UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
ESPECIALIZAÇÃO EM ENGENHARIA DE *SOFTWARE*

JOSÉ ARNILDO WELTER NETO

TESTE DE SOFTWARE EM UM AMBIENTE VIRTUAL DE APRENDIZAGEM MOODLE: UMA ABORDAGEM UTILIZANDO *RISK BASED TESTING*

MONOGRAFIA DE ESPECIALIZAÇÃO

MEDIANEIRA
2011

JOSÉ ARNILDO WELTER NETO

TESTE DE SOFTWARE EM UM AMBIENTE VIRTUAL DE APRENDIZAGEM MOODLE: UMA ABORDAGEM UTILIZANDO *RISK BASED TESTING*

Monografia apresentada como requisito parcial à obtenção do título de Especialista na Pós Graduação em Engenharia de *Software*, da Universidade Tecnológica Federal do Paraná – UTFPR – Campus Medianeira.

Orientador: Prof. Me.Fernando Schütz.

MEDIANEIRA

2011

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Especialização em Engenharia de *Software*

TERMO DE APROVAÇÃO

Teste de *software* em um ambiente virtual de aprendizagem Moodle: uma abordagem utilizando *risk based testing*

Por

José Arnildo Welter Neto

Esta monografia foi apresentada às 08h50min do dia 10 de dezembro de 2011 como requisito parcial para a obtenção do título de Especialista no curso de Especialização em Engenharia de *Software*, da Universidade Tecnológica Federal do Paraná, Câmpus Medianeira. Os acadêmicos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Me. Fernando Schütz
UTFPR – Câmpus Medianeira
(orientador)

_____
Prof. M.Eng. Juliano Lamb
UTFPR – Câmpus Medianeira

_____
Prof. Me. Pedro Luiz de Paula
UTFPR – Câmpus Medianeira

Dedico esse trabalho aos meus pais, José e Marciana, a minha noiva e minhas irmãs, pela paciência, pelo tempo cedido para meus estudos, por entenderem minha distância durante esse tempo e pelo apoio recebido, nunca esquecendo Deus que esta por trás de todas estas conquistas e também ao meu Orientador.

“Experiência? Quem a tem,

se a todo o momento tudo se renova”.

(autor desconhecido)

RESUMO

WELTER NETO, José A. Teste de *software* em um ambiente virtual de aprendizagem Moodle: uma abordagem utilizando *risk based testing*. 2011. 54p. Monografia (Especialização em Engenharia de *Software*). Universidade Tecnológica Federal do Paraná, Medianeira, 2011.

Este trabalho apresenta a abordagem de teste de *software risk based testing* onde uma de suas técnicas será aplicada ao Ambiente Virtual de Aprendizagem Moodle. Tal abordagem possui o foco direcionado aos riscos associados ao *software*, assim propondo que o processo de teste pode ser reduzido a partir da identificação e priorização dos riscos, possibilitando o direcionamento às áreas de maior probabilidade de ocorrência de falhas. O estudo de caso é realizado utilizando a análise dos riscos técnicos para aplicação da metodologia baseada em heurística de fora para dentro, a fim de atestar a validade, agilidade e efetividade da abordagem e como a técnica pode ser utilizada para testar o *template* customizado para o Ambiente Virtual de aprendizagem Moodle.

Palavras-chave: risco, qualidade de *software*, engenharia de *software*.

ABSTRACT

WELTER NETO, José A. Test of software in a learning management system Moodle: an approach using risk based testing. 2011. 54p. Monografia (Especialização em Engenharia de *Software*). Universidade Tecnológica Federal do Paraná, Medianeira, 2011.

This paper presents the approach of software testing where a risk based testing their techniques will be applied to the Moodle Virtual Learning Environment. Such an approach has the focus directed to the risks associated with the software, thus proposing that the testing process can be reduced by identifying and prioritizing risks, allowing the direction to areas of higher probability of failure. The case study is performed using risk analysis techniques for application of the methodology based on heuristics from the outside in order to attest to the validity, responsiveness and effectiveness of the approach and how the technique can be used to test the custom template for the virtual learning environment Moodle.

Keywords: risk, software quality, software engineering.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com o crescimento constante da informatização, a demanda por *softwares* e sistemas que atendessem a esta demanda de forma ágil, com qualidade e baixo custo estimulou empresas e desenvolvedores a buscarem metodologias e técnicas de engenharia de *software*, visando diminuir os custos e prazos de desenvolvimento e em paralelo aumentando a qualidade e confiabilidade dos *softwares* produzidos. A engenharia de *software* apresenta métodos e modelos de desenvolvimento que objetivam a obtenção de qualidade, e um dos processos vitais a garantia de qualidade é o teste de *software*, no entanto o processo de teste não é uma tarefa fácil de ser executada, pois requer profissionais capacitados com conhecimento específico do *software* em desenvolvimento e tempo, assim podendo gerar atrasos e aumento nos custos do projeto.

Segundo BASTOS, RIOS, *et al.*(2007) no fim da década de 90 e inicio do ano 2000 ocorre a explosão da Internet e o surgimento de inúmeras novas tecnologias, com isso a imagem das empresas começou a ser exposta em um cenário de grande veiculação, tal fato levou as organizações a investirem em testes mais elaborados com intuito de garantir que seus negócios não fossem prejudicados transformando então o teste de *software* em um processo fundamental e independente na busca por garantir a qualidade dos *softwares* desenvolvidos.

Para que está demanda fosse atendida, BASTOS, RIOS, *et al.*(2007) afirmam que o processo de teste de *software* passou a ser executado não mais no processo de desenvolvimento, mas sim em paralelo ganhando organizaçao, planejamento e sendo conduzido por profissionais especializados e qualificados. Os resultados desse cenário surgiram imediatamente, empresas preocupadas em reduzir os custos de desenvolvimento e em manter sua imagem perante os clientes e concorrentes, passaram a investir em teste de *software*.

No entanto, em paralelo a essa demanda, houve também a necessidade de investimento, o que acrescentava custo e tempo no processo, onde em muitos casos a execução dos teste tomava mais tempo que as atividades de planejamento e codificação do próprio *software*.

Com o aumento da concorrência, os prazos ficaram cada vez mais curtos, não havendo tempo suficiente para uma efetiva execução dos testes, em grande parte

dos casos o *software* acabava sendo entregue ao cliente com pouca cobertura de testes, deixando assim que erros graves passasem despercebidos e consequentemente prejudicando a empresa e seus clientes (BARTIÉ, 2002).

A partir desse cenário, o *risk based testing* tem se tornado cada vez mais útil, utilizando de maneira mais efetiva os recursos envolvidos e o tempo de teste, executando processos que foquem nas áreas de maior risco.

1.1 OBJETIVO GERAL

Descrever a aplicação da abordagem de teste de *software risk based testing* em um ambiente virtual de aprendizagem Moodle.

1.2 OBJETIVOS ESPECÍFICOS

- Desenvolver um estudo bibliográfico sobre engenharia de *software* focando qualidade, teste, análise de riscos e suas técnicas;
- Aplicar o método de fora para dentro no teste de um ambiente moodle com *template* customizado, apresentando seus resultados.

1.3 JUSTIFICATIVA

Devido a crescente demanda pelo desenvolvimento ágil com baixo custo e de excelente qualidade, a proposta da abordagem se torna viável por explorar as áreas de maior probabilidade de riscos o que é um dos aspectos que mais viabilizam a presença de falhas.

A escolha do ambiente virtual de aprendizagem Moodle para o estudo de caso foi devido à crescente demanda por ambientes virtuais personalizados para

diferentes projetos na Fundação Parque Tecnológico Itaipu e em sua grande maioria são projetos que necessitam ser executados em curto prazo.

A aplicação do método de fora para dentro é viável pois é aplicada diretamente nas áreas em que a customização tem maior impacto nos projetos executados, assim reduzindo o tempo de entrega do produto testado eliminando as falhas mais prováveis.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está estruturado em cinco capítulos:

Primeiro capítulo: apresenta as informações sobre o trabalho, à introdução e os objetivos gerais e específicos;

Segundo capítulo: apresenta os principais conceitos da engenharia de *software,* qualidade, teste e a abordagem *risk based testing*;

Terceiro capítulo: apresenta o software que será testado e o estudo de caso;

Quarto capítulo: apresenta os resultados do estudo de caso;

Quinto capítulo: é composto pela conclusão do tema e apresentação de trabalhos futuros;

2 ESTUDO BIBLIOGRÁFICO

Este capítulo aborda os resultados de pesquisas bibliográficas sobre o ambiente virtual de aprendizagem Moodle, Engenharia de *Software*, focando principalmente a qualidade, teste e os riscos associados ao desenvolvimento.

2.1 MOODLE

De acordo com NAKAMURA (2008) a palavra Moodle refere-se ao acrônimo, *Modular Object-Oriented Dynamic Learning Enviroment*, ou seja, sistema modular de ensino à distância orientado a objetos, em inglês a palavra é também um verbo que que descreve a ação, que com frequência, conduz a resultados criativos, de deambular com preguiça, enquanto se faz com gosto o que for aparecendo para fazer.

O termo orientado a objetos está, na verdade relacionado ao desenvolvimento do sistema, pois se trata de um paradigma de análise, projeto e programação de *software*, baseado na composição e interação entre diversas unidades chamadas de objetos (PRESSMAN, 2002).

Criado em 2001, segundo Martin Dougiamas desenvolvedor do projeto e mantenedor até o presente momento a proposta do Moodle é bastante diferenciada. Trata-se de aprender em colaboração em ambiente on-line, baseada na pedagogia sócio construtiva. Tratando a aprendizagem como atividade social além de concentrar a atenção na aprendizagem que ocorre enquanto construímos ativamente os artefatos (textos, imagens, vídeos, etc.) para que os outros utilizem (NAKAMURA, 2008).

Segundo o site Moodle.org atualmente é utilizado em mais de 220 países e possuí mais de 72.000 sites registrados. É distribuído gratuitamente, pode ser instalado em diversos sistemas operacionais, conta com um grupo de desenvolvimento ativo que trabalha em conjunto com usuários, adaptando a ferramenta para diferentes necessidades. Atualmente encontra-se na versão 2.2 (NAKAMURA, 2008).

Figura 1 – Aplicação Moodle no tema Padrão, em um navegador WEB

2.2 ENGENHARIA DE *SOFTWARE*

Engenharia de *software* é uma área do conhecimento da informática voltada à especificação, desenvolvimento e manutenção de *software*, aplicando métodos e práticas objetivando organização, produtividade e qualidade.

Segundo SOMMERVILLE(2007), engenharia de *software* é uma disciplina relacionada com os aspectos da produção de *software*, está presente desde o inicio das especificações até a manutenção.

Mesmo que existam diversas definições para a engenharia de *software*, PRESSMAN (2002) frisa que todas reforçam a exigência da mesma no desenvolvimento.

Para PRESSMAN (2002) a engenharia de *software* envolve um conjunto básico de três elementos fundamentais para o seu desenvolvimento, sendo eles:

- Ferramentas: oferecem suporte automatizado aos métodos;
- Métodos: especificam qual a melhor maneira de se desenvolver;
- Procedimentos: são interligações entre ferramentas e métodos.

Assim, proporcionando o desenvolvimento de um produto de qualidade, a partir de processos ágeis e transparentes de desenvolvimento (PRESSMAN, 2002; SOMMERVILLE, 2007).

Atualmente inúmeras empresas utilizam e dependem de sistemas, na maior parte dos casos, sistemas complexos são responsáveis por automatizar quase todas as áreas de uma empresa. O uso de sistemas se torna a cada instante mais comum em nosso cotidiano, assim a necessidade de desenvolver e manter *softwares* cresce em volume constante, requerendo que custos e agilidade de desenvolvimento sejam reduzidos. Analisando esse contexto, SOMMERVILLE (2007) afirma que a engenharia de *software* tem foco no desenvolvimento de qualidade e baixo custo.

Para que sistemas de qualidade sejam fabricados, são fornecidas metodologias, técnicas e ferramentas com o objetivo de alcançar a qualidade desejada.

É uma tecnologia baseada em camadas, onde qualquer área deve ter como foco principal a qualidade. Assim a camada de qualidade serve de apoio à para que o processo de evolução se mantenha focado na qualidade, gerando com isso maturidade e consistência nas abordagens seguindo as metodologias apresentadas (PAULA FILHO, 2003).

Uma área de conhecimento que objetiva garantir a qualidade através da definição e normatização de processos de desenvolvimento é a qualidade. O principal objetivo da qualidade de *software* é garantir um produto final que atenda as necessidades do cliente, dentro do que foi acordade no projeto.

2.3 ENGENHARIA PARA *WEB*

Segundo PRESSMAN (2002), sistemas e aplicativos web são caracterizados por disponibilizar grande quantidade de conteúdo e funcionalidade para um grande número de usuários, é portanto o processo utilizado para criar aplicativos *Web* de alta qualidade. A Engenharia *Web* não é igual à Engenharia de *Software* tradicional, mas compartilham muitos conceitos e princípios fundamentais, com ênfase nas mesmas técnicas de gerenciamento e atividades. Existem pequenas diferenças na maneira como essas atividades são conduzidas, mas a filosofia que dita uma

abordagem disciplinada para o desenvolvimento de um sistema de computador é a mesma.

Ao mesmo tempo em que adota muitos princípios da Engenharia de *Software*, a Engenharia *Web* incorpora novas abordagens, metodologias, ferramentas, técnicas e normas para atender os requisitos exclusivos dos sistemas para a *Web*. O desenvolvimento é significantemente diferente do desenvolvimento de *software* tradicional e apresenta vários desafios adicionais. Também é um erro achar que o desenvolvimento de aplicativos para *Web* é apenas a criação de páginas HTML, e da mesma forma é equivocado achar que desenvolvimento para a *Web* envolve apenas a manipulação de diversas mídias e criação de conteúdo. O desenvolvimento para a *Web* é uma mistura de publicações impressas e desenvolvimento de *software*, entre *marketing* e computação, entre comunicações internas e relações externas, e entre arte e tecnologia (SOMMERVILLE, 2007).

A construção de um sistema para a *Web* necessita do conhecimento de pessoas de diferentes áreas. Como resultado, a Engenharia para a *Web* é multidisciplinar, e dela participam áreas como: análise de sistemas e projetos; engenharia de *software*; engenharia de hipermídia e hipertexto; engenharia de requisitos; interação humano-computador; desenvolvimento de interface de usuário; engenharia de informação; indexação e recuperação de informações; teste; modelagem e simulação; gerenciamento de projetos; e projeto gráfico e apresentação (PRESSMAN, 2002).

2.4 QUALIDADE DE *SOFTWARE*

Um *software* de qualidade precisa ser de fácil utilização, funcionamento estável, de pouca e simples manutenção e que mantenha a integridade dos dados em caso de falhas. Porém esse cenário nem sempre é encontrado, pois a maior parte dos *softwares* é de difícil usabilidade, apresentam falhas constantes e nem sempre suprem as necessidades do usuário, tornando-se assim necessária a presença de técnicos especializados para que modificações, correções ou inserção de novas funcionalidades possam ser executadas. Isso ocorre porque são *softwares* que possuem estrutura de código difíceis de serem compreendidos, falham

constantemente e não mantém a integridade dos dados em caso de falhas (ROCHA, MALDONADO e WEBER, 2001).

Segundo PRESSMAN (2002), há algum tempo atrás poucos comentários surgiam a respeito de custos resultantes de defeitos e erros ocasionados por falhas de *software*, isso tanto por parte de usuários como de desenvolvedores. Com o aumento constante do uso de tecnologias em todos os níveis de atividades a qualidade de *software* têm se tornado cada vez mais importante e consistente.

A qualidade de *software* tornou-se algo que todos buscam. Os gerentes de projeto reconhecem que precisam ter alta qualidade em seus *softwares* e os usuários reconhecem que seu trabalho por meio de *software* deve ser consistente e confiável, assim o desenvolvimento de *software* de qualidade depende de no mínimo duas pessoas, o desenvolvedor e o usuário, além dos casos de testes e métodos de métrica trabalhando em conjunto (KOSCIANSKI e SOARES, 2007).

De acordo com MOLINARI (2008), utiliza-se teste de *software* como atividade principal para garantir a qualidade e métricas para fornecer suporte as tomadas de decisão no desenvolvimento, fornecendo métodos para a avaliação e redução dos riscos inerentes ao desenvolvimento.

De acordo ROCHA, MALDONADO e WEBER (2001), para ajudar a definir a qualidade a ISO (International Organization for Standardization) e a IEC (International Eletctrotechnical Comission) que são organizações normatizadoras com reconhecimento internacional no seguemento de *software*, reuniram-se a fim de estabelecer normas internacionais para a obtenção de qualidade tanto do *software*, quanto dos processos de desenvolvimento.

O gerenciamento da qualidade diminui custos no desenvolvimento por detecção antecipada de defeitos que podem ser corrigidos previamente, prevenindo assim que o número de falhas encontradas posteriormente aumente os custos com manutenção do *software*.

A qualidade dificilmente é alcançada através de um produto já pronto. O ponto forte é prevenir o nível de defeitos e deficiências, obtendo qualidade a partir de medições, com estruturação de processos, métodos, ferramentas e técnicas conforme proposto por MOLINARI (2003).

Conforme descrito por BARTIÉ (2002), os fatores principais na redução dos custos de desenvolvimento são a prevenção de defeitos, que é a inspeção do produto avaliando e medindo a conformidade com os padrões e especificações, a

busca por falhas internas que devem ser corrigidas antes da entrega do produto e falhas externas que são os problemas ocorridos após a entrega do *software*. Sendo as falhas externas as que mais geram custos no desenvolvimento.

O retorno do investimento é garantido com a detecção e prevenção, obtendo-se assim a redução dos defeitos encontrados pelos usuários, não necessitando de manutenções posteriores (KOSCIANSKI E SOARES, 2007).

2.4.1 Garantia de qualidade de software

O objetivo da garantia de qualidade de *software* é fornecer aos gerentes de projeto a visibilidade da eficácia do processo que está sendo utilizado pelo projeto de desenvolvimento, bem como da qualidade do produto que esta sendo desenvolvido.

MOLINARI (2003) específica que a garantia de qualidade de *software* parte do principio que, se o sistema não for construído seguindo o enfoque da qualidade de *software*, o processo de desenvolvimento pode perder o seu foco. Um trabalho de garantia de qualidade abrange seis dimensões para métodos e ferramentas de desenvolvimento que são:

- Revisões formais;
- Estratégias de testes;
- Controle de documentação;
- Histórico de manutenções;
- Procedimento de adequação a padrões de desenvolvimento;
- Mecanismos de medição.

O grande problema é que em grande parte dos *softwares* duas dessas dimensões não são aplicadas, sendo elas as estratégias de testes e os mecanismos de medição, conforme apresentado por MOLINARI (2003).

Segundo SOMMERVILLE (2007), os testes são desvalorizados pelos desenvolvedores. Existem diversas ferramentas de automação de testes, entre tanto é necessário selecionar a que melhor se adapta ao *software* em desenvolvimento, para que se tenha um processo de teste claro e definido. O tempo gasto com os

testes será recompensado durante o próprio processo de testes, sendo diluído no tempo economizado posteriormente na solução de falhas que foram eliminadas durante este processo.

De acordo com BARTIE (2002), o grande problema com os mecanismos de medição é a falta de definição sobre o que será medido. Existem dois grandes grupos que podem ser medidos de:

- Forma direta: número de erros do *software*, linhas de código;
- Forma indireta: usabilidade, manutenibilidade.

A garantia de qualidade de *software* resume-se em um conjunto de passos necessários para proporcionar a confiança de que os processos sejam praticados, melhorados continuamente e organizados de uma maneira que possa atender as especificações do escopo do projeto, visando atender as necessidades de uso do *software* (ROCHA, MALDONADO e WEBER, 2001).

A garantia de qualidade de *software* é atingida através de um plano de SQA (*Software Quality Assurance*) que estabeleça os métodos a serem utilizados no projeto, visando garantir que os artefatos e produtos sejam gerados e revisados a cada passo do projeto (PRESSMAN, 2006).

2.4.2 Componentes de qualidade de Software

Grande parte das atividades de garantia de qualidade pode ser categorizada entre teste de *software*, gerenciamento de configuração e controle de qualidade, no entanto dependem também de um grupo de padrões, práticas, convenções e especificações.

O controle de qualidade garante que as atividades do projeto ocorram conforme o planejado. As atividades de controle da qualidade também podem descobrir falhas no projeto e indicar mudanças que devem melhorar a qualidade do *software*.

O gerenciamento de configuração tende a controlar as alterações, estabelecendo um relacionamento entre o produto e o trabalho, definindo métodos de gerenciamento de versões. É uma atividade de apoio ao desenvolvimento de *software*, o qual ajuda a trabalhar com as mudanças que ocorrem durante o ciclo de

vida do *software* e que é parte fundamental do CMMI (*Capability Maturity Model Integration*).

2.4.3 O CMMI

Uma das premissas do CMMI é o foco no processo e no produto, pois focado somente no produto não se obtém o conhecimento necessário para melhor produzi-lo, isso por desenvolver sem ter analisado, projetado e constantemente melhorado o processo de desenvolvimento do *software* (SOMMERVILLE, 2007).

De acordo com PRESSMAN (2006) a melhoria contínua é baseada em pequenos passos evolutivos e não em grandes inovações. O CMMI estabelece cinco níveis de maturidades para o alcance da melhoria contínua, sendo eles:

- Nível 1 – Realizado: completa falta de planejamento e controle dos processos;
- Nível 2 – Gerenciado: processos básicos de gerenciamento de projetos para planejar custos, prazos e funcionalidades;
- Nível 3 – Definido: atividades de gerenciamento básico e atividades de engenharia de *software* são documentadas, padronizadas e integradas;
- Nível 4 – Quantitativamente gerenciado: métricas detalhadas do processo de *software* e da qualidade do produto são coletadas;
- Nível 5 – Otimização: a melhoria contínua do processo é estabelecida por meio de sua avaliação quantitativa e da implantação planejada e controlada de tecnologias e ideias.

TONSIG (2008) explica que o CMMI identifica os níveis a partir dos quais uma organização deve evoluir para estabelecer uma cultura de excelência em engenharia de *software*. Sendo que através dos cinco níveis de maturidade, a capacidade do processo interage com as pessoas e tecnologias e vai aumentando gradativamente conforme o amadurecimento da organização.

2.4.4 MPS.BR

Segundo a SOFTEX (2011) o MPS.BR está em acordo com as normas ISO (*Intenational Organization for Standardization*) e com o CMMI (*Capability Maturity Model Integration*), através da inclusão de processos e resultados distribuídos em níveis de maturidade, assim tornado possível que a implementação se adeque a cultura das empresas brasileiras, possibilitando uma evolução mais gradativa nos níveis de maturidade, com custos mais acessíveis.

De acordo com a SOFTEX (2011) são sete os níveis de maturidade do MPS.BR:

- Nível A: em otimização;
- Nível B: gerenciado quantitativamente;
- Nível C: definido;
- Nível D: parcialmente definido;
- Nível E: largamente definido;
- Nível F: parcialmente gerenciado;
- Nível G: gerenciado.

O progresso e o alcance de um determinado nível de maturidade são obtidos quando são atendidos todos os requisitos e resultados do processo referente ao nível correspondente.

2.4.5 Norma ISO/IEC 9126

A ISO/IEC 9126 oferece um grupo de itens e subitens que possibilitam mensurar a qualidade de um produto, é composta por quatro documentos, que são:

- ISO/IEC 9126-1;
- ISO/IEC 9126-2;
- ISO/IEC 9126-3;
- ISO/IEC 9126-4;

A ISO/IEC 9126-1 apresenta um grupo de itens para definir a qualidade e podem ser utilizadas em qualquer produto de *software*, e é formado por duas partes, modelo de qualidade interno e externo e modelo de qualidade em uso.

A fim de avaliar os itens de qualidade externas são utilizadas as métricas externas, ou seja, métricas baseadas nas necessidades do usuário, apresentadas na ISO/IEC 9126-2. Para avaliar os itens de qualidade interna, as métricas internas são utilizadas, que estão dispostas na ISO/IEC 9126-3. A ISO/IEC 9126-4 apresenta métricas para a qualidade da usabilidade (ISO COPYRIGHT OFFICE, 2001).

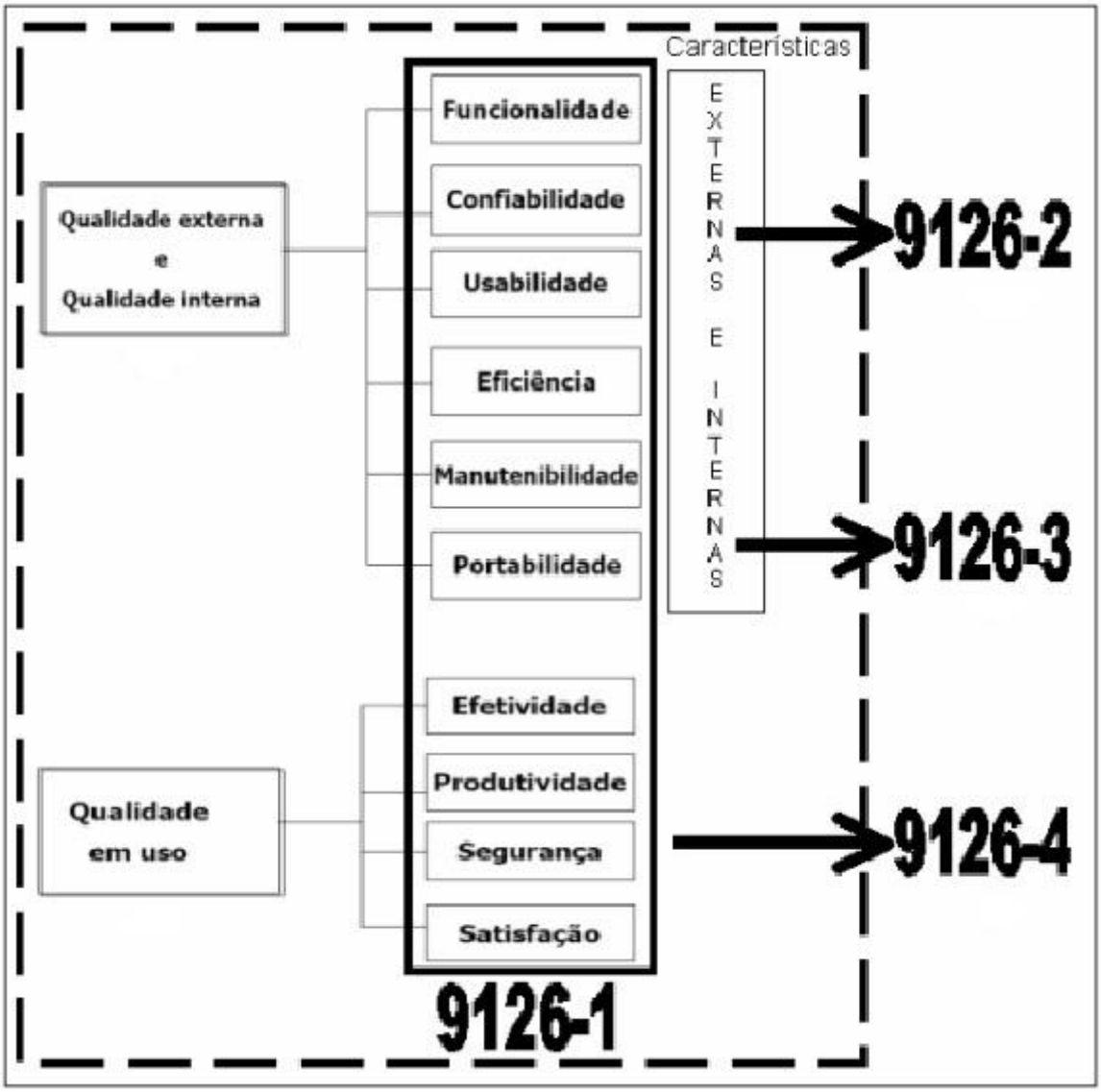

Figura 2 - Representação da norma ISO/IEC 9126
Fonte: Adaptado de (ISO COPYRIGHT OFFICE, 2001)**.**

2.5 TESTE DE *SOFTWARE*

Teste de *software* é a estratégia mais comum de gerenciamento de riscos, é usada para verificar a conformidade dos requisitos com o *software* (PRESSMAN, 2006). Os testes de *software* são utilizados para a verificação das funcionalidades do *software*, podem ser aplicados de diversas formas e abrangência a fim de atingir o maior número possível de combinações de testes, ressaltando que nem todas as falhas podem ser encontradas durante este processo (MOLINARI, 2003).

A partir da visão de ROCHA, MALDONADO e WEBER (2001), o propósito principal das atividades de verificação e validação é garantir que o projeto, a codificação e a documentação estejam em conformidade com os requisitos definidos no escopo do projeto. O esforço de verificação toma menos tempo e não é tão complexo de ser conduzido através do processo de desenvolvimento.

Diversas definições para teste de *software* são descritas por diversos autores, como:

- Para BASTOS, RIOS, *et al.* (2007) testar é verificar se o *software* está realmente executando o que deveria, em conformidade com seus requisitos e se está fazendo o que não deveria.
- DELAMARO, MALDONADO e JINO (2007) definem teste como o processo de avaliar o *software* ou componente de um *software* através de meios manuais ou automatizados para verificar se ele atende os requisitos especificados ou identificar se existe diferença entre os resultados obtidos e os esperados.
- De acordo com BACH (1999), teste de *software* é o processo de utilizar um *software* objetivando o encontro de falhas. Um caso de teste de sucesso é aquele que possui grande possibilidade de que falhas não encontradas anteriormente sejam reveladas.

Segundo DELAMARO, MALDONADO e JINO (2007), o desenvolvimento de *software* não é uma atividade simples, ao contrário pode ser muito complexa dependendo das dimensões e características do *software* a ser desenvolvido. Por este motivo o desenvolvimento está sujeito a inúmeras formas de problemas que poderão resultar em um *software* que não atende aos requisitos esperados.

A fim de que problemas como estes sejam encontrados antes que o *software* seja entregue ao cliente, é de vital importância que ele seja submetido a um processo de revisão, para comprovar que os requisitos estejam em conformidade com os solicitados pelo cliente. A atividade de teste de *software* representa perfeitamente este processo, sendo que segundo PRESSMAN (2007) essa atividade é essencial para garantir a qualidade final do *software*.

Existem duas grandes áreas de divisão na atividade de teste, conhecidas como validação e verificação. Elas propiciam que o desenvolvimento vá de encontro às necessidades do usuário. São constituídas de um conjunto de atividades que são iniciadas em conjunto com a revisão dos requisitos, da análise, do projeto e pela inspeção do código fonte até os testes (KOSCIANSKI e SOARES, 2007).

Segundo BASTOS, RIOS, *et al.* (2007) alguns exemplos de V&V (verificação e validação) são:

- Verificação: revisão de requisitos, revisão de modelos, revisão de códigos e inspeções técnicas em geral;
- Validação: testes de integração, testes de *software*, teste unitários, testes de aceitação e homologação.

As atividades de V&V estão dispostas por todas as etapas do processo de teste, no entanto cada atividade, possui suas características distintas entre elas, que são os testes dinâmicos e estáticos (BASTOS et al., 2007).

A fim de que se tenha entendimento sobre teste de *software* é necessário que o conceito de erro, defeito e falha sejam bem entendidos. Conforme a nomenclatura padrão definida pelo IEEE (*Institute of Electrical and Eletronics Engenieers*) para a engenharia de *software* (RIOS, 2005).

- Erro é algo causado por seres humanos. Um programador que não entendeu perfeitamente os requisitos do *software*;
- Defeito é a consequência de um erro. O programador desenvolveu uma aplicação com requisitos incoerentes aos especificados;
- Falha é a inconsistência entre o resultado apresentado e o resultado requerido pelo usuário. Ao executar a aplicação percebe-se que os resultados requeridos não condizem com os apresentados pela aplicação.

2.5.1 Processo de Teste de Software

De acordo com Molinari (2008) um processo da engenharia de *software* é um grupo de atividades parcialmente ordenado, com o objetivo de entregar um *software* eficaz e que atenda as necessidades do cliente.

Para o SOFTEX (2011), um processo de teste de *software* tem por finalidade obter as seguintes definições:

- Aumentar a qualidade;
- Reduzir o trabalho;
- Aumentar a produtividade;
- Reduzir o tempo de atendimento;
- Aumentar a competitividade;
- Aumentar a precisão das estimativas;
- Acompanhar a satisfação do cliente.

Os testes deixaram de ser tratados como atividades do processo de desenvolvimento e tornaram-se um processo independente. Os testes receberam metodologia própria e são aplicados desde o início do desenvolvimento. Quanto mais cedo os testes forem executados, menor será o custo com correções de defeitos posteriormente encontrados (BASTOS, RIOS, *et al.*, 2007).

As relações entre as fases de desenvolvimento e testes conforme a figura 2 é:

- Desenho lógico e físico, fase em que os testes de verificação garantem que o projeto esteja em conformidade com os requisitos especificados;
- Construção, fase onde os testes unitários são aplicados paralelamente à codificação, podendo assim testar os componentes desenvolvidos;
- Implementação, fase onde o sistema já está codificado e suas funcionalidades estão em funcionamento para que os testes de integração e de sistema sejam executados;
- Entrega, fase onde os testes de aceitação são planejados e implementados com o apoio do usuário.

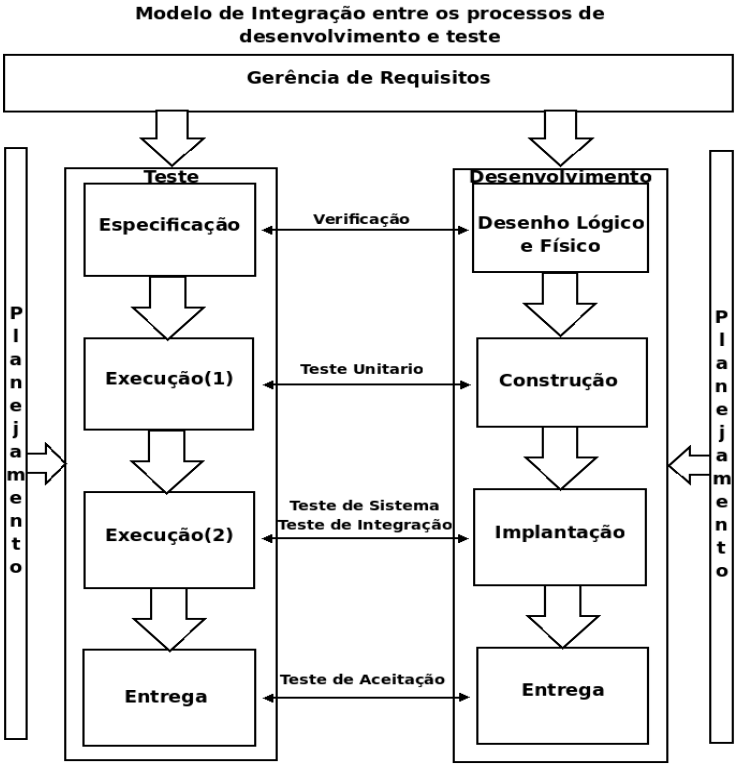

Figura 3 - Integração entre os processos de desenvolvimento e teste.
Fonte: Adaptado de (BASTOS, RIOS, *et al.*, 2007).

2.5.1.1 Ciclo de vida

BASTOS, RIOS, *et al.* (2007), descreve o ciclo de vida do processo de teste constituído de fases e etapas, sendo quatro delas em cascata ou sequência e duas em paralelo, conforme a figura 3.

- Procedimentos iniciais, etapa em que os objetivos do processo de teste são definidos, os requisitos de negócio, as atividades que serão executadas e os recursos;
- Planejamento, etapa de desenvolvimento e revisão das estratégias e do planos de teste;
- Preparação, etapa de adequação e organização do ambiente de testes, da rede, dos equipamentos, do pessoal, do *software* e das ferramentas;
- Especificação, etapa de desenvolviemento dos casos de teste;
- Execução, etapa de execução dos casos de teste;

- Entrega, etapa de término do processo de testes e a entrega do *software*.

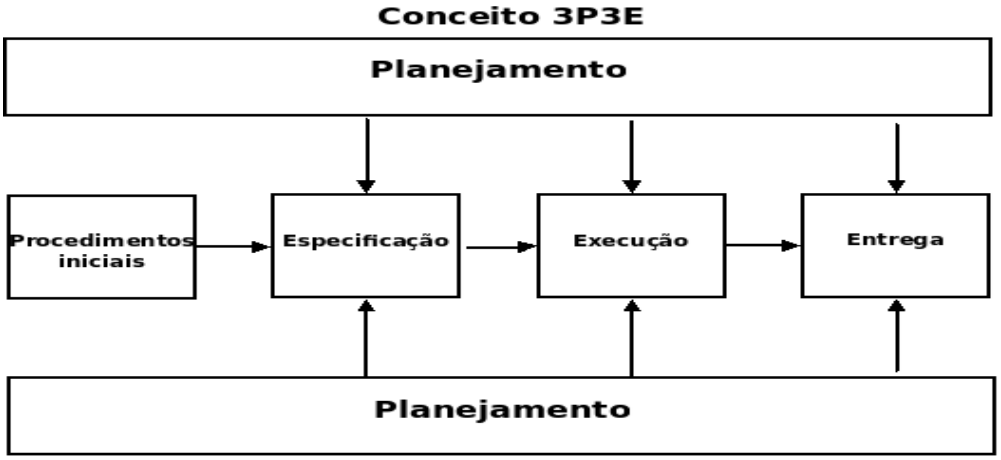

Figura 4 - Conceito 3P3E.
Fonte: Adaptado de (BASTOS, RIOS, *et al.*, 2007)**.**

2.5.1.2 Modelo de Teste

Segundo DELAMARO, MALDONADO e JINO (2007) um modelo de teste é um grupo de técnicas que representam as dimensões do teste, ou seja, em qual etapa do desenvolvimentoum determinado teste deve ser aplicado.

BASTOS, RIOS, *et al.* (2007) apresenta as técnicas abaixo como as principais técnicas de teste de *software*:

- Teste de unidade, técnica que geralmente é executada pelo próprio desenvolvedor com objetivo de testar os componentes do *software*, visando assegurar que a conformidade com as especificações do *software*;
- Teste de integração, técnica que visa garantir que um ou mais componentes do *software* funcionem corretamente;
- Teste de sistema, técnica orientada a validar a exatidão na execução das funcionalidades do *software*;

- Teste de aceitação, ultima tecnica de teste anterior a implantação do *software*, é de responsibilidade do usuário com objetivo de averiguar se o *software* esta em conformidade com os requisitos levantados.

2.5.2 Análise de risco

O teste de *software* está diretamente associado ao risco. Quando um testador não testa algum componente ou funcionalidade do *software* por motivo de custo ou de tempo este está assumindo os riscos. Se este risco vier a se materializar, ou seja, se o componente ou funcionalidade do *software* ao qual não foi testado vier a apresentar falhas, a probabilidade de que as falhas venham a gerar prejuízos irrecuperáveis para o cliente.

O gerenciamento de risco é um processo que tem o objetivo de reduzir a probabilidade de ocorrência de problemas que podem ter impacto negativo no produto final. De acordo com RIOS (2005), risco é um dos mais importantes elementos a ser trabalhado quando um plano de teste de um produto é elaborado. Para que uma análise de risco seja feita é necessário avaliar a probabilidade de ocorrencia do risco e o impacto causado pela falha associada ao risco.

Tais elementos devem ser considerados no momento de se planejar os testes. Para o teste de *software*, risco é definido pela possibilidade de uma falha em uma aplicação vezes o prejuízo que pode ser causado, caso a falha ocorra (RIOS, 2005).

Risco = Probabilidade (possibilidade de falha) X Impacto (prejuízo causado pela falha).

Segundo MOLINARI (2008) os riscos devem ser analisados pelos seguintes motivos:

- Previnir impactos negativos;
- Avaliar consequencias em caso de falhas;
- Definir ações corretas e necessárias antes e após caso o risco ocorra;
- Proteger clientes e usuários;
- Definir prioridades aos riscos;

- Obter critério e direção antes e após a execução dos testes;
- Reconhecer as áreas que necessitam maior atenção;
- Gerenciar melhor os recursos materiais, humanos e o tempo;
- Antecipar o que pode dar errado.

No geral ao se analisar os riscos no desenvolvimento do *software* torna-se possivel obter um melhor conhecimento sobre as áreas e os recusos necessários para que se obtenha bons testes, assim possibilitando que o desperdicio de recursos seja reduzido focando os testes em partes mais criticas do *software*.

2.5.2.1 Riscos

Segundo o *SOFTWARE ENGINEERING INSTITUTE* (2004) risco é a possibilidade de sofrer perdas. Em um projeto de *software* a perda refere-se a reducao da qualidade, custos elevados, atrasos na conclusão ou falhas no produto.

Um risco não possui certeza de que irá ocorrer. Se algo possui 100% de probabilidade de acontecer, ou seja, realmente irá ocorrer não é um risco, mas sim um problema. Um risco pode ser evitado previamente o que não acontece com um problema. BASTOS, RIOS, *et al.* (2007) define risco com um acontecimento no futuro, cuja existencia podera implicar em algum problema no projeto, no produto ou no negócio.

Não existe a certeza de que algo realmente irá ocorrer, no entanto se ocorrer poderá ocasionar graves problemas. De acordo com RIOS (2005) um problema para o teste é impossibilitar que um erro no *software* passe despercebido por que um componente importante não foi testado.

RIOS (2005) cita como exemplo o caso de um avião que apresenta risco contínuo de cair, porém a perca só exitirá se o fato realmente ocorrer. É obrigação das empresas aéreas tomarem todas as providências necessárias para que o risco de problemas que possam causar a queda de uma aeronave seja o menor possível.

No *software* ao realizar o processo de análise de risco, os esforços são focados nas areas onde um falha possa gerar um grande impacto ao cliente, assim como no exemplo do avião o intuito é diminuir consideravelmente a probabilidade de que ocorra uma falha grave (RIOS, 2005).

2.5.2.2 Classificação dos Riscos

Para BASTOS, RIOS, *et al.* (2007) os riscos de maior importância em um projeto de teste de *software* são:

- Riscos técnicos são riscos inerentes a qualidade do *software* que será desenvolvido, como por exemplo falhas no *software*, na implementação, na interface ou a não conformidade dos requisitos com as especificações;
- Riscos de projeto são risco diretamente agregados ao projeto, como por exemplo problemas com recursos, prazos, especificações e cliente;
- Riscos para o negócio são riscos que atingem as partes envolvidas no projeto, podem ser mudanças de prazo, mudanças no orçamento e problemas de negócio.

Este trabalho aborda os riscos técnicos, ou seja, os riscos que podem ocorrer como falha no *software*.

A atividade de análise de risco existe para que os gerentes de projeto possam se prevenir com relação a problemas futuros. Um risco é um sinal de que se pode ter um problema no futuro, caso nenhuma medida de precaução seja tomada. Os riscos podem prejudicar o sucesso de um projeto caso os planos de mitigação e contenção de riscos não tenham sido planejados. Controlar riscos siginifica identifica-los, analisa-los, planeja-los e monitora-los. Outra questão importante é ressaltar que o risco pode ser gerenciado dentro de um projeto, como estabelecido pelo PMI (*Project Management Institute*) ou em um processo conforme definido pelo CMMI ou pelo MPS.BR (RIOS, 2005).

Os testes são um exercício de controle dos riscos, pois suas estimativas devem ser baseadas no risco do negócio e sua estratégia deve ser elaborada juntamente com as áreas envolvidas, no qual o seu investimento reduz custos futuros. Encontrando um erro em fase de desenvolvimento o defeito será solucionado o mais rapido possivel, assim podendo garantir a qualidade.

2.5.3 Risk based testing

James Bach publicou um artigo na revista *American Programmer* em 1995 com o titulo de *The Challenge of Good Enough Software*, ou seja, o desafio do *software* bom o bastante. Neste artigo James Bach apresentou uma nova abordagem de teste de *software* o *risk based testing*, ou seja, o teste baseado em risco. Está abordagem tem como base um grupo de atividades que auxiliam na identificação de riscos associados aos requisitos do *software* (RIOS, 2005).

Segundo MOLINARI (2008) para que a abordagem *risk based testing* realmente ocorra três situações devem ser levadas em conta:

- É inviavél ou impossivel testar todas as funcionalidades de um *software*, neste caso é de suma importancia que o teste seja muito bem planejado para que o plano de teste abranja as areas de maior importancia do *software*;
- Na maioria dos projetos os prazos de desenvolvimento não são cumpridos, nesta situação é necessário priorizar as partes mais importantes do *software*;
- A análise de risco não é uma atividade comum, é uma atividade bem complexa e abstrata para que seja executada é de suma importância que o testador tenha grande conhecimento sobre o produto.

Com os riscos priorizados e com base nas estratégias de acompanhamento e tratamento dos riscos previamente identificados, os casos de teste são gerados.

De acordo com BACH (1999) o *risk based testing*, tem como objetivo examinar:

- A cobertura dos testes;
- O número de testes a serem conduzidos;
- A escolha dos tipos de testes e revisões;
- O uso e o balanceamento entre os testes, as inspeções e as revisões;
- A priorização dos testes, o planejamento e a execução.

BACH (1999) define da seguinte maneira as etapas do *risk based testing*:

- Elaboração de uma lista de riscos e suas prioridades, identificação e análise dos riscos do *software*;

- Realização de testes que explorem cada risco, criação e execução de teste que checam se os riscos identificados realmente existem;
- Gerenciamento de riscos, quando um risco é solucionado, um ou mais riscos podem surgir, assim os testes devem ser ajustados para os riscos com maior prioridade.

Mesmo sendo simples, está definição consegue apresentar a principal ideia da abordagem *risk based testing*. Essa abordagem foca em analisar o *software* e gerar o plano de testes com base nas áreas que possuem maior probabilidade de apresentar erros que possam causar impactos negativos ao *software* (RIOS, 2005).

Segundo MOLINARI (2008) ao avaliar riscos em um projeto de *software* o testador procura localizar as falhas que provavelmente causarão prejuizos para o cliente. Com o levantamento destas informações o testador tem os objetos necessários para a criação dos casos de teste que irão abordar o *software*, mais especificamente em suas partes mais criticas. Desta maneira consegue-se prever se existirá ou não a necessidade de escrever um caso de teste para uma determinada situação.

No geral o *risk based testing* é explicado como o processo de identificar riscos de um *software*, analisar os riscos identificados, priorizá-los, desenvolver casos de teste que mitigam os riscos e gerenciar sua execução (KOSCIANSKI e SOARES, 2007).

2.5.4 Risco Baseado em Heurística

BACH (1999) explica duas metodologias de análise baseado em heurística para a identificação dos riscos, que são o método de fora para dentro e o método de dentro para fora.

2.5.4.1 Método de Dentro para Fora

Neste método é feita uma análise geral e os riscos são pré-definidos com base em experiência de *software* semelhantes. Geralmente são utilizadas fontes como listas gerais de riscos, riscos de qualidade e catálogo de riscos documentados e pré-documentados. Três tipos de listas são sugeridos por BACH (1999):

- Lista de categorias e critérios de qualidade, desenvolvida a partir da norma ISO/IEC 9126:
  - Capacitação: pode desempenhar as funções necessárias?
  - Confiabilidade: vai funcionar corretamente e resistir às falhas em todas as situações exigidas?
  - Usabilidade: o usuário realizará com sucesso as tarefas através do *software*?
  - Desempenho: o *software* terá respostas em tempo aceitável pelo usuário, na realização de suas operações?
  - Instabilidade: o *software* funcionará normalmente nos principais sistemas operacionais disponíveis no mercado?
  - Compatibilidade: o *software* é compatível com os outros componentes?
  - Suportabilidade: o suporte aos usuários do *software* está em conformidade com o orçamento do projeto?
  - Testabilidade: como o *software* pode ser testado efetivamente?
  - Manutebilidade: corrigir, melhorar ou construir novos componentes para o *software* não será mais custoso do que o planejado?
  - Portabilidade: a tecnologia utilizada no desenvolvimento do *software* permitirá reusabilidade em outros *softwares*?
  - Localizabilidade: a internacionalização do *software* não será mais custosa do que o planejado?
- Lista de riscos genéricos tem relação com risos comuns para qualquer *software*. Segundo BASTOS, RIOS, *et al.* (2007) as principais funções onde os riscos de se encontrar um defeito são maiores, são:
  - Funções com grande complexidade;

- Funções completamente novas;
- Funções que são alteradas com frequencia;
- Funções onde uma nova ferramenta é usada pela primeira vez no desenvolvimento;
- Funções que foram transferidas de um programador para outro durante o desenvolvimento;
- Funções desenvolvidas sob pressão;
- Funções que sofrem muitas customizações;
- Funções onde muitas falhas foram encontradas;
- Funções que possuem muitas interfaces.

- Lista de catálogo de risco é definida como um rascunho de riscos pertencentes a um determinado domínio. Uma versão resumida do catálogo é apresentada por (RIOS, 2005):
  - Escolher qual função ou componente pretende analisar;
  - Determinar a escala de prioridade;
  - Coletar informações sobre os itens que serão analisados.

Segundo BACH (1999) para uma efetiva utilização desse método deve-se analisar em cada lista cada área de risco e definir sua prioridade, no caso de algum risco não estar listado deve-se refazer a lista, reagrupar os riscos e conferir a distribuição dos riscos para com isto poder utilizar os artefatos de mitigação.

2.5.4.2 Lista de Observação de Risco

De acordo com BACH (1999) é a forma mais fácil de organizar os riscos, é uma lista que o testador revisa constantemente e questiona-se sobre o que os testes mostraram sobre os riscos.

2.5.4.3 Tabela Risco X Tarefa

Para BACH (1999) a matriz risco x tarefa, é uma matriz que contém os riscos e as tarefas que serão utilizadas para reduzir a frequência ou o impacto dos riscos, é de grande utilidade a partir do momento em que os riscos estejam organizados por prioridade, assim ajudando na decisão de quais tarefas serão realizadas.

Tabela 1 - Tabela Risco X Tarefa.

Qualidade / Fase	Confiabilidade	Correção
Análise	O controle de integridade dos dados foi estabelecido?	A especificação funcional foi feita?
Projeto	O controle de integridade dos dados foi criado?	O projeto está em conformidade com o que foi especificado?
Codificação	O controle de integridade dos dados foi implementado?	O programa está em conformidade com o que foi projetado?
Testes	Testes que verificam a integridade foram criados e aplicados?	Testes que verificam as funcionalidades foram criados e aplicados?

Fonte: Adaptado (BASTOS, RIOS, *et al.*, 2007).

2.5.4.4 Matriz Componente X Risco

De acordo com BACH (1999) a matriz componente x risco, possui três colunas. A coluna esquerda apresenta os componentes do *software*, a do centro o grau de risco e a direita as heurísticas que expõem o componente ao risco.

Tabela 2 - Matriz Componente X Risco.

Componente	Risco	Heurística
Impressão	Normal	Distribuído, popular.
Geração de relatórios	Alto	Novo, estratégico, terceirizado e crítico.
Instalação	Baixo	Popular, usabilidade e modificado.
Biblioteca de imagens	Baixo	Complexo.

Fonte: Adaptado (BASTOS et al., 2007).

A grande vantagem dessa abordagem é que conforme o projeto se desenvolve a atenção é distribuída para diferentes componentes de acordo com os níveis de risco associados ao componente (BACH, 1999).

2.5.4.5 Método de Fora para Dentro

Criada por BACH (1999) esta abordagem analisa um *software* utilizando as seguintes questões:

- Vulnerabilidade: Quais são as possíveis falhas ou fragilidades que existem nesse componente?
- Ameaça: Quais situações ou entradas podem ocorrer e que podem encontrar a vulnerabilidade e causar uma falha nesse componente?
- Vitimas: Quem ou o quê poderá ser atingido poderá ser atingido por uma provável falha e quão prejudicial isso seria?

Para melhorar explicação desta abordagem um estudo de caso será realizado em um *software* real.

3 MATERIAIS E MÉTODOS

Para melhor exemplificar o conceito teórico apresentado neste trabalho, este capítulo visa demostrar de forma prática a utilização dos conceitos.

3.1 O AMBIENTE

Como ambiente para aplicação dos testes o ambiente virtual de aprendizagem Moodle na versão 1.9.13+, foi utilizado. A versão utilizada foi customizada assim intitulada Ambiente WRA, e está sendo utilizada em um curso de Atualização em Energias do Biogás, oferecido pelo Projeto Web Rádio Água do Parque Tecnológico Itaipu.

O ambiente WRA tem como objetivo fornecer um espaço virtual de ensino a distância, que possua estabilidade, facilidade de iteração e visual agradável para que os participantes possam aproveitar melhor o curso oferecido.

3.2 ESTUDO DE CASO

Como o método de fora para dentro tem foco nas ameaças, vulnerabilidades e vítimas, não existe a necessidade de se utilizar itens internos do *software*, como código fonte, banco de dados ou arquitetura de desenvolvimento utilizada.

Será testado o fluxo desde o acesso ao sistema até a visualização do material didático, midiateca e seus anexos.

3.2.1 Requisitos

Para melhor entendimento e elucidação do software, apresentam-se os requisitos funcionais e não funcionais essenciais para ambiente, levantados como necessidades do projeto.

Tabela 3 - Documento de Requisitos

Sigla	Descrição	Visibilidade
RF01	Software com interface *web.*	Evidente
RF02	O aluno deve ter acesso somente à turma em que está inscrito, caso acesse outra turma uma mensagem deve ser exibida.	Evidente
RNF01	O layout deve manter e seguir o mesmo padrão em todas as páginas.	Importante
RNF02	Página Inicial deve apresentar a lista de cursos disponíveis, formulários de contato, espaço para o *login*, link para o tutorial, bloco para validação do certificado e bloco com a hora do servidor.	Evidente
RNF03	Depois de ser autenticado, na página inicial devem aparecer os dados do usuário, nome completo, foto, opção de atualizar o perfil e sair.	Importante
RNF04	Página da turma deve apresentar o cronograma geral do curso, *menu* com os itens do curso, bloco de usuários online, bloco de e-mail e bloco administração.	Evidente
RNF05	O *menu* da página da turma deve ter os links para fórum de avisos, material didático, atividades, fórum, chat, midiateca, lista de participantes da turma, avaliação do curso.	Evidente
RNF06	Página material didático deve listar os módulos disponíveis para o aluno.	Evidente
RNF07	Página de introdução do módulo deve apresentar o titulo do modulo, o nome do professor e o menu dos capítulos do módulo.	Importante
RNF08	Apresentação do conteúdo do módulo, deve apresentar um menu com os capítulos do módulo e o conteúdo deve ser apresentado nesta página.	Evidente
RNF09	Página midiateca apresenta uma descrição do que é a midiateca e links para os textos, vídeos e material didático complementar.	Evidente
RNF10	Página textos deve apresentar a lista dos arquivos disponíveis e a sinopse de cada arquivo.	Evidente
RNF11	Página vídeos deve apresentar a lista de vídeos disponíveis, o vídeo, opção para download do vídeo e uma introdução.	Evidente
RNF12	Página material didático complementar deve apresenta a lista dos arquivos disponíveis.	Evidente

3.2.2 Requisitos Implementados

A seguir é apresentado telas do ambiente após a implementação dos requisitos. A figura 5 apresenta a tela do sistema relacionada ao Requisito Funcional RF02.

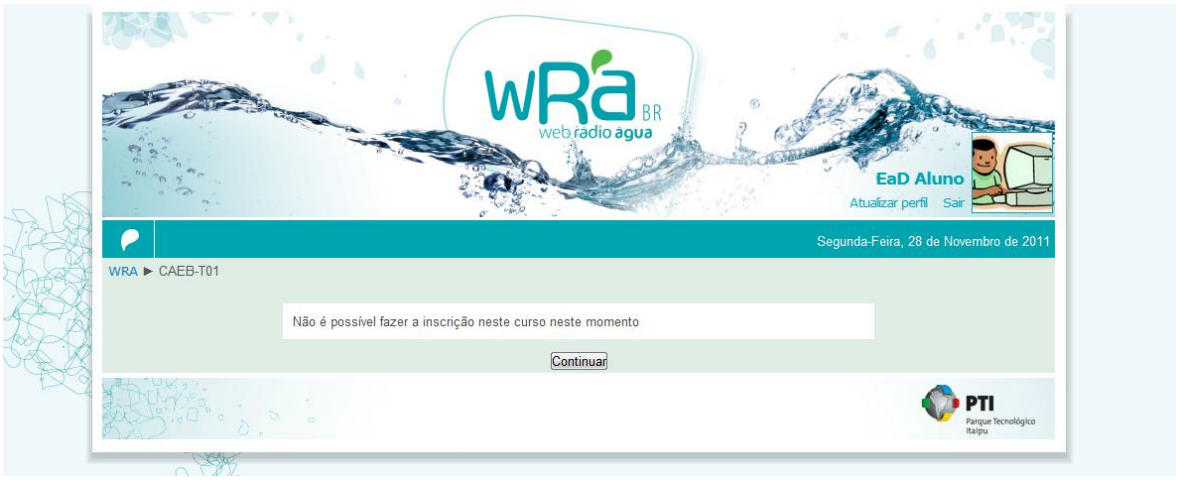

Figura 5 - RF02 - Tentativa de acesso à turma em que não está inscrito.

A figura 6 mostra um *screenshot* da tela inicial do Moodle formatado com o tema desenvolvido para o evento, onde foi efetuado o estudo de caso. A tela está relacionada ao RNF02.

Figura 6 – RNF02 - Tela Inicial do Ambiente WRA.

A figura 7 apresenta um *screenshot* da tela inicial do sistema formatado para o evento, mas desta vez com o usuário EaD aluno autenticado. A tela está relacionada ao RNF03.

Figura 7 – RNF03 - Tela inicial após login efetuado.

Conforme o RNF04, a figura 8 mostra um *screenshot* da página de atividades, onde apresenta o cronograma do evento relacionado.

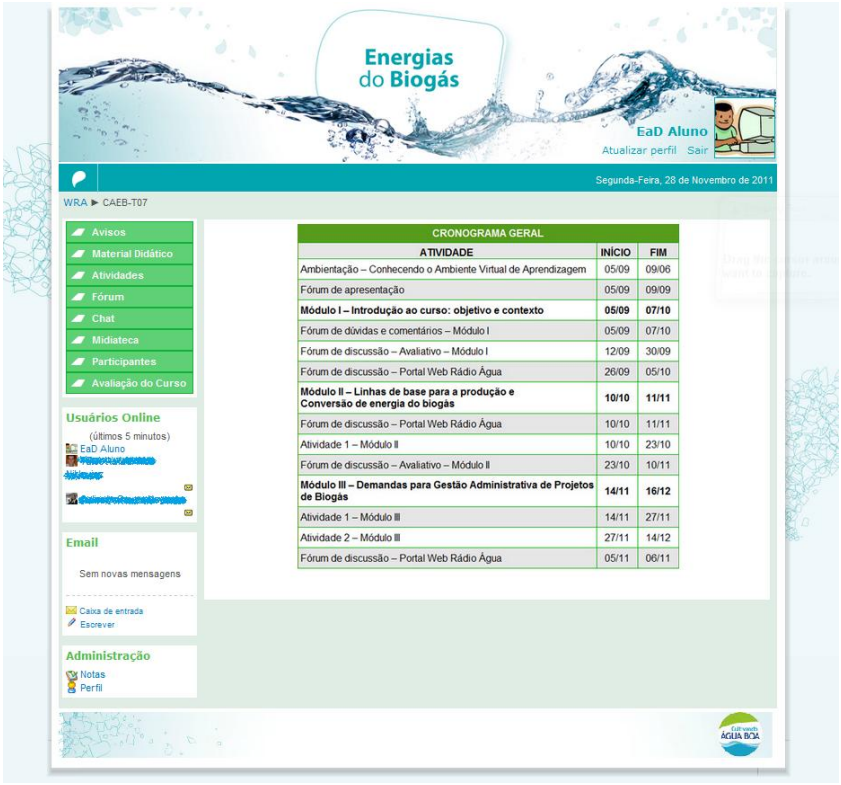

Figura 8 – RNF04 - Página da turma.

A figura 9 destaca o *menu* da página de atividades da turma, e está relacionado com o RNF05 e também com o RNF04.

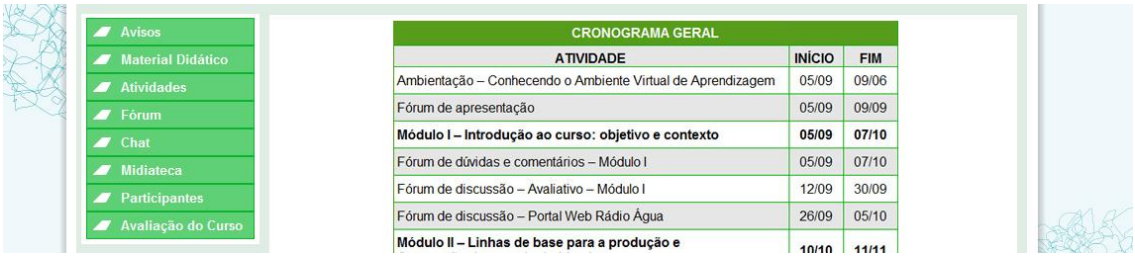

Figura 9 – RNF05 - Menu página da turma.

A figura 10 apresenta um *screenshot* da parte de materiais didáticos, que está relacionado RNF06.

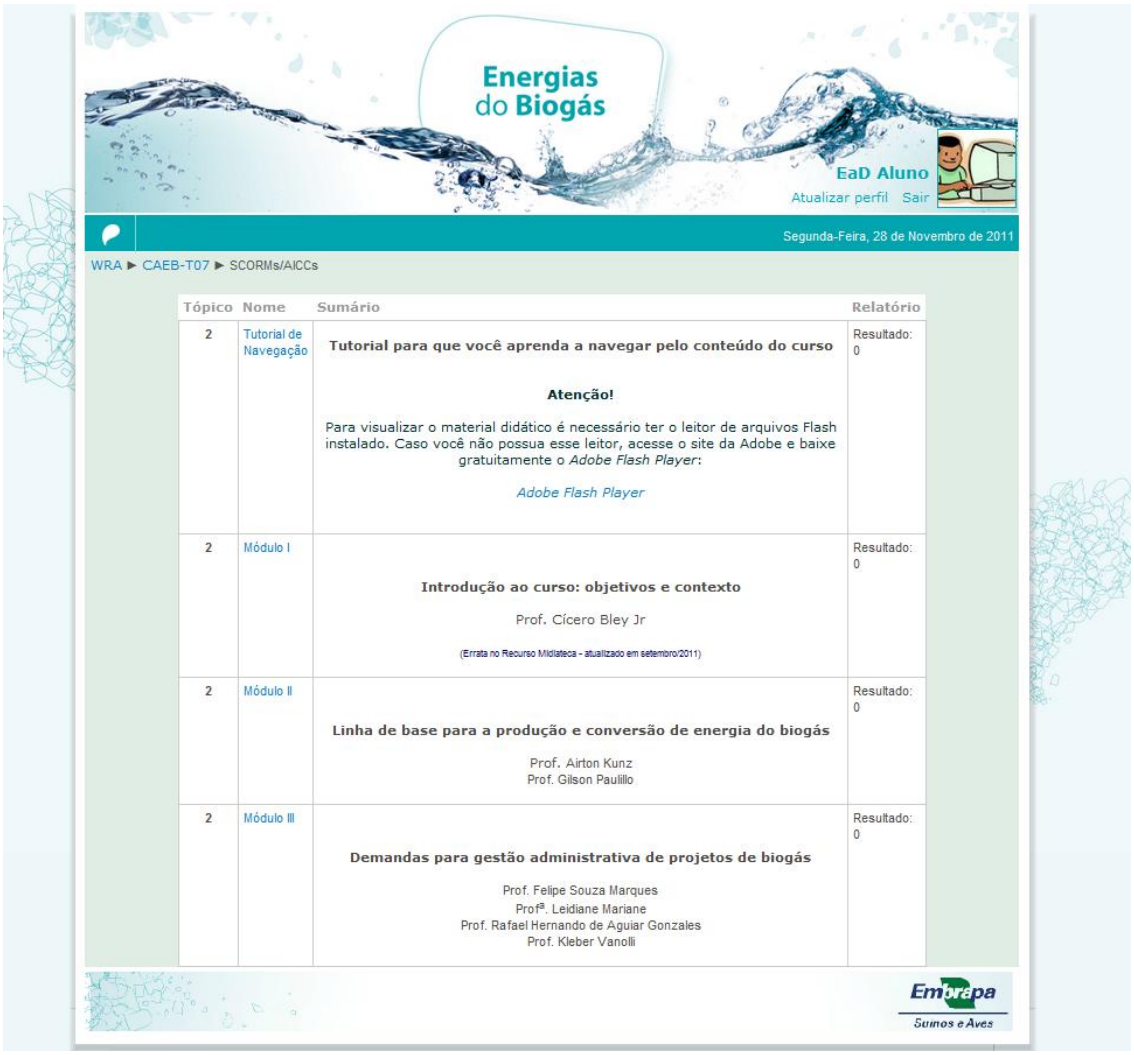

Figura 10 – RNF06 - Página dos materiais didáticos.

A página de introdução ao módulo pode ser vista na figura 11 e está relacionada com o RNF07.

Figura 11 – RNF07 - Página de introdução do módulo.

A figura 12 destaca um vídeo da página de conteúdos, e está relacionado com o RNF08.

Figura 12 – RNF08 - Página de apresentação do conteúdo do módulo.

A figura 13 mostra o conteúdo da página da midiateca, onde estão arquivados (para acesso dos alunos) artigos na forma de textos, vídeos diversos e outros materiais didáticos. A midiateca está relacionada com o RNF09.

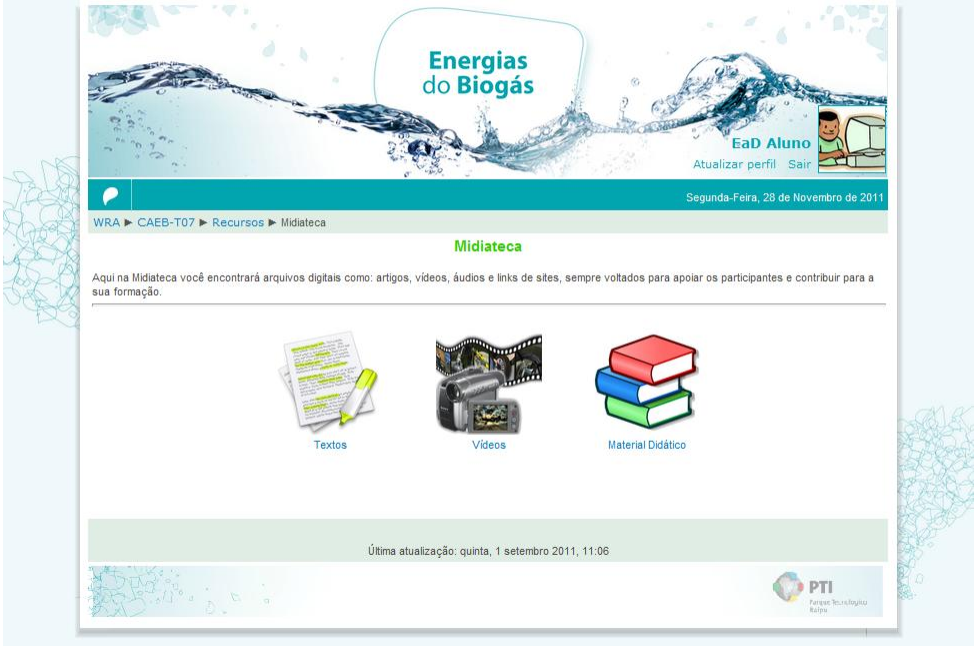

Figura 13 – RNF09 - Página midiateca.

A figura 14 ilustra os conceitos necessários apontados no RNF10, e apresenta uma lista de arquivos e sinopses para apontar os textos armazenados.

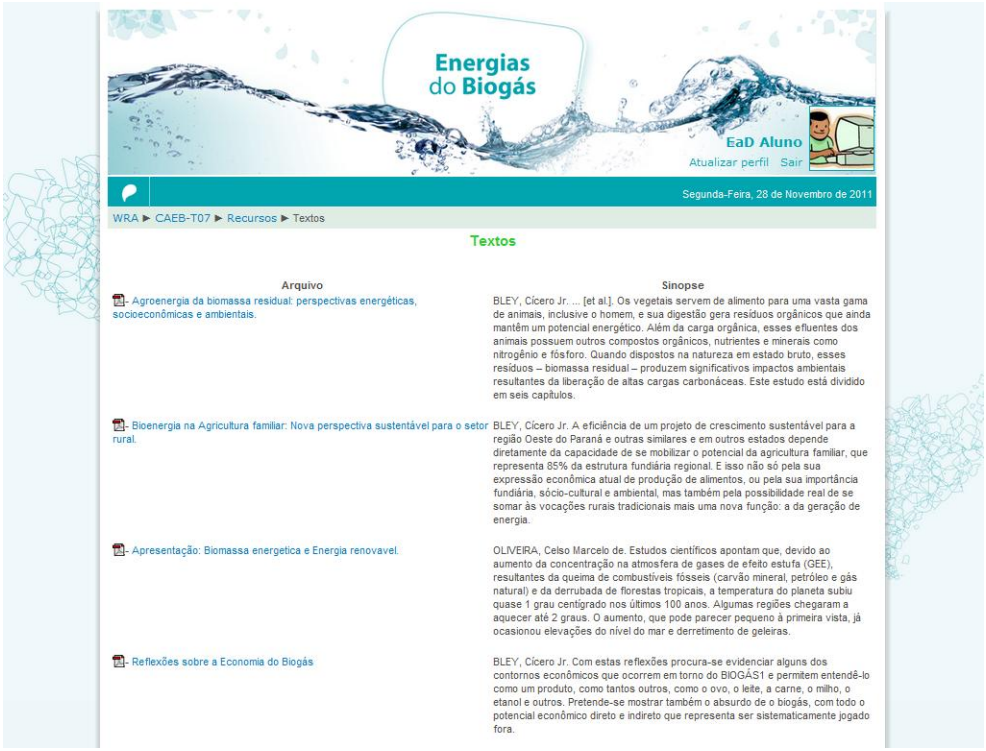

Figura 14 – RNF10 - Página textos.

Na figura 15 demonstra-se um vídeo (no exemplo a Atualização em Energias do Biogás) e a disposição dos conteúdos necessários aos requisitos apontados no RNF011.

Figura 15 – RNF11 - Página vídeos.

A figura 16 apresenta a página de material didático complementar, onde se pode observar que há uma lista de arquivos com dados relevantes para o aluno poder situar-se em cada um deles. Tal exemplo está relacionado com o RNF12.

Figura 16 – RNF12 - Página material didático complementar.

Com os requisitos do sistema especificados e implementados, o próximo passo é identificar os riscos associados, as ameaças, vulnerabilidades e vitimas, conforme os critérios da abordagem.

4 RESULTADOS E DISCUSSÕES

O presente capítulo apresenta o plano de testes e os resultados de sua aplicação no ambiente virtual de aprendizagem Moodle, bem como discussões sobre o procedimento efetuado.

4.1 LISTA DE VULNERABILIDADES

A lista que possui as prováveis vulnerabilidades que o ambiente WRA pode apresentar está disposta Tabela 3.

Tabela 4 - Lista de vulnerabilidades.

Item	Vulnerabilidades
1	O ambiente pode não funcionar corretamente em todos os navegadores disponíveis no mercado.
2	O layout do ambiente pode não estar nos padrões de desenvolvimento podendo não ser apresentado corretamente ou apresentar deformidades.
3	Determinados itens podem não ser apresentados.
4	O formulário de *login* pode não autenticar corretamente o aluno.
5	As turmas podem não validar a tentativa de acesso do aluno a uma turma em que ele não esteja inscrito.
6	Os links podem estar quebrados, desatualizados ou direcionando a itens que não são os esperados.
7	Os arquivos disponíveis para download podem estar corrompidos.
8	Os vídeos podem não ser apresentados corretamente.
9	O layout do ambiente pode não ser intuitivo e agradável.
10	O conteúdo dos módulos pode não ser apresentado corretamente.

Com a lista de vulnerabilidades em mãos a próxima etapa é listar as ameaças e associar as vulnerabilidades.

4.2 LISTA DE AMEAÇAS

As ameaças podem estar relacionadas às vulnerabilidades, pois dependendo da ameaça ela pode acionar um ou mais itens da lista de vulnerabilidades. Como apresentado na tabela 4.

Tabela 5 - Lista de ameaças.

Item	Ameaças	Vulnerabilidades
1	O aluno pode estar utilizando um navegador com versão inferior ou superior a utilizada no desenvolvimento, utilizando um navegador que não está entre os mais utilizados do mercado, utilizando sistema operacional diferente do utilizado no desenvolvimento.	1, 2, 3, 6, 8, 10
2	O aluno pode tentar utilizar dados incorretos.	4
3	O aluno pode não ter instrução suficiente para utilizar o ambiente.	9
4	O aluno pode tentar acessar uma turma que não seja a que ele está inscrito.	4, 5
5	O ambiente pode ser acessado em diferentes resoluções de vídeo e tamanho de monitores.	2, 3
6	A ação de um botão ou link pode não ser a mesma em diferentes navegadores ou sistemas operacionais.	6
7	O aluno pode não ter os plug-ins para vídeos necessários para a visualização.	8
8	O aluno pode estar com a versão do flash player desatualizado.	10
9	O aluno não consegue se encontrar e navegar no ambiente.	9
10	O aluno não consegue abrir os arquivos baixados do ambiente.	7, 8

A lista de ameaças está feita e relacionada às vulnerabilidades, o próximo passo é listar as vitimas que poderão ser afetadas caso haja alguma falha.

4.3 LISTA DE VITÍMAS

Descreve as vitimas e procura detalhar o quão prejudicial uma falha pode representar a esta vitima. Como apresentado na tabela 5.

Tabela 6 - Lista de vitimas.

Item	Vitima
1	O aluno, pois pode ficar prejudicado caso algo saia errado, pode perder o prazo de uma atividade, pode diminuir seu rendimento no decorrer do curso.
2	O professor pode não obter o resultado esperado de participação de sua turma.
3	O Projeto Web Rádio Água, pois pode perder novos alunos devido ao fato de que alunos que tiveram problemas irão criticar o curso.

Tendo em mãos as lista, pode-se então partir para a lista de testes que irão explorar os itens listados. A tabela 6 apresenta a lista de testes necessários para mitigar os riscos baseados nas vulnerabilidades, ameaças e vitimas listado.

4.4 LISTA DE TESTES PARA MITIGAÇÃO DOS RISCOS

Tabela 7 - Lista de testes para mitigação dos riscos.

Item	Teste
1	Testar o ambiente em diferentes navegadores e versões.
2	Testar o ambiente em diferentes sistemas operacionais.
3	Testar o sistema em diferentes monitores e resoluções de vídeo.
4	Listar e conferir todos os campos estão visíveis e legíveis conforme os requisitos e a tarefa a ser executada.
5	Testar o funcionamento dos botões e links e se estão direcionando para os itens corretos.
6	Testar com usuários leigos a usabilidade do ambiente.
7	Testar a integridade dos arquivos disponíveis para download.
8	Testar a visualização dos vídeos.
9	Testar o conteúdo dos módulos em versões diferentes do flash player.

Agora o testador possui subsídios suficientes para executar os testes voltados a mitigar os riscos, tendo como base estas informações e quais testes executar para reduzir os riscos.

4.5 VANTAGENS E RESTRIÇÕES

As principais vantagens da utilização desta técnica estão relacionadas às questões de prazos, custo e esforço. BACH (1999) recomenda a utilização da abordagem quando outras abordagens demandam de custo, tempo e recurso maiores do que o disponível. Outra vantagem apresentada por BACH (1999) é que como o foco da abordagem está direcionada as áreas de maior risco do *software*, ela justifica os esforços da atividade de teste, pois vai de encontro com as falhas mais prováveis.

Levando em consideração que a abordagem tem foco na redução dos esforços utilizando-se dos riscos levantados, BACH (1999) especificou que a abordagem não deve ser utilizada em aplicações criticas, onde uma falha pode provocar perdas financeiras de alto custo ou em casos onde envolvem risco a vida, nestes casos abordagens mais formais devem ser aplicadas.

5 CONSIDERAÇÕES FINAIS

Este trabalho apresentou uma visão geral dos processos de engenharia, qualidade e teste de *software*, no entanto teve foco na área de teste baseado em risco, apresentando a abordagem *risk based testing*. Técnica que foca seus esforços para testar as áreas com maior risco de falhas, assim tornando a atividade de teste mais ágil e menos custosa, diferente de outras abordagens que visam cobrir todas as funcionalidades do *software*.

Mesmo que a abordagem *risk based testing* pareça simples, os gerentes de projeto e testadores ainda enfrentam dificuldade em executá-la, principalmente porque a análise de risco não é uma atividade simples, mas sim uma atividade complexa que demando de conhecimento da aplicação e do negocio. Não existem ferramentas comerciais especificas para o *risk based testing* oque provavelmente tem dificultado sua aplicação e disseminação, respaldando a afirmação de SOUZA (2007).

No entanto é preciso ressaltar que a abordagem possui limitações, principalmente com relação a sistemas críticos, onde falhas podem ter custo muito alto e consequências graves.

5.1 CONCLUSÕES

É possível afirmar que a constante busca pelo desenvolvimento de *software* de qualidade e com baixo custo tem despertado e motivado inúmeras empresas e desenvolvedores a efetuarem pesquisas na área de engenharia de *software* e grande parte delas com foco nos riscos associados ao *software* e também que a abordagem no cumprimento do que ela se propõe atende seus objetivos e cumpre com suas especificações.

A partir da implementação do estudo de caso foi possível atestar a real validade da abordagem e sua simplicidade de aplicação, pois diferente de outras abordagens de teste de *software* que necessitam um esforço maior para sua

execução, esta abordagem demonstrou agilidade na execução, o que possibilitou sua execução de forma ágil e eficaz no ambiente testado.

5.2 TRABALHOS FUTUROS

Como perspectiva de trabalhos futuros pode-se relacionar:

- Um estudo que possa apresentar dados sobre os benefícios da utilização da abordagem apresentando informações sobre o percentual de esforço reduzido e a confiabilidade alcançada.
- Aplicar a metodologia de dentro para fora em uma aplicação real como estudo de caso.
- Desenvolver um estudo visando criar listas padrões de riscos conhecidos para diferentes segmentos.
- Implementar um sistema automatizado de identificação de riscos baseado em requisitos predefinidos.
- Implementar um sistema que automatize os casos de testes baseados nos riscos identificados.

REFERÊNCIAS

BACH, J. **Risk-Based Testing: How to conduct heuristic risk analysis.**, 1999. Disponivel em: <http://www.satisfice.com/articles/hrbt.pdf>. Acesso em: 05 ago. 2011.

BARTIÉ, A. **Garantia da Qualidade de Software:** Adquirindo Maturidade Organizacional. 1ª Edição. ed. Rio de Janeiro: Campus, v. I, 2002.

BASTOS, A. et al. **Base de Conhecimento em Teste de Software**. 1ª Edição. ed. São Paulo: Martins Editora, v. I, 2007.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao Teste de Software**. 2ª Edição. ed. Rio de Janeiro: Elsevier, 2007.

ISO COPYRIGHT OFFICE **ISO/IEC 9126-1**., 2001. Disponivel em: <http://webstore.iec.ch/preview/info_isoiec9126-1%7Bed1.0%7Den.pdf>. Acesso em: 10 out. 2011.

KOSCIANSKI, A.; SOARES, M. D. S. **Qualidade de Software**. 2ª Edição. ed. São Paulo: [s.n.], 2007.

MOLINARI, L. **Testes de Software:** Produzindo Sistemas melhores e mais confiáveis. 1ª Edição. ed. São Paulo: Érica, 2003.

MOLINARI, L. **Testes Funcionais de Software**. 1ª Edição. ed. Santa Catarina: Visual Books, 2008.

NAKAMURA, R. **Moodle:** Como criar um curso usando a plataforma de Ensino a Distância. 1ª Edição. ed. São Paulo: Farol Forte, 2008.

PAULA FILHO, W. D. P. **Engenharia de Software:** Fundamentos, Métodos e Padrões. 2ª Edição. ed. Rio de Janeiro: LTC, 2003.

PRESSMAN, R. S. **Ingenieria del Software:** Un Enfoque Pratico. 5ª Edição. ed. España: McGraw-Hill, 2002.

PRESSMAN, R. S. **Engenharia de Software**. 6ª Edição. ed. São Paulo: McGraw-Hill, 2006.

RIOS, E. **Análise de Risco em Projetos de Teste de Software.** 1ª Edição. ed. Rio de Janeiro: Alta Books, 2005.

ROCHA, A. R. C. D.; MALDONADO, J. C.; WEBER, K. C. **Qualidade de Software:** Teoria e Prática. 1ª Edição. ed. São Paulo: Prenttice Hall, v. I, 2001.

SOFTEX **MPS.BR - Guia Geral.**, 2011. Disponivel em: <http://www.softex.br/mpsbr/_guias/guias/MPS.BR_Guia_Geral_2011.pdf>. Acesso em: 15 out. 2011.

SOFTWARE ENGINEERING INSTITUTE **Risk Based Diagnostics**., 2004. Disponivel em: <http://www.sei.cmu.edu/reports/04tn013.pdf>. Acesso em: 20 out. 2011.

SOMMERVILLE, I. **Engenharia de Software**. 8ª Edição. ed. São Paulo: Pearson, 2007.

TONSIG, S. L. **Engenharia de Software:** Análise e Projeto de Sistemas. 2ª Edição. ed. Rio de Janeiro: Ciência Moderna, 2008.