após as alterações realizadas, armazenar no banco de dados o cadastro modificado. Se esse procedimento ocorrer corretamente, o risco de estabilidade é praticamente nulo, no entanto, se, por exemplo, o banco de dados não salvar as alterações, mudanças no sistema serão inevitáveis e o risco da estabilidade passa a existir.

Completude: para não haver risco de completude, o sistema deve: possibilitar a busca de um cadastro, no banco de dados, por sua chave primária ou por qualquer outra chave,

como queria; possibilitar a edição de todos os campos cadastrais; possibilitar o armazenamento dos novos dados, excluindo os dados antigos. Se a alteração segue esses passos, não há risco de completude.

Clareza: não há fator de risco associado à alteração de um cadastro.

Validade: o risco de validade, para alteração, ocorrerá se houver algum erro ao se buscar um cadastro, ou se não for possível alterar algum campo desejado, ou ainda se as alterações não forem armazenadas corretamente. Ocorrendo qualquer um desses eventos, haverá risco de validade.

Precedência: Não há risco de precedência para a alteração, pelo mesmo motivo da inclusão. São funcionalidades comuns à grande maioria dos softwares e como tal, já implementada várias vezes.

Viabilidade: Não há risco de viabilidade para alteração. É um requisito complementar à inclusão e extremamente necessário.

4.3.4 Consulta

Analisaremos agora os riscos, do ponto de vista da consulta de cadastros.

Estabilidade: a consulta é um caso de uso que apenas retorna algo que foi solicitado. Sendo assim, só haverá alteração nesse requisito se o sistema não retornar corretamente, ou simplesmente não retornar o que foi pedido. De caráter evolutivo, algumas mudanças que podem ocorrer são: buscar, no banco de dados, cadastros que possuam um atributo comum (pessoas do sexo **feminino**, ou carros com a cor **vermelha**, ou filmes de **comédia**); o sistema gerar relatórios temporais e formulários, a partir de consultas; os relatórios e os formulários possam ser impressos. Prestando atenção em todos esses aspectos, possivelmente não haverá risco de estabilidade.

Completude: a consulta é um requisito completo. Uma busca efetuada deve ser retornada. Sendo assim, a completude não é um risco que ameace este requisito.

Clareza: o requisito de consulta é considerado claro e, portanto, não sofre ameaça do risco de clareza.

Validade: para esse requisito, as inexatidões passíveis de ocorrer são: ao solicitar um cadastro, o banco de dados retornar um registro errado; erros ao gerar os relatórios e formulários; erros de busca ao banco de dados; erros na solicitação de impressão. Evitando todos esses erros, não haverá risco de exatidão.

Precedência: o risco da precedência não ameaça a consulta.

Viabilidade: a consulta é um requisito viável, pois, é onde teremos acesso aos registros que criaremos. No entanto, se o cliente desejar gerar relatórios e formulários em algum programa que não suporte importar os dados do software que será construído, ou ainda por um programa desconhecido da equipe desenvolvedora, a viabilidade passa a ser um risco que ameaça a consulta.

4.3.5 Visão do Documento de Requisitos

Escala: Para que se tenha risco de escalabilidade, o sistema solicitado deve ser considerado complexo e grande, ou então a empresa que desenvolverá o sistema não deve ter tido experiência com sistemas semelhantes. Na Tabela 4.1 a seguir, podemos observar quais riscos ameaçam cada funcionalidade de um sistema.

Tabela 4.1 – Requisitos *versus* Riscos

REQUISITOS	RISCOS						
	Estabilidade	Completude	Clareza	Validade	Precedência	Escala	Viabilidade
INCLUSÃO							
REMOÇÃO							
ALTERAÇÃO							
CONSULTA							
DOC. REQUISITOS							

4.4 Testes Baseados em Riscos

A partir da análise feita na Seção 4.3, podemos concentrar os testes nas partes do software mais vulneráveis aos riscos. Essa seção traz como proposta, uma série de testes focados nos riscos que identificamos acima. Será utilizado com base, o estudo de Bach[3], que propõe a análise de riscos baseado em heurísticas.

Em seu estudo, visto na Seção 3.2.1, Bach cria os casos de testes a partir das vulnerabilidades do sistema, ou seja, a partir das fraquezas ou possíveis falhas existentes em um determinado componente. Seguindo os passos de Bach, identificaremos os itens de risco

pertinentes às vulnerabilidades, ameaças e vítimas para as quatro funcionalidades exploradas acima.

1. **Quanto às vulnerabilidades:**

Inclusão:

1. Os dados de entrada podem não ser suficientes para um cadastro ser efetivado.
2. O tamanho dos campos reservados para o cadastro pode não ser suficiente.
3. O banco de dados pode não armazenar corretamente os registros.
4. O sistema pode não gerar relatórios e formulários a partir cadastros.
5. O sistema pode não possibilitar a impressão dos cadastros.
6. Os atributos de cada entidade a ser cadastrada podem estar incompletos.
7. As entidades a serem cadastradas podem não ser necessárias, tornando o cadastro inviável.
8. Alguns atributos de cada entidade a ser cadastrada podem ser desnecessários.

Remoção:

9. O banco de dados pode não remover o registro selecionado para exclusão.
10. O sistema pode excluir diretamente o registro sem que haja uma confirmação do usuário.
11. O cliente pode querer, ao invés de excluir o registro do sistema, arquivar em outro local.
12. O cliente pode não remover os registros a partir de uma seleção temporal.

Alteração:

13. O sistema pode não retornar o cadastro correto para edição.
14. Alguns campos cadastrais podem, erroneamente, não ser editáveis.
15. O banco de dados pode não salvar corretamente as alterações.
16. O sistema pode salvar as alterações sem que haja uma confirmação do usuário, causando retrabalho caso algum erro no cadastro ocorra.

Consulta:

17. O sistema pode não retornar corretamente ou simplesmente não retornar a consulta solicitada.
18. O sistema pode não permitir a busca de dados a partir de atributos comuns.
19. O sistema pode não permitir a emissão de relatórios e formulários.
20. O cliente pode querer que os relatórios e formulários sejam gerados a partir de um programa que não suporte essa ação.
21. Os desenvolvedores podem não conhecer o programa que o cliente deseja utilizar.

22. O sistema pode não permitir a impressão dos relatórios e formulários.

Para o documento de requisitos como um todo:

23. O sistema pode ser considerado grande ou complexo.

24. A empresa desenvolvedora pode nunca ter tido experiência com sistemas desse tipo.

2. Quanto às ameaças:

1. O usuário efetua um cadastro sem nenhum dado de entrada inserido (esta ameaça dispara a vulnerabilidade 1).
2. O desenvolvedor pode disponibilizar menos caracteres que necessário para um determinado campo do cadastro (esta ameaça dispara a vulnerabilidade 2).
3. Ao se buscar um cadastro, o mesmo não é encontrado no banco de dados (esta ameaça dispara as vulnerabilidades 3,17).
4. O usuário solicita a geração de formulários e relatórios (esta ameaça dispara as vulnerabilidades 4, 19, 20, 21).
5. O usuário pode querer imprimir um cadastro, um formulário, ou um relatório (esta ameaça dispara as vulnerabilidades 5, 22).
6. O cliente pode ter optado pelo cadastro de uma entidade e não estabelecer todos os atributos necessários para caracterizá-lo no sistema (esta ameaça dispara a vulnerabilidade 6).
7. O cliente pode ter solicitado a criação de um cadastro que não seja necessário ao sistema (esta ameaça dispara a vulnerabilidade 7).
8. O cliente pode escolher atributos não necessários ao cadastro de uma entidade (esta ameaça dispara a vulnerabilidade 8).
9. Um cadastro anteriormente “removido” é encontrado numa consulta posterior (esta ameaça dispara a vulnerabilidade 9).
10. O usuário pode excluir um cadastro errado (esta ameaça dispara a vulnerabilidade 10).
11. O usuário pode necessitar a recuperação de um cadastro considerado “morto” (esta ameaça dispara a vulnerabilidade 11).
12. O usuário deseja excluir os cadastros realizados há algum intervalo de tempo (esta ameaça dispara a vulnerabilidade 12).
13. O usuário solicita uma busca e o sistema retorna um registro incorreto (esta ameaça dispara a vulnerabilidade 13,17).

14. O usuário tenta editar um campo e essa edição não é permitida (esta ameaça dispara a vulnerabilidade 14).
15. O usuário, ao consultar um cadastro, anteriormente editado, percebe que as modificações não foram salvas (esta ameaça dispara a vulnerabilidade 15).
16. O usuário altera um cadastro por engano (esta ameaça dispara a vulnerabilidade 16).
17. O usuário solicita a busca de todos os cadastros utilizando algum atributo como filtro (esta ameaça dispara a vulnerabilidade 18).
18. O desenvolvimento demora mais tempo que o previsto e o orçamento não foi suficiente para o término do projeto (esta ameaça dispara as vulnerabilidades 23 e 24).

3. Quanto às vítimas:

1. O desenvolvedor, que pode desenvolver as funcionalidades erroneamente por falta de um documento mais preciso.
2. O cliente, que pode não ser entendido ou mal interpretado e receber um software que não atenda suas funcionalidades.
3. O usuário, que pode não ter suas necessidades atendidas.
4. A organização desenvolvedora, que pode ter problemas orçamentais e temporais graças a um documento de requisitos mal desenvolvido.

A análise das vítimas não interfere diretamente na criação dos casos de teste, serve apenas para ajudar a compreender a severidade das falhas, pois, dependendo da vítima, pode não ser viável criar casos de teste para alguma situação.

4.4.1 Casos de Teste

A partir das vulnerabilidades encontradas, podemos criar casos de teste para cada situação acima. É importante salientar que os testes propostos aqui devem ser aplicados ao sistema desde o documento de requisitos até o desenvolvimento, ou seja, os riscos que encontramos no documento de requisitos nos dão subsídios para criarmos testes em várias fases do desenvolvimento. Esses testes podem ser manuais, que são aqueles aplicados diretamente ao documento de requisitos; e automatizados, que são aplicados ao sistema em desenvolvimento.

Caso de teste 1: Verificar se foi decidido quais dados precisam ser preenchidos para efetivar o cadastro.

Caso de teste 2: Testar a realização de um cadastro vazio, ou preenchido parcialmente, ou com apenas alguns campos preenchidos, ou totalmente preenchido.

Caso de teste 3: Verificar se o formato de todos os atributos é conhecido.

Caso de teste 4: Testar o preenchimento de todos os campos (atributos), para ver se o tamanho estabelecido está correto.

Caso de teste 5: Realizar um cadastro fictício e verificar se o mesmo encontra-se armazenado no banco de dados através de uma consulta.

Caso de teste 6: Verificar se foi acordado a geração de relatórios e formulários pelo sistema.

Caso de teste 7: Criar cadastros para o teste e gerar vários formulários e relatórios para verificar se estão sendo formados corretamente.

Caso de teste 8: Verificar se o sistema deve imprimir os cadastros, os relatórios e os formulários criados.

Caso de teste 9: Gerar cadastros, relatórios e formulários e tentar imprimi-los.

Caso de teste 10: Para cada caso de uso de inclusão, verificar se ele é realmente necessário, rastreando seu relacionamento com as demais entidades.

Caso de teste 11: Para cada caso de uso de inclusão, verificar se todos os atributos que o cliente solicitou, para uma determinada entidade, são suficientes.

Caso de teste 12: Para cada caso de uso de inclusão, verificar se os atributos que o cliente solicitou são realmente necessários.

Caso de teste 13: Excluir um cadastro e realizar uma consulta do mesmo para verificar se a exclusão foi realizada com sucesso.

Caso de teste 14: Verificar se o cliente optou pela exclusão direta ou exclusão com confirmação.

Caso de teste 15: Realizar a remoção de cadastros e testar se os *popups* para confirmação aparecem.

Caso de teste 16: Verificar se o cliente deseja a exclusão total dos registros ou o arquivamento em outro banco de dados dos cadastros considerados obsoletos.

Caso de teste 17: Arquivar um cadastro e tentar recuperá-lo, para verificar se o registro saiu do banco de dados do sistema.

Caso de teste 18: Acessar o arquivo e verificar se o cadastro está armazenado nele.

Caso de teste 19: Verificar se o cliente solicitou a exclusão de registros realizados ou modificados a partir de um intervalo de tempo.

Caso de teste 20: Realizar cadastros e configurar o tempo de 1(um) dia, 1(uma) semana e 1(um) mês para a exclusão dos cadastros, verificar se as exclusões foram efetivadas.

Caso de teste 21: Realizar vários cadastros e verificar se ao consultar um determinado cadastro o sistema está retornando corretamente.

Caso de teste 22: Verificar quais campos do cadastro devem ser fixos ou editáveis.

Caso de teste 23: Testar se todos os campos são editáveis ao se alterar um cadastro.

Caso de teste 24: Realizar alterações em alguns cadastros e verificar se foram realmente salvas.

Caso de teste 25: Verificar se o cliente optou pelo armazenamento direto ou após uma confirmação.

Caso de teste 26: Realizar uma alteração e ao armazenar, testar se os *popups* para confirmação aparecem.

Caso de teste 27: Cadastrar vários objetos com atributos semelhantes. Testar a busca por algum atributo comum.

Caso de teste 28: Verificar se o cliente optou a geração de relatórios e formulários por algum programa particular. Em caso positivo, repetir o caso de teste 7(sete) verificando se o sistema suporta essa ação.

Caso de teste 29: Verificar a equipe desenvolvedora conhece o programa escolhido. Em caso negativo, pedir para a equipe sugerir outra opção e apresentá-la ao cliente.

Caso de teste 30: A empresa desenvolvedora, através de alguma métrica, caracteriza se a complexidade e o tamanho do sistema é perigoso e em caso positivo, estima prazos e custos com uma maior margem de segurança.

Caso de teste 31: A empresa desenvolvedora verifica se já foi construído algum sistema semelhante ao solicitado. Em caso positivo, utilizar como base essas estimativas; e em caso negativo, estimar prazos e custos com uma maior margem de segurança.

Na tabela a seguir, observaremos quais casos de testes devem ser utilizados em cada requisitos, dado os riscos que os ameaçam.

Tabela 4.2 – Requisitos *versus* Riscos *versus* Testes

REQUISITOS	RISCOS						
	Estabilidade	Completude	Clareza	Validade	Precedência	Escala	Viabilidade
INCLUSÃO	1, 2, 3, 4, 5	10, 11, 12	1, 6, 8, 11, 12	1, 6, 8, 11, 12			10, 11, 12
REMOÇÃO	13, 14, 15, 16, 17	14,15		13			16, 17, 18, 19
ALTERAÇÃO	21, 22, 23, 24, 25, 26	21, 22, 23, 24, 25, 26		21, 22, 23, 24			
CONSULTA	6, 7, 8, 9, 27, 28			21, 29			29

DOC. REQUISITOS						30, 31	

4.5 Prova de Conceito

Para demonstrar uma aplicação da abordagem proposta na Seção 4.4, analisaremos um documento de requisitos (Anexo I) sob essa perspectiva. Assim, iremos analisar os riscos nos requisitos desse documento, indicar as vulnerabilidades, ameaças e vítimas e propor os respectivos casos de testes.

4.5.1 Requisitos

O documento de requisitos analisado é o de um sistema chamado *Methodology Explorer*. Esse sistema é uma ferramenta para o processo de desenvolvimento de software. Fornece uma maneira intuitiva e eficiente para definir componentes adequados a uma empresa/projeto. Um componente é uma unidade da metodologia que pode ser manipulada isoladamente, por exemplo, artefato, atividade etc. Utilizando a ferramenta, o usuário - em geral, engenheiro de processos ou projetista de metodologias - poderá cadastrar novos componentes ou criar componentes a partir de outros já existentes. Além disso, poderá alterar, remover e consultar componentes já criados. Tais componentes podem ser exportados da ferramenta, gerando um documento texto, páginas HTML ou um arquivo PDF que podem ser visualizados sem utilizar a ferramenta. Resumidamente, os requisitos são os seguintes:

Cadastro

1. Criar Componente: este caso de uso permite que o usuário crie e armazene um novo componente no sistema.
2. Excluir componente: este caso de uso permite que o usuário exclua um componente do cadastro de componentes do sistema. Um componente pode ser excluído de qualquer instanciação de metodologia (árvore).
3. Alterar componente: este caso de uso permite que o usuário altere os dados de um componente.

Interface

1. Visualizar Componente: este caso de uso permite que o usuário visualize os dados de um determinado componente (todos os seus atributos, exceto aqueles que são considerados suas propriedades).
2. Copiar componente: este caso de uso permite que o usuário copie um componente do cadastro de componentes do sistema. Ou seja, copia o componente de onde ele estava e manda a cópia para a área de transferência.
3. Colar componentes: este caso de uso permite que o usuário cole o componente armazenado na área de transferência do sistema no local indicado. O conteúdo da área de transferência continua inalterado. Aqui, local refere-se a uma pasta que contém componentes.

Compilação

1. Compilar componente: este caso de uso permite que o usuário compile metodologias. Essa compilação permite que as metodologias sejam analisadas e comparadas entre si.

Importação/Exportação

1. Anexar documentos: este caso de uso permite anexar documentos gerais a componentes. Por exemplo, anexar o *template* do Documento de Requisitos ao fluxo de requisitos.
2. Exportar metodologia: este caso de uso permite ao usuário a possibilidade de exportar uma metodologia num determinado formato, como XML, por exemplo. O usuário também tem a opção de escolher se o componente deve ou não ser exportado juntamente com seus anexos.
3. Importar metodologia: este caso de uso permite que componentes de uma metodologia exportada sejam importados do sistema de arquivos e apresentados no *Methodology Explorer*. Os componentes, para serem importados precisam estar no mesmo formato utilizado no caso de uso [Importação/Exportação.]. Importar um componente apenas permite manipular o componente dentro do *Methodology Explorer*. Para inseri-lo de fato, é preciso realizar o caso de uso [Importação/Exportação].
4. Salvar metodologia: este caso de uso permite salvar as alterações realizadas nos componentes de uma metodologia.
5. Gerar *site* de metodologia: este caso de uso permite que um *site* seja gerado para uma metodologia já compilada. O *site* deve conter também os possíveis artefatos que foram anexados.

Usabilidade

A interface com o usuário é de vital importância para o sucesso do sistema. Principalmente por ser um sistema que não será utilizado diariamente, o usuário não possui tempo disponível para aprender como utilizar o sistema. O sistema terá uma interface amigável ao usuário primário sem se tornar cansativa aos usuários mais experientes. Em especial, o módulo de publicação HTML possuirá um *wizard* para ajudar o usuário.

Desempenho

Embora não seja um requisito essencial ao sistema, deve ser considerada por corresponder a um fator de qualidade de software.

Hardware e Software

Visando criar um produto com maior extensibilidade, reusabilidade e flexibilidade, deve-se adotar como linguagem principal de desenvolvimento a linguagem "Java", seguindo cuidadosamente as técnicas de orientação a objetos. Entretanto, outras linguagens também poderão ser usadas quando indicações técnicas recomendem.

4.5.2. Identificação de Riscos

Dados os requisitos, vamos analisar os riscos que podem ameaçar o bom andamento do sistema.

1. **Estabilidade** – devemos identificar quais as possíveis mudanças que poderão acontecer no sistema, para assim, antes mesmo de o desenvolvimento começar, ajustarmos os requisitos e evitarmos falhas futuras.

Para esse documento de requisitos, os possíveis riscos de estabilidade existentes são:

- O documento não especifica quais os atributos necessários para criação do cadastro de um componente, bem como a quantidade de caracteres que cada atributo deve possuir.
- O documento não especifica qual o mínimo de campos que deverão ser preenchidos para efetivar o cadastro de um componente.
- O documento não especifica se haverá confirmação de remoção ou de alteração de cadastros.
- O documento não especifica como será o acesso ao sistema.
- O documento não especifica se o sistema deverá imprimir algum documento.
- O documento não especifica se o sistema deverá gerar relatórios.

2. **Completude** – devemos identificar se os requisitos estão completos. Nesse estudo, muitos dos riscos de estabilidade são também riscos de completude, pois se os riscos não estão completos, mudanças nos requisitos são necessárias. Assim, podemos estabelecer os seguintes riscos:

- O cadastro de componente não está bem especificado, ou seja, não se caracteriza o componente, não se tem os atributos do componente.
- A interface não está bem definida.
- A chave de busca de um componente não foi especificada.

3. **Clareza –** para determinar os riscos de clareza, devemos verificar se o documento consegue mostrar as reais intenções do cliente. Desse modo, temos os seguintes riscos de clareza:

- Os requisitos não mostram o que constitui um componente.
- O documento não especifica como se dá o relacionamento dos componentes com as metodologias.
- De um modo geral, os requisitos não estão claros, podendo causar erros gravíssimos no desenvolvimento.

4. **Validade –** os riscos de validade estão relacionados com os requisitos incorretos. Como estamos analisando um documento de requisitos aleatório, sem participar da fase de elicitação dos requisitos, não podemos julgar se um requisito está correto ou não. Assim, temos que determinar que o sistema não possua risco de validade.

5. **Precedência –** o risco de precedência ameaça esse sistema por se tratar de um sistema incomum. Assim podemos identificar os seguintes fatores de risco:

- É possível que nenhum sistema dessa natureza tenha sido implementado anteriormente.
- O documento possui um caso de uso de compilação de uma metodologia, mas não a caracteriza bem, podendo dificulta a implementação de tal funcionalidade.
- O sistema permite a utilização de outras linguagens de programação, não apenas a principal – Java, e assim, pode ser que a instituição não possua pessoas capazes de utilizar outras linguagens.
- A linguagem “Java” pode não dá suporte às outras linguagens utilizadas.

6. **Escala –** O sistema é considerado pequeno, porém suas funcionalidades são complexas. Assim, podemos determinar que o sistema possua risco de escala.

7. **Viabilidade –** os riscos de viabilidade ocorrem quando não se tem uma negociação eficiente com o cliente, podendo o sistema possuir funcionalidades não viáveis. Dentre as funcionalidades desse documento, podemos considerar duas delas não viáveis:

- A geração de *sites* de metodologias.
- A compilação de metodologias.

A questão da não viabilidade de tais funcionalidades se dá porque elas não são consideradas fundamentais, além de estarem mal especificadas. Deve haver uma negociação com os clientes para decidir a verdadeira necessidade de se implementar tais funcionalidades.

A partir dessa análise, podemos levantar as vulnerabilidades, ameaças e vítimas:

⇒ **Quanto às vulnerabilidades:**

1. Um componente pode precisar de um mínimo necessário de atributos para o cadastro ser efetivado.
2. Os tamanhos dos campos reservados para os atributos podem não ser suficientes.
3. O banco de dados pode não armazenar corretamente os registros.
4. O sistema pode não gerar relatórios e formulários a partir dos cadastros.
5. O cliente pode precisar imprimir documentos do sistema.
6. O componente precisa de uma ou mais chaves de busca.
7. O sistema pode excluir ou salvar alterações de um componente sem que haja uma confirmação do usuário.
8. O cliente pode querer, ao invés de excluir o registro do sistema, arquivar em outro local.
9. O cliente pode não remover os registros a partir de uma seleção temporal.
10. O sistema pode não permitir que *backups* periódicos do sistema sejam feitos.
11. Alguns campos do cadastro de componentes podem, erroneamente, não ser editáveis.
12. O banco de dados pode não salvar corretamente as alterações.
13. O sistema pode não possuir uma interface com diretórios e pastas.
14. O sistema pode permitir uma compilação sem que haja nenhuma metodologia salva no sistema.
15. O sistema pode emitir um resultado errado da compilação.
16. O sistema pode não conseguir anexar determinados documentos aos componentes.
17. O sistema pode não exportar/importar as metodologias em algum formato específico.

18. Os desenvolvedores podem não conhecer as linguagens que serão utilizadas, além da linguagem “Java”.
19. A linguagem “Java” pode não interpretar as demais linguagens utilizadas.
20. O sistema pode ser considerado complexo.
21. A empresa desenvolvedora pode nunca ter tido experiência com sistemas desse tipo.

⇒ **Quanto às ameaças:**

- O usuário efetua o cadastro sem nenhum dado de entrada inserido (esta ameaça dispara a vulnerabilidade 1).
- O desenvolvedor pode disponibilizar menos caracteres que necessário para um determinado campo do cadastro (esta ameaça dispara a vulnerabilidade 2).
- Ao se buscar um cadastro, o mesmo não é encontrado no banco de dados (esta ameaça dispara as vulnerabilidades 3).
- O usuário solicita a geração de formulários e relatórios (esta ameaça dispara as vulnerabilidades 4).
- O usuário pode querer imprimir um documento (esta ameaça dispara as vulnerabilidades 5).
- O usuário pode necessitar efetuar a busca de um componente (esta ameaça dispara a vulnerabilidade 6).
- O usuário realiza uma remoção indesejada de um componente por falta de confirmação (esta ameaça dispara a vulnerabilidade 7).
- O usuário salva as alterações erradas de um componente ou de uma metodologia por falta de confirmação (esta ameaça dispara a vulnerabilidade: 7)
- O usuário pode necessitar a recuperação de um cadastro considerado “morto” (esta ameaça dispara a vulnerabilidade 8).
- O usuário deseja excluir os cadastros realizados há algum intervalo de tempo (esta ameaça dispara a vulnerabilidade 9).
- O usuário pode tentar realizar um backup do sistema (esta ameaça dispara a vulnerabilidade 10).
- O usuário tenta editar um campo e essa edição não é permitida (esta ameaça dispara a vulnerabilidade 11).
- O usuário, ao consultar um cadastro, anteriormente editado, percebe que as modificações não foram salvas (esta ameaça dispara a vulnerabilidade 12).

- O usuário tenta transferir um componente de um diretório para outro (esta ameaça dispara a vulnerabilidade 13).
- O usuário tenta compilar uma metodologia (esta ameaça dispara as vulnerabilidades 14 e 15).
- O usuário tenta anexar um documento a um componente (esta ameaça dispara a vulnerabilidade 16).
- O usuário necessita importar/exportar um componente ou uma metodologia. (esta ameaça dispara a vulnerabilidade 17).
- Partes do sistema não funcionam (esta ameaça dispara a vulnerabilidade 19).
- O desenvolvimento demora mais tempo que o previsto e o orçamento não foi suficiente para o término do projeto (esta ameaça dispara as vulnerabilidades 18, 20 e 21).

⇒ **Quanto às vítimas:**

- O desenvolvedor, que pode desenvolver as funcionalidades erroneamente por falta de um documento mais preciso ou até por falta de conhecimento das tecnologias solicitadas.
- O cliente, que pode não ser entendido ou mal interpretado e receber um software que não atenda suas funcionalidades.
- O usuário, que pode não ter suas necessidades atendidas e não conseguir entender nem manipular o sistema.
- A organização desenvolvedora, que pode ter problemas orçamentais e temporais por causa de um documento de requisitos mal desenvolvido.

4.5.3 Casos de Teste

1. Cadastro

Caso de Teste 1: Verificar se foi decidido quais atributos um cadastro de componente deve possuir.

Caso de Teste 2: Verificar quais os campos obrigatórios que precisam ser preenchidos para efetivar o cadastro de um componente.

Caso de Teste 3: Testar a realização de um cadastro vazio, ou preenchido parcialmente, ou com apenas alguns campos preenchidos, ou totalmente preenchido.

Caso de Teste 4: Verificar se o formato de todos os atributos é conhecido.

Caso de Teste 5: Testar o preenchimento de todos os campos (atributos), para ver se o tamanho estabelecido está correto.

Caso de Teste 6: Realizar um cadastro fictício e verificar se o mesmo encontra-se armazenado no banco de dados através de uma consulta.

Caso de Teste 7: Verificar se foi acordada a geração de relatórios e formulários pelo sistema.

Caso de Teste 8: Criar cadastros para o teste e gerar vários formulários e relatórios para verificar se estão sendo formados corretamente.

Caso de Teste 9: Verificar se o sistema deve imprimir documentos e se sim realizar impressões para teste.

Caso de Teste 10: Excluir um cadastro e realizar uma consulta do mesmo para verificar se a exclusão foi realizada com sucesso.

Caso de Teste 11: Verificar se o cliente optou pela exclusão direta ou exclusão com confirmação.

Caso de Teste 12: Realizar a remoção de cadastros e testar se os *popups* para confirmação aparecem.

Caso de Teste 13: Verificar se o cliente deseja a exclusão total dos registros ou o arquivamento em outro banco de dados dos cadastros considerados obsoletos.

Caso de Teste 14: Arquivar um cadastro e tentar recuperá-lo, para verificar se o registro saiu do banco de dados do sistema.

Caso de Teste 15: Acessar o arquivo e verificar se o cadastro removido está armazenado nele.

Caso de Teste 16: Verificar se o cliente solicitou a exclusão de registros realizados ou modificados a partir de um intervalo de tempo.

Caso de Teste 17: Realizar cadastros e configurar o tempo de 1(um) dia, 1(uma) semana e 1(um) mês para a exclusão dos cadastros, verificar se as exclusões foram efetivadas.

Caso de Teste 18: Realizar *backups* periódicos no sistema para verificar se estão salvando corretamente.

Caso de Teste 19: Verificar quais campos do cadastro devem ser fixos ou editáveis.

Caso de Teste 20: Testar se todos os campos são editáveis ao se alterar um cadastro.

Caso de Teste 21: Realizar alterações em alguns cadastros e verificar se foram realmente salvas.

Caso de Teste 22: Verificar se o cliente optou pelo armazenamento direto ou após uma confirmação.

Caso de Teste 23: Realizar uma alteração e, ao armazenar, testar se os *popups* para confirmação aparecem.

2. Interface

Realizar a busca de um componente por cada um de seus atributos verificando se as chaves de busca foram bem estabelecidas.

Caso de Teste 24: Verificar se a interface foi bem especificada: possuirá diretórios, janelas, ícones, quais as telas que compõem o sistema.

3. Compilação

Caso de Teste 25: Realizar a compilação de uma metodologia e verificar os seguintes aspectos:

a) A compilação é efetuada se não existir outras metodologias no sistema - se sim, efetuar o caso de teste 26 b.), se não, a implementação está correta.

b) O resultado obtido – se houver mais de uma metodologia no sistema, verificar se a análise e comparação foram feitas corretamente, se não houver outras metodologias para comparação, verificar se o sistema retorna tal situação, informando que não é possível fazer comparações.

4. Importação/Exportação

Caso de Teste 26: Tentar anexar vários documentos de diversos formatos diferentes.

Caso de Teste 27: Importar e exportar vários documentos de formatos diferentes e verificar as limitações do sistema.

5. Hardware e Software

Caso de Teste 28: Verificar se a equipe desenvolvedora conhece as linguagens de programação que serão utilizadas. Em caso negativo, pedir para a equipe sugerir outras opções e apresentá-la ao cliente.

Caso de Teste 29: Verificar se a linguagem de programação principal – Java – reconhece as demais linguagens utilizadas.

Caso de Teste 30: A empresa desenvolvedora, através de alguma métrica, caracteriza se a complexidade e o tamanho do sistema é perigoso e em caso positivo, estima prazos e custos com uma maior margem de segurança.

Caso de Teste 31: A empresa desenvolvedora verifica se já foi construído algum sistema semelhante ao solicitado. Em caso positivo, utiliza essas estimativas como base; e em caso negativo, estimar prazos e custos com uma maior margem de segurança.

4.6 Resumo do Capítulo

Este capítulo mostrou a criação de uma nova abordagem de testes, testes baseados em riscos encontrados nos requisitos de software. Inicialmente foram estabelecidos os requisitos comuns a qualquer documento de requisitos. Em seguida analisamos os riscos de requisitos e como eles afetam as funcionalidades estabelecidas como genéricas. Utilizando uma metodologia de R*isk-based Testing*, foram levantadas as vulnerabilidades, ameaças e vítimas de um sistema genérico e para cada vulnerabilidade, casos de testes foram criados. Após essa análise, podemos observar a análise de um documento de requisitos real, que visa mostrar a viabilidade dessa nova abordagem criada.

Capítulo 5

Conclusões e Trabalhos Futuros

Testar é fundamental para desenvolver software de alta qualidade e garantir a tranqüilidade das operações, pois, as conseqüências que algumas falhas podem causar são terríveis, além disso, a importância da atividade de testes em cada etapa do processo de desenvolvimento de software é fortalecida pelo fato de que quanto mais próximos da sua origem os erros forem detectados, menor será o custo e a dificuldade em efetuar correções.

O uso da análise de riscos como base para o planejamento dos testes é de grande valia, pois, os riscos são fontes de falha e a função do teste é identificar falhas, assim, focar os testes nas áreas de risco pode otimizar a fase de teste, trazendo benefícios relativos ao custo, ao tempo e aos recursos do projeto.

5.1 Contribuições

Este trabalho tem como contribuição principal a proposta de uma nova abordagem para testes de software baseados em riscos, utilizando o documento de requisitos como fonte única de informação para análise de riscos.

Com isso, busca-se otimizar a realização de testes, pois muitos são os benefícios esperados quando se utiliza uma abordagem de testes baseados em riscos. As vantagens mais importantes estão relacionadas a questões de custo, prazo e até mesmo, cultura organizacional.

Outro resultado esperado é dar visibilidade ao estudo na área de testes baseados em riscos no Brasil. Poucos são os artigos escritos e as pesquisas realizadas nesse assunto no país e, visto que é um assunto interessante e aplicável, será bom para que pesquisadores nacionais possam ampliá-lo e adequá-lo a realidade das organizações nacionais.

No entanto, uma consideração deve ser feita a respeito dessa técnica de teste de software: não se tem nenhuma indicação clara de em quanto é reduzido o tempo, os custos e a utilização de recursos, ou seja, não existe uma comparação numérica de volume de atividades de teste

baseados em riscos e outras abordagens de teste tradicionais. Desta forma, argumenta-se a necessidade de mais estudos e aplicações para análise dos resultados.

5.2 Dificuldades Encontradas

Ao realizar esse estudo, podemos destacar algumas dificuldades que limitaram seu desenvolvimento. A primeira delas diz respeito ao conteúdo abordado. Foram encontrados apenas 6 (seis) estudos na área de *Risk-based Testing*, e nenhuma das propostas encontradas foram realizadas por brasileiros. Além disso, apenas o estudo de Bach [3] se preocupa em mostrar como deve ser a organização do esforço de teste, os outros autores focam na análise de riscos. Foi por esse motivo que Bach [3] foi utilizado como base para o desenvolvimento da abordagem proposta nesse estudo.

Outra dificuldade se deve ao fato da diferença sutil dos casos de testes baseados em riscos para os casos de testes tradicionais, pois, por se tratar de casos de uso genéricos, a criação dos casos de teste fica limitada e os testes baseados em riscos acabam, algumas vezes, sendo semelhantes dos casos de testes tradicionais.

Por fim, podemos considerar como dificuldades, as limitações que a própria estrutura do trabalho impõe, em relação ao prazo e ao tamanho do estudo.

5.3 Trabalhos Relacionados

Para o desenvolvimento desse estudo, dois trabalhos são igualmente fundamentais e se relacionam diretamente com a abordagem proposta aqui.

O primeiro deles seria o estudo de Bach [3], explicado na Seção 3.2.1. Sua idéia, chamada *Inside-out*, que propõe a análise das vulnerabilidades, ameaças e vítimas como base para os testes, foi utilizada como modelo para criação da nova abordagem proposta nesse estudo.

Outro trabalho importante é o de Carr et al. [17], que mostra a taxonomia de Riscos proposta pela SEI e que auxiliou esse estudo a identificar os riscos pertinentes aos requisitos.

5.4 Trabalhos Futuros

Dentro do tema “Teste baseado em risco”, várias são as possibilidades de desenvolver novos estudos, ampliando o conhecimento sobre o assunto.

A proposta inicial, dando continuidade a esse trabalho, seria a criação de um “guia” para testes de software baseados em riscos nos requisitos, pois, percebeu-se que muitos dos casos de testes se repetirão para qualquer documento de requisitos e com esse guia poderíamos aplicar a abordagem *outside-in* proposta por Bach [3].

Em seguida, outra proposta para ampliação desse estudo e a análise de risco para o processo completo de desenvolvimento de um sistema, desde o documento de requisitos, estudo já realizado aqui, até a entrega do produto. Para isso, será necessário incluir a análise de outras fontes de risco, ou seja, dentro da taxonomia de riscos, na classe de “Engenharia do Produto”, realizar uma análise de risco de todos os elementos (requisitos, design, teste de código e unidade, integração e teste, engenharia de especialidades). E, a partir dessa análise, determinar a melhor forma de realizar os testes.

Outra proposta é incrementar o processo de testes baseados em riscos com práticas de *Test-Driven Development-TDD*, que é uma técnica de teste onde o desenvolvimento é dirigido por testes. Ora, se vamos realizar uma análise completa de riscos em todo o processo de desenvolvimento podemos utilizar o TDD como base para orientar a melhor maneira de gerar esses testes.

Finalmente seria interessante a realização de uma comparação prática entre abordagens de teste tradicionais e abordagens de testes baseados em risco. Com esse estudo poderá ser possível identificar quando e quanto é mais vantajoso usar uma técnica em detrimento da outra.

Referências Bibliográficas

[1] SOMMERVILLE, Ian. Software Engineering, 6ed, Harlow, Addison-Wesley, 2001, 742p.

[2] AGUIAR, Maurício. Gerenciando objetivamente seu projeto. Developers' Magazine, v. 6, n. 66, p. 46-47, fev. 2002.

[3] BACH, James; Heuristic Risk-based Testing; Software Testing & Quality Engineering Magazine; v.1, n. 6, p. 23-28, nov 1999.

[4] TELES, Vinícius Manhães. Desenvolvimento Orientado a Testes. Rio de Janeiro, Brasil: UFRJ, 2006. Disponível em: <http://www.improveit.com.br/xp/praticas/tdd>. Acesso em: 18 setembro 2007.

[5] HETZEL, William. Guia completo ao teste de software. Rio de Janeiro: Campus, 1987, 206p.

[6]PRESSMAN, Roger S. Engenharia de Software. São Paulo: Makron Books, 1995. 1056p.

[7] PRANGE, Henrique Feliciano. Uma Avaliação Empírica de um Ambiente Favorável para o Desenvolvimento Dirigido por Testes. 2007. 116f. Dissertação (Mestrado em informática) – Programa de Pós-Graduação em Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2007.

[8] GRAF, Clayton; SERRANO, Adriano dos Santos; Teste de Software, Revista de divulgação técnico-científico do ICPG, v. 3, n. 9, p. 17-21, jul-dez. 2006.

[9] COPSTEIN, Bernardo; OLIVEIRA, Flávio. Teste Unitário – Introdução. FACIN – PUCRS, Rio Grande do Sul, 2003. Disponível em: <http//:www.inf.pucrs.br/~flavio/tsi/TesteUnitarioIntroducao.ppt>. Acesso em: 23 Agosto 2007.

[10] MARTINS, Eliane; VIEIRA, Vanessa Gindri. Teste de Regressão. São Paulo: Instituto de Computação, UNICAMP, 2001. Disponível em: <http//:www.ic.unicamp.br/~eliane/Cursos/SeminarioTestes/Teste_Regressao.ppt>. Acesso em: 4 outubro 2007.

[11] BARTIÉ, Alexandre. Garantia da qualidade de software. Rio de Janeiro: Campus/Elsevier, 2002, 328p.

[12] ESTEVAM, Alex, Qualidade de Software, UNIP: São Paulo, 2007. Disponível em: <http://www.plugmasters.com.br/sys/colunistas/169/Alex-Estevam>. Acesso em: 30 setembro 2007

[13] Um Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos (Guia PMBOK) Terceira edição, 2004, Project Management Institute.

[14] ROCHA, Pascale C.; BELCHIOR, Arnaldo Dias. Mapeamento do Gerenciamento de Riscos no PMBOK, CMMI-SW e RUP. In: Simpósio Internacional de Melhoria de Processo de Software, 5., 2004, São Paulo, Brasil.

[15] MARTINS, Claudia Garrido. Aplicação das Técnicas de Identificação de Risco em Projetos de E & P. Monografia (Pós-Graduação - MBA em Engenharia Econômica e Financeira), Universidade Federal Fluminense, Rio de Janeiro, Niterói, 2006.

[16] PIETRO-DIÁZ, R., ARANGO, G. Domain analysis and software systems modeling. Califórnia: IEEE Computer Society. 1991, 312p.

[17] CARR, M. et al. Taxonomy Based Risk Identification. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. USA. 1993, 90p.

[18] CAMPELLO, Antonio, et al. OntoPRIME:Ontologia de Riscos para Ambiente de Desenvolvimento de Software Multiprojetos, Pós-Graduação em Ciencia da computação, Universidade Federal de Pernambuco, Recife, 2004.

[19] ROSENBERG,Linda H.; STAPKO, Ruth; GALLO, Albert; Risk-based Object Oriented Testing; In: Annual Software Engineering Workshop, 14., 1999, Greenbelt, Maryland, United States of American.

[20] GERRARD, Paul; Risk-Based E-Business Testing-Part 1: Risk and Test Estrategy. 2000, 17p. Disponível em: http:// www.gerrardconsulting.com/articles/EBTestingPart1.pdf

[21] CHEN, Yanping; PROBER, Rbert L.; Risk-Based Regression Test Selection Strategy; In: IEEE International Symposium on Software Reliability Engineering, 14. 2003, Denver, Colorado, United States of American.

[22] AMLAND, Stale; Risk based Testing and Metrics: Risk Analysis Fundamentals and Metrics for software testing including a financial Application case study. In: EuroSTAR, 5., 1999, Barcelona, Spain.

[23] REDMILL, Felix, Theory and Practice of Risk-based Testing, In: Software Testing, Verification and Reliability, v. 15, n. 1, p. 3-20, mar 2005.

[24] LOPES, Paulo Sérgio N. D. Uma taxonomia da pesquisa na área de engenharia de requisitos. Dissertação (Mestrado em Ciência da Computação), Instituto de matemática e estatística, Universidade de São Paulo, São Paulo, 2002.

[25] GILB, T.. Towards the Engineering of Requirements. In: International Symposium on Systems Engineering, 12., 2002, Seattle, Washington, United States of American.

[26] KENDALL, Richard P., et al. A Proposed Taxonomy for Software Development Risks for High-Performance Computing (HPC) Scientific/Engineering Applications. Pittburgh, PA: Software Engineering Institute, Carnegie-Mellon University. USA, 2007, 38p.

Anexo A

Este anexo exibe o documento de requisitos analisado no estudo de caso da seção 4.5.

Documento de Requisitos do Sistema
Methodology Explorer

Versão 3.0

Histórico de Alterações

Data	Versão	Descrição	Autor
17/11/2002	3.0	Fechamento do escopo com definição de todos os requisitos a serem implementados no TG de Carlos.	Carlos R. S. Júnior
12/08/2002	2.9	Reestruturação do documento. Avaliação dos requisitos anteriores e criação de novos visando adequação às novas funcionalidades desejadas.	Carlos R. S. Júnior
28/09/2001	2.8	Modificação na seção Descrição da interface com o usuário e nos casos de uso do sistema.	Suzana Mesquita
24/09/2001	2.7	Modificação do caso de uso Cadastrar e retirada de vários fluxos secundários	Suzana Mesquita
25/08/2001	2.6	Modificação do caso de uso Exportar	Suzana Mesquita
18/08/2001	2.5	Modificações gerais no documento	Suzana Mesquita
13/08/2001	2.4	Modificação na seção Descrição da interface com o usuário	Suzana Mesquita
23/07/2001	2.3	Modificações gerais no documento	Hermano Perrelli e Suzana Mesquita
14/04/2001	2.2	Modificação da definição da estrutura de árvore no tópico Interface com o usuário.	Suzana Mesquita
13/04/2001	2.1	Modificação do caso de uso RF09: Colar um Componente	Suzana Mesquita

Conteúdo

Introdução

Este documento especifica os requisitos do sistema *Methodology Explorer*, fornecendo aos desenvolvedores as informações necessárias para o projeto e implementação, assim como para a realização dos testes e homologação do sistema.

1.1 Visão geral do documento

Além desta seção introdutória, as seções seguintes estão organizadas como descrito abaixo.

Seção 2 – Descrição geral do sistema: apresenta uma visão geral do sistema, caracterizando qual é o seu escopo e descrevendo seus usuários.

Seção 3 – Requisitos funcionais (casos de uso): especifica todos os casos de uso do sistema, descrevendo os fluxos de eventos, prioridades, atores, entradas e saídas de cada caso de uso a ser implementado.

Seção 4 – Requisitos não-funcionais: especifica todos os requisitos não funcionais do sistema, divididos em requisitos de usabilidade, confiabilidade, desempenho, segurança, distribuição, adequação a padrões e requisitos de hardware e software.

Seção 5 – Referências: apresenta referências para outros documentos utilizados para a confecção deste documento.

1.2 Convenções, termos e abreviações

A correta interpretação deste documento exige o conhecimento de algumas convenções e termos específicos, que são descritos a seguir.

Identificação dos requisitos

Por convenção, a referência a requisitos é feita através do nome da subseção onde eles estão descritos, seguidos do identificador do requisito, de acordo com a especificação a seguir:
[*nome da subseção. identificador do requisito*]
Por exemplo, o requisito funcional [Recuperação de dados.RF016] deve estar descrito em uma subseção chamada “Recuperação de dados”, em um bloco identificado pelo número [RF016]. Já o requisito não-funcional [Confiabilidade.NF008] deve estar descrito na seção de requisitos não-funcionais de Confiabilidade, em um bloco identificado por [NF008].
Os requisitos devem ser identificados com um identificador único. A numeração inicia com o identificador [RF001] ou [NF001] e prossegue sendo incrementada à medida que forem surgindo novos requisitos.

Prioridades dos requisitos

Para estabelecer a prioridade dos requisitos, nas seções 4 e 5, foram adotadas as denominações “essencial”, “importante” e “desejável”.

Essencial é o requisito sem o qual o sistema não entra em funcionamento. Requisitos essenciais são requisitos imprescindíveis, que têm que ser implementados impreterivelmente.

Importante é o requisito sem o qual o sistema entra em funcionamento, mas de forma não satisfatória. Requisitos importantes devem ser implementados, mas, se não forem, o sistema poderá ser implantado e usado mesmo assim.

Desejável é o requisito que não compromete as funcionalidades básicas do sistema, isto é, o sistema pode funcionar de forma satisfatória sem ele. Requisitos desejáveis podem ser deixados para versões posteriores do sistema, caso não haja tempo hábil para implementá-los na versão que está sendo especificada.

Descrição geral do sistema

1.3 Abrangência e sistemas relacionados

O sistema *Methodology Explorer* é uma ferramenta para o processo de desenvolvimento de software. Fornece uma maneira intuitiva e eficiente para definir componentes adequados a uma empresa/projeto. Um componente é uma unidade da metodologia que pode ser manipulada isoladamente, por exemplo artefato, atividade etc.
Utilizando a ferramenta, o usuário - em geral, engenheiro de processos ou projetista de metodologias - poderá cadastrar novos componentes ou criar componentes a partir de outros já existentes. Além disso, poderá alterar, remover e consultar componentes já criados. Tais componentes podem ser exportados da ferramenta, gerando um documento texto, páginas HTML ou um arquivo PDF que podem ser visualizados sem utilizar a ferramenta.
A ferramenta conterá também testes de validação sobre os componentes criados. Estes são baseados no Rational Unified Process [2] (metodologia proposta pela empresa Rational Software Corporation [5]) e servem de ajuda aos usuários, evitando que este cometa pequenos erros.
Diante da facilidade de se definir metodologias, o *Methodology Explorer* contribui de modo decisivo para melhorar a qualidade do processo de desenvolvimento dos projetos de software de uma empresa.

Requisitos funcionais (casos de uso)

1.4 Cadastro

[RF001] Criar componente

Descrição do caso de uso: Este caso de uso permite que o usuário crie e armazene um novo componente no sistema.

Prioridade: ■ Essencial ☐ Importante ☐ Desejável

Entradas e pré-condições: não tem.

Saídas e pós-condição: um componente é cadastrado no sistema

[RF002] Excluir componente

Descrição do caso de uso: Este caso de uso permite que o usuário exclua um componente do cadastro de componentes do sistema. Um componente pode ser excluído de qualquer instanciação de metodologia (árvore).

Prioridade: ■ Essencial ☐ Importante ☐ Desejável

Entradas e pré-condições: recebe como entrada o componente que se deseja excluir

Saídas e pós-condição: o usuário consegue excluir o componente que deseja

[RF003] Alterar componente

Descrição do caso de uso: Este caso de uso permite que o usuário altere os dados de um componente.

Prioridade: ■ Essencial ☐ Importante ☐ Desejável

Entradas e pré-condições: recebe como entrada o componente que se deseja alterar.

Saídas e pós-condição: um componente é alterado no sistema.

1.5 Interface

[RF001] Visualizar Componente

Descrição do caso de uso: Este caso de uso permite que o usuário visualize os dados de um determinado componente (todos os seus atributos, exceto aqueles que são considerados suas propriedades).

Prioridade: ■ Essencial ☐ Importante ☐ Desejável

Entradas e pré-condições: deve receber como entrada o componente que se deseja visualizar.

Saídas e pós-condição: o usuário visualiza o componente desejado

[RF002] Copiar componente

Descrição do caso de uso: Este caso de uso permite que o usuário copie um componente do cadastro de componentes do sistema. Ou seja, copia o componente de onde ele estava e manda a cópia para a área de transferência.

Prioridade: ■ Essencial ☐ Importante ☐ Desejável

Entradas e pré-condições: recebe como entrada o componente que se deseja copiar.

Saídas e pós-condição: o usuário consegue copiar o componente que deseja

[RF003] Colar componentes

Descrição do caso de uso: Este caso de uso permite que o usuário cole o componente armazenado na área de transferência do sistema no local indicado. O conteúdo da área de transferência continua inalterado. Aqui, local refere-se a uma pasta que contém componentes.

Prioridade: ■ Essencial ☐ Importante ☐ Desejável

Entradas e pré-condições: recebe como entrada o componente que se deseja colar e tem como pré-condição a necessidade de existência de alguma informação na área de transferência do sistema.

Saídas e pós-condição: o usuário consegue colar o componente no local desejado.

1.6 Compilação

[RF001] Compilar componente

Descrição do caso de uso: Este caso de uso permite que o usuário compile metodologias. Essa compilação permite que as metodologias sejam analisadas e comparadas entre si.

Prioridade: ☐ Essencial ■ Importante ☐ Desejável

Entradas e pré-condições: deve receber como entrada as metodologias a serem compiladas.

Saídas e pós-condição: os componentes das metodologias são compilados no sistema.

1.7 Importação/Exportação

[RF001] Anexar documentos

Descrição do caso de uso: Este caso de uso permite que anexar documentos gerais a componentes. Por exemplo, anexar o *template* do *Documento de Requisitos* ao fluxo de requisitos.

Prioridade: ☐ Essencial ■ Importante ☐ Desejável

Entradas e pré-condições: deve receber como entrada o caminho absoluto para um arquivo no sistema de arquivos.

Saídas e pós-condição: O documento é anexado ao componente.

[RF002] Exportar metodologia

Descrição do caso de uso: Este caso de uso permite ao usuário a possibilidade de exportar uma metodologia num determinado formato, como XML, por exemplo. O usuário também

tem a opção de escolher se o componente deve ou não ser exportado juntamente com seus anexos.

Prioridade: ■ Essencial ☐ Importante ☐ Desejável

Entradas e pré-condições: A entrada é uma metodologia a ser exportado e seus sub-componentes, ou seja, todos os componentes que um determinada metodologia.

Saídas e pós-condição: Os componentes são exportados para um arquivo em um determinado formato (como XML).

[RF003] Importar metodologia

Descrição do caso de uso: Este caso de uso permite que componentes de uma metodologia exportada sejam importados do sistema de arquivos e apresentados no *Methodology Explorer*. Os componentes, para serem importados precisam estar no mesmo formato utilizado no caso de uso [Importação/Exportação.RF002]. Importar um componente apenas permite manipular o componente dentro do Methodology Explorer. Para inseri-lo de fato, é preciso realizar o caso de uso [Importação/Exportação.RF004]

Prioridade: ■ Essencial ☐ Importante ☐ Desejável

Entradas e pré-condições: A entrada é o caminho absoluto para um arquivo no sistema de arquivos.

Saídas e pós-condição: O componente importado será inserido na(s) árvore(s) de componentes adequada.

[RF004] Salvar metodologia

Descrição do caso de uso: Este caso de uso permite salvar as alterações realizadas nos componentes de uma metodologia.

Prioridade: ☐ Essencial ■ Importante ☐ Desejável

Entradas e pré-condições: A entrada é uma metodologia.

Saídas e pós-condição: um componente é persistido no *Methodology Explorer*.

[RF005] Gerar site de metodologia

Descrição do caso de uso: Este caso de uso permite que um *site* seja gerado para uma metodologia já compilada. O site deve conter também os possíveis artefatos que foram anexados.

Prioridade: ☐ Essencial ■ Importante ☐ Desejável

Entradas e pré-condições: Um componente metodologia é a entrada para o caso de uso que tem, como pré-condição, que a toda a metodologia já esteja salva.

Saídas e pós-condição: um site completo é gerado no sistema de arquivos contendo os arquivos HTML e os artefatos anexados à metodologia.

Requisitos não-funcionais

[NF001] Usabilidade

A interface com o usuário é de vital importância para o sucesso do sistema. Principalmente por ser um sistema que não será utilizado diariamente, o usuário não possui tempo disponível para aprender como utilizar o sistema.

O sistema terá uma interface amigável ao usuário primário sem se tornar cansativa aos usuários mais experientes. Em especial, o módulo de publicação HTML possuirá um wizard para ajudar o usuário.

Prioridade: ■ Essencial ☐ Importante ☐ Desejável

[NF002] Desempenho

Embora não seja um requisito essencial ao sistema, deve ser considerada por corresponder a um fator de qualidade de software.

Prioridade: ☐ Essencial ■ Importante ☐ Desejável

[NF003] Hardware e Software

Visando criar um produto com maior extensibilidade, reusabilidade e flexibilidade, deve ser adotar como linguagem principal de desenvolvimento Java seguindo cuidadosamente as técnicas de orientação a objetos. Entretanto, outras linguagens também poderão ser usadas quando indicações técnicas recomendem.

O uso da linguagem Java permite não especificar qual será o sistema operacional e a máquina em que o programa irá executar. No entanto, essa máquina deverá se comunicar com um sistema de banco de dados.

Prioridade: ☐ Essencial ■ Importante ☐ Desejável

Referências

1. Furlan, J. D. **Modelagem de Objetos através da UML**. São Paulo, Makron Books, 1998.
2. Kruchten, P. **The Rational Unified Process – An introduction**. Addison-Wesley, 1998.
3. Página da disciplina Análise e Especificação de Requisitos. www.cin.ufpe.br/~if119.
4. Página da disciplina Metodologia e Desenvolvimento de Software www.cin.ufpe.br/~mds.
5. Página da empresa Rational Software Corporation www.rational.com.
6. Página do projeto de instanciação de ambientes de desenvolvimento de software convencionais e orientados a domínios (visitada em 18/01/2001) www.cos.ufrj.br/~taba.