PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° MCC07/06

Teste de Software Baseado em Risco

Edson Andrade de Moraes

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. MCC07/06 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Março, 2006

Teste de Software baseado em Risco

Edson Andrade de Moraes

Abstract. This paper presents a bibliographical survey aimed at investigating software testing methodologies based on risk, commonly known by the term risk-based testing. This paper touches the fundamentals of these techniques, the advocated advantages of these, the differences among them, as well as their limitations according to the authors of the papers cited here.

Keywords: software testing, risk.

Resumo. Esta monografia apresenta uma pesquisa bibliográfica realizada com o objetivo de investigar os métodos de testes de software baseados em risco, conhecidos pelo termo em inglês *risk- based testing*. São abordados os fundamentos destas técnicas, as vantagens pretendidas pelas mesmas,as diferenças entre elas bem como suas limitações segundo os autores dos artigos aqui citados.

Palavras-chave: testes de software, risco.

Responsável por publicações:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Esta monografia apresenta o conjunto de propostas encontradas publicadas na literatura de teste de software que enfocam o teste de software baseado em risco. Seu principal objetivo é abordar diferentes metodologias propostas, enfocando os ganhos que são defendidos por seus autores, a aplicabilidade e restrições de uso. Esta é uma pesquisa bibliográfica, feita a partir de artigos disponíveis em meio eletrônico que foram publicadas em revistas especializadas, conferências e sites especializados. O termo *risk-based testing* não possui uma tradução oficial ou aceita para o português, nesta monografia será utilizado o termo teste baseado em risco como tradução do termo em inglês.

Esta monografia inicia-se introduzindo o conceito de risco de software no capítulo 2, e continua no capítulo 3 mostrando o conceito de teste baseado em risco. No capítulo 4 são apresentadas várias técnicas de teste baseada em risco coletadas em diversos artigos, de vários autores. No capítulo 5 é mostrado um exemplo da aplicação de uma das técnicas do capítulo 4 e, por fim, no capítulo 6 estão as conclusões da monografia.

2 Riscos de software

Antes que se possa abordar as metodologias de teste que se baseiam em risco, é necessária uma caracterização do mesmo, que esteja adequada a estas. Dadas às várias definições possíveis que podem ser atribuídas ao conceito, existe a restrição de que, para a execução de testes de software, nos interessam somente riscos que possam se materializar em falhas de software. Não nos interessam riscos de natureza administrativa ou que estejam ligados ao escopo de projeto, como por exemplo, falta de mão de obra adequada ou entraves legislativos. Embora alguns dos artigos abordados utilizem definição de risco de projeto, como em Chen; Prober (2003).

Serão utilizadas duas definições de risco para que possam estar contempladas todas as formas como este é tratado nos artigos pesquisados. A primeira definição é a mesma utilizada no modelo CMM[1], de que risco técnico de software pode ser definido como uma medida da probabilidade e severidade de efeitos adversos inerentes ao desenvolvimento do software que não se adequa aos requisitos funcionais e de performance pretendidos (HIGUERA; RAIMES, 1996). Esta definição restringe de forma eficiente o escopo do conceito de falha de software (não-adequação a requisitos funcionais e não funcionais) e denota que o risco pode ser medido. A segundo definição é mais informal, definimos risco como simplesmente algo que pode ocasionar uma falha no software. Esta definição se faz necessária porque, como veremos adiante, nem todas as metodologias tratam o risco como uma medida.

3 Definições de Teste Baseado em Risco

Em Bach(1999) é utilizada uma definição simplificada do que vem a ser um processo de teste baseado em risco, é a seguinte:

1. Faça uma lista de riscos por prioridade.

---

1 CMM – Capability Maturity Model (HIGUERA; RAIMES, 1996)

2. Realize testes que exploram o risco.
3. A medida que os riscos evaporam e riscos novos aparecem, ajuste seu esforço de teste para focar na tarefa atual.

Apesar de simples, esta definição contém a essência da técnica, ou seja, o foco do teste baseado em risco é na análise do software e na criação de um plano teste baseado nas áreas que possuam a maior probabilidade de apresentarem problemas que teriam o maior impacto sobre o mesmo (MCMAHOM,1998 apud ROSENBERG;STAPKO; GALLO ,1999).

Para melhor ilustrar este conceito, aplicaremos uma das técnica de teste baseado em risco, com uma aplicação muito comum em sites de jornais disponíveis na internet. Em geral, nestes jornais eletrônicos encontramos ao final das matérias a opção de enviar a matéria para outra pessoa. Esta opção é mostrada na figura 1 tendo como exemplo um jornal brasileiro disponível na internet.

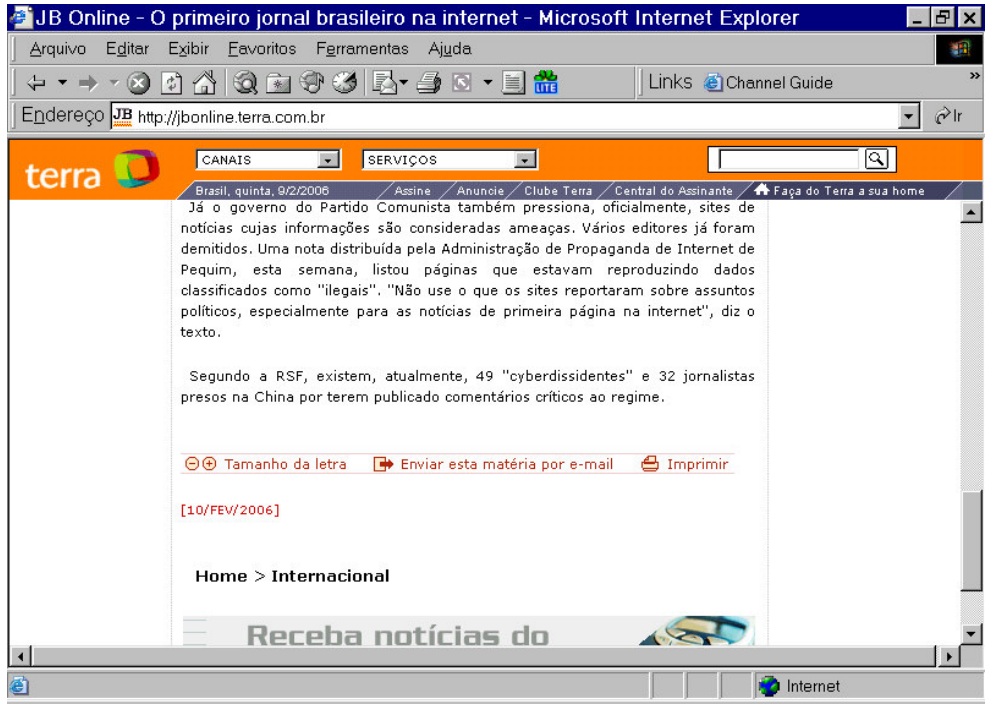

Figura 1 – Opção para enviar a matéria a outra pessoa (JBONLINE.TERRA.COM.BR, 2006)

Utilizaremos para tal, uma das abordagens desenvolvidas por Bach(1999) , a qual este chama de abordagem de dentro para fora. Nesta abordagem um produto é estudado utilizando-se as seguintes perguntas:

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

A partir destas informações então geramos os casso de teste. É interessante notar que apesar do levantamento das vítimas não subsidiar a construção de casos de teste, ela ajuda a compreender o severidade da falha, assim podemos concluir se devemos ou não escrever um caso de teste para aquela situação. Segundo o autor, esta abordagem requer substancial conhecimento técnico do que está sendo desenvolvido. Portanto o membro da equipe encarregado do teste pode não possuir esse conhecimento, assim é necessária a participação do desenvolvedor.

Complementando as informações sobre a aplicação, vejamos que ao clicar sobre a opção enviar matéria, que é retratada na figura acima, o leitor é levado para uma outra página onde ele encontra as opções retratadas na figura 2, para enviar a matéria .

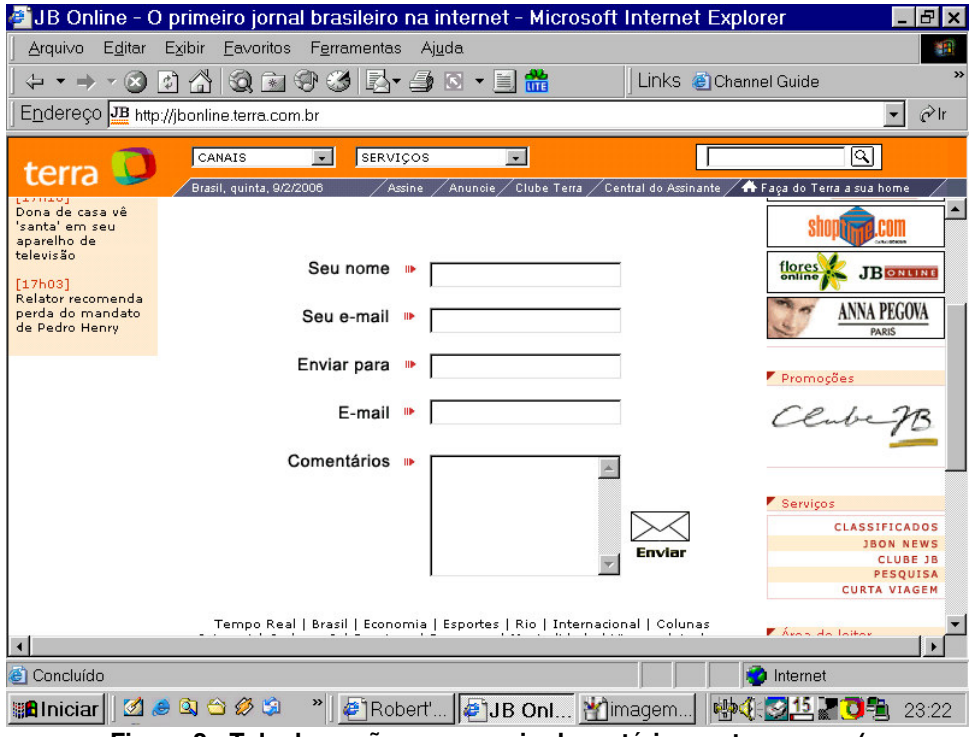

Figura 2 - Tela de opções para envio da matéria a outra pessoa(JBONLINE.TERRA.COM.BR, 2006)

Sendo que como o autor da monografia não possui acesso aos detalhes internos da aplicação vamos supor aqui que ela conta com os seguintes componentes arquiteturais:

- Uma página html que serve como interface para envio da matéria.
- Uma rotina no lado do servidor para enviar as mensagens de correio com a matéria.
- Rotinas de validação do lado servidor para averiguar a se o endereço de e-mail está correto.
- Rotinas que interceptam palavras de baixo calão.

Identifiquemos a seguir os itens de risco pertinentes as ameaças, vulnerabilidades e vítimas. Quanto às vulnerabilidades:

1. O código html pode não funcionar em todos os navegadores.
2. As fontes podem não ser desenhadas corretamente para o usuário, ou seja, ficarem pequenas demais ou grandes demais.

3. Determinado formato de imagem pode não aparecer.
4. Os links podem estar "quebrados"(não levam a página alguma).
5. O e-mail com a matéria pode não ser enviado.
6. A rotina de validação do e-mail pode não funcionar adequadamente.
7. A página pode permitir o envio de e-mails anônimos.
8. A página pode permitir o envio de e-mail com conteúdo ofensivo.

Quanto as ameaças são estas:

1. O usuário pode estar utilizando um navegador menos comum no mercado, uma versão antiga de um navegador popular ou estar rodando o navegador em outra plataforma tecnológica que não seja muito comum (esta ameaça dispara as situações 1, 2, 3).
2. O usuário pode estar com uma resolução de vídeo muito alta ou muito baixa (esta ameaça dispara as situações 2).
3. Uma falha de código durante o desenvolvimento, resultando em nome errado do endereço internet presente no link (esta ameaça dispara as situações 4).
4. O mecanismo de envio de e-mail pode estar fora do ar momentaneamente(esta ameaça dispara as situações 5).
5. O usuário pode entrar com um e-mail incorreto ou que não exista (esta ameaça dispara as situações 5,6).
6. O usuário pode tentar enviar uma mensagem sem endereço do emissor(esta ameaça dispara a situação 7)
7. O usuário pode escrever conteúdo ofensivo(esta ameaça dispara a situação 8.)

Quanto as vítimas, são estas:

1. O usuário uma vez que este pode não obter o serviço que ele espera(enviar um mensagem com uma matéria jornalística em anexo).
2. O destinatário da mensagem, que pode receber uma mensagem indesejada ou de conteúdo ofensivo sem remetente.
3. A empresa de mídia dona do jornal. Uma vez que, este pode ser processado por responsabilidade em permitir que um usuário envie uma mensagem de conteúdo ofensivo sem remetente.
4. Os anunciantes do jornal, uma vez que a maioria dos anúncios de publicidade on-line são veiculados com imagens. Caso estas não carreguem ou seus links estejam quebrados, haveria um dano material a um anunciante do jornal que é uma fonte importante de receita.

Tendo em vista as vulnerabilidades encontradas, vamos criar os casos de teste correspondentes a cada uma das situações acima. Na tabela seguir estão listados então os testes baseados nas vulnerabilidades e ameaças e vulnerabilidades encontradas. Para efeito, consideramos todas a situações como causadoras de danos significativos tendo em vista as vítimas envolvidas.

Situação	Teste
1	Testar nos três navegadores mais usados no mercado em pelo menos duas plataformas diferentes.
2	Testar a exibição da página em pelo menos cinco resoluções diferentes e testar em, monitores de raios catódicos e de cristal líquido.
3	Repetir o teste da situação observando as imagens.
4	Clicar em todos os links da página e ver se vão para o lugar correto(supondo que existem poucos links).
5	Enviar 10 (dez) matérias por e-mail para diversas caixas de correio e verificar se elas chegam. O teste deve ser repetido em horários diferentes para garantir que não há indisponibilidade momentânea durante certos momentos do dia.
6	Criar uma lista com e-mails que estão com endereço incorretos e submetê-los a rotina de validação de e-mail
7	Tentar enviar uma mensagem sem emissor e averiguar se a página aceita.
8	Entra no campo assunto e comentários com palavras de baixo calão e averiguar se a rotina de interceptação destas palavras está funcionando.

Tabela 1 - Testes que exploram os riscos analisados

4 Metodologias

Ao analisar as metodologias propostas podemos notar que três fases bem definidas compõe todas elas:

- Análise dos riscos do software que será alvo do teste.
- Criação do plano de teste baseado nos riscos.
- Execução do plano de teste.

Os artigos analisados focam essencialmente na primeira fase, ou seja na análise dos riscos, com exceção de Bach(1999) que também descreve modos de organização do esforço de teste. Dentre os artigos analisado temos também a seleção de cenários e casos de teste, para teste de regressão, a partir do calculo de exposição ao risco (CHEN; PROBER, 2003), utilizando um modelo desenvolvido em Amland(1999). Nos próximos itens serão abordadas cada uma desta técnicas, essencialmente quanto à fase de análise de riscos para os fins de teste, com exceção do trabalho de Bach(1999) onde também será abordada a organização do esforço de teste.

4.1 Análise de risco baseada em heurística

Bach (1999) defende a utilização de duas abordagens distintas, baseadas em heurísticas, para identificação dos riscos a serem explorados pelo teste. A primeira já foi mostrada no capítulo 3 desta monografia. A outra abordagem seria de fora para dentro, que é executada da seguinte forma: provido de um conjunto de riscos em potencial, estes vão sendo relacionados aos detalhes da situação. Nesta abordagem, se consulta uma lista pré-definida de riscos e se determina se estes riscos são aplicáveis aqui e ago-

ra. Esta lista pode ser baseada em experiência própria ou em listas já existentes. O autor sugere a utilização de três tipos de listas: categorias de critérios de qualidade, listas de riscos genéricos e catálogos de riscos(BACH, 1999).

Nas listas de categorias de critérios de qualidade temos categorias desenvolvidas para atenderem a diferentes tipos de requisitos. As listas abaixo foi desenvolvida a partir dos critérios ISO 9126 [2] a HP Furps [3]:

- Capacitação
- Confiabilidade
- Usabilidade
- Performance
- Instalabilidade
- Compatibilidade
- Suportabilidade
- Testabilidade
- Manutenibilidade
- Portabilidade
- Localizabilidade(BACH, 1999)

Finalmente as listas genéricas de riscos são aquelas que possuem riscos universais a qualquer sistema:

- Complexo: qualquer coisa desproporcionalmente grande, intrincado ou convoluto.
- Novo: qualquer coisa que não possua histórico no produto.
- Modificado: qualquer coisa que tenha sido modificada ou melhorada.
- Dependência em componente superior: qualquer coisa que cuja falha causaria um efeito cascata de falhas no resto do sistema
- Dependente de componente inferior: qualquer coisa que seja extremamente dependente de falhas no resto do sistema.
- Crítico: qualquer coisa cuja falha poderia causar um dano substancial.
- Preciso: qualquer coisa que deva atender a requisitos exatos.
- Popular: que será muito usado.
- Estratégico: qualquer coisa que tenha especial importância para o seu negócio, como uma característica que lhe distingue da competição.
- Terceirizado: qualquer coisa usada no seu produto, más que foi desenvolvida fora do projeto.
- Distribuído: qualquer coisa que esteja espalhada em relação a tempo ou espaço, e que seus elementos devam trabalhar juntos.

---

2 ISO 9126 – Normas de Engenharia de Software; Qualidade do Produto (ISO.ORG, 2006).

3 FURPS(Functionality, Usability, Reliability, Performance, Supportability) – Sistema de classificação de requisitos (EELES, 2005).

- Defeituoso: qualquer coisa que conhecidamente tenha problemas
- Falhou recentemente: qualquer coisa com uma história recente de falha (BACH, 1999)

Os catálogos de risco são listas de riscos que pertencem a um domínio em particular. A seguir há uma versão resumida do que seria um catálogo de riscos para um instalador:

- Instalação dos arquivos errados.
- Arquivos corrompidos.
- Outras aplicações corrompidas.
- Hardware não foi configurado de forma apropriada.
- O protetor de tela interfere no instalador.
- Não há detecção de aplicações incompatíveis.
- O instalador silenciosamente substitui ou modifica arquivos críticos ou parâmetros.
- O processo de instalação é muito lento.
- O processo requer o constante monitoramento do usuário.
- O processo de instalação é confuso(Ibid, 1999).

Bach (1999) também sugere três formas de organização do esforço de teste em torno de riscos. A Lista de observação de riscos é a mais simples de todas. Uma lista de observação de riscos é apenas um lista de riscos que você periodicamente revê e pergunta a si mesmo o que a sua atividade de teste revelou sobre os mesmos.

A Matriz Risco/Tarefa consiste de uma tabela com duas colunas. Na esquerda está uma lista de riscos, na direita está uma lista de tarefas de mitigação associada a cada risco. Ordene os riscos pela importância com a os riscos mais importantes no topo (entenda-se aqui por atividade, a criação e execução dos testes associados aquele risco).

A Matriz Componente/Risco é uma matriz com três colunas como no exemplo a seguir:

Componente	**Risco**	**Heurística**
Impressão	Normal	Distribuído, Popular
Geração de Relatórios	Alto	Novo, estratégico, terceirizado,crítico
Instalação	Baixo	Popular, Usabilidade,Modificado
Biblioteca de imagens	Baixo	Complexo

Tabela 2 - Matriz Componente/Risco(BACH, 1999)

A coluna da esquerda indica o componente do sistema, a do centro o grau de severidade do risco, e a da direita a(s) heurísticas que expõem o risco. A vantagem desta abordagem é que à medida que o projeto progride a atenção é dispensada para os diferentes componentes do sistema de acordo com os níveis de risco associados a este.

4.2 Teste Baseado em Risco para Negócios Eletrônicos

Gerrard (2005) desenvolve um arcabouço(termo em inglês *framework*) para criação de uma estratégia de testes de sistemas de negócios eletrônicos (conhecidos pela termo em inglês *E-Business*) a partir dos cinco principais riscos em aplicações dessa natureza. São eles usabilidade, performance, segurança, disponibilidade e funcionalidade.

Seus argumentos para a utilização de uma abordagem diferenciada de testes se baseiam nas especificidades dos projetos de sistemas deste tipo. De forma resumida são estas:

- Mudanças tecnológicas significativas a cada seis meses. Causando um eterno grau de inexperiência nos desenvolvedores.
- Em algumas organizações tais projetos podem ser mantidos e gerenciados por pessoas que não são possuem conhecimento técnico em computação.
- A prioridade é se lançar ao mercado primeiro, sem que muitas vezes seja atingido o nível de qualidade necessário.
- Existem obstáculos culturais, práticos, técnicos e de cronograma à implementação de práticas mais robustas de controle de qualidade e teste.

Segundo o autor os principais desafios ao teste nestes ambientes são:

- Identificar e endereçar riscos rapidamente, e adquirir consenso sobre a estratégia de teste como um todo.
- Desenvolver e implementar estratégias flexíveis de teste que sejam direcionados aos riscos de maior preocupação.
- Criar novas técnicas de teste e métodos que atendem as restrições particulares da área de negócios eletrônicos.
- Fazer o melhor uso possível da automação de teste em cada área da atividade de teste
- Prover evidencia de teste detalhada para ajudar aos colaboradores no projeto a tomarem a decisão correta sobre o lançamento da versão.

Para Gerrard (2005) usar riscos para priorizar os testes, significa que os alocados ao teste podem se concentrar na concepção de testes mais efetivos em encontrar falhas e não se preocupar em ter realizado testes à menos que o necessário.

A metodologia desenvolvida por autor se difere segundo o próprio, pela execução de testes em todas as fases do projeto e não só ao final do projeto, quando da existência de um sistema executável. Esta forma de trabalho fica representada pelo modelo W da figura abaixo:

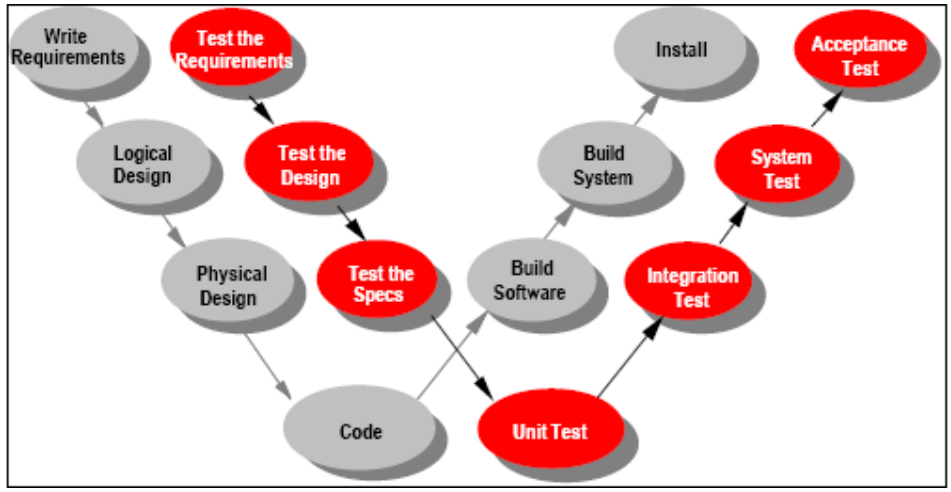

Figura 3 - Modelo W (GERRARD, 2000).

Esta técnica a qual o autor batizou de Total Testing, é um implementação da empresa Evolutif para modelo W. O modelo W promove a idéia de que para toda atividade que gera um artefato que pode se entregue, cada um destes artefatos deve ter uma atividade teste associada (GERRARD, 2000).

O autor faz também algumas consideração em relação a formulação da estratégia de teste:

- Abordagem focada em automação: projete os testes para serem automatizados desde o início. Não haverá tempo suficiente para automatizar os testes manuais depois.
- Teste do desenvolvedor: considere a opção de amarrar os testes realizados no código do desenvolvedor aos procedimentos de check-in/check-out[4]. Em primeiro lugar, isto irá garantir que eles realizarão alguma espécie de teste e em segundo lugar, um conjunto confiável de testes de regressão será mantido durante o desenvolvimento.
- Considere o uso de *test drivers*: estes são frequentemente programas simples que aceitam dados de teste, constroem as chamadas ao código que está no servidor, executam as transações e guardam os resultados para avaliação posterior.

Para atender os riscos de um projeto de negócios eletrônicos, foram identificados 20 tipos distintos de testes. Cada um está direcionada uma área específica de risco. Esta arcabouço tem o propósito de auxiliar na construção de um processo de teste específico para o projeto que está sendo desenvolvido. Os tipos de teste estão agrupados em cinco categorias:

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

4 Termo usado para definir a colocação e extração de códigos fonte em softwares de controle de versão de programas

Os testes ainda podem ser considerados estáticos(não precisa ser executar o programa) ou dinâmicos(precisa executar o programa), e devem ser priorizados segundo a seguinte ordem de área de risco:

- Teste de fumaça(o sistema suporta pelo menos um sessão de uso sem apresentar falha)
- Usabilidade
- Performance
- Funcionalidade

Os testes também são classificados quanto ao estágio:

1. Teste no computador de desenvolvimento(em linhas gerais o que o navegador internet executa)
2. Teste de infra-estrutura(o que roda no servidor)
3. Teste de sistema(do sistema completa em isolamento)
4. Alta escala de integração(com outros sistemas)
5. Monitoramento após entrada em produção(manter os testes de automação para monitorar o site)

Acrescente-se finalmente o fato de que os testes podem ser automatizados(A) ou manuais(M). A partir deste conjunto de classificações utiliza-se uma tabela, na qual partir da análise dos riscos do projeto, se constrói o processo de teste mais adequado ao mesmo.

Suponhamos que atividade de testes de "checagem do conteúdo" , pertence aos riscos relacionados à usabilidade, deve ser testado de forma estática, com uma parte sendo feita via automação e outra sendo executada de forma manual, na fase de desenvolvimento na estação de trabalho. Estas informação estão expressas na primeira linha da tabela 2 que mostra o exemplo de um processo de desenvolvimento, utilizando o arcabouço. Tendo em vista as prioridades anteriormente citadas sabemos também que devemos primeiro executar as atividades que estão marcadas como S na coluna "Fumaça", depois as da coluna "Usabilidade", e assim por diante.

	Prioridades de teste					Tipo de Teste Mapeados para Está-gios 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/ confiabili-dade	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

Tabela 3 - Arcabouço de Processo de Teste (GERRARD, 2000)

4.3 Software Orientado a Objetos

Em Rosenberg; Stapko; Gallo (2002), é proposta a utilização de uma metodologia para identificação de classes que estejam mais propensas a erro(ou seja, representam maior risco de falha) e eventualmente a aplicação de testes nas mesmas. Segundo os autores foi comprovado anteriormente que o código que é mais complexo tem uma maior incidência de erros ou problemas(PFLEEGER, 1990 apud ROSENBERG; STAPKO; GALLO ,2002). Sendo assim através de métricas de complexidade de software orientado a objetos pode se chegar as classes que tem maior probabilidade de falha.

Os autores defendem a utilização de seis métricas de medição de projetos orientadas a objeto, identificadas e aplicadas pelo *Software Assurance Technology Center (SATC)* do *NASA Goddard Space Flight Center*. São estas:

- Número de Métodos (*Number of Methods – NOM*): é uma simples contagem dos diferentes métodos existentes em uma classe.
- Número Ponderado de Métodos Por Classe(*The Weighted Methods per Class – WMC*): é uma soma ponderada dos métodos em uma classe(CHIDAMBER, 1991 apud ROSENBERG; STAPKO; GALLO,2002). Se os pesos são iguais, equivalem à métrica anterior. A Complexidade Ciclomática (MCCABE, 1976 apud ROSENBERG; STAPKO; GALLO, 2002) é utilizada para avaliar. Adicionar pesos aos métodos com sua respectiva complexidade cria uma métrica mais informativa sobre a classe
- Acoplamento entre objetos (*Coupling Between Objects – CBO*): é uma contagem do número de outras classes para a qual uma classe está acoplada, é medida pela contagem do número de hierarquias de classe distintas, excluindo-se herança, relacionadas das quais uma classe depende (CHIDAMBER, 1991 apud ROSENBERG; STAPKO; GALLO,2002) . Classes acopladas devem ser distribuídas junto com a classe a qual se acoplam ou modificadas se elas precisam ser reutilizadas
- A resposta à uma classe (*The Response for a Class - RFC*) é a cardinalidade do conjunto de todos os métodos que podem ser invocados em resposta à uma mensagem para um objeto da classe ou por algum método que pode ser invocado em resposta à uma mensagem para um objeto da classe ou por algum método da classe (CHIDAMBER, 1991 apud ROSENBERG; STAPKO; GALLO,2002)
- Profundidade na árvore (*Depth in Tree – DIT*) A profundidade de uma classe de acordo coma hierarquia de herança 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 herança múltipla utilize o maior DIT.
- Número de filhos (*Number of Children – NOC*): O número de filhos é o número de subclasses que herdam diretamente da classe na hierarquia.

Por mais de três anos, o SATC tem coletado e analisado código orientado a objetos escritos tanto em C++ e em Java. Mais de 20.000 classes foram analisadas, de mais de 15 programas. Os seguintes valores limites para as métricas individuais foram derivados do estudo da distribuição das métricas coletadas.

Métrica	Observação
Número de Métodos:	Preferencialmente menor que 20 e aceitável até 40
Número Ponderado de Métodos Por Classe	Preferencialmente menor que 25 e aceitável até 40
Acoplamento entre objetos	Aceitável até 5
A resposta à uma classe	Aceitável até 50
Profundidade na árvore	Aceitável até 5
Número de filhos	Não há um número de consenso. Más quanto maior, maior a probabilidade de erro.

Tabela 4 - Métricas e Valores Limites(ROSENBERG; STAPKO; GALLO, 2002)

Uma única métrica nunca deveria ser utilizada sozinha para avaliar os riscos do código, é necessária 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 (ROSENBERG; STAPKO; GALLO,2002) . A tabela a seguir é um exemplo de informação que poderia ser obtida a partir de um projeto. As classes excedem os limites estão sombreadas. Esta informação foi obtida a partir das classes de um projeto em Java.

Class	# Methods	CBO	RFC	RFC/NOM	WMC	DIT	NOC
Class 1	54	8	536	9.9	175	1	0
Class 2	7	6	168	24	71	4	0
Class3	33	4	240	7.2	105	2	0
Class7	54	8	361	6.7	117	2	2
Class8	62	6	378	6.1	163	2	0
Class 10	63	7	235	3.7	156	2	0
Class 11	81	10	285	3.5	161	2	0
Class 12	42	5	127	3.0	69	3	0
Class 14	20	17	324	16.2	139	4	4
Class 18	46	5	186	4.0	238	1	3

Tabela 5 - Exemplo de Métricas em um Projeto Java (ROSENBERG; STAPKO; GALLO, 2002)

4.4 Seleção de áreas de teste

Em Amland (1999), o autor desenvolve um conjunto de métricas para a análise de risco com o objetivo de subsidiar um processo de teste. Estas métricas foram aplicadas em

um estudo de caso de uma aplicação em uma instituição financeira. Este foi desenvolvido pelo autor e relatado no artigo. A análise do risco foi desenvolvida antes do início dos testes no sistema, más foi continuamente atualizada durante a sua execução. Como a aplicação possuía dois módulos um on-line e outro que processava em lote (*batch*), análises separadas foram executadas para cada um deles. Para os fins desta monografia darei ênfase ao modelo de análise utilizado no módulo em lote , descrito a seguir.

A empresa financeira, referida como provedora do serviço e vendedor, desenvolveu um modelo para o cálculo da exposição ao risco baseado em:

- A probabilidade de um erro
- O custo (conseqüência) de um erro na função correspondente, tanto para o provedor do serviço como para o cliente.

Na sua metodologia existem três principais fontes de análise de risco:

- Qualidade da função (área) a ser testado. Entenda-se por função o módulo ou programa. Isto foi utilizado como indicação da probabilidade de uma falha-P(f). A presunção de que um função sofre de um projeto de má qualidade, programador inexperiente, funcionalidade complexa, etc, está mais exposta a falhas que funções baseadas em um projeto de melhor qualidade, que foram feitas por um programador mais experiente, etc.
- As conseqüências de uma falha em uma função do ponto de vista de um cliente em uma situação de produção, isto é, probabilidade de ameaça legal, perder posicionamento no mercado, não cumprimento de regulamentações governamentais, etc, por causa de falhas. Esta conseqüência representa o custo para o consumidor C(c).
- As conseqüências de uma falha em uma função do ponto de vista do vendedor do serviço, isto é a probabilidade de: publicidade negativa , altos custos de manutenção de software, etc. , devido a uma função com falhas. Estas conseqüências representam um custo para o vendedor – C(v) (AMLAND, 1999).

A suposição era de que o custo para o consumidor é igualmente importante na análise do risco em relação ao custo do vendedor, e exposição de risco seria dada pela função:

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

Um exemplo de áreas com diferentes perfis de risco seriam ás áreas de "cálculo de juros" e área de "impressão de relatórios internos". Um falha no cálculo de juros poderia facilmente ocasionar um ameaça legal ao banco que utilizasse o software, enquanto uma falha na impressão de relatórios internos não chegaria nem ao conhecimento do público (Ibid, 1999).

Tendo em vista os graus de exposição ao risco, as áreas com risco mais alto tem a maior prioridade no teste. E, durante o teste a medida que falhas vão ocorrendo e novas prioridades podem ser adicionadas ao projeto, os graus de exposição ao risco vão sendo atualizados, a prioridade de teste pode ser modificada.

As áreas de processamento em lote da aplicação foram dividas em sub-áreas(ou funções) como parte de um estágio de projeto lógico. Estas funções foram então utilizadas par análise de risco:

- Cálculo de juros

- Calculo de multas
- Análise de lucratividade
- Exclusão de dados
- Geração de relatórios
- Capitalização

O grande desafio segundo Amland(1999) foi a identificação dos indicadores de custo de falha e de qualidade. Foi utilizada uma abordagem simples más segundo o autor satisfatória:

- Custo de falha: foi indicada por um numero de 1 à 3 com 1 representando o custo mínimo no caso de uma falha nesta função. Os elementos a serem considerados eram:
    - Recurso de manutenção a serem alocados no caso de uma falha(dado que o vendedor proveria o serviço 24 h por dia).
    - Conseqüências legais no caso do vendedor não cumprir com requisitos governamentais.
    - Conseqüências de uma má reputação.

A probabilidade de uma falha foi indicado dando a 4 indicadores um número que varia de 1 à 3, onde 1 era bom com probabilidade de falha baixa. Os indicadores eram:

- Função que foi modificada ou é uma nova funcionalidade.
- Qualidade do projeto da função. Este item era medido pelo número de pedidos de mudança no projeto da função.
- Tamanho. Foi assumido que o número de sub-funções afetaria o número de falhas introduzidas pelo programador.
- Complexidade. A habilidade do programador em compreender as funções que ele estava programando irão sempre ter um efeito sobre o número de falhas.

Para o custo relacionado a cada função em lote foi calculada a média entre o custo par o cliente e o custo para o vendedor. Os indicadores utilizados para calcular a probabilidade de falha de uma função em particular foram ponderados isto é, o peso iria variar de 1 até 5, sendo a nota 5 indicando a maior probabilidade de falha.

Os pesos utilizados pelo vendedor foram

- Função que foi modificada ou nova funcionalidade: 5
- Qualidade do projeto: 5
- Tamanho: 1
- Complexidade: 3

Um exemplo do grau de exposição ao risco calculado para a função "Fechar contas" é mostrado na tabela abaixo. O grau de exposição ao risco foi calculado para todas as funções e a lista foi ordenada para identificar quais áreas deveriam ser focadas durante o teste. A probabilidade é calculada como a Media Ponderada de uma função em particular dividida pela maior média ponderada de todas as funções, dando uma probabilidade na faixa [0,1].

	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

Tabela 6- Cálculo do grau de exposição ao risco (AMLAND, 1999).

4.5 Estratégia de Seleção de Casos de Teste

Em Chen; Prober (2003) encontramos a utilização de métricas de riscos para seleção de suítes de teste a serem aplicadas. Os testes de regressão são essenciais para a garantia da qualidade do software. Ele é o processo de validação do software modificado para prover confiança as partes que foram mudadas do software se comportam como pretendido e que as partes do software que não foram modificadas não foram adversamente afetadas pelas modificações (HARROLD ET AL.,2001 apud CHEN; PROBER, 2003). Seleção de teste de regressão baseados em código é boa para teste de unidade más tem problemas de escalabilidade. À medida que o tamanho do sistema sob teste cresce, se torna mais difícil gerenciar a informação do teste e criar matrizes de rastreabilidade (Ibid, 2003).

Os principais objetivos do teste de regressão são garantir a estabilidade e a confiabilidade de sistemas. A abordagem baseada em risco de Chen; Prober (2003) foca em casos de teste que testam as áreas que possuem maior risco. Como conseqüecia, esta pode nos auxiliar a atingir um nível de confiabilidade adequado na qualidade do software. A metodologia proposta por Chen; Prober (2003) consiste de duas etapas, seleção dos casos de teste e seleção dos cenários de teste ponta-a-ponta.

Esta metodologia utiliza o modelo de risco desenvolvido por Amland(1999) apresentado anteriormente. A fase de seleção de casos de teste envolve as seguintes atividades:

1. Estimar o custo de cada caso de teste. Entenda-se por custo não o custo laboral de executar o teste e sim o custo que uma falha que ocorra naquela área possa causar.
2. Derive a severidade para cada caso de teste
3. Calcule o grau de exposição de risco para cada caso de teste
4. Selecione os casos de teste que tem os maiores valores de exposição ao Risco como Casos Seguros (casos de segurança checam que falhas graves não ocorrem).

Quanto à Seleção de cenários de teste baseados em risco, desde que cenários ponta-a-ponta envolvem muitos componentes do sistema trabalhando juntos, ele são muito efetivos em encontrar falhas de regressão.A seleção de cenários 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 seleção de casos de teste tem os seguintes passos:

1. Calcule a exposição de riscos para cada cenário
2. Selecione os cenários com maior exposição ao risco
3. Atualize a matriz de rastreabilidade (remova os cenários selecionados e os testes já cobertos e recalcule a exposição ao risco)
4. Repita os passos 1 e 2 o quanto desejar

5 Vantagens e Restrições de Testes baseados em Risco

As principais vantagens de utilização destas técnica estão relacionadas a questões de custo, prazos e cultura organizacional. Bach(1999) recomenda a utilização de teste baseado em risco quando outras formas de organização do esforço de teste demandam mais tempo ou recursos que você tenha disponível. Outra vantagem que o autor cita é que uma vez que o foco deste tipo de teste em áreas de maior risco para o software, este acaba por justificar o esforço de teste em relação a missão da atividade de teste, encontrar falhas. O que pode ser útil em culturas organizacionais avessas ao teste.

Realidade esta, a qual é relatada por Gerrard(2005) quando se refere ao ambiente de negócios eletrônicos. Onde o ciclo de vida das versões é muito pequeno, inviabilizando processos formais de qualidade.

Amland(1999) foca suas metodologia na utilização de riscos para fins de teste, explicitamente com o objetivo de redução do custo da fase de teste do projeto e a redução de futuros custo de produção em potencial pela otimização do processo de teste.

Para Chen; Prober(2003), o crescimento contínuo das suítes de teste em tamanho e volume a medida que um sistema de software evolui em funcionalidades pode tornar inviável, ou extremamente custosa a execução de um teste de regressão completo. Por isso uma abordagem baseada em riscos reduz o volume de teste de regressão a ser executado, possibilitando que as áreas mais críticas sejam testadas.

Tendo em vista que as técnicas apresentadas focam na redução do esforço de teste utilizando risco, conclui-se obviamente que, as mesmas não devem ser utilizadas em ambientes de alta criticidade em que uma falha pode causar perda de vida humana ou grande perda financeira. Como exemplo podemos citar sistemas de apoio à vida. Para estes casos é necessário a utilização de abordagens mais formais.

6 Conclusão

Esta monografia analisou algumas das propostas de testes de software baseados em risco. Estas técnicas focam o esforço de teste em áreas do software que possuam maior risco de falha, em comparação com outras técnicas que buscam cobrir todas as funcionalidades do software. Assim obtendo uma atividade de teste menos custosa e mais rápida.

È necessário portanto fazer algumas considerações a respeito. Como citada na seção anterior esta técnica possui limitações. Principalmente em se tratando de sistemas de alta criticidade, nos quais o custo da falha é muito alto.

Nota-se também uma certa semelhança destas técnica, principalmente a proposta de Bach(1999), com uso de listas de condições de contorno. Em que é apresentado um conjunto de condições que podem ocorrer durante a execução do programa, e então averiguamos como este se comporta nestas condições. Ou seja podemos dize que há uma semelhança com testes baseados em checklist de forma geral.

Também é necessário levar em consideração que a identificação e a avaliação dos riscos técnicos de um software pode não ser uma tarefa simples, dependendo do domínio da aplicação, tamanho da mesma, publico alvo, tecnologia utilizada, etc. Estes fatores podem complicar o processo de análise de risco. Neste casos, não há nenhum indicativo claro que abordagens de teste funcional caixa-preta, por exemplo, não seriam mais baratas ou até mesmo mais rápidas que o uso técnicas baseadas em risco.

Torna-se esta então uma das maiores questões em aberto nos artigos. Não existe uma indicação clara de em quanto realmente é reduzido o esforço(custo, tempo, mão-de-obra, etc.) de teste. Não há uma comparação numérica de volume de atividades de teste usando teste baseado em risco e outras abordagens consideradas mais tradicionais.

Dentre as técnicas pesquisadas, Chen; Prober(2003) merecem um certo destaque. pois as dúvidas colocadas acima não se aplicam tão fortemente em relação a abordagem dos autores. Eles utilizam a análise de risco par selecionar testes de regressão. Ou seja estamos falando de uma aplicação que é conhecida dos desenvolvedores, logo avaliar seus riscos não constitui uma tarefa tão complexa que poderia aumentar o esforço ao invés de reduzir.

7 Referências Bibliográficas

AMLAND, Ståle; **Risk based Testing and Metrics**; EuroSTAR '99, November 8 - 12, 1999, Barcelona, Spain. Tradução livre do autor desta Monografia. Disponível também em:
http://www.amland.no/WordDocuments/EuroSTAR99Paper.doc

BACH, James; **James Bach on Risk-Based Testing: How to conduct heuristic risk analysis**; Software Testing & Quality Engineering Magazine; p. 23-28, nov. de 1999. Tradução livre do autor desta Monografia.

CHEN, Yanping; PROBER, Rbert L.; **Risk-Based Regression Test Selection Strategy**; 14th. IEEE International Symposium on Software Reliability Engineering – ISSRE 2003. Disponível em:
http://www.chillarege.com/fastabstracts/issre2003/161-FA-2003.pdf

CHIDAMBER S.R. & Kemerer, C.F., **Towards a Metrics Suite for Object Oriented**

Design Proc. OOPSLA, 1991.

EELES, Peter; **Capturing Architectural Requirements.** Disponível em:
http://www-128.ibm.com/developerworks/rational/library/4706.html

GERRARD, Paul; **Risk-Based E-Business Testing-Part 1**, **Risk and Test Estrategy**. Disponível em:
http://www.software-engineer.org acessado em 26 de out. de 2005.

HARROLD, Mary Jean; JONES,James A.; LI, Tongyu; LIANG Donglin; **Regression Test Selection for Java Software**, Proc. of the ACM Conf. on OO Programming, Systems, Languages, and Applications (OOPSLA '01), 2001, p. 312 – 326.

HIGUERA, Ronald P.; RAIMES, Yacov, Y.; **Software Risk Management-Technical Report CMU-SEI-96-TR-012**; Jun. de 1996. Tradução livre do autor desta Monografia. Disponível em:
http://www.sei.cmu.edu/pub/documents/96.reports/pdf/tr012.96.pdf

ISO.ORG ; **ISO 9126 : Software Engineering -- Product Quality – part1: Quality Model.** Disponível em:
http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=22749&ICS1=35&ICS2=80&ICS3. Acessado em 14 de fev. de 2006.

JBONLINE.TERRA.COM.BR; **JB Online**; Disponível em:
http://jbonline.terra.com.br . Acessado em 12 de fev. de 2005.

MCMAHON, Keith; **Risk Based Testing**, ST Labs, WA, 1998 apud ROSENBERG,Linda H.; STAPKO, Ruth; GALLO, Albert ;Risk-based Object Oriented Testing; NASA SEW24 - Twenty-Fourth Annual Software Engineering Workshop. Tradução livre do autor desta Monografia.

PFLEEGER, S.L. and Palmer, J.D.; **Software Estimation for Object Oriented Systems**; Int'l. Function Point Users Group Fall conference, San Antonio TX, 1990

ROSENBERG,Linda H.; STAPKO, Ruth; GALLO, Albert ;**Risk-based Object Oriented Testing**; NASA SEW24 - Twenty-Fourth Annual Software Engineering Workshop. Disponível em:
http://sel.gsfc.nasa.gov/website/sew/1999/topics/rosenberg_SEW99paper.pdf