CRISTIANA YUKIE MASUDA

PROCESSO DE AUTOMAÇÃO DE TESTES DE *SOFTWARE* COM FERRAMENTAS *OPEN SOURCE*: UM ESTUDO DE CASO COM INTEGRAÇÃO CONTÍNUA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação da Universidade do Sul de Santa Catarina, como requisito parcial à obtenção do título de bacharel em Sistemas de Informação.

Orientadora: Profa. Maria Inês Castiñeira, Dra.

Palhoça

2009

CRISTIANA YUKIE MASUDA

PROCESSO DE AUTOMAÇÃO DE TESTES DE *SOFTWARE* COM FERRAMENTAS *OPEN SOURCE*: UM ESTUDO DE CASO COM INTEGRAÇÃO CONTÍNUA

Este Trabalho de Conclusão de Curso foi julgado adequado à obtenção do título de Bacharel em Sistemas de Informação aprovado em sua forma final pelo Curso de Graduação em Sistemas de Informação da Universidade do Sul de Santa Catarina.

Palhoça, (dia) de (mês) de (ano da defesa).

_____

Professora e orientadora Maria Inês Castiñeira, Dr.
Universidade do Sul de Santa Catarina

_____

Lisiane Reis Gesychter, Me.
Nexxera Tecnologia & Serviços

_____

Lúcio Lopes Diniz, Engenheiro de telecomunicações.
Governança Brasil

Dedico esta monografia a todos que de uma forma ou outra me apoiaram na sua confecção, principalmente aos meus avós, que estão sempre presentes, principalmente nos momentos mais difíceis da minha vida.

AGRADECIMENTOS

Principalmente a minha orientadora Maria Inês Castiñeira pela força, auxílio e principalmente pela amizade, simpatia e presteza.

Agradeço ao Marcos Antonio Silva, diretor de TI que possibilitou e incentivou para o desenvolvimento desta monografia, abrindo as portas da empresa para pesquisa.

Aos amigos do trabalho que sempre com alegria ajudavam, discutiam e trocavam informações sobre o assunto da monografia, e tentavam me descontrair nos dias mais complicados e desanimados que encontramos no decorrer da confecção do tcc.

Ao professor Ivo com suas incansáveis, difíceis e pacientes revisões de português.

Aos amigos, Edinilton e minha nova amiga, Leandra e amigo Felipe que conheci aqui na Unisul, que me acompanharam e apoiaram nestes anos de faculdade, com sorrisos, companheirismo e ouvindo as minhas reclamações.

Aos novos amigos e colegas que conheci, nesta procura de informações sobre o tema discutido na monografia, em fóruns, e-mails entre outros.

Carinhosamente ao meu namorido pela compreensão, incentivo e companheiro de noites e finais de semanas de estudo e pesquisa.

Agradeço a todos os professores que estiveram presentes estes anos e que sabiamente tentaram passar o seu conhecimento.

Aos convidados da banca que aceitaram o convite, e dispuseram de seu tempo para contribuir com sua experiência e conhecimento.

Finalmente a Deus e a minha mãe por me dar a oportunidade de estar aqui e acreditar em mim. Aos meus avós eternamente presentes no meu coração e que me passaram sua energia positiva para a conclusão deste trabalho.

Muito obrigada!

“Efeito Titanic: não existe tecnologia de testes que detecte as falhas de uma forma linear, porque, muitas das vezes, um defeito não visualizado pode ser a ponta de um iceberg. Visualizar o iceberg a tempo é fundamental” (Leonardo Molinari).

RESUMO

Esta monografia apresenta um trabalho de pesquisa sobre metodologias, processos e ferramentas de auxílio para elaboração de um processo de testes automatizado, com integração contínua. Seguindo os padrões e normas de testes, apoiadas por ferramentas *open source* e usando métricas, para minimizar os problemas que ocorrem com as aplicações em produção, esta proposta tem como objetivo gerar uma padronização nas atividades do processo de teste, garantindo a qualidade do *software*. Uma empresa de médio porte, desenvolvedora de *software*, prestou-se como espaço de pesquisa. A escolha de um produto da organização tornou-se um estudo de caso apropriado para a criação de um projeto de testes associado ao desenvolvimento contínuo do produto, sendo o processo a base para futuras confecções de projeto de testes e de desenvolvimento. Um dos resultados deste trabalho foi a definição desse processo de testes, assim como a materialização e planejamento de diversos artefatos, documentos de testes e relatórios. Para apoiar esse processo foi criado um ambiente automatizado, totalmente composto por ferramentas *open source*, cada uma delas aportando uma diferente funcionalidade: Mantis (gestão de defeitos), Cobertura (cobertura de código), ferramenta JUnit com apoio da DBUnit (identificação do *bug*), CVS (controle de versões) Hudson (integração contínua). A integração dos testes, identificação de bugs, requisitos do produto trouxe ao *software* maior controle de qualidade e rastreamento de todos os documentos necessários para avaliar de forma mais rápida as regras, defeitos e histórico associados às funcionalidades. Esses resultados, expressos através de números, são indicadores importantes que podem ser utilizados pela organização para a avaliação e tomada de decisão estratégica, de forma a minimizar as ameaças e gerar oportunidades que permitam tornar a empresa cada vez mais competitiva no mercado.

Palavras-chave: Qualidade de *software*. Processo de testes. Ferramentas *open source*. Processo de automação de testes. Integração contínua.

ABSTRACT, RESUMÉ OU RESUMEN

This monograph is a work of research over methodologies, process and tools that helps to elaborate a process of test automation with continuous integration. Following tests standars and rules, supported by open source tools and using metrics to minimize problems that might occur in production applications, this proposal has as objective generate a standard in test process activities, granting software quality. A medium size software developer company is the subject of this research. The fact of choosing one product among all others within the organization became an appropriate case study to create a test project associated with continuos product development, so the process could be the base for future test projects and developments. One of the results of this work was the definition of this test process, not to mention plannig and materialization of several artifacts, test documents and reports. In order to support this process an automated environment, totally based on open source tools, was created. Each open source tool had its purpose: Mantis (defect management), Cobertura (code coverage), JUnit with DBUnit (bug identification), CVS (version control) and Hudson (continuous integration). Tests integration, bugs identification and product requirements have brought to software much more quality control, nevertheless the ability to kept track over all documents necessary to evaluate faster the rules, defects and history related to features. These results, expressed through numbers, are important indicators that can be used by the organization so it can evaluate and make the right strategic choise in order to minimize threats, and giving it the opportunity to make it more competitive in the market.

Keywords, Mots-clés ou palavras chave: software quality. Process of test. Open source tools. Automatic. Process of test automation. Continuous integration

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTAS DE SIGLAS

ABNT – Associação Brasileira de Normas Técnicas

ALATS – Associação Latino-Americana de Teste de *Software*

CELEPAR – Companhia de Informática do Paraná

CENPRA – Centro de Tecnologia da Informação Renato Archer

CESAR – Centro de Estudos e Sistemas Avançados de Recife

COPPE/UFRJ – Conselho de Ensino Pós Graduados e Pesquisa/Universidade Federal do Rio de Janeiro

CMMI – *Capacity Maturity Model Integrated* – Integração de Modelos de Maturidade da Capacidade

CVS – *Concurrent Version System*

ERD – Eficácia de Remoção de Defeitos

EDD – Eficácia de Detecção de Defeitos

HTML – Linguagem de Marcação de Hipertexto

IEEE – *Institute of Electrical and Electronic Engineers*

IEC – *Commission Electrotechnique Internationale*

ISO – Organização Internacional de Padronização

MA-MPS – Método de Avaliação para Melhoria de Processo de *Software*

MN-MPS – Modelo de Negócio para Melhoria de Processo de *Software*

MPS.BR – Melhoria de Processos de *Software* Brasileiro

MPT – Melhoria de Processo de Teste

MR-MPS – Modelo de Referência para Melhoria de Processo de Teste

NBR – Norma Brasileira Regulamentadora

OO – Orientado a Objeto

PMBOK – *Project Management Book of Knowledge*

PMI – *Project Management Institute*

RA – Roteiro de atualização

RTO – Roteiro técnico de ocorrência

SGDB – Sistema de gerenciamento de banco de dados

SOFTEX – Sociedade para Promoção da Excelência do *Software* Brasileiro

SQL – *Structured Query Language*

SUCESU – Sociedade de Usuários de Computadores e Equipamentos Subsidiários

TCC – Trabalho de Conclusão de Curso

TI – Tecnologia da Informação

UCB – Universidade Católica de Brasília

UML – Linguagem Modelo Unificada

WEB – World Wide Web

WEB-APP – WEB Aplication

SUMÁRIO

1 INTRODUÇÃO

O fenômeno da globalização mudou o cenário do mercado fazendo com que as organizações tenham maior liberdade de escolha, aumentando com isso suas exigências, entre outras, elevando também as exigências nas questões de qualidade. Assim, a qualidade tornou-se um ponto estratégico de competitividade nas organizações. Pode-se afirmar que qualidade são atitudes que diminuem os desperdícios, melhoram os processos e aumentam o prestígio das empresas.

Na área de desenvolvimento de *software*, Bartié (2002, p. 5) afirma que "Um processo de garantia da qualidade de *software* é a parte de uma estratégia de sobrevivência em um mercado cada vez mais exigente e competitivo".

Com a engenharia de *software*, pode-se controlar o processo de desenvolvimento englobando três itens fundamentais: métodos, ferramentas e procedimentos. A inexistência de erros (ou a minimização destes), em um produto de *software*, pode ser um dos indicadores da qualidade de produto. Não pode se afirmar que um sistema ou aplicativo tem qualidade se ele apresentar diversos erros de execução ou não conformidades. Assim, para controlar as falhas em um *software,* existem atividades que podem minimizar os *bugs* e as não conformidades dos requisitos do *software*, são os chamados testes.

Os testes são processos que possuem o objetivo de descobrir erros antes do *software* chegar ao usuário final. O processo de desenvolvimento de *software* deve ser planejado de forma a suportar a validação e verificação de erros.

Atualmente, existem várias ferramentas *open source* que ajudam no desenvolvimento de testes automatizado. Estas ferramentas podem ser utilizadas para a produção de testes, permitindo visualizar a que melhor se encaixa para determinados produtos, negócio e cobertura de testes, gerando a qualidade desejada.

> Trata-se da utilização de ferramentas de testes que possibilitem simular usuários ou atividades humanas de forma a não requerer procedimentos manuais no processo de execução dos testes. Requer profissional especializado e tempo no desenvolvimento da automação dos testes (BARTIÉ, 2002, p.196).

O tema central deste trabalho são os processos, métodos e ferramentas (open source) de testes que podem ser aplicados no desenvolvimento de *software*. Para constatar esses conceitos será realizado um estudo de caso em uma empresa de desenvolvimento de *software*.

1.1 PROBLEMA

Hoje exige-se muito da tecnologia e dos sistemas informatizados, conseqüentemente, a complexidade do desenvolvimento de *software* tem aumentado. Novos *releases*[1] ou solicitações de alterações no *software*, para que fique a "cara" de cada cliente, são feitos como a compra de uma roupa nova. É exigido rapidez e eficiência na entrega. Vale lembrar que: "os erros na produção são extremamente caros" (BARTIÉ, 2002, p.31). Segundo Myers[2], quanto antes forem descobertos os erros da aplicação, mais barato será a sua manutenção. O custo aumenta exponencialmente a cada etapa do processo na qual ele é encontrado, ou seja, o custo da correção é maior para cada fase do ciclo de desenvolvimento do *software* em que o erro passou (2004, p.22). Esse comportamento pode ser observado na figura 1, a seguir.

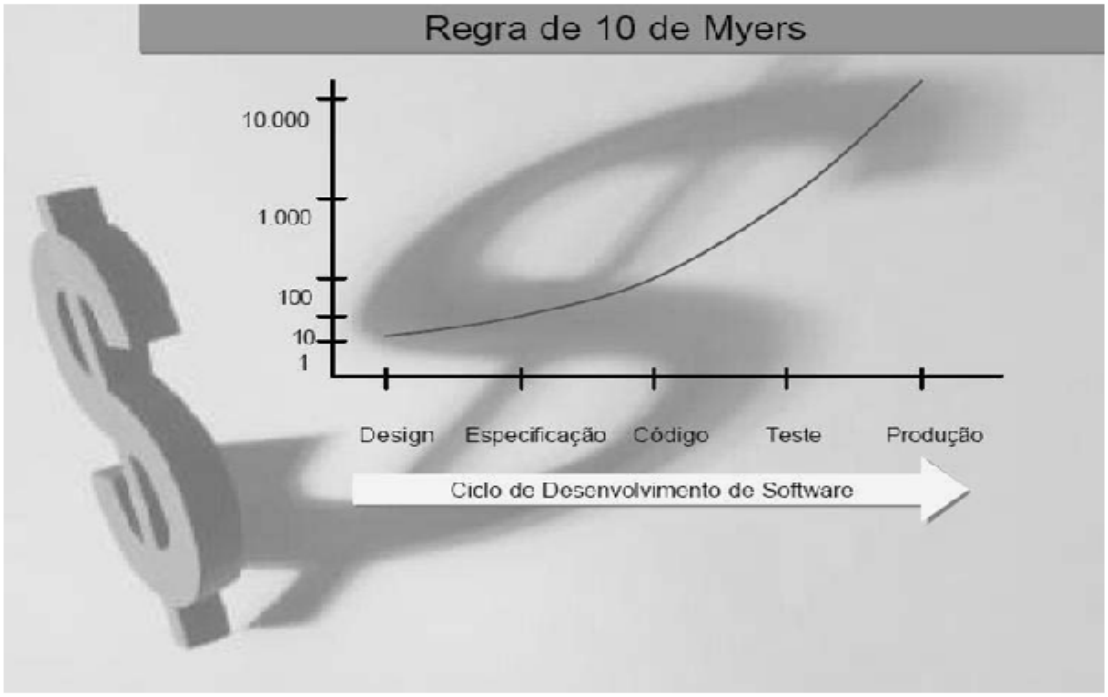


Figura 1: Regra 10 de Myers
Fonte: RIOS, 2006. p. 19

> A atividade de teste é muitas vezes encarada como a simples tarefa de navegar pelo código e corrigir problemas já conhecidos. Em muitas empresas tais tarefas eram realizadas pelos próprios desenvolvedores, não existindo recursos dedicados a essa atividade. Desta forma, os testes são somente realizados tardiamente, quando o produto já está pronto ou quase. (BARTIÉ, 2002, p.3).

[1] *Releases*: do inglês, significa lançamento de versões de *software*

[2] Myers: autor que popularizou a atividade de Teste de *Software*, muitos profissionais comemora a data de lançamento do livro como o aniversário da área de teste.

A falta de um processo de testes planejados e automáticos demora a entrega do produto final ao cliente e prejudica as manutenções, além de poder encarecer o custo do produto para conseguir manter a qualidade do mesmo. Esse problema pode ser resumido na seguinte pergunta de pesquisa: É possível construir um processo de teste para uma empresa de desenvolvimento de *software* com o intuito de apoiar o processo de qualidade?

1.2 OBJETIVOS

Serão apresentados a seguir como objetivo geral e específico.

1.2.1 Objetivo geral

Definir um processo de testes para o desenvolvimento de *software* de uma empresa e validar esse processo através da sua aplicação em um produto da organização.

1.2.2 Objetivos específicos

a) estudar metodologias e processos de testes;
b) pesquisar e utilizar ferramentas de testes *open source*;
c) identificar principais requisitos para implantação de processo de teste e sua automação;
d) propor um processo de teste para um determinado produto da empresa;
e) usando ferramentas open source, criar um ambiente integrado para automatizar, na medida do possível, o processo de testes definido;
f) aplicação de processo de testes e sua automação;
g) verificação de cobertura de testes.

1.3 JUSTIFICATIVA

Uma aplicação mal testada poderá gerar grandes problemas, desde a recodificação total da aplicação, prejuízos financeiros (tanto para a empresa desenvolvedora como para os clientes e/ou usuários) e, dependendo da área de aplicação, pode até causar risco de vida às pessoas (por exemplo, em sistemas para controle de tráfego aéreo, etc.).

"Por que esperar para descobrir como os usuários vão reagir ao sistema desenvolvido quando existem diversas maneiras de descobrir este fato" (MOLINARI, 2003, p.37). Assim, como o ditado diz: é melhor prevenir do que remediar, isto vale para a elaboração de um sistema. É melhor que o erro ocorra durante a execução dos testes dentro do ciclo de desenvolvimento do que no usuário final.

"Os testes de *software* passaram a ser o carro-chefe do processo de qualidade de *software*, porém muitas empresas estão descobrindo, na prática, que teste é fundamental" (MOLINARI, 2003, p. 41). Estes podem prevenir erros e economizar recursos da empresa como, por exemplo, tempo dos desenvolvedores e recursos financeiros.

Para que um produto final seja de qualidade, todas as fases de processo de desenvolvimento devem ser bem estudadas, elaboradas, testadas e monitoradas, a fim de garantir que cada funcionalidade resulte como esperado, que cada regra de negócio esteja sendo atendida e que sua integração não gere erros. Com esta qualidade de *software* acentuada, a empresa terá uma vantagem competitiva no exigente mercado atual.

Após 30 anos da publicação do livro de Myers, a maioria de seus conceitos são válidos ainda, por exemplo: regra 10 de Myers, importância da verificação, importância dos testes unitários e de sistemas e os casos de testes (CORREIA, 2009, p. 19).

Os conceitos de Myers, ainda persistentes após tanto tempo, continuam sendo assuntos atuais. Esses foram temas de debates no primeiro encontro mensal da Associação Latino-Americana de teste de *software* (ALATS) [3] que aconteceu em São Paulo em abril de 2009. Continua-se afirmando que: os testes unitários e testes de sistemas removem de 30% a 50% dos defeitos do *software* lançados no mercado, assim como os sistemas podem ser colocados em produção com 49% de defeitos, sendo que a revisão de código pode reduzir de

[3] ALATS: É uma entidade sem fins lucrativos que foi criada com a finalidade de reunir profissionais das áreas de teste e de qualidade de sistemas com o intuito de servir de apoio a troca constante de informações objetivando a melhoria dos processos de teste, mantendo-se a sua integração como o processo de desenvolvimento de aplicações.

20% a 30% desses 49% de defeitos (CORREIA, 2009, p. 22). Conclui-se que, com os testes e revisões de código, pode-se lançar no mercado um produto com 80% de defeitos removidos, um índice alto, considerando que defeito zero não existe. Uma aplicação bem testada diminui o custo monetário e aumenta a satisfação do cliente.

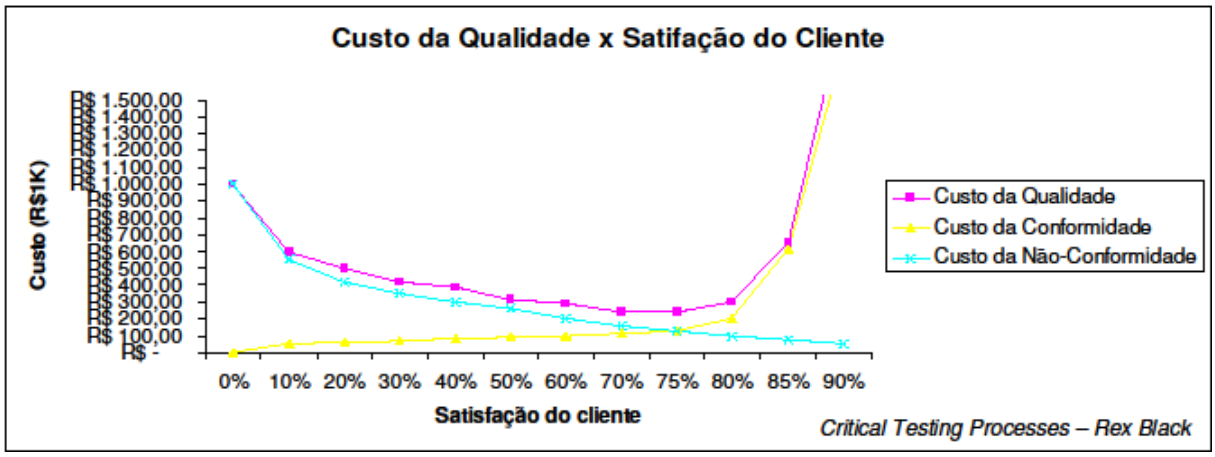


Figura 2: Custo da qualidade x satisfação do cliente
Fonte: *Qualiti software processes*, 2002.

Segundo o gráfico acima, cuja fonte é a empresa, *Qualiti Software Processes*[4], o custo da qualidade de um *software* é o valor do custo da conformidade mais o custo da não conformidade, em que o custo da conformidade é o valor do custo da prevenção mais o custo de detecção e o custo da não conformidade é o valor do custo da falha interna mais o custo da falha externa, representada pela fórmula abaixo:

CQualidade = (CPrevenção + CDetecção) + (CFInterna + CFExterna)

Quadro 1: custo da qualidade
Fonte*: Qualiti software processe*s, 2002

Com essa análise, é possível verificar onde está o problema e tomar decisões para o melhoramento do processo de qualidade.

[4] *Qualiti software processes*: Empresa especializada em processos, arquiteturas e ferramentas de apoio ao desenvolvimento, responsável em desenvolver *software* de qualidade dentro do prazo e custos previstos.

1.4 ESTRUTURA DA MONOGRAFIA

A estrutura da monografia está contida em seis capítulos:

Capítulo 1 – Introdução: Fornece informações sobre o assunto abordado, relatando o problema e justificativa para a pesquisa e objetivos desejados após a revisão bibliográfica.

Capítulo 2 – Revisão Bibliográfica: Contém as informações da pesquisa feitas sobre a teoria, objetivando uma base de conhecimento da área de qualidade de *software*, focando a atividade de testes de aplicações.

Capítulo 3 – Método: Neste capítulo é apresentada a metodologia adotada, assim como a caracterização da pesquisa, considerando as delimitações encontradas.

Capítulo 4 – Descrição do estudo de caso: Após toda a pesquisa de revisão bibliográfica e do método, este capítulo faz a descrição de um caso de estudo com o levantamento de atividades para a proposta de desenvolvimento da solução do problema.

Capítulo 5 – Elaboração de planejamento de testes: Finalmente, neste capítulo, será colocado em prática todo o conhecimento adquirido no decorrer da monografia. Nesta fase a confecção do processo de teste para a aplicação escolhida será feita com apóio de ferramentas *open source*.

Capítulo 6 – Conclusões e trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Para o desenvolvimento da monografia, foram selecionadas algumas bibliografias para estudo com a finalidade de fundamentar a proposta de um processo de planejamento, execução e análise de testes.

Foram revisados vários conceitos relacionados à engenharia de *software*, sempre focando, principalmente, os testes. Alguns assuntos abordados são: Engenharia de *software*, processos de qualidade, testes, ferramentas, documentação, planejamento e métodos de aplicação de testes.

2.1 ENGENHARIA DE *SOFTWARE* E QUALIDADE DE *SOFTWARE*

A disciplina que estuda o processo de desenvolvimento de aplicações e sistemas de *software* é chamada de Engenharia de *Software*. "A engenharia de *software* visa à criação de produtos de *software* que atendam as necessidades de pessoas e das instituições" (PAULA FILHO, 2007, p. 8).

> Os métodos de engenharia de *software* fornecem a técnica de "como fazer" para construir softwares. Eles abrangem um amplo conjunto de tarefas que incluem comunicação, análise de requisitos, modelagem de projeto, construção de programas, testes e manutenção (PRESSMAN, 2006, p. 18).

O assunto e os métodos focados, nesta monografia, serão aqueles relacionados aos testes.

"As ferramentas de engenharia de *software* fornecem apoio automatizado ou semi-automatizado para o processo e para os métodos" (PRESSMAN, 2006, p.18). Algumas dessas ferramentas serão analisadas e usadas para o planejamento e gerenciamento de testes.

Segundo Bartié (2002, p.8), "A cada ano mais informações, técnicas, metodologias, ferramentas e empresas especializam-se em assuntos cada vez mais voltados a como aprimorar o processo de engenharia de *software*.

É importante salientar que os testes pertencem ao conjunto de atividades da engenharia de *software*, os testes, por si só, não irão garantir a total qualidade, porém

ajudarão. A engenharia de *software* está presente em todas as etapas da construção de um *software*, desde a análise de requisitos até o presente ciclo de vida da aplicação.

Para garantir a qualidade de um *software*, é necessário que todas as etapas de sua confecção sejam monitoradas e que sigam um processo metódico (métricas estabelecidas) atingindo o objetivo de cada etapa. "Se desejarmos produzir *software* com alta qualidade, é necessário investir em qualidade em todos os pontos do processo" (BARTIÉ, 2002, p. 16).

Nesse cenário exigente e dinâmico, as empresas desenvolvedoras têm o desafio de acompanhar a evolução tecnológica na elaboração dos produtos de *software*. Nesse sentido, *software* auxilia as empresas a optar pela melhor solução que traga a segurança e qualidade exigida nos serviços e produtos.

Segundo Bartié, qualidade de *software* é um processo sistemático que focaliza todas as etapas e artefatos[5] produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos (2002, p. 3).

2.1.1 Dimensão da qualidade

A qualidade está ligada a duas definições: adequação aos requisitos, conforme especificação elaborada do *software* e adequação de uso, conforme as necessidades do usuário (BARTIÉ, 2002, p. 17). Adequações de requisitos e de uso estão associadas aos testes de verificação e validação.

Segundo Pressman (2006, p.290), a aplicação correta das ferramentas, métodos, revisões de código, monitoração das atividades e métricas levam à qualidade que é confirmada durante os testes.

[5] artefatos: é um tipo de recurso produzido ou consumido em uma atividade, pode ser um desenho, requisito, código, diagramas, etc.

2.1.1.1 Validação X verificação de *software*

Verificação é, geralmente, confundida com a validação de *software*. Apesar de ambas possuírem a mesma idéia (identificar erros), elas possuem atividades diferentes. Para Molinari, na verificação, o teste vai ao encontro do requerimento especificado durante o desenvolvimento do *software* e a validação checa, se a aplicação atende aos requerimentos dos clientes (2003, p. 23).

Inthurn considera que os procedimentos de validação e verificação são complementares, a verificação pode ser aplicada desde o início do processo de desenvolvimento e a validação executada nas fases que abrangem os testes de integração e sistemas (2001, p. 47). Molinari possui a mesma idéia que Inthurn, quando diz que são complementares e acrescenta que, em hipótese nenhuma, podem ser considerada atividades redundantes, pois são de natureza e objetivo final divergentes, agregando ao processo de detecção de erros na aplicação e melhorando, assim, a qualidade do *software* (2002, p. 37).

Segundo Pressman, verificação refere-se ao conjunto de tarefas que garantem que o *software* implemente corretamente uma função específica, e validação ao conjunto de tarefas que o *software* construído está conforme o que o cliente deseja (2006, p. 289).

As organizações que utilizam apenas da atividade de validação, no final da confecção do *software*, possuem uma tendência de encontrar mais problemas no final do projeto, porque os testes são feitos apenas quando a aplicação já se encontra completa, não houve uma verificação das funcionalidades especificadas.

A verificação, em geral, tem seu resultado ao longo da etapa de desenvolvimento de *software*, pois esse tipo de teste é feito antes da criação do aplicativo, objetivando garantir que todas as decisões e definições sejam entendidas e atendidas pela equipe do processo responsável pela construção do *software*. Esta verificação deve ser aplicada durante todas as fases do produto, ou seja, na documentação, manuais, implementação, entre outros.

“O maior propósito das atividades de verificação e validação é garantir que o projeto do *software*, a codificação e a documentação 'encontrem' todos os requerimentos que foram definidos” (MOLINARI, 2003, p. 28).

Resumidamente, a verificação está relacionada diretamente com a qualidade de processos para o desenvolvimento da aplicação (revisões de requisitos, inspeções de códigos...) enquanto a validação está alinhada à qualidade do produto (testes de integração,

testes de aceitação, testes de regressão...), ou seja, testes efetuados na aplicação já desenvolvida.

2.1.2 Garantindo a qualidade de processo

Os testes de verificação devem garantir a qualidade do processo de desenvolvimento do *software*, avaliando a documentação e atividades com objetivo de encontrar erros e quebras dos processos. Baseia-se na garantia de que o que foi especificado irá funcionar corretamente. O processo deverá ser feito da melhor forma, permitindo custo e cronograma aceitável, resultando em um produto de qualidade (BARTIÉ, 2002, p. 74).

O quadro apresentado, a seguir, sintetiza as atividades e o foco de um processo com garantia de qualidade.

Foco	- documentação (chamadas de revisões); - tarefas (chamadas de auditoria).
Atividades	- ser sistemáticas (obedecer a regras); - ter a participação de todos os profissionais; - ter as funcionalidades a serem discutidas e revistas (regra de negócio); - ter inspeções técnicas (analisar se os documentos e tarefas estão sendo feitos dentro do modelo escolhido, padronização); - ser flexíveis de forma que as mudanças não gerem impactos grandes e que as atividades de verificação sejam executadas de forma a manter a documentação atualizada; - ter todas as funcionalidades, recebendo a atenção por todos da equipe e gerar revisões com objetivo de identificar problemas.

Quadro 2: Garantindo a qualidade
Fonte: Baseado em BARTIÉ, 2002)

2.1.3 Garantindo a qualidade do produto

Nesta fase do processo, o *software* encontra-se completo. Os testes de *software* buscam todos os meios e recursos possíveis para gerar uma estrutura que consiga capturar a maioria dos erros (BARTIÉ, 2002, p. 103).

A figura resume as diferentes dimensões que caracterizam a qualidade do produto.

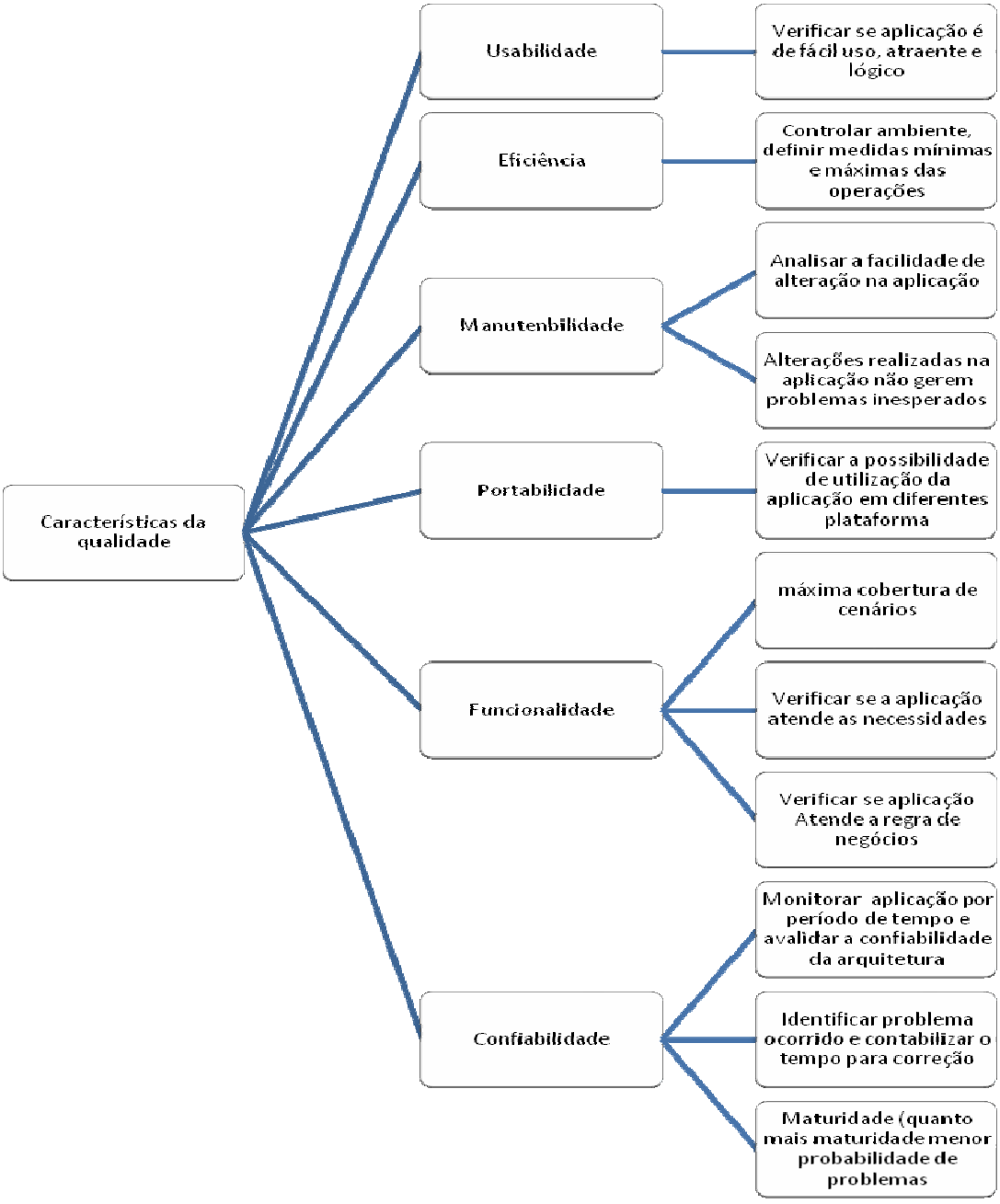


Figura 3: Características da qualidade do produto.
Fonte: Baseado em Koscianski, modelo de qualidade (2006, p.210).

2.1.3.1 Influência dos requisitos na qualidade

Os requisitos são definidos pelas análises, especificações, rastreamento de processos (ciclo de vida do requisito), validações das funcionalidades e comportamentos que um *software* deverá atender. Caso esses requisitos não atendam aos desejos e necessidades dos usuários finais, eles prejudicam a qualidade de *software*.

Segundo Paula Filho (2003), para que um requisito seja de alta qualidade, ele deve ser claro, completo, sem ambigüidade, implementável, consistente e testável. Caso estes atributos não sejam atendidos, os requisitos devem ser revisados e renegociados com o cliente, porque uma boa engenharia de requisitos é essencial para o desenvolvimento de um bom produto (incluindo desenvolvimento e testes do que será implementado).

Para que um *software* seja construído da forma que o cliente deseja, é preciso saber claramente os requisitos. Um *software* pode atingir a qualidade desejada para o que foi especificado, porém não para o que era esperado.

> Essas falhas se devem às dificuldades em entender o que o usuário quer, descrições incompletas e mudanças não controladas nos requisitos... Portanto, realizar corretamente o levantamento e administração de requisitos é vital para a qualidade do software.(KOSCIANSKI, 2006, p. 176).

Quando os requisitos não são suficientes para o entendimento da aplicação ou não atendem as necessidades do usuário final, pode-se, também, estar testando a funcionalidade de forma errada. Para que isso não aconteça, é necessário que os requisitos sejam sempre bem claros para o entendimento de todos da equipe e com aval do cliente.

> Definir com precisão os requisitos de um *software* permite que todos os recursos da empresa e a energia da equipe de desenvolvimento sejam direcionados a um fim claro. Sem uma definição precisa daquele que se pretende construir, perde-se tempo, mais erros são cometidos e a qualidade do produto final é incerta (KOSCIANSKI, 2006, p. 172).

Os requisitos podem ser divididos em requisitos funcionais e requisitos não funcionais:

- requisitos funcionais: comportamentos que a aplicação deverá gerar diante certa ação do usuário (PAULA FILHO, 2003, p. 5).

- requisitos não funcionais: comportamentos que representam a qualidade de uma funcionalidade, o nível do serviço requerido, por exemplo: uso da aplicação em termos de desempenho, segurança, etc.

2.2 NORMAS DE QUALIDADE

Normas de qualidade são padrões que indicam e delimitam os esforços necessários através de processos para atingir os objetivos. Fornecem modelos que ajudam a confecção e a avaliação da qualidade de um produto.

"Uma norma de qualidade nasce de uma necessidade de padronização" (KOSCIANSKI, 2007, p.52). Com o crescimento da complexidade na construção e funcionalidades de *software*, existe a necessidade de criar padrões e normas para um controle maior e parâmetros para avaliação da qualidade do produto e processos, assim, possibilitando identificar oportunidades, visar a melhorias e a realizar um plano de ações.

2.2.1 Norma ISO 9126/NBR 13596 - Qualidade de produtos de *software*

A Organização Internacional de Padronização (ISO) é uma Organização não governamental, atuando desde 1947, coordenando o trabalho de órgãos de 127 países membros para promover a padronização de normas técnicas em âmbito mundial. Ela define padrões de referências para atingir a qualidade de algo. "As normas ISO série 9000 tornaram-se sinônimo de preocupação com qualidade em todo o mundo; a sigla é conhecida por consumidor dos mais diversos produtos e funciona, muitas vezes, como um eficiente apelo de *marketing*" (KOSCIANSKI, 2006, p. 156).

A ISO 9126 tem como atingir os objetivos de funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade (BASTOS, 2007, p. 261).

Segundo Koscianski (2006, p. 207), a norma ISO/IEC 9126 é composta pelas seguintes partes:

a) **ISO/IEC 9126-1 Modelo de qualidade**: definição de um modelo com características de qualidade segundo os atores envolvidos para a confecção de um produto (desenvolvedores, cliente, gerentes entre outros);

b) **ISO/IEC 9126-2 Métricas externas**: testes que avaliam o produto como uma caixa preta[6] (nada se conhece sobre o código), verificar a qualidade das funcionalidades do produto;

c) **ISO/IEC 9126-3 Métricas internas**: testes que avaliam o produto como uma caixa branca[7], ou seja, verifica-se a organização, complexidade algorítmica do código;

d) **ISO/IEC 9126-4 Métricas de qualidade de utilização**: referente ao *hardware* e ao ambiente utilizado, treinamento oferecido aos usuários do software.

A NBR 13596 é a versão brasileira da norma ISO 9126, possuem o objetivo de avaliar o produto de *software* de qualidade de diretrizes para o seu uso no Brasil.

2.2.2 IEEE. 829 Padrões para documentação do teste de *software*

Os padrões de documentações são um ponto de partida para as empresas organizarem os seus processos.

O IEEE (*Institute Electrical and Electronics Engineers*) é uma associação sem fins lucrativos, composta por profissionais interessados em tecnologias avançadas. Dedicam-se ao avanço da teoria e da prática da engenharia nos campos da eletrônica, eletricidade e computação. No escopo da norma IEEE.829, os testes são visualizados como um projeto paralelo ao projeto de desenvolvimento de *software.* Segundo a Sociedade de computadores e equipamentos subsidiários (SUCESU)[8], esta norma abrange as seguintes documentações (2004, p. 16):

[6] Caixa preta: Na subseção 2.3.3.3 são apresentadas as características e informações sobre a estratégia de caixa preta.

[7] Caixa branca: Na subseção 2.3.3.2 são apresentadas as características e informações sobre a estratégica de caixa branca.

[8] SUSESU: É uma sociedade civil sem fins lucrativos e sem caráter político-partidário, de representação política dos seus associados e de coordenação, promoção, execução e indução de atividades no setor de informática e telecomunicações.

Documentação	Tarefas
Planejamento de teste	• Definir o objetivo do teste, se o teste é funcional, de integração, de sistema ou de unidade. • Definir o limite de cada teste e sua abrangência. • Obter as referências de documentos que são relevantes e os conhecimentos necessários. • Verificar quais são os recursos existentes para o planejamento dos testes. • Definir cronograma das atividades de testes. • Identificar a configuração dos ambientes de testes.
Especificação do projeto de teste	• Detalhamento do plano de teste, capturando as características que deverão ser testadas. Estas características podem ser vitais para a manutenção da aplicação.
Especificação de casos de teste	• Detalhamento de cada característica que será testada.
Especificação do procedimento de testes	• Detalha os procedimentos para a execução dos casos de testes.
Logs[9] dos testes (diário de bordo)	• Detalhes importantes da execução dos testes.
Relatório de incidente	• Relatam incidentes que deverão ser testados de forma mais detalhada.
Relatório com resumo dos testes	• Resultado dos testes.

Quadro 3: Documentação de teste de *software*
Fonte: SUCESU (2004).

[9] *Logs*: descrição dos passos de um processo executado

2.2.3 MPS.BR: Melhoria de processo do *software* brasileiro

O MPS.BR iniciou em 2003. É Composta por pesquisadores pertencentes as seguintes instituições:

SOFTEX (Sociedade para Promoção da Excelência do *Software* Brasileiro)	Organização não governamental que possui o objetivo de promover, executar e apoiar as atividades de inovação e de desenvolvimento científico e tecnológico. Promove capital humano e transfere o conhecimento adquirido através de educação, oferecendo treinamentos apropriados de natureza técnica e mercadológica em tecnologia de *software*.
COPPE/UFRJ (Conselho de Ensino Pós Graduados e Pesquisa/Universidade Federal do Rio de Janeiro).	Incubadora de empresas de programas de pós-graduação em engenharia da Universidade do Rio de Janeiro.
CESAR (Centro de Estudos e Sistemas Avançados de Recife)	Instituto privado do estado de Pernambuco. Utiliza tecnologia de comunicação para a criação de produtos, serviços, processos e empresas inovadoras.
CELEPAR (Companhia de Informática do Paraná)	Empresa que possui o objetivo de aperfeiçoar os processos de qualidade de *software* baseados nas normas ISO/IEC 15504 das pequenas e médias empresas de desenvolvimento de *software*. Ela é hospedeira do subcomitê de *software* da ABNT (Associação Brasileira de Normas Técnicas).

Quadro 4: Instituições que compõem a MPS-BR.
Fonte: SUCESU (2004).

Segundo Koscianski (2006, p.142-144), a MPS.BR possui modelos usados como referência para obter a qualidade de *software*:

a) **Modelo de referência de melhoria de processos de *software* (MR-MPS):** referências com requisitos que precisam ser cumpridos pelas organizações, definindo os níveis de capacitação e processos.

b) **Método de Avaliação (MA-MPS):** é um guia de avaliação de processos, que permite visualizar o desempenho e maturidade da empresa, assim, conseguindo visualizar qual será o aproveitamento futuro da organização.

c) **Método de Negócio (MN-MPS):** possui regras para a inclusão do modelo de referência de melhoria no processo de *software*.

2.2.4 MPT – Melhoria de Processo de Teste Brasileiro

O MPT foi criado pela Softex com o princípio de ser compatível com o modelo MPS.BR, e está baseado em alguns critérios pelo CMMI (*Capability Maturity Model Integration*) [10]. O objetivo do modelo é ajudar a área de testes de *software* das empresas, independente de seu porte. Assim, permite às empresas avaliar e estimular as equipes de testes a alcançarem um nível maior de maturidade sem modificações grandes em custos operacionais.

Esse modelo exige alterações na composição de recursos humanos da empresa, dependendo do nível de maturidade das áreas de testes, visando ao papel de gerência de projetos de teste de *software*, por exemplo [nível 1] ou, além do papel anterior, visar o papel de gerência de requisitos de testes [nível 2] . Este modelo é composto por oito níveis, sendo que as empresas que não usam o modelo MPT encontram-se no nível zero (RIOSOFT, 2008).

[10] CMMI: Formado pelas melhores práticas de desenvolvimento e manutenção de produtos e serviços, que cobrem o ciclo de vida do produto desde sua concepção até sua entrega ou manutenção.

2.3 DEFINIÇÃO DE TESTES DE *SOFTWARE*

Segundo Bartié (2002, p. 22) e Myers (2004, p. 6), teste são processos analisados, sistematizados e planejados com o objetivo de encontrar erros.

Para Molinari, o teste de *software* é a mais popular estratégia de gerenciamento de risco. São usados para verificar o encontro dos requisitos com o produto (2003, p.28).

Geralmente, ao efetuar um teste, idealiza-se um cenário positivo, ou seja, testa-se apenas o esperado, porém, para diminuir a probabilidade de erros, devem-se imaginar todos os cenários possíveis de ações da aplicação, seja ela positiva ou negativa. Por isso, segundo Myers, a aplicação não deve ser testada pelo próprio desenvolvedor porque este tem a tendência de sempre ver apenas o cenário positivo (2004, p.16).

“O teste exige que o desenvolvedor descarte noções preconcebidas da 'corretividade' do *software* recém-desenvolvido e, depois, trabalhe duro para projetar casos de testes que 'quebrem' o *software*” (PRESMAN, 2006, p. 315).

> Teste é uma área da engenharia de *software* e tem como objetivo aprimorar a produtividade e fornecer evidências da confiabilidade e qualidade do *software* em complemento a outras atividades de qualidade ao longo do processo de desenvolvimento de *software* (INTHURN, 2001, p. 51).

Myers afirma que teste, com zero-defeito, é impossível de ser realizado, sempre haverá erros que não foram descobertos nos testes, porém podem-se definir processos que minimizem ao máximo incidências de problemas (2004). A procura da perfeição é contínua, porém, como o *software* é confeccionado por seres humanos, a tendência de ter erros é evidente, devido às pessoas, independente de sua qualificação, serem propensas a erros.

A aplicação de testes é mais fácil quando o *software* se encontra em fase de elaboração. Os testes deveriam andar em paralelo com o seu desenvolvimento. Quando a aplicação já se encontra em produção, a elaboração de testes pode ser mais complexa, por exemplo, devido à falta de documentação de regras de negócio. Às vezes, essas aplicações foram feitas através de definição de regras transmitidas “boca-a-boca” em reuniões. Porém a aplicação de testes para essas aplicações são uma grande oportunidade de documentação.

Um teste é considerado bom, segundo Myers (2004, p.20), quando encontra *bugs* que não foram anteriormente encontrados.

Pressman (2006, p. 312) defende que o objetivo dos testes em um *software* é descobrir erros, e que existe uma série de passos que podem ser seguidos como: testes de

unidades, testes de integração, testes de validação e verificação de sistema. Essas etapas devem ser planejadas e, após, serem executadas. Esses passos de testes possuem uma sistemática de técnicas que auxiliam no projeto de casos de testes.

Ao longo de 40 anos, foram criados os sete princípios do processo de teste que afirmam todos os conceitos dos autores citados, anteriormente, e que serão vistos no decorrer da monografia (REX[11], et al, 2005, p.22).

	Princípio	**Descrição**
1	Teste demonstra a presença de defeitos	Teste reduz a quantidade de erros de uma aplicação, mas não pode provar que eles não existem.
2	Teste exaustivo é impossível	Teste exaustivo é testar todas as combinações de entradas e pré-condições para um teste, sendo que a falta de tempo para a conclusão de projeto é frequente, os testes são focados em riscos e prioridades. Testes exaustivos são feitos exceto em casos triviais, quando o risco é muito alto, como o lançamento de um foguete.
3	Teste antecipado	Os testes devem começar o mais cedo possível no ciclo de desenvolvimento do *software*.
4	Agrupamentos de defeitos	Segue o princípio de Pareto[12] que um pequeno número de causas é responsável pela maioria dos problemas. Geralmente 20% dos problemas causa 80% dos erros.
5	Paradoxo do pesticida	Os casos de testes devem ser frequentemente revisados, atualizados e retestados.
6	Teste depende do contexto	Dependendo do contexto existem formas diferentes de se testar.
7	A ilusão da ausência de erros	Se a aplicação não atende as necessidades do cliente é um erro.

Quadro 5: Princípios do processo de testes.
Fonte: baseado em REX, et al, 2005, p.22

[11] Rex Black: presidente do *International Software Testing Qualifications Board* (ISTQB) e presidente e principal consultor da Rex Black Consulting Services (RBCS), uma líder em *software*, *hardware* e sistemas de testes. Publicou vários livros sobre testes incluindo versões em japonês, chinês e indiano como: *Critical Testing Processes, Foundations of Software Testing,* Teste de *Software*. http://www.rbcs-us.com/about-us/our-team

[12] Princípio de Pareto: técnica desenvolvida no final do século XIX pelo italiano Vilfredo Pareto (1848-1923). Este princípio também é conhecido como regra dos 80/20. (ENGEL, 1982, p. 37).

2.3.1 Processo de testes

Segundo Moreira Filho (2004, p. 22), processo de teste é caracterizado pela execução das principais etapas da atividade de testes e de suas subetapas.

O objetivo dos processos de teste é direcionar e indicar a próxima atividade a ser feita. Esses processos são conjuntos de boas práticas que orientam para chegar ao objetivo da melhor maneira (vistos anteriormente na seção sobre as normas). Servem para minimizar os riscos causados por defeitos originados no processo de desenvolvimento. A figura, a seguir, apresenta o processo de testes.

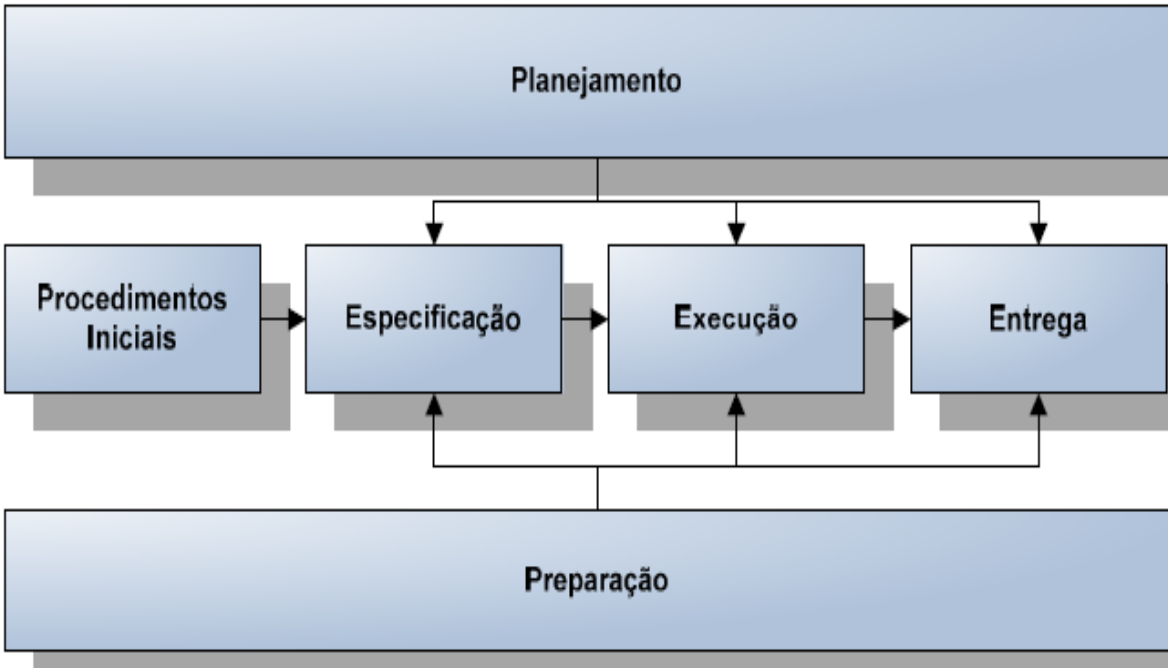


Figura 4: Fases do processo de teste – 3P x 3E.
Fonte: RIOS, et al, 2003, p. 9.

"O processo de testes deve basear-se em uma metodologia aderente ao processo de desenvolvimento, em pessoal técnico qualificado, em ambiente e ferramentas adequadas" (RIOS, et al, 2003, p. 9).

A seguir, faz-se a descrição das fases do processo de testes:

Etapa	Descrição	Distribuição de tempo
Procedimentos iniciais	Esboço do processo de testes (principais atividades, recursos de pessoal e de ambiente)	Não é medido, é o tempo utilizado para adquirir todos os artefatos necessários para a elaboração do projeto de testes.
Planejamento e preparação	Acompanha todo o processo de teste, são atividades complementares ao processo. Captação de requisitos, planejamento do projeto, análise de riscos e preparação de ambiente de testes.	10%
Especificação	Elaboração e revisão de casos de testes e roteiros de testes	40%
Execução	Execução de testes conforme roteiros estabelecidos para os testes. Executar sempre que houver mudanças na aplicação. Analisar os testes executados com sucesso e os testes com defeitos.	45%
Entrega	Finalização do projeto de testes. Registrar toda a documentação e relatar todas as incidências relevantes à melhoria do processo. Relatório dos resultados dos testes (conformidades e não conformidades).	5%

Quadro 6: Descrição das fases do processo de teste
Fonte: Baseado em RIOS, et al, 2003

2.3.2 Ciclo de vida do processo de testes

Segundo Bastos, Rios, Cristalli e Moreira (2007, p. 40), um ciclo de vida de teste começa simultaneamente com o ciclo de vida dos processos de desenvolvimento. A equipe de desenvolvimento e de teste faz uso dos mesmos documentos que a equipe de testes e, em alguns pontos do processo do desenvolvimento, a equipe de teste utilizará as técnicas de testes para avaliar o *software* com o propósito de descobrir defeitos. A validação e verificação são realizadas durante a evolução do desenvolvimento da aplicação com as correções efetuadas.

A seguir, apresenta-se um modelo de processo de testes integrado ao processo de desenvolvimento.

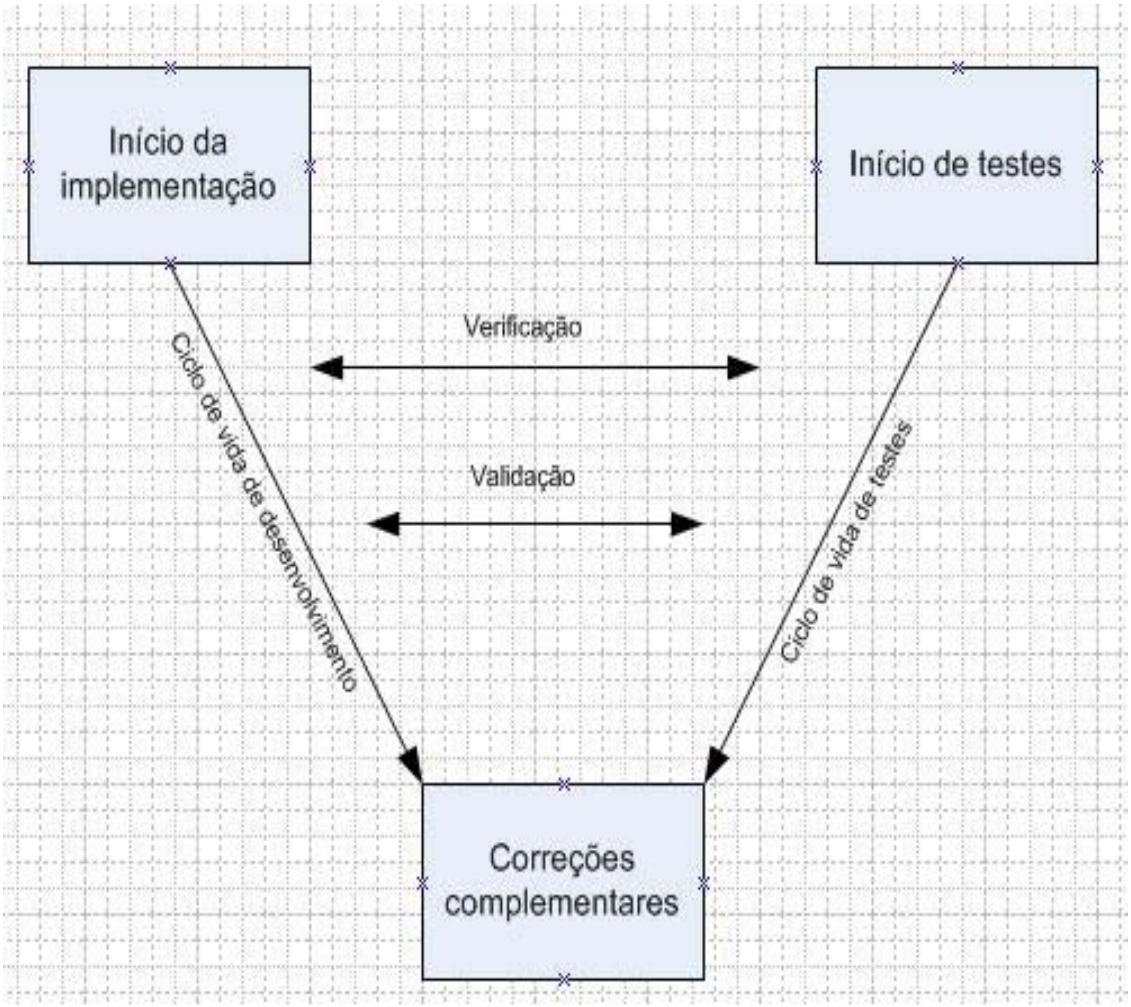


Figura 5: Conceito V de teste de *software*.
Fonte: RIOS, et al., 2007, p. 41.

A seguir, apresentam-se algumas atividades desenvolvidas no ciclo de vida do processo de testes:

Passo	Objetivo	Descrição
1	Acesso ao plano de desenvolvimento	Indispensável para a construção do plano de teste. Avaliar e estimar a quantidade de recursos necessários para testar a aplicação.
2	Desenvolvimento do plano de testes	Preparação do plano de testes.
3	Inspeção ou teste dos requisitos do *software*	Avaliação dos requisitos para confecção dos testes.
4	Inspeção ou teste do desenho de *software*	Verificação do desenho de testes conforme os requisitos.
5	Inspeção ou teste da construção do *software*	Verificação do método de desenvolvimento do *software* de forma a adequar um método para a construção de testes.
6	Execução dos testes	Execução do testes, abordagem de ferramentas e os métodos especificados no plano de testes.
7	Teste de aceitação	Avaliação do *software* pelo cliente.
8	Informação dos resultados dos testes	Resultados dos testes executados.
9	Testes de instalação no *software*	Verificar se o *software* está apto para implantação no ambiente de produção.
10	Testes das mudanças no *software*	Considerações para a implantação do *software*.
11	Avaliação da eficácia dos testes	Avaliar a eficácia dos testes realizados.

Quadro 7: Processo de testes.
Fonte: RIOS, et al., 2003, p. 43.

Os cinco primeiros passos do processo de testes utilizam principalmente a técnica de verificação para validar e corrigir o desenvolvimento do *software*, ou seja, verificam os documentos antes de validar a aplicação. Os próximos seis passos utilizam a técnica de

validação, validando se o sistema está conforme o documento verificado (BASTOS, et al., 2007, p. 43).

2.3.3 Terminologia de testes usada neste trabalho

Com o amadurecimento do ciclo de testes, novas técnicas, tipos e modelos são criados e adaptados para atender todos os ambientes e necessidades das aplicações. O requisito mais importante, para aplicação dos testes, é possuir o conhecimento básico de teste, saber como e onde será executada a aplicação em desenvolvimento ou desenvolvida.

Segundo Molinari, não existe uma unicidade, nem um padrão dominante na nomenclatura dos tipos de testes. Os principais autores sobre testes, seja, livro ou artigos, não conseguem chegar a um consenso sobre nomes diferentes dados para os conceitos, quando falando em testes, apesar disso ser considerado um ponto estratégico para a difusão e disseminação dessas práticas de elaboração de um planejamento de testes. Alguns autores classificam os testes segundo tipos ou abordagens, outros, segundo estratégias, outros, segundo técnicas, embora a definição seja a mesma. Por exemplo, alguns autores dividem os testes em: técnicas de caixa preta e caixa branca, outros em estratégias de caixa preta e branca e, ainda, terceiros chamam de tipo ou abordagens de testes de caixa preta e caixa branca (2003, p. 158).

Para o desenvolvimento da monografia, será adotada a nomenclatura de testes, representada pela figura a seguir em relação às técnicas, aos tipos e às estratégias de testes.

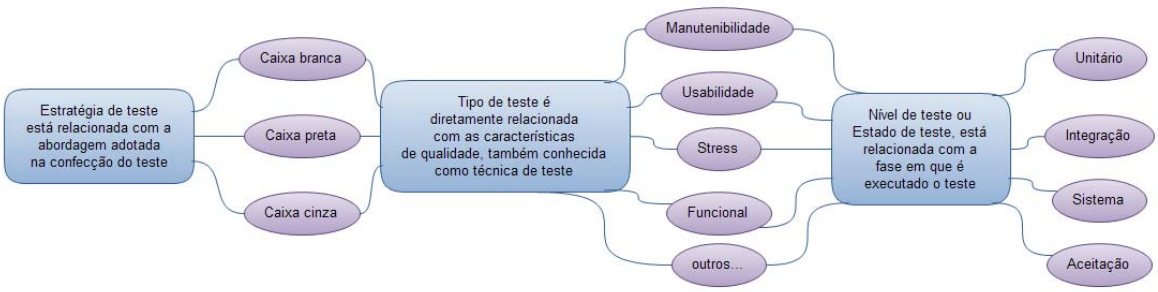


Figura 6: Terminologia de testes de *software*[13]
Fonte: Elaboração da autora, 2009

[13] Os conceitos dos testes informados na figura serão vistos no decorrer da monografia, segundo autores estudados.

2.3.3.1 Estratégias de testes

Segundo Rios e Moreira Filho, a estratégia de testes é elaborada conforme a abordagem que será aplicada, ou seja, descrição de como o teste será feito, identificando o nível, o modelo, as técnicas e as ferramentas que serão adotados no processo (2003).

A estratégia de teste deve levar em consideração sempre os riscos, os custos e os prazos. Deveria se basear em uma política corporativa que definisse os parâmetros básicos e os critérios para sua aplicação, esta, geralmente, representada pelo papel do comercial da empresa. Para uma organização de desenvolvimento de *software*, essas políticas corporativas são fatores críticos para o sucesso da aplicação. (MOREIRA FILHO et al, 2003, p. 18).

Existem, basicamente, dois métodos técnicos de testes aplicáveis, tanto para sistemas estruturados como para sistemas orientados a objetos: tipo caixa branca e tipo caixa preta, porém, atualmente, também, se observa a existência do tipo de caixa cinza que será descrito mais a frente.

Atualmente, existem muitas maneiras de se testar um *software*. Mesmo assim, existem as técnicas que sempre foram muito utilizadas em sistemas desenvolvidos sobre linguagens estruturadas que ainda hoje têm grande valia para os sistemas orientados a objeto. Apesar de os paradigmas de desenvolvimento ser diferentes, os objetivos principais dessas técnicas continuam a ser os mesmos: encontrar falhas no *software* (DIAS NETO, 2007, p. 56-57).

Segundo Bartié, a estratégia escolhida [método de teste] determina o modo de como será estabelecido o problema e de como serão conduzidos os processos de testes visando sempre ao menor esforço e a ampliar a probabilidade de encontrar *bugs* na aplicação (2002, p.104).

Apesar de existirem dois tipos de testes, isso não visa a escolha apenas de um deles. Segundo Bartié, essas duas estratégias [tipos de testes] são complementares e não exclusivas, a sua associação resulta em um produto [*software*] com uma qualidade maior.

A aplicação dos métodos e o conhecimento das regras são os responsáveis pelo funcionamento adequado dos testes. De nada adianta a ferramenta sem a técnica. Cada tipo de teste possui estágios, e estes dependem da fase em que o teste é aplicado.

Pressman acredita que uma estratégia de testes deve ser flexível na confecção de testes, porém rígida para promover planejamento e acompanhamento gerencial à medida que a

aplicação sofre modificações (2006, p. 288). Para ele, a estratégia de teste deve conter os testes desde unitários até os testes de sistema que validam as principais funções da aplicação com base nos requisitos.

2.3.3.2 Testes de caixa branca (teste estrutural)

Para Paula Pádua (2007), Molinari (2003), Myers (2004) e Bastos (2007), o método de caixa branca ou teste estrutural tem como objetivo garantir a estrutura interna sólida, testando a robustez e o funcionamento do *software* no contexto técnico em que será instalado.

Segundo Bastos, Rios, Cristalli e Moreira (2007), o objetivo dos testes estruturais é saber se a aplicação possui uma arquitetura confiável e em perfeito funcionamento, buscando verificar as tecnologias usadas em sua confecção e se a aplicação funciona como uma unidade coesa, ou seja, testar todos os fluxos internos possíveis da unidade em questão.

Para a realização dos testes do tipo caixa branca também conhecida como teste de baixo nível, necessita-se que o profissional tenha conhecimento e acesso à arquitetura da aplicação, acesso ao código fonte, estrutura de dados, etc. Por exigir conhecimento do código fonte e banco de dados, entre outros, este tipo de testes é geralmente, feito pelos próprios programadores. Esta prática pode vir a ser uma estratégia perigosa, devido ao programador sempre ver o lado positivo dos testes ou executar os chamados “testes viciados” ou apenas os testes positivos. O ideal seria outra pessoa para realizar esses testes.

“O teste caixa-branca permite ao avaliador concentrar a atenção nos pontos mais importantes ou 'perigosos' do código, de uma maneira mais precisa do que no caso do testes de caixa-preta”. (KOSCIANSKI, 2007, p. 345).

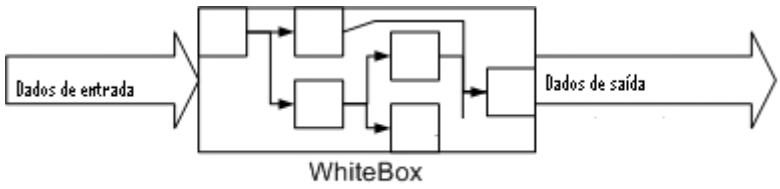


Figura 7 – Técnica de testes de caixa branca.
Fonte: Baseado em Bartié (2002).

Os testes de caixa branca ou estruturais, ainda, podem ser classificados pelas seguintes técnicas (tipos de testes):

a) **testes de carga:** nesse teste, é verificada a capacidade da aplicação quando submetido ao aumento progressivo de dados nos seus processos;

b) **testes de conformidade:** esse teste possui o objetivo de verificar se o *software* foi desenvolvido conforme as normas, padrões e procedimentos da área de tecnologia da informação. É visto a metodologia aplicada para o desenvolvimento com o intuito de diminuir a manutenibilidade do sistema, assim, como o aumento de qualidade. Avalia se a documentação da aplicação está completa e compreendida;

c) **testes de desempenho ou performance:** são responsáveis em verificar se os níveis de desempenho atendem as necessidades dos usuários finais, ou seja, o tempo de resposta de um processo solicitado pelo cliente;

d) **testes de estresse**: avalia o comportamento da aplicação em condições mínimas de operação ou situações anormais como grande volume de dados ou de acessos.

e) **testes de execução:** esse tipo de teste possibilita a verificação dos pontos mais críticos ou habilidade da aplicação (satisfaz os critérios de desempenho), confirmar se o *hardware* ou *software* atende a capacidade de processamento necessária;

f) **testes de operação:** verificar se a aplicação opera adequadamente em ambiente de produção, analisar se a documentação ou manual é instintiva para encaminhar o usuário ao objetivo da funcionalidade;

g) **testes de recuperação (contingência):** teste de capacidade da recuperação da aplicação, ou seja, a garantia da continuidade dos processos após algum problema ocorrido no *software*;

h) **testes de segurança:** avalia os processos de proteção e confidencialidade dos dados contra invasão de terceiros indevidos.

2.3.3.3 Testes de caixa preta (teste funcional)

O teste de caixa preta trabalha com a verificação de dados de entrada e saída, como foi implementada a estrutura interna não importa neste momento. Serve para verificar se a funcionalidade atende aos requisitos. Nessa etapa de testes, o código não é preocupação, mas o funcionamento do sistema ou módulo.

Figura 8 – Técnica de testes de caixa preta
Fonte: Baseado em Bartié (2002)

Algumas das técnicas funcionais ou tipo de testes de caixa preta são:

a) **teste de controle:** assegurar que o processamento de dados, arquivos e trilhas de auditoria, assim, como *backup* dos dados, mantenham-se completos e consistentes, atendendo as necessidades dos clientes. Em caso de erros, no sistema, que seja possível a reconstituição e verificação de erro ocorrido com os dados processados;

b) **teste de interconexão ou integração:** visualizar se as transações de dados entre os *softwares* da aplicação chegam íntegras;

c) **testes paralelos:** verifica se a mesma aplicação com versões diferentes resultam no esperado de cada versão. Esse teste é feito para eliminar a dúvida com relação à correção do problema encontrado na versão utilizada;

d) **testes de requisitos:** são efetuadas através de criações de condições dos testes e *checklists*[14] de funcionalidades, conforme documentado e conformidade das políticas e regulamentos da organização. (RIOS, et al, 2003, p. 59);

e) **testes de regressão:** segundo Paula Filho (2003, p.185), essa técnica de teste é especial, eles devem ser sempre automatizados, pois seu objetivo é garantir que as

[14] *Checklists:* Termo inglês que significa lista de verificação.

alterações (manutenções) feitas na aplicação não prejudiquem outras partes do sistema já testado. Essas alterações podem introduzir erros em outras partes do sistema;

f) **testes de suporte manual:** verificar se os *checklists* e manuais estão documentados e atualizados, conforme o produto produzido e pessoal adequadamente treinado;

g) **testes de tratamento de erros:** o objetivo dessa técnica é verificar se todas as condições de possíveis erros são esperadas e tratadas pelo sistema.

2.3.3.4 Testes de caixa cinza

Esse teste de caixa cinza faz uso das técnicas de caixa-preta e caixa-branca, ou seja, seus testes englobam o conhecimento tanto da estrutura interna (não necessariamente o acesso do código-fonte e sim do conhecimento da lógica em que foram implementadas as funcionalidades) como o comportamento da aplicação através de um evento disparado por um usuário.

Segundo Nogueira (2008), esse tipo de teste de *software* é recente. Sua definição seria um ponto de equilíbrio virtual entre os testes de caixa preta e testes de caixa branca. Nesse teste, junta-se o conhecimento interno do produto e saídas esperadas. O conhecimento interno é dado através de um modelo de arquitetura da aplicação ou diagrama que represente a classe ou o diagrama de sequencia.

2.3.4 Estágio ou nível de testes

O nível de testes, segundo Rios (2003), Myers (2004) e Bastos (2007), é uma das dimensões dos testes que representa em que fase do desenvolvimento do *software* se aplica cada um dos testes a seguir.

2.3.4.1 Testes unitários

Os testes unitários são os testes efetuados em uma unidade de código, ou seja, em uma lógica de funcionalidade. Essa unidade de código pode ser uma classe ou grupo de classes dependentes.

Segundo Pressman (2006, p. 295), testes unitários focalizam os esforços na menor unidade de projeto do *software* (componente ou módulo do *software*), enfocam a lógica interna de processo e a estrutura de dados dentro do limite de um componente ou módulo de *software*. Esses testes, geralmente, capturam erros de cálculos aritméticos, fluxo de controle inadequado, laços de repetição e limites, regra de um método não atendido com determinada entrada de dados, tratamento de exceções, entre outros.

O conceito de teste unitário no contexto Orientação a Objetos (OO) sofre algumas modificações. Muda tanto na estratégia como na sua tática de teste.

Em OO devido a hierarquia de classes[15], para testar uma única operação isoladamente como efetuada em testes estruturados, um método deverá ser testado no contexto de cada subclasse. Segundo Pressman (2006, p. 303), os testes unitários, para *software* OO, devem ser gerados através das operações encapsuladas[16] na classe e pelo estado de comportamento da classe.

2.3.4.2 Testes de integração

Testes de integração são efetuados na interconexão das funcionalidades testadas de forma unitária. Isto é, a validação se o conjunto de unidades implementadas funcionam em conjunto. Eles “Garantem que um ou mais componentes combinados (ou unidades) funcionam corretamente” (MOLINARI, 2003, p. 160).

No contexto OO**, s**egundo Pressman (2006, p.303), os testes integrados podem ser realizados de duas formas:

[15] Classes: Modelo que indica os tipos de estados e de comportamento que o objetos do seu tipo possui (Sierra et al, 2006, p. 2)

[16] Encapsuladas: Evitar acesso de dados das propriedades dos objetos diretamente, acesso somente por métodos (Inthurn, 2001, p.68)

- **caminho de execução:** para esses testes são necessárias a utilização de todas as classes necessárias para a realização de uma operação, ou seja, são utilizadas todas as classes que compõem a efetivação do evento. Essas classes executam as tarefas da operação e então é analisada sua saída. Para cada possível saída, é feito um teste integrado com diferentes dados e analisado se a saída é a esperada para determinada entrada de dados.
- **baseado no uso:** todas as classes são testadas independentemente e, conforme o uso dessas classes em outras partes do sistema, é testado novamente, ou seja, a camada de classes que chamam as classes independentes é testada.

2.3.4.3 Teste de sistema

Este teste envolve o ambiente inteiro da aplicação, *software*, *hardware* e interfaces. É o teste da aplicação integrado ao ambiente no qual irá funcionar, ou seja, verifica se todos os elementos combinam de forma adequada e se as funcionalidades e desempenho ao todo são atingidos (PRESSAM, 2006, p. 291). Podem ser subdivididos nos seguintes testes:

- **teste de recuperação:** teste que força a falha na aplicação e verifica se a recuperação é feita adequadamente e em tempo satisfatório;
- **teste de segurança:** verifica se a aplicação possui mecanismos de proteção incorporados para evitar invasão imprópria. À medida que o conteúdo da informação do sistema é mais importante, maior deverá ser sua proteção e mais testes feitos;
- **teste de estresse:** testes que submetem a aplicação em situações anormais. Verificação da estabilidade das aplicações com grandes volumes de acessos, tráfego de dados, entre outros que podem gerar um processamento inadequado;
- **teste de desempenho:** são testes acoplados aos testes de estresse que têm como objetivo medir a utilização de alguma funcionalidade da aplicação. Essa medida, por exemplo, é feita por ciclos de processamento.

2.3.4.4 Teste de aceitação

Segundo Molinari (2003, p.161), o teste de aceitação é voltado à validação do usuário, ou seja, o aceite ou não do usuário para a funcionalidade implementada. Assim, a validação da aplicação significa verificar se aplicação está atendendo aos requisitos definidos pelo usuário.

Segundo Welter do Instituto Lexia (2007, p. 124), o teste de aceitação é composto por duas categorias:

alfa: testes efetuados em ambiente controlado para acompanhar as atividades dos usuários, de forma que os erros sejam registrados no momento em que o cliente esteja utilizando a aplicação;

beta: testes simulados pelos clientes sem controle da empresa desenvolvedora. O usuário encontra-se, nessa fase, livre para fazer simulações de uso reais no sistema.

2.3.4.5 Técnicas de testes no ciclo de desenvolvimento

Resumidamente, as informações dessa subseção, o quadro mostrado, a seguir, sintetizam as técnicas de teste que podem ser usadas em cada etapa do desenvolvimento de *software*.

	Unitário	**Integração**	**Sistemas**	**Aceitação**
Estratégias de testes				
Caixa Preta (funcional)	X	X	X	X
Caixa Branca (estrutural)	X	X		
Etapas do ciclo de vida	Construção (implementação)		Testador	Implantação com cliente

Quadro 8: Estratégias de testes x níveis de testes.
Fonte: Baseado em Molinari (2003).

2.3.5 Métricas de testes de *software*

As métricas são utilizadas para medir a qualidade de um *software*, ou seja, são as características que um teste precisa ter para atingir o mínimo de qualidade desejável. São abordagens estatísticas que controlam o processo de testes de *software*.

Adotar métricas no processo é a forma mais fácil de assegurar a qualidade, pois desta forma podemos definir padrões e verificar a efetividade das tarefas através de medidas. Segundo Pressman (2006, p. 294), as métricas são usadas para abordagem estatística de controle para os processos de testes de *software*.

Para Murillo (2008), os testes são fontes inesgotáveis de métricas, números gerados através de testes, porém esses números de nada servem se não for feita uma análise que transforme esses números em informações que ajudem a alcançar e decidir a melhor decisão ou estratégia de negócio. Essa análise precisa ser feita de forma clara e direta para não gerar dupla interpretação, o que pode ocasionar que os responsáveis não tomem decisões ou que os números sejam apenas dados não interpretados.

> Delimitar objetivos e estratégias é a única maneira de fazer uma leitura precisa sobre os dados gerados pelos testes, aí sim podendo interferir nos processos para melhorar a cadeia de produção. Uma especificação incompleta ou ambígua pode levar a resultados incorretos ou inadequados e então os testes, por mais completos que sejam, não serão capazes de trazer valor real aos negócios. (MURILLO, 2008).

Molinari classifica atributos referentes a métricas (2003, p. 61-62), como:

métricas de "ausência de eficiência": são indicadores de problemas encontrados no processo. Medidas que definem o nível de eficiência da atividade relacionada ao desenvolvimento do *software*, como percentual de defeitos encontrados, percentual de defeitos resolvidos, medidas de cobertura de testes (verificar o quanto de código foi devidamente testado). A análise da ausência de eficiência possibilita avaliar quais mecanismos de detecção de defeitos devem ser revistos e melhorados,

métricas de evolução do *software:* o histórico de indicadores de problemas fornece informações para avaliar como está a eficiência dos testes na etapa atual. Avalia-se se os índices de detecção de defeitos estão de acordo com a qualidade estabelecida. Essa evolução é empregada para auxiliar o gerenciamento e condução de construção de testes para a garantia e melhoria no processo de testes.

Segundo Supreme Quality[17] (2009), nas métricas de “ausência de eficiência” além das medidas mostradas por Molinari, há um atributo de indicação de severidade dos defeitos. Outra métrica utilizada são os números de *help desk* e questionários feitos com os clientes, ou seja, números que indicam a aceitação do produto pelos clientes.

2.4 TESTE DE APLICAÇÃO *WEB* E *BATCH*

Segundo Pressman (2006, p.455), testes em aplicações *WEB* estão relacionadas em descobrir erros no conteúdo, função, usabilidade, navegabilidade, desempenho, capacidade e na segurança da *WEB Application* (*WEB-APP*). Os testes começam nos tópicos que são visíveis ao usuário, seguindo para a tecnologia usada e infra-estrutura da aplicação, ou seja: testes de conteúdo (erros sintáticos, tipográficos e gramaticais), interface, navegabilidade, componente (funções), configuração e desempenho de segurança.

Segundo Rios e Moreira Filho, os testes feitos para aplicações *batch* não satisfazem necessidades da aplicação *web*, devido a alguns cuidados especiais que precisam ser vistos. Por exemplo, o teste de usabilidade deve ser efetuado principalmente para aplicações *web*, e as páginas devem ser avaliadas conforme o seu conteúdo, clareza de linguagem (mensagens de informações, indicadores de valores entre outras).

A seguir apresenta-se uma comparação de testes de uma aplicação *batch* com uma aplicação *web*.

[17] Supreme Quality: empresa que oferece soluções, consultoria e treinamento em qualidade, com foco em testes de *software*. Atua em toda região de Santa Catarina (SUPREME QUALITY, 2009).

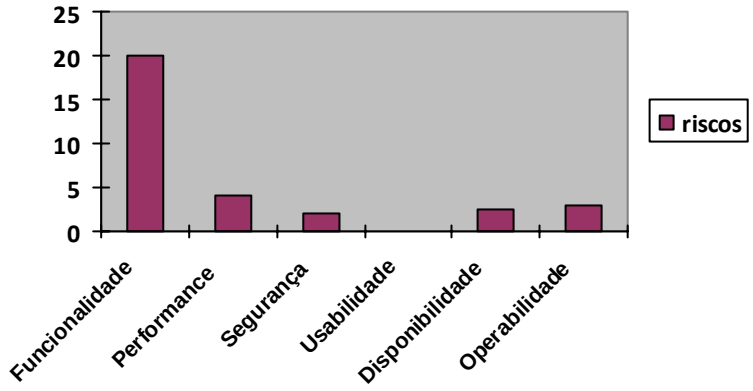


Figura 9: Riscos de teste em uma aplicação *batch*.
Fonte: MOLINARI, 2003, p. 180.

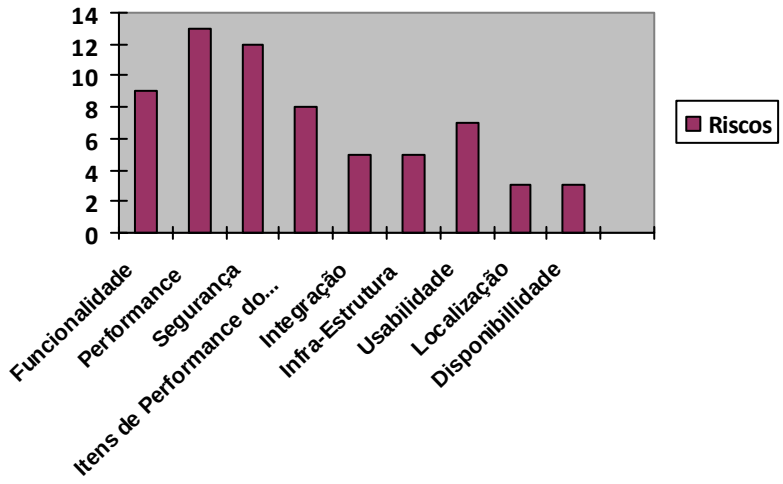


Figura 10: Riscos de testes em uma aplicação *web*.
Baseado: Molinari, 2003, p. 181.

2.5 CASOS DE TESTES

Os casos de testes podem definir, entre outros, os conjuntos de entradas de dados, que são processadas em determinadas condições. Após a execução, os resultados obtidos são comparados com aqueles previstos. O objetivo é verificar se a execução resultou nos dados esperados. As descrições dos casos de testes, também podem, possuir a configuração do ambiente, a hora em que serão simuladas as funcionalidades testadas

(algumas funcionalidades executadas em horários diferentes podem gerar saídas divergentes), assim como também conter a interdependências dos casos de testes.

> Um caso de teste é um documento que descreve uma entrada, ação ou evento e uma resposta esperada, a fim de determinar se uma característica da aplicação está executando corretamente ou não. Um caso de teste deve conter identificação, objetivos, condições de entrada, seqüência de passos e resultados esperados (INTHURN, 2001, p. 78)

A construção de casos de testes pode ser elaborada antes dos casos de usos (desenvolvimento formal) serem enviados para desenvolvimento, assim, podendo identificar e reparar erros bem como prever dificuldades. Dessa forma, é possível acelerar o processo de confecção do *software*, economizar tempo, diminuir custos e garantir que a funcionalidade seja bem sucedida.

2.6 OBTENÇÃO DOS CASOS DE TESTES

Os casos de testes podem ser obtidos através das abordagens por requisitos. No desenvolvimento de *software*, os casos de uso ou documento de análise definem os requisitos de um *software*, os quais são usados para a elaboração de casos de testes.

No desenvolvimento mais formal, como na metodologia *Rational Unified Process* da IBM/*Rational* (RUP), há a utilização de casos de uso (*Use case*) na linguagem de modelo unificada (UML)[18], representados por diagramas. Estes, então, são utilizados para a criação de casos de testes. Os fluxos e cenários usados, nos casos de usos, ajudam a gerar os casos de testes.

Outra forma de construção de casos de teste, por exemplo, nas metodologias ágeis, que não, necessariamente, descrevem os casos de uso, é a utilização de cartões de história (*Use Story*), ou seja, criação de casos de testes através de funcionalidades e comportamentos.

A geração de requisitos de testes através dos requisitos de negócio facilita traçar objetivos capazes de cobrir as necessidades especificadas pelos usuários, permitindo garantir

[18] De acordo com a metodologia RUP um caso de uso descreve inteiramente uma seqüência das ações executadas por um sistema fornecendo resultados observáveis a uma pessoa ou outro sistema (MOLINARI, 2008, p. 139)

a cobertura de todos os principais requisitos que necessitam ser testados. A idéia é manter uma conexão entre o caso de teste e o requisito de negócio ao que se refere (BASTOS, et al., 2007, p. 32).

Os casos de testes podem ser: positivos, isto é, verificar se a funcionalidade faz o que foi requerido (fluxo básico) ou casos de testes negativos, são os testes que agridem o *software*, incluem valores estranhos, testes com memória insuficiente, ou seja, são todos os testes seguem o fluxo alternativo. (HMELJEVSKI, 2009). Esses tipos de casos de testes estão relacionados com o fluxo dos eventos, a seguir, a figura representa os caminhos alternativos, básico e de exceção.

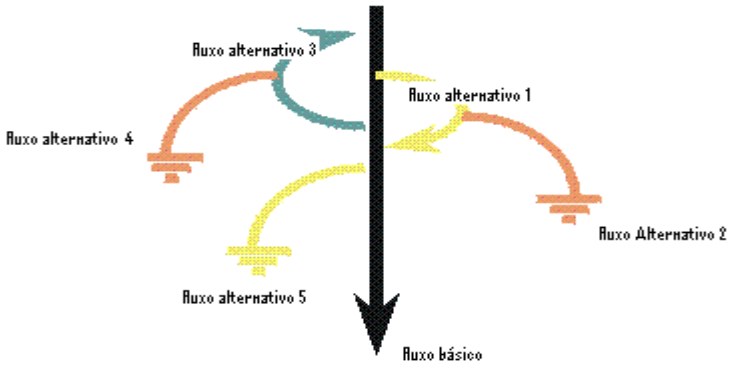


Figura 11: Fluxo de eventos
Fonte: Baseado em Molinari (2008, p. 141)

Cada fluxo de evento pode representar vários cenários de caso de uso, com isto é possível gerar cenários de testes[19] com base nos cenários do caso de uso.

2.6.1 Métodos de elaboração de casos de testes de caixa branca

Nesses métodos é levado em consideração a melhor maneira de executar as estruturas internas de uma aplicação.

[19] Cenário de teste: caminho ou situação a ser testada

A seguir representam-se alguns dos métodos utilizados e seus objetivos específicos.

Método	Descrição
Cobertura de linha de código	Exercitar ao menos uma vez todas as linhas de código-fonte. São medidos pelo número de linhas que são chamadas no teste. Ex: Código fonte é composto por 1000 linhas e ao executar o teste foram chamadas 840 linhas, isto significa que os testes feitos atingiram 84% de cobertura do código
Cobertura de caminhos	Exercitar todos os fluxos alternativos de processamento do código-fonte, assim, identificando falhas como: inicialização de variáveis, fluxos não previstos que podem gerar erros na sua execução.
Cobertura de desvios condicionais	Exercitar condições lógicas no código-fonte. Ex: avaliação de todos os if, then, else... while, next entre outros

Quadro 8: Métodos de elaboração de casos de testes de caixa branca
Fonte: Baseado em Bartié (2002, p. 126 - 134)

2.6.2 Métodos de elaboração de casos de testes de caixa preta

Esses métodos tratam de gerar casos de testes funcionais, conseqüentemente, podem ser aplicados em nível de teste de unidade, de integração, de sistema e teste de aceitação do cliente (MOLINARI, 2008, p. 149).

Método	Descrição do método
Decomposição de requisitos	Primário (situação básica de um requisito), Alternativos (variações de situações dentro de um cenário primário) e Exceção(situações que gerem problemas e inconsistências que impossibilitem a finalização de um requisito)
Análise de documentos Ex: diagramas de atividades estados.	Analisar documentos que detalham o comportamento e regras de negócio. Para testes de interfase, usabilidade, segurança de acesso e desempenho é possível gerar casos de testes, analisando documentações de padrões de interface ou especificação de usabilidade estabelecida pela empresa, o tipo de aplicação entre outros.

Quadro 10: Métodos de elaboração de casos de testes de caixa preta
Fonte: Baseado em Bartié (2002, p. 126 - 134)

2.6.3 Técnicas de refinamento para elaboração de casos de testes

O objetivo destas técnicas é aumentar a extensão de cobertura e ampliar os cenários que representam os casos de testes alternativos e de exceção, identificando os casos de testes que possuam maior probabilidade de apresentar defeitos, eliminando os casos de testes redundantes. Ou seja, com um mínimo de casos de testes, é possível aumentar a eficiência de rastreamento de defeitos com um menor esforço (BARTIÉ, 2002, p. 134).

A seguir representam-se algumas técnicas utilizadas para diminuir a quantidade de casos de testes de maneira eficiente:

Técnica	Descrição
Classes de equivalência (refinamento por partição de equivalência)	Grupos de valores equivalentes, ou seja, um valor de um grupo irá representar todos os demais pertencentes ao grupo. Ex: a participação de lucros de uma empresa só é dada para os funcionários que possuem de 3 até 6 filhos. Os casos de testes a serem gerados, por exemplo, será: 1- Funcionários com 5 filhos 2- Funcionários com 2 filhos 3- Funcionários com 8 filhos
Análise de valores limites	1- É complementar à partição por equivalência, porém nesse caso, são usados os valores fronteiras dos domínios. Seguindo o exemplo anterior da participação de lucros de uma empresa, por exemplo, serão gerados os casos de testes para os valores: 2- Funcionário com 3 filhos 3- Funcionários com 6 filhos 4- Funcionários com 2 filhos 5- Funcionários com 7 filhos
Probabilidade de erro	Valores em branco, nulo, negativos, inválidos.

Quadro 11: Técnicas de refinamento para a elaboração de casos de testes
Fonte: Baseado em Bartié (2002, p. 135 - 137)

2.7 PADRÃO DE QUALIDADE DO CASO DE TESTE

A figura, a seguir, resume as diferentes dimensões que caracterizam a qualidade do caso de teste.

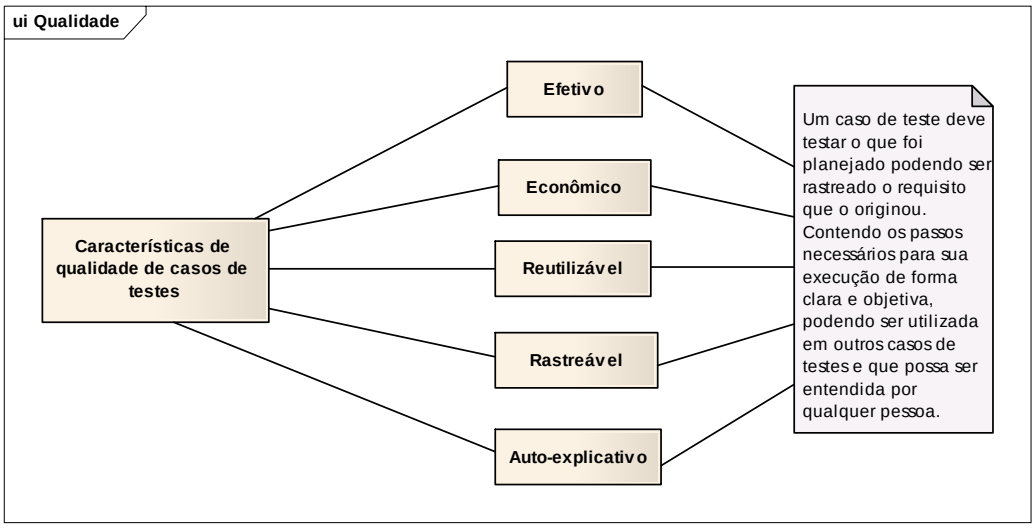


Figura 12 Características de qualidade do caso de teste
Fonte: Baseado em Rios, et al., 2007, p. 154

2.7.1 Desafios de um bom caso de teste

Mesmo com o emprego de técnicas e de normas de confecção de testes, ainda surgem problemas que dificultam a criação de um processo de testes eficiente (Rios, et al., 2007, p. 161).

A possibilidade dos seguintes eventos acontecerem devem ser considerados na elaboração do caso de teste:

a) mudança de requisitos;
b) mudança de cronograma;
c) rotatividade da equipe de testes.

2.8 AUTOMAÇÃO DE TESTES

A automação de testes consiste na utilização de ferramentas de testes que simulam usuários ou atividades humanas de forma a não requerer, ou a minimizar, procedimentos manuais na execução dos testes. (BARTIÉ, 2002, p. 197).

Apesar da tecnologia estar cada vez mais avançada e a complexidade das aplicações aumentando, apenas a execução de testes exploratória é feita na maioria das organizações. A forma exploratória de testes, ainda, é mais rápida no curto prazo, mas, quando houver a necessidade de manutenção no *software*, isso pode ser prejudicial, pois, quando os testes são manuais e uma nova versão é entregue geralmente apenas a nova funcionalidade, ou aquela que foi alterada, é testada. Não é realizada uma bateria de testes completa, como na primeira versão. Essa prática pode trazer outros cenários de erros.

Segundo Myers, quando corrigimos um erro, podemos estar criando um novo erro, por isso a preocupação não deve ser somente com a solução do *bug* encontrado, mas, também, com os efeitos colaterais indesejáveis, os quais podem introduzir um novo erro. (2004, p. 172). Com o teste automatizado há uma maior confiabilidade de que a alteração feita, não afetou outras partes do sistema.

> A automação dos testes é altamente desejada por diversos fatores, inclusive em termos de custos finais. Como esse processo requer um investimento inicial, a automação passa a ser encarada como mais um trabalho a ser realizado. À medida que reexecutamos os testes, o ganho de tempo, controle, confiabilidade e diversas possibilidades existentes com essa tecnologia ficam claros a vantagem inerente a esse processo (BARTIÉ, 2002, p. 63-64).

As ferramentas de automação dos testes possibilitam o desenvolvimento de *scripts* automatizados, de forma a viabilizar um processo de teste com as atividades de entrada e conferência de valores totalmente automatizados (BARTIÉ, 2002, p. 188).

A automação de testes serve para diminuir as horas de execução e ter um resultado dos testes mais cedo. O processo de realizar todos os testes, testes escritos anteriormente e os atuais para verificação de novos *bugs* com a funcionalidade nova, chama-se suíte de regressão. Para Bartié (2002, p. 181), a automação dos testes requer um esforço inicial, porém fornece eficiência, rapidez e confiabilidade que não é possível ser atingida com os procedimentos manuais.

Os testes de automação são agrupados conforme a forma de como os testes interagem com o *software*: baseados na interface gráfica ou baseados na lógica de negócio (CAETANO, 2007, p. 31).

2.8.1 Pilares para automação de testes

As seguintes características são consideradas fundamentais para a automação de testes:

a) ferramentas: apoiam o processo de testes, simulando o processo manual. Possibilitam de criar *scripts* automatizados de forma a indicar os valores de entrada e conferência de dados de saída conforme indicado e esperado;

Para Rios e outros (2007, p. 230), com as ferramentas de automação, é possível um nível maior de produtividade, pois com essas é mais fácil a repetição de todos os testes que já foram executados, podendo ser executados quantas vezes for necessário.

b) metodologia de desenvolvimento e testes: define os procedimentos formais associados aos testes para cada fase do ciclo do desenvolvimento do *software*;

c) infra-estrutura: ambientes em conformidade com o ambiente real em que a aplicação funcionará.

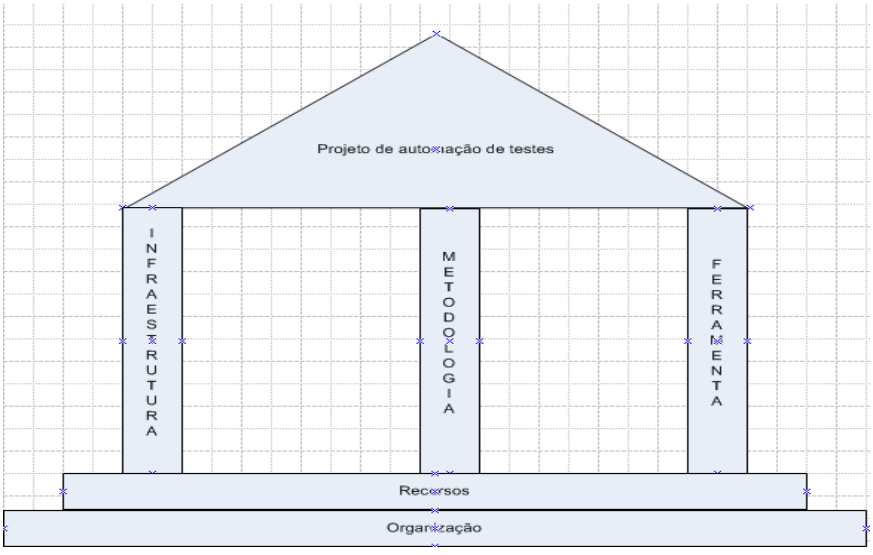


Figura 13: Pilares de automação de testes.
Fonte:RIOS, et al., 2003, p. 156

2.9 FERRAMENTAS DE TESTES

Somente nos anos 1990 é que ferramentas de teste começaram a ser produzidas. Determinados testes que não eram possíveis de serem executados agora poderiam ser feitos através de ferramentas desenhadas com esse objetivo (BARTIÉ, 2002, p. 4). Atualmente, podem-se encontrar inúmeras ferramentas que atendem as várias plataformas de desenvolvimento, objetivos de testes (*batch*, *web*, etc.) e em todas as fases de desenvolvimento de um *software*. Elas são o apoio dos profissionais que agilizam a confecção dos testes, assim, como a criação de massa de dados, simulando a complexidade e o tráfego das aplicações em produção.

"As ferramentas trariam alta produtividade e qualidade no processo de teste. Hoje, entendemos que a aquisição de ferramentas é vital ao sucesso e à viabilização de um trabalho ... – a implantação de um processo de garantia da qualidade de *software*" (BARTIÉ, 2002, p.4). Cada vez mais, a aquisição de uma ferramenta de testes é importante, seja ela para acelerar o processo ou facilitar a codificação dos testes e automatização, levando em conta que a exigência do mercado aumenta a complexidade para chegar a uma qualidade desejável.

> A escolha da ferramenta apropriada é um aspecto importante do processo de testes. As técnicas são poucas e muito abrangentes ao passo que as ferramentas são numerosas e restritas. Cada uma delas oferece diferentes opções, projetada para alcançar um objetivo de teste específico (BASTOS, MOREIRA, 2007, p. 87)

Vivemos em uma era que a tecnologia nos fornece ferramentas para aprimorar a própria tecnologia. No ciclo de vida de um *software*, existem várias ferramentas que nos apóiam para prover agilidade e qualidade.

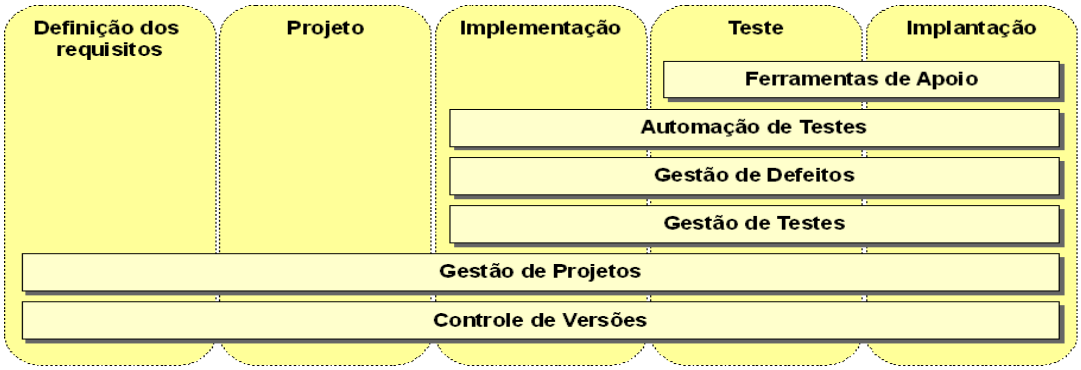


Figura 14: Processo de teste com ferramentas
Fonte: CAETANO (2007, p. 12)

Segundo Molinari (2003, p. 109), algumas características importantes precisam ser observadas na escolha das ferramentas, como:

Compatibilidade	• sistema operacional; • ambiente de criação das aplicações; • *software* de terceiros, utilizados pela aplicação em testes.
Audiência com a ferramenta (investigação sobre a ferramenta)	• quem irá usar a ferramenta? • necessita de treinamento? • fácil utilização?
Regra financeira	• quanto pode-se gastar?
Requerimentos de negócio	• é possível testar a aplicação em questão? • atende a confecção dos testes funcionais da aplicação?

Quadro 12: Características a serem observadas para escolha de ferramenta.
Fonte: MOLINARI (2003, p. 109).

A seguir, apresenta-se um resumo de ferramentas de *software* livre, podem ser utilizadas no processo de desenvolvimento.

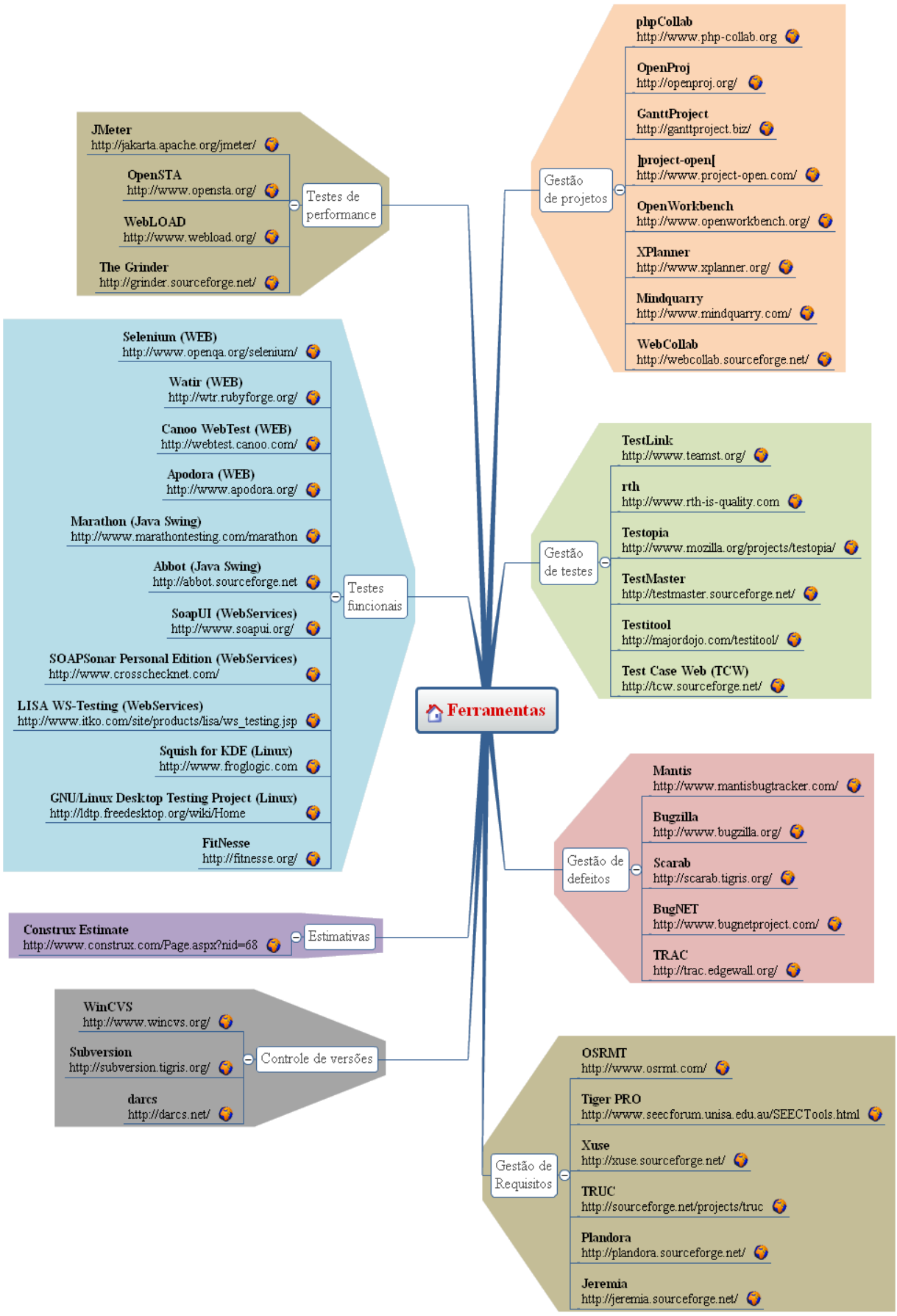


Figura 15: Ferramentas utilizadas no processo de testes e desenvolvimento.
Fonte: Caetano, 2007, p. 171.

2.10 **AMBIENTE PARA OS TESTES**

A criação de ambientes para a realização dos testes está diretamente relacionada com a estratégia que será adotada para a execução dos testes. Esse ambiente vai além de configuração de *hardware*, pois nesta etapa deve ser planejada toda a estrutura necessária a cada técnica de teste que será executada, como a massa de dados, configuração de sistema operacional e *browsers* entre outros (RIOS, 2007, p. 77).

À medida que o nível (estágios) de teste é maior, o ambiente de teste deve ser mais próximo ao ambiente de produção (BARTIÉ, 2002, p. 165). O quadro a seguir, representa a distribuição do ambiente por nível de testes

Atributos	**Nível ou estágio**			
	Unitário	Integração	Sistema	Aceitação
Ambiente	Desenvolvimento	Desenvolvimento	Testes	Testes/Produção

Quadro 13: Distribuição de ambiente por níveis do teste.
Fonte: Baseado RIOS et al, 2007, p.79.

Para que os testes alcancem os melhores resultados, encontrando grande parte dos problemas que poderão ocorrer em produção, o ambiente de teste deve ser uma simulação bem próxima ao de produção. Isso envolve configuração de navegador, sistema operacional, banco de dados e rede, entre outras configurações.

Quando o *software* é testado em ambiente de desenvolvimento, os resultados poderão ser influenciados pelas configurações do meio em que é desenvolvido. Assim, é preciso testar a aplicação em um ambiente similar ao que a aplicação enfrentará em produção, podendo simular e verificar erros reais (erros idênticos aos que ocorreriam no cliente), evitando surpresas de divergências de ambientes, quando a aplicação é liberada para o cliente.

2.11 DOCUMENTAÇÃO DE TESTES DE *SOFTWARE*

De forma simples, a documentação de testes é representada por dois grupos de documentos: Descrição dos testes do *software* e relatórios dos testes do *software*. Cada uma dessas classes, por sua vez, pode ser subdividida em diversos documentos como apresentado no quadro 3 da seção 2.2.2.

Os documentos de testes de *software* são necessários para formalizar o processo de testes que possibilitam acompanhar a evolução e rastrear os testes realizados e em que condições foram executadas.

3 MÉTODO

Este capítulo apresenta o planejamento, a caracterização da pesquisa, descrição das etapas da proposta e as delimitações.

3.1 METODOLOGIA CIENTÍFICA APLICADA

Para a definição e aplicação da metodologia, há a necessidade de uso de ferramentas e caminhos, procedimentos já traçados que nos guiam para atingir o objetivo. O método apresenta processos de construção de conhecimento, orientando as decisões nas investigações exploratórias (conhecimento que geram outros conhecimentos), assim, ajudando a resolver problemas e tomadas de decisões.

Metodologia, segundo Molinari, são técnicas, métodos e estratégias voltadas para o desenvolvimento ou aplicação de algo, que, geralmente, pode ser um produto ou serviço (2004, p. 114). São passo-a-passo de uma trajetória que já foi homologada e, por consequencia, levada a um resultado satisfatório.

“Muitos estudiosos da Metodologia Científica procuram entender a ciência por ângulos diferentes, mas todos caminham para um único objetivo que é a demonstração da verdade por meio da observação e da experimentação”. (OLIVEIRA, 1998, p. 49).

O objetivo desta monografia é desenvolver conhecimentos científicos para uma proposta de processos de testes automatizados em uma empresa de desenvolvimento de *software*.

3.2 CARACTERIZAÇÃO DO TIPO DE PESQUISA

Pesquisa é uma forma de revelar respostas para problemas de forma sistemática com procedimentos científicos, identificando referências bibliográficas que possam auxiliar na construção do marco teórico, na análise do problema e na obtenção de solução.

"A pesquisa é fundamentada e metodologicamente construída, objetivando a resolução ou o esclarecimento de um problema. O problema é o ponto de partida da pesquisa. A formulação da resolução dependerá do desenvolvimento da pesquisa". (SILVA et al., 2005). O problema nos leva à pesquisa, à procura de uma resposta e à possível solução pra trazer benefícios às empresas que desejam sobreviver no competitivo e inovador mercado da área de confecção de *software*.

Metodológica porque o conhecimento adquirido é através de regras, processos técnicos, com idéias ordenadas, logicamente, que foram comprovadas em experiências anteriores.

Essa atividade não é cópia de informações encontradas nos livros, *internet* ou revistas, é a reunião de informações analisadas, interpretadas, comparadas que exige reflexão para a confecção da monografia. (CERVO, 2002, p. 186).

A natureza desta pesquisa classifica-se como pesquisa aplicada, segundo Silva (2005), pois possui o objetivo de gerar conhecimentos para utilizá-los de forma prática na solução de um problema.

A forma de abordagem da pesquisa desta monografia será uma mistura de quantitativa com relação aos números gerados no relatório final de cobertura de testes e será qualitativa com relação ao estudo e à análise do mundo real, do qual serão capturados os dados de informações, atributos, relações e processos para análise, os que não são possíveis de representação por números, apresentando, assim, uma forma ordenada, lógica e conclusiva (SILVA et al. 2005).

Pelo fato do tema da pesquisa ser um assunto bastante discutido, atualmente, foi possível obter informações recentes de artigos disponibilizados na internet e revistas, assim, como fóruns de discussão e, principalmente, de livros. Desta forma, a pesquisa, também pode ser classificada como uma pesquisa bibliográfica.

3.3 ETAPAS METODOLÓGICAS

As etapas metodológicas são passos dados para a realização da pesquisa. São orientações para a execução desta monografia.

O fluxograma, a seguir, demonstra as etapas que foram seguidas para a obtenção de conhecimento e para a resolução da proposta de solução para o problema do estudo de caso, contido neste documento.

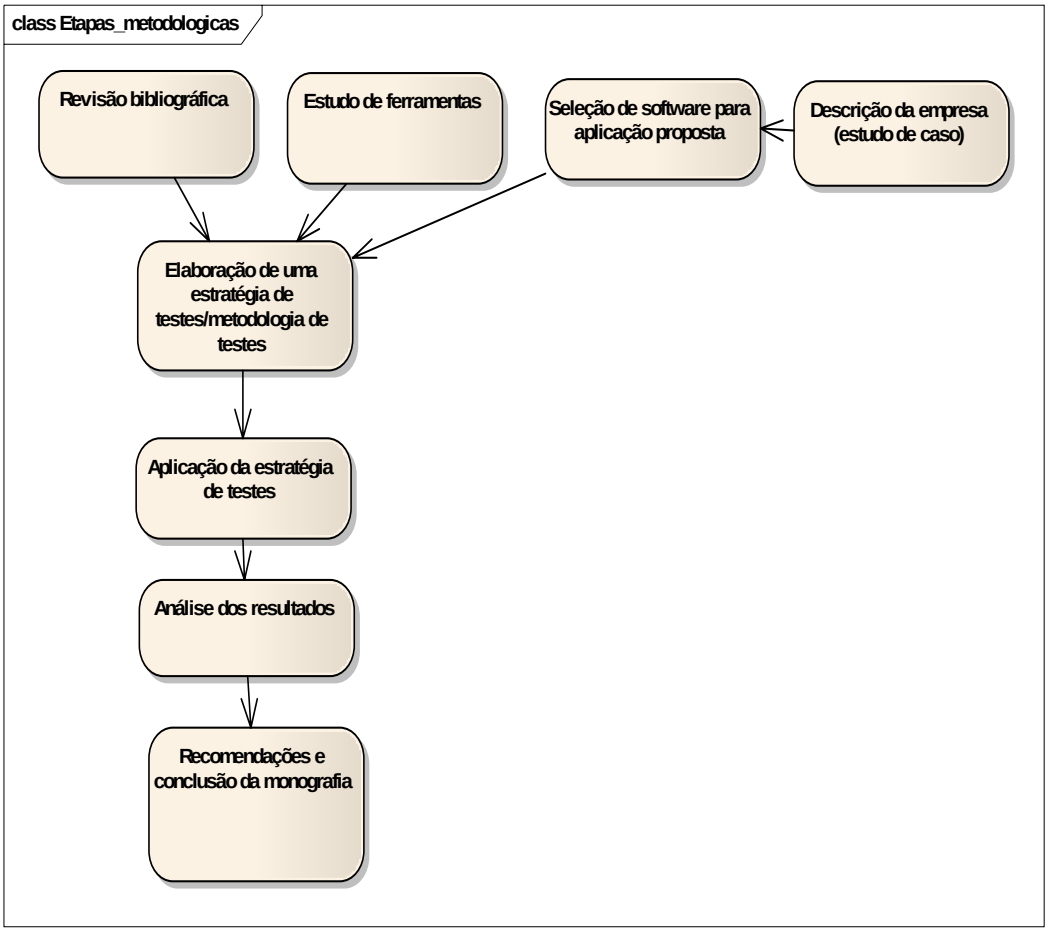


Figura 16: Etapas Metodológicas.
Fonte: Elaboração da autora, 2009.

3.4 DELIMITAÇÕES

O objeto de estudo da monografia será a confecção de testes automatizados e sua aplicação, não será desenvolvida uma ferramenta de testes. Os planos e implantação dos testes deverão se adequar ao conjunto de ferramentas open source existente hoje no mercado e aos requisitos existentes na empresa.

A metodologia será desenvolvida em uma aplicação da organização de confecção de *software.* Não será aplicada em todos os produtos que ela oferece, devido à grande quantidade de aplicações existentes. Além do estudo dos processos de testes, serão efetuados planejamentos, considerando características específicas da aplicação. Esses processos poderão ser usados, posteriormente, pela empresa em outros projetos.

Os planos de testes serão confeccionados sem exposição do código fonte e informações estratégicas, assim como os dados dos clientes para a preservação da empresa.

4 DESCRIÇÃO DA EMPRESA E PRODUTO ESTUDO DE CASO

Este estudo de caso tem como objetivo adotar uma prática pesquisada para melhorar o processo de teste de uma empresa que confecciona aplicações para o atual mercado competitivo.

Para a realização deste trabalho, foi selecionada uma empresa de desenvolvimento de *software* da região e um dos seus projetos para demonstração de conhecimento adquirido na revisão bibliográfica e aplicação da proposta.

4.1 EMPRESA DE DESENVOLVIMENTO DE *SOFTWARE* SELECIONADA

A Nexxera é uma empresa que provê soluções estratégicas e operacionais para conexão com bancos, fornecedores e clientes, proporcionando integração e visibilidade da cadeia de negócios, gestão de riscos de crédito, controle do risco operacional, mais rentabilidade financeira.

O *software* é oferecido ao cliente, como serviços, quem utiliza as funcionalidades oferecidas não precisa fazer grandes investimentos em infra-estrutura de Tecnologia da Informação (TI), sua forma de cobrança é apenas pela utilização dos *software* com o tráfego de dados.

A empresa se mantém competitiva no mercado, porque anda paralela à maturidade do processo e ao ciclo de desenvolvimento do *software*. Os papéis e responsabilidades existentes procuram obter um maior controle do que é feito, com manutenções pela equipe de engenharia de *software*. A empresa possui hoje uma célula de Suporte de terceiro nível[20] em desenvolvimento na área de Engenharia de *Software*, juntamente com equipe de desenvolvedores, analistas de requisitos e produtos. A empresa ampliou o processo de avaliação dos produtos para atingir vantagens de controle, facilidade, agilidade e gerenciamento dos projetos e requisitos.

[20] Suporte de terceiro nível: responsável pela investigação, diagnóstico, rastreamento e dos incidentes. O suporte de segundo nível é a equipe que primeiro entra em contato com o defeito reportado pelo cliente, caso não consiga solucionar o problema é enviado para o suporte de terceiro nível.

As aplicações estão sempre em aprimoramento (utilização de novas ferramentas, novas funcionalidades, otimização de código e execução) o que, automaticamente eleva os riscos de erros, com isso, a necessidade constante de validações sobre o *software* ficou indispensável, porque a cada funcionalidade adicionada outros erros poderiam aparecer em outras já testadas. Esses aprimoramentos nos *software* se dão devido à flexibilidade de que a empresa possui para se adequar aos seus clientes, sempre procurando melhorar os seus processos. A necessidade de atualização é indispensável, porém sempre existe uma probabilidade de problemas.

Essa nova célula do setor de engenharia de *software* tem a funcionalidade de automatizar os processos de testes existentes e brindar uma cobertura maior de testes, procurando a qualidade e a segurança máxima do que é oferecido no mercado.

4.2 PROPOSTA DE DESENVOLVIMENTO

Para um melhor aproveitamento e elaboração do planejamento de teste, faz-se necessário o conhecimento de algumas metodologias, ferramentas e processos existentes no mercado e dentro da empresa. Este estudo tem como objetivo motivador a formação da base científica e metodológica para a confecção de um processo e um planejamento de testes adequados, aplicação do mesmo e observação de evolução da aplicação selecionada. Serão analisados os processos, métodos, padrões, conceitos e melhores práticas juntamente com as ferramentas mais utilizadas no mercado.

Com o objetivo de melhorar os processos de qualidade de *software*, uma aplicação específica será escolhida para a aplicação da metodologia, com a realização de planejamento dos processos de testes, execução e acompanhamento dos mesmos.

4.3 DESCRIÇÃO DA APLICAÇÃO

O projeto que será testado faz parte de uma gestão avançada de negócios que integra um ambiente eletrônico a uma cadeia de valor (bancos, clientes, fornecedores, etc).

4.3.1 Pagamento Eletrônico

O Pagamento Eletrônico permite o gerenciamento completo, com mobilidade, dos processos e compromissos financeiros automaticamente. Fornece uma interface com o usuário para administração de contas a pagar, visualizando as transações de pagamento e a liquidação dos compromissos. O pagamento eletrônico possui a flexibilidade, a economia e a vantagem de permitir aos usuários utilizarem o *layout* de arquivo que desejar, assim, podendo efetivar o pagamento em qualquer banco, mesmo que este possua um *layout* diferente ao banco do cliente.

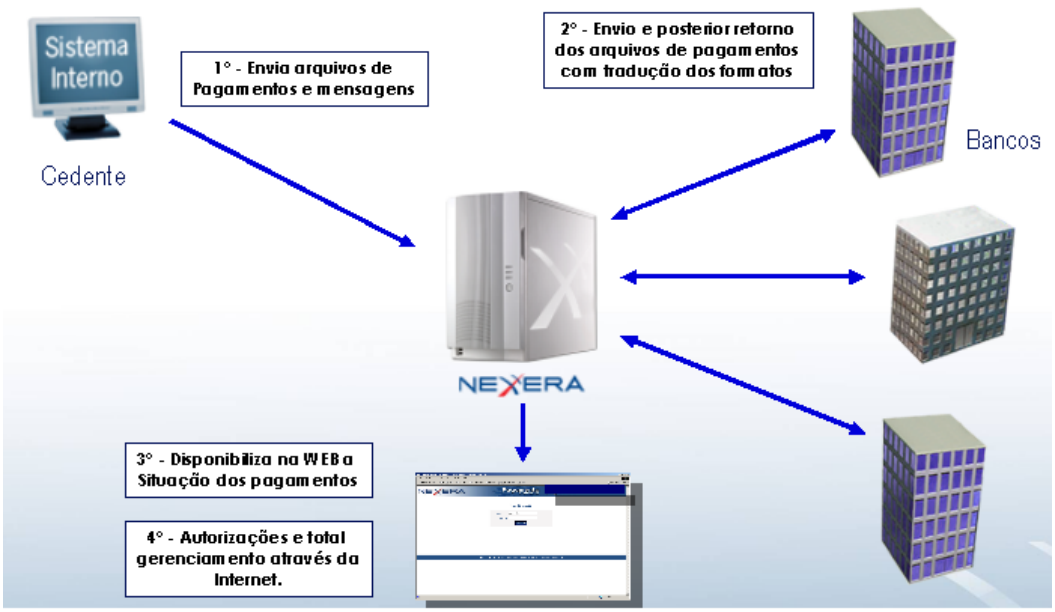


Figura 17: Fluxo do Pagamento Eletrônico.
Fonte: Baseado Nexxera (2009).

4.3.1.1 Problemas do Pagamento Eletrônico

A maioria dos problemas encontrados nesse *software* legado[21] é decorrente de erros funcionais regressivos, ou seja, após uma atualização de funcionalidades que executavam corretamente e deixam de responder adequadamente, decorrentes da alteração anterior, seja ela uma nova funcionalidade agregada ao *software* ou correção de algum *bug*.

Esses problemas não chegaram a provocar sérios prejuízos, porém a busca da credibilidade e de confiança do cliente no produto é essencial e faz parte da missão e objetivos da empresa.

4.3.2 Tecnologias utilizadas para implementação do *software*

Algumas tecnologias utilizadas para a confecção da aplicação, Pagamento Eletrônico foram:

a) Hibernate: Ferramenta de mapeamento objeto/relacional para Java (LINHARES, 2005, p. 1);
b) Struts: é um *framework open source* do projeto *Jakarta*[22] que auxilia a construção de aplicações para a *web* (APACHE, 2009);
c) Java: Linguagem de programação;
d) Oracle: Ferramenta cliente/servidor para gestão de base de dados (ALVAREZ, 2004).

A seguir a estrutura tecnológica das aplicações:

[21] *Software* Legado: *software* que se encontra em produção, já em uso.

[22] Jakarta: projeto que oferece *frameworks open-source* para desenvolvimento de aplicações. Os produtos Jakarta são licenciados pela Apache.

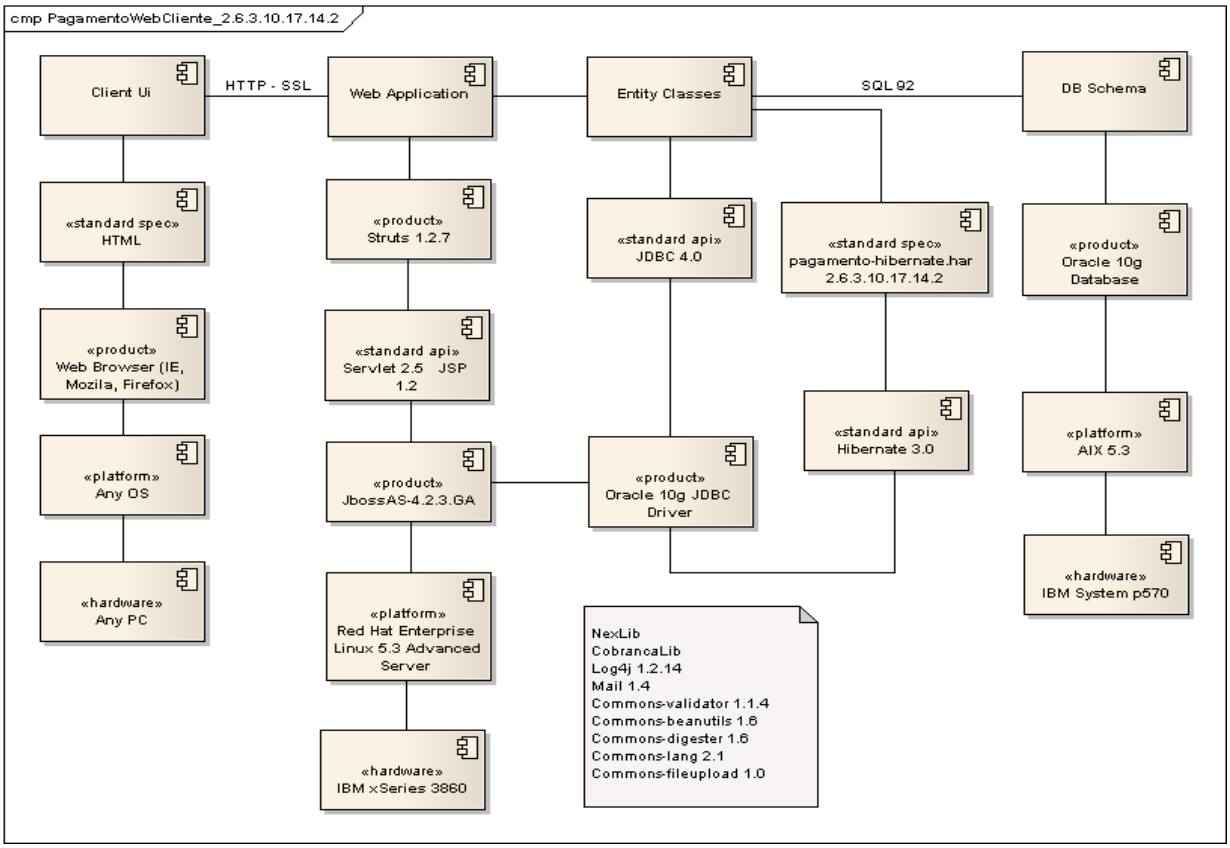


Figura 18: Pagamento Eletrônico - aplicação *Web*
Fonte: Elaboração da autora, 2009

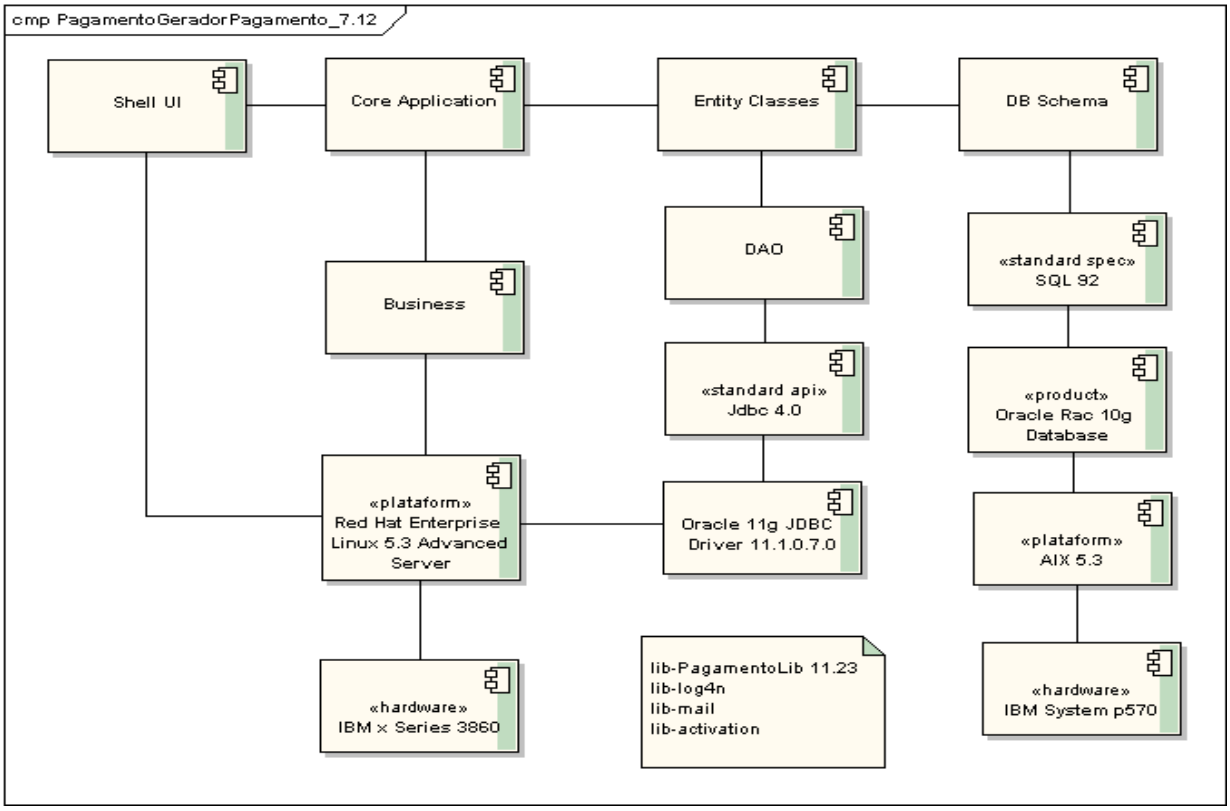


Figura 19: Pagamento Eletrônico – aplicação *batch* (gerador de pagamentos) - todas as aplicações *batch* segue a mesma estrutura tecnológica.
Fonte: Elaboração da autora, 2009

4.4 CONSIDERAÇÕES SOBRE O CAPÍTULO

Este capítulo apresenta a empresa na qual a pesquisa será desenvolvida e o produto de *software* para o qual será elaborado o processo de testes. Esse processo e a sua aplicação serão descritos a seguir

5 ELABORAÇÃO DE PLANEJAMENTO DE TESTES

Finalmente, neste capítulo, será colocado em prática todo o conhecimento adquirido no decorrer da monografia. Nesta fase será apresentado o processo de teste definido, assim como a sua aplicação no produto escolhido, com apóio e ferramentas *open source*.

O objetivo desta etapa é o levantamento e adaptação de todos os artefatos necessários para a elaboração do projeto de testes, assim como a definição da infra-estrutura, responsabilidades das equipes, avaliação inicial de riscos do produto e qualquer outro evento relevante para a sua confecção. Com todos os requisitos expostos será possível elaborar os roteiros de testes e chegar a sua efetivação, chegando à etapa de entrega de um projeto de testes.

O processo de planejamento e elaboração do projeto de teste foi a forma encontrada de padronizar os trabalhos a serem executados, modelando as etapas de controle e servindo de guia para a confecção dos projetos de teste, podendo, assim, também ser ampliado a toda a organização para projetos futuros na empresa.

A metodologia adotada para documentação de testes será baseada no contexto das normas ISO e modelo CMMI, a IEEE -829: 1998 apresentadas no capítulo 2.2.2, tratando da documentação de cada etapa do processo de testes. A confecção dos documentos foi apoiada pelas ferramentas que serão apresentadas na seção 5.3.3.

5.1 REPRESENTAÇÃO GRÁFICA

O produto a ser testado encontra-se em produção, então, houve a necessidade de adaptar o processo proposto nos livros para essa realidade de *software* legado. No processo de teste ideal os testes começam no início do projeto de *software*, no estudo de caso realizado o *software* já está em utilização.

O detalhamento das atividades para a geração de processo de testes será representado na figura, a seguir:

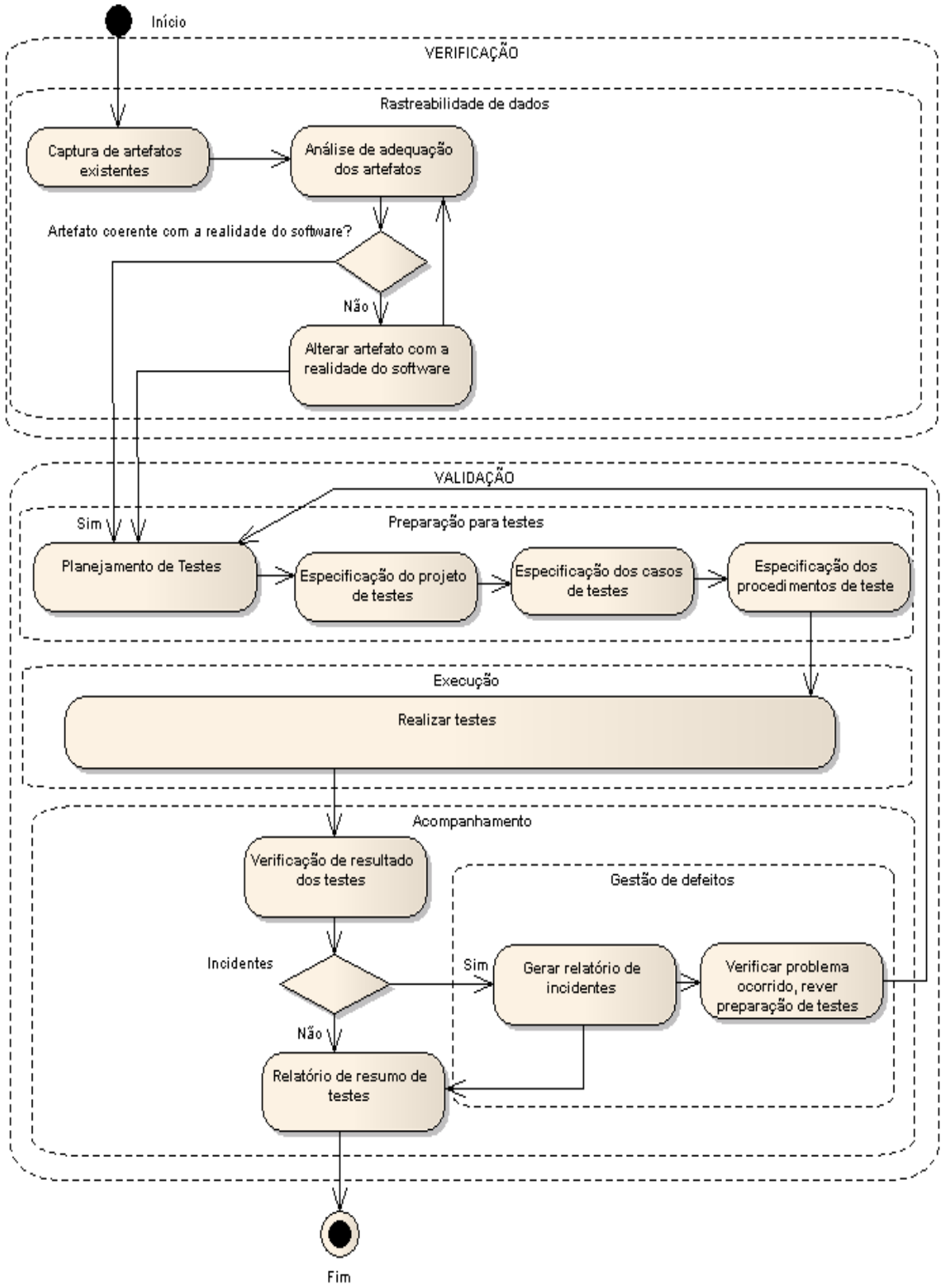


Tabela 20: Representação de atividades para a geração de processo de testes na empresa
Fonte: Elaboração da autora, 2009

Como pode-se observar na figura, o processo foi dividido em quatro etapas chamadas de:

- rastreabilidade de dados;
- preparação de testes;
- execução;
- acompanhamento.

A descrição de cada uma dessas etapas será apresentada, a seguir.

5.2 RASTREABILIDADE DE DADOS

Foram reunidos todos os artefatos existentes para a elaboração do projeto de testes, tendo em vista que o *software* já está em produção e o projeto deveria andar paralelo ao desenvolvimento. Alguns documentos encontravam-se desatualizados conforme as mudanças de regras de negócio e de usabilidade implementadas para atender ao cliente, então foi necessária a execução das funcionalidades no produto para validar os documentos.

Os seguintes artefatos foram utilizados:

Artefato	**Descrição**
Documento de requisitos	Especificação dos requisitos das funcionalidades do sistema;
Diagrama de sequencia	Diagramas, contendo o fluxo das funcionalidades (artefato não encontrado para todo o sistema);
Diagrama de estado	Visualização do estado do objeto, após cada execução de funcionalidade (artefato não encontrado para todo o sistema);

Roteiro técnico de ocorrência (RTO) ou incidência problemática	Documento, contendo a descrição do erro ocorrido no ambiente de produção. Solução paliativa adotada e plano de ação para solução definitiva do problema e precauções necessárias para que o erro não ocorra novamente (RTOs registradas e gerenciadas a partir de março de 2009).
Roteiro de atualização (RA)	Roteiro de atualização de produto (para cada funcionalidade ou *bug* resolvido é criado um roteiro de atualização e descrição da funcionalidade adicionada ou bug resolvido) As RAs registradas e gerenciadas a partir de março de 2009.
Check List do suporte (planilhas com funcionalidades a serem testadas, após atualização de versão em ambiente de produção)	Resultado do *check list* das funcionalidades, atualizadas no produto de cada versão.
Configuração de servidores	Contém dados de configuração dos ambientes de teste e produção.
Layout padrão Nexxera	*Layout* padrão utilizado pela Nexxera nos sistemas desenvolvidos internamente.
Metodologia de desenvolvimento de banco de dados	Regras de nomenclatura e ferramentas utilizadas na administração de banco de dados.

Quadro 14: Artefatos analisados
Fonte: Elaboração da autora, 2009

5.3 INFORMAÇÕES PARA PREPARAÇÃO DOS TESTES

A arquitetura estrutural interna do sistema atual impossibilita a confecção de alguns tipos de testes. Não serão feitos no momento, por exemplo, os testes unitários. Com a reestruturação planejada e amadurecimento de equipes treinadas para as atividades referentes à qualidade de *software* futuramente serão integradas cada vez mais etapas nos processos de testes.

5.3.1 Processo para a escolha da estratégia de validação

A complexidade do produto Pagamento Eletrônico, requer uma avaliação clara dos riscos envolvidos para o negócio da empresa. Considerando um risco, a incerteza do alcance de um objetivo, o efeito do risco pode ser considerado positivo ou negativo.

Um risco positivo é quando uma ameaça pode ser transformada em oportunidade e o risco negativo como conseqüência, os defeitos. Para o gerenciamento e controle de problemas ocorridos no *software*, as atividades foram coordenadas, para a análise de riscos negativos e sua priorização.

5.3.1.1 Identificar e analisar riscos para o negócio

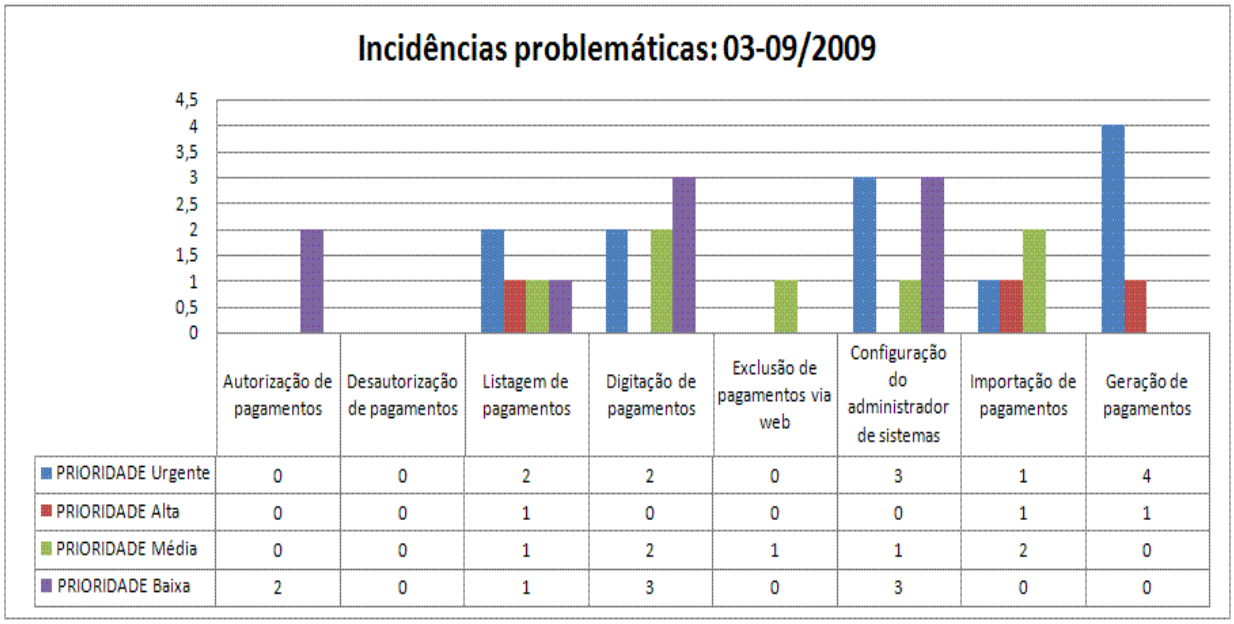
Por se tratar de um *software* legado e pelo fato da empresa estar registrando os defeitos encontrados no sistema através de RTO, ou Incidência problemática, desde março de 2009 pode-se, através destes documentos, fazer uma análise quantitativa dos riscos que a aplicação enfrenta. A seguir pode-se agrupá-los de forma que a estratégia de negócio da empresa classifique os riscos negativos, assim, priorizando os casos de testes que serão elaborados primeiro, direcionando o esforço e diminuindo o impacto dos defeitos.

A classificação dos riscos, a seguir, está relacionada à dependência que a empresa tem sobre a aplicação, ou seja: se houver a interrupção do processamento da funcionalidade, a empresa irá deixar de receber o dinheiro pelo serviço? A empresa terá de alguma forma multas pelo atraso do processamento?

Riscos (negativo)	**Descrição**
Urgente	Problema ocorrido em produção que está impedindo o uso da aplicação.
Alta	Prioridade alta na resolução do problema na aplicação. Funcionalidade possui alternativa paliativa para a realização de atividade.
Média	Verificação de erro no sistema, mas não detectado pelo cliente
Baixa	Verificação de melhoria no sistema para a próxima versão.

Quadro 15: Classificação dos riscos
Fonte: Elaboração da autora, 2009

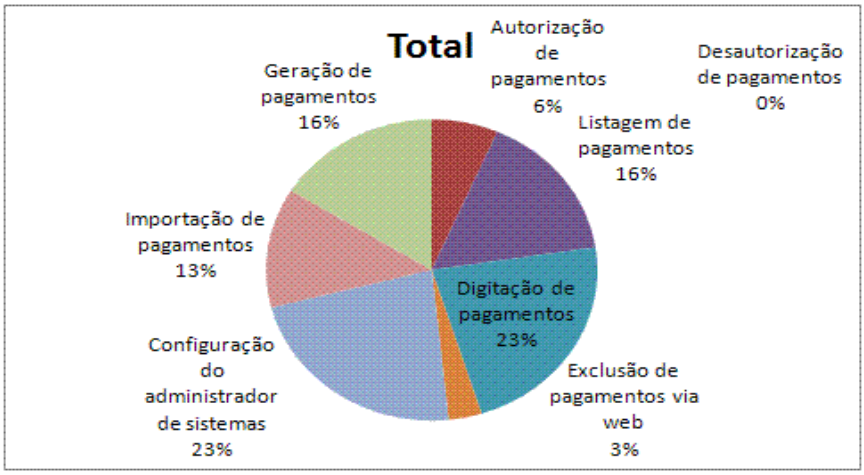
A seguir apresenta-se a listagem de problemas ocorridos e registrados no período de março até setembro de 2009:



	Autorização de pagamentos	Desautorização de pagamentos	Listagem de pagamentos	Digitação de pagamentos	Exclusão de pagamentos via web	Configuração do administrador de sistemas	Importação de pagamentos	Geração de pagamentos
PRIORIDADE Urgente	0	0	2	2	0	3	1	4
PRIORIDADE Alta	0	0	1	0	0	0	1	1
PRIORIDADE Média	0	0	1	2	1	1	2	0
PRIORIDADE Baixa	2	0	1	3	0	3	0	0

Figura 21: Incidências problemáticas do mês de março até setembro de 2009
Fonte: Elaboração da autora, 2009

Através dos riscos ocorridos, no período de março a setembro, pode-se definir a sua magnitude, assim, indicamos onde estão os maiores problemas ocorridos no projeto do Pagamento Eletrônico.



Quadro 22: Percentual de erros ocorridos no Pagamento Eletrônico
Fonte: Elaboração da autora, 2009

Apesar do percentual de incidências serem maior na digitação de pagamentos juntamente com a configuração do administrador de sistemas, os problemas que ocorreram

nessas funcionalidades, possuem riscos negativos com prioridades menores que a geração de pagamentos, o qual ocorre com uma freqüência de 16%. A maioria dos erros ocorridos no processamento de geração de pagamentos é de nível urgente, fazendo com que a severidade destes erros seja maior que os demais. Essa situação pode se verificar melhor considerando um índice de severidade para cada risco negativo, como apresentado nas ilustrações a seguir.

Riscos(negativos)	Severidade
Urgente	4
Alta	3
Média	2
Baixa	1

Quadro 16: Severidade de cada risco negativo
Fonte: Elaboração da autora, 2009

A seguir é apresentada a fórmula utilizada para gerar o grau de risco (severidade):

Grau de Risco = (Quant. Defeitos urgentes * 4) + (Quant. Defeitos alta * 3) +
(Quant. Defeitos média * 2) +(Quant. Defeitos baixa * 1)

A seguir o gráfico de defeitos, associados a sua respectiva severidade, encontrados no projeto do Pagamento Eletrônico.

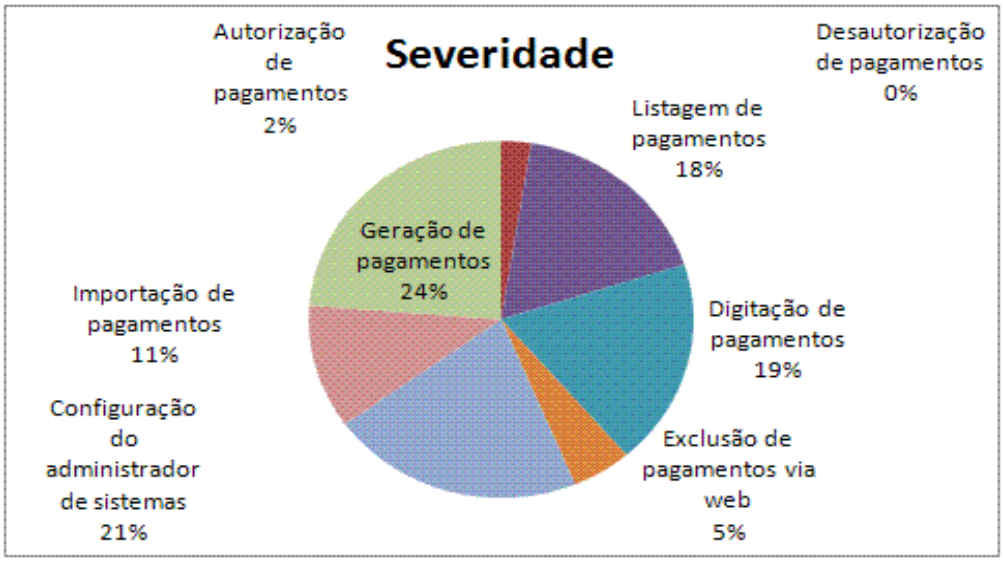


Figura 23: Grau dos riscos do Pagamento Eletrônico entre 3 a 9 de 2009
Fonte: Elaboração da autora, 2009

5.3.1.2 Causas das incidências problemáticas

Através do rastreamento das incidências problemáticas, foram levantadas as seguintes causas para os problemas ocorridos na geração de pagamentos:

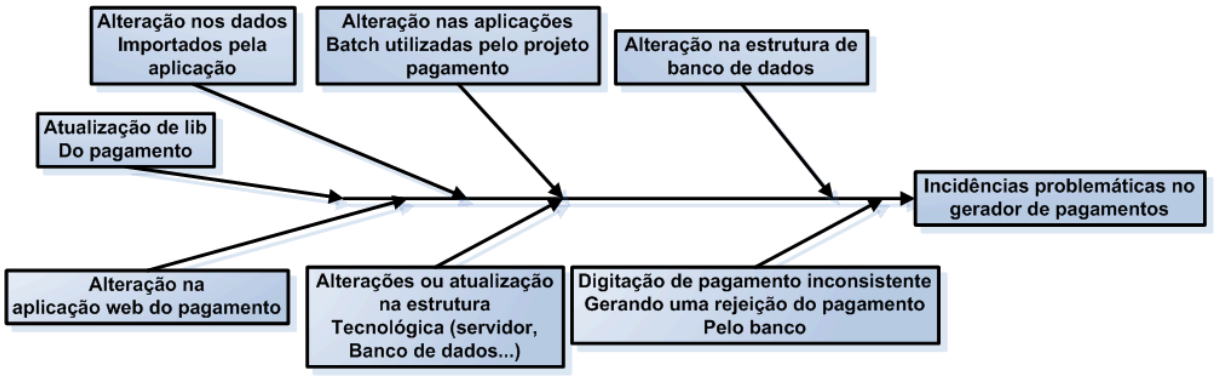


Figura 24: Gráfico de causa e efeito (espinha de peixe) – busca da causa
Fonte: Elaboração da autora, 2009

A maioria dos problemas, ocorridos na geração de pagamentos para o cliente ou banco, foi decorrente de outras alterações ou atualização dos outros produtos de Pagamento Eletrônico. Ao atualizar, por exemplo, a estrutura de banco de dados, para atender uma nova demanda no importador de pagamentos, afetou-se de alguma forma a geração de pagamentos, de forma similar aconteceu com outras atualizações efetuadas na *web* e na digitação de pagamentos, entre outros.

A alteração das demandas fez com que o serviço funcionasse perfeitamente na parte da aplicação alterada, porém, por não haver um teste geral de todos os produtos do Pagamento Eletrônico e sem teste de regressão automatizado, corre-se o risco de acontecer algum problema no fluxo normal da aplicação. Deve se ressaltar que o teste geral em todos os produtos do Pagamento Eletrônico não tem sido realizado por falta de tempo, pois isso levaria no mínimo 2 meses para cobrir todos os casos, se realizado de forma manual.

A cada versão nova de algum produto do Pagamento Eletrônico em produção, é feito um monitoramento deste ambiente pelo suporte de terceiro nível que, também, realiza um *check-list* das funcionalidades alteradas, mas estes procedimentos só apontam os problemas, quando os riscos são materializados.

O objetivo é o rastreamento dos riscos, antes que se tornem reais em ambiente de produção, assim, diminuindo as incidências problemáticas, aumentando a estabilidade e confiança nos produtos do Pagamento Eletrônico.

5.3.1.3 Priorização de casos de testes

Após análise de gráficos gerados pelas incidências problemáticas ocorridas nestes seis meses a priorização dos casos de testes foi elaborada com a seguinte sequencia:

Ordem dos casos de testes	**Funcionalidades estratégicas do Pagamento Eletrônico**	**Severidade**
1	Geração de pagamentos	24%
2	Configuração do administrador de sistemas	21%
3	Digitação de pagamentos	19%
4	Listagem de pagamentos	18%
5	Importação de pagamentos	11%
6	Exclusão de pagamentos via *web*	5%
7	Desautorização de pagamentos	0%

Quadro 17: Ordem de prioridade dos casos de testes.
Fonte: Elaboração da autora, 2009

5.3.2 Estratégia adotada

Para que a ameaça de novas incidências não venha a se tornar um risco negativo de perdas para empresa, surgiu a necessidade de criação de um mecanismo de controle. De forma que, este venha a controlar melhor o surgimento de erros na aplicação, após alguma manutenção, inclusão de novas funcionalidades ou até mesmo descoberta de erros já existentes na aplicação.

A seguir, a estratégia de validação, que será utilizada no processo de testes proposto é apresentada graficamente.

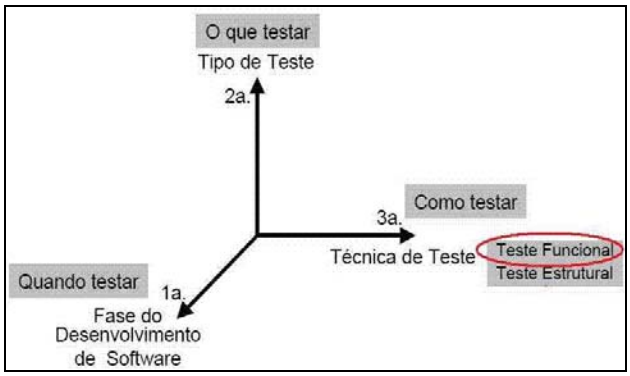


Figura 25: Dimensões dos testes com a técnica de teste escolhida.
Fonte: RIOS et al, 2007, p.141.

	Estratégia	Nível de teste	Tipo de teste	Cobertura
Processo de testes (Pagamento Eletrônico)	Caixa preta	Integração	Teste Funcional	Funções do sistema que apresentam maior risco negativo para o negócio da empresa.

Quadro 18: Estratégia de validação
Fonte: Elaboração da autora, 2009

A razão da escolha do tipo de teste funcional é porque, hoje, a preocupação maior são os resultados do processo e não como ele foi implementado, pelo fato de ser um *software* legado e os riscos negativos que geraram as incidências problemáticas afetaram, principalmente, uma das características de qualidade: funcionalidade. Hoje, o principal objetivo é fazer com que as funcionalidades continuem processando de forma correta.

Testes	Objetivo
Requisitos	As funcionalidades da aplicação seguem os requisitos especificados?
Regressão	O que foi modificado na aplicação pode trazer algum risco? Irá gerar algum problema?

Quadro 19: Testes funcionais que serão feitos no projeto do pagamento.
Fonte: Elaboração da autora, 2009

5.3.3 Ferramentas

A seguir as ferramentas de apoio distribuídas no processo de testes:

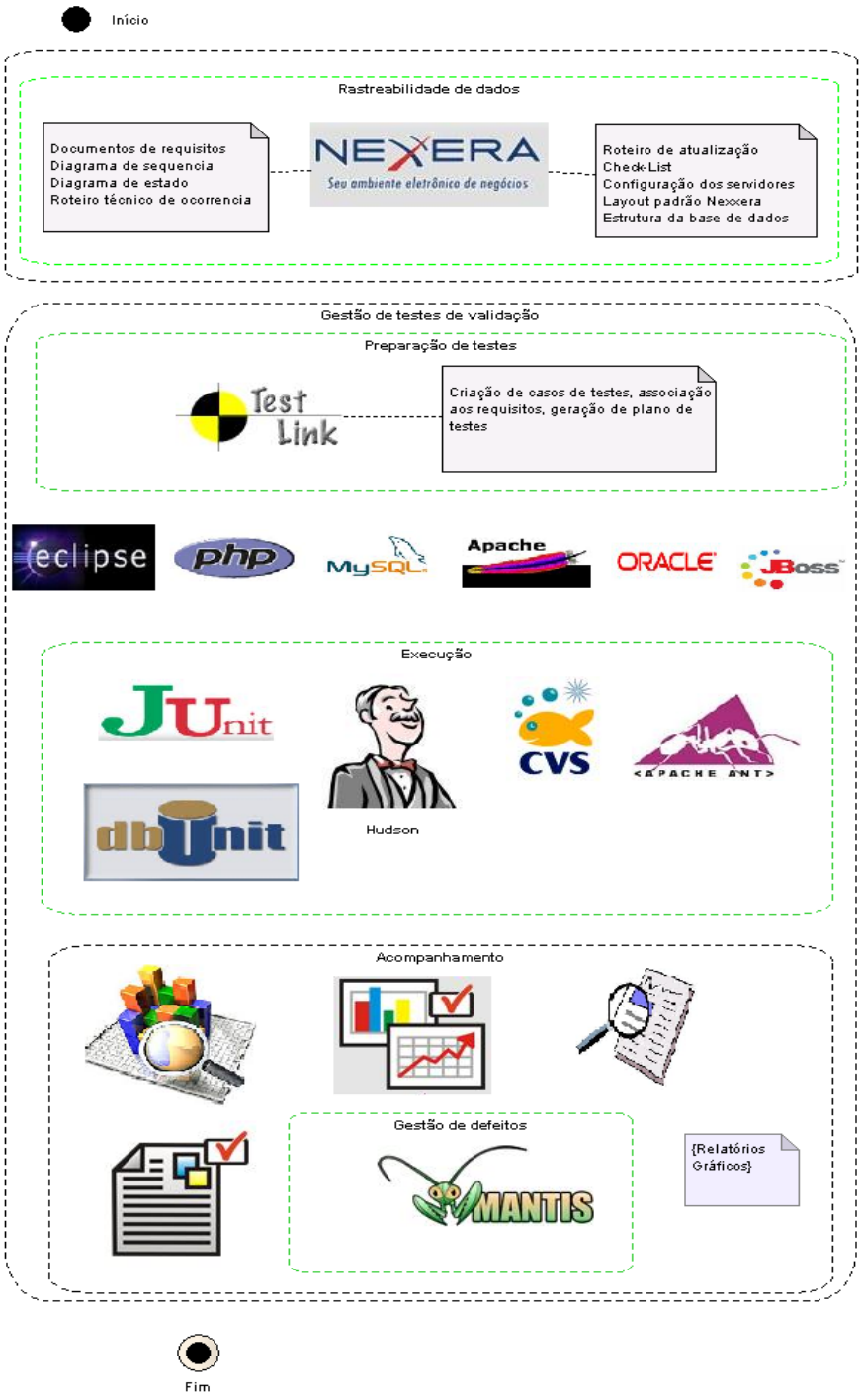


Figura 26: Ferramentas utilizadas no processo.
Fonte: Elaboração da autora, 2009

O conjunto de ferramentas escolhidas e utilizadas para a elaboração do processo de testes cumpre a atividade de auxiliar no gerenciamento de dados, documentos, requisitos, especificação e automação, assim, facilitando, também, o gerenciamento de *check-list* das funcionalidades desenvolvidas, sendo inclusos os dados de forma padronizada e gerando conteúdo para análise posterior. No apêndice C, encontram-se os sites que contém informações e através dos quais é possível realizar o download das ferramentas open source utilizadas nesta monografia.

Cabe salientar que estas ferramentas não possuem o objetivo de criar um processo de testes e, sim de acelerar um processo já definido. Por esta razão, ao adotar uma ferramenta de auxílio o processo já deve estar definido.

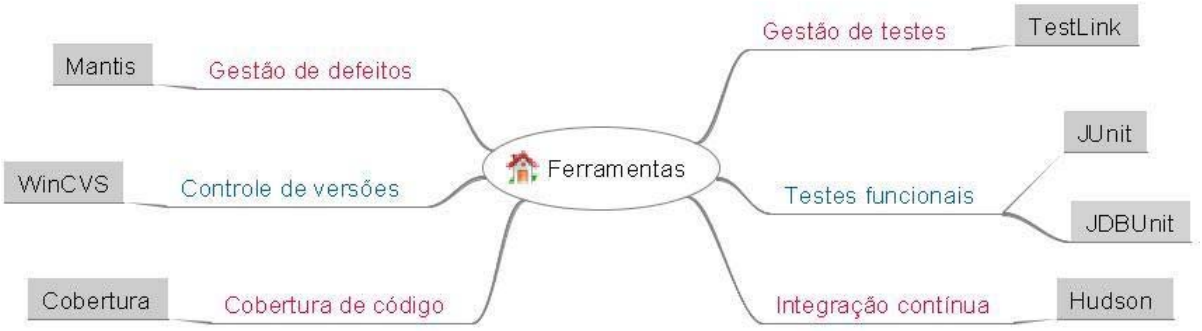


Figura 27: Resumo das ferramentas.
Fonte: Elaboração da autora, 2009

5.3.3.1 Gestão de testes: TestLink[23]

Gestão dos defeitos consiste no controle, planejamento e acompanhamento do processo de testes. Para isto, foi utilizada a ferramenta TestLink, a ferramenta mais popular do mercado segundo pesquisa feita pelo site, que concentra várias ferramentas open source - http://opensourcetesting.org

[23] Testlink : http://www.teamst.org/ - (versão: 1.8.3)

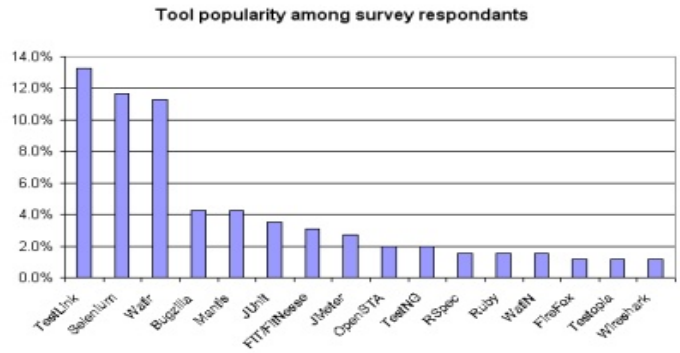


Figura 28: Ferramenta *open source* mais populares segundo pesquisa
Fonte: *Home of TestLink developers Community*, 2008

Esta ferramenta open source oferece um ambiente *web*, que usa linguagem de programação interpretada, *hypertext preprocessor* (PHP) e o sistema de gerenciamento de banco de dados (SGDB), Mysql. O TestLink pode ser acessado pelos vários usuários cadastrados no mesmo com suas relativas atribuições.

Recursos oferecidos pelo TestLink:

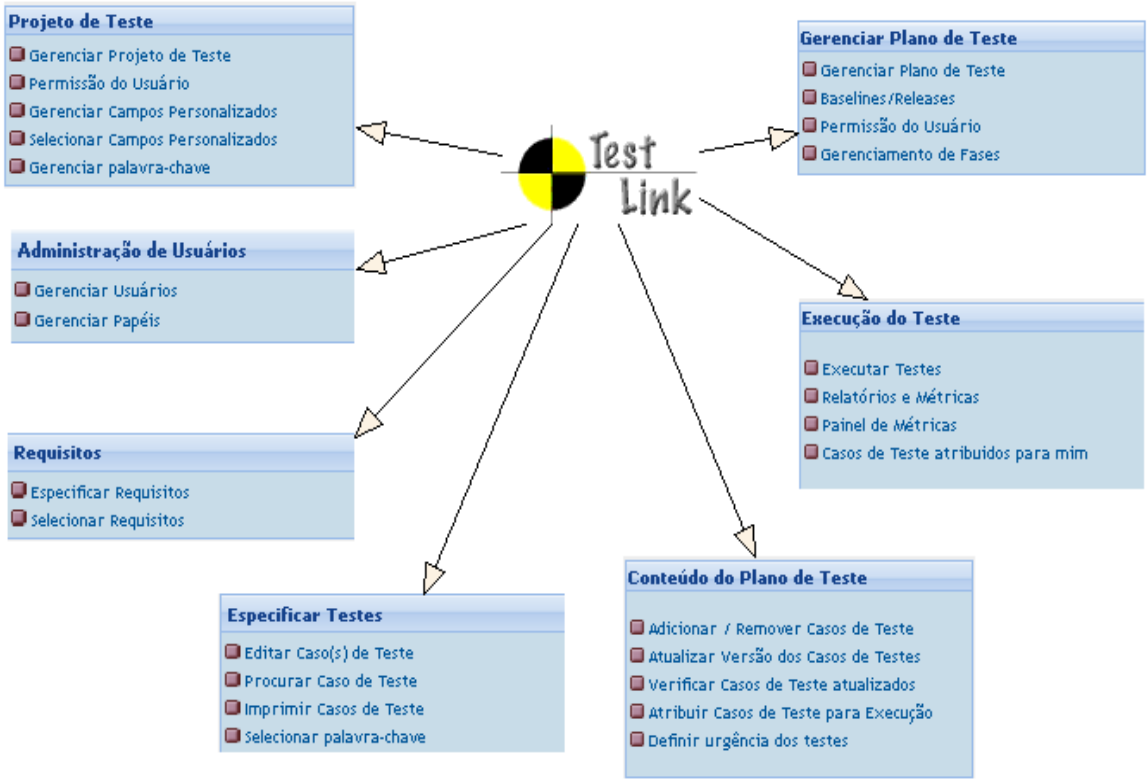


Figura 29: Recursos disponíveis no TestLink
Fonte: Elaboração da autora, 2009

Com a ferramenta é possível gerenciar algumas atividades do processo de testes de forma organizada e padronizada podendo gerar um plano de testes com todos os casos,

requisitos e relatórios de execução e cobertura. É possível o relacionamento do requisito com os casos de testes, facilitando o rastreamento das regras para cada caso criado e a alteração das regras se for necessário, dessa forma mantendo sempre a documentação atualizada. No apêndice A é possível visualizar o relatório gerado pelo TestLink.

Esta ferramenta também permite a integração com uma ferramenta de gestão de defeitos aumentando a rastreabilidade entre eles (caso de teste, requisito e *bug*), essa questão será vista mais a frente, na seção 5.3.3.3.

A seguir algumas telas capturadas durante a utilização e confecção do projeto de testes.

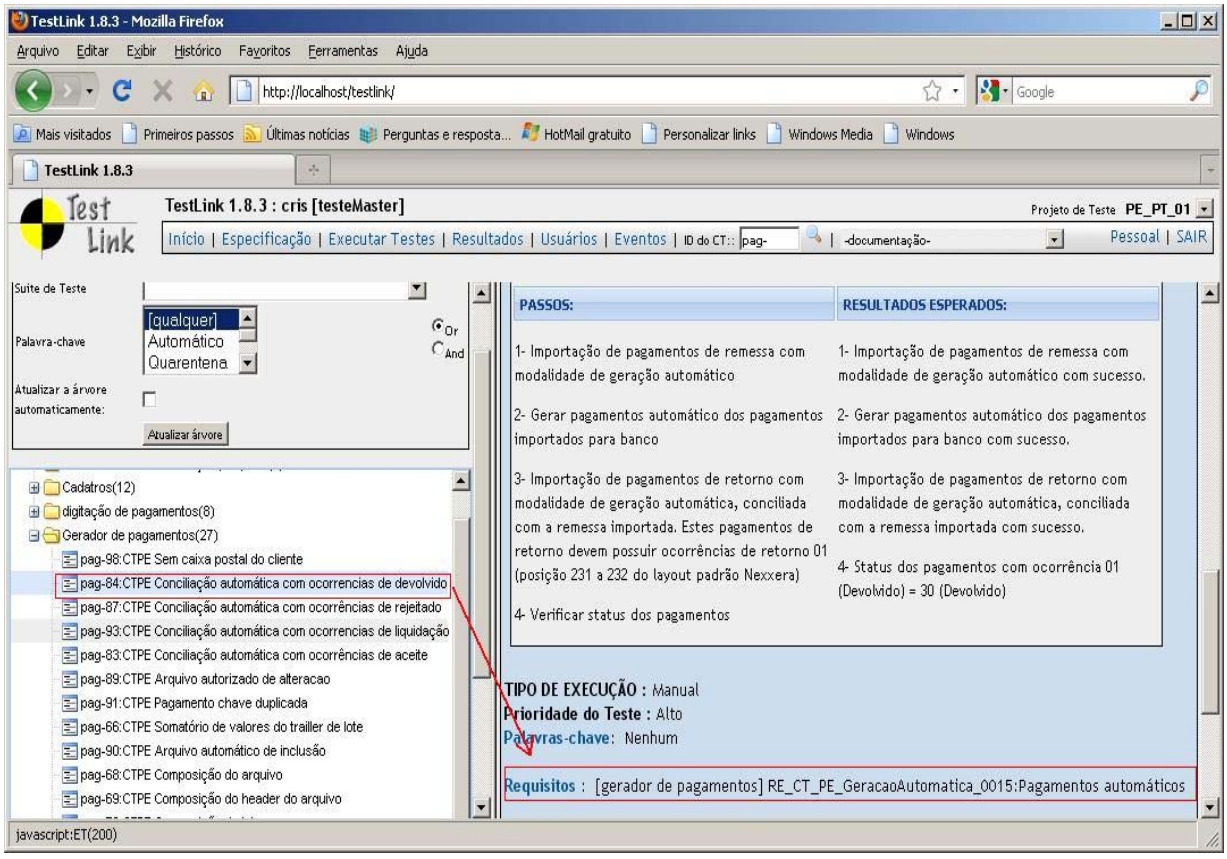


Figura 30: Requisitos da geração de pagamentos autorizados coberto pelo caso de testes
Fonte: Elaboração da autora, 2009

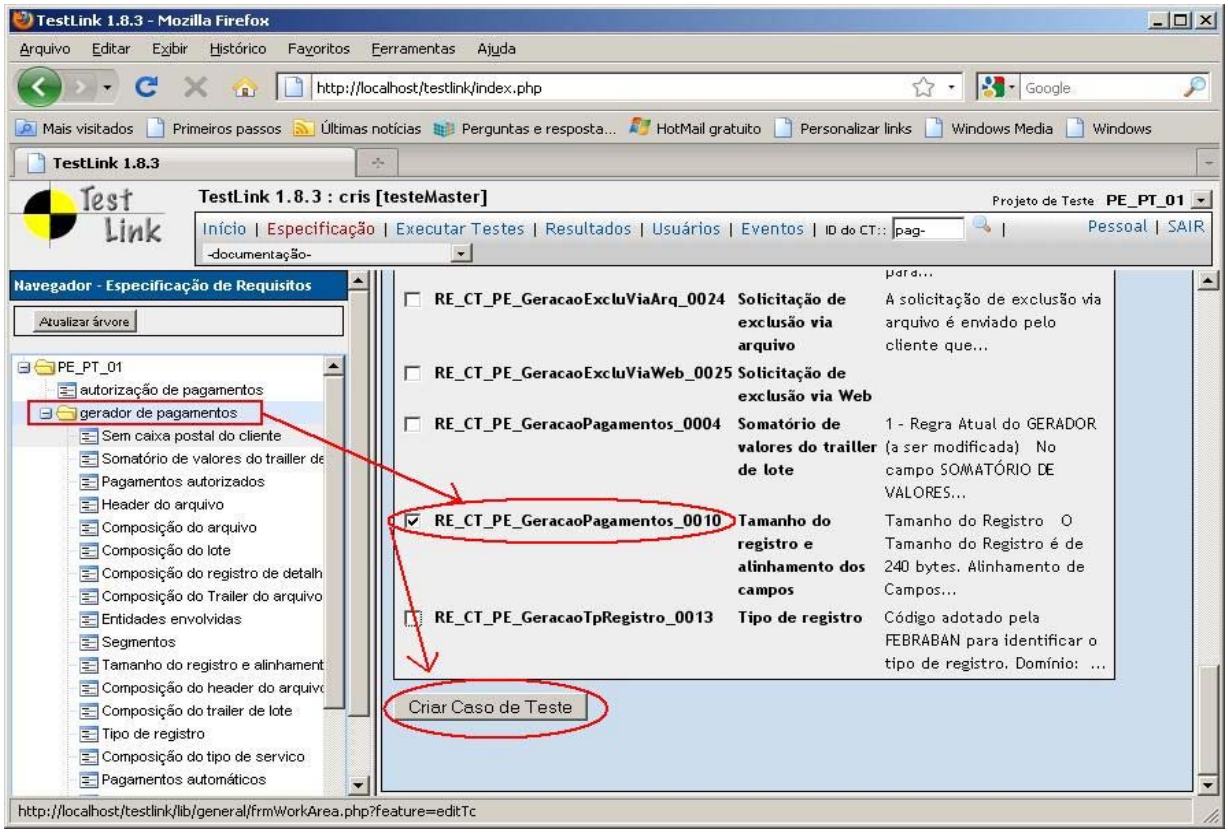


Figura 31: Criação de casos de testes para os requisitos
Fonte: Elaboração da autora, 2009

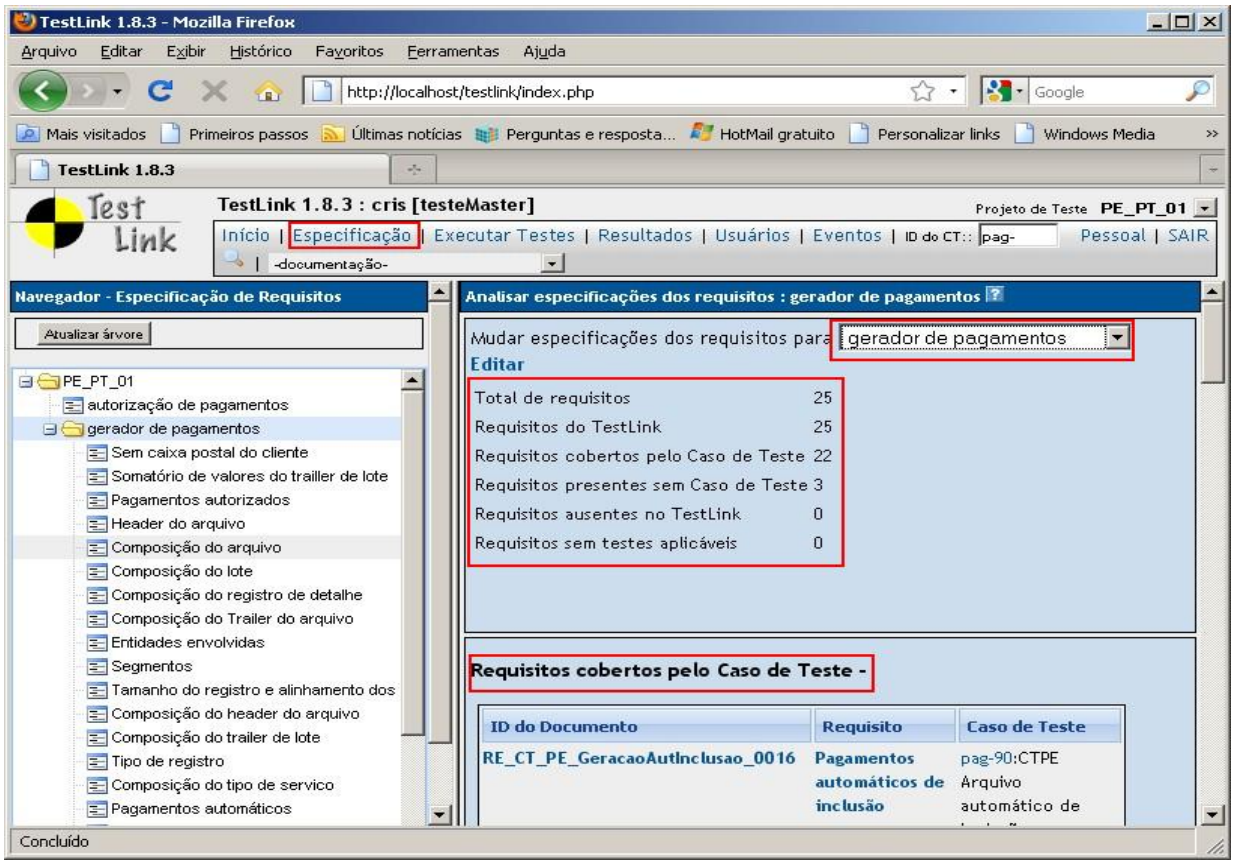


Figura 32: Visualização de requisitos cobertos por casos de testes e vice-versa
Fonte: Elaboração da autora, 2009

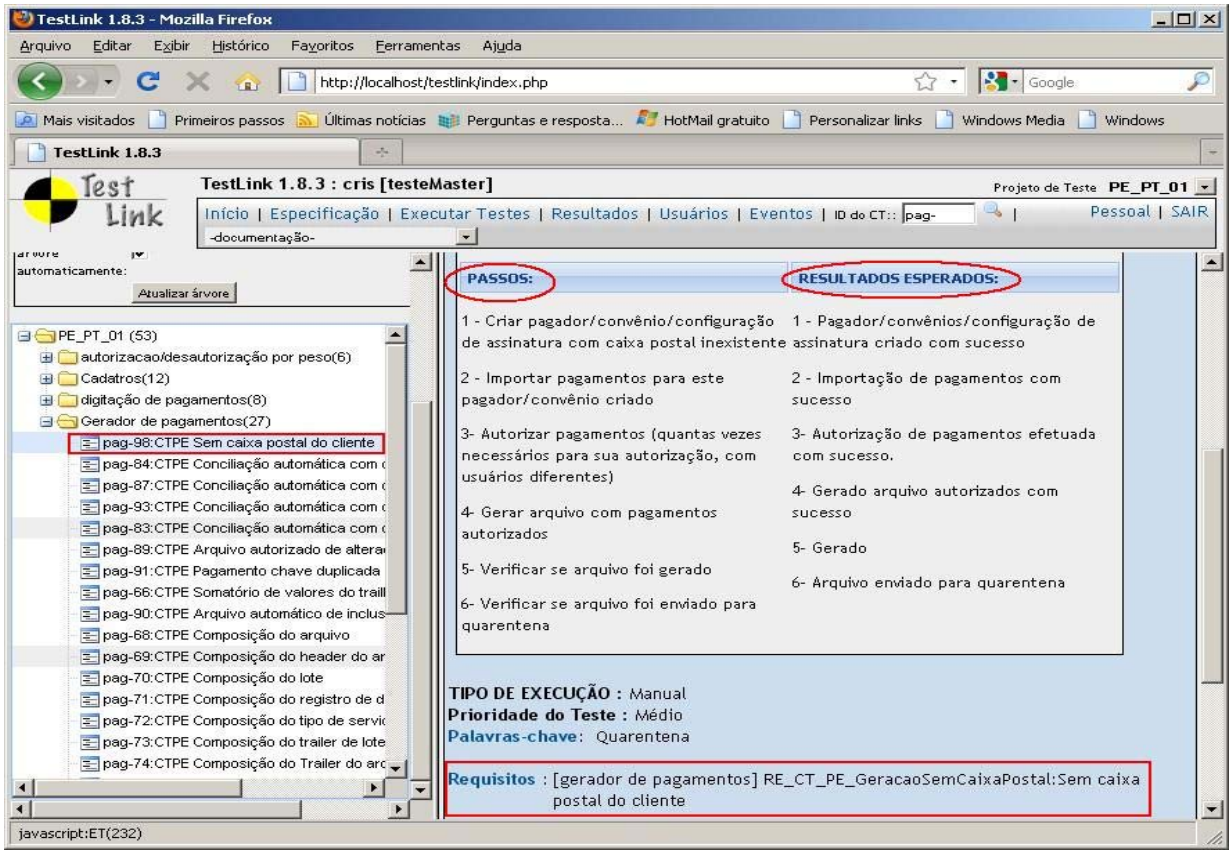


Figura 33: Visualização do caso de teste relacionado com requisito
Fonte: Elaboração da autora, 2009

5.3.3.2 Gestão de defeitos: Mantis[24]

Um gerenciador de defeitos possui o objetivo de dar suporte ao processo de gestão de defeitos, desde a sua origem até o seu fechamento, assim como apoiar a prevenção e minimização de problemas.

A ferramenta Mantis gera diversos gráficos e indicadores que podem ser avaliados gerando relatórios para análises promovendo uma melhoria contínua em todo o processo. O apêndice B contém o relatório gerado pelo Mantis.

Nesta ferramenta é possível o gerenciamento de defeitos novos, corrigidos, re-abertos e previstos, associar a sua severidade e realizar agrupamento de defeitos por testadores, assim como, avisá-los via-email sobre a atribuição e correção de defeitos.

[24] Mantis: http://www.mantisbt.org/ - (versão 1.1.8)

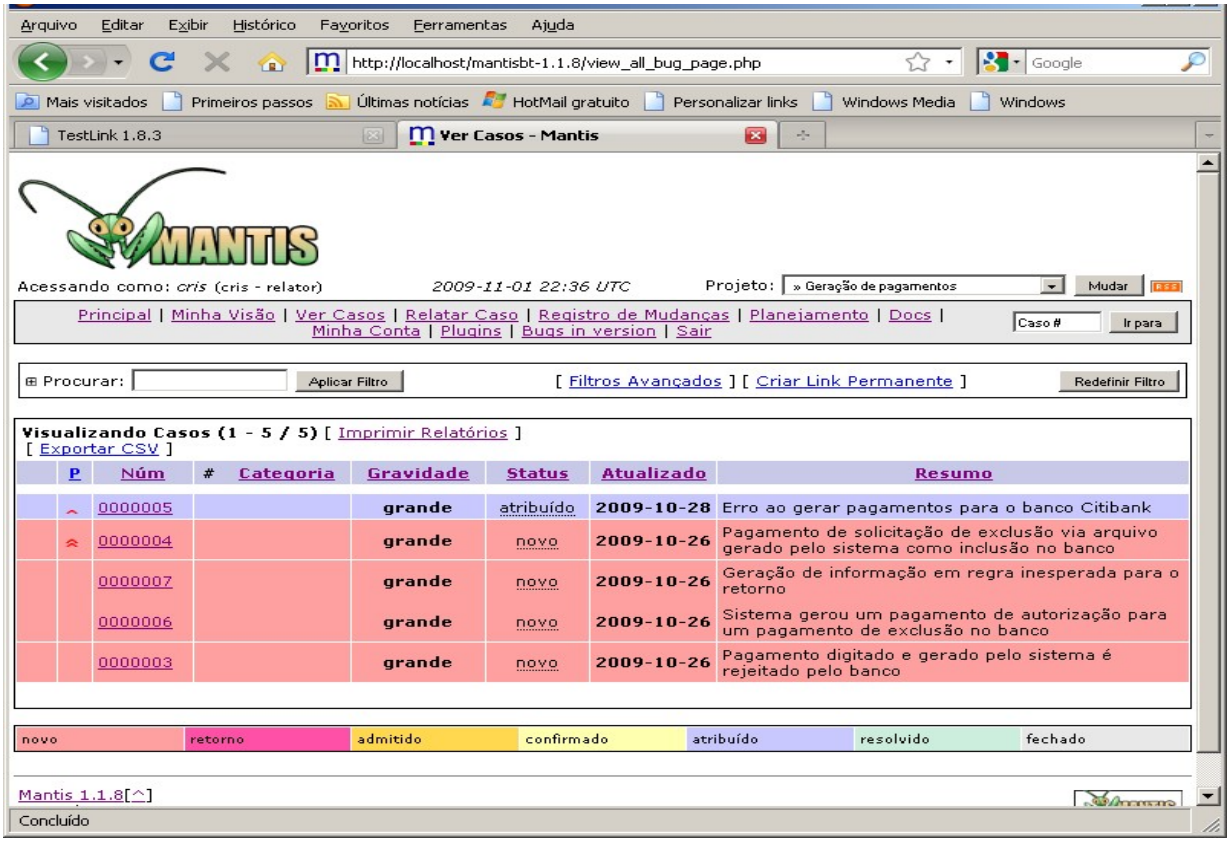


Figura 34: Gerenciamento de defeitos com o Mantis
Fonte: Elaboração da autora, 2009

O TestLink gera alguns gráficos, mostrado a seguir, para melhor visualização do andamento dos defeitos encontrados no produto, sua solução, prioridades, categorias e porcentagens de defeitos.

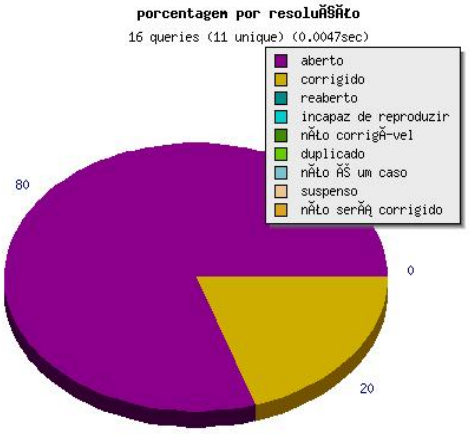


Figura 35: Gráfico de porcentagem de resolução
Fonte: Elaboração da autora, 2009

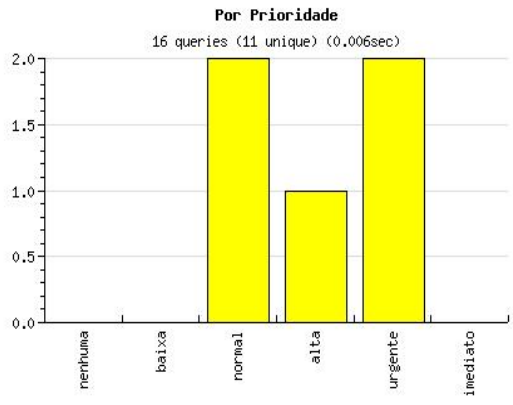


Figura 36: Gráfico por prioridade de defeitos
Fonte: Elaboração da autora, 2009

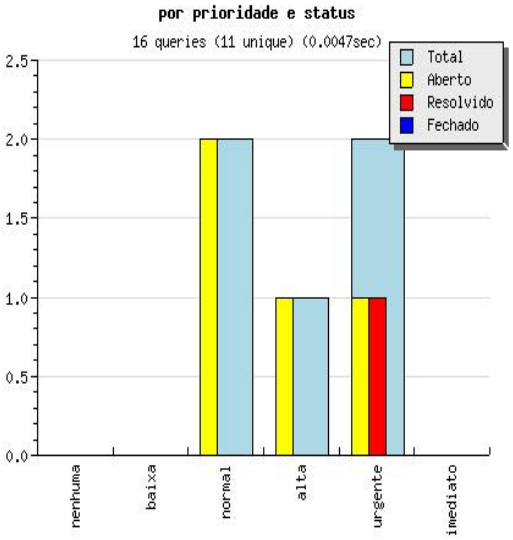


Figura 37: Gráfico de defeitos por prioridade e status
Fonte: Elaboração da autora, 2009

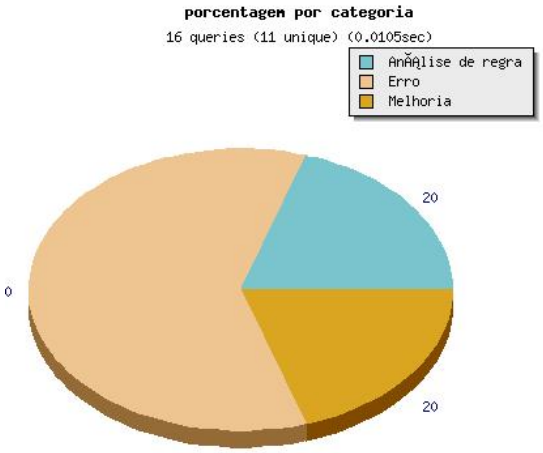


Figura 38: Gráfico de defeitos por categoria
Fonte: Elaboração da autora, 2009

5.3.3.3 Integração TestLink e Mantis

Para uma maior rastreabilidade entre os requisitos, casos de testes e defeitos, é possível a integração das ferramentas open source: Testlink e Mantis. As figuras apresentadas a seguir mostram a potencialidade dessa integração, através de dois exemplos. O primeiro mostrando como resultado o aparecimento de um defeito e no segundo exemplo o bug com a sua correspondente correção.

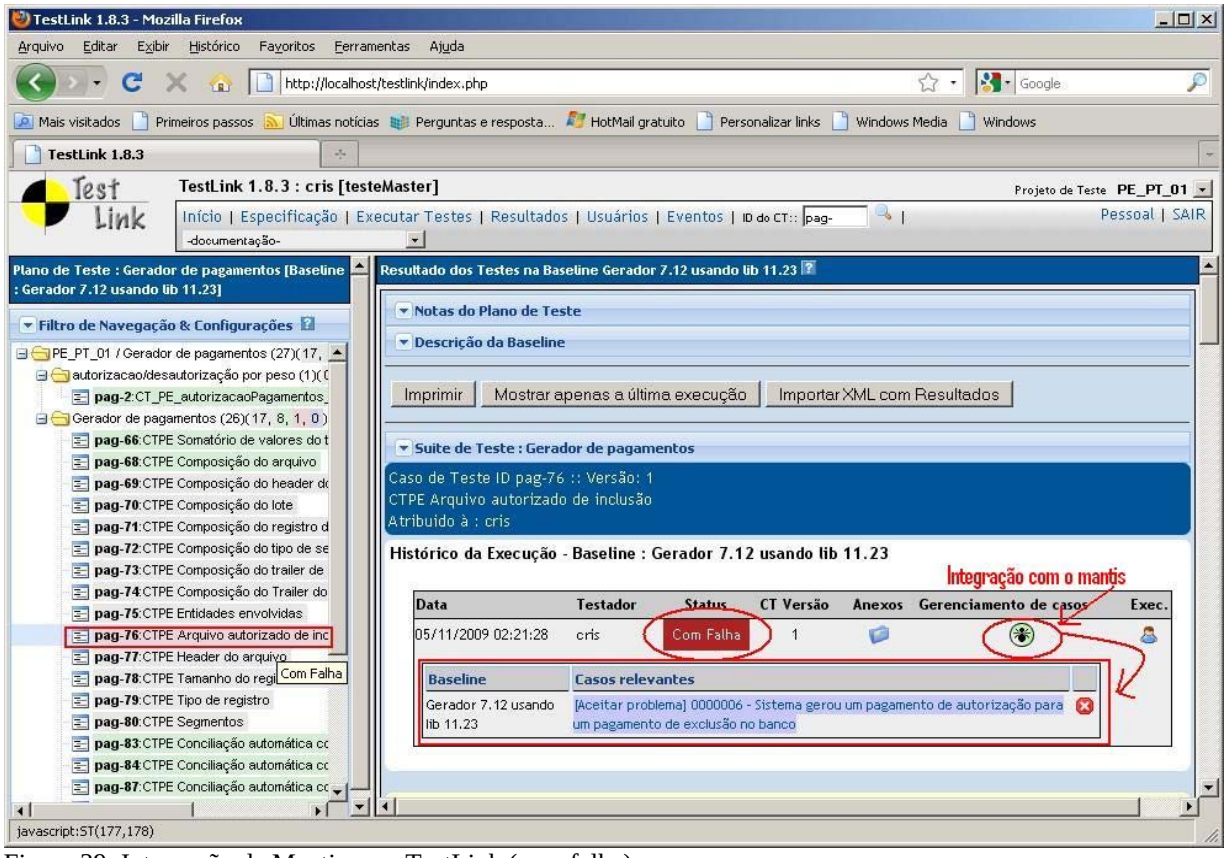


Figura 39: Integração de Mantis com TestLink (com falha)
Fonte: Elaboração da autora, 2009

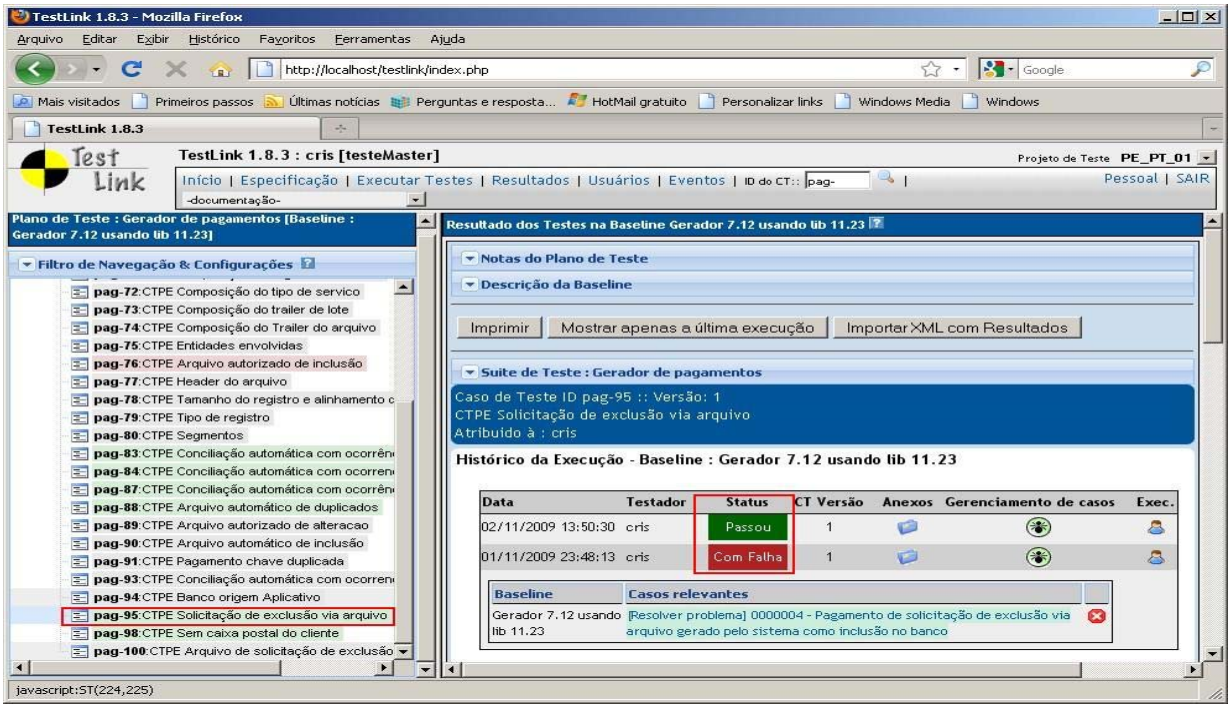


Figura 40: Integração de Mantis com TestLink (com falha e correção)
Fonte: Elaboração da autora, 2009

5.3.3.4 JUnit[25]

É um *framework*[26] *open-source* para criação de testes automatizados na linguagem de programação Java. Com ele é possível verificar a execução das funcionalidades. Usado geralmente para testes unitários, podendo também, ser usado para testes de integração.

Os comandos básicos são:

Comando	Descrição do comando
assertEquals (objEsperado, objRecebido)	Comparar dois valores (igualdade)
assertTrue (expBooleana)	Avalia um retorno booleano, neste caso deve retornar verdadeiro senão irá gerar erro.
assertNull(obj)	Verifica se o retorno do objeto é nulo, neste caso espera-se que esteja nulo senão irá gerar erro.
assertNotNull(obj)	Verifica se retorno do objeto é nulo, se não for nulo irá gerar erro.
assertSame (obj1, obj2)	Compara dois objetos (igualdade)
fail(mensagem)	Indica falha, usado para indicar a falha que não deveria ter acontecido.

Quadro 20: Comandos JUnit
Fonte: Elaboração da autora, 2009

Este *framework* é introduzido no ambiente de desenvolvimento, possibilitando a criação de classes de testes.

A seguir uma demonstração de execução do JUnit, dentro do eclipse com resultado de sucesso.

[25] JUnit: http://www.junit.org/ - (versão 4.6)

[26] *Framework*: estrutura de suporte, projetados com objetivo de facilitar o desenvolvimento.

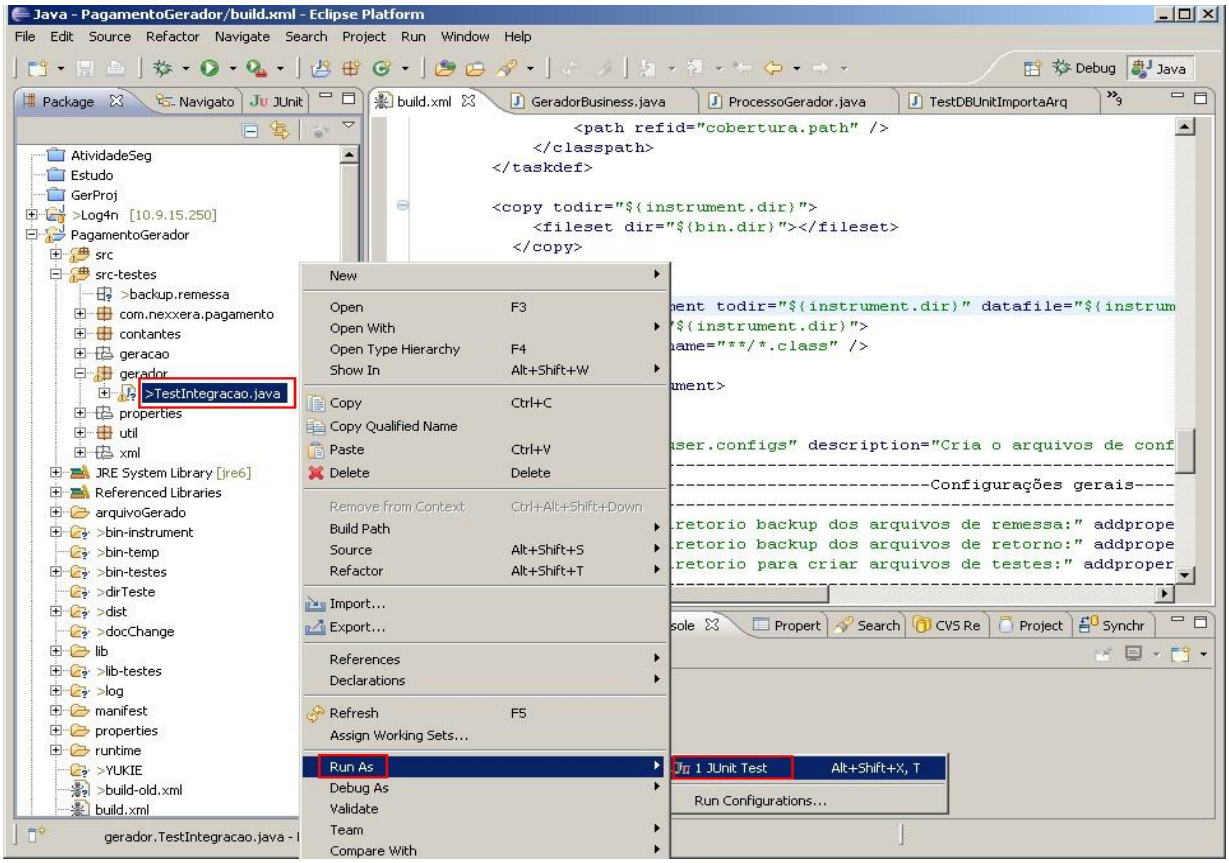


Figura 41: JUnit integrado ao Eclipse para a criação de classe de teste
Fonte: Elaboração da autora, 2009

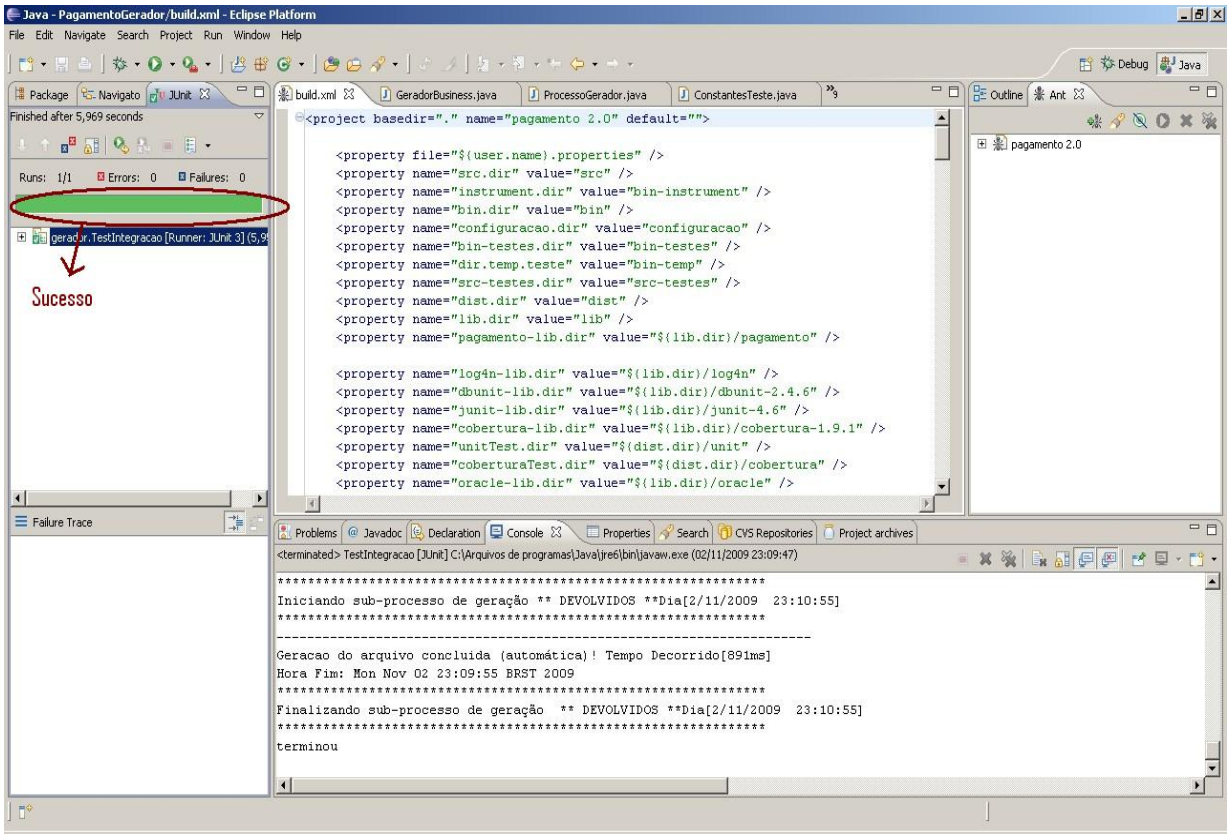


Figura 42: Execução de teste de sucesso no JUnit.
Fonte: Elaboração da autora, 2009

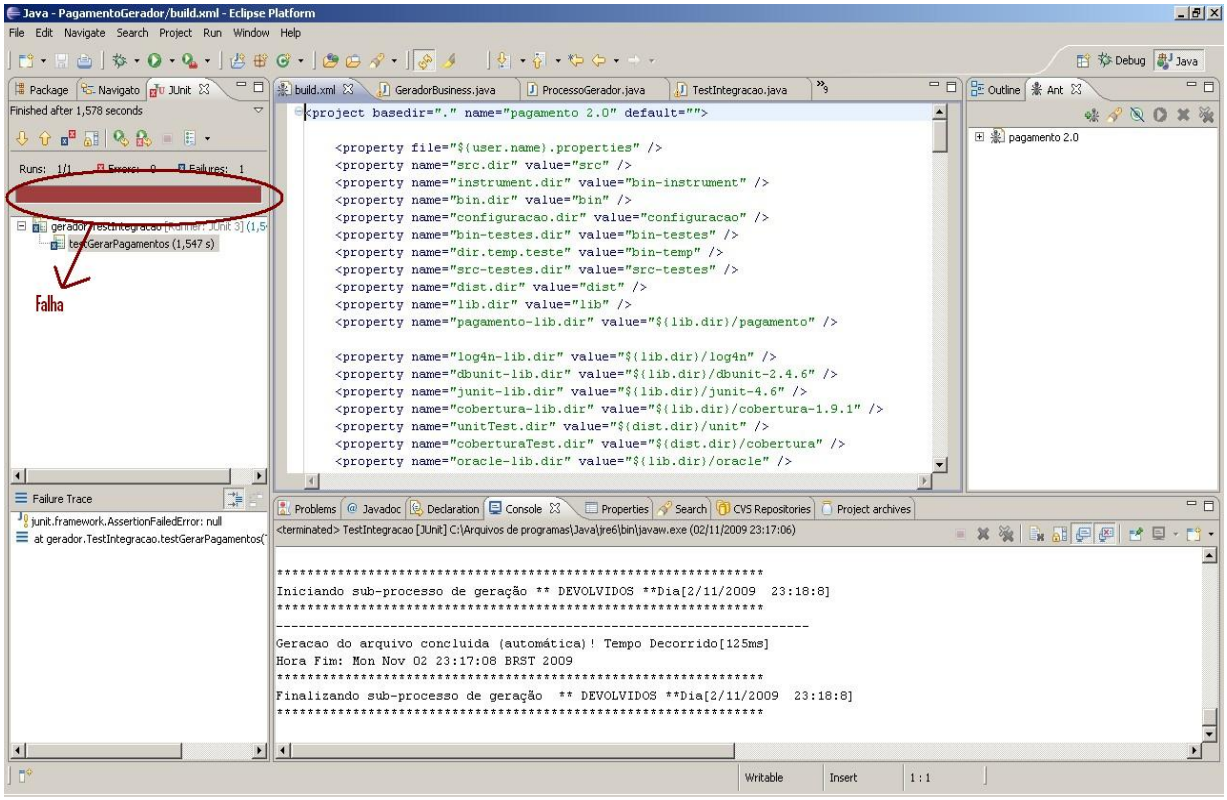


Figura 43: Execução de teste com falha no JUnit
Fonte: Elaboração da autora, 2009

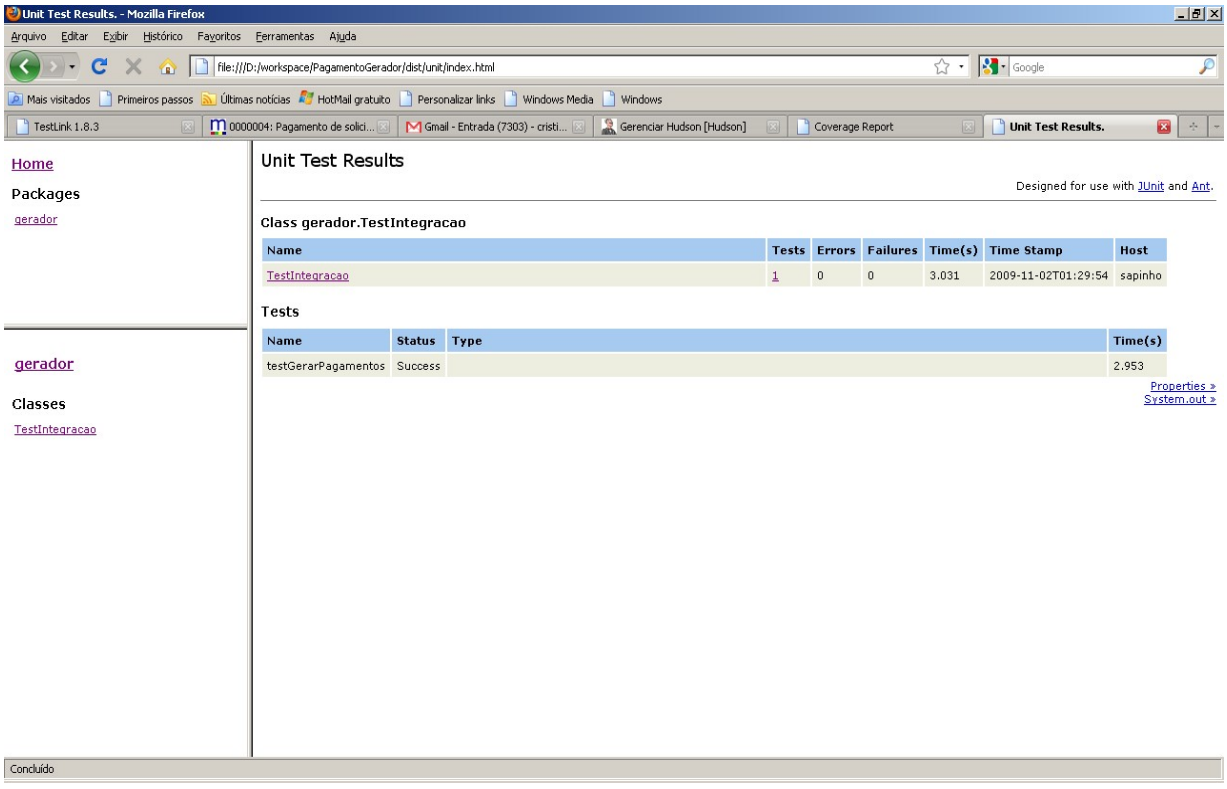


Figura 44: Relatório de teste do JUnit
Fonte: Elaboração da autora, 2009

5.3.3.5 DBUnit[27]

DBUnit é uma extensão do *framework* JUnit, cujo objetivo é popular a base de dados conforme o cenário necessário para validar o teste.

Utiliza arquivos XML para facilitar a carga e limpeza do banco de dados a cada teste executado, ou final do mesmo. A vantagem deste arquivo XML é a preparação antecipada de dados utilizados a cada teste feito, permitindo a previsão dos dados de saída, podendo assim ser feita a comparação dos dados através de *asserts*[28] da ferramenta JUnit.

5.3.3.6 Integração contínua: Hudson[29]

A integração contínua permite que várias pessoas trabalhem no mesmo projeto realizando sincronização do código de forma frequente. Esta sincronização reduz os custos de *refactoring*[30] de código com detecção prévia dos defeitos, pois os problemas são visualizados antes da integração final, ou seja, o objetivo de uma integração contínua é garantir a consistência ao final de cada integração. Geralmente usado por equipes que trabalham no mesmo projeto.

A integração contínua utilizada no estudo de caso é a chamada assíncrona, ou seja, há a utilização de ferramenta para o auxílio da automação, possibilitando também os testes contínuos, a cada build do projeto é disparada uma lista testes automatizados, para a garantia de consistência do código adicionado ou alterado.

---

[27] DBUnit: http://dbunit.sourceforge.net/ - (versão 2.4.6)

[28] *asserts*: técnicas de comparação

[29] Hudson: https://hudson.dev.java.net/ - (versão 1.332)

[30] Refactoring: processo de reescrever um código, com objetivo de melhorá-lo

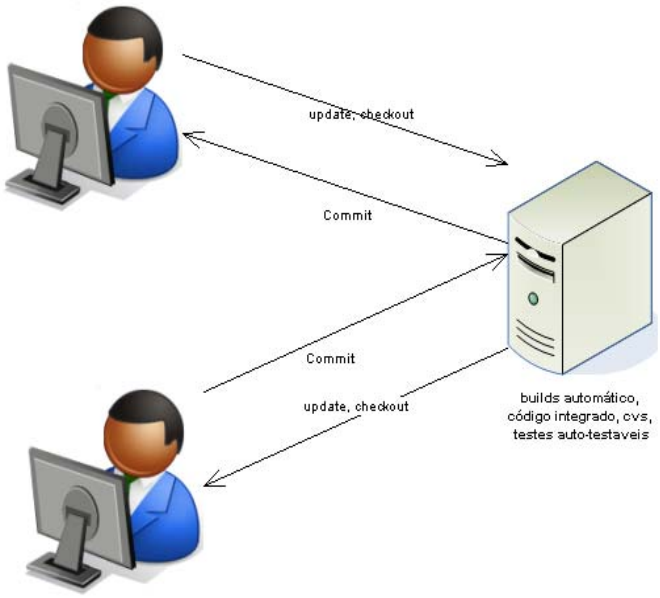


Figura 45: Integração contínua
Fonte: Elaboração da autora, 2009

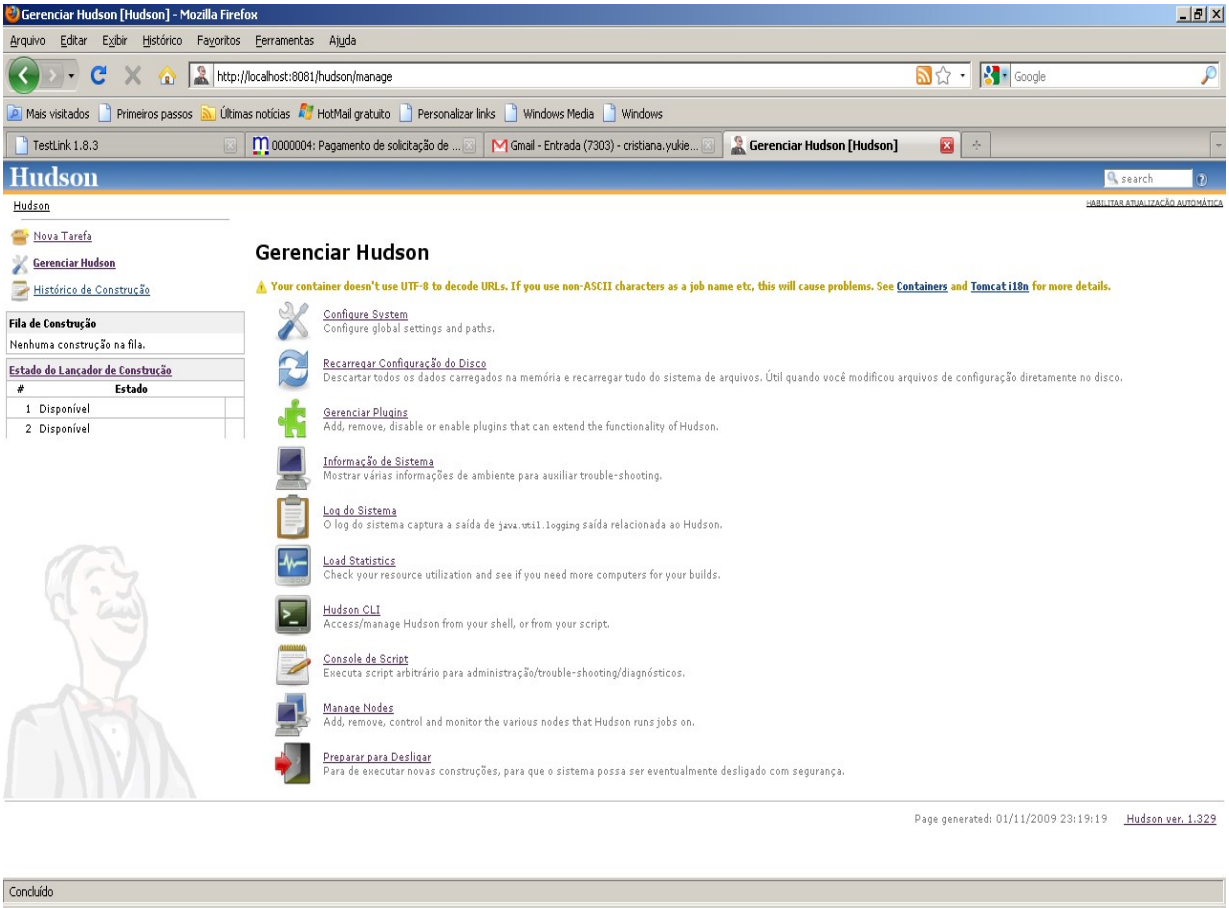


Figura 46: Gerenciamento do Hudson
Fonte: Elaboração da autora, 2009

5.3.3.7 Cobertura: Cobertura[31]

Cobertura é uma ferramenta que analisa quais pontos do código foram testados, isto é, verifica todas as linhas que foram executadas na conclusão de um teste. Com esta ferramenta é possível ter uma visão mais clara, das partes do programa que faltam ser testadas.

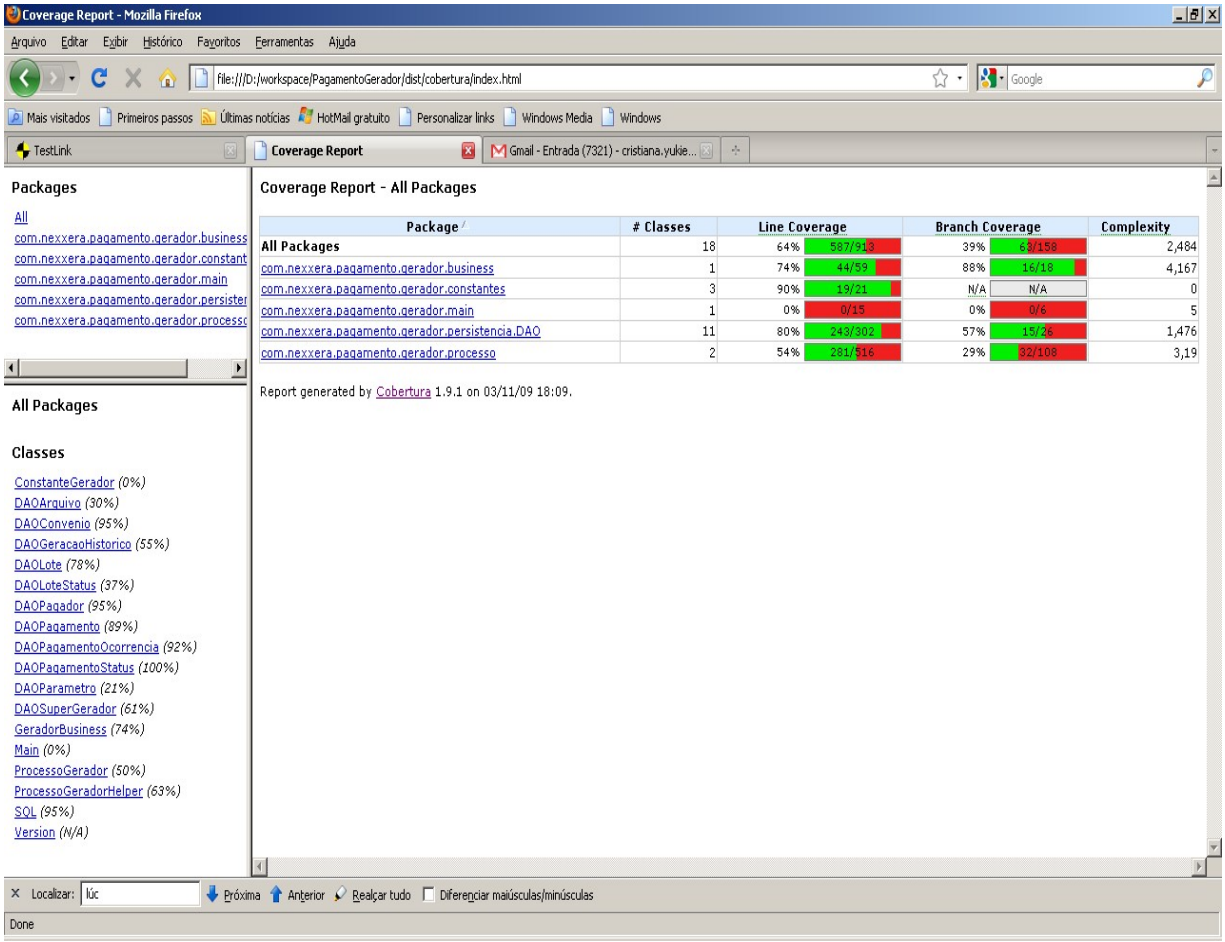


Figura 47: Relatório de cobertura
Fonte: Elaboração da autora, 2009

[31] Cobertura: http://cobertura.sourceforge.net/ (versão 1.9.1)

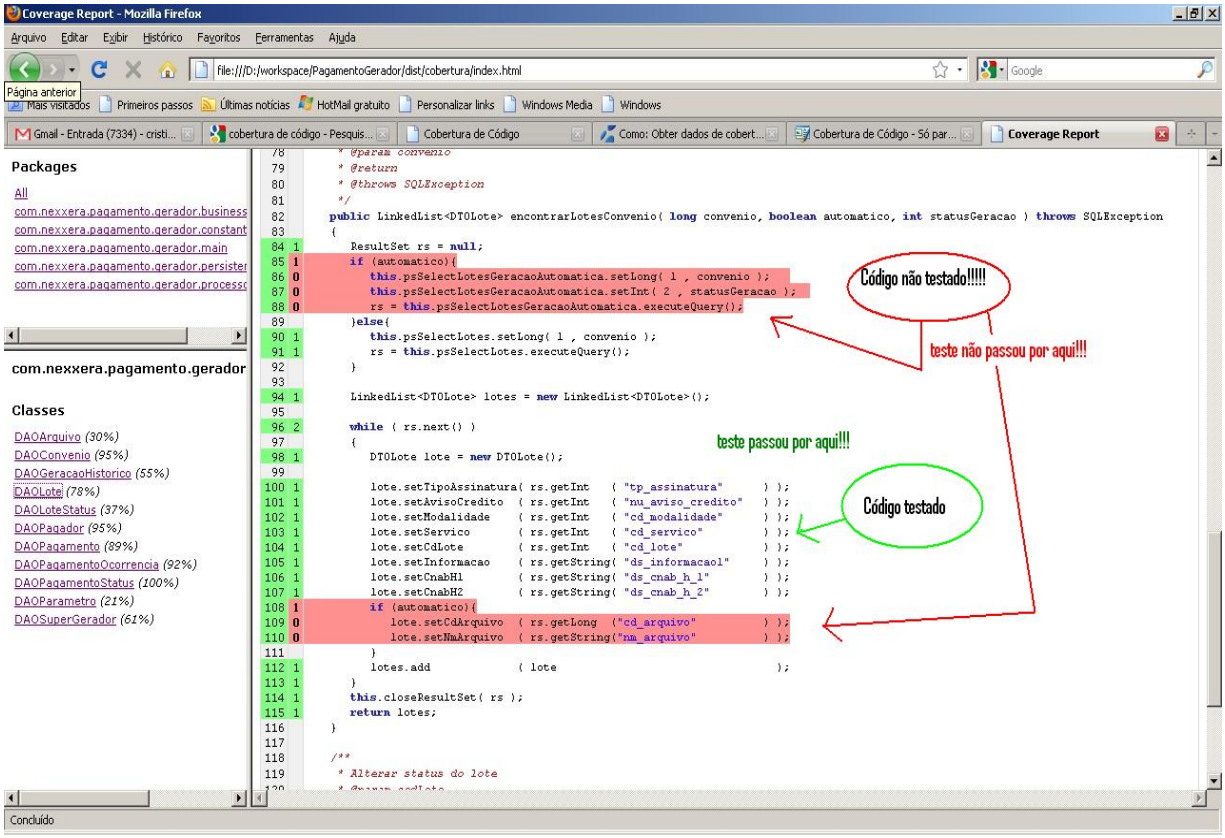


Figura 48: Cobertura de código
Fonte: Elaboração da autora, 2009

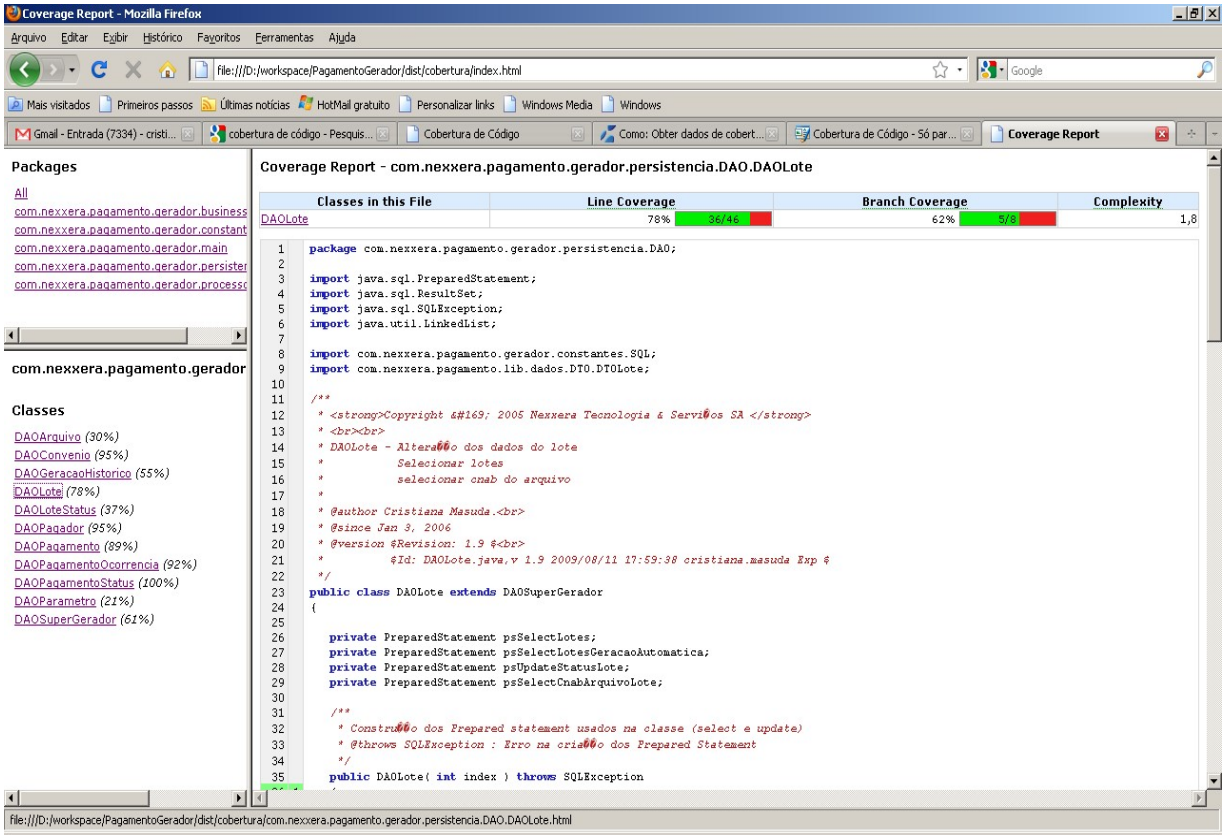


Figura 49: Porcentagem de cobertura de código
Fonte: Elaboração da autora, 2009

5.3.3.8 Controle de versão: Sistema de versões concorrentes (CVS[32])

O CVS é uma ferramenta open source usada com o objetivo de gerenciamento de versões em um repositório centralizado. Isto possibilita que várias pessoas trabalhem no mesmo código, a ferramenta realiza a sincronização quando for feita a integração permitindo controle de alterações, gerenciamento de conflitos, rastreamento e recuperação de dados anteriores.

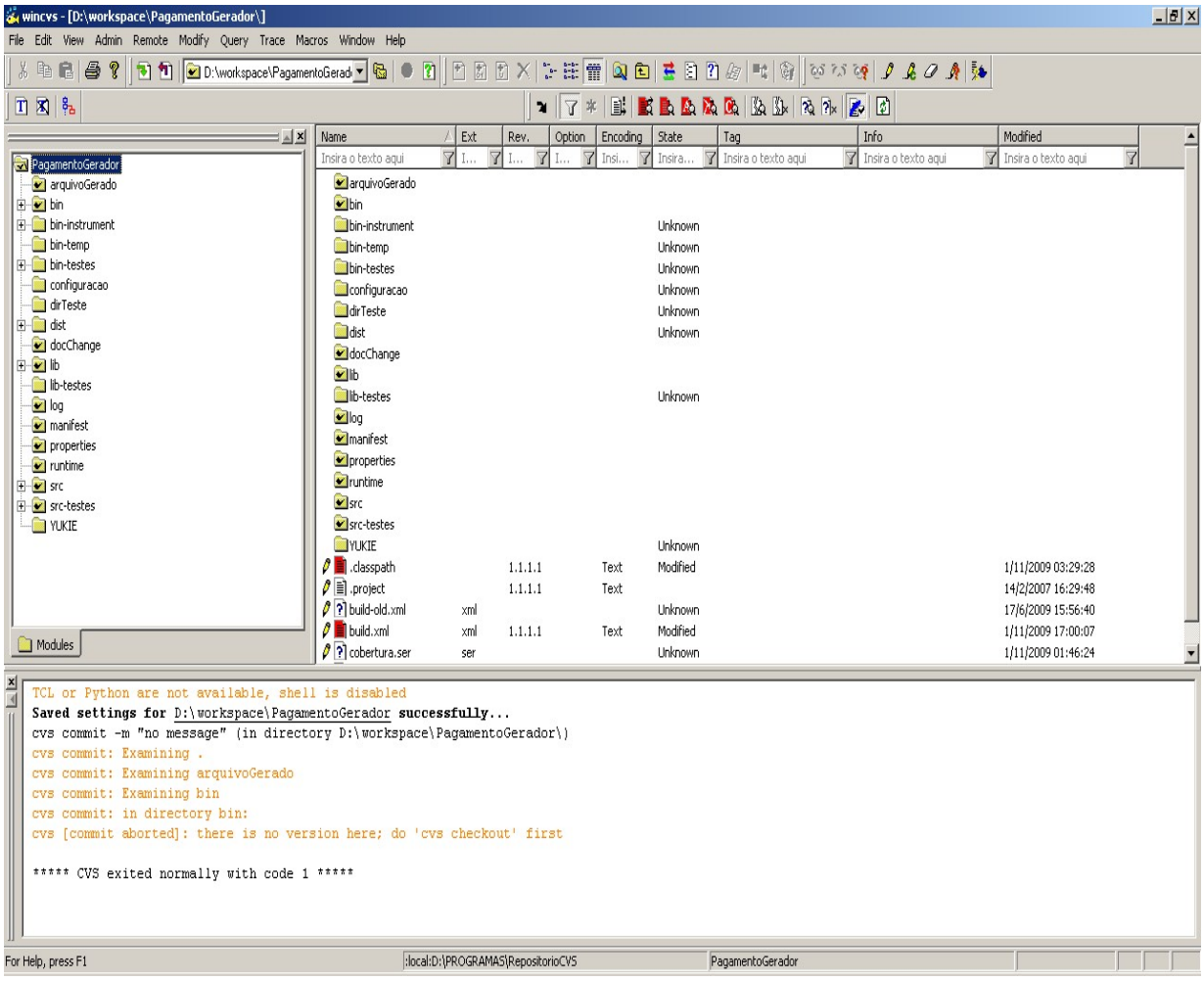


Figura 50: Tela de gerenciamento do CVS para o Gerador de pagamento
Fonte: Elaboração da autora, 2009

[32] CVS: http://www.wincvs.org/ - (versão 2.0.2.4)

5.4 EXECUÇÃO

A execução dos testes foi feita com o auxílio do ambiente de desenvolvimento Eclipse, integrado com o JUnit e DBUnit, populando a base de dados Oracle a cada teste executado.

Foram elaborados tantos testes conforme os casos gerados no Testlink. Como não há integração do TestLink com o JUnit, não foi possível a atualização automática de execução dos casos de testes. Assim sendo, a alternativa a esta falta de integração, foi que a cada *build* que gera uma falha no *software*, há a necessidade de documentação do defeito e acionamento automático de solicitação da gestão do teste.

O TestLink fornece a porcentagem de cobertura dos requisitos pelos casos de testes, casos de testes não executados e atribuição de confecção de testes para os usuários do sistema, como ilustrado nas seguintes figuras.

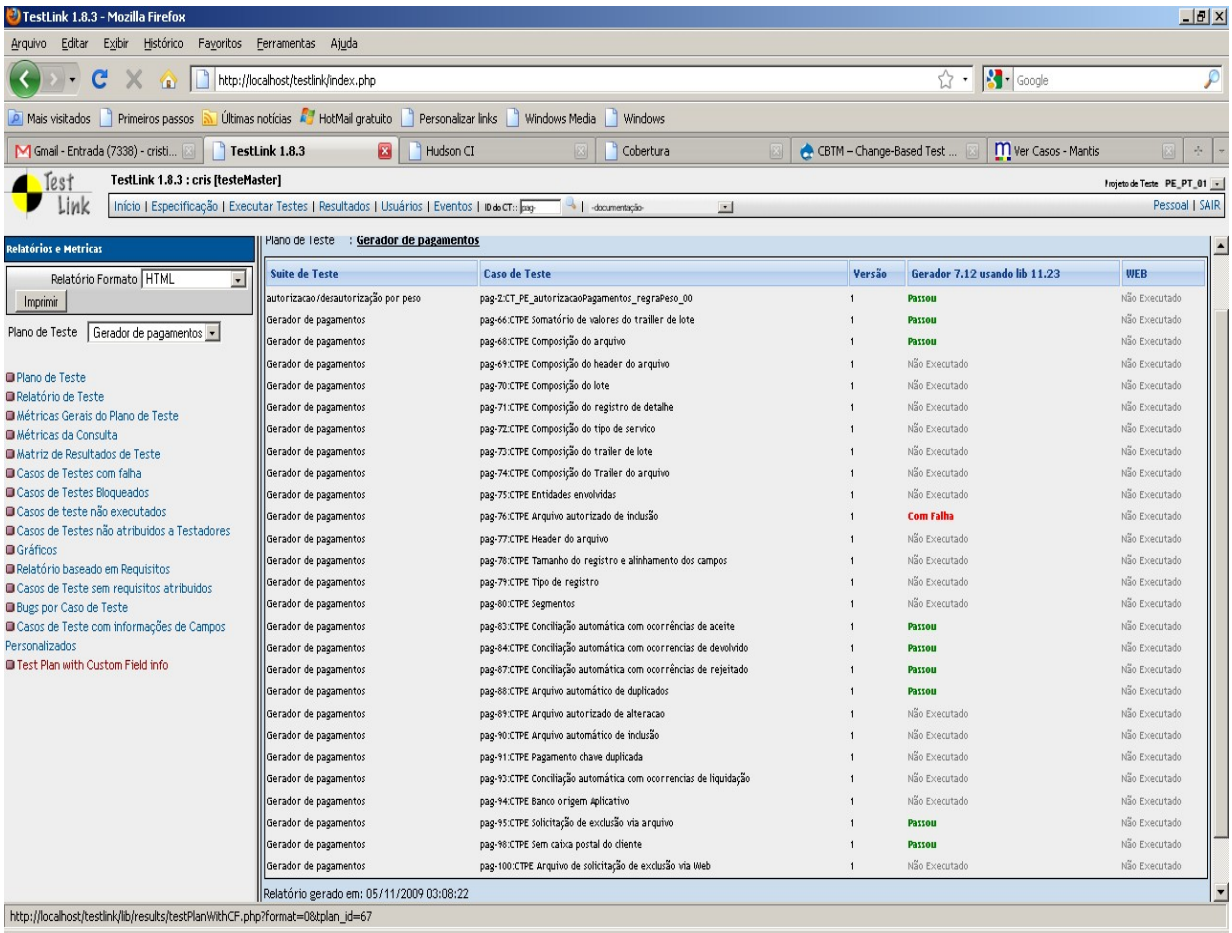


Figura 51: Resultados dos testes dos casos de testes
Fonte: Elaboração da autora, 2009

Lista de Casos de Teste com Falha

Projeto de Teste : **PE_PT_01**

Plano de Teste : **Gerador de pagamentos**

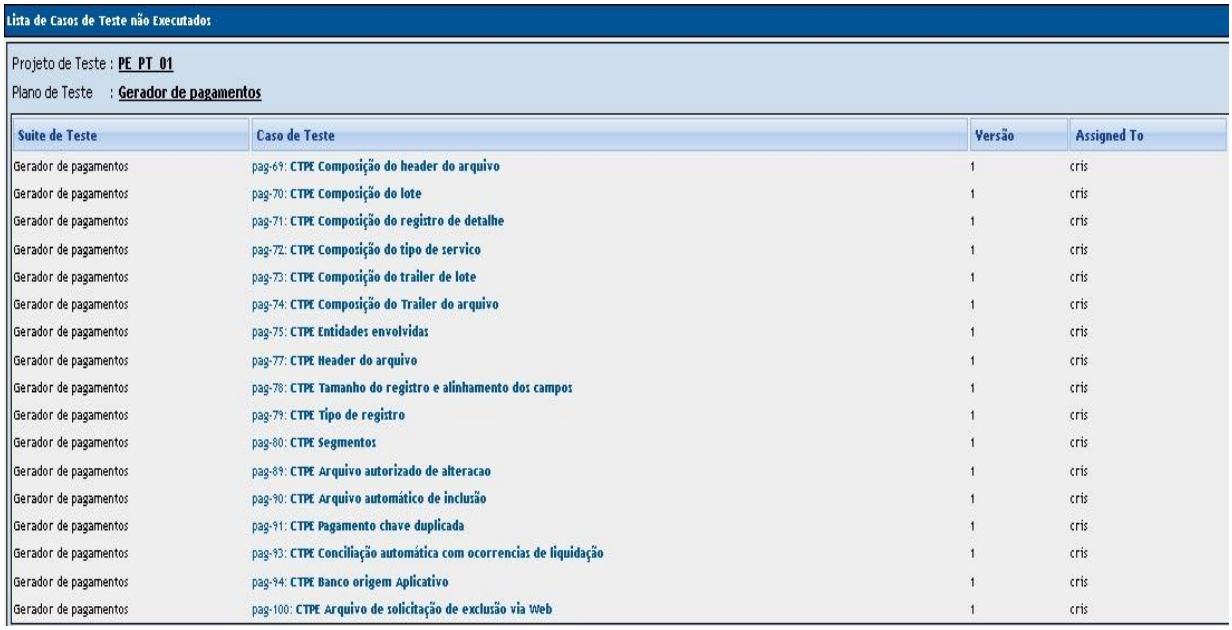
Suite de Teste	Caso de Teste	Versão	Baseline	Executado por	Data	Descrição	Bugs
Gerador de pagamentos	pag-76: CTPE Arquivo autorizado de inclusão	1	Gerador 7.12 usando lib 11.23	cris	05/11/2009 02:21:28		[Novo problema] 0000006 - Sistema gerou um pagamento de autorização para um pagamento de exclusão no banco

Test Cases with missing/linked Bugs0

Apenas os resultados mais recentes são listados. Por exemplo, o Caso de Teste não é mostrado se o resultado do teste da baseline posterior tiver outro status (exceto não executado).

Relatório gerado em: 05/11/2009 03:11:54

Figura 52: Lista de casos de testes com falha
Fonte: Elaboração da autora, 2009



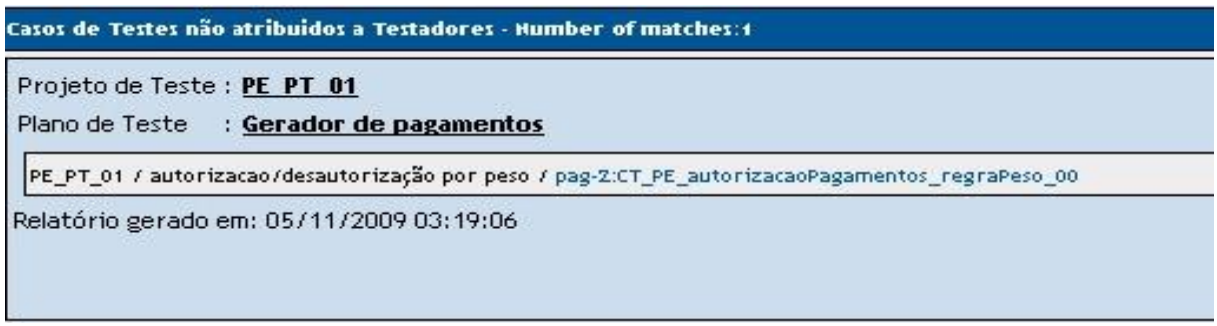
Lista de Casos de Teste não Executados

Projeto de Teste : **PE_PT_01**

Plano de Teste : **Gerador de pagamentos**

Suite de Teste	Caso de Teste	Versão	Assigned To
Gerador de pagamentos	pag-69: CTPE Composição do header do arquivo	1	cris
Gerador de pagamentos	pag-70: CTPE Composição do lote	1	cris
Gerador de pagamentos	pag-71: CTPE Composição do registro de detalhe	1	cris
Gerador de pagamentos	pag-72: CTPE Composição do tipo de servico	1	cris
Gerador de pagamentos	pag-73: CTPE Composição do trailer de lote	1	cris
Gerador de pagamentos	pag-74: CTPE Composição do Trailer do arquivo	1	cris
Gerador de pagamentos	pag-75: CTPE Entidades envolvidas	1	cris
Gerador de pagamentos	pag-77: CTPE Header do arquivo	1	cris
Gerador de pagamentos	pag-78: CTPE Tamanho do registro e alinhamento dos campos	1	cris
Gerador de pagamentos	pag-79: CTPE Tipo de registro	1	cris
Gerador de pagamentos	pag-80: CTPE Segmentos	1	cris
Gerador de pagamentos	pag-89: CTPE Arquivo autorizado de alteracao	1	cris
Gerador de pagamentos	pag-90: CTPE Arquivo automático de inclusão	1	cris
Gerador de pagamentos	pag-91: CTPE Pagamento chave duplicada	1	cris
Gerador de pagamentos	pag-93: CTPE Conciliação automática com ocorrencias de liquidação	1	cris
Gerador de pagamentos	pag-94: CTPE Banco origem Aplicativo	1	cris
Gerador de pagamentos	pag-100: CTPE Arquivo de solicitação de exclusão via Web	1	cris

Figura 53: Lista de casos de testes não executados
Fonte: Elaboração da autora, 2009



Casos de Testes não atribuidos a Testadores - Number of matches:1

Projeto de Teste : **PE_PT_01**

Plano de Teste : **Gerador de pagamentos**

PE_PT_01 / autorizacao/desautorização por peso / pag-2:CT_PE_autorizacaoPagamentos_regraPeso_00

Relatório gerado em: 05/11/2009 03:19:06

Figura 54: Casos de testes não atribuídos a testadores
Fonte: Elaboração da autora, 2009

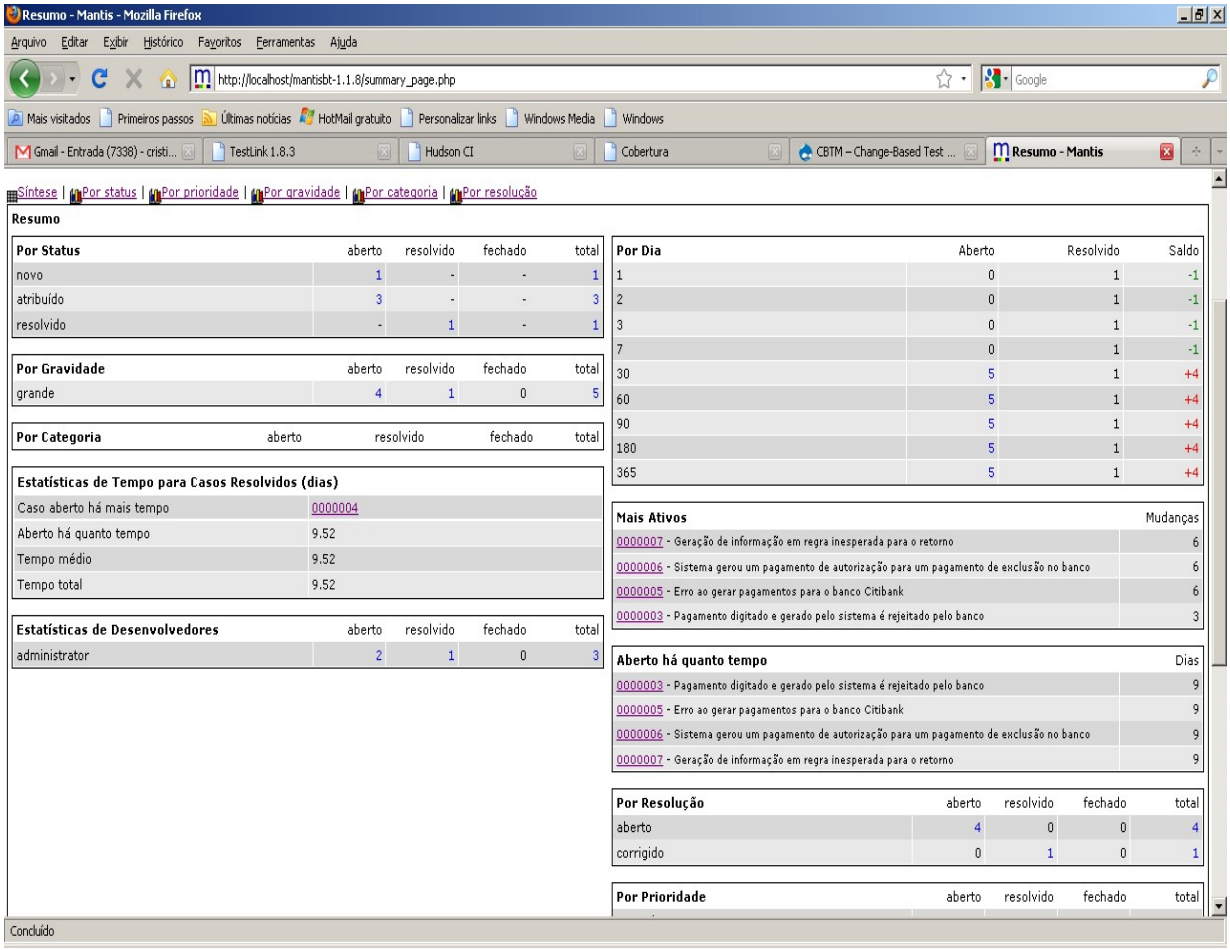


Figura 55: Sumário de *bugs*
Fonte: Elaboração da autora, 2009

5.5 ACOMPANHAMENTO

O acompanhamento dos testes é feito de forma contínua, isto é fácil quando se tem uma integração também contínua. A cada atualização do código no CVS é iniciado o processo de *build*. Caso os testes sejam quebrados pela atualização e o *build* não seja realizado com sucesso, é possível de forma mais rápida identificar os erros e corrigi-los, através dos relatórios gerados.

Os defeitos encontrados em produção são simulados no ambiente de integração percorrendo o mesmo processo de monitoração, identificação e correção, consequentemente as alterações necessárias e novas inclusões de casos de testes e requisitos são feitas nesta fase. Nesta etapa também deve ser realizada a gestão de defeitos, a qual pode ser realizada com apoio da ferramenta Mantis.

Então esta fase de acompanhamento é dividida em dois semi-processos:

- Acompanhamento de funcionalidades novas;
- Acompanhamento de funcionalidade de produção.

5.5.1 Acompanhamento de funcionalidades novas

São testes feitos para funcionalidades agregadas à aplicação antes de entrar em produção. O mais importante dessa fase é agregar o teste desta nova funcionalidade as já existentes, para que estas não prejudiquem as regras antigas. São os chamados de testes de regressão.

5.5.1.1 Inclusão de novos casos de testes no processo existentes

Os passos executados para funcionalidades novas (demandas novas para atender requisito dos clientes) são preenchidas pelos papéis existentes na empresa: programadores, analistas de sistemas, analistas de produtos, equipe de infra-estrutura, equipe de suporte de terceiro nível e de segundo nível.

A inclusão de novos testes no processo existente e a constante atualização do mesmo permitem o rastreamento mais rápido e a verificação do impacto que as novas funcionalidades podem trazer à aplicação legada.

A seguir a sequencia de passos executada para a inclusão de novos casos de testes para uma aplicação legada.

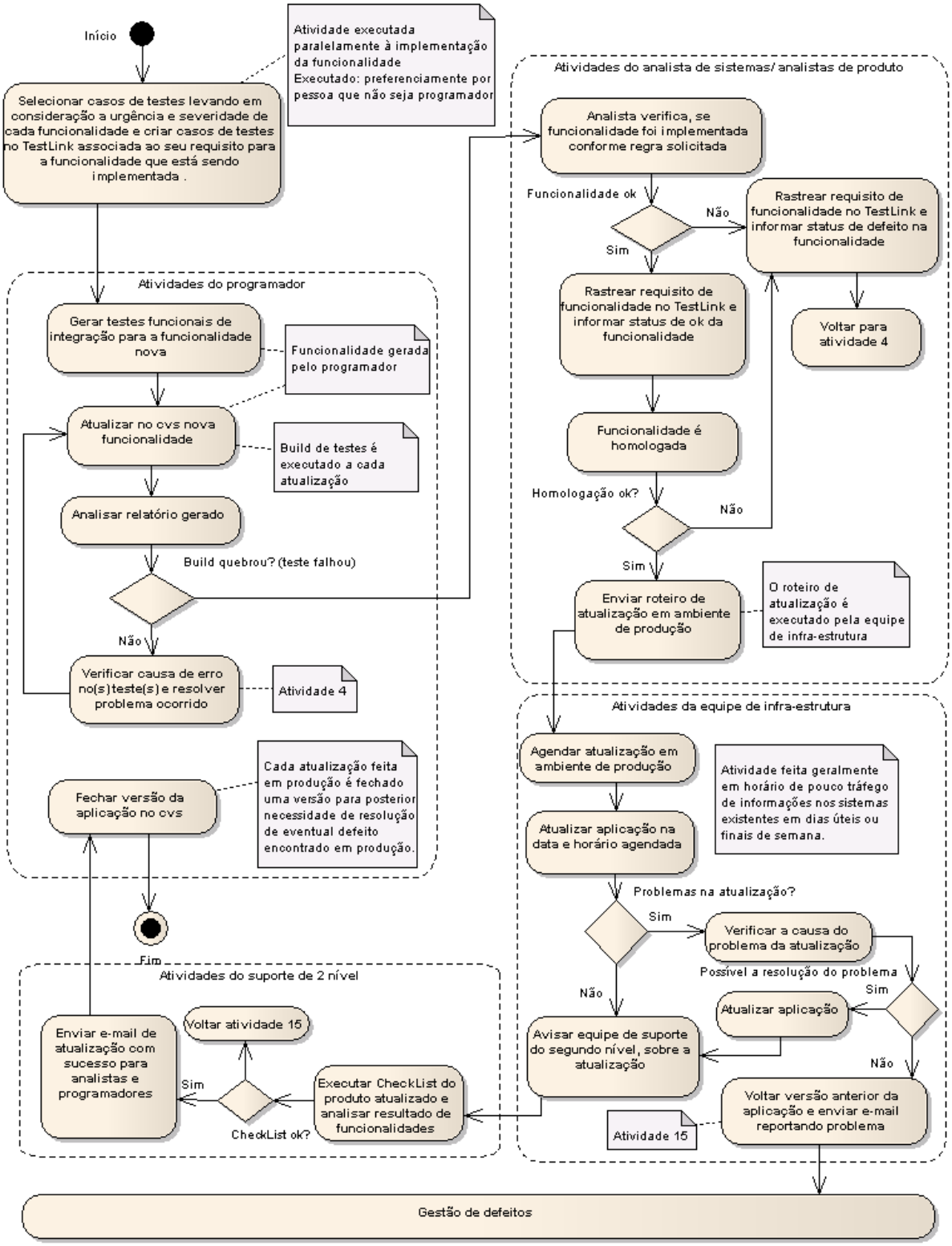


Figura 56: Processo de inclusão de novos casos de testes
Fonte: Elaboração da autora, 2009

5.5.2 Produção

Os problemas ocorridos em ambiente de produção passam pelas seguintes etapas: gestão de defeito no Mantis, gestão de teste no TestLink (criação de casos de testes ou alteração, inclusão de requisitos ou alteração), sincronização de dados no CVS, testes de regressão, análise de relatórios gerados para a verificação de não conformidade com os requisitos ou efeitos colaterais de defeitos no código.

5.5.2.1 Gestão de defeitos

O reporte de falha consiste no gerenciamento de incidências, estes nos fornecem experiências para resolução de problemas similares ou iguais e controle de auditoria. As falhas retratam o andamento do processo de um produto, avaliando suas necessidades e pontos fracos a serem investigados e trabalhados posteriormente para o encontro de uma solução.

Quando as falhas são registradas é possível ter a visibilidade e acompanhar a qualidade do *software*, assim melhorando o processo de inclusão de qualidade de um produto. Além das falhas registradas o acompanhamento de mudança de status do mesmo é documentado, este registro inicia-se no reporte de falha e termina no seu fechamento.

A seguir os passos para o gerenciamento de incidências:

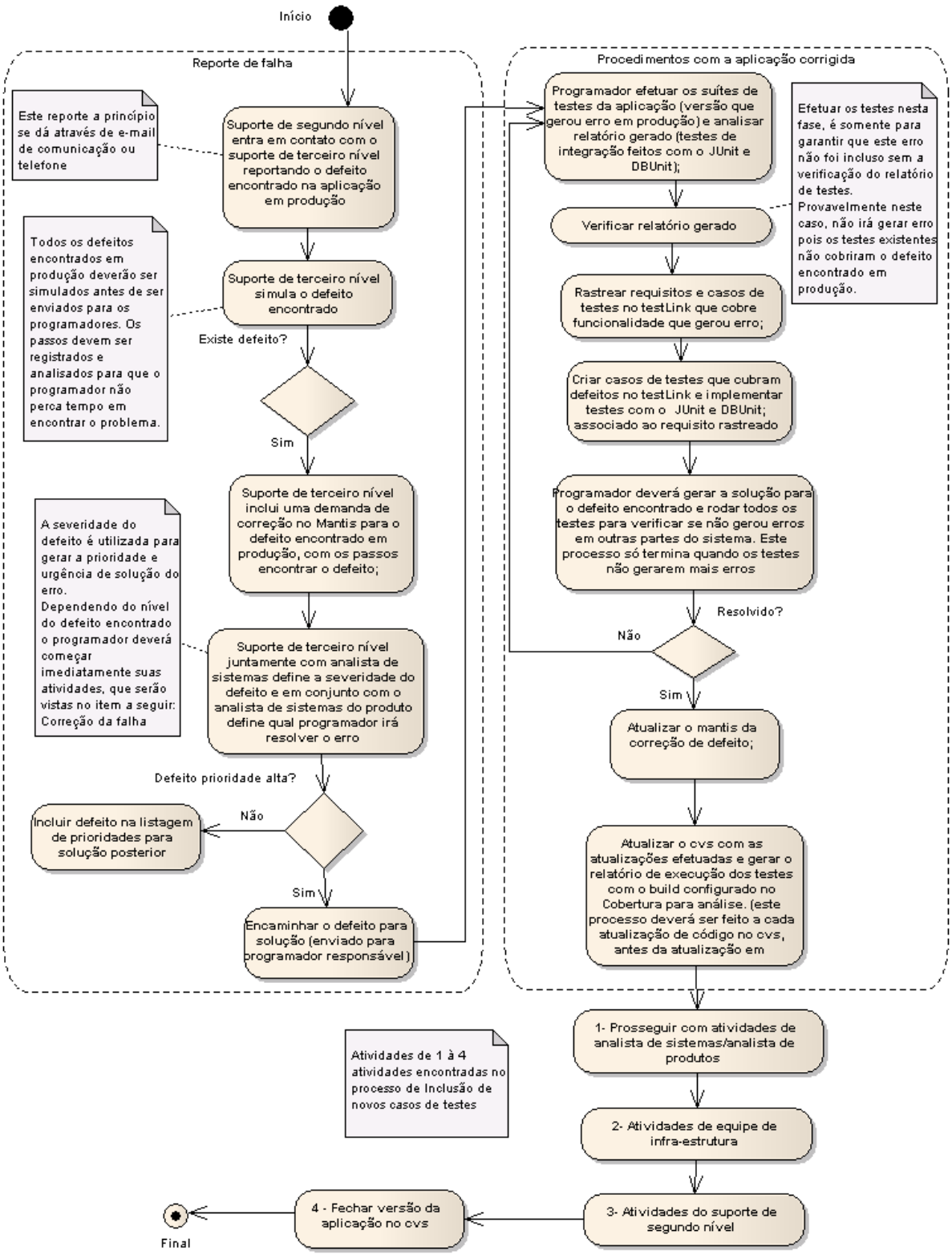


Figura 57: Gerenciamento de incidências
Fonte: Elaboração da autora, 2009

5.6 CONSIDERAÇÕES FINAIS

Este capítulo descreve o processo de teste que foi definido para ser aplicado em um produto da empresa estudo de caso. Pode-se destacar que, como parte inicial desse processo foi realizada uma análise de risco para verificar quais das funcionalidades do produto deveriam ser testadas inicialmente, as etapas subseqüentes realizaram o planejamento, considerando as fases e estratégias de teste a serem realizadas, assim como as ferramentas de *software* a ser usadas. Cada etapa desse processo foi descrita detalhadamente de forma textual ou através de fluxogramas. Também foram definidos quais relatórios, gerados pelas ferramentas, devem ser considerados para dessa forma poder apoiar efetivamente o processo de qualidade de *software*.

Finalizando, a seguir serão apresentadas as conclusões e trabalhos futuros desta monografia.

6 CONCLUSÕES E TRABALHOS FUTUROS

O objetivo de confecção de um processo de testes foi realizado com sucesso neste trabalho. Com o amadurecimento da equipe Nexxera este processo será cada vez mais aperfeiçoado com alterações necessárias para cada projeto específico.

A definição deste processo agregado ao processo de desenvolvimento, permite a rastreabilidade e maior controle sobre as aplicações existentes. Os relatórios com números e gráficos permitem a visualização exata e direta dos resultados obtidos, sem "apegos emocionais" ou "maquiagens". Apoiando, dessa forma, o processo de qualidade, aperfeiçoado a cada etapa explorada, ou seja, o conhecimento do processo aliado ao uso de ferramentas da qualidade nos torna cada vez mais íntimos e ousados para explorá-lo.

A definição de um processo de testes, a um *software* legado foi a parte mais trabalhosa e complexa de toda a monografia. Sendo que a maioria das fontes bibliográficas mostravam a integração do processo de testes a um *software* em desenvolvimento, a qual é a fase apropriada para a inclusão de projeto de testes. Como a situação da aplicação escolhida para testes já encontrava-se em produção, a construção de um processo e sub-processos do controle de qualidade, gestão de defeitos foram feitos em cima de duas realidades diferentes: funcionalidade em produção e funcionalidades novas.

Os projetos de desenvolvimento que estão por nascer, já podem se beneficiar do processo de testes desde o início. Isto irá trazer maior controle e satisfação de qualidade para a empresa, visto que, cada vez mais a qualidade está sendo valorizada e diferenciada no mercado.

A utilização de ferramentas *open source*, em todo o processo, trouxe um grande desafio o qual exigiu uma boa dose de ousadia na procura de informações, para a resolução de *bugs* hoje encontradas nestas ferramentas. Durante a etapa de consulta e participação nos fóruns de testes ou fóruns dessas ferramentas tive a oportunidade de interagir com destacados profissionais da área, o qual tem acrescentado à minha formação profissional. Cabe ressaltar que essas ferramentas estão cada vez mais próximas ao aperfeiçoamento, com tantas cabeças pensando na resolução de suas falhas, na sua integração com outras ferramentas e na facilidade do seu uso. A vantagem maior que vejo nestas ferramentas, são o seu custo, a facilidade e diversidade de informações sobre elas, assim como a gratificação de estar mais íntimos participando e colaborando para a resolução e aprimoramento da mesma, apesar desse fato exigir tempo e dedicação.

Conscientemente sabe-se que a utilização das funcionalidades, oferecidas pelas ferramentas, não foram totalmente e exaustivamente utilizadas nesta primeira etapa, assim sendo, estes aperfeiçoamentos serão feitos futuramente no decorrer do processo de testes e segundo as necessidades de seu uso. Porém, a certeza da inclusão de um processo de testes, por mais que não contemple todo o produto, é melhor do que nenhum teste.

Para trabalhos futuros, sugiro a integração de todas as ferramentas *open source* utilizadas no processo de testes. Nesta monografia a utilização de ferramentas de gestão de documentos na fase de validação não foi utilizada, porém sua utilização agregaria valor ao controle e rastreamento de defeitos para as aplicações. Sentiu-se a falta de integração das ferramentas *open source* para gestão de testes com a execução dos mesmos, foi percebido apenas a integração de requisitos aos testes. Dessa forma a integração de casos de testes com a efetivação dos mesmos, adicionaria uma grande vantagem e qualidade ao processo.

REFERÊNCIAS

ALVAREZ, Miguel Angel. **O que é Oracle?** Disponível em: <http://www.criarweb.com/artigos/230.php> Acesso em 03 maio 2009.

APACHE, ***The Apache software foundatin, Struts***. Disponível em: <http://www.apache.org/> Acesso em 03 maio 2009

BARTIÉ, Alexandre. **Garantia da qualidade de *software***- Rio de Janeiro: Campus, 2002

BASTOS, Anderson, CRISTALLI, Emerson Rios Ricardo, MOREIRA, Trayahu. **Base de conhecimento em teste de *software***, 2 edição : Martins editora livraria Ltda - SP Martins, 2007

BLACK, Rex, EVANS, Isabel, VEENENDAAL, Eric Van, GRAHAM, **Dorothy. *Foundations of software testing,*** ISTQB certification, 2005.

CAETANO, Cristiano. **Aumentando a produtividade com as principais soluções *open Source* e Gratuitas,** 1 Edição, 2007.

CERVO, Amado L, BERVIAN, Pedro A. **Metodologia científica**, 5 edição - São Paulo: Prentice Hall, 2002.

CORREIA, José. A arte de testar Software: 30 anos depois e além. In: **Associação latino-americana de teste de s*oftware,*** 2009.

ENGEL, Peter. **Princípios de organização japoneses (teoria Z)**, 1 edição: Editora Tecnoprint, 1982.

HMELJEVSKI, Érika. **Fundamentos em teste de *software* –** Material didático. Modulo 4. Empresa Supreme Quality, Florianópolis**,** 2009.

INTHURN, Cândida. **Qualidade e teste de *software*.** - Visual Books, 2001

KOSCIANSKI, André, SOARES, Michel dos Santos. **Qualidade de *software*- Aprenda as metologias e técnicas modernas para o desenvolvimento de *software*** - São Paulo: Novatec editora ltda., 2006.

LINHARES, Maurício. **Introdução ao Hibernate 3.** Disponível em: <http://www.guj.com.br/content/articles/hibernate/intruducao_hibernate3_guj.pdf> Acesso em: 3 maio 2009.

MOLINARI, Leonardo. **Testes de *software*: Produzindo sistemas melhores e mais confiáveis** - São Paulo: Érica, 2003.

MOLINARI, Leonardo. **Testes funcionais de *software*:** - Florianópolis: Editora Visual Books, 2008.

MOREIRA FILHO, Trayahú R., RIOS Emerson. **Projeto & engenharia de *software* teste de *software***, 1 edição – Rio de Janeiro: Editora alta books, 2003

MURILLO, Roberto, **O que fazer com métricas geradas por testes de *software*?, 2008** Disponível em: <http://imasters.uol.com.br/artigo/10874/o_que_fazer_com_metricas_geradas_por_testes_de_software> Acesso em: 25 maio 2009.

MYERS, Glenford J. ***The art of software testing***-New Jersey, 2004

NETO, Arilo Cláudio Dias. Introdução a testes de software. **Engenharia de *software* magazine**,v 13, n.01, p.04-08, 2007.

NEXXERA. Pagamento Eletrônico. **A agenda eletrônica de compromissos financeiros.** Disponível em: <http://www.nexxera.com.br/pageletronico.htm> Acesso: 01 marco 2009

NOGUEIRA, Elias, **Testes de caixa cinza (Gray Box).** Disponível em: <http://www.testexpert.com.br/?q=node/838> Acesso em: 25 maio 2009.

OLIVEIRA, Silvio Luiz de. **Tratado de metodologia científica**, 1 edição - São Paulo: Editora Pioneira, 1998.

PAULA FILHO, Wilson de Pádua. Alguns fundamentos da engenharia de *software*. **Engenharia de *software Magazine***,v 13, n.01, p.04-08, 2007.

PRESSMAN, Roger S. Engenharia de *software*. 6 edição – Editora Mc Graw – Hill do Brasil Ltda.

RIOS, Emerson. **Documentação de teste de *software*, dissecando o padrão IEEE 829**, 1 edição – RJ Imagem art studio, 2008

RIOS, Emerson. O que todo gerente sempre quis saber sobre teste de *software*. In: **Associação latino-americana de teste de *software***, 2006. p.19

RIOSOFT, MPT – Melhoria de Processo de Teste Brasileiro. In: **Associação latino-americana de teste de *software***, 2008

SCHEID, Márcia Daniella. **Instituto Lexia: Introdução ao teste de *software*,** 1 Edição, 2007.

SIERRA, Kathy, BATES, Bert. **Certificação Sun para programador java 5**, 2 edição – RJ Alta Books, 2006

SILVA, Edna Lúcia ,MENEZES, Estera Muszkat Menezes **. Metodologia da pesquisa e elaboração de dissertação,** UFSC Florianópolis**,** 4 Edição, 2005.

SUPREME QUALITY. Disponível em: <http://www.supremequality.com.br/>. Acesso em: 15 out. 2009

SUCESU.Disponível em:< *www.inf.furb.br/~egrahl/disciplinas/qualidade/.../slides_sucesu.ppt*>. Acesso em: 10 out. 2009

APÊNDICES

APÊNDICE A – Relatório de teste gerado pelo TestLink

Gerador de pagamentos

Relatório de Teste

Projeto: PE_PT_01

Autor: cris

Impresso por Testlink em 05/11/2009

SUMÁRIO

---

1 Suite de Teste : Gerador de pagamentos

Caso de Teste pag-66: CTPE Somatório de valores do trailler de lote	
Autor:	cris

PRÉ-CONDIÇÃO::

1 – Regra Atual do GERADOR (a ser modificada)

No campo SOMATÓRIO DE VALORES referente ao TRAILLER DE LOTE de quaisquer modalidades de pagamento, é incluída a soma dos valores do campo VALOR DE PAGAMENTO.

2 – Regra do Layout Padrão Nexxera

-> Trailler de Lote referente a títulos de cobrança

Neste campo, deve ser incluída a soma de valores do campo VALOR DO DOCUMENTO.

PASSOS::

1 - Cadastro de dados iniciais (configuração no arquivo dadosIniciais.xml);

2 - Executar o Importador do arquivo de remessa para pagador x (Arquivo de remessa =arqRemessaPagadorX.txt)

3 - Autorizar pagamentos importado;

4 - Executar gerador de pagamentos;

5- Verificar valor do trailler dos lotes (pos 24 – 41).

RESULTADOS ESPERADOS::

1 - Cadastro de dados iniciais efetuado com sucesso;

2 - Importação do arquivo de remessa para pagador x efetuado com sucesso;

3 - Autorizar pagamentos importado efetuado com sucesso;

4 - Gerador de pagamentos efetuado com sucesso;

5- Validação dos lotes conforme arquivo traillerLotesPagadorX;

Último Status:	**Passou**
Baseline	Gerador 7.12 usando lib 11.23
Tester	cris
Requisitos	RE_CT_PE_GeracaoPagamentos_0004: Somatório de valores do trailler de lote
Palavras-chave:	Nenhum

Caso de Teste pag-68: CTPE Composição do arquivo

Autor:	cris

PRÉ-CONDIÇÃO::

O arquivo apresenta uma visão geral dos tipos de pagamento disponíveis e o contexto em que ocorrem, identificando as entidades origem e destino de cada fluxo de troca de informações.

A estrutura do arquivo define a composição do arquivo (header, detalhes (pagamento) e trailer), conceituando cada tipo de registro existente e especificando a forma de utilização de cada um deles por tipo de pagamento, e apresenta o layout do header e do trailer de arquivo. O serviço/produto apresenta de forma detalhada cada pagamento disponível e o contexto em que ocorrem identificando as entidades origem e destino de cada fluxo de troca de informações. Os pagamentos são conceituados através da definição do objetivo, das entidades envolvidas e do fluxo de troca de informações, e apresenta o layout dos registros/segmentos a serem utilizados na sua implementação. Este tipo de serviço/produto possibilita o pagamento de salários, fornecedores, dividendos, etc., através de crédito em conta, cheque, op, doc, pagamento com autenticação ou pagamento de títulos de cobrança. O arquivo de troca de informações entre bancos e empresas é composto de um registro de header de arquivo, um ou mais lotes de pagamentos e um registro trailer de arquivo.

ARQUIVO	Registro Header de Arquivo		Tipo - 0
	LOTES	Registro Header de Lote	Tipo - 1
		Registros de Detalhe Segmentos	Tipo - 3
		Registro Trailer de Lote	Tipo - 5

Com a estrutura apresentada, um único arquivo pode conter vários lotes de pagamentos distintos. Este procedimento permite que as empresas e bancos consolidem em um só arquivo todas as informações que desejam trocar entre si, deve ser previamente acordados entre cada banco e empresa cliente.

PASSOS::

1 - Cadastro iniciais (configuração no arquivo dadosIniciaisComposicaoArquivo.xml);

2 - Executar o Importar arquivo de remessa (Arquivo de remessa =arqRemessaDadosIniciaisComposicaoArquivo.txt);

3 - Autorizar pagamentos importados;

4 - Gerar pagamentos

RESULTADOS ESPERADOS:

1 - Cadastro iniciais executado com sucesso;

2 - Importação efetuada com sucesso;

3 - Autorizar pagamentos importados;

4 - Gerador efetuado com sucesso gerando 1 arquivo;

5-Validar dados da composição do arquivo (arquivo de arqRemessaComposicaoArquivoGerado.txt), validação com sucesso.

Último Status:	**Passou**
Baseline	Gerador 7.12 usando lib 11.23
Tester	cris
Requisitos	RE_CT_PE_GeracaoPagamentos_0001: Composição do arquivo
Palavras-chave:	Nenhum

Caso de Teste pag-69: CTPE Composição do header do arquivo

Autor:	cris

PRÉ-CONDIÇÃO::

Campo					Posição		Nº Dig	Nº Dec	Form	Default	Desc
					De	Até					
01.0	Controle	Banco		Código do Banco na Compensação	1	3	**3**		Num		**G001**
02.0	Controle	Lote		Lote de Serviço	4	7	4		Num	"0000"	G002
03.0	Controle	Registro		Tipo de Registro	8	8	1		Num	"0"	G003
04.0	CNAB			Uso Exclusivo NEXXERA	9	17	9		Alfa	Brancos	G004
05.0	Empresa	Inscrição	Tipo	Tipo de Inscrição da Empresa	18	18	1		Num		G005
06.0	Empresa	Inscrição	Número	Número de Inscrição da Empresa	19	32	14		Num		G006
07.0	Empresa	Convênio		Código do Convênio no Banco	33	52	20		Alfa		G007
08.0	Empresa	Conta Corrente	Agência								
				Agência Mantenedora da Conta	53	57	5		Num		G008
09.0	Empresa	Conta Corrente	DV	Dígito Verificador da Agência	58	58	1		Alfa		G009
10.0	Empresa	Conta Corrente	Conta								
			Número	Número da Conta Corrente	59	70	12		Num		G010
11.0	Empresa	Conta Corrente	DV	Dígito Verificador da Conta	71	71	1		Alta		G011
12.0	Empresa	Conta Corrente	DV	Dígito Verificador da Ag/Conta	72	72	1		Alta		G012
13.0	Empresa	Nome		Nome da Empresa	73	102	30		Alta		G013
14.0	Nome do Banco			Nome do Banco	103	132	30		Alta		G014
15.0	Van			Nome da Van	133	142	10		Alta	NEXXERA	NX03
16.0	Arquivo	Código		Código Remessa / Retorno	143	143	1		Alta		G015
17.0	Arquivo	Data de Geração		Data de Geração do Arquivo	144	151	8		Num		G016

18.0		Hora de Geração	Hora de Geração do Arquivo	152	157	6		Num		G017
19.0		Seqüência (NSA)	Número Seqüencial do Arquivo	158	164	7		Num		G018
20.0		Layout do Arquivo	Nº da Versão do Layout do Arquivo	165	167	3		Num	"020"	G019
21.0		Densidade	Densidade de Gravação do Arquivo	168	172	5		Num		G020
22.0	Reservado Banco		Para Uso Reservado do Banco	173	191	19		Alta		G021
23.0	Reservado Empresa		Para Uso Reservado da Empresa	192	211	20		Alta		G022
24.0	CNAB		Uso Exclusivo NEXXERA	212	240	29		Alta		G004

Controle - Banco origem ou destino do arquivo.

Empresa - Empresa que firmou o convênio de prestação de serviços com o Banco.

Conta Corrente (Empresa) - Número da conta do corrente do convênio firmado entre Banco e Empresa para a prestação de um tipo de serviço. Quando o arquivo contiver mais que um tipo de serviço diferente, os dados da conta corrente a serem colocados aqui devem ser acordados entre o Banco e a Empresa.

Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoPagamentos_0011: Composição do header do arquivo
Palavras-chave:	Nenhum

Caso de Teste pag-70: CTPE Composição do lote	
Autor:	Cris

PRÉ-CONDIÇÃO::

Um lote de Pagamento típico é composto de um registro header de lote, um ou mais registros detalhe, e um registro trailer de lote. Os registros header (1) e trailer (5) de lote e os de detalhe (3) são compostos de campos fixos, comuns a todos os tipos de Pagamentos e campos específicos, padrões para cada um dos tipos de Pagamento.

Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoPagamentos_0005: Composição do lote
Palavras-chave:	Nenhum

Caso de Teste pag-71: CTPE Composição do registro de detalhe	
Autor:	Cris
PRÉ-CONDIÇÃO:: Um registro de detalhe é composto de um ou mais segmentos, dependendo do tipo de Pagamento. Existem vários tipos de segmentos diferentes e cada um deles pode ser utilizado em um ou mais lotes Pagamentos, tanto nos fluxos de **Remessa** (Cliente enviando informações para o Banco) como nos fluxos de **Retorno** (Banco enviando informações para o Cliente), conforme discriminados a seguir: com exceção de pagamento de títulos.	
Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoPagamentos_0006: Composição do registro de detalhe
Palavras-chave:	Nenhum

Caso de Teste pag-72: CTPE Composição do tipo de servico	
Autor:	Cris
PRÉ-CONDIÇÃO: Número seqüencial para identificar univocamente um lote de serviço. Criado e controlado pelo responsável pela geração magnética dos dados contidos no arquivo. Preencher com '0001' para o primeiro lote do arquivo. Para os demais: número do lote anterior acrescido de 1. O número não poderá ser repetido dentro do arquivo. Se registro for Header do Arquivo preencher com '0000' Se registro for Trailer do Arquivo preencher com '9999'	
Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoTpServico_0014: Composição do tipo de serviço
Palavras-chave:	Nenhum
Caso de Teste pag-73: CTPE Composição do trailer de lote	
Autor:	cris
PRÉ-CONDIÇÃO::	

Campo		Posição		De	Até	Nº Dig	Nº Dec	Form	Default	Desc
01.5	Controle	Banco	Código do Banco na Compensação	1	3	3		Num		G001
02.5		Lote	Lote de Serviço	4	7	4		Num		G002
03.5		Registro	Tipo de Registro	8	8	1		Num	'5'	G003
04.5	Nexxera	Uso Exclusivo Nexxera		9	17	9		Alfa	Brancos	G004
05.5	Totais	Qtde de Registros	Quantidade de Registros do Lote	18	23	6		Num		G057
06.5		Valor	Somatório dos Valores	24	41	16	2	Num		L001
07.5		Qtde de Moeda	Somatório de Quantidade de Moedas	42	59	13	5	Num		G058
08.5	Número Aviso Débito	Número Aviso Débito		60	65	6		Num		G066
09.5	Nexxera	Uso Exclusivo Nexxera		66	230	165		Alfa	Brancos	G004
10.5	Ocorrências	Códigos das Ocorrências para Retorno		231	240	10		Alfa		G059

Controle - Banco origem ou destino do arquivo (Banco Pagador).

Empresa - Cliente (Pagador) que firmou o convênio de prestação de serviços com o banco.

Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoPagamentos_0012: Composição do trailer de lote
Palavras-chave:	Nenhum

Caso de Teste pag-74: CTPE Composição do Trailer do arquivo	
Autor:	cris
PRÉ-CONDIÇÃO::	

Campo				Posição De	Posição Até	Nº Dig	Nº Dec	Form	Default	Desc
01.9	Controle	Banco	Código do Banco na Compensação	1	3	3		Num		G001
02.9		Lote	Lote de Serviço	4	7	4		Num	"9999"	G002
03.9		Registro	Tipo de Registro	8	8	1		Num	"9"	G003
04.9	Nexxera		Uso Exclusivo Nexxera	9	17	9		Alfa	Brancos	G004
05.9	Totais	Qtde de Lotes	Quantidade de Lotes do Arquivo	18	23	6		Num		G049
06.9		Qtde de Registros	Quantidade de Registros do Arquivo	24	29	6		Num		G056
07.9		Qtde de Contas Concil.	Quantidade de Contas para Concil. (Lotes)	30	35	6		Num		G037
08.9	Nexxera		Uso Exclusivo Nexxera	36	240	205		Alfa	Brancos	G004

Controle - Banco origem ou destino do arquivo.

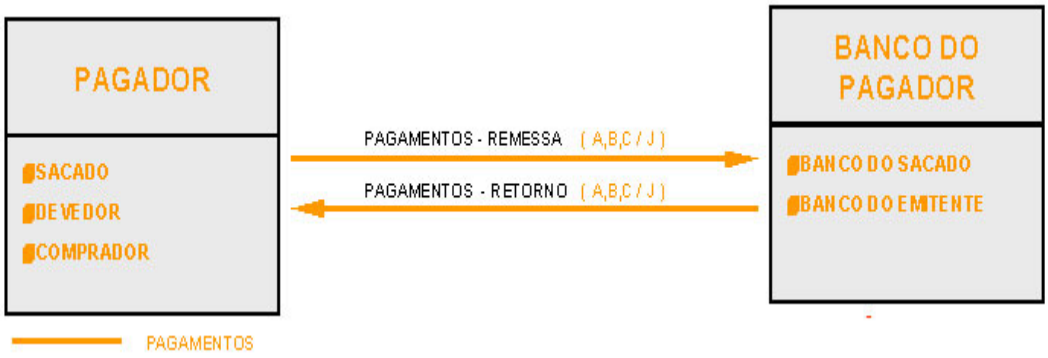
Totais - Totais de controle para checagem do arquivo.

Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoPagamentos_0007: Composição do Trailer do arquivo
Palavras-chave:	Nenhum

Caso de Teste pag-75: CTPE Entidades envolvidas	
Autor:	cris

PRÉ-CONDIÇÃO::

O fluxo de informações abaixo mostra uma visão geral dos tipos de Pagamentos disponíveis e das entidades participantes em cada um deles.



Entidades Participantes

Entidade	Descrição
Pagador	**Pessoa Física ou Jurídica que irá efetuar o pagamento de um compromisso financeiro.**
Banco do Pagador	**Banco detentor da conta corrente do Pagador, a qual será debitada para efetivação de um compromisso financeiro**
Beneficiário	**Pessoa Física ou Jurídica que irá receber os créditos de um compromisso financeiro.**
Banco do Beneficiário	**Banco detentor da conta corrente do Beneficiário, a qual será creditada na liquidação de um compromisso financeiro**

Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoPagamentos_0008: Entidades envolvidas
Palavras-chave:	Nenhum

Caso de Teste pag-76: CTPE Arquivo autorizado de inclusão	
Autor:	cris
PRÉ-CONDIÇÃO:: Fluxo "normal" de pagamentos 1- Cliente envia pagamentos a ser autorizado para Aplicação do pagamento;	

2- Cliente visualiza os pagamentos a ser autorizado pelo sistema do pagamento eletrônico;

3- Autorizador(es) autoriza os pagamentos que deseja pagar ;

4- Assim que o pagamento recebe a quantidade de assinaturas necessárias o mesmos ficam aptos para a geração;

5- O gerador monta um arquivo de pagamentos que será enviado ao banco (arquivo gerado por convênio/pagador conforme modalidade, tipo de serviço e tipo de instrução);

Os pagamentos são gerados levando em consideração:

5.1: Lista de bancos parametrizados;

5.2: Lista de convênios pagadores com pagamentos autorizados;

5.3: Lista de lotes dos convênios com pagamentos autorizados;

5.3.1 Os lotes encontram-se agrupados por modalidade e tipo de serviço;

5.3.1 Modalidade: identifica a operação que está contida no lote (ex: tributo, crédito em conta-corrente, liquidação de títulos do próprio banco);

5.3.2 Tipo de serviço: identifica o processo contido no arquivo/lote (ex: pagamento de fornecedor, pagamento salários, ted);

6- O banco recebe o arquivo, analisa os pagamentos retorna os pagamentos conforme suas regras manifestadas por ocorrências associadas ao documento;

7- Arquivo de retorno é importado e ocorrências associadas ao pagamento interpretado e disponibilizado para o cliente via web;

O produto Pagamentos tem por objetivo fornecer, aos Clientes (Pagadores) dos Bancos, os meios para racionalizar o processo de Contas a Pagar.

Este processo envolve pagamentos de compromissos que podem ser efetuados através de crédito em conta, cheque administrativo, DOC, TED, ordem de pagamento (OP), pagamento com autenticação ou títulos em cobrança.

O Pagador agenda junto ao Banco Pagador, os Pagamentos a serem efetuados pelo Banco. Caso seja agendado um pagamento bloqueado é necessário enviar uma informação para liberar a execução do pagamento posteriormente e, nos casos em contrário, se foi agendado um pagamento liberado é possível fazer o bloqueio do mesmo. Também é possível o Pagador efetuar alterações em alguns dados dos pagamentos, antes que o mesmo seja efetuado.

O Banco Pagador, na data prevista, efetua o débito na conta corrente do Pagador e executa a instrução para crédito do pagamento ao Favorecido. Este crédito poderá ser efetuado nos seguintes modos:

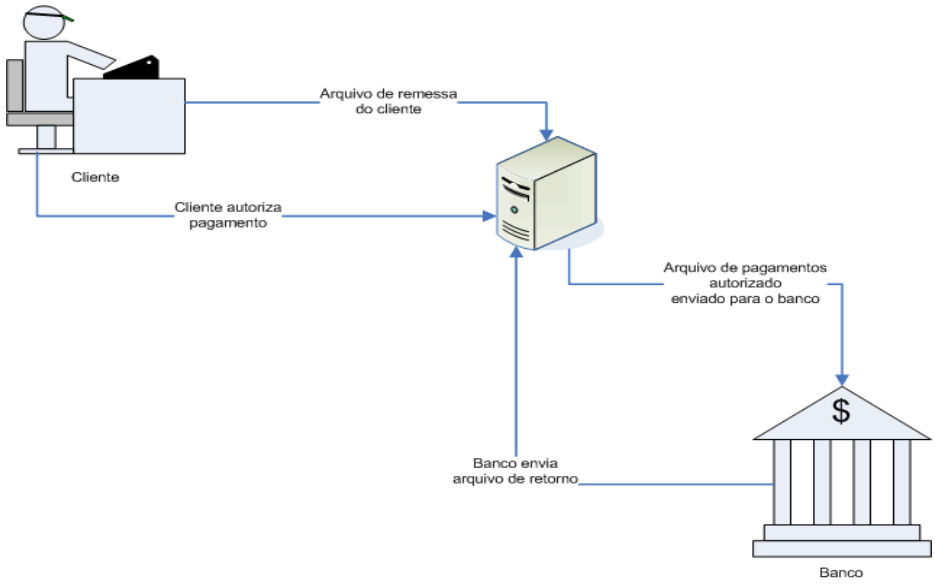
- Diretamente ao Favorecido

Através de cheque administrativo ou ordem de pagamento (OP).

- Ao Banco do Favorecido

Através de crédito em conta, quando o Banco do Pagador é o mesmo Banco do Favorecido, ou através de DOC, TED e títulos em cobrança, via compensação.

Diagrama



Último Status:	Com Falha
Baseline	Gerador 7.12 usando lib 11.23
Tester	cris

Bugs reportados

[Novo problema] 0000006 - Sistema gerou um pagamento de autorização para um pagamento de exclusão no banco.

Requisitos	RE_CT_PE_GeracaoPagamentos_0003: Pagamentos autorizados
Palavras-chave:	Nenhum

Caso de Teste pag-77: CTPE Header do arquivo	
Autor:	cris
PRÉ-CONDIÇÃO:: Se campo **ds_cnab_h_3** do primeiro lote a ser gerado for diferente de nulo então ds_cnab_h_3 do header do arquivo = ds_cnab_h_3; Se campo estiver nulo, 1- verificar o banco do pagador, Se o banco do pagador for Caixa Econômica e possuir aplicativo ds_cnab_3 do header do arquivo = aplicativo associado ao banco Caixa econômica senão ds_cnab_3 do header do arquivo = ""; fim se	
Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoPagamentos_0002: Header do arquivo
Palavras-chave:	Nenhum

Caso de Teste pag-78: CTPE Tamanho do registro e alinhamento dos campos	
Autor:	cris
PRÉ-CONDIÇÃO: **Tamanho do Registro** O Tamanho do Registro é de **240** bytes. **Alinhamento de Campos** - Campos Numéricos (Num) = Sempre à direita e preenchidos com zeros à esquerda. - Campos Alfanuméricos (Alfa) = Sempre à esquerda e preenchidos com brancos à direita.	

Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoPagamentos_0010: Tamanho do registro e alinhamento dos campos
Palavras-chave:	Nenhum

Caso de Teste pag-79: CTPE Tipo de registro	
Autor:	cris
PRÉ-CONDIÇÃO:: Código adotado pela FEBRABAN para identificar o tipo de registro. Domínio: 0 = Header de Arquivo 1 = Header de Lote 2 = Registros Iniciais do Lote 3 = Detalhe 4 = Registros Finais do Lote 5 = Trailer de Lote 9 = Trailer de Arquivo	
Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoTpRegistro_0013: Tipo de registro
Palavras-chave:	Nenhum

Caso de Teste pag-80: CTPE Segmentos	
Autor:	cris
PRÉ-CONDIÇÃO::	

O tipo de serviço/produto indica quais segmentos são obrigatórios e quais são opcionais.

Lote	Serviço/Produto	Segmentos	
		Remessa	Retorno
Pagamento através de Crédito em Conta Corrente, Cheque, OP, DOC ou Pagamento com Autenticação	Pagamentos	A (Obrigatório) B (Opcional) C (Opcional)	A (Obrigatório) B (Opcional) C (Opcional)
Pagamento de Títulos de Cobrança	Pagamentos	J (Obrigatório) K (Opcional)	J (Obrigatório) K (Opcional)
Pagamento de Tributos	Pagamento de Contas e Tributos com Código de Barras	O (Obrigatório) W* (Opcional) Z (Opcional) *obrigatório para o pagamento de FGTS, convênios 0181 e 0182	O (Obrigatório) W* (Opcional) Z (Opcional) *obrigatório para o pagamento de FGTS, convênios 0181 e 0182
	Pagamento de Tributos sem Código de Barras	N (Obrigatório) B (Opcional) W(Opcional) Z (Opcional)	N (Obrigatório) B (Opcional) W(Opcional) Z (Opcional)
Consulta de Tributos a Pagar. A utilização desse serviço deverá ser previamente acordada com o banco.			N (Obrigatório)

Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoPagamentos_0009: Segmentos
Palavras-chave:	Nenhum

Caso de Teste pag-83: CTPE Conciliação automática com ocorrências de aceite

Autor:	cris

PRÉ-CONDIÇÃO::

1 - VER CTPE Arquivo automático.

2 - Identificação de pagamentos de retorno de devolve:

As ocorrências de devolve são enviadas pelo banco, caracterizando um arquivo de retorno.

Ocorrência	LPNP	Arquivo
02 - Crédito ou débito cancelado pelo pagador/credor	02	Aceite
BD - Inclusão efetuada com sucesso	BD	Aceite
BE - Alteração efetuada com sucesso	BE	Aceite
BF - Exclusão efetuada com sucesso	BF	Aceite

3 - Pré-condição

Pagador/convênio existente no sistema, pertencer a um grupo de modelo de processamento = Geração automática;

- Pagamentos de retorno com status atual de devolve.

Ver:

- CTPE Conciliação automática com ocorrências de devolvido;
- CTPE Conciliação automática com ocorrências de rejeição;
- CTPE Conciliação automática com ocorrências de liquidação;
- CTPE Arquivo de duplicados

PASSOS::

1- Importação de pagamentos de remessa com modalidade de geração automática;

2- Gerar pagamentos automáticos dos pagamentos importados para banco;

3- Importação de pagamentos de retorno com modalidade de geração automática, conciliada com a remessa importada. Estes pagamentos de retorno devem possuir ocorrências de retorno de aceite (posição 231 a 232 do layout padrão Nexxera);

4- Verificar status dos pagamentos.

RESULTADOS ESPERADOS:

1- Importação de pagamentos de remessa com modalidade de geração automático com sucesso.

2- Gerar pagamentos automático dos pagamentos importados para banco com sucesso.

3- Importação de pagamentos de retorno com modalidade de geração automática, conciliada com a remessa importada com sucesso.

3- Status dos pagamentos com ocorrência de aceite = 31 (Aceito)

Último Status:	**Passou**
Baseline	Gerador 7.12 usando lib 11.23
Tester	cris
Requisitos	RE_CT_PE_GeracaoAutomatica_0015: Pagamentos automáticos
Palavras-chave:	Automático

Caso de Teste pag-84: CTPE Conciliação automática com ocorrencias de devolvido

Autor:	cris

PRÉ-CONDIÇÃO::

1 - VER CTPE Arquivo automático.

2 - Identificação de pagamentos de retorno de devolve:

As ocorrências de devolve são enviadas pelo banco, caracterizando um arquivo de retorno.

Ocorrência	LPNP	Arquivo
01 - Insuficiência de Fundos – Débito Não Efetuado	01	Devolvido

3 - Condição necessária para construção do teste

- Pagador/convênio existente no sistema, pertencer a um grupo de modelo de processamento = Geração automática;.
- Pagamentos de retorno com status atual de devolve.

VER:

- CTPE Conciliação automática com ocorrências de devolvido;
- CTPE Conciliação automática com ocorrências de rejeição;

- CTPE Conciliação automática com ocorrências de liquidação;
- CTPE Conciliação automática com ocorrências de aceite;
- CTPE Arquivo de duplicados

PASSOS::

1- Importação de pagamentos de remessa com modalidade de geração automático

2- Gerar pagamentos automático dos pagamentos importados para banco

3- Importação de pagamentos de retorno com modalidade de geração automática, conciliada com a remessa importada. Estes pagamentos de retorno devem possuir ocorrências de retorno 01 (posição 231 a 232 do layout padrão Nexxera)

4- Verificar status dos pagamentos

RESULTADOS ESPERADOS:

1- Importação de pagamentos de remessa com modalidade de geração automático com sucesso.

2- Gerar pagamentos automático dos pagamentos importados para banco com sucesso.

3- Importação de pagamentos de retorno com modalidade de geração automática, conciliada com a remessa importada com sucesso.

4- Status dos pagamentos com ocorrência 01 (Devolvido) = 30 (Devolvido)

Último Status:	**Passou**
Baseline	Gerador 7.12 usando lib 11.23
Tester	cris
Requisitos	RE_CT_PE_GeracaoAutomatica_0015: Pagamentos automáticos
Palavras-chave:	Nenhum

Caso de Teste pag-87: CTPE Conciliação automática com ocorrências de rejeitado

Autor:	cris

PRÉ-CONDIÇÃO::

1 - VER CTPE Arquivo automático.

2 - Identificação de pagamentos de retorno de devolve:

Códigos de ocorrência para retorno enviadas pelo banco, caracterizando um arquivo de

retorno.

Ocorrência	LPNP	Arquivo
AA - Controle Inválido	AA	Rejeitado
AB - Tipo de Operação Inválido	AB	Rejeitado
AC - Tipo de Serviço Inválido	AC	Rejeitado
AD - Forma de Lançamento Inválida	AD	Rejeitado
AE - Tipo/Número de Inscrição Inválido	AE	Rejeitado
AF - Código de Convênio Inválido	AF	Rejeitado
AG - Agência/Conta Corrente/DV Inválido	AG	Rejeitado
AH - Número Seqüencial do Registro no Lote Inválido	AH	Rejeitado
AI - Código de Segmento de Detalhe Inválido	AI	Rejeitado
AJ - Tipo de Movimento Inválido	AJ	Rejeitado
AK - Código da Câmara de Compensação do Banco do Favorecido / Depositário Inválido	AK	Rejeitado
AL - Código do Banco do Favorecido ou Depositário Inválido	AL	Rejeitado
AM - Agência Mantenedora da Conta Corrente do Favorecido Inválida	AM	Rejeitado
AN - Conta Corrente/DV do Favorecido Inválido	AN	Rejeitado
AO - Nome do Favorecido não Informado	AO	Rejeitado
AP - Data Lançamento Inválida/Vencimento Inválido	AP	Rejeitado
AQ - Tipo/Quantidade da Moeda Inválido	AQ	Rejeitado
AR - Valor do Lançamento Inválido	AR	Rejeitado
AS - Aviso ao favorecido - Identificação Inválida		Rejeitado

3 - Condição necessária para construção do teste

- Pagador/convênio existente no sistema, pertencer a um grupo de modelo de

processamento = Geração automática;.

- Pagamentos de retorno com status atual de rejeitado.

VER:

- CTPE Conciliação automática com ocorrências de devolvido;
- CTPE Conciliação automática com ocorrências de liquidação;
- CTPE Conciliação automática com ocorrências de aceite;
- CTPE Arquivo de duplicados

PASSOS::

1- Importação de pagamentos de remessa com modalidade de geração automático

2- Gerar pagamentos automático dos pagamentos importados para banco

3- Importação de pagamentos de retorno com modalidade de geração automática, conciliada com a remessa importada. Estes pagamentos de retorno devem possuir ocorrências de retorno de rejeitados (posição 231 a 232 do layout padrão Nexxera)

4- Verificar status dos pagamentos

RESULTADOS ESPERADOS::

1- Importação de pagamentos de remessa com modalidade de geração automático com sucesso.

2- Gerar pagamentos automático dos pagamentos importados para banco com sucesso.

3- Importação de pagamentos de retorno com modalidade de geração automática, conciliada com a remessa importada com sucesso.

4- Status dos pagamentos com ocorrência de rejeitado = 32 (Rejeitado)

Último Status:	**Passou**
Baseline	Gerador 7.12 usando lib 11.23
Tester	cris
Requisitos	RE_CT_PE_GeracaoAutomatica_0015: Pagamentos automáticos
Palavras-chave:	Automático

Caso de Teste pag-88: CTPE Arquivo automático de duplicados	
Autor:	cris

PRÉ-CONDIÇÃO::

O objetivo geral deste requisito, é registrar a correta configuração no ambiente skyline para troca de arquivos de folha de pagamento entre a empresa Nestlé e o banco Santander. Esta configuração envolve a elaboração de Scripts Shell em conjunto com chamadas para bcs de tradução e Importação no pagamento eletrônico.
Para compreensão deste, é necessário conhecimento na infra-estrutura do skyline, LPN, utilização do tradutor Nexxera.

Contextualização A nestlé necessita aplicar regras e tratamentos específicos para agendar registros de folha de pagamento no banco Santander. O pagamento eletrônico deve centralizar estas regras, interceptando os arquivos de remessa antes de serem gerados para o banco, e conciliando as informações de retorno antes de reenviar para a empresa cliente.

Processos de Remessa

O processo de remessa recebe arquivos no layout Santander 240, enviados pela empresa nestlé.

Importação de Remessa

Os arquivos de remessa recebidos da empresa devem ser importados no pagamento eletrônico (não sendo mais enviados diretamente ao banco), após as customizações de tradução conforme os passos a seguir:

Passo1: O arquivo de remessa original enviado pela empresa deve passar pelo processo de BC1 – Ajuste de data de crédito para customização das datas de crédito de pagamento. Neste processo, o novo arquivo gerado deve manter o nome original enviado pela empresa.

Passo2: O arquivo gerado no passo1 deve passar pelo processo de BC2 – Genérico: Pagamento LPN – Santantder240 para traduzir o layout Sandander240 para LPNP. Neste processo, assim como no passo1, o arquivo gerado deve manter o nome original enviado pela empresa.

Passo3: O arquivo gerado no passo2 deve ser importado no pagamento eletrônico.

Geração de Remessa

O registro de folha de pagamentos no banco será iniciado através do gerador do pagamento eletrônico, conforme os registros importados na remessa. Não haverá um processo de assinatura na web, os arquivos importados no processo Importação de Remessa passarão por um conjunto de regras e caso estejam de acordo com as condições previstas, serão gerados para agendamento junto ao banco. Os passos a seguir descrevem este processo:

Passo1: O gerador de pagamentos é acionado em CRON e cria o arquivo de remessa a ser enviado ao banco, com os registros de folha de pagamento no LPNP.

Passo2: O arquivo gerado no passo1 deve passar pelo processo BC2 – Genérico:

Pagamento LPN – Santantder240 para ser traduzido de LPNP para o layout Santander.

Passo3: O arquivo gerado no passo2 deve ser divido em vários arquivos, conforme o número de lotes presente no arquivo. Este processo já existe em ambiente de produção, sendo executado pelo processo de BC3 – Particionamento de convênios por lote. A regra de nomenclatura deve ser mantida, pois é necessária para o processo de conciliação no retorno do banco.

Passo4. O arquivo gerado no passo3 deve ser enviado ao banco.

Processo de Retorno

- O processo de retorno recebe arquivos no layout Santander 240, enviados pelo banco.

Importação de Retorno

Os arquivos de retorno recebidos do banco, devem ser importados no pagamento eletrônico (não sendo mais enviados diretamente para a empresa), após as customizações de tradução conforme os passos aseguir:

Passo1: O(s) arquivo(s) de retorno enviados pelo banco (podem ser mais de um, conforme o número de lotes: Ver passo3 do Geração de Remessa) devem passar pelo processo de BC4 – Junção de arquivos de retorno em convênio único. Importante ressaltar que caso exista mais de um arquivo (múltiplos lotes na remessa) os arquivos são represados através de scriptShell até que todos tenham chegado. Processo já existente em ambiente de produção e deve ser mantido.

Passo2: O arquivo gerado no passo1 deve passar pelo processo BC2 – Genérico: Pagamento LPN – Santantder240 para traduzir o arquivo do layout Santander para o LPNP.

Geração de Retorno

O registro de folha de pagamentos será iniciado através do gerador do pagamento eletrônico. Não haverá um processo de assinatura na web, os arquivos importados no processo 3.1 passarão por um conjunto de regras e caso estejam de acordo com as condições previstas, serão gerados para agendamento junto ao banco. Os passos a seguir descrevem este processo:

Passo1: O gerador de pagamentos é acionado em CRON e cria o arquivo de retorno a ser enviado à empresa, com os registros de folha de pagamento no LPNP.

Passo2: O arquivo gerado no passo1 deve passar pelo processo BC2 – Genérico: Pagamento LPN – Santantder240 para ser traduzido de LPNP para o layout Santander (Utilizado pela empresa). Neste processo, o novo arquivo gerado pelo tradutor deve manter o mesmo nome do arquivo gerado pelo Pagamento eletrônico.

Passo3: O arquivo gerado no passo2 deve ser enviado para a empresa.

PASSOS::

1- Importação de pagamentos de remessa com modalidade de geração automático

2- Importação de pagamentos de remessa com:

data de vencimento = ao pagamento importado anteriormente

seu número != ao pagamento importado anteriormente

convênio = ao pagamento importado anteriormente

valor de pagamento = ao pagamento importado anteriormente

3- Gerar pagamentos automáticos

4- Verificar status dos pagamentos

5- Verificar arquivo gerado

RESULTADOS ESPERADOS::

1- Importação de pagamentos de remessa com modalidade de geração automático com sucesso

2- Importação de pagamentos de remessa com sucesso

3- Gerar pagamentos automáticos com sucesso

4- Status do pagamento do segundo arquivo com status de duplicado

4- Gerado arquivo duplicado para o cliente

Último Status:	Passou
Baseline	Gerador 7.12 usando lib 11.23
Tester	cris
Requisitos	RE_CT_PE_GeracaoAutomatica_0015: Pagamentos automáticos
Palavras-chave:	Nenhum

Caso de Teste pag-89: CTPE Arquivo autorizado de alteracao	
Autor:	cris
PRÉ-CONDIÇÃO::	

Fluxo "normal" de pagamentos

1- Cliente envia pagamentos a ser autorizado para Aplicação do pagamento

2- Cliente visualiza os pagamentos a ser autorizado pelo sistema do pagamento eletrônico

3- Autorizador(es) autoriza os pagamentos que deseja pagar

4- Assim que o pagamento recebe a quantidade de assinaturas necessárias o mesmos ficam aptos para a geração

5- O gerador monta um arquivo de pagamentos que será enviado ao banco (arquivo gerado por convênio/pagador conforme modalidade, tipo de serviço e tipo de instrução).

Os pagamentos são gerados levando em consideração:

5.1: Lista de bancos parametrizados

5.2: Lista de convênios pagadores com pagamentos autorizados

5.3: Lista de lotes dos convênios com pagamentos autorizados

5.3.1 Os lotes encontram-se agrupados por modalidade e tipo de serviço

5.3.1 Modalidade: identifica a operação que está contida no lote (ex: tributo, crédito em conta-corrente, liquidação de títulos do próprio banco)

5.3.2 Tipo de serviço: identifica o processo contido no arquivo/lote (ex: pagamento de fornecedor, pagamento salários, ted)

6- O banco recebe o arquivo, analisa os pagamentos retorna os pagamentos conforme suas regras manifestadas por ocorrências associadas ao documento.

7- Arquivo de retorno é importado e ocorrências associadas ao pagamento interpretado e disponibilizado para o cliente via web.

O produto Pagamentos tem por objetivo fornecer, aos Clientes (Pagadores) dos Bancos, os meios para racionalizar o processo de Contas a Pagar.

Este processo envolve pagamentos de compromissos que podem ser efetuados através de crédito em conta, cheque administrativo, DOC, TED, ordem de pagamento (OP), pagamento com autenticação ou títulos em cobrança.

O Pagador agenda, junto ao Banco Pagador, os Pagamentos a serem efetuados pelo Banco. Caso seja agendado um pagamento bloqueado é necessário enviar uma informação para liberar a execução do pagamento posteriormente e, nos casos em contrário, se foi agendado um pagamento liberado é possível fazer o bloqueio do mesmo. Também é possível o Pagador efetuar alterações em alguns dados dos pagamentos, antes que o mesmo seja

efetuado.

O Banco Pagador, na data prevista, efetua o débito na conta corrente do Pagador e executa a instrução para crédito do pagamento ao Favorecido. Este crédito poderá ser efetuado nos seguintes modos:

Diretamente ao Favorecido

Através de cheque administrativo ou ordem de pagamento (OP).

Ao Banco do Favorecido

Através de crédito em conta, quando o Banco do Pagador é o mesmo Banco do Favorecido, ou através de DOC, TED e títulos em cobrança, via compensação.

Diagrama

Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoPagamentos_0003: Pagamentos autorizados
Palavras-chave:	Nenhum

Caso de Teste pag-90: CTPE Arquivo automático de inclusão	
Autor:	cris

PRÉ-CONDIÇÃO::

1 - VER CTPE Arquivo automático.

2 - Condição necessária para construção do teste

- Pagador/convênio existente no sistema, pertencer a um grupo de modelo de processamento = Geração automática;.
- Pagamentos de remessa com status de Agendamento automático'.

VER:

- CTPE Conciliação automática com ocorrências de devolvido;
- CTPE Conciliação automática com ocorrências de rejeição;
- CTPE Conciliação automática com ocorrências de liquidação;
- CTPE Conciliação automática com ocorrências de aceite;
- CTPE Arquivo de duplicados

PASSOS::

1- Cadastro de dados iniciais (configuração no arquivo dadosIniciaisModeloProcessamentoAutomatico.xml);

2- Executar o Importar arquivo de remessa para pagador XY (Arquivo de remessa =arqRemessaModeloProcessamentoAutomaticoXY.txt)

3 - Verificar o status dos pagamentos importados

4 - Executar gerador de pagamentos

RESULTADOS ESPERADOS::

1 - Cadastro de dados iniciais efetuado com sucesso;

2 - Importação do arquivo de remessa para pagador XY efetuado com sucesso

3 - Status dos pagamentos importados = Geração automática

4-Gerar 2 arquivos (dados consistentes conforme arquivo: arqRemessaModeloProcessamentoAutomáticoXYGerado.txt)

5- Status dos pagamentos importados = Aguardando Retorno

Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoAutInclusao_0016: Pagamentos automáticos de inclusão
Palavras-chave:	Nenhum

Caso de Teste pag-91: CTPE Pagamento chave duplicada	
Autor:	cris

PRÉ-CONDIÇÃO::

Caso o pagamento seja considerado como duplicado por conciliação, sistema deve ignorar a importação do registro e registrar um log que o pagamento foi ignorado.

Os registros de pagamento devem ser classificados como duplicados por conciliação caso já exista um registro de pagamento na base de dados sobre as seguintes condições:

1. Pertencer a um mesmo convênio de pagador
2. Possuir as mesmas informações para os atributos: Seu número, Data de pagamento, Valor do pagamento.

PASSOS::

1-Cadastro de pagadores (configuração no arquivo dadosIniciaisPagadoresModeloProcessamento.xml);

2 - Executar o Importar arquivo de remessa para pagador 1 (Arquivo de remessa =arqRemessaPagador1ModeloGeracaoNormal.txt).

3 - Executar o Importar arquivo de remessa para pagador 1 (Arquivo de remessa =arqRemessaPagador1ModeloGeracaoNormal2.txt)

4 - Verificar se o segundo arquivo foi importado

RESULTADOS ESPERADOS::

1 - Cadastro de pagadores efetuado com sucesso;

2 - Importar arquivo de remessa para pagador 1 com sucesso;

{}

3 - Problemas ao importar arquivo de remessa para pagador 1;

4- Apenas o primeiro arquivo será importado.

Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoDuplicado_0017: Conciliação pagamento duplicado
Palavras-chave:	Automático

Caso de Teste pag-93: CTPE Conciliação automática com ocorrências de liquidação

Autor:	cris

PRÉ-CONDIÇÃO::

1 - VER CTPE Arquivo automático.

2 - Identificação de pagamentos de retorno de devolve: As ocorrências de liquidação são enviadas pelo banco, caracterizando um arquivo de retorno.

Ocorrência	LPNP	Arquivo
00 - Crédito ou Débito Efetivado	00	Liquidado
03 - Débito Autorizado pela Agência – Efetuado	03	Liquidado

3 - Condição necessária para construção do teste

- Pagador/convênio existente no sistema, pertencer a um grupo de modelo de processamento = Geração automática;.
- Pagamentos de retorno com status atual de liquidação.

VER:

- CTPE Conciliação automática com ocorrências de devolvido;
- CTPE Conciliação automática com ocorrências de rejeição;
- CTPE Conciliação automática com ocorrências de aceite;
- CTPE Arquivo de duplicados

PASSOS::

1- Importação de pagamentos de remessa com modalidade de geração automático

2- Gerar pagamentos automático dos pagamentos importados para banco

3- Importação de pagamentos de retorno com modalidade de geração automática, conciliada com a remessa importada. Estes pagamentos de retorno devem possuir ocorrências de retorno Liquidação (posição 231 a 232 do layout padrão Nexxera)

4- Verificar status dos pagamentos

RESULTADOS ESPERADOS::

1- Importação de pagamentos de remessa com modalidade de geração automático com sucesso.

2- Gerar pagamentos automático dos pagamentos importados para banco com sucesso.

3- Importação de pagamentos de retorno com modalidade de geração automática, conciliada com a remessa importada com sucesso.

4- Status dos pagamentos com ocorrência de liquidação = 7 (liquidado)

Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoConciliaAut_0019: Conciliação automática liquidado
Palavras-chave:	Automático

Caso de Teste pag-94: CTPE Banco origem Aplicativo	
Autor:	cris
PRÉ-CONDIÇÃO::	
Último Status	**Não Executado**
Requisitos	RE_CT_PE_GeracaoBcoOrAplic_0022: Banco origem Aplicativos
Palavras-chave:	Nenhum

Caso de Teste pag-95: CTPE Solicitação de exclusão via arquivo	
Autor:	cris
PRÉ-CONDIÇÃO:: A solicitação de exclusão via arquivo é enviado pelo cliente que deverá ser gerado um arquivo de solicitação de exclusão para o banco. Este pagamento será gerado sem passar pelo processo de autorização, sendo gerado na próxima chamada do gerador. O pagamento deverá ser gerado com instrução de alteração. Se a solicitação é a exclusão do pagamento no banco este já deve estar no sistema bancário. Após a geração deste pagamento o seu status deverá ser alterado para Aguardando solicitação de exclusão no banco.	
PASSOS:: 1 - Importação de remessa de pagamentos para autorização 2- Autorização dos pagamentos 3- Geração dos pagamentos para o banco 4 - Importação de remessa de pagamentos com solicitação de exclusão via arquivo 5 - Verificar status do pagamento 7 - Geração de pagamentos para o banco 8 - Verificar status do pagamento	
RESULTADOS ESPERADOS:: 1 - Importação de remessa de pagamentos para autorização com sucesso. 2- Autorização dos pagamentos com sucesso.	

3- Geração dos pagamentos para o banco com sucesso.

4 - Importação de remessa de pagamentos com solicitação de exclusão via arquivo com sucesso

5 - Status do pagamento = Solicitação de exclusão do pagamento

7 - Geração de pagamentos para o banco com sucesso com pagamentos solicitando a exclusão do pagamento.

8 - Verificar status do pagamento

Último Status:	**Passou**
Baseline	Gerador 7.12 usando lib 11.23
Tester	cris
Requisitos	RE_CT_PE_GeracaoExcluViaArq_0024: Solicitação de exclusão via arquivo
Palavras-chave:	Nenhum

Caso de Teste pag-98: CTPE Sem caixa postal do cliente	
Autor:	cris

PRÉ-CONDIÇÃO::

Os arquivos gerados que não possuem uma caixa postal configurada será enviada para Quarentena examinada posteriormente pelo suporte de 3 nível.

PASSOS::

1 - Criar pagador/convênio/configuração de assinatura com caixa postal inexistente

2 - Importar pagamentos para este pagador/convênio criado

3- Autorizar pagamentos (quantas vezes necessários para sua autorização, com usuários diferentes)

4- Gerar arquivo com pagamentos autorizados

5- Verificar se arquivo foi gerado

6- Verificar se arquivo foi enviado para quarentena

RESULTADOS ESPERADOS::

1 - Pagador/convênios/configuração de assinatura criado com sucesso 2 - Importação de pagamentos com sucesso 3- Autorização de pagamentos efetuada com sucesso. 4- Gerado arquivo autorizados com sucesso 5- Gerado 6- Arquivo enviado para quarentena

Último Status:	**Passou**
Baseline	Gerador 7.12 usando lib 11.23
Tester	cris
Requisitos	RE_CT_PE_GeracaoSemCaixaPostal: Sem caixa postal do cliente
Palavras-chave:	Quarentena

APÊNDICE B - Relatório Gerado pelo Mantis

Visualização Avançada do Caso					
Núm:	**Categoria:**	**Gravidade:**	**Freqüência:**	**Data de Envio:**	**Última Atualização:**
7	[Geração de pagamentos]	grande	às vezes	2009-10-26 15:51	2009-11-05 03:25

Relator:	cris	**Plataforma:**	Windows	
Atribuído a:	administrator	**SO:**	windows xp	
Prioridade:	normal	**Versão SO:**	5.1	
Status:	atribuído	**Versão do Produto:**		
Build do Produto:	07.12	**Resolução:**	aberto	
Projeção:	grande recriação			
Tempo Estimado:	2 a 3 dias			

Resumo:	Geração de informação em regra inesperada para o retorno
Descrição:	Cliente Hering digitou o pagamento no valor de R$ 5.000,00, arquivo remessa autorizado foi gerado corretamente e enviado ao banco Banrisul, recebemos o retorno com a ocorrência “BD” “Agendamento Confirmado”, porém ao tentar importar no PE, apresentou erro. Devido a um problema com o cliente Cutrale, todas as regras para digitação/geração de TEDs foram revistas, implementadas e atualizadas no ambiente de produção do Pagamento Eletrônico. Sendo assim, a correção do defeito mencionado nesta RTO, relacionada ao cliente Hering, certamente está contemplada nesta atualização.
Passos para Reproduzir:	1 - Gerar pagamentos do banco banrisul 2 - Verificar a câmara centralizadora gerada
Informações Adicionais:	Em análise ao processo verificamos que o cliente está enviando o campo camara de compensação nas regras do Banrisul, inclusive de forma incorreta, pois para o Banrisul uma ted deve ser gerada com camara contendo '018' ou '810' e o mesmo está gerando com '010'. Como o layout do cliente é o LPNP ele tem que gerar nestas regras. Por exemplo, se o mesmo gerasse com camara '018' o lpn converteria para o código esperado pelo banco e no retorno este código seria convertido para o esperado pelo sistema. O nosso layout possui este campo numérico, mas as condições para este banco são tratadas, portanto gerando o campo em branco(alfa), significa que a forma de lançamento não está dentro da regras esperadas em cada layout, o que acaba ocasionando a geração incorreta do campo camara de compensação e não importando na aplicação.
Descrição do	Compilação 2600.xpsp_sp3_gdr.090804-1435: service pack 3

Sistema	
Arquivos Anexados:	RTO - HERING - Pagamento Eletrônico - 20090623.doc (109 KB) 2009-10-26 15:51

Visualização Avançada do Caso

Núm:	Categoria:	Gravidade:	Freqüência:	Data de Envio:	Última Atualização:
6	[Geração de pagamentos]	grande	sempre	2009-10-26 15:42	2009-11-05 03:24

Relator:	cris	**Plataforma:**	Windows	
Atribuído a:	administrator	**SO:**	windows xp	
Prioridade:	normal	**Versão SO:**	5.1	
Status:	atribuído	**Versão do Produto:**		
Build do Produto:	07.09	**Resolução:**	aberto	
Projeção:	correção pequena			
Tempo Estimado:	2 a 3 dias			

Resumo:	Sistema gerou um pagamento de autorização para um pagamento de exclusão no banco
Descrição:	O cliente solicitou uma Exclusão de Pagamento no Banco, via Sistema Pagamento Eletrônico Nexxera e o sistema enviou uma autorização.
Passos para Reproduzir:	1 - Importar pagamento de retorno com ocorrência WS 2 - Solicitar a exclusão via web do pagamento 3 - Gerar pagamento de exclusão via web
Informações Adicionais:	Nenhuma repercursão até o momento, pois estávamos acompanhando o processo em virtude dos erros ocorridos com a mesma funcionalidade na semana passada. Desta forma, identificamos o ocorrido e enviamos manualmente um arquivo com o código 9 de exclusão. O gerador de pagamento não estava considerando o pagamento excluído via web, quando este era inserido no sistema através de ocorrencia de retorno WS. Para pagamentos incluídos pelo fluxo de arquivo de remessa, o problema não ocorria.
Descrição do Sistema	Compilação 2600.xpsp_sp3_gdr.090804-1435: service pack 3
Arquivos Anexados:	RTO - CUTRALE - PAGAMENTO ELETRONICO - 2009-05-13.doc (153 KB) 2009-10-26 15:42

Visualização Avançada do Caso

Núm:	Categoria:	Gravidade:	Freqüência:	Data de Envio:	Última Atualização:
5	[Geração de pagamentos]	grande	não se tentou	2009-10-26 15:02	2009-10-28 01:07

Relator:	cris	**Plataforma:**		
Atribuído a:		**SO:**		
Prioridade:	alta	**Versão SO:**		
Status:	atribuído	**Versão do Produto:**		
Build do Produto:	07.11	**Resolução:**	aberto	
Projeção:	nenhuma			
Tempo Estimado:	nenhum			

Resumo:	Erro ao gerar pagamentos para o banco Citibank
Descrição:	No dia 27/02 foi corrigido um problema no Gerador de Pagamentos do Pagamento Eletrônico Nexxera, onde o mesmo não estava inserindo no Segmento B posição 226-228 a informação 001, indispensável para processamento de TED´s e DOC´s no CITIBANK. Após atualização do gerador de pagamentos no dia 06/04 às 10:56 o problema voltou a ocorrer. Favor verificar com urgência, visto que já temos um montante de R$146.285.74 pagamentos rejeitados.
Passos para Reproduzir:	1 - Gerar pagamento para o banco Citibank com segmento B 2 - Verificar a posição 226-228 do arquivo (deverá possuir o valor 001)
Informações Adicionais:	Até a presente data já temos um montante de R$146.285.74 de pagamentos rejeitados, o banco Citibank nos cobra uma posição do que está ocorrendo visto que trata-se de uma correção efetuada a mais de um mês.
Arquivos Anexados:	RTO - URGENTE - TENDTUDO - PAGAMENTO ELETRONICO - 20090408-1.doc (233 KB) 2009-10-26 15:02 Não há anotações anexadas a este caso.

Visualização Avançada do Caso

Núm:	Categoria:	Gravidade:	Freqüência:	Data de Envio:	Última Atualização:
4	[Geração de pagamentos]	grande	sempre	2009-10-26 14:57	2009-11-05 03:23

Relator:	cris	**Plataforma:**	Windows	
Atribuído a:	administrator	**SO:**	windows xp	
Prioridade:	urgente	**Versão SO:**	5.1	
Status:	resolvido	**Versão do Produto:**		
Build do Produto:	07.09	**Resolução:**	corrigido	
Projeção:	correção pequena			
Tempo Estimado:	nenhum			

Resumo:	Pagamento de solicitação de exclusão via arquivo gerado pelo sistema como inclusão no banco
Descrição:	Cliente enviou remessa com código de movimento de exclusão, o pagamento foi importado no PE como “Exclusão Solicitada Via Arquivo” e o arquivo gerado pelo PE foi enviado ao banco como inclusão.
Passos para Reproduzir:	1 - Importar pagamentos de remessa para exclusão via arquivo 2 - Gerar pagamentos de exclusão via arquivo para o banco 3 - Verificar se o tipo de movimento do pagamento é de exclusão ou de Inclusão (regra correta: gerar pagamento de solicitação de exclusão via arquivo)
Informações Adicionais:	Defeito de prioridade alta. Além de não excluir o pagamento no banco, foi feita outra inclusão do mesmo compromisso. Para tentar reverter a situação, foram enviados dois arquivos com solicitação de exclusão ao banco (neste caso, o arquivo original do cliente).
Descrição do Sistema	Compilação 2600.xpsp_sp3_gdr.090804-1435: service pack 3
Arquivos Anexados:	Não há anotações anexadas a este caso.

Visualização Avançada do Caso

Núm:	Categoria:	Gravidade:	Freqüência:	Data de Envio:	Última Atualização:
3	[Geração de pagamentos]	grande	às vezes	2009-10-26 14:47	2009-10-26 14:47

Relator:	cris	**Plataforma:**	Windows	
Atribuído a:		**SO:**	windows xp	
Prioridade:	normal	**Versão SO:**	5.1	
Status:	novo	**Versão do Produto:**		
Build do Produto:	07.09	**Resolução:**	aberto	
Projeção:	nenhuma			
Tempo Estimado:	nenhum			

Resumo:	Pagamento digitado e gerado pelo sistema é rejeitado pelo banco
Descrição:	Quando é efetuada a digitação de pagamentos no sistema Pagamento Eletrônico Nexxera, onde o banco do favorecido é o mesmo banco do pagador e títulos onde o banco do favorecido é diferente do pagador o sistema gera o arquivo com apenas um header de lote, sendo assim rejeitado o pagamento para outra titularidade, pois deveriam ser gerados dois lotes.
Passos para Reproduzir:	1 - Login com usuário com perfil de digitação de pagamentos 2 - Escolher aba de digitação 3 - Digitar pagamento com modalidade Outra titularidade 3.1 - Escolher banco do favorecido diferente ao banco do pagador/convenio 4 - Digitar pagamento com modalidade mesma titularidade 3.1 - Escolher banco do favorecido igual ao banco do pagador/convenio 5 - Preencher demais campos obrigatórios 6 - Gravar dados do pagamento digitado 7 - Remeter pagamento 8 - Autorizar pagamento digitado e remetido 9 - Após autorização completa do pagamento, gerar arquivo de Remessa para os banco com o documentos digitado, remetidos e autorizados. 10- Verificar quantidade de lotes gerados no arquivo (deveria gerar 2) 10.1 - lote de pagamento de mesma titularidade 10.2 - lote de pagamento de outra titularidade
Informações Adicionais:	Cliente: Trebeschi
Descrição do Sistema	Compilação 2600.xpsp_sp3_gdr.090804-1435: service pack 3
Arquivos Anexados:	RTO - Trebeschi - Pagamento Eletrônico - 20090318.doc (100 KB) 2009-10-26 14:47

APÊNCICE C - Sites de ferramentas utilizados no processo de testes

Cobertura – Cobertura de código
http://cobertura.sourceforge.net/

DBUnit - Testes
http://dbunit.sourceforge.net/

Hudson – Integração Contínua
https://hudson.dev.java.net/

JUnit - Testes
http://www.junit.org/

Mantis – Gestão de defeitos
http://www.mantisbt.org/

TestLink – Gestão de testes
http://www.teamst.org/index.php?option=com_content&task=view&id=61&Itemid=2
http://www.zezologs.org/blog/questionamentos-sobre-o-testlink-1/

WinCVS – Controle de versão
http://www.wincvs.org/

APÊNCICE D – Fóruns, blogs e sites acessados para aprendizado

http://br.groups.yahoo.com/group/scteste/
http://br.groups.yahoo.com/group/DFTestes/
http://diariodaqualidade.blogspot.com/
http://java-source.net/open-source/testing-tools
http://sembugs.blogspot.com/
http://softwarequality.spaces.live.com/
http://testexpert.com.br/
http://www.abramti.org.br
http://www.alats.org.br/
http://www.testesdesoftware.blogspot.com/
http://www.kaner.com/